

# 스마트 컨트랙트 기반 포전거래 시스템



201824549 이 세 진  
201724435 김 재 현  
201924500 신 예 주

지도교수 김 호 원

---

# 목 차

## 1. 서론

- 1.1 과제 배경
- 1.2 과제 목표 및 내용

## 2. 이론적 배경 및 관련 연구

- 2.1 블록체인
  - 2.1.1 블록체인 개념
  - 2.2.2 블록체인 종류
- 2.2 하이퍼레저 패브릭
- 2.3 포전거래
- 2.4 GAP (농산물 인증 개념)
- 2.5 Go 언어
- 2.6 JavaScript

## 3. 스마트 컨트랙트 기반 포전거래 시스템 구현

- 3.1 스마트 컨트랙트 기반 포전거래 시스템 네트워크 설계
- 3.2 스마트 컨트랙트 기반 포전거래 시스템 DApp 설계
- 3.3 스마트 컨트랙트 기반 포전거래 시스템 구현
  - 3.3.1 개발환경
  - 3.3.2 스마트 컨트랙트 기반 포전거래 시스템 구현
  - 3.3.3 스마트 컨트랙트 기반 포전거래 시스템 DApp 구현

## 4. 결론 및 향후 연구

## 5. 구성원별 역할 및 개발 일정

- 5.1 구성원별 역할
- 5.2 개발 일정

## 참고 문헌

---

# 1. 서론

## 1.1 과제 배경

세계적인 미래학자이자 탭스콧그룹 CEO 인 돈 탭스콧은 인터넷이 지난 30~40년을 지배해온 것처럼 앞으로는 블록체인이 30년 이상 지배할 것이며, 차세대 핵심 기술로 세상을 변화시킬 것이라고 언급하였다. 블록체인은 탈중앙화 기술을 이용하여 거래의 기록 및 관리 권한을 중앙기관 없이 P2P 네트워크를 통하여 분산하여 블록으로 관리하고 기록하며 거래 당사자들끼리 직접 거래할 수 있다. 또한, 암호화 기술인 해시함수를 이용하여 높은 보안성을 제공한다.

특히, Hyperledger Fabric의 경우 채널을 도입함으로써 제 3자들 간의 거래에 기밀성을 높인다 (이동영 등, 2017). 이런 블록체인의 장점들을 활용하여 디지털 신분증, 전자 문서 관리, 부동산 거래, 온라인 투표 등 다양한 분야에 적용되고 있으며 데이터의 신뢰성이 중요한 개인 인증, 가상화폐, 유통 등의 분야를 중심으로 더욱 발전하고 있다 (국경완, 2019).

포전거래는 선도거래의 일종으로, 생산자가 수확하기 이전의 경작 상태에서 면적단위 또는 수량단위로 매매하는 것이다. 농작물은 재화의 특성 상 수요가 비탄력적이고 공급 측면에서 기상·기후·병충해 등에 영향을 받기 때문에 매년 가격 변동이 크다. 따라서 산지에서 농민과 산지유통인은 포전거래를 주요 가격변동 위험 관리 수단으로 사용한다. 2019년 기준 포전거래가 이루어지는 품목은 총 20개이며 이 품목들의 평균 포전거래율은 48.1%이다 (한국농수산물유통공사, 「2019년 유통실태 종합」). 그 중 포전거래의 비율이 가장 높은 작물은 배추, 무 등의 엽근채류이며 배추의 연중 평균 포전거래 비율은 83%, 무는 88%이다.

이러한 포전거래는 산지유통인의 오랜 경험과 정보를 바탕으로 이뤄지므로 상품 선별, 출하 시기 조정, 도매시장에서의 신용구축 측면에서 우수하다. 또한 가격 위험을 어느 정도 헤지(hedge)할 수 있다. 또한, 농가는 계약금을 통해서 출하 이전에 영농자금 확보가 가능하며 산지유통인은 수확 이전에 물량 확보가 가능하다는 장점이 있다. 하지만 포전거래는 산지유통인과 농민(농가) 간의 구두계약으로 이루어지기에 계약의 안정성이 부족하다는 점, 산지유통인이 가격 결정에 있어서 우월한 지위를 가진다는 것, 농산물 유통단계의 투명도를 떨어트리고 정보화를 방해하여 선진적 농업-유통 구조 형성을 방해한다는 단점이 있다. 실제로 생산자(농민)에 대한 설문조사에서도 포전 거래의 문제점으로 계약이 너무 늦게

---

이루어진다는 점, 산지유통인의 계약 파기가 잦다는 점, 가격정보 면에서 산지유통인에 비해 불리하다는 점, 계약금의 수준이 낮다는 점 등이 지적되었다(하용현, 2009).

## 1.2 과제 목표 및 내용

과거에도 이러한 농산물 포전거래에 대한 표준화와 보증의 필요성을 강조하는 연구들이 학계에서 있었다 (양승룡 등, 2009). 하지만 당시 실제로 도입되지 못한 것은 농민과 산지유통인이 시공간적 제약을 받지 않고 접근할 수 있는 통신매체가 부족했고, 기술적·금융적 진입장벽이 낮은 유통 플랫폼이 부재했기 때문이다. 그러나 현재 전국민의 95% 이상이 스마트폰을 보유하고 있기에 시공간적 제약 없이 접근 가능한 통신매체 환경은 이미 갖추어졌으며, 기술적·금융적 진입장벽이 낮은 유통 플랫폼만이 요구되는 상황이다. 따라서 우리는 스마트폰 어플리케이션을 이용하여 농산물 포전거래 시스템을 사용 가능한 플랫폼을 제안하는 바이다.

본 과제에서는 허가형 프라이빗 블록체인 중 하나인 Hyperledger Fabric을 통해 농민과 산지유통인 간의 신뢰를 높인 스마트 컨트랙트 기반 농산물 포전거래 플랫폼을 구현하였다. Go 언어와 JavaScript 라이브러리 중 하나인 Node.js를 사용하여 DApp을 구현하였다. 국립농산물품질관리원이 제공하는 GAP 인증정보를 데이터베이스에 저장함으로써 농산물 품질을 검증하고, 거래 기록도 블록체인으로 관리함으로써 유통의 신뢰성을 더욱 향상시켰다.

---

## 2. 이론적 배경 및 관련 연구

본 장에서는 본 과제의 이론적 배경과 관련된 연구에 대해 살펴본다. 먼저 2.1절에서는 본 과제의 주요 기술인 블록체인에 대하여 설명한다. 2.2절에서는 블록체인의 한 종류이며 본 과제에 사용한 Hyperledger Fabric에 대하여 설명하고, 2.3절에서는 국가에서 제공하는 농산물품질인증시스템인 GAP에 대해 설명한다.

### 2.1 블록체인

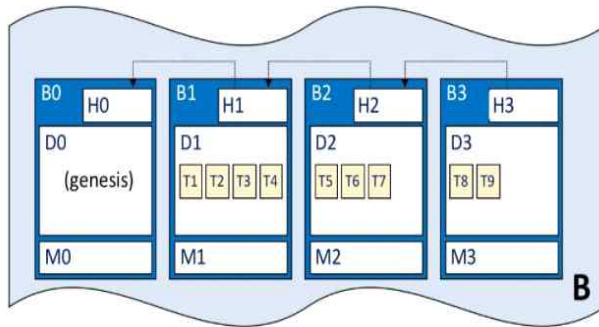
#### 2.1.1 블록체인

블록체인은 중앙 기관이 없어도 사용자 간에 신뢰할 수 있는 탈중앙화 시스템이다. 공유 원장과 P2P 네트워크를 사용하여 블록체인 사용자가 거래를 변경할 시 내역이 모든 사용자의 원장에 기록된다.

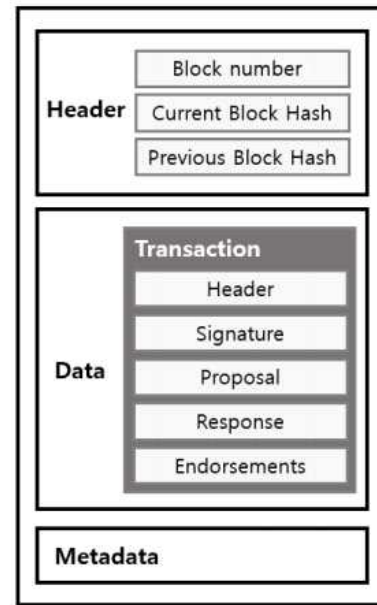
P2P 네트워크는 'Peer-to-Peer'의 약자로 기존의 Client-Server 모델과는 달리 개인과 개인이 직접적으로 통신하는 탈중앙화 네트워크이다. Client-Server 모델은 중앙의 서버가 모든 데이터를 저장하고, 대부분의 서비스를 서버에 요청하여 제공받는다 (아키하네 요시하루 등, 2017). 이는 중앙 집중형 모델이기에 서버에 장애 발생 시 모든 서비스가 이용 불가하며 해킹당한다면 모든 클라이언트들에게 잘못된 정보가 전달된다. 하지만 P2P 네트워크는 사용자가 클라이언트인 동시에 서버이기에 하나의 노드가 해킹당하더라도 다른 노드들이 정확한 정보를 가지고 있기에 해킹 방지가 가능하다.

블록체인의 또 다른 주요 기술로는 해시함수가 있다. 블록체인의 데이터들은 해시함수로 해시화 시킨 뒤 원장에 저장되기 때문에 수정이 절대 불가능하다. 해시함수는 시간을 정하여 일정 간격 별로 데이터를 블록화하여 해시화시키고, 해시화된 블록들끼리는 사슬처럼 연결되어 있다. <그림 1>처럼 블록체인은 상호연결되어있는 구조로 각 블록은 이전 블록의 해시 정보를 포함한다. 특정 블록을 해킹하려면 이후 블록들도 모두 수정해야하기에 해커로부터 데이터 보호가 가능하다.

<그림 2>를 보다시피 블록은 Header, Data, Metadata로 이루어져 있다. Header의 Previous Block Hash에는 이전 블록에 대한 해시값이 포함되어 있다. 모든 블록은 앞 블록의 해시값을 포함하는 체인 구조로 연결되어 있어 데이터 위변조가 불가하며 데이터 신뢰도가 높다.



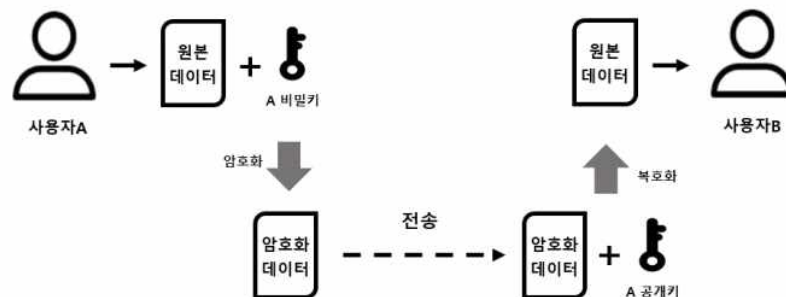
<그림 1> 블록체인 구조



<그림 2> 블록체인의 블록 구조

<그림 1>처럼 블록에 저장되는 data는 여러 개의 트랜잭션으로 구성되는데 각각의 트랜잭션은 Header, Signature, Proposal, Response, Endorsements로 이루어져 있다. Signature에는 트랜잭션 생성자의 암호화된 서명이 포함되어 있다.

블록체인의 인증은 PKI(Public Key Infra) 기반의 디지털 인증서를 이용한다. 이의 4가지 인증 핵심 요소로는 디지털 인증서, 공개키 및 비밀키, CA, CRL이 있다. 디지털 인증서는 사람의 여권처럼 세계적으로 신원 인증이 가능한 신원 인증서와 비슷하다. 공개키 및 비밀키는 공개키 암호화 방식이라고 하는데 <그림 3>처럼 사용자 A가 자신의 비밀키로 데이터를 암호화한 뒤 사용자 B에게 전송한다. 사용자 B는 A의 공개키를 이용하여 데이터 복호화를 성공시키며 인증한다.



<그림 3> 공개키 암호화 방식

CA는 인증기관이며 공개키 암호화 방식의 취약점을 해결한다. 사용자들은 CA에 자신의 공개키를 등록하고 다른 사용자의 공개키가 필요할 시 CA에 공개키를

요청한다(윤대근, 2018). CRL은 폐기된 인증서 목록으로 이를 통해 폐기된 인증서인지 확인 후 유효 여부를 파악 가능하다.

블록체인의 또 다른 핵심 기술로는 합의 알고리즘이 있다. 앞서 설명한 P2P 네트워크는 해킹 위험은 적지만 정보 지연과 정보 미도달로 인한 이중 송신에 따른 처리 중복, 잘못된 정보에 의한 오작동 등의 단점이 있다(아카하네 요시하루 등, 2017). 이를 해결하고자 블록체인 네트워크에서는 PoW, PoS, PoET, Paxos, Raft, SBFT 등의 합의 알고리즘을 사용한다.

스마트 컨트랙트는 블록체인의 핵심 구성요소이다. 이는 특정 조건이 충족되면 자동으로 계약이 실행되는 전자 계약을 의미한다(Szabo, Nick, 1997). 블록체인이 떠오르며 강한 신뢰가 필요한 스마트 컨트랙트에 블록체인 기술을 합침으로써 더 완벽한 스마트 컨트랙트가 완성되었으며, 거래 당사자의 개입 없이 자동적으로 거래 이행이 가능하여 굉장히 효율적이다.

### 2.1.2 블록체인의 종류

블록체인의 종류는 크게 비허가형 블록체인과 허가형 블록체인으로 나뉜다.

비허가형 블록체인은 퍼블릭 블록체인이라고도 불리며, 누구나 참여 가능하며 누구나 트랜잭션을 생성 가능하다. 다수가 참여하기에 비용이 많이 들고 느리지만 신뢰성이 높다. 허가형 블록체인은 프라이빗 블록체인과 컨소시엄 블록체인으로 나뉜다. 프라이빗 블록체인은 중앙기관이 관리하기 때문에 비용이 적고 빠르지만, 탈중앙화가 이루어지지 않아 신뢰성이 떨어진다. 컨소시엄 블록체인은 퍼블릭 블록체인과 프라이빗 블록체인의 중간이다. 공동 사업체로 참가하는 조직만 참여 가능한 블록체인이다 (시미즈 토모노리 등, 2019). 본 과제에서는 컨소시엄형 블록체인 중 Hyperledger fabric을 사용하였다.

	퍼블릭형	컨소시엄형
관리 주체	없음	다수의 조직, 기업
참가자	자유	허가제
합의 방식	PoW, PoS, DPoS 등	퍼블릭과 프라이빗 이용
구현 사례	비트코인, 이더리움	하이퍼레저 패브릭

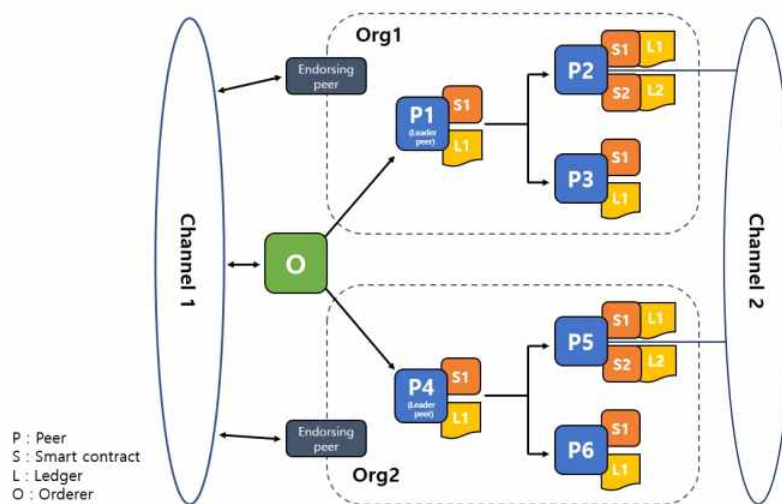
<표 1> 퍼블릭형과 컨소시엄형 블록체인

## 2.2 하이퍼레저 패브릭

리눅스 재단의 하이퍼레저 프로젝트 중 가장 활발하며 다양한 업무 시스템 구축 및 운영이 가능하다. 컨소시엄형 블록체인이며 MSP(Membership Service Provider)라는 기술로 신원 확인 및 인증이 가능하다.

특징	설명
신원 관리	MSP를 이용한 회원 자격 증명 서비스
개인정보 보호 및 비밀 유지	채널 생성을 통해 공유하고 싶은 조직끼리 정보 공유
효율적인 병렬 처리	트랜잭션 처리 과정을 3단계로 분리하여 병렬 방식을 사용하여 효율성 높아짐
체인 코드	스마트 컨트랙트의 기능인 체인 코드와 채널의 규칙을 정의하는 시스템 체인코드 사용
모듈식 디자인	블록체인 설계자가 원하는 형태로 서비스 제공이 가능하도록 모듈식 아키텍처를 사용

<표 2> 하이퍼레저 패브릭의 특징



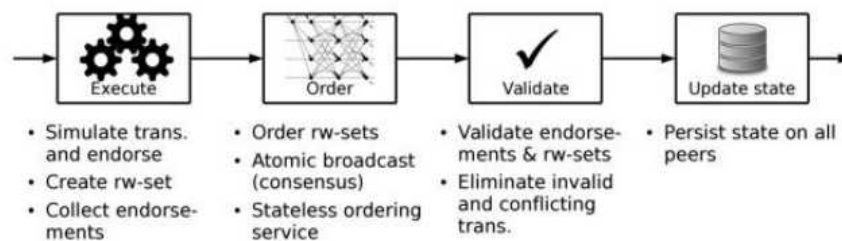
<그림 4> 하이퍼레저 패브릭의 구성

하이퍼레저 패브릭의 전체적 구성은 <그림 4>과 같다. peer는 endorsing, committing의 역할을 하는 하나의 네트워크 노드이다(P1~P6). 분산원장(Ledger)과 스마트 컨트랙트가 peer에 저장되기에 peer를 통해서만 분산원장과 체인 코드에 접근이 가능하다(윤대근, 2017). Orderer는 블록을 생성하고 peer들에게 전파한다. Organization은 블록체인에 참여하는 기관 및 조직을 나타낸다. <그림 4>에서는 P1, P2, P3가 Organization1에 속해 있으며 P4, P5, P6은



Organization2에 속해 있다. 앞서 언급한 것처럼 하이퍼레저 패브릭은 채널 개념을 도입하여 개인정보 보호 및 비밀 유지를 제공해준다. peer들은 채널을 통해 연결되는데 하나의 채널은 하나의 분산원장을 공유한다. <그림 4>에서 P1, P2, P3, P4, P5, P6은 채널 1에 참여하며 그 중 P2와 P5는 채널 2에도 참여한다. 이렇듯 peer들은 2개 이상의 채널에 참여 가능하다.

하이퍼레저 패브릭은 효율성을 위해 트랜잭션 처리 과정을 총 3단계인 Execute, Order, Validate으로 나누었다(<그림 5>). Execute 단계에서는 발생한 트랜잭션을 DApp을 통해 Endorsing peer에게 전달한다. Order 단계에서는 Orderer 노드가 자신이 전달받은 트랜잭션을 정렬하여 최신 블록을 생성한다. 마지막으로, Validation 단계에서는 Orderer가 생성한 블록들을 Organization의 Leader peer에 전달한다. Leader peer들은 자신이 속해 있는 Organization의 peer들에게 블록을 전파하고, 그를 전달받은 peer들은 올바른 블록인지 판단 후 문제가 없다면 최신 블록으로 추가하고 state를 업데이트한다.



<그림 5> Execute-order-validate architecture of Fabric

각 조직의 peer들은 orderer로부터 분산원장 업데이트가 가능한데 조직의 모든 peer가 orderer에게 계속 요청한다면 orderer 노드는 과부하가 된다(윤대근, 2017). 이러한 과부하를 해결하기 위해 하이퍼레저 패브릭은 peer들 중 하나를 뽑아서 Leader peer로 두고 orderer 노드와 연결시킨다. Leader peer는 무조건적으로 살아있어야 하며, 만약 브로드캐스트 메시지에 응답이 없다면 자동으로 새로운 Leader peer를 선출한다.

## 2.3 GAP (농산물 인증 제도)

본 과제에서는 포전 거래 시 농산물의 안전성을 검증하고자 국립농산물품질관리원에서 제공하는 GAP 인증을 활용하였다. GAP 인증은 안전하고 위생적인 농산물을 위해 생산부터 유통까지의 위해 화학적, 생물학적 등의 위해요소를 종합적으로 관리하는 제도이다. GAP 인증의 유효기간은 2년이며 인증을 위해서는 직접 농산물품질관리원장이 지정한 인증기관에 신청해야 한다. 하지만 인증 심사가

까다롭고 소요되는 비용이 많아 농가에서 속여 파는 경우가 있다. 이러한 경우를 방지하고자 국립농산물품질관리원에서는 GAP 인증을 속이는 경우 3년 이하의 징역 또는 3천만원 이하의 벌금을 부과한다. 우리 플랫폼에서는 GAP 인증을 받은 농산물만 등록할 수 있도록 함으로써 눈속임을 방지하고 우수한 품질의 농산물이 유통될 수 있도록 한다.

## 2.4 Go 언어

최근 가상화 컨테이너 Docker의 발전으로 인해 Docker 개발 언어인 Go 언어가 떠오르게 되었다. 구글이 개발한 Go 언어는 유닉스와 B언어, 유닉스의 개발자인 켄 톰슨, 린 파이크, 로버트 그리즈머를 주도로 최초 설계하였다(이재홍, 2015). 이 언어는 특징들을 <표 3>에 정리해두었다(장재휴, 2016). 본 과제에서는 블록체인의 체인 코드 구현을 위해 Go 언어를 사용하였다.

특징	설명
유연하고 간결한 문법	객체 지향을 표현, 상속이 아닌 조합의 방식으로 코드 재사용
정적 타입 언어, 동적 프로그래밍	정적 타입의 언어이지만 동적 언어의 특성도 수용, 덕 타이핑 방식으로 동작
병행 프로그래밍	통신 순차 프로세스 방식에 근간을 두고 메모리 공유가 아닌 메시지 전달 방식으로 동기화시켜 병행처리
가비지 컬렉션 제공	C++, C와 달리 가비지 컬렉션을 제공함으로써 개발의 편리성 높임
쉬운 협업	가이드 제시를 통해 개발 시 같은 작업 환경 구성, gofmt 도구를 사용해 코드 스타일의 문제 해결
빠른 컴파일 속도	실제 사용되는 부분만 컴파일하므로 속도 빠름

<표 3> Go 언어의 특징

## 2.5 JavaScript

JavaScript는 웹 위에서 동작하는 Front-End 개발 언어로써 정적인 웹 문서를 동적으로 바꾸어준다(정인용, 2018).

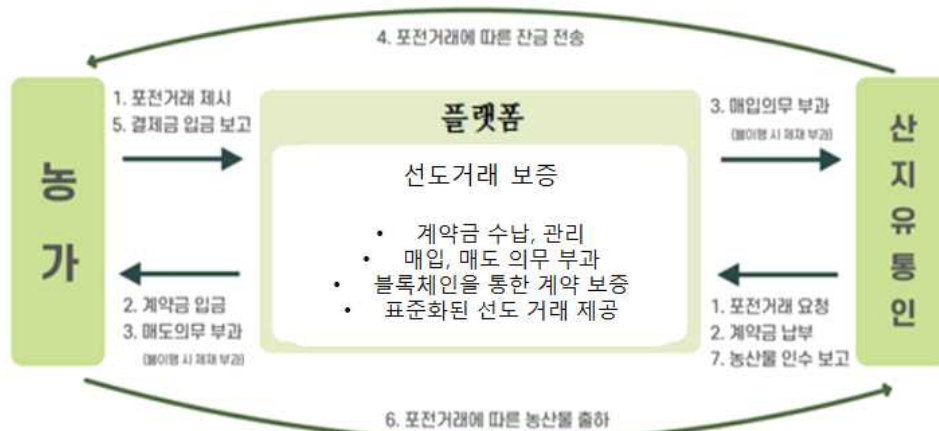
제어 쿼리, NodeJs, React 등 JavaScript 기반의 라이브러리가 사랑받으며 수많은 개발자들은 웹 구현을 위해 JavaScript 언어를 사용하였다. 본 과제에서는 여러 JavaScript 기반의 라이브러리 중 NodeJs를 사용하여 스마트 컨트랙트 기반 포전거래 시스템 DApp의 API 서버를 구축하였다.

### 3. 스마트 컨트랙트 기반 포전거래 시스템

본 장에서는 블록체인을 활용한 스마트 컨트랙트 기반 포전거래 시스템에 대하여 설명한다. 3.1절에서는 스마트 컨트랙트 기반 포전거래 시스템의 시나리오를 설명하고, 3.2절에서는 사용자가 스마트 컨트랙트 포전거래 시스템의 네트워크의 설계에 대해 설명한다. 3.3절에서는 구현한 DApp의 설계와 마지막으로 3.4절에서는 설계한 내용을 바탕으로 직접 구현한 과정과 결과에 대해 자세히 설명한다.

#### 3.1 스마트 컨트랙트 기반 포전거래 시스템 전체 시나리오

스마트 컨트랙트 기반 포전거래 플랫폼은 농민과 산지유통인 간의 공정한 포전거래를 위한 시스템이다. 스마트 컨트랙트에 기반하여 조건 만족 시 자동으로 계약이 처리가 됨으로써 구두 계약이나 불공정 거래의 단점을 방지 가능하다.



<그림 6> 플랫폼 상 거래 시나리오

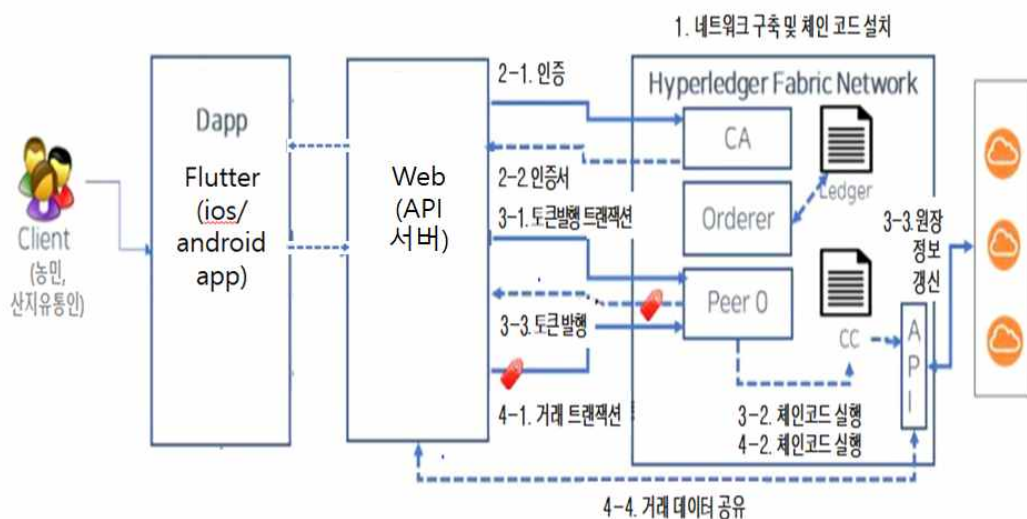
<그림 6>은 개발한 스마트 컨트랙트 기반 포전거래 시스템의 거래 시나리오이다. 거래 과정은 다음과 같다.

- ① 농민은 거래할 농산물의 종류와 가격을 시스템 상에 등록한다.
- ② 산지 유통인은 시스템에 등록되어 있는 매물 목록을 보고 원하는 조건의 거래에 대해 구매 요청을 하고, 시스템에 저장되어 있는 정보들을 이용하여 계약서가 생성되며 계약이 가능해진다.
- ③ 토큰을 이용하여 계약금을 농민에 지불하고, 체인 코드가 실행된다.

④ 거래가 성사되면 농산물 거래를 진행한다. 이는 온라인, 오프라인 상의 어떤 방법으로든 가능하다. 이 부분에 대해서는 현재 시스템 상에서는 자세히 구현하지 않았다.

⑤ 거래가 완료되면 잔금 처리를 실시하고, DApp 상에서 거래가 완료되었다는 알림을 받아볼 수 있다.

⑥ 만약 거래가 완료되지 않고 파기되었을 경우에는 포전거래 표준매매계약서의 조항에 따른다.



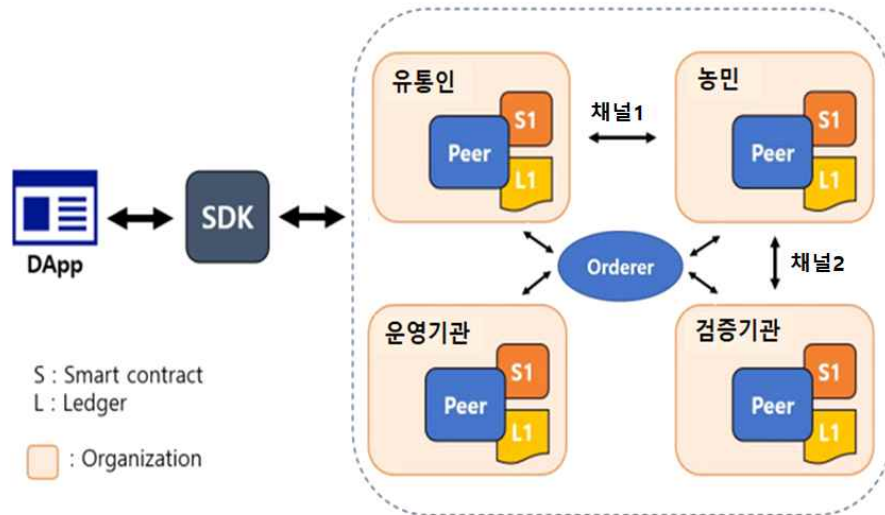
<그림 7> 시스템 시나리오

### 3.2 스마트 컨트랙트 기반 포전거래 시스템 네트워크 설계

스마트 컨트랙트 기반 농산물 포전거래 시스템은 기존의 농민, 산지유통인 간의 포전거래 시스템에 블록체인 기반의 스마트 컨트랙트를 적용한 상호 신뢰 시스템이다. 본 과제에서는 보증된 특정 조직 간의 거래 네트워크 구현을 위해 허가형 블록체인인 Hyperledger Fabric을 선택하였으며 블록체인의 참여자는 농민 (Seller), 산지유통인 (Buyer), 운영기관 (Manager), 검증기관 (Validation)이다. 우수한 품질의 농산물 유통을 위해 농산물이 국립농산물품질관리원의 GAP 인증을 받았는지 검증하는 검증기관을 블록체인의 참여자로 추가하였다. 검증기관은 GAP 인증정보를 블록체인을 통해 관리하며 농민의 농산물을 보증하고, 관리기관은 농민과 산지유통인의 신원 정보와 농산물 거래 내역 정보 등을 블록체인을 통해 저장하며 관리한다.

스마트 컨트랙트 기반 농산물 포전거래 시스템의 네트워크 구조는 농부-산지유통인 채널, 농부-검증기관 채널로 2개의 채널이 존재하도록 설계하였다. 각 조직

은 하나의 peer로 구성되어 있으며 각 peer들은 체인 코드, 원장을 관리하고 있다. 시스템 사용자들은 peer 내의 체인 코드를 이용하여 자신의 원장에 접근 가능하다. 사용자들이 DApp을 통해 트랜잭션을 요청하면 DApp은 SDK를 통해 각 기관들과 연결한다. 각 기관의 peer들은 자신의 체인 코드를 호출하여 분산원장에 접근 후 트랜잭션을 실행한다. 트랜잭션 실행 후에는 그 결과값을 SDK를 통해 다시 DApp으로 반환하여 사용자가 볼 수 있도록 한다.



<그림 8> 스마트 컨트랙트 기반 포전거래 시스템 네트워크 구조 설계

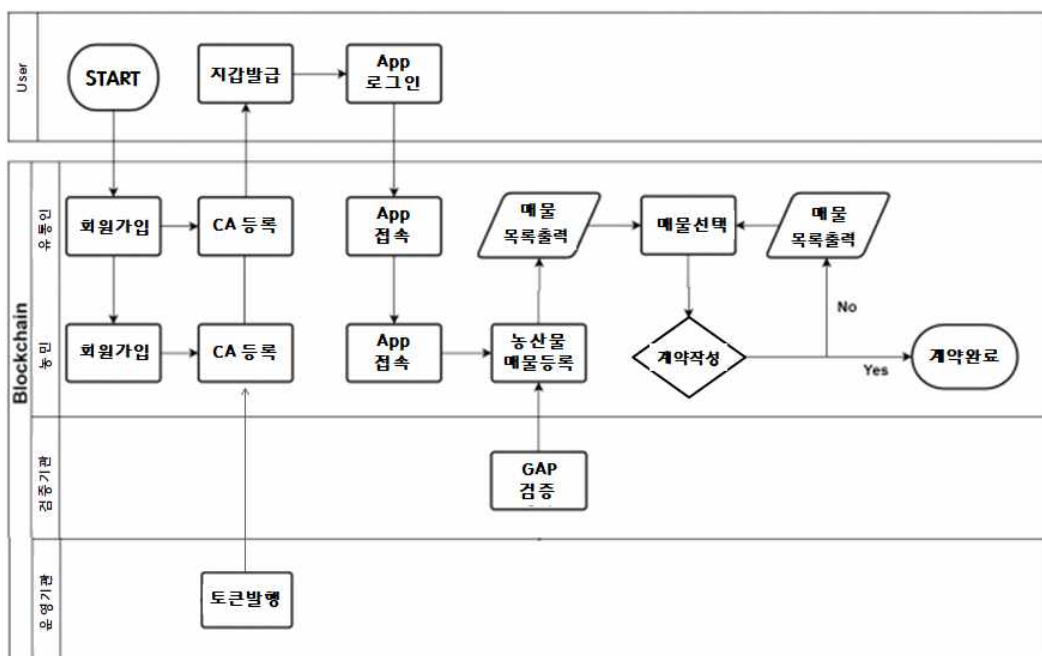
각 조직은 <표 4>처럼 자신이 가진 data를 블록체인으로 관리하며 위변조를 방지한다. 또한, 시스템 사용자가 DApp을 사용하면서 추가적으로 생성되는 상품 거래 내역에 관한 데이터들도 블록체인으로 관리함으로써 상호 간의 신뢰성을 높인다. 본 과제에서의 데이터베이스는 CouchDB를 사용하였다.

기관	자산
농민	농민 회원가입 정보 및 거래 농산물 내역, 매물 내역
산지유통인	산지유통인 회원가입 정보 및 거래 농산물 내역, 구매한 매물 내역
운영기관	거래 토큰 발행
검증기관	GAP 인증 정보

<표 4> 기관별 블록체인 자산

### 3.3 스마트 컨트랙트 기반 포전거래 시스템 DApp 설계

<그림 9>는 3.1절에서 설명한 네트워크 구조를 바탕으로 구현한 DApp의 Swimlane이다. 농민과 산지유통인은 회원가입 시 CA에 자신의 신원 정보를 등록하고 인증 받는다. 이후 지갑을 발급받으면 농민과 산지유통인은 DApp 접속이 가능해진다. 농민은 자신이 판매하고 싶은 매물과 포전거래에 대한 정보를 입력할 수 있는데, 이 때 국립농산물품질관리원에서 발급받은 GAP 인증번호를 입력해야 한다. 이 인증번호가 시스템의 데이터베이스에 저장되어 있는 GAP 인증데이터와 일치하면 거래 제시가 완료된다. 산지유통인은 농민들이 입력한 거래 목록을 보고 자신이 원하는 농산물에 대한 거래를 선택한 후, 계약서 작성을 통해 거래가 성사된다.



<그림 9> 스마트 컨트랙트 기반 포전거래 시스템 DApp Swimlane

### 3.4 스마트 컨트랙트 기반 포전거래 시스템 구현

#### 3.4.1 개발 환경

본 과제는 Ubuntu 가상 환경을 이용하여 개발하였다. <표 5>에 그 외의 개발 환경에 대하여 기술해두었다. Hyperledger Fabric 네트워크에서 체인코드는 Go 언어로 개발하였고, DApp의 API 서버 구현에는 JavaScript 기반의 라이브러리 중 NodeJS를 사용하였다. 또한 개발의 편리성과 효율을 위해 Docker를 사용하였다. 또한, 본 과제에서 사용하는 data는 Hyperledger Fabric에서 지원하는 데이터베이스 중 JSON 포맷 저장방식의 CouchDB를 사용하여 관리하였다.

개발 도구	버전
OS	Ubuntu 20.04 LTS
Docker	19.03.8
Docker-compose	2.2.2
NPM	8.12.1
Node	18.4.0
Go	1.12.10
Visual Studio Code	1.71
Hyperledger Fabric	2.0
CouchDB	3.1.1

<표 5> 개발 환경

### 3.4.2 스마트 컨트랙트 기반 포전거래 시스템 구현

본 과제의 블록체인 네트워크 구성은 4개의 조직(농민, 산지유통인, 운영기관, 검증기관)이 각각 하나의 peer를 가지며, 채널은 두 개로 구성되어 있다. 사용자는 DApp을 사용하여 SDK를 통해 블록체인 네트워크에 연결 가능하다.

스마트 컨트랙트 기반 포전거래 시스템의 모델은 크게 회원 정보, 매물 정보, 포전 거래 내역, GAP 인증번호 등으로 구성되어 있다. 보고서 상에서는 시스템의 큰 기능들을 간단히 설명한다.

#### (1) Org 구성

스마트 컨트랙트 기반 포전거래 시스템의 네트워크는 configtx.yaml 파일을 읽고 검사하는 것에서부터 시작한다. Readers, Writers, Amins, Endorsement에 각각 정책을 설정해주는 것이다. <그림 10>은 본 과제의 configtx.yaml의 일부 중 Org1의 정책이며 Endorsement를 “OR('Org1MSP.peer')”로 지정함으로써 Org1MSP 구성원인 peer가 사인해야함을 명시한다.

```

- &Org1
# DefaultOrg defines the organization which is used in the sampleconfig
# of the fabric.git development environment
Name: Org1MSP

# ID to load the MSP definition as
ID: Org1MSP

MSPDir: ../organizations/peerOrganizations/org1.14.49.119.248/msp

# Policies defines the set of policies at this level of the config tree
# For organization policies, their canonical path is usually
# /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
Policies:
  Readers:
    Type: Signature
    Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
  Writers:
    Type: Signature
    Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
  Admins:
    Type: Signature
    Rule: "OR('Org1MSP.admin')"
  Endorsement:
    Type: Signature
    Rule: "OR('Org1MSP.peer')"

```

<그림 10> Org 구성

## (2) Channel 생성 및 구성

블록체인 및 스마트 컨트랙트의 가장 중요한 개념 중 하나인 채널을 생성하였다. <그림 12>에서 보드시피 본 과제에서는 농민과 유통인 사이의 거래를 위한 채널, 농민과 검증기관 사이의 농산물 검증을 위한 채널로 총 2개를 생성하였다. Org1은 운영기관, Org2는 검증기관, Org3은 산지유통인, Org4는 농민을 나타낸다.

```

function createChannel() {
  # Bring up the network if it is not already up.
  bringUpNetwork="false"

  if ! $CONTAINER_CLI info > /dev/null 2>&1 ; then
    fatalln "$CONTAINER_CLI network is required to be running to create a channel"
  fi

  # check if all containers are present
  CONTAINERS=($(CONTAINER_CLI ps | grep hyperledger/ | awk '{print $2}'))
  len=$(echo ${#CONTAINERS[@]})

  if [[ $len -ge 4 ]] && [[ ! -d "organizations/peerOrganizations" ]]; then
    echo "Bringing network down to sync certs with containers"
    networkDown
  fi

  [[ $len -lt 4 ]] || [[ ! -d "organizations/peerOrganizations" ]] && bringUpNetwork="true" || echo "Network Running"

  if [ $bringUpNetwork == "true" ]; then
    infofn "Bringing up network"
    networkUp
  fi

  # now run the script that creates a channel. This script uses configtxgen once
  # to create the channel creation transaction and the anchor peer updates.
  scripts/createChannel1.sh trade $CLI_DELAY $MAX_RETRY $VERBOSE
  scripts/createChannel2.sh validation $CLI_DELAY $MAX_RETRY $VERBOSE
}

```

<그림 11> Channel 생성



```

TradeChannel:
  <<: *ChannelDefaults
  Orderer:
    <<: *OrdererDefaults
    Organizations:
      - *OrdererOrg
    Capabilities: *OrdererCapabilities
  Application:
    <<: *ApplicationDefaults
    Organizations:
      - *Org1
      - *Org3
      - *Org4
    Capabilities: *ApplicationCapabilities
ValidationChannel:
  <<: *ChannelDefaults
  Orderer:
    <<: *OrdererDefaults
    Organizations:
      - *OrdererOrg
    Capabilities: *OrdererCapabilities
  Application:
    <<: *ApplicationDefaults
    Organizations:
      - *Org1
      - *Org2
      - *Org4
    Capabilities: *ApplicationCapabilities

```

<그림 12> Channel 구성

### (3) 네트워크 생성 및 체인코드 배포

네트워크를 생성하고 채널에 체인코드를 배포해야 한다. network.sh라는 셸 스크립트 파일을 이용하여 Fabric 네트워크를 시작할 수 있다. deployCC 명령으로 체인코드를 배포할 채널 이름, 체인코드 이름, 배포할 체인 코드의 프로그래밍 언어 (Go), 체인코드 버전 등을 지정하여 채널에 체인코드 배포가 가능하다.

```

function deployCC() {
  scripts/deployCC1.sh trade asset ../chaincode/asset/chaincode-go/ go 1.0 1 $CC_INIT_FCN $CC_END_POLICY $CC_COLL_CONFIG $CLI_DELAY
  scripts/deployCC1.sh trade wallet ../chaincode/token/chaincode-go/ go 1.0 1 $CC_INIT_FCN $CC_END_POLICY $CC_COLL_CONFIG $CLI_DELAY
  scripts/deployCC1.sh trade receipt ../chaincode/receipt/chaincode-go/ go 1.0 1 $CC_INIT_FCN $CC_END_POLICY $CC_COLL_CONFIG $CLI_DELAY
  scripts/deployCC2.sh validation validation ../chaincode/validation/chaincode-go/ go 1.0 1 $CC_INIT_FCN $CC_END_POLICY $CC_COLL_CONFIG $CLI_DELAY

  if [ $? -ne 0 ]; then
    fatalln "Deploying chaincode failed"
  fi
}

## Call the script to deploy a chaincode to the channel
function deployCCAAS() {
  scripts/deployCCAAS.sh $CHANNEL_NAME $CC_NAME $CC_SRC_PATH $CCAAS_DOCKER_RUN $CC_VERSION $CC_SEQUENCE $CC_INIT_FCN $CC_END_POLICY

  if [ $? -ne 0 ]; then
    fatalln "Deploying chaincode-as-a-service failed"
  fi
}

```

<그림 13> deployCC 함수

#### (4) wallet 생성

토큰을 이용한 거래를 위해서는 우선 지갑인 wallet이 필요하다. 농민 및 산지유통인이 시스템 가입 후 인정 절차를 거치면 <그림 14>와 같은 구조의 wallet을 발급받게 된다. 지갑의 주소는 CreateWallet이라는 이름의 함수를 통해 owner값을 hash화하여 생성하였다.

```
type Wallet struct{
    ID          string `json:"id"`
    Owner       string `json:"owner"`
    Balance     int    `json:"balance"`
}
```

<그림 14> Wallet Model

```
func (s *SmartContract) CreateWallet(ctx contractapi.TransactionContextInterface, owner string) error {
    hash := sha256.New()
    hash.Write([]byte(owner))
    md := hash.Sum(nil)
    id := hex.EncodeToString(md)
    exists, err := s.WalletExists(ctx, id)
    if err != nil {
        return "", err
    }
    if exists {
        return "", fmt.Errorf("the wallet %s already exists", id)
    }
    balance := 0
    wallet := Wallet{
        ID:      id,
        Owner:   owner,
        Balance: balance,
    }
    return id, nil
}
```

<그림 15> CreateWallet 함수

#### (5) 토큰 생성 및 잔액 확인

스마트 컨트랙트 상의 거래는 토큰을 이용하여 진행된다. 토큰 발행은 운영기관의 역할이므로 발행 전 운영기관인 Org1이 맞는지 확인한다. 또한 <그림 17>와 같이 계정 내 토큰을 반환하고 모든 지갑의 토큰 균형을 추적하는 balanceof()라는 함수를 사용하여 토큰의 잔액을 확인하기 위한 기능도 추가하였다.

```
func (s *SmartContract) TokenInitialize(ctx contractapi.TransactionContextInterface,
    // Check minter authorization - this sample assumes Org1 is the central banker w
    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil {
        return false, fmt.Errorf("failed to get MSPID: %v", err)
    }
    if clientMSPID != "Org1MSP" {
        return false, fmt.Errorf("client is not authorized to initialize contract")
    }
}
```

<그림 16> 토큰 생성

```
func (s *SmartContract) BalanceOf(ctx contractapi.TransactionContextInterface, id string) (int, error) {
    //check if contract has been intilized first
    initialized, err := checkInitialized(ctx)
    if err != nil {
        return 0, fmt.Errorf("failed to check if contract ia already initialized: %v", err)
    }
    if !initialized {
        return 0, fmt.Errorf("Contract options need to be set before calling any function, call Initia
    }
    wallet, err := s.ReadWallet(ctx, id)
    if err != nil {
        return 0, err
    }
    return wallet.Balance, nil
}
```

<그림 17> 잔액 확인

## (6) 토큰 Transfer

거래가 결정되면 계약금 및 잔금을 전송해야 하는데 이를 위해 토큰 Transfer 기능을 구현하였다. <그림 18>과 같이 TransferToken 함수를 만들었으며 transferevent의 struct를 사용하였다. 계약금 및 잔금은 산지유통인이 농민에게 전송해야 한다.

```

func (s *SmartContract) TransferToken(ctx contractapi.TransactionContextInterface, client string,

//check if contract has been intilized first
initialized, err := checkInitialized(ctx)
if err != nil {
    return fmt.Errorf("failed to check if contract ia already initialized: %v", err)
}
if !initialized {
    return fmt.Errorf("Contract options need to be set before calling any function, call Ini
}

// Get ID of submitting client identity

err = s.transferHelper(ctx, client, recipient, amount)
if err != nil {
    return fmt.Errorf("failed to transfer: %v", err)
}

// Emit the Transfer event
transferEvent := event{client, recipient, amount}
transferEventJSON, err := json.Marshal(transferEvent)
if err != nil {
    return fmt.Errorf("failed to obtain JSON encoding: %v", err)
}
err = ctx.GetStub().SetEvent("Transfer", transferEventJSON)
if err != nil {
    return fmt.Errorf("failed to set event: %v", err)
}

return nil
}

```

<그림 18> 토큰 Transfer

## (7) 계약서 작성

산지유통인이 매물 내역을 보고 자신이 원하는 품종과 계약조건이 있으면 구매결정을 통해 거래를 시작할 수 있다. 포전거래는 농산물 포전매매 표준계약서를 기반으로 진행되어야 하기 때문에 구매 결정을 하고 거래를 시작하기 전, 계약서에 동의하여야 한다. 블록체인 내에 저장되어 있는 정보들을 통해 자동으로 계약서가 생성되며, 산지유통인이 이에 동의한다면 거래가 시작된다. 계약서의 구조는 <그림 19>와 같이 구매자 및 판매자, 계약금, 잔금, 반출일 등의 정보로 이루어져 있다. 이 계약서 생성을 위해 <그림 20>과 같이 CreateReceipt라는 함수를 생성하였다.

```

type Receipt struct {
    ID          string `json:"id"`
    AssetID     string `json:"assetID"`
    From        string `json:"from"`      //구매자
    To          string `json:"to"`        //판매자
    Balance     int    `json:"balance"`   //잔금
    DownPayment int    `json:"downpayment"` //계약금
    ReleaseDate string `json:"releasedate"` //반출일
    timestamp   string `json:"timestamp"`  //계약일
}

```

<그림 19> 계약서 Model

```

func (t *SimpleChaincode) CreateReceipt(ctx contractapi.TransactionContextInterface, assetID string) {
    hash := sha256.New()
    hash.Write([]byte(from+to+assetID+timestamp))
    md := hash.Sum(nil)
    id := hex.EncodeToString(md)
    receipt := &Receipt{
        ID:          id,
        AssetID:      assetID,
        From:         from,           //구매자
        To:          to,             //판매자
        DownPayment:  downpayment,    //계약금
        Balance:     balance,        //잔금
        ReleaseDate:  releasedate,    //반출일
        timestamp:    timestamp,      //계약일
    }
    receiptBytes, err := json.Marshal(receipt)
    if err != nil {
        return err
    }

    err = ctx.GetStub().PutState(id, receiptBytes)
    if err != nil {
        return err
    }
}

```

<그림 20> CreateReceipt 함수

#### (8) 매물(자산) 정보 생성

농민이 시스템 상에 등록한 농산물에 대한 정보 생성을 위해 Asset이란 model을 생성하였다. <그림 21>에서 볼 수 있듯이 Asset은 id, 품종명, 소재지, 재배면적 등의 정보로 이루어져 있다. 이러한 Asset을 생성하기 위한 <그림 22>와 같은 CreateAsset 함수도 생성하였다. 이 때, 자산의 ID는 자산 정보들을 hash화하여 생성하였다.

```

type Asset struct {
    ID          string `json:"id"`
    Owner       string `json:"owner"`
    Location    string `json:"location"`
    Category    string `json:"category"`
    Kind        string `json:"kind"`
    TotalPrice  int    `json:"totalprice"`
    Area        int    `json:"area"`
    Price       int    `json:"price"`
    Description bool   `json:"description"`
}

```

<그림 21> Asset Model

```

func (s *SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, owner string, location string,
    category string, kind string, totalprice int64, area int64, price int64, description string) (string, error) {
    hash := sha256.New()
    hash.Write([]byte(owner+location+category+kind+strconv.Itoa(totalprice)+strconv.Itoa(area)+strconv.Itoa(price)+description))
    md := hash.Sum(nil)
    id := hex.EncodeToString(md)

    exists, err := s.AssetExists(ctx, id)
    if err != nil {
        return "", err
    }
    if exists {
        return "", fmt.Errorf("the asset %s already exists", id)
    }
    description := true
    asset := Asset{
        ID:          id,
        Owner:       owner,
        Location:    location,
        Category:    category,
        Kind:        kind,
        TotalPrice:  totalprice,
        Area:        area,
        Price:       price,
        Description: description,
    }
    assetJSON, err := json.Marshal(asset)
    if err != nil {
        return "", err
    }
    err = ctx.GetStub().PutState(id, assetJSON)
    if err != nil {
        return "", err
    }
    return id, nil
}

```

<그림 22> CreateAsset 함수

### (9) 매물(자산) 정보 읽기

시스템 상에 저장되어 있는 자산 정보를 읽어올 수 있는 ReadAsset 함수를 생성하였다. 자산의 ID를 통해 조회 가능하다.

```

func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (*Asset, error) {
    assetJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }
    if assetJSON == nil {
        return nil, fmt.Errorf("the asset %s does not exist", id)
    }

    var asset Asset
    err = json.Unmarshal(assetJSON, &asset)
    if err != nil {
        return nil, err
    }

    return &asset, nil
}

```

<그림 23> ReadAsset 함수

#### (10) 매물(자산) 정보 Update

시스템 상에 이미 저장되어 있는 자산 정보가 수정되었을 때 시스템 상에도 반영될 수 있도록 UpdateAsset 함수를 생성하였다. 이 때에도 자산의 ID를 통해 정보 조회 가능하다.

```
func (s *SmartContract) UpdateAsset(ctx contractapi.TransactionContextInterface, id string, owner string, location
exists, err := s.AssetExists(ctx, id)
if err != nil {
    return err
}
if !exists {
    return fmt.Errorf("the asset %s does not exist", id)
}
description := true
// overwriting original asset with new asset
asset := Asset{
    ID:      id,
    Owner:   owner,
    Location: location,
    Category: category,
    Kind:    kind,
    TotalPrice: totalprice,
    Area:    area,
    Price:   price,
    Description: description,
}
assetJSON, err := json.Marshal(asset)
if err != nil {
    return err
}
return ctx.GetStub().PutState(id, assetJSON)
}
```

<그림 24> UpdateAsset 함수

#### (11) 매물(자산) Transfer

산지유통인이 매물 내역을 보고 원하는 조건의 매물을 선택하여 구매를 결정하고, 계약금을 전송한다면 시스템 상에서 매물에 대한 정보도 transfer되어야 한다. 이를 위해 <그림 25>과 같이 TransferAsset 함수를 생성하였다. 우선 ReadAsset 함수를 통해 매물 정보를 불러온 후, 매물의 owner를 바꾸어준다.



```

func (s *SmartContract) TransferAsset(ctx contractapi.TransactionContextInterface, id string, newOwner string)
asset, err := s.ReadAsset(ctx, id)
if err != nil {
    return err
}
if (asset.Description == false) {
    fmt.Errorf("This item has already been traded.")
}
asset.Description = false
asset.Owner = newOwner
assetJSON, err := json.Marshal(asset)
if err != nil {
    return err
}

err = ctx.GetStub().PutState(id, assetJSON)
if err != nil {
    return err
}

return nil
}

```

<그림 25> 매물 Transfer

## (12) 모든 매물(자산) 정보 조회

시스템에 저장되어 있는 모든 자산 정보를 한 번에 조회 가능한 GetAllAsset 함수를 생성하였다. Iterator를 사용하여 모든 자산 조회가 가능하다.

```

func (s *SmartContract) GetAllAssets(ctx contractapi.TransactionContextInterface) ([]*Asset, error) {
    // range query with empty string for startKey and endKey does an
    // open-ended query of all assets in the chaincode namespace.
    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil {
        return nil, err
    }
    defer resultsIterator.Close()

    var assets []*Asset
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()
        if err != nil {
            return nil, err
        }

        var asset Asset
        err = json.Unmarshal(queryResponse.Value, &asset)
        if err != nil {
            return nil, err
        }
        assets = append(assets, &asset)
    }
    return assets, nil
}

```

<그림 26> GetAllAssets 함수

## (13) GAP 인증

유통되는 우수한 농산물의 품질을 위해 시스템에 매물 등록 시 GAP 인증번호를 입력해야 하며, 이를 데이터베이스에 저장된 인증번호와 일치한지 여부를 판단 후에 매물로 등록이 가능하도록 시스템을 구성하였다고 했다. 이를 위해 <그림 27>과 같이 GAP 인증번호, 품명, 위치 등의 정보를 가지고 있는 GAP model을 생성하였으며 입력받은 인증번호가 시스템 내에 존재하는지 여부를 판단하는 GapExists 함수



수를 생성하였다.

```
type Gap struct {
    GapNum:    string `json:"gapnum"`
    Name:      string `json:"name"`
    Kind:      string `json:"kind"`
    Location:  string `json:"location"`
    Area:      string `json:"area"`
}
```

<그림 27> GAP model

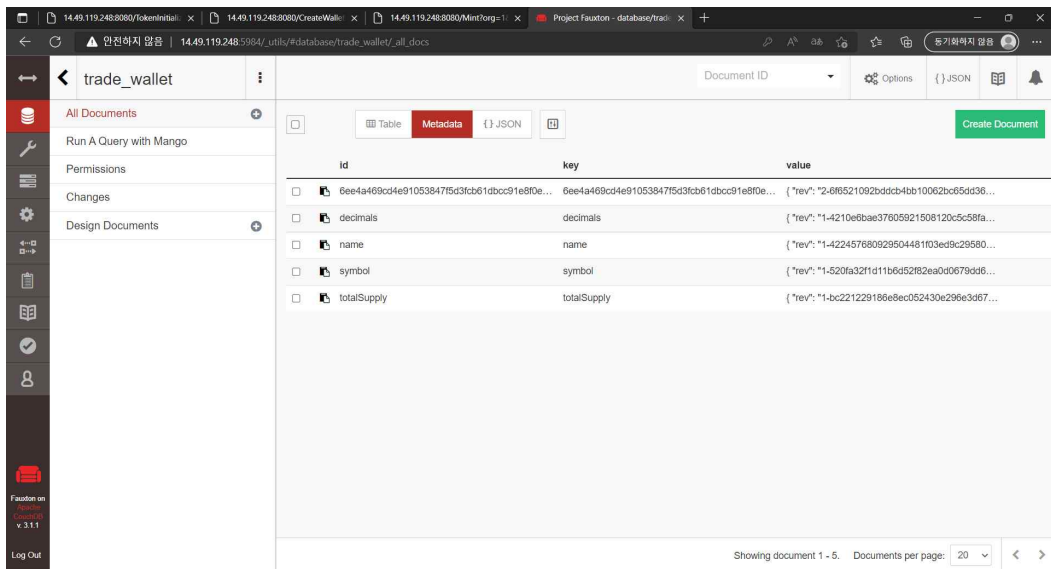
```
func (s *SmartContract) GapExists(ctx contractapi.TransactionContextInterface, gap string,
gapJSON, err := ctx.GetStub().GetState(id)
if err != nil {
    return false, fmt.Errorf("failed to read from world state: %v", err)
}
if gapJSON == nil {
    return false, fmt.Errorf("the Gap %s does not exist", id)
}

var gap Gap
err = json.Unmarshal(gapJSON, &gap)
if err != nil {
    return false, err
}
if (gap.Name != name ){
    return false, fmt.Errorf("the Gap %s information error", id)
}
if (gap.Kind != kind ){
    return false, fmt.Errorf("the Gap %s information error", id)
}
if (gap.Area - location != 0 ){
    return false, fmt.Errorf("the Gap %s information error", id)
}
if (gap.Area - area != 0 ){
    return false, fmt.Errorf("the Gap %s information error", id)
}
return true, nil
}
```

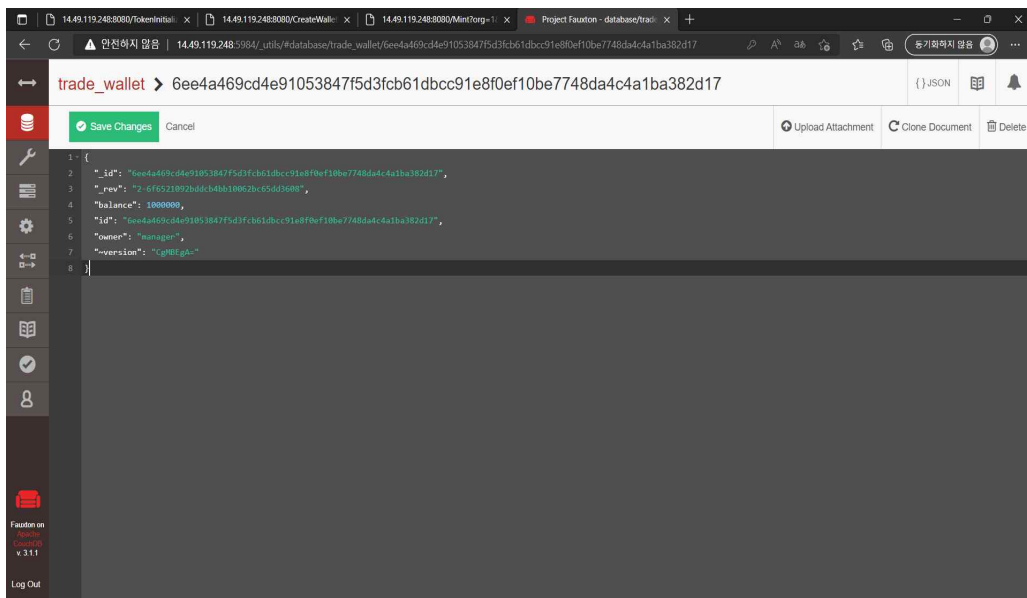
<그림 28> GapExists 함수

#### (14) CouchDB

본 과제에서 사용하는 data는 Hyperledger Fabric에서 지원하는 데이터베이스 중 JSON 포맷 저장방식의 CouchDB를 사용하여 관리하였다. <그림 29>에서 CouchDB에 저장되어 있는 JSON형식의 데이터들을 볼 수 있다. <그림 30>은 토큰을 initialize 하였을 때의 토큰 발행 관련 정보들을 나타낸다. <그림 31>은 Hyperledger Fabric 상에서 CouchDB를 설정하는 코드의 일부이다.



<그림 29> Couch DB 데이터



<그림 30> 토큰 initialize 했을 때

```

services:
  couchdb0:
    container_name: couchdb0
    image: couchdb:3.1.1
    labels:
      service: hyperledger-fabric
    # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
    # for CouchDB. This will prevent CouchDB from operating in an "Admin Party" mode.
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=adminpw
    # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
    # for example map it to utilize Fauxton User Interface in dev environments.
    ports:
      - "5984:5984"
    networks:
      - test

  peer0.org1.14.49.119.248:
    environment:
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
      # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
      # provide the credentials for ledger to connect to CouchDB. The username and password must
      # match the username and password set for the associated CouchDB.
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
    depends_on:
      - couchdb0

  couchdb1:
    container_name: couchdb1
    image: couchdb:3.1.1

```

<그림 31> CouchDB 설정

#### (15) API 서버 구축

Hyperledger Fabric SDK는 블록체인과 상호작용할 수 있는 강력한 API를 제공한다. 이 API를 이용하여 채널을 만들거나 체인 코드 설치, 트랜잭션 호출 등 다양한 동작을 할 수 있다. User들은 DApp을 이용하여 트랜잭션을 호출하면 Client에 있는 웹이 서버에 있는 Hyperledger Fabric Client SDK, Rest API를 이용하여 체인 코드까지 연결되어 블록체인 네트워크에 접속 가능하다. <그림 33>의 gateway는 네트워크 컴포넌트 간의 transaction proposal, ordering, notification 등의 역할을 함으로써 트랜잭션 생성, 제출 등에 기여한다.

```

module.exports = function(app) {

  app.get('/GetAllAssets', function (req, res) {
    var org = req.query.org;
    let args = [];
    AssetOrgController(org, false, 'GetAllAssets', args, res);
  });

  app.get('/AssetExists', function (req, res) {
    var org = req.query.org;
    var id = req.query.id;
    let args = [id];
    AssetOrgController(org, false, 'AssetExists', args, res);
  });

  app.get('/DeleteAsset', function (req, res) {
    var org = req.query.org;
    var id = req.query.id;
    let args = [id];
    AssetOrgController(org, true, 'DeleteAsset', args, res);
  });

  app.get('/ReadAsset', function (req, res) {
    var org = req.query.org;
    var id = req.query.id;
    let args = [id];
    AssetOrgController(org, false, 'ReadAsset', args, res);
  });

  app.get('/TransferAsset', function (req, res) {
    var org = req.query.org;
    var id = req.query.id;
    var newOwner = req.query.newOwner;
    let args = [id, newOwner];
    AssetOrgController(org, true, 'TransferAsset', args, res);
  });
}

```

<그림 32> 서버 컨트롤러

```

async function initGatewayForOrg4() {
  console.log(`${GREEN}--> Fabric client user & Gateway init: Using Org4 identity to Org4 Peer${RESET}`);
  const ccpOrg4 = buildCCPOrg4();
  const caOrg4Client = buildCAClient(FabricCAServices, ccpOrg4, 'ca.org4.14.49.119.248');

  const walletPathOrg4 = path.join(__dirname, 'wallet', 'org4');
  const walletOrg4 = await buildWallet(wallets, walletPathOrg4);

  await enrollAdmin(caOrg4Client, walletOrg4, org4);
  await registerAndEnrollUser(caOrg4Client, walletOrg4, org4, Org4UserId, 'org4.department1');

  try {
    // Create a new gateway for connecting to Org's peer node.
    const gatewayOrg4 = new Gateway();
    await gatewayOrg4.connect(ccpOrg4,
      { wallet: walletOrg4, identity: Org4UserId, discovery: { enabled: true, asLocalhost: true } });
    return gatewayOrg4;
  } catch (error) {
    console.error(`Error in connecting to gateway for Org4: ${error}`);
    process.exit(1);
  }
}

```

<그림 33> SDK의 gateway init

```

async function send4(type, func, args, res) {
  try {
    /** ***** Fabric client init: Using Org4 identity to Org4 Peer ***** */
    const gatewayOrg4 = await initGatewayForOrg4();
    const networkOrg4 = await gatewayOrg4.getNetwork(channelName);
    const contractOrg4 = networkOrg4.getContract(chaincodeName);
    try {
      if(type == true) { // type true : submit transaction, not only query
        let result = await contractOrg4.submitTransaction(func, ...args);
        const resultJSONString = result.toString();
        res.send(resultJSONString);
        console.log('Submit transaction success');
      } else {
        const result = await contractOrg4.evaluateTransaction(func, ...args);
        console.log('Evaluate transaction success');
        const resultJSONString = prettyJSONString(result.toString());
        console.log(`${resultJSONString}`);
        const resultString = result.toString();
        console.log(`${resultString}`);
        res.send(resultJSONString);
      }
    }
    finally {
      // Disconnect from the gateway when the application is closing
      // This will close all connections to the network
      gatewayOrg4.disconnect();
    }
  } catch (error) {
    console.error(`***** FAILED to run the application: ${error}`);
  }
}

```

<그림 34> SDK 네트워크의 트랜잭션 submit or evaluate

### 3.4.3 스마트 컨트랙트 기반 포전거래 시스템 DApp 구현

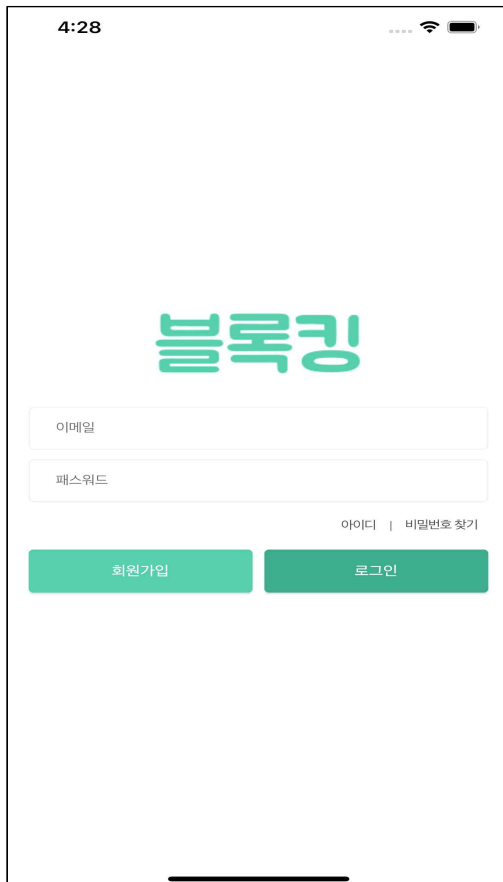
DApp 구현을 위해서는 Flutter를 사용하였다. Flutter는 구글에서 개발한 무료 오픈소스 모바일 UI 프레임워크이며 하나의 코드 베이스로 안드로이드, iOS, 리눅스, Windows, macOS 및 웹 브라우저에서 모두 동작되는 앱을 개발 가능하다. ‘스마트 컨트랙트 기반 농산물 포전거래 시스템’이라는 목적에 맞게 DApp의 UI를 초록색 계열로 꾸몄다. <그림 35>는 본 과제에서 개발한 DApp의 초기화면이다.



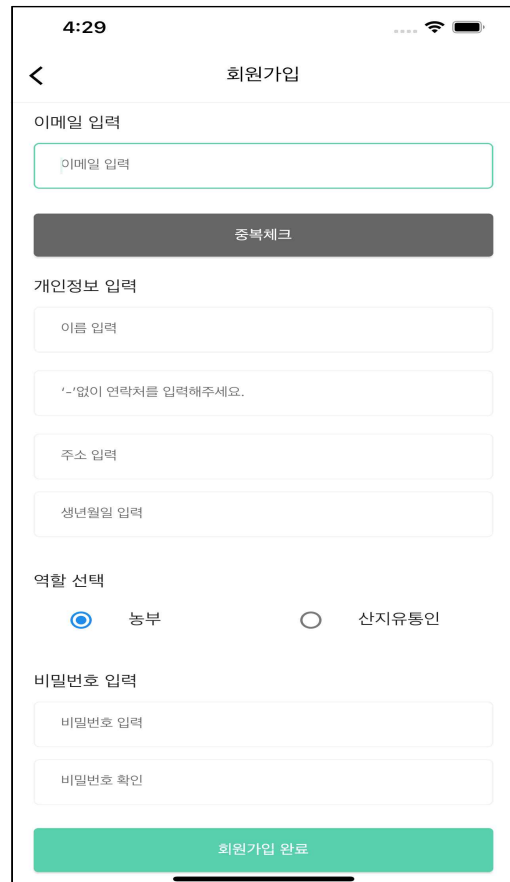
<그림 35> DApp 초기화면

#### (1) DApp 로그인 및 회원가입

<그림 36>은 로그인 화면으로 User 정보 입력 시 체인코드에서 User 정보를 검색하여 정보가 일치하는지 확인한다. 만약 정보가 일치한다면 로그인이 완료되며, 아직 User로 등록되어 있지 않은 경우에는 <그림 37>처럼 회원가입을 진행함으로써 체인코드에 회원 정보를 등록해야 한다.



<그림 36> DApp 로그인 화면



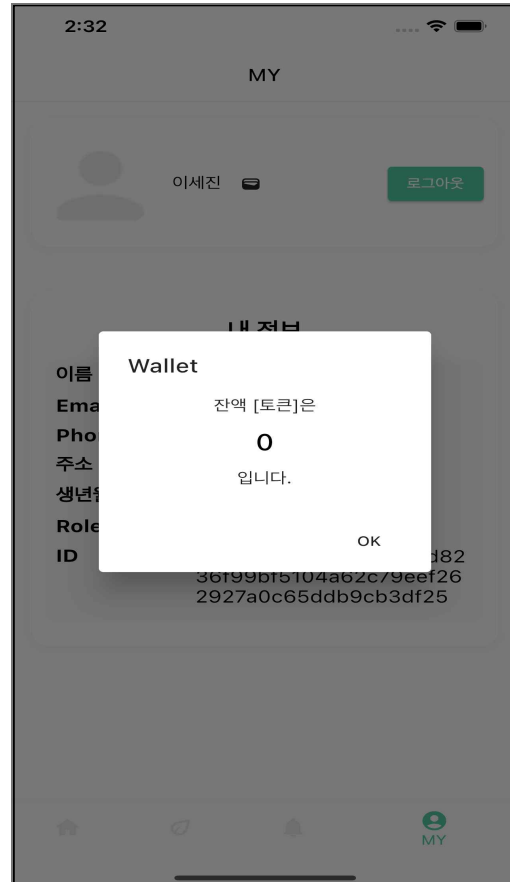
<그림 37> DApp 회원가입 화면

## (2) 회원 정보 내역

회원 가입 후 "My"버튼을 누르고 들어가면 사용자 본인의 정보에 대해 볼 수 있다. 사용자의 이름부터 Email, 생년월일 등을 알 수 있으며 Hyperledger Fabric 상의 동작을 위해 DApp 가입 시 개인별로 주어지는 ID가 무엇인지도 확인 가능하다. 이는 <그림 38>과 같다. 또한 <그림 39>처럼 이름 옆의 지갑 모양의 버튼을 누르면 거래에 필요한 토큰을 얼마나 가지고 있는지 잔액을 확인 가능하다.



<그림 38> DApp User 정보

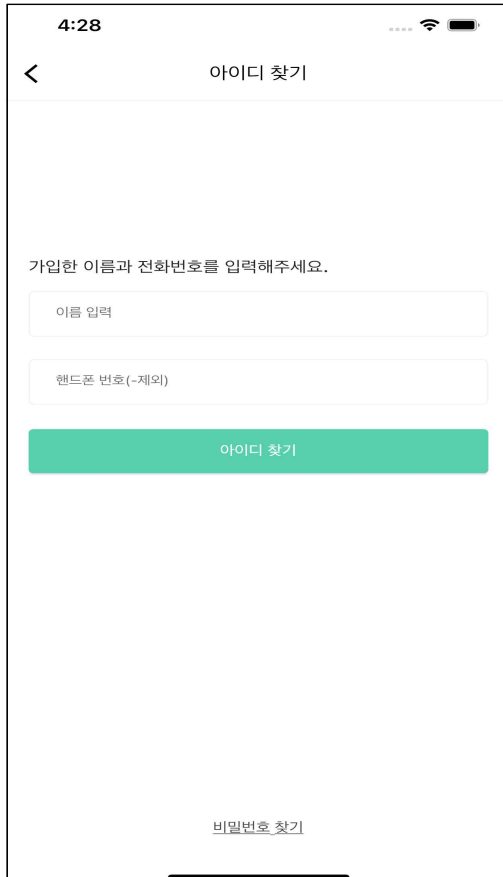


<그림 39> DApp 지갑 정보



### (3) 아이디 및 비밀번호 찾기

아이디 및 비밀번호를 잊어 DApp에 로그인할 수 없다면 아이디 및 비밀번호 찾기 기능을 이용하여 체인코드 상에 저장되어 있는 User 정보를 조회함으로써 알아낼 수 있다.



4:28

< 아이디 찾기

가입한 이름과 전화번호를 입력해주세요.

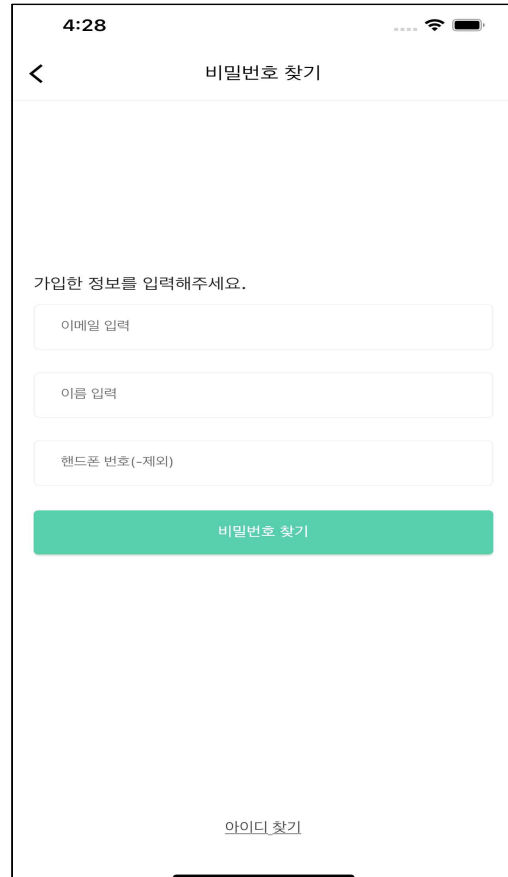
이름 입력

핸드폰 번호(-제외)

아이디 찾기

비밀번호 찾기

<그림 40> DApp 아이디 찾기 기능



4:28

< 비밀번호 찾기

가입한 정보를 입력해주세요.

이메일 입력

이름 입력

핸드폰 번호(-제외)

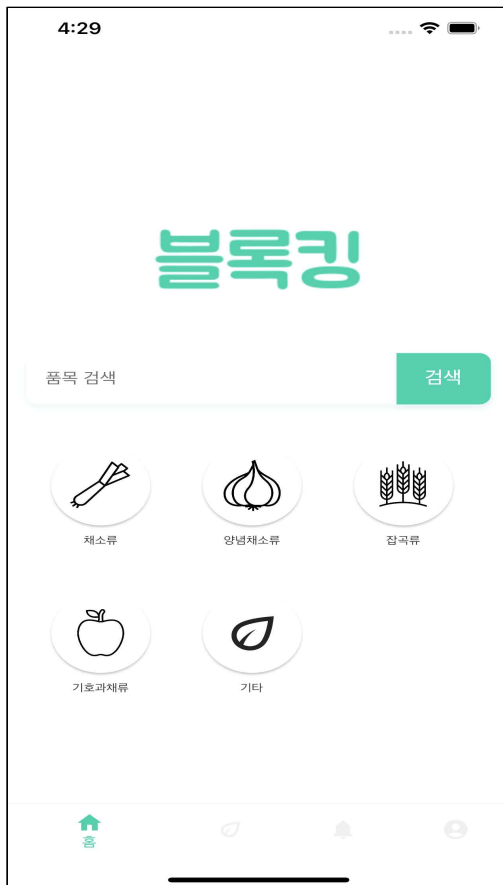
비밀번호 찾기

아이디 찾기

<그림 41> DApp PW 찾기 기능

### (4) 메인화면 및 매물 내역 확인

농부 혹은 산지유통인의 자격으로 DApp에 가입한 뒤 볼 수 있는 첫 화면은 <그림 42>와 같다. 원하는 품목을 검색하거나 채소류, 잡곡류 등 분류별로 버튼을 누르면 어떠한 매물들이 올라와있는지 확인 가능하다. <그림 43>은 ‘매매’ 버튼을 누르면 볼 수 있는 화면으로 사용자들이 게시한 매물 내역에 대한 글들을 확인 가능하며, 원하는 조건의 농산물에 대해 구매 요청을 통해 포전 거래 계약 진행이 가능하다.



<그림 42> DApp 메인 화면



<그림 43> DApp 매물 내역

#### (5) 매물 등록

농민인 User들은 자신이 거래를 원하는 농산물을 DApp 상에 등록 가능하다. <그림 44>와 같이 품종명, 밭의 면적, 소재지, 최종출거일 등의 정보를 입력할 수 있다. User가 자신의 GAP 인증번호를 입력하면 시스템 내 CouchDB에 저장되어 있는 GAP 인증 data를 불러오게 되고, CouchDB에 저장되어 있는 인증 번호와 일치한다면 등록 가능하다. 이를 통해 GAP 인증번호 허위 인증 등과 같은 범죄 예방이 가능해지며, 국립농산물품질관리원부터 인증을 받은 우수한 농산물들 간의 거래가 가능해진다.

4:31

<

판매 등록

제목

제목 입력

품종명

품종명 입력

종류

종류 입력(채소류, 양념채소류, 잡곡류, 기호과채류, 기타)

면적

면적 입력(평수)

소재지

소재지 입력(주소)

재배방식

재배방식 입력(저농약, 관행...)

최종출거일

최종출거일

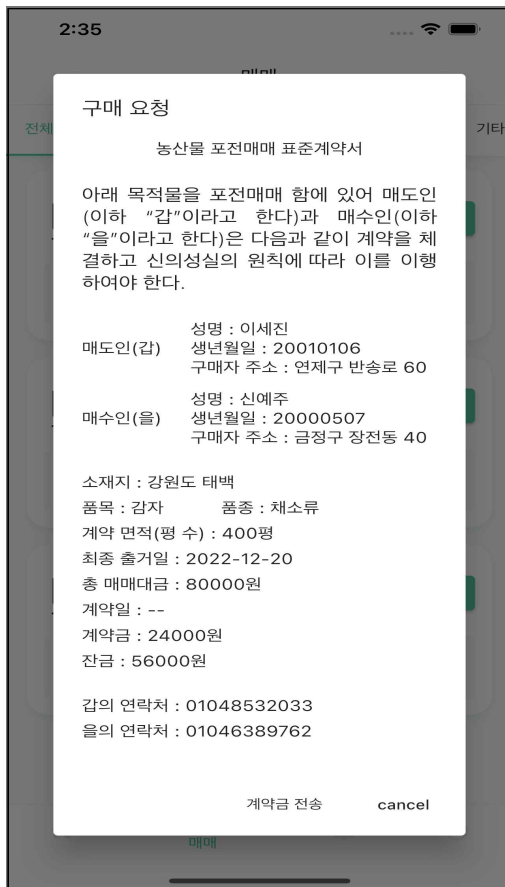
평당 가격

평당 가격 입력

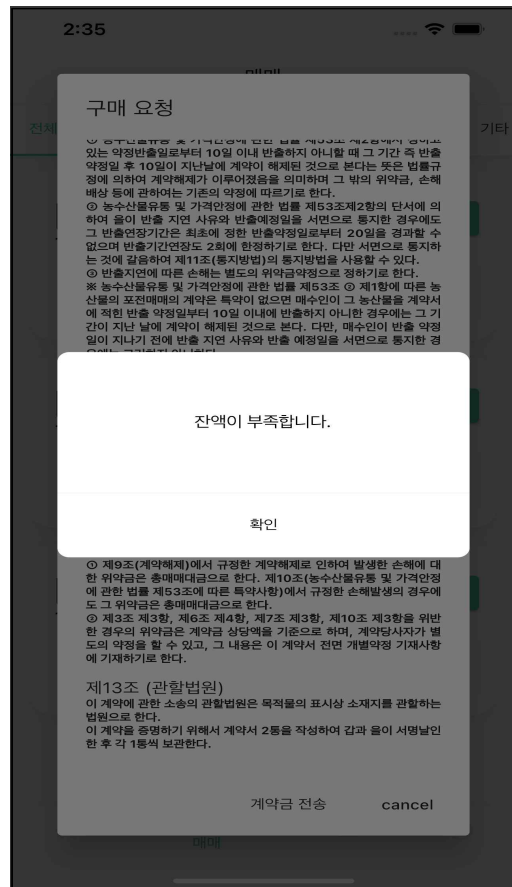
<그림 44> DApp 매물 등록

## (6) 거래 체결

산지유통인들은 농산물을 구매하기 위해 앞서 <그림 43>과 같은 매물 목록을 보고 원하는 품목과 계약조건에 대해 ‘구매 요청’을 할 수 있다. 포전거래는 농산물 포전매매 표준계약서를 기반으로 진행되어야 하며, 스마트 컨트랙트는 계약조건 충족 시 계약이 자동 체결되는 시스템이기 때문에 이 시스템 상에서도 계약서의 중요성이 크다. 산지유통인이 원하는 농산물에 대해 구매 요청을 하면 자동으로 블록체인에서 농민의 이름, 생년월일, 주소, 연락처의 data를 가져와 계약서를 생성한다. 산지유통인이 계약서 상의 품목, 계약 면적, 계약금 등의 정보를 확인한 뒤 ‘계약금 전송’을 누르면 토큰이 계약금만큼 농부에게로 이동한다. 만약 계정의 토큰이 부족하여 계약금 전송이 불가능하다면 <그림 46>과 같이 잔액이 부족하다는 문구가 뜨며 계약 진행이 불가능하다.



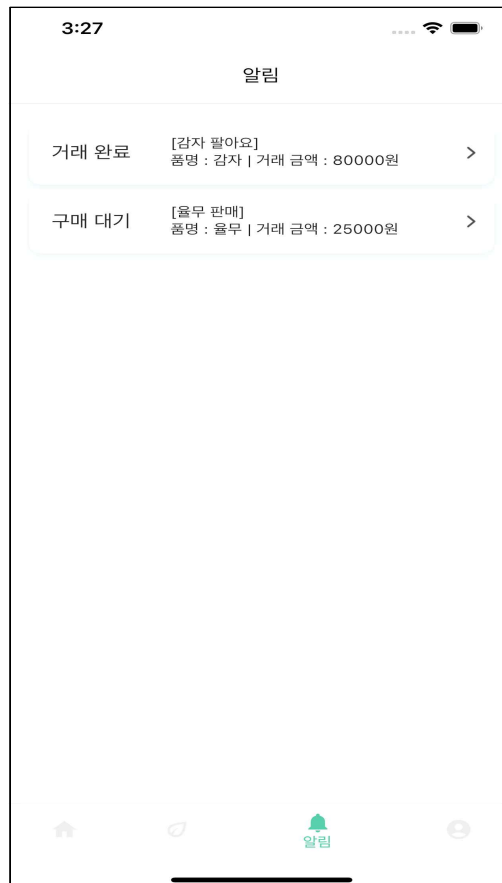
<그림 45> DApp 구매 요청



<그림 46> DApp 잔액 확인

## (6) DApp 내 알림

<그림 47>과 같은 ‘알림’ 화면에서는 자신의 거래 상황에 따라 알림을 받을 수 있다. 아직 잔금이 처리되지 않아 거래가 완료되지 않은 상태에 대해서는 구매 대기, 잔금 처리가 완료된 건에 대해서는 거래 완료 등 현재 거래 상황에 대한 알림을 받아볼 수 있다.



<그림 47> DApp 알림

#### (7) MySQL

User가 회원 가입 시 입력했던 회원 정보와 DApp 알림을 위한 거래 내역 저장은 MySQL을 사용하였다. 회원가입 시 입력받은 user의 기본 정보인 이름, 주소, 연락처, role(농민, 산지유통인) 등을 저장하는 user\_info와 Id, 거래날짜, 금액 등의 정보를 저장하는 transaction을 생성하였다. 회원가입 시 user의 기본 정보를 데이터베이스에 저장하고, 거래 알림을 띄우기 위한 정보들을 저장하여 각 user한테 구매대기, 거래요청 등의 알림을 띄울 수 있도록 하였다.

Name: user\_info Schema: blockking

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
email	VARCHAR(30)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
password	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
name	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
phone	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
address	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
birth	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
role	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
id	VARCHAR(64)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
<click to edit>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

<그림 48> 회원가입 정보

Name: transaction Schema: blockking

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
assetId	VARCHAR(64)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fromId	VARCHAR(64)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
told	VARCHAR(64)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
agree	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
balance	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
date	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
<click to edit>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

<그림 49> 거래 정보

---

## 4. 결론 및 향후 연구

본 과제에서는 농민과 산지유통인 간의 포전거래 계약 불이행과 불공정 거래 등을 방지하고 불법 산지유통인의 활동을 방지하고자 스마트 컨트랙트 기반의 농산물 포전거래 시스템을 제안 및 구현하였다. GAP 인증 정보와 거래 내역 정보 등 포전 거래 시 발생하는 많은 데이터를 블록체인으로 관리하며 거래에 대한 신뢰도를 높였다. 본 과제에서는 허가된 사용자만 참여할 수 있는 블록체인을 구현함으로써 거래 관련 데이터를 위변조 불가하므로 시스템에 대한 신뢰도를 더욱 증가시켰다.

본 과제의 의의는 기존 포전거래 제도에 블록체인 기술을 접목시킴으로써 거래에 대한 신뢰도를 높이고 다양한 기대효과를 불러온다는 점에 있다. 여러 블록체인 중 Hyperledger Fabric을 사용함으로써 허가된 사용자만 네트워크에 참여할 수 있게 하여 효율성을 높였다. 또한 스마트 컨트랙트 기반의 시스템으로써 계약 불이행, 불공정 계약 등의 위험이 줄어들었기에 농민과 산지유통인, 넓게 보면 시장 및 정책당국에게도 긍정적인 효과를 미칠 수 있다. 농민에게는 공정 포전거래를 통한 농산물 가격변동 위험의 안정적인 헤징(hedge) 가능, 포전거래 과정 시 신용위험 감소로 인한 농작물의 담보가치 증가, 포전매매 가격 결정 과정에서의 정보비대칭성 해소, 계약 과정의 주도적 참여(기존의 산지유통인 중심의 계약 관행 타파) 등의 긍정적인 기대효과 있다. 산지유통인에게는 포전거래 체결을 위한 탐색 및 협상비용 절약, 미등록 불법 산지유통인의 양성화 유도, 포전거래 계약의 안정성 및 담보성 증가 등의 효과를 미칠 것이다. 시장 및 정책당국에게는 불법·불공정 거래 관행 타파, 불법 산지유통인의 소득탈루 방지, 농업·유통의 정보화 실현이 가능하다는 기대효과가 있다.

본 연구는 가상환경에서 구현하였기에 아직 테스트 단계이며 거래의 최소 기능만 구축하였기에 상용화하기에는 여러 문제점이 존재한다. 거래 성사 및 잔금 처리 이후에 발생하는 여러 상황들을 시나리오화하여 대응 및 관리할 필요가 있다. 또한 본 시스템의 범위를 포전거래에 국한하지 않고 다양한 채널의 추가를 통한 다양한 거래로 확장시킬 수 있다.

## 5. 구성원별 역할 및 개발 일정

### 5.1 구성원별 역할

이 름	역할
이세진	<ul style="list-style-type: none"> <li>- Flutter ios/android 앱 개발</li> <li>- DApp 퍼블리싱 및 상세 기능 구현</li> </ul>
김재현	<ul style="list-style-type: none"> <li>- 블록체인 네트워크 구축 (네트워크 조직 정의, 채널 정의, artifacts 생성 등)</li> <li>- 스마트 컨트랙트 체인코드 개발</li> </ul>
신예주	<ul style="list-style-type: none"> <li>- AWS 서버 구축 및 SSH 서버 구축</li> <li>- 유저 데이터베이스 정의 및 구축</li> <li>- DApp UI 디자인</li> <li>- 보고서 작성</li> </ul>

### 5.2 개발 일정

6월		7월					8월					9월				
4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주
블록 체인 스터디 [부동산 Dapp 개발]																
		서버&DB 구축														
		앱 UI 디자인														
		하이퍼레저 패브릭 체인코드 개발														
				중간 보고서 작성												
				앱 퍼블리싱 및 개발												
						API 웹 서버 개발										
										rest API 앱 연동						
										테스트 및 디버깅						
										오류 수정						
														최종 발표 준비 및 보고서 제작		



---

## 참 고 문 헌

### <국내문헌>

- 국경완(2019), “블록체인 기술 및 산업 분야별 적용 사례”, 주간기술동향, 13-27
- 이동영, 박지우, 이준하, 이상록, 박수용(2017), “블록체인 핵심 기술과 국내외 동향”, 정보과학회지, 22-28
- 박상곤, 최영동, 이채원, 정명진, 김정숙, 정덕화, 심원보(2015), “Good Agricultural Practices (GAP) 모델 개발을 위한 부추 및 생산환경에서의 위해요소 조사”, 한국식품위생안전성학회지, 28-34
- 하용현(2009), “농산물 파생상품거래의 도입 가능성 검토 : 선물거래 및 옵션계약을 중심으로”, 농촌경제, RE32-6-106
- 양승룡, 서경남, 이춘수(2009), “농산물 선도거래 청산소 도입 방안”, 유통연구, 14(5), 83-106
- 아카하네 요시히루, 아이케이 마나부, 이소 토모히로, 혼다 히로시, 노지마 리이치로, 호조 마사시, 시바타 마사유키, 이나바 타카히로, 나리키요 요시히로, 나가이 야스토시(양현 옮김)(2017), **블록체인 구조와 이론 : 예제로 배우는 핀테크 핵심 기술**, 파주, 위키북스
- 시미즈 토모노리, 타마치 교코, 우에노하라 하야토, 사토우 타쿠요시, 사이토신, 콘도 히토시, 하라야마 츠요시, 카사하라 아키히로, 이와사키 타츠야, 오가사와라 카즈유키(양현, 장승일, 연구흠 옮김)(2019), **하이퍼레저 패브릭 철저 입문 : Hyperledger Fabric을 이용한 블록체인 기반 시스템 구축과 운용**, 경기도, 위키북스
- 윤대근(2018), **하이퍼레저 패브릭으로 배우는 블록체인**, 파주, 제이펍
- 이재홍(2015), **가장 빨리 만나는 Go 언어**, 서울, 길벗
- 장재휴(2016), **Go 언어 웹 프로그래밍 철저 입문**, 서울, 길벗
- 정인용(2018), **Do it! 자바스크립트 + 제이쿼리 입문**, 전면개정판, 서울, 이지스퍼블리싱

---

<국외문헌>

Szabo, N.(1997), Formalizing and securing relationships on public networks.  
First Monday, 2(9).

<웹사이트>

Hyperledger Fabric Documentation,  
<https://hyperledger-fabric.readthedocs.io/en/latest/>

국립농산물 품질관리원 GAP정보서비스,  
<https://www.gap.go.kr/portal/main/main.do>