Team No.

PKNU실전문제연구팀(X-Corps) 최종연구결과 보고서

연구과제명	IoT 센서를 이용한 블록체인 기반 식품 유통망 관리 시스템 연구		
팀명(팀구분)	LACUC(단일팀)	구성인원	총 (7)명 지도교수(1), 대학원생(1), 학부생(4)명, 산업체(1)명
연구 주제 구분	☐ 지정주제(연구팀)	☐ 자유주제(연구팀)	
	☐ 기계.부품소재 ☑ 인공지능 및 블록체인 스마트시티 ☐ 친환경에너지 생산 및 저장	☐ 도전적 연구주제 발굴 분야 과제 ☐ 자발적인 창의적 아이디어 제안 과제 ☐ 산업체 현장문제 해결형 R&D	
	택1		

팀장	전공(학과)/과정학기	성명	연락처(휴대폰)	이메일
	정보보호학협동과정 / 3	정병규	010-2221-2502	holine0622@pukyong.ac.kr

구분	소속학과	학번	학년	성명	휴대폰	이메일
팀원 1	IT융합응용공학과	201412146	4	김왕록	010-8930-2795	korgnawmik@naver.com
팀원 2	IT융합응용공학과	201712057	4	전미현	010-9123-8798	jmh3850@naver.com
팀원 3	IT융합응용공학과	201512119	4	오동의	010-8524-1451	deo1915@naver.com
팀원 4	IT융합응용공학과	201512114	4	심재익	010-6292-5346	wodlr2007@naver.com
팀원 5						

멘토					
	지도교수	전공(학과)	성 명	연락처(휴대폰)	이메일
		IT융합응용공학과	신상욱	010-2574-3954	shinsu@pknu.ac.kr
	산업체 멘토	업체명	성 명	연락처(휴대폰)	이메일
		㈜스마트엠투엠	김동규	010-7229-7628	donggyukim@smartm2m.co.kr

부경대학교 동남권 지역산업 밀착형 PKNU 실전문제연구단 과제를 성실하게 수행한 최종결과보고서를 제출합니다.

◆ 개인정보의 수집 및 이용에 동의합니다.

○ 수집·이용목적 현장맞춤형 이공계 인재양성사업의 실전문제연구팀 참여자 최종결과 자료로 활용합니다.

○ 수집·이용할 개인정보 항목

- 필수정보 (성명, 소속, 학과, 학년, 전공, 전화번호, 이메일)

○ 보유·이용기간

- 위 개인정보는 수집·이용에 관한 동의일로부터 5년(60개월)간 보유·이용되며 기간 경과 후 바로 파기합니다.

○ 동의를 거부할 권리 및 동의를 거부할 경우의 불이익

- 개인정보 제공 및 활용 동의를 거부할 수 있으나, 미동의시 차년도 현장맞춤형 이공계 인재양성사업의 실전문제연구팀 지원이 불가할 수 있음을 유념하시기 바랍니다.

☑ 위 사항에 동의합니다.(지도교수) ☑ 위 사항에 동의합니다. (산업체멘토)
☑ 위 사항에 동의합니다. (팀장) ☑ 위 사항에 동의합니다. (팀원1) ☑ 위 사항에 동의합니다. (팀원2)
☑ 위 사항에 동의합니다. (팀원3) ☑ 위 사항에 동의합니다. (팀원4) ☑ 위 사항에 동의합니다. (팀원5)

2020년 04월 17일

(팀 장) 정 병 규 (인 또는 서명)
(지도교수) 신 상 욱 (인 또는 서명)

동남권 지역산업 밀착형 PKNU 실전문제 연구단장 귀하

1. 세부 보고서

PKNU실전문제연구팀(X-Corps) 최종연구결과 보고서	
연구과제명	IoT 센서를 이용한 블록체인 기반 식품 유통망 관리 시스템 연구
1. 연구 목적 및 필요성	2017년 8월 국내 살충제 달걀 파동은 1,239곳 농장 중 49곳의 농장에서 오염된 달걀이 발견된 사건이다. 정부는 이러한 달걀 파동이 일어난 후 달걀 GP 센터 유통을 의무화했고 이력 추적제, 달걀의 산란일자 표기 의무화, 난각 코드 도입등 다양한 조치를 취했지만 난각 코드의 형식이 통일되어 있지 않고 생산 농가의 추적과 유통 경로를 파악하기가 어려웠으며 달걀 GP 센터 유통을 의무화하는 데는 3년 이상의 시간이 소요된다는 단점이 있었다. 또한 미국의 시금치라 불리는 로메인 상추 유통 사고에서는 진상 규명에 2주 이상 시간이 소요되기도 했다. 중국에서는 돼지에게 약품이나 중금속이 섞이 사료를 먹여 단기간에 출하하거나 플라스틱 쌀과 같은 식품에 대한 안전성 문제가 심각하다. 이와 같이 과거의 식료품 공급 과정은 전혀 디지털화되어 있지 않고 대부분 종이 문서로 적혀있는 경우가 많다. 디지털화되어 있다 하더라도 다양한 시스템에서 상호작용이 불가능하며 상품의 종류와 수가 무수히 많아 상품관리가 복잡하고 유통과정이 투명하지 않다는 문제점이 있다. 블록체인은 탈중앙화된 데이터베이스로써, 보안성, 투명성, 추적성의 특징을 가진다. 만약 식품 유통망에 블록체인을 적용한다면 상품의 신뢰와 소비자의 안정을 보장하고 더 나아가 비용 절감과 효율성 향상 등의 긍정적인 효과를 얻을 수 있다. 유통과정에서 발생하는 데이터들을 블록체인에 기록해두면 제품의 원산지, 배송 과정, 보관 상태를 비롯한 정보들을 추적할 수 있게 된다. 이로 인해 제품의 안전 신뢰도를 향상시킬 수 있다. 또한 블록체인 기술이 진품 여부 확인, 유통 이력 추적과 상호 모니터링에 활용된다면 식품을 안전하게 유통하는데 도움이 되며 중개 수수료를 지불해야 하는 거대 플랫폼 없이도 거래가 가능해진다. 블록체인 기록을 통해 이력을 추적하면 현재의 소유권 상태와 거래 내역을 실시간으로 확인할 수 있고 관리 비용이나 검증 수수료를 내지 않고도 사기를 방지할 수 있다. 또한 허위 매물 등록이나 거래가 원천적으로 불가능하며 인증과정이 간소화되고, 소유권 이전 비용의 절감도 가능하다. 하지만 이러한 블록체인의 장점에도 해결해야 할 최초 1마일 문제가 존재한다. 유통 과정에는 블록체인에 기록이 제대로 남더라도 블록체인과 사람이 접촉하는 지점에서는 허점이 생길 수 밖에 없다. 본 연구에서는 이러한 최초 1마일 문제에 중점을 두고 해결방안을 제시한다. 데이터에 대한 신뢰성을 보장하기 위해 사람이 수기로 입력하는 데이터가 아닌 IoT 센서 데이터가 블록체인 스마트 계약에 의해 자동으로 입력된다. IoT 기기와 5G 네트워크의 발전을 통해 엣지 컴퓨팅이 가능해지고 블록체인의 과부하를 방지할 수 있다. IoT 센서가 라즈베리 파이로 전달되고 라즈베리 파이가 블록체인 네트워크에 데이터를 전달하기 위해 전처리 작업을 수행한다. 또한 허가된 노드들만 참

	여할 수 있는 허가형 프라이빗 블록체인 플랫폼인 하이퍼레저 패브릭을 개발 도구로 선정하여 사용자 및 데이터 프라이버시를 달성한다.
2. 연구과제의 수행 내용 및 방법	선행 연구 및 기술 동향 분석을 위해 첫 한 달 동안은 연구실 자체 세미나를 진행하였다. 블록체인, 스마트 컨트랙트, 합의 알고리즘, 블록체인 플랫폼, 5G, 라즈베리 파이, 커넥티드 카, 이더리움, 하이퍼레저, 식품 유통망과 유통망 블록체인 현황에 대해 세미나 발표를 진행했다. 블록체인은 P2P 네트워크에서 블록으로 연결된 데이터 구조로써, 각 블록은 체인을 형성하는 해시 포인터라는 특수 포인터로 이전 블록에 연결되는 추가 전용 시스템이다. 이는 모든 참가자가 실시간으로 기록을 확인하여 데이터 자체를 검토하고 정확성을 확인할 수 있는 영구적인 기록이기 때문에 위,변조가 불가능하고 중앙 관리자가 필요 없다는 특징을 가지고 있다. 블록체인의 종류는 크게 세가지로 분류된다. 우선 현재 대부분의 시스템은 중앙집중화되어 있다. 모든 데이터를 중앙에서 관리하기 때문에 사용자 입장에서는 편리하다는 장점이 있지만 서버라는 단일 지점에 문제가 생길 경우 시스템 전체가 위험하다는 문제점이 존재한다. 퍼블릭 블록체인은 아무나 노드로 참여할 수 있는 블록체인으로써 투명성과 공개성을 보장하지만 데이터 프라이버시를 보장하지 않는다. 또한 초당 트랜잭션 처리 속도가 매우 느려 현실 세계에 적용하기 어렵다. 퍼블릭 블록체인은 TTP(Trusted Third Party)라고 불리는 신뢰기간이 존재하지 않는다. 퍼블릭 블록체인의 예로 비트코인과 이더리움이 있다. 프라이빗 블록체인은 한 집단의 독자적인 블록체인이고 컨소시엄 블록체인은 특정한 몇몇의 집단이 참가해 해당 집단만 사용이 가능한 프라이빗 블록체인이다. 프라이빗 블록체인은 특정 노드들에게만 접근 권한이 부여되는 블록체인이다. 멤버십 서비스를 두고 신원이 보장된 노드들만 블록체인 네트워크에 접근이 가능하다. 따라서 트랜잭션과 블록의 내용이 공개되어서는 안되는 응용분야에 적합하다. 퍼블릭 블록체인과는 다르게 초당 트랜잭션 처리 속도가 높다. 하지만 참여자의 권한 부여를 위한 어쩔 수 없는 TTP가 존재한다. 프라이빗 블록체인의 예로 하이퍼레저와 넥스레저가 있다. 하이퍼레저 패브릭(Hyperledger Fabric)은 모듈형 아키텍처로서, 블록체인 솔루션과 응용 프로그램을 개발하기 위한 플랫폼이다. 하이퍼레저 패브릭은 허가받은 사용자만 참여할 수 있는 허가형 블록체인으로서, 프라이빗 블록체인의 일종이다. 오픈소스를 활용하여 비즈니스 환경의 기밀 유지와 확장성을 지원한다. 컨테이너 기술을 활용하여 시스템의 응용 프로그램 로직을 구성하는 체인코드라는 스마트 계약을 호스팅한다. 하이퍼레저 패브릭은 Java, Go, Node.js와 같은 범용 프로그래밍 언어로 작성된 스마트계약을 지원하는 최초의 분산원장 플랫폼이다. 또한 암호화폐 없이 합의 프로토콜을 활용할 수 있다. 하이퍼레저 패브릭 1.1 버전은 분산원장, 스마트 계약, 합의, 기밀성, 탄력성, 확장성 등에 초점이 맞춰져 있다. 하이퍼레저 패브릭은 다음과 같은 블록체인 네트워크 기능을 제공한다. 신원 관리, 개인 정보보호 및 기밀 유지, 효율적인 처리, 체인코드 기능, 모듈식 디자인. 하이퍼레저 패브릭을 활용한 프로젝트의 대표적인 예로 IBM과 세계 최대 컨테이너 운송 물류 회사인 머스크의 공급망 프로젝트가 있다. 개발 블록체인 솔루션을 이용하면 공급망을 전 세

	계의 수많은 화물 컨테이너를 추적 관리할 수 있다. 라즈베리 파이는 영국의 라즈베리 파이 재단이 학교와 개발 도상국에서 기초 컴퓨터 과학의 교육을 증진시키기 위해 개발한 신용카드 크기의 싱글 보드 컴퓨터이다. 라즈베리 파이는 그래픽 성능이 뛰어나면서도 가격이 저렴하고 크기가 작다는 것이 특징이다. 컴퓨터와 마찬가지로 RAM, CPU, USB port, HDMI, LAN port가 존재하며 리눅스 기반의 OS를 사용한다. 따라서 IoT 센서 데이터를 블록체인 네트워크에 전달하기 위한 매개체로써 엣지 컴퓨팅이 가능한 시스템이다. 본 연구에서는 IoT 센서 데이터를 이용한 블록체인 기반 식품 유통망 모델을 제안하고 개발 분석했다. 기존에 연구 개발된 블록체인 기반 유통 시스템에서는 최초 1마일의 문제가 있다. 유통 과정에는 블록체인 기록이 제대로 남더라도 블록체인과 사람이 접촉하는 지점에서는 허점이 생길 수 밖에 없다는 문제이다. 다시 말해, 이미 블록체인에 저장된 데이터에 대해서는 데이터에 대한 무결성과 신뢰성을 보장하지만 데이터를 블록체인에 저장하는 과정에서 과연 그 데이터가 올바른 데이터인지 모른다는 것이다. 이러한 문제를 해결하기 위해 본 연구에서는 IoT 센서를 라즈베리 파이가 전처리하고 하이퍼레저 패브릭의 체인코드로 하드코딩하여 블록체인 시스템에 의해 인가되고 검증된 개체만 올바른 데이터를 저장할 수 있도록 하였다. 블록체인은 비가역적인 특성을 가지고 있다. 블록체인에 이미 저장된 데이터에 대해 수정이나 삭제가 불가능하고 오직 추가만 가능하다. 또한 이러한 데이터를 저장하는 원장은 블록체인 참여자가 모두 동일하게 가지고 있다. 따라서 용량이 큰 데이터가 지속적으로 블록체인에 저장될 경우 블록체인 시스템 자체에 과부하가 걸리게 된다. 이러한 문제를 해결하기 위해 본 연구에서는 최소한의 데이터만 블록체인에 저장하며 용량이 큰 원본 데이터는 암호화해서 외부 스토리지에 저장하는 방식을 취했다. 외부 스토리지에 데이터를 암호화하여 저장하는 작업은 시스템 참여자가 직접 생성한 대칭키를 가지고 수행되기 때문에 블록체인 참여자가 아닌 소비자는 블록체인에 저장된 데이터를 볼 수 없다. 하이퍼레저의 채널을 통해 시스템 참여자들 간에는 데이터를 투명하게 공개하고 MSP를 이용해 블록체인 참여를 제한한다. IoT 센서, 라즈베리 파이, 하이퍼레저 패브릭, 외부스토리지와 블록체인의 분리, 암호화 저장을 통해 식품 유통망을 구현한다면 데이터를 더욱 안전하게 저장하며 저장한 데이터에 대해서도 신뢰성을 보장할 수 있으며 유통 과정에서 생기는 문제를 추적하는 것도 불과 몇 초안에 이뤄질 수 있다. 본 연구에서 제안하는 시스템 모델을 개발하기 위해 IoT 센서로 온도 센서, 습도 센서, 인체 감지 센서, 충격 센서, 와이파이 및 GPS 등을 이용하고 라즈베리 파이는 전처리 작업이 수월하게 이뤄질 수 있도록 모델 3B를 채택했다. CPU로 Quad Cortex-A53@1.2GHz를 지원하고 1GB의 RAM, 10/100의 Ethernet을 지원한다. 하이퍼레저 패브릭 네트워크를 구현하기 위해 VMware Workstation 12 pro와 Ubuntu 16.04 버전을 이용했다. 운영체제 환경은 4GB 이상의 메모리, 50GB 이상의 HDD를 이용했다. 도커를 설치하기 위한 cURL은 7.58.0 버전, 컨테이너 기반의 오픈 소스 가상화 플랫폼인 도커는 19.03.7의 버전, 여러 개의 도커 컨테이너를 정의하고 실행

하는 개발자 편의 도구인 도커 컴포저는 1.25.4 버전, Go 언어는 1.11.1 버전, 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자 간에 파일의 작업을 조율하기 위한 분산 버전관리 시스템인 Git은 2.17.1 버전, 파이썬은 2.7.17 버전, 확장성 있는 네트워크 애플리케이션 개발에 사용되는 소프트웨어 플랫폼인 Node.js와 npm은 각각 8.17.0, 5.6.0 버전을 사용했다. 사용자와 시스템 참여자에게 보인는 프론트엔드 개발을 위해 VScode 1.44 버전, 센서 데이터 저장 서버로 ThingSpeak 클라우드 서버와 웹페이지 서버로 netify를 사용하고 개발 언어로는 HTML 5.0, CSS 2.1, Javascript 1.5 버전을 사용했다.

다음은 실제 하이퍼레저 패브릭 네트워크 구축 과정이다.

```
bbainq@ubuntu:~/go/src$ curl -sSL http://bit.ly/2ysbOFE | bash -s -- 1.
```

하이퍼레저 패브릭의 특정 버전 (1.4.3) 버전을 다운로드하고, 바이너리 툴, 하이퍼레저 패브릭의 도커 이미지까지 한번에 src 디렉터리에 다운로드 한다. 그런 다음 하이퍼레저 패브릭 네트워크를 구성해 실제로 동작하는지 확인한다.

```
bbainq@ubuntu:~/go/src$ cd $GOPATH/src/fabric-samples/first-network
bbainq@ubuntu:~/go/src/fabric-samples/first-network$ ls
base              ccp-template.json  configtx.yaml         crypto-confi
byfn.sh           ccp-template.yaml  connection-org3.json  docker-compo
ccp-generate.sh   channel-artifacts  connection-org3.yaml  docker-compo
bbainq@ubuntu:~/go/src/fabric-samples/first-network$ ./byfn.sh
```

제네시스 블록과 인증서를 생성한다. 그 과정은 다음과 같다.

1. cryptogen 도구를 사용해 인증서 생성
2. configtxgen 도구를 사용해 오더링 서비스 노드의 제네시스 블록 생성
3. configtxgen 도구를 사용해 채널을 생성
4. configtxgen 도구를 사용해 조직의 MSP에 대한 앵커 피어 노드 트랜잭션 파일 생성
5. configtxgen 도구를 사용해 다른 조직의 MSP에 대한 앵커 피어 노드 트랜잭션 파일 생성

그리고 이렇게 구성한 네트워크가 실제로 제대로 동작하는지 up 명령어를 통해 확인한다.

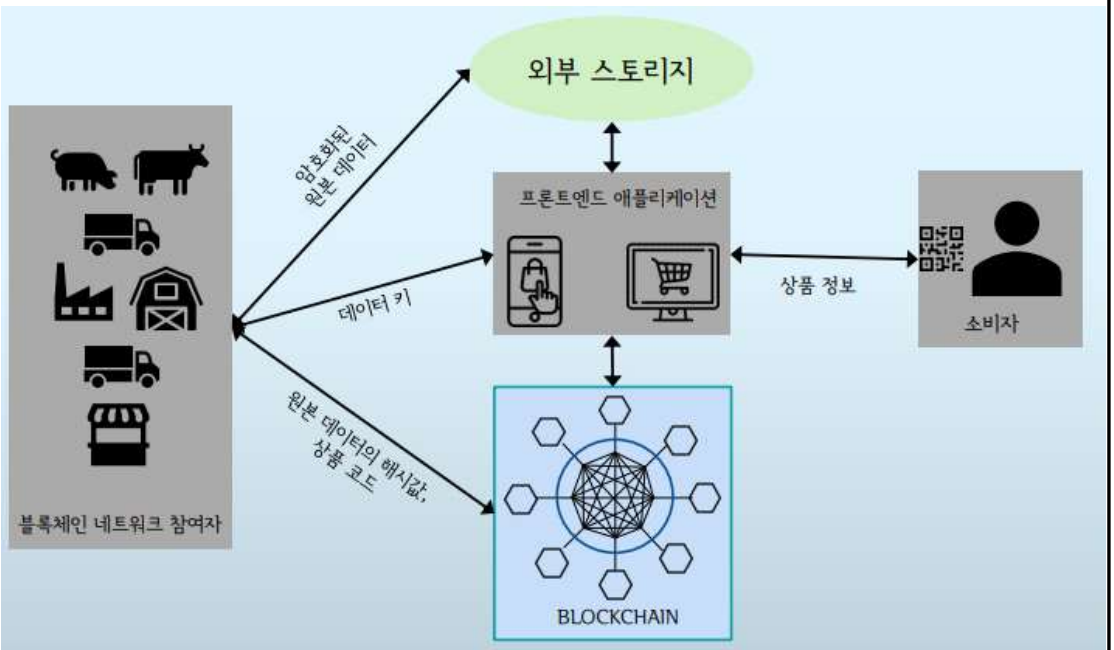
START부터 END까지 로그가 정상적으로 출력됨을 확인하면 네트워크가 정상적으로 작동된 것이다.

3. 과제 수행을 위한 연구팀의 역할 수행				
	구 분	**성 명**	**역 할**	**참여도(%)**
	지도교수	신상욱	연구과제에 대한 가이드라인 제시, 개발 시스템에 대한 기술 타당성 제시	10
	산업체멘토	김동규	개발 시스템에 대한 문제점 및 개선사항 제시	10
	팀장	정병규	연구과제 기획 및 총괄 관리	20
	팀원	김왕록	블록체인 기술 연구 및 유통망 시스템 적용 방안 연구	15
	팀원	전미현		15
	팀원	심재익	식품 유통망 시스템과 IoT 센서 데이터 연구	15
	팀원	오동의		15

지도 교수는 이론적인 연구 수행에 있어서 지도 교육을 했다. 공부할 방향과 연구 개발 방향에 대해 가이드라인을 제시하고 지켜야할 수칙을 당부했다. 1월에는 블록체인과 IoT에 대해 연구실 자체 세미나를 진행했다. 학생들이 발표한 것에 대해 피드백을 했고 앞으로 나아갈 방안을 제시했다. 퍼블릭 블록체인보다는 프라이빗 및 컨소시엄 블록체인의 적합성을 교육했고 IoT 센서를 제어하는 라즈베리 파이를 공부할 것과 하이퍼레저 패브릭 블록체인 플랫폼을 공부할 것을 당부했다. 2월에는 블록체인 기반 유통망 시스템 모델을 구현하기 위해 필요한 사전 기술에 대해 교육했다. 유통망 시스템에 적용하기 위한 블록체인의 형태와 합의 매커니즘, 그리고 블록 채굴 및 보상 문제를 해결하기 위한 방안을 모색하라고 당부했다. 3월에는 라즈베리 파이와 센서의 결합, 하이퍼레저 패브릭 네트워크 구축에 대해 진행 상황을 보고 받고 앞으로 나아갈 방향에 대해 가이드라인을 제시했다. 라즈베리 파이 구현에 있어서 연결해야 할 IoT 센서의 종류에 대해 교육하고 하이퍼레저 패브릭 네트워크 구축에 있어서 설치할 기본적인 프로그램에 대해 교육했다. 4월에는 연구 주제에 맞게 구체적인 블록체인 네트워크 프레임워크를 구상하고 그에 맞춰 하이퍼레저 패브릭을 구현할 수 있도록 지도했다.

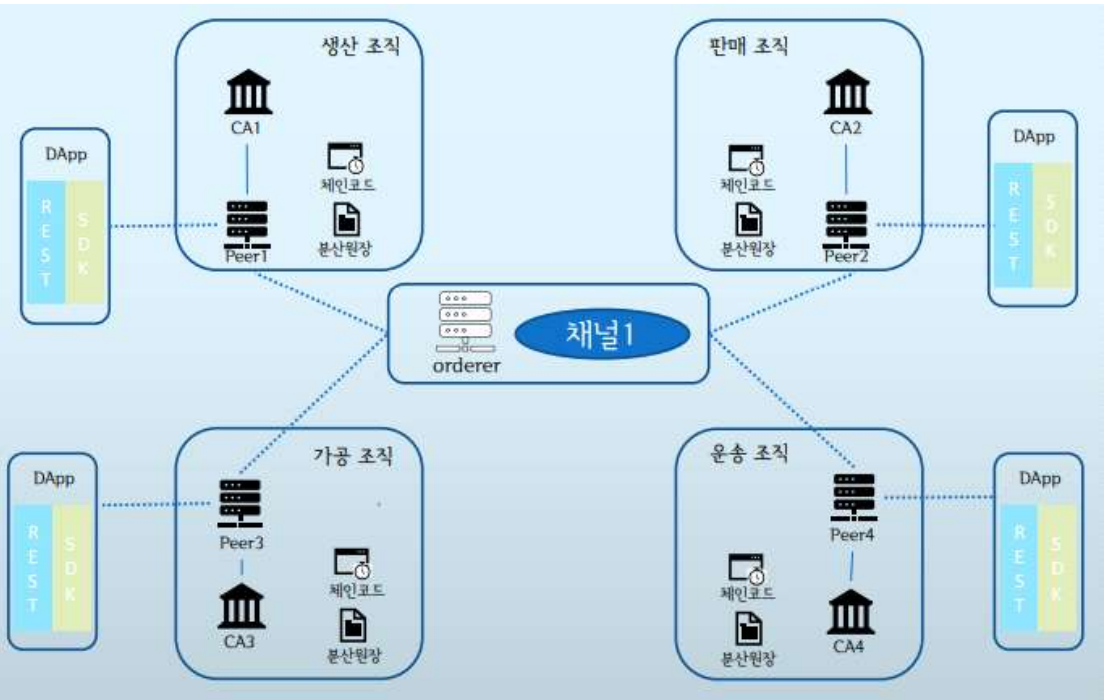
산업체 멘토는 실제 산업 현장에서 경험한 기술을 바탕으로 연구 수행의 기술적인 부분을 지도했다. 1월에는 블록체인과 IoT 연구에 대한 가이드라인을 제시했다. 특히 하이퍼레저의 특징과 설치 방법, 설치 도중의 오류 문제를 해결하는데 방안을 제시했다. 2월에는 유통망 시스템에 블록체인을 적용하기 위한 요구사항을 제시했다. 하이퍼레저 패브릭에서의 peer의 역할 부여 및 검증 정책에 대한 요구사항을 제시하고 전체적인 개발에 앞선 가이드라인을 제시했다. 3월에는 하이퍼레저 패브릭의 버전별 오류와 해결 방법을 지도하고 하이퍼레저 패브릭 네트워크의 채널과 조직에 대해 지도함으로써 하이퍼레저 패브릭 네트워크 구축에 대한 가이드라인을 제시했다. 라즈베리 파이에서 인체 감지 센서와 관련된 오류를 설명하고 이후 개발 계획을 논의했다. 4월에는 하이퍼레저 패브릭 네트워크 구현에 있어서 2개의 채널이 현실적으로는 불가능하다는 지적과 함께 채널 1개로 구현할 수 있는 방법을 제안했다. 더 나아가 프론트엔드 애플리케이션 구현과 관련된 피드백을 제시했다.

팀장은 전체 연구 과제를 기획하고 총괄 관리했다. 학부생들의 논문을 지도하고 연구 수행에 필요한 사전 지식을 교육했다. 블록체인과 하이퍼레저 패브릭, 식품 유통망, 이더리움과 같은 연구에 중심이 되는 주제를 가지고 자체 세미나를 진행했다. IoT 센서 데이터

	처리 방법, 블록체인 네트워크 구축에 대한 가이드라인을 제시하고 역할 분담했다. 매주 진행된 사항들에 대해 연구노트를 작성하고 중간보고서 및 최종보고서 등 각종 서류를 처리했다. 개발 도중에 발생한 오류나 문제점들에 대해 해결방안을 제시하고 연구 일정에 차질이 없도록 개발을 도왔다. 팀원들은 연구 개발에 필요한 선행 연구를 실시했다. 표준 문서, 논문을 활용해 자료를 수집하고 분석했다. 기존 유통망 시스템의 문제를 파악하고 개선 사항을 분석했으며 커넥티드카와 5G 기술, 그리고 비트코인과 이더리움을 통한 블록체인 기초 기술을 연구했다. 이러한 블록체인을 가지고 유통망 시스템에 적용 방안을 모색했으며 적합한 블록체인의 형태와 합의 매커니즘, 블록의 채굴과 보상 문제를 연구했다. 유통 각 단계에서 생성되는 IoT 센서 데이터의 종류를 파악하고 이러한 센서 데이터를 전처리할 수 있는 방법과 블록체인 네트워크에 송신할 수 있는 방법을 모색했다. 그리고 블록체인 기반 기술을 개발했다. 유통망 시스템 참여자를 위한 인증 시스템을 연구했고 유통 트랜잭션에 대한 프라이버시 보호 기법을 연구했다. 하이퍼레저 패브릭을 이용해 블록체인 네트워크를 구현했으며 라즈베리 파이로 IoT 센서 데이터를 전처리하고 블록체인에 저장했다. 팀장과 팀원은 LACUC라는 한 연구실에 소속되어 있기 때문에 세미나 및 회의하기가 용이했고 평소 수평적이고 화목한 연구실 분위기 덕에 큰 트러블 없이 좋은 의견을 수렴할 수 있었다. 팀장은 대학원생이여야하고 3, 4학년으로 이루어진 팀원이어야 한다는 지침은 연구 개발에 있어서 수월한 진행을 달성할 수 있었다. 실제 산업에 종사하는 멘토의 피드백을 통해 단순 이론 연구가 아닌 좀 더 실질적인 연구 개발이 될 수 있었다. 매주 쓰는 연구노트와 매달 쓰는 월별 보고서로 인해 전체 연구 프레임워크를 단단히 잡을 수 있었고 중간 발표와 최종 발표는 지쳐가는 연구 개발에 활력을 심어준 동기가 되었다.
4. 연구결과물에 대한 설명	다음은 본 연구에서 제안한 IoT 센서를 이용한 블록체인 기반 식품 유통망 모델을 나타낸다.  블록체인 네트워크 참여자는 유통과정에서의 각 단계를 표현한 생산지, 가공지, 운송업체, 판매지이다. 생산지, 가공지, 판매지, 운송 차량에는 각종 IoT 센서가 설치되어 있어 온도 및 습도, 위치, 날짜 및 시간 데이터가 자동으로 생성된다. 기존

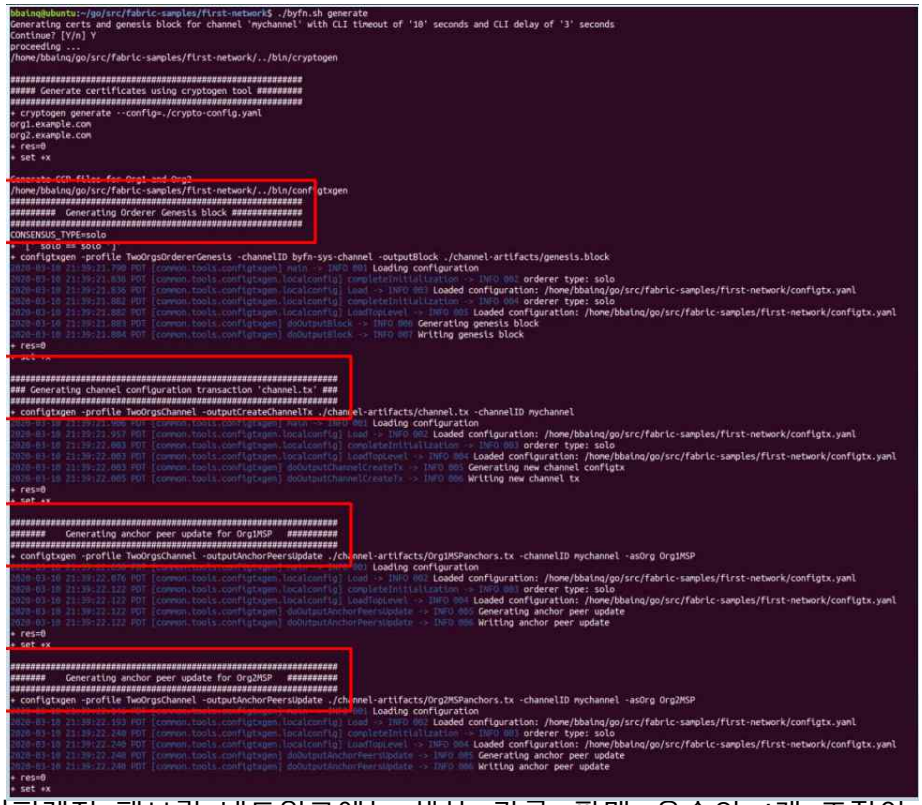
유통망 블록체인은 최초 1마일의 문제가 존재했다. 블록체인에 이미 저장된 데이터에 대해서는 무결성과 안전성을 제공하지만 과연 블록체인에 저장할 데이터가 올바른 데이터인가 하는 문제이다. 본 연구에서는 IoT 센서와 라즈베리 파이를 이용해 최초 1마일의 문제를 해결한다. 센서 데이터들을 라즈베리 파이가 수집하고 블록체인 네트워크로 전송하기 전 전처리 작업을 수행한다. 이는 블록체인의 수많은 데이터 처리로 인해 발생하는 과부화를 방지하기 위함이다. 라즈베리 파이의 전처리 작업에는 사전 정의해놓았던 기준을 토대로 상품의 등급을 정하는 작업, 날 것의 데이터를 트랜잭션 포맷에 맞게 정리하는 작업이 포함된다. 해당 모델은 허가형 프라이빗 블록체인이기 때문에 사전에 인증되고 허가받은 참여자만 블록체인 데이터를 공유할 수 있다. 상품을 구매하는 소비자는 블록체인 네트워크 참여자가 아니다. 소비자는 식품을 구매하기 전 상품에 표시된 QR코드를 통해 유통과정의 신뢰도를 판단할 수 있다. 소비자는 블록체인 네트워크의 참여자가 아니기 때문에 블록체인에 저장된 모든 데이터를 볼 수 없다. 이 QR코드는 판매지에서 자동으로 생성되는데, 판매지 peer의 ID를 가진 개체가 트랜잭션을 생성하면 블록체인 시스템은 유통의 과정이 끝났다고 판단하고 블록체인에 입력된 상품코드를 역추적해 소비자를 위한 데이터를 생성한다. 이때 생성되는 데이터는 사용자의 프라이버시 보호를 위해 문제 발생 시에만 필요한 관리자 정보, 운전자 정보와 같은 개인정보를 제외한다. 그래서 QR코드를 통해 보여지는 데이터는 제품명, 제조일, 유통기한, 출고처, 출고일과 같이 신뢰도를 판단할 수 있는 최소한의 정보만 표시된다.

블록체인은 비가역적인 특징을 가지고 있다. 다시 말해, 블록체인에 저장된 데이터에 대해 수정 및 삭제할 수 없고, 오직 추가만 가능하다. 따라서 모든 데이터를 블록체인에 저장하는 것은 비효율적이다. 본 연구에서는 데이터의 저장을 블록체인과 외부 스토리지로 분리한다. 여기서 외부 스토리지는 IPFS(InterPlanetary File



System)나 클라우드 스토리지가 될 수 있다. 외부 스토리지에는 IoT 센서로부터 생성된 모든 데이터를 암호화해서 저장한다. 암호화에 사용되는 키는 하이퍼레저 패브릭에 의해 블록체인 참여자인지 확인하고 프론트엔드 플랫폼을 통해 키가 분배된다. 여기서 키는 공개키로써 데이터 암호화 뿐만 아니라 생성한 데이터의 소유권을 주장할 수 있는 디지털 서명의 기능을 한다. 블록체인에는 외부 스토리지에 저장된 원본 데이터의 위치와 무결성 검증을 위한 해시값, 추적을 위한 상품 코드가 저장된다. 이렇게 데이터를 분리 저장함으로써 데이터에 대한 프라이버시 보호와 안전성을 기여할 수 있다.

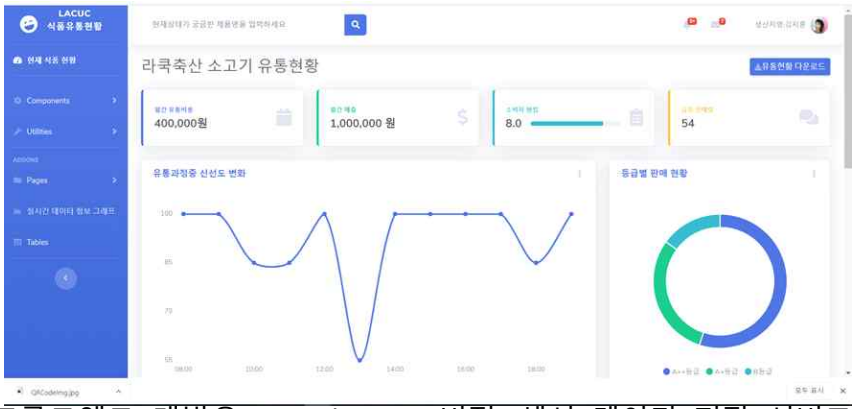
하이퍼레저 패브릭 네트워크 구현을 설명함에 있어서 네트워크 자체가 추상적이고 코드 위주라 결과 화면 대신 구상도로 표현한다.



하이퍼레저 패브릭 네트워크에는 생산, 가공, 판매, 운송의 4개 조직이 있으며 구현의 편의를 위해 각 조직에는 한 개의 peer가 있다고 가정했다. 물론 실제 유통망 블록체인 시스템에는 한 개의 조직에 적게는 수십개, 많게는 수천개가 있을 수 있다. 각 피어는 채널1을 통해 데이터를 공유하고 있으며 한 개의 채널 당 한 개의 분산원장이 생성되기 때문에 각 피어들은 동일한 분산원장을 보유하고 있다. 여기서 말하는 분산원장은 위 문단에서 언급한 블록체인과 동일한 역할을 한다. 각 피어들은 DApp을 통해 트랜잭션을 생성하고 Orderer 노드에게 전달한다. Orderer 노드는 말 그대로 받은 트랜잭션을 시간 순으로 정렬해주는 역할을 한다. Orderer 노드가 받은 트랜잭션을 정렬하고 블록을 생성하면 다시 각 조직의 peer에게 블록을 전달한다. 각 조직의 peer는 받은 블록을 검증하여 각자 보유한 분산원장에 블록을 연결한다. 이러한 블록들은 이전 블록의 해시값을 통해 연결되어 있기 때문에 한 개의 블록을 해킹하기 위해서는 모든 블록을 해킹해야 하며

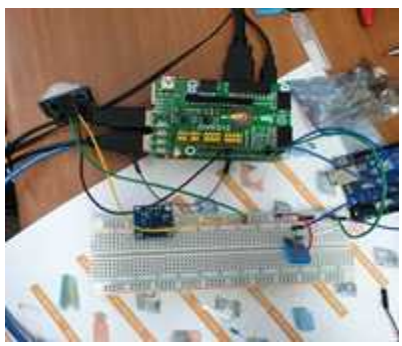
한 peer의 분산원장만 해킹하는 것이 아닌 참여하고 있는 모든 peer의 분산원장을 해킹해야 하기 때문에 굉장히 안전하다고 할 수 있다. 아래 그림은 실제 리눅스 상에서 하이퍼레저 패브릭 네트워크를 구현한 사진이다.

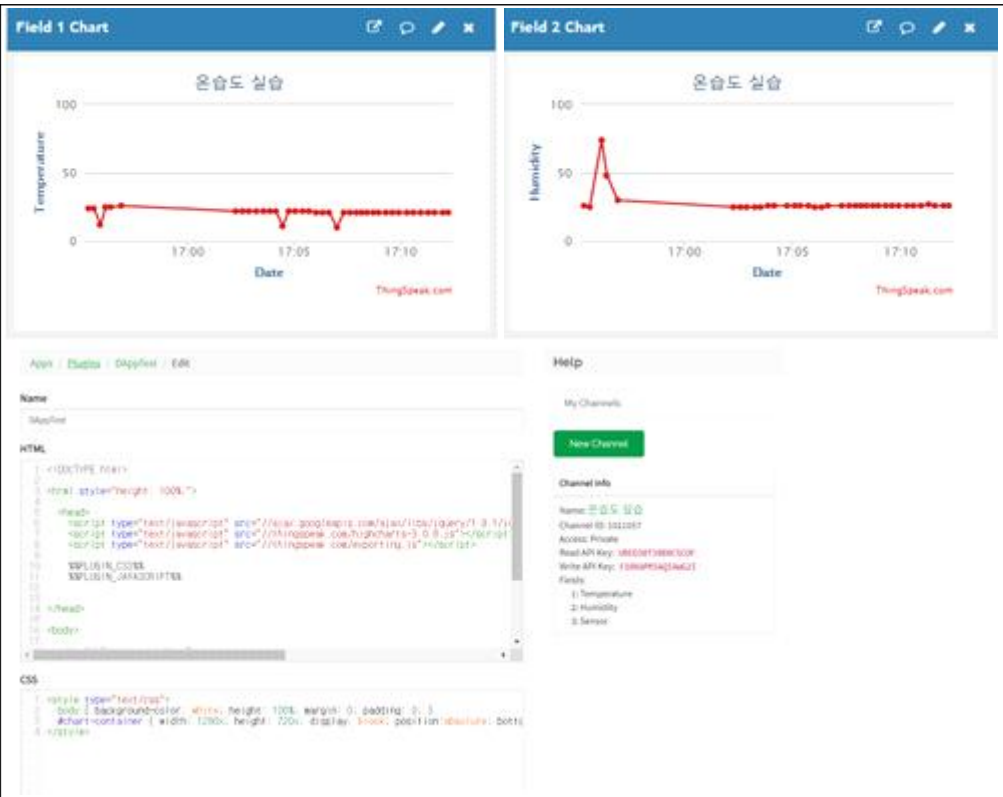
사용자가 QR코드를 통해 확인할 수 있는 데이터는 웹 애플리케이션으로 구현했다.



프론트엔드 개발은 VScode 1.44 버전, 센서 데이터 저장 서버로 ThingSpeak 클라우드 서버와 웹페이지 서버로 netify를 사용하고 개발 언어로는 HTML 5.0, CSS 2.1, Javascript 1.5 버전을 사용했다. 판매지의 각 상품마다 붙어있는 QR코드를 인식하면 위와 같은 사진처럼 웹 애플리케이션이 실행된다. 이를 통해 소비자 평점, 유통과정 중 신선도의 변화를 수치와 그래프로 확인할 수 있다.

IoT 센서와 라즈베리 파이의 연동은 아래 사진과 같다.



	 온도 및 습도 센서, 인체 감지 센서, wifi 모듈, GPS 등과 같은 각종 센서들을 라즈베리 파이에 연결하고 수집된 센서 데이터를 ThingSpeak라는 클라우드 서버로 전송한다. 여기서 이 클라우드 서버는 위에서 언급한 외부스토리지이며 아래 코드는 블록체인과 외부 스토리지로 데이터를 전송하는 역할을 한다. 운송 차량에서는 마찬가지로 IoT 센서 데이터를 라즈베리 파이가 수집하고 라즈베리 파이와 연결된 차량용 게이트웨이를 통해 5G와 같은 무선 네트워크를 이용해 블록체인과 외부 스토리지로 데이터가 송신된다.
5. 연구결과의 기대효과 및 활용방안	본 연구에서는 IoT를 이용한 블록체인 기반 식품 유통망 모델을 제안하고 적용 방안을 제시했다. 블록체인의 무결성, 안전성, 투명성, 추적성이라는 특징은 효율적이고 안전한 식품 유통을 가능하게 하고 QR코드를 통한 네트워크 참여자가 아닌 소비자의 블록체인 데이터를 확인할 수 있게 했다. 따라서 소비자들도 안심하고 제품을 구매할 수 있다. 또한 유통시 발생하는 제품 손상이나 사고에 대한 책임 추적도 가능하다. IoT 센싱 데이터를 이용한 자동화된 트랜잭션 처리로 인한 무결성 및 신뢰성을 달성할 수 있다. 기존 유통망 블록체인에 존재했던 최초 1마일의 문제를 해결하고 데이터에 대한 신뢰성을 달성했다. 블록체인 플랫폼으로 허가형 프라이빗 블록체인인 하이퍼레저 패브릭을 채택함으로써 사용자 프라이버시 보호 및 기밀성을 달성하고 저장 공간을 블록체인 및 외부 스토리지로 분리함으로써 데이터 프라이버시 보호 및 저장의 효율성을 극대화했다. 더 나아가 돼지고기로 한정했던 본 연구의 시나리오를 넘어 다른 식품에도 충분히 적용 가능하며 식품 유통이 아닌 다른 상품에도 적용 가능할 것으로 보인다. 특히 블록체인 특구로 지정된 부산광역시 지역 산업 수용에 부응할 수 있을 것으로 생각된다.

2. 연구결과물에 대한 산업체 멘토 의견

1. 연구과제 진행을 통한 현장 문제 기술 해결 여부	①해결 ❷부분해결 ③미해결
2. 연구결과물에 대한 지도교수 의견	정보보안은 IT 분야의 모든 방면을 알아야 하는 어려운 학문이다. 특히, 정보보안 분야 중 블록체인은 학부생 커리큘럼에는 없는 새롭고 어려운 기술이다. 짧은 시간안에 어려운 블록체인을 공부하고 개발했기 때문에 완성된 연구 결과물에 대해서는 미흡한 부분이 있지만 블록체인 네트워크를 구축했다는 것만으로도 엄청난 성과라고 볼 수 있다. 현존하는 유통망 블록체인의 문제점을 파악하고 연구에 차별화를 둔 점, 아직 견고한 블록체인 개발 플랫폼이 없기 때문에 발생하는 수많은 오류들을 스스로 해결하려고 했다는 점을 높이 평가한다. 지도교수 신 상 욱 (서명)
3. 연구결과물에 대한 산업체멘토 의견	물론 개발 시스템을 당장 현 산업에 적용하기에는 미흡한 점들이 많다. 하지만 제안하는 모델과 적용 방안은 충분히 잠재 가치가 있다고 판단된다. 짧은 시간 안에 어려운 블록체인을 연구하고 개발했다는 점을 높이 평가한다. 산업체 멘토 소속 ㈜스마트엠투엠 김 동 규 (인)

3. 연구 결과물 관련 증빙자료

시작품, 시제품, 결과모형, 도면, S/W, 설계도면, 조사 및 분석 결과, 보고서, 이미지, 영상, 학회지 논문, 학술대회 발표, 특허 출원 등

본 연구 개발은 컴퓨터 프로그래밍이 주를 이룬다. 따라서 현실 세계에서 보이는 실체가 없으며 연구 결과물 관련 증빙 자료는 주요 개발 코드로 대체한다.

1. 하이퍼레저 패브릭 네트워크에서 가공지 등록 및 조회, 키 생성

```
package main
import (
        "bytes"
        "encoding/json"
        "fmt"
        "strconv"
        "github.com/hyperledger/fabric/core/chaincode/shim"
        "github.com/hyperledger/fabric/protos/peer"
)type procArea struct {
        BusinessName string `json:"businessname"`
        Manager      string `json:"manager"`
        Phone        string `json:"phone"`
        Address      string `json:"address"`
        Serial       string `json:"serial"`
```

```
}type procAreaSerial struct {
	Key string
	Idx int
}type SmartContract struct {
}func (t *SmartContract) Init(APIstub shim.ChaincodeStubInterface) peer.Response {
	return shim.Success(nil)
}func (t *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) peer.Response {
	fn, args := APIstub.GetFunctionAndParameters()
	if fn == "setProcArea" {
		return t.setProcArea(APIstub, args)
	} else if fn == "getProcArea" {
		return t.getProcArea(APIstub, args)
	}
	fmt.Println("Check your function : " + fn)
	return shim.Error("Unknown function")
}func (t *SmartContract) setProcArea(APIstub shim.ChaincodeStubInterface, args []string)
peer.Response {
	var procSerial = procAreaSerial{}
	json.Unmarshal(generateProcAreaSerial(APIstub), &procSerial)
	serialIdx := strconv.Itoa(procSerial.Idx)
	fmt.Println("Key : " + procSerial.Key + ", Idx : " + serialIdx)
	var buffer bytes.Buffer
	var proc = procArea{BusinessName: args[0], Manager: args[1]}
	var serialString = procSerial.Key + serialIdx
	proc.Serial = serialString
	procAsJSONBytes, _ := json.Marshal(proc)
	buffer.WriteString("serialString: ")
	buffer.WriteString(serialString)
	fmt.Println("serialString is " + serialString)
	err := APIstub.PutState(serialString, procAsJSONBytes)
	if err != nil {
		return shim.Error(fmt.Sprintf("Failed to record process area catch: %s", procSerial))
	}
	procSerialAsByte, _ := json.Marshal(procSerial)
	APIstub.PutState("latestProcSerial", procSerialAsByte)
	buffer.WriteString(" proc.Serial: ")
	buffer.WriteString(proc.Serial)
	return shim.Success(buffer.Bytes())
}func (t *SmartContract) getProcArea(stub shim.ChaincodeStubInterface, args []string)
```

```
peer.Response {
	procAsBytes, err := stub.GetState(args[0])
	if err != nil {
		fmt.Println(err.Error())
	}
	proc := procArea{}
	json.Unmarshal(procAsBytes, &proc)
	var buffer bytes.Buffer
	buffer.WriteString("[")
	bArrayMemberAlreadyWritten := false
	if bArrayMemberAlreadyWritten == true {
		buffer.WriteString(",")
	}
	buffer.WriteString("{₩"Business Name₩":")
	buffer.WriteString("₩"")
	buffer.WriteString(proc.BusinessName)
	buffer.WriteString("₩"")
	buffer.WriteString(", ₩"Serial₩":")
	buffer.WriteString("₩"")
	buffer.WriteString(proc.Serial)
	buffer.WriteString("₩"")
	buffer.WriteString("}")
	bArrayMemberAlreadyWritten = true
	buffer.WriteString("]")
	return shim.Success(buffer.Bytes())
}func generateProcAreaSerial(stub shim.ChaincodeStubInterface) []byte {
	var isFirst bool = false
	procSerialAsByte, err := stub.GetState("latestProcSerial")
	if err != nil {
		fmt.Println(err.Error())
	}
	procSerial := procAreaSerial{}
	json.Unmarshal(procSerialAsByte, &procSerial)
	var tempIdx string
	tempIdx = strconv.Itoa(procSerial.Idx)
	fmt.Println(procSerial)
	fmt.Println("Key is" + strconv.Itoa(len(procSerial.Key)))
	if len(procSerial.Key) == 0 || procSerial.Key == "" {
		isFirst = true
```

```
                procSerial.Key = "PROC"
        }
        if !isFirst {
                procSerial.Idx = procSerial.Idx + 1
        }
        fmt.Println("Last serial is " + procSerial.Key + " : " + tempIdx)
        returnValueByte, _ := json.Marshal(procSerial)
        return returnValueByte
}func main() {
        err := shim.Start(new(SmartContract))
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
        }
}
```

2. 하이퍼레저 패브릭 네트워크에서 생산지 등록 및 조회, 키 생성

```
package main
import (
        "bytes"
        "encoding/json"
        "fmt"
        "strconv"
        "github.com/hyperledger/fabric/core/chaincode/shim"
        "github.com/hyperledger/fabric/protos/peer"
)type prodArea struct {
        BusinessName string `json:"businessname"`
        Country      string `json:"country"`
        Manager      string `json:"manager"`
        Phone        string `json:"phone"`
        Address      string `json:"address"`
        Serial       string `json:"serial"`
}type prodAreaSerial struct {
        Key string
        Idx int
}type SmartContract struct {
}func (t *SmartContract) Init(APIstub shim.ChaincodeStubInterface) peer.Response {
        return shim.Success(nil)
}func (t *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) peer.Response {
        fn, args := APIstub.GetFunctionAndParameters()
```

```
	if fn == "setProdArea" {
		return t.setProdArea(APIstub, args)
	} else if fn == "getProdArea" {
		return t.getProdArea(APIstub, args)
	}
	fmt.Println("Check your function : " + fn)
	return shim.Error("Unknown function")
}func (t *SmartContract) setProdArea(APIstub shim.ChaincodeStubInterface, args []string)
peer.Response {
	if len(args) != 5 {
		return shim.Error("Incorrect number of arguments. Expecting 5")
	}
	var prodSerial = prodAreaSerial{}
	json.Unmarshal(generateProdAreaSerial(APIstub), &prodSerial)
	serialIdx := strconv.Itoa(prodSerial.Idx)
	fmt.Println("Key : " + prodSerial.Key + ", Idx : " + serialIdx)
	var buffer bytes.Buffer
	var prod = prodArea{BusinessName: args[0], Country: args[1], Manager: args[2], Phone:
args[3], Address: args[4]}
	var serialString = prodSerial.Key + serialIdx
	prod.Serial = serialString
	prodAsJSONBytes, _ := json.Marshal(prod)
	buffer.WriteString("serialString: ")
	buffer.WriteString(serialString)
	fmt.Println("serialString is " + serialString)
	err := APIstub.PutState(serialString, prodAsJSONBytes)
	if err != nil {
		return shim.Error(fmt.Sprintf("Failed to record production area catch: %s",
prodSerial))
	}
	prodSerialAsByte, _ := json.Marshal(prodSerial)
	APIstub.PutState("latestProdSerial", prodSerialAsByte)
	buffer.WriteString(" prod.Serial: ")
	buffer.WriteString(prod.Serial)
	return shim.Success(buffer.Bytes())
}func (t *SmartContract) getProdArea(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
	prodAsBytes, err := stub.GetState(args[0])
	if err != nil {
```

```
                fmt.Println(err.Error())
        }
        prod := prodArea{}
        json.Unmarshal(prodAsBytes, &prod)
        var buffer bytes.Buffer
        buffer.WriteString("[")
        bArrayMemberAlreadyWritten := false
        if bArrayMemberAlreadyWritten == true {
                buffer.WriteString(",")
        }
        buffer.WriteString("{₩"Business Name₩":")
        buffer.WriteString("₩"")
        buffer.WriteString(prod.BusinessName)
        buffer.WriteString("₩"")
        buffer.WriteString(", ₩"Serial₩":")
        buffer.WriteString("₩"")
        buffer.WriteString(prod.Serial)
        buffer.WriteString("₩"")
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
        buffer.WriteString("]")
        return shim.Success(buffer.Bytes())
}func generateProdAreaSerial(stub shim.ChaincodeStubInterface) []byte {
        var isFirst bool = false
        prodSerialAsByte, err := stub.GetState("latestProdSerial")
        if err != nil {
                fmt.Println(err.Error())
        }
        prodSerial := prodAreaSerial{}
        json.Unmarshal(prodSerialAsByte, &prodSerial)
        var tempIdx string
        tempIdx = strconv.Itoa(prodSerial.Idx)
        fmt.Println(prodSerial)
        fmt.Println("Key is" + strconv.Itoa(len(prodSerial.Key)))
        if len(prodSerial.Key) == 0 || prodSerial.Key == "" {
                isFirst = true
                prodSerial.Key = "PROD"
        }
        if !isFirst {
```

```
                prodSerial.Idx = prodSerial.Idx + 1
        }
        fmt.Println("Last serial is " + prodSerial.Key + " : " + tempIdx)
        returnValueByte, _ := json.Marshal(prodSerial)
        return returnValueByte
}func main() {
        err := shim.Start(new(SmartContract))
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
        }
}
```

3. 하이퍼레저 패브릭 네트워크에서 식품 등급 등록 및 조회

```
package main
import (
        "bytes"
        "encoding/json"
        "fmt"
        "github.com/hyperledger/fabric/core/chaincode/shim"
        "github.com/hyperledger/fabric/protos/peer"
)type foodInfo struct {
        FoodName        string `json:"foodname"`
        Grade           string `json:"grade"`
        Origin          string `json:"origin"`
        Temperature     string `json:"temperature"`
        Humidity        string `json:"humidity"`
        ProdSerial      string `json:"prodserial"`
        FoodSerial      string `json:"foodserial"`
        ProcSerial      string `json:"procserial"`
        TransportSerial string `json:"transportserial"`
        SaleSerial      string `json:"saleserial"`
}type procArea struct {
        BusinessName string `json:"businessname"`
        Manager      string `json:"manager"`
        Phone        string `json:"phone"`
        Address      string `json:"address"`
        Serial       string `json:"serial"`
}type procAreaSerial struct {
        Key string
```

```
        Idx int
}type SmartContract struct {
}func (t *SmartContract) Init(APIstub shim.ChaincodeStubInterface) peer.Response {
        return shim.Success(nil)
}func (t *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) peer.Response {
        fn, args := APIstub.GetFunctionAndParameters()
        if fn == "setFoodGrade" {
                return t.setFoodGrade(APIstub, args)
        } else if fn == "getFoodInfo" {
                return t.getFoodInfo(APIstub, args)
        }
        fmt.Println("Check your function : " + fn)
        return shim.Error("Unknown function")
}func (t *SmartContract) setFoodGrade(APIstub shim.ChaincodeStubInterface, args []string)
peer.Response {
        foodAsBytes, _ := APIstub.GetState(args[0])
        food := foodInfo{}
        json.Unmarshal(foodAsBytes, &food)
        food.Grade = args[1]
        food.ProcSerial = args[2]
        foodAsBytes, _ = json.Marshal(food)
        err := APIstub.PutState(args[0], foodAsBytes)
        if err != nil {
                return shim.Error(fmt.Sprintf("Failed to set food grade."))
        }
        var buffer bytes.Buffer
        buffer.WriteString("Food grade: ")
        buffer.WriteString(food.Grade)
        return shim.Success(buffer.Bytes())
}func (t *SmartContract) getFoodInfo(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
        foodAsBytes, err := stub.GetState(args[0])
        if err != nil {
                fmt.Println(err.Error())
        }
        food := foodInfo{}
        json.Unmarshal(foodAsBytes, &food)
        var buffer bytes.Buffer
        buffer.WriteString("[")
```

```
		bArrayMemberAlreadyWritten := false
		if bArrayMemberAlreadyWritten == true {
			buffer.WriteString(",")
		}
		buffer.WriteString("{₩"Food Name₩":")
		buffer.WriteString("₩"")
		buffer.WriteString(food.FoodName)
		buffer.WriteString("₩"")
		buffer.WriteString(", ₩"Serial₩":")
		buffer.WriteString("₩"")
		buffer.WriteString(food.Grade)
		buffer.WriteString("₩"")
		buffer.WriteString("}")
		bArrayMemberAlreadyWritten = true
		buffer.WriteString("]")
		return shim.Success(buffer.Bytes())
}func main() {
		err := shim.Start(new(SmartContract))
		if err != nil {
			fmt.Printf("Error starting Simple chaincode: %s", err)
		}
}
```

4. 하이퍼레저 패브릭 네트워크에서 식품 등록 및 조회, 키 생성

```
package main
import (
		"bytes"
		"encoding/json"
		"fmt"
		"strconv"
		"github.com/hyperledger/fabric/core/chaincode/shim"
		"github.com/hyperledger/fabric/protos/peer"
)type foodInfo struct {
		FoodName		string `json:"foodname"`
		Grade		string `json:"grade"`
		Origin		string `json:"origin"`
		Temperature		string `json:"temperature"`
		Humidity		string `json:"humidity"`
		ProdSerial		string `json:"prodserial"`
```

```
        FoodSerial      string `json:"foodserial"`
        ProcSerial      string `json:"procserial"`
        TransportSerial string `json:"transportserial"`
        SaleSerial      string `json:"saleserial"`
}type foodInfoSerial struct {
        Key string
        Idx int
}type SmartContract struct {
}func (t *SmartContract) Init(APIstub shim.ChaincodeStubInterface) peer.Response {
        return shim.Success(nil)
}func (t *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) peer.Response {
        fn, args := APIstub.GetFunctionAndParameters()
        if fn == "setFoodInfo" {
                return t.setFoodInfo(APIstub, args)
        } else if fn == "getFoodInfo" {
                return t.getFoodInfo(APIstub, args)
        }
        fmt.Println("Check your function : " + fn)
        return shim.Error("Unknown function")
}func (t *SmartContract) setFoodInfo(APIstub shim.ChaincodeStubInterface, args []string)
peer.Response {
        var foodSerial = foodInfoSerial{}
        json.Unmarshal(generateFoodInfoSerial(APIstub), &foodSerial)
        serialIdx := strconv.Itoa(foodSerial.Idx)
        fmt.Println("Key : " + foodSerial.Key + ", Idx : " + serialIdx)
        var buffer bytes.Buffer
        var food = foodInfo{FoodName: args[0], Origin: args[1]}
        var serialString = foodSerial.Key + serialIdx
        food.FoodSerial = serialString
        foodAsJSONBytes, _ := json.Marshal(food)
        buffer.WriteString("serialString: ")
        buffer.WriteString(serialString)
        fmt.Println("serialString is " + serialString)
        err := APIstub.PutState(serialString, foodAsJSONBytes)
        if err != nil {
                return shim.Error(fmt.Sprintf("Failed to record production area catch: %s",
foodSerial))
        }
        prodSerialAsByte, _ := json.Marshal(foodSerial)
```

```
        APIstub.PutState("latestFoodSerial", prodSerialAsByte)
        buffer.WriteString(" food.Serial: ")
        buffer.WriteString(food.FoodSerial)
        return shim.Success(buffer.Bytes())
}func (t *SmartContract) getFoodInfo(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
        foodAsBytes, err := stub.GetState(args[0])
        if err != nil {
                fmt.Println(err.Error())
        }
        food := foodInfo{}
        json.Unmarshal(foodAsBytes, &food)
        var buffer bytes.Buffer
        buffer.WriteString("[")
        bArrayMemberAlreadyWritten := false
        if bArrayMemberAlreadyWritten == true {
                buffer.WriteString(",")
        }
        buffer.WriteString("{₩"Food Name₩":")
        buffer.WriteString("₩"")
        buffer.WriteString(food.FoodName)
        buffer.WriteString("₩"")
        buffer.WriteString(", ₩"Serial₩":")
        buffer.WriteString("₩"")
        buffer.WriteString(food.FoodSerial)
        buffer.WriteString("₩"")
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
        buffer.WriteString("]")
        return shim.Success(buffer.Bytes())
}func generateFoodInfoSerial(stub shim.ChaincodeStubInterface) []byte {
        var isFirst bool = false
        foodSerialAsByte, err := stub.GetState("latestFoodSerial")
        if err != nil {
                fmt.Println(err.Error())
        }
        foodSerial := foodInfoSerial{}
        json.Unmarshal(foodSerialAsByte, &foodSerial)
        var tempIdx string
```

```
	tempIdx = strconv.Itoa(foodSerial.Idx)
	fmt.Println(foodSerial)
	fmt.Println("Key is" + strconv.Itoa(len(foodSerial.Key)))
	if len(foodSerial.Key) == 0 || foodSerial.Key == "" {
		isFirst = true
		foodSerial.Key = "FOOD"
	}
	if !isFirst {
		foodSerial.Idx = foodSerial.Idx + 1
	}
	fmt.Println("Last serial is " + foodSerial.Key + " : " + tempIdx)
	returnValueByte, _ := json.Marshal(foodSerial)
	return returnValueByte
}func main() {
	err := shim.Start(new(SmartContract))
	if err != nil {
		fmt.Printf("Error starting Simple chaincode: %s", err)
	}
}
```

5. 프론트엔드 웹 애플리케이션 HTML

* HTML 코드가 너무 긴 관계로 구글 드라이브 공유 url로 대체

https://drive.google.com/open?id=1c4TCB1U5O64IJr9qLieepv_yRjzeplGS

6. 라즈베리 파이의 IoT 센서 데이터 처리

```
import Adafruit_DHT
import RPi.GPIO as GPIO
import time
import datetime
import urllib

sensor =Adafruit_DHT.DHT11
pin = 4
startTime = time.time()
preH=0
preT=0
standard_Gap_of_T = 3
standard_Gap_of_H = 3
getRight = False
```

```
while True:

        wtime = datetime.datetime.now()
        h,temp = Adafruit_DHT.read_retry(sensor,pin)
#       gapT = temp-preT
#       gapH = h - preH

        if h is not None and temp is not None:
                getRight = True
                print (wtime,'Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(temp,h))
                time.sleep(30)
                html =
urllib.urlretrieve("https://api.thingspeak.com/update?api_key=F10XAPR5AQIAWG2I&field1="+str(tem
p)+"&field2="+str(h))

        elif preH+3<h or preH-3>h or preT+3<temp or preT-3>temp :
                html =
urllib.urlretrieve("https://api.thingspeak.com/update?api_key=F10XAPR5AQIAWG2I&field1="+str(tem
p)+"&field2="+str(h))

        else:
                getRight = False
                print('Data Input Error')

        if getRight is True:
                preH = h
                preT = temp
```

4. 실전문제연구팀 활동 사진

연구팀 연구 수행과 관련한 사진
1. 멘토와 하이퍼레저 패브릭 개발 실습 2. 멘토와 블록체인 기초 및 라즈베리 파이 이론 세미나

3. 교수님과 개발 방향에 관한 회의

4. 교수님, 멘토와 회의