DID 탈중앙화 신원인증 신분증

팀 명: 최종장박봉

지도교수: 이병천 교수님

팀 장: 이현종

팀 명: 박주형

장예진

최유진

이강봉

2022. 11.

중부대학교 정보보호학과

목차

1. 서론

1.1 연구 배경

현 주민등록증은 사용자의 개인정보가 그대로 노출되어 타인에게 쉽게 개인정보가 유출되기도 하며 각종 첨단 장비를 통한 위 · 변조를 통하여 이를 방지하기 어려워지면서 이를 방지하기 위해 탈중앙화 신원증명(Decentralized Identity, DID)기반 모바일 신분증을 기획하게 되었다.

1.2 연구 필요성

코로나19 확산으로 비대면 서비스 수요가 증가함에 따라 디지털상에서 물리적 신분증을 대체할 새로운 인증 수단의 필요성이 점차 대두했다. 기존의 온라인 금융 서비스의 경우 실명 확인이 필요하며, 신원 확인을 위해 별도의 인증을 거쳐야 하는 등의 한계점을 가졌다.

또한, 기존 신분증은 개인정보를 포함하고 있어 확인 과정에서 민감 정보가 노출되고, 분실 시 정보 유출 및 도용 가능성이 꾸준히 제기되기도 했다. 해외의 도입 사례의 경우 표면에는 사용자의 신분을 식별할 수 있는 최소한의 정보만을 표기하고 민감한 사용자 정보는 IC 칩에 저장하며, 전자신분증을 통해 사용자 인증 시 강력한 암호화 방식인 공인인증서를 인증의 수단으로 사용했음에도 불구하고 보관성 문제로 인한 피해가 발생하였다. 아울러, 각국에서 개인정보 공개 및 자기주권 보장 필요성이 대두해 국가, 신뢰기간 등의 중개자 없이 스스로가 자신을 인증하고 데이터를 관리할 수 있도록 하는 요구가 늘어났다. 이에 주민등록증, 운전면허증, 여권 등의 국가 신분증을 디지털화 한 상태로 모바일 기기에 저장한 모바일 신분증을 고안하였다.

1.3 연구 목적 및 주제 선정

이번 연구는 위 · 변조 및 도용 또한 현재 사용하고 있는 전자 신분증의 문제점들을 보안하기위해 블록체인을 활용하여 모바일 신분증을 구현하게 되었다. DID 기술은 개인정보를 암호화하고 위 · 변조를 불가능하게 할 뿐 아니라 중간자의 공격, 스니핑, 리더기의 변조, 복제 등에 개인정보 유출될 우려가 거의 없기 때문에 주제로 선정하게 되었다.

2. 관련 연구

2.1 Java

자바(Java)는 1995 년 썬 마이크로시스템즈에서 발표한 객체 지향 프로그래밍 언어다. 자바는 가능한 적은 종속성을 갖도록 설계되었으며 **"Programmers wirte once, run anywhere(WORA)"** 와 같이 한번 작성한 코드를 모든 플랫폼에서 작동시킬 수 있는 범용적인 언어로 전 세계의 많은 Back end 개발자가 선택하는 언어이며 전 세계적으로 보고된 개발자는 9 백만명이다. 또한 Android 앱 개발을 위한 유일한 공식 언어로 Amazon, Twitter, Netflix 등 많은 서비스에서 사용하고 있으며 게임 콘솔, 슈퍼컴퓨터 등 많은 곳에서 실행이 가능하다는 장점이 있다. 대한민국 전자정부표준 또한 Java 프레임워크인 Spring 을 사용한다.

2.2 React

리액트는 UI 자바스크립트 라이브러리로써 싱글 페이지 애플리케이션의 UI(User Interface)를 생성하는데 집중한 라이브러리이다. 리액트는 자바스크립트에 HTML을 포함하는 JSX(JaveScript XML)이라는 간단한 문법과 단방향 데이터 바인딩(One-way Data Binding)을 사용하고 있다. 또한 가상 돔(Virtual DOM)이라는 개념을 사용하여 웹 어플리케이션의 퍼포먼스를 최적화한 라이브러리다.

2.3 JavaScript

자바스크립트는 객체 기반의 스크립트 프로그래밍 언어이다. 이 언어는 웹 브라우저 내에서 주로 사용되며, 다른 응용 프로그램의 내장 객체에도 접근할 수 있는 기능을 가지고 있다. HTML의 특성 요소(들)을 선택하여 다양한 이벤트(마우스 클릭, 키보드 입력 등)에 따라 어떤 동작을 하도록 기능을 넣을 수 있으며 발생하는 이벤트에 따라 HTML, CSS를 조작할 수도 있고 Node.js와 같은 런타임 환경과 같이 서버 프로그래밍에도 사용되고 있다

2.4 Block Chain

블록체인은 분산 컴퓨팅 기술 기반의 데이터 위 · 변조 방지 기술이다. P2P 방식을 기반으로 하여 소규모 데이터들이 사슬 형태로 무수히 연결되어 형성된 '블록'이라는 분산 데이터 저장 환경에 관리 데이터를 저장함으로써 누구도 임의로 수정할 수 없고 누구나 변경의 결과를 열람할 수 있게끔 만드는 기술이다.

-

2.5 DID

탈중앙화 신원증명(decentralized Identifier, **DID**) 또는 분산 아이디는 기존 신원확인 방식과 달리 중앙 시스템에 의해 통제되지 않으며 개개인이 자신의 정보에 완전한 통제권을 갖도록 하는 기술이다. 블록체인을 기반으로 한 신원증명, 인증이며 이용자 스스로 자신의 신원정보를 관리하고 통제할 수 있도록 하는 디지털화된 신원관리 체계이다. 우리가 지갑에 주민등록증을 보관하고 필요할 때 나를 증명하는 것처럼, 사용자가 퍼블릭 블록체인에 연동된 디지털 지갑에 내 개인정보를 담아 필요할 때 개인키를 입력해 나를 증명하는 방식이다. 개인정보사용 및 제공의 주체가 기업에서 개인으로 변화하고 있는 상황에서 DID를 도입하면 개인이 특정 기관과 상호작용할 때, 신원주체가 그 흐름을 통제할 수 있어 신원정보를 투명하게 관리할 수 있다.

2.6 Hyperledger INDY

하이퍼레저 인디는 분산원장에 대한 독립적인 아이덴티티를 지원하는 하이퍼레저 프로젝트이다. 블록체인 또는 기타 분산원장을 기반으로 하는 디지털 ID를 제공하기 위한 도구, 라이브러리 및 재사용 가능한 구성 요소를 제공한다. 하이퍼레저 인디는 인증에 특화된 프로젝트이다. 높은 프라이 빗과 보안, 강한 아이덴티티를 위한 소프트웨어 생태계를 제공한다. 인디는 오픈소스인 분산원장을 사용하며 관리 도메인, 애플리케이션 등 서로 상호 호환이 가능하다.

2.7 Docker

도커는 리눅스 컨테이너서라는 커널 컨테이너 기술을 이용하여 만든 컨테이너 기술 중 하나다. 운영체제를 가상화 하지 않는 컨테이너 기술을 사용해 가상머신에 비해서 가볍다는 장점이 있으며, VM을 포함하여 한 대의 서버에 여러 개의 서비스를 구동하기 좋다. 또한 보안상, 서비스가 노출되더라도 원래의 서버에 영향을 미치기가 쉽지 않은 격리된 구조인 만큼, 가상화의 장점을 상당 부분 활용할 수 있다.

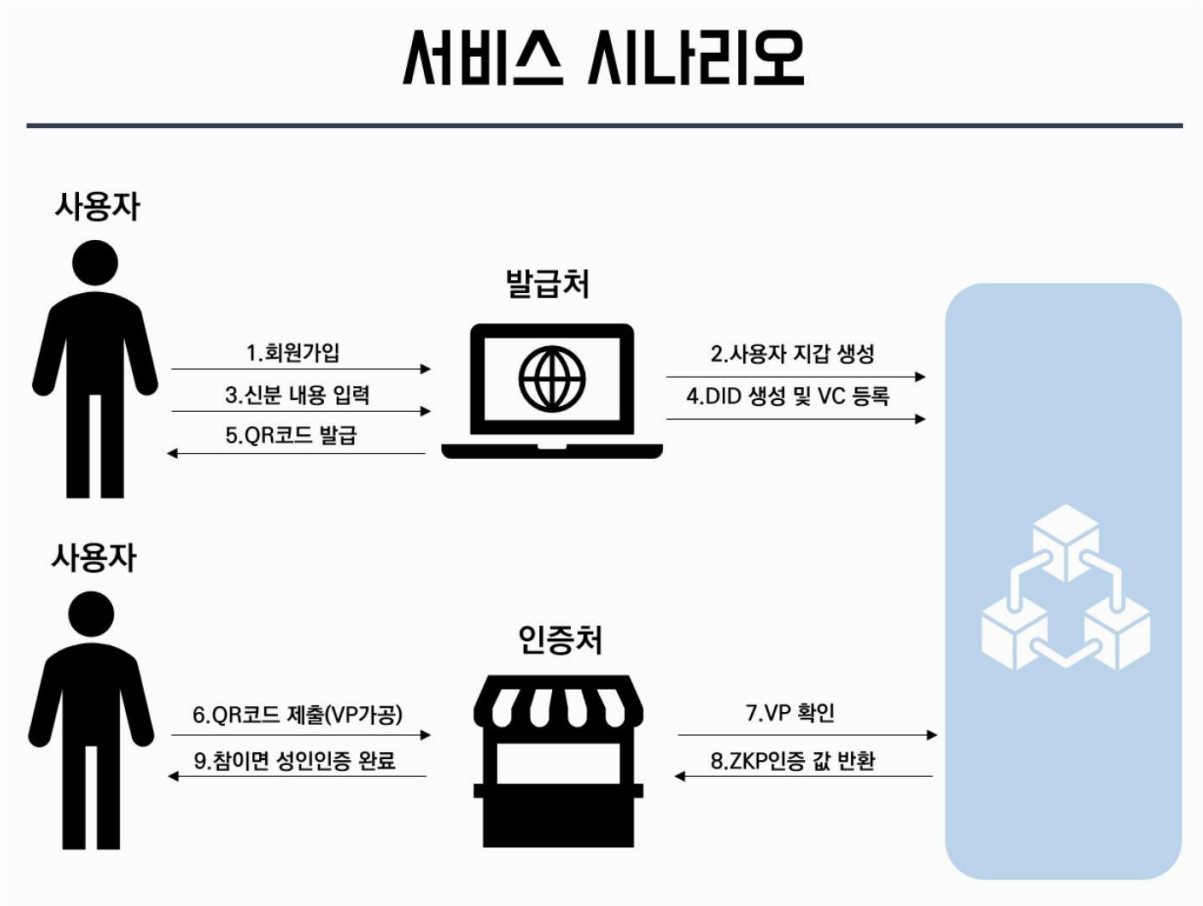
2.8 Mongo DB

몽고DB는 크로스 플랫폼 도큐먼트 지향 데이터베이스 시스템이다. NoSQL 데이터베이스로 분류되는 몽고DB는 JSON과 같은 동적 스키마형 도큐먼트들을 선호함에 따라 전통적인 테이블 기반 관계형 데이터베이스 구조의 사용을 삼간다. 이로써 특정한 종류의 애플리케이션을 더 쉽고 더 빠르게 데이터 통합을 가능케 한다.

3. 본론

3.1 서비스 구성

웹 어플리케이션 제작으로 웹 환경에서 서비스 프로그램이 사용되도록 제작을 진행하였다.

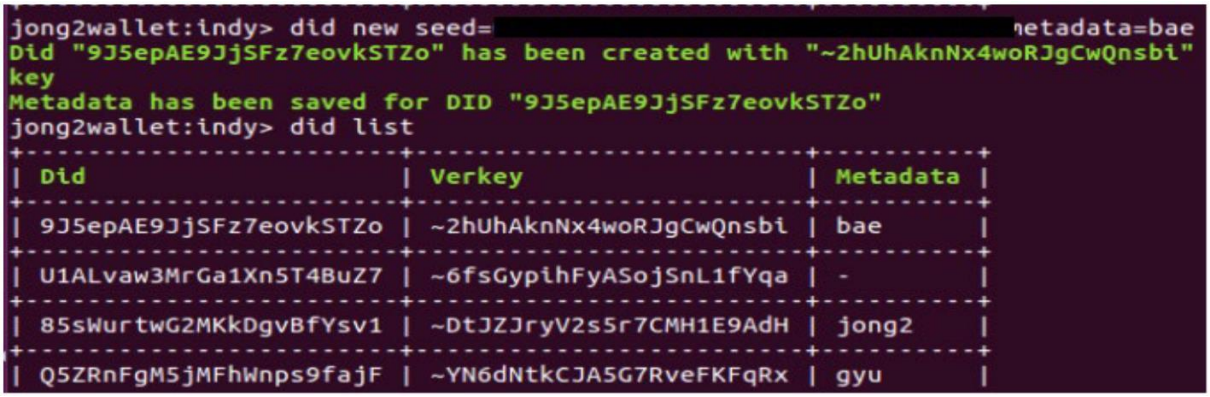


[그림 1. 서비스 구성도**]**

3.2 프로그램 구성

3.2.1 블록체인 네트워크

Hyperledger Indy 기반으로 블록체인 네트워크를 구축하였으며, Indy-cli로 제네시스 파일 제작을 위한 권한을 부여할 지갑과 DID 생성을 진행하였다.

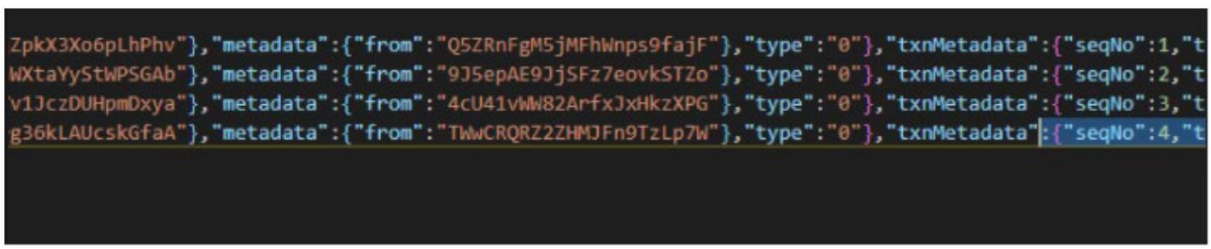


[그림 2. 블록체인 네트워크 초기 **DID** 생성**]**

지갑을 생성하기 위해서는 지갑의 이름과 key값이 필요하며, DID를 생성하기 위해서는 해당 지갑에서 생성 해야 한다. 이 때 DID에 seed값이 필요하며, 추가적으로 메타데이터를 입력할 수 있다.

[그림 3. 도메인 제네시스 파일**]**

해당 서비스에서 권한을 부여할 DID는 세 가지 이므로(하나만 부여해도 된다.)세 개의 did를 생성했으며, 해당 did에 적절한 role값을 부여하였다. 여기서 최고 권한을 가진 role은 0번이다. 이를 토대로 도메인 제네시스 파일을 생성하였으며, 이를 토대로 pool 트랜잭션 파일을 생성하였다.



[그림 4. 풀 트랜잭션 파일**]**

생성한 풀 트랜잭션 파일과 도메인 제네시스 파일을 통해 도커파일을 생성하였으며, 이 도커파일을 토대로 블록체인 네트워크를 구축한다. 블록체인 네트워크에 사용될 노드의 개수는 4

개로 지정하였다.

```
[program:node3]\n\
command=start_indy_node Node3 0.0.0.0 9705 0.0.0.0 9706\n\
directory=/home/indy\n\
stdout_logfile=/tmp/node3.log\n\
stderr_logfile=/tmp/node3.log\n\
\n\
[program:node4]\n\
command=start_indy_node Node4 0.0.0.0 9707 0.0.0.0 9708\n\
directory=/home/indy\n\
stdout_logfile=/tmp/node4.log\n\
stderr_logfile=/tmp/node4.log\n"\
>> /etc/supervisord.conf

USER indy

RUN awk '{if (index($1, "NETWORK_NAME") != 0) {print("NETWORK_NAME = \"sandbox\"")} else print($0)}' /etc/indy/indy_config.py> /
RUN mv /tmp/indy_config.py /etc/indy/indy_config.py

ARG pool_ip=[redacted]

RUN generate_indy_pool_transactions --nodes 4 --clients 5 --nodeNum 1 2 3 4 --ips="$pool_ip,$pool_ip,$pool_ip,$pool_ip"

EXPOSE 9701 9702 9703 9704 9705 9706 9707 9708

COPY --chown=indy:indy pool_transactions_genesis /var/lib/indy/sandbox/
COPY --chown=indy:indy domain_transactions_genesis /var/lib/indy/sandbox/


CMD ["/usr/bin/supervisord"]
```

[그림 5. Dockerfile]

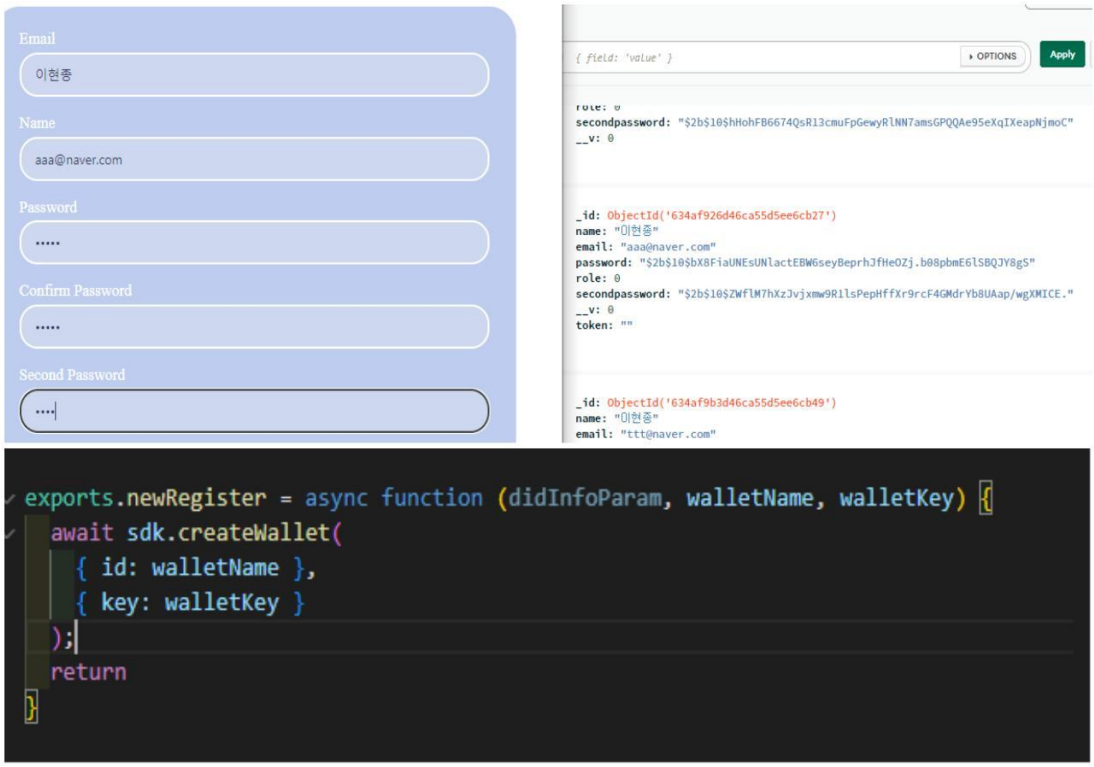
해당 도커파일을 토대로 indy_pool을 실행하고, 블록체인 네트워크를 구축하였다. 아래 이미지에서 도커로 실행 중인 indy_pool을 확인할 수가 있다. 이렇게 블록체인 네트워크를 구축하였고, 추가적인 트랜잭션은 indy-sdk 라이브러리를 사용하여, JavaScript로 진행하였다.

```
Step 26/27 : COPY --chown=indy:indy domain_transactions_genesis /var/lib/indy/sandbox/
 ---> Using cache
 ---> 822874945056
Step 27/27 : CMD ["/usr/bin/supervisord"]
 ---> Using cache
 ---> 15114379a557
Successfully built 15114379a557
Successfully tagged indy_pool:latest
bcbbb93d1e31d34163c7217c3d972306015baed7043af9adc6d48579ffd405a1
root@ubuntu:~/indy-test# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                        NAMES
bcbbb93d1e31        indy_pool           "/usr/bin/supervisord"   7 seconds ago       Up 5 seconds        0.0.0.0:9701-9708->9701-9708/tcp             indy_pool
root@ubuntu:~/indy-test#
```

[그림 6. indy_pool 실행 docker ps]

3.2.2 웹 어플리케이션

먼저, 회원가입 진행 시 사용자는 요구하는 정보들을 입력하고, 회원가입을 진행한다. 해당 내용은 DB에 저장되게 된다. 비밀번호와 2차비밀번호는 안전하게 저장해야 하므로 해시화를 진행하여, 저장하였다. 회원가입 진행과 동시에 사용자는 블록체인에 사용자의 지갑을 생성하게 되고, 저장하게 된다.

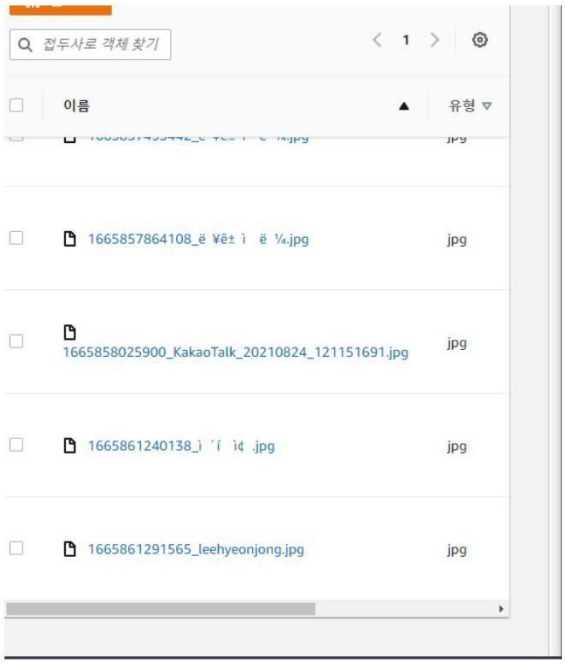


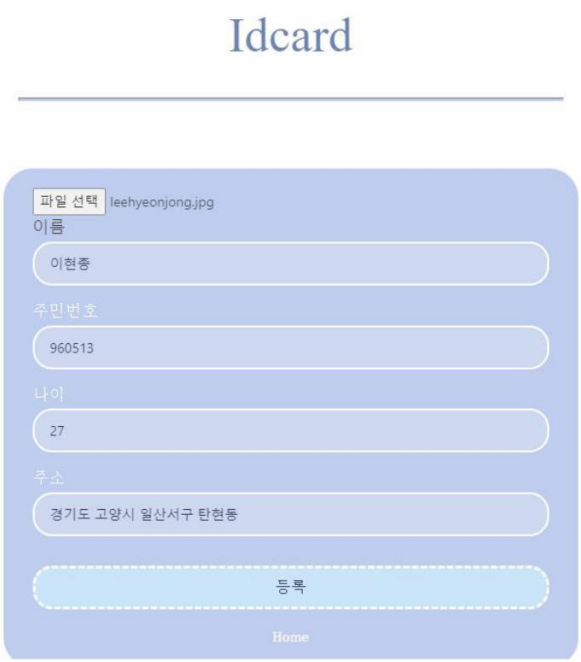
[그림 7. 회원가입**]**

로그인 시 사용자는 사용자의 이메일과 사용자 비밀번호를 입력하게 되며, 비밀번호는 입력 시 해시화하고, DB에 저장 되어 있는 값과 비교하여 올바른 비밀번호인지 확인하게 된다. 이때 로그인한 사용자에게 토큰을 발급해주며, 사용자의 로그인이 유지될 수 있도록 하였다.

[그림 8. 로그인**]**

사용자의 민증발급 시 필요한 사용자의 민증 ID 발급 등록페이지이다. [그림9] 사용자는 개인정보를 입력하고, 사용자 이미지를 등록하게 된다. 이 때 저장한 이미지는 AWS S3를 통해 저장 되며, 사용자의 등록한 정보를 블록체인의 CredentialDefinition 형식에 맞게 등록해주었다.

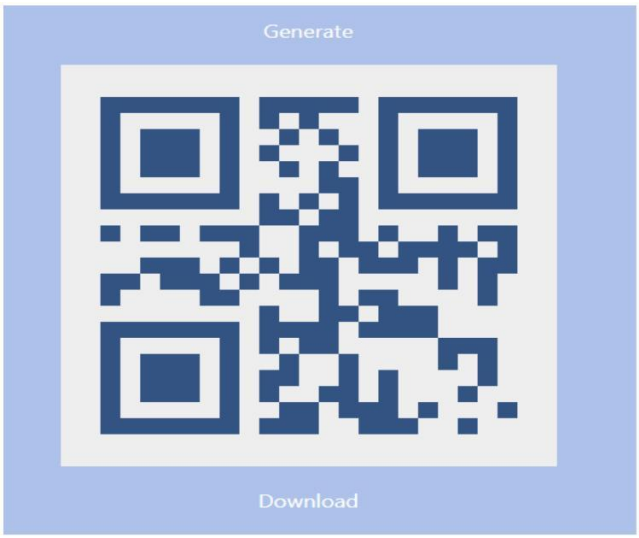


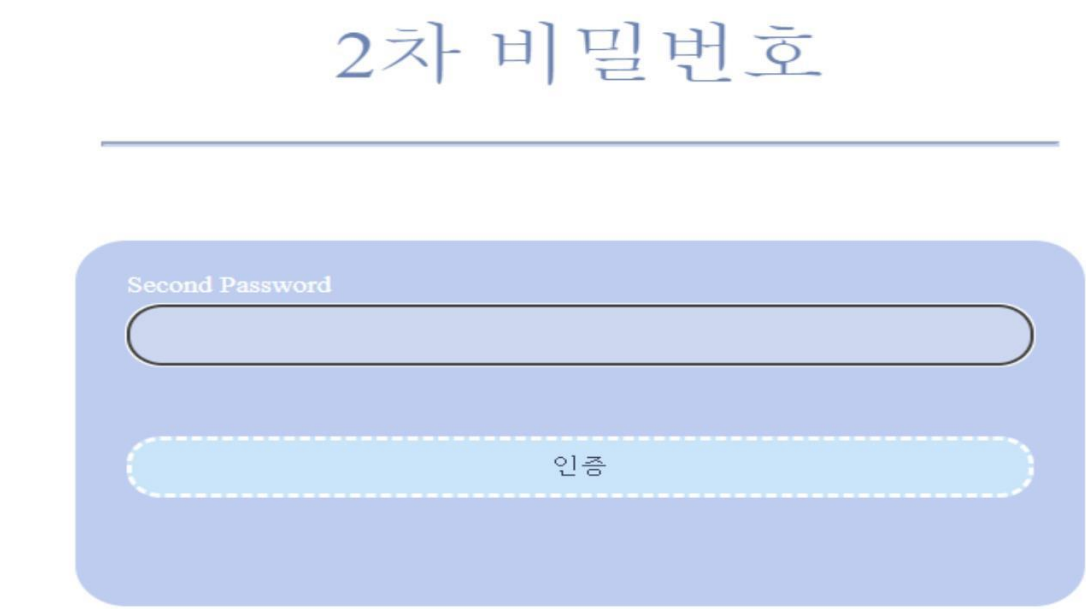



```
exports.CreateCredentialProcess = async (walletName, walletKey, value) => {
  let seedInfo = await indy.utils.walletKeyHash(walletName, walletKey);
  console.log(seedInfo);
  let proverWallet = await indy.wallet.get(walletName, walletKey);
  let issuerWallet = await indy.wallet.get(process.env.COMMUSERVEICECENTER_WALLET_NAME, process.env.COMMUSERVEICECENTER_WALLET_KEY);
  let [userDid, userVerkey] = await indy.did.createDid(seedInfo, proverWallet);
  let credOffer = await exports.sendCredOffer(issuerWallet)
  let [credReq, credReqMetaData] = await exports.sendCreateCredReq(proverWallet, credOffer)
  let [credential, revId, revRegDelta, credId] = await indy.credentials.acceptRequestCreateCredential(proverWallet, issuerWallet, credOffer, credReq, credReqMetaData, value);
  await sdk.closeWallet(issuerWallet);
  return [proverWallet, userDid, userVerkey, credential, revId, revRegDelta, credId]
}
```

[그림 9. 주민등록증 발급**]**

사용자 전자주민등록증에 대한 인증을 하고 싶다면, QR코드를 발급 받아야 하며, 이는 사용자 QR페이지에서 가능하다. 사용자 QR코드 페이지를 진입 시에 2차 비밀번호가 필요하며 2차 비밀번호를 올바르게 입력할 시에 QR코드를 발급 받을 수 있다. QR코드 발급 버튼을 누를 시 블록 체인에 접근하여 사용자 지갑에 접근하게 되고, 사용자 지갑을 통해 제출해야 할 정보들을 가공한 정보들을 암호화하여 저장하게 된다. 이 때, 2차비밀번호를 입력하고 인증을 하게 되면, 지갑 페이지에 진입이 가능해지는데, 이때 DB에는 인증 값이 저장되게 된다.(encryptedMessage에서 객체의 값으로 가짐) 이 때, 인증 값은 시간에 따라 변하는 값이다. 그렇기 때문에 QR코드의 값이 위 변조가 되어도 올바른 인증 값이 매칭이 되지 않는다면, 블록체인에서 인증이 되지 않는다. QR코드는 해당 사용자의 이메일로 입력데이터를 받았으며, 만약 공격자가 사용자의 이메일을 알게 되어 QR코드로 변환하여 QR스캐너에 입력한다 하더라도, 사용자가 2차비밀번호 확인과 로그인이 되지 않아 인증에 실패하게 된다.



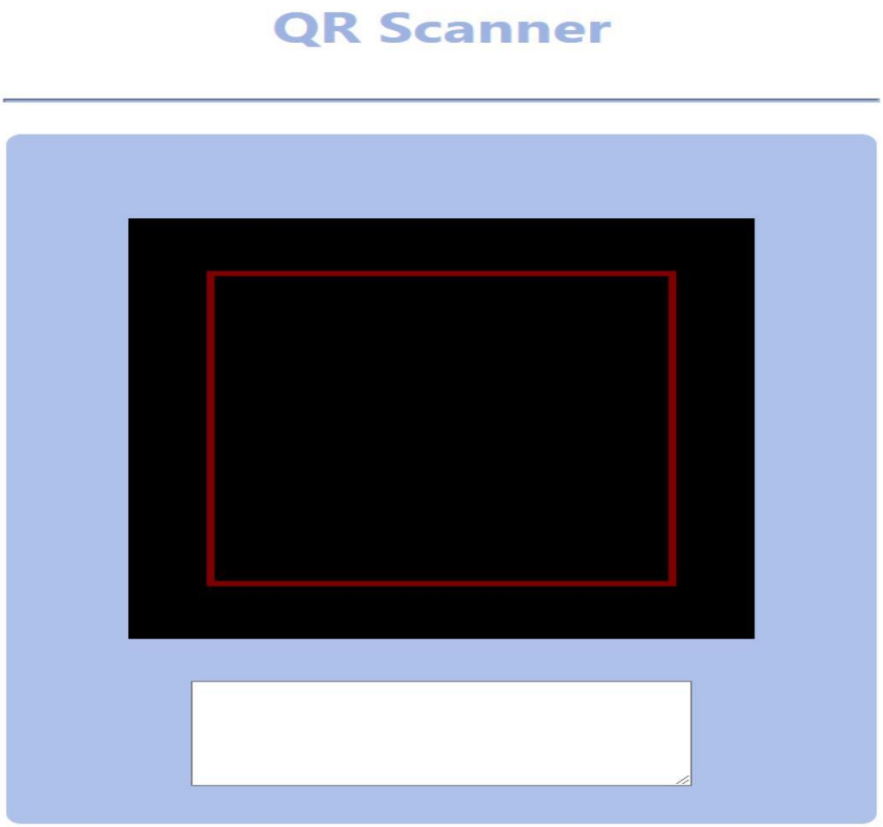



```
exports.ProverSubmitPresentation = async (proverWallet) => {
  let issuerWallet = await indy.wallet.get(
    process.env.COMMUSERVEICECENTER_WALLET_NAME,
    process.env.COMMUSERVEICECENTER_WALLET_KEY)
  let verifierWallet = await indy.wallet.get(
    process.env.STORE_WALLET_NAME,
    process.env.STORE_WALLET_KEY)
  let issuerDid = await indy.did.getDidFromWallet(issuerWallet);
  let revRegDefId = await indy.did.getEndpointDidAttribute(issuerWallet, 'revocation_registry_id')

  let [proverRevRegDelta, timestampOfDelta] = await indy.ledger.getRevRegDelta(await indy.pool.get(), issuerDid, revRegDefId[0], 0, indy.utils.getCurrentTimeInSeconds())
  let [proofRequest, credsForProof, requestedCreds] = await exports.createVerificationPresentation(issuerWallet, proverWallet, timestampOfDelta)

  let message = [proverWallet, proverRevRegDelta, timestampOfDelta, proofRequest, credsForProof, requestedCreds]
  console.log(message)
  let authCryptMessage = await indy.crypto.authCrypt(proverWallet, verifierWallet, message)
  await sdk.closeWallet(issuerWallet)
  await sdk.closeWallet(verifierWallet);
  return authCryptMessage;
```

```
__v: 0
encryptedMessage: "{"protected":"eyJlbmMiOiJ4Y2hhY2hhMjBwb2x5MTMwNV9pZXRmIiwidHlwIjoiSldN…"
```

[그림 10. 사용자 인증 **QR**코드 발급 진입 및 코드**]**

사용자의 QR코드 스캔 시, 관리자 권한을 부여 받은 인증처에서만 진입이 가능하며, 인증처에서 성인인증을 원하는 사용자가 QR코드를 발급받아 스캔 시, RAW데이터로 사용자의 이메일 값이 반환이 되며, 이와 동시에 블록체인에는 사용자의 이메일을 통해 해시화 된 데이터들을 반환을 진행하고, 반환 받은 값을 복호화를 진행하고, 이와 동시에 블록체인에 검증자의 지갑에 접근하며, 복호화된 정보를 통해 사용자의 지갑과 DID에 접근하여, 레저에 저장된 필요한 정보만을 가공한 VP에 적합한 지 True 혹은 False 값으로 반환하게 된다. 이 때 True값으로 반환을 한다면, 사용자의 성인인증에 성공하게 된 것이고, 성인인증에 성공하였다고 알람이 뜨게 된다.



```
exports.verifyProof = async (encryptedMessage) => {
  let verifierWallet = await indy.wallet.get(process.env.STORE_WALLET_NAME, process.env.STORE_WALLET_KEY);
  let verifierDid = await indy.did.getDidFromWallet(verifierWallet);
  let decryptedMessage = await indy.crypto.authDecrypt(verifierWallet, encryptedMessage)
  let userData = JSON.parse(decryptedMessage["message"]);

  console.log(typeof userData);
  let proverWallet = userData[0];

  let proverDid = await indy.did.getDidFromWallet(proverWallet);

  let masterSecretId = await indy.crypto.getMasterSecretId(proverWallet);
  let [provSchemas, provCredDefs, provRevocStates] = await indy.ledger.proverGetEntitiesFromLedger(proverWallet, proverDid, userData[4], userData[1], userData[2]);

  let proof = await sdk.proverCreateProof(proverWallet, userData[3], userData[5], masterSecretId, provSchemas, provCredDefs, provRevocStates);
  let [schemas, credDefs, revRegDefs, revRegs] = await indy.ledger.verifierGetEntitiesFromLedger(verifierDid, proof["identifiers"])
  const result = await sdk.verifierVerifyProof(userData[3], proof, schemas, credDefs, revRegDefs, revRegs);

  await sdk.closeWallet(proverWallet);
  await sdk.closeWallet(verifierWallet);
  return result
}
```

[그림 11. 관리자 **QR**코드 스캔**]**

로그아웃 시 사용자의 발급된 토큰이 빈 값으로 변경이 되며, 사용자의 쿠키 값이 DB에서 빈 값으로 저장이 되게 된다. 해당 사용자는 로그인 유지가 해지가 되며, 사용자는 서비스를 사용해야할 시 재로그인을 해야한다.

```
module.exports = {
  get: (req, res) => {
    console.log(req.user._id);
    User.findOneAndUpdate({ _id: req.user._id }, { token: "" }, (err, user) => {
      if (err) return res.json({ success: false, err });
      return res.status(200).send({
        success: true,
      });
    });
  },
};
```

[그림 12. 로그아웃**]**

4. 결론

4.1결론 및 기대효과

모바일 신분증을 사용하면 일일이 신분증을 가지고 다닐 필요성이 사라지고, 스캔 등의 절차 없이 온라인 환경에서도 간편하게 이용할 수 있다는 장점을 가지고 있다. 확인하는 사람이 원하는 정보만 제공할 수도 있어서 개인정보 노출에 대한 우려도 적어 안전하고 편리한 서비스를 제공해 줄 것이다.

4.2 향후 과제

추가적인 기능이 제대로 구현되어 있지 않다. 제일 적합한 서비스는 모바일 어플리케이션이 제일 적합하며, 가시성을 위해서 웹 어플리케이션으로 제작하였다. 추가적인 모바일 어플리케이션 개발을 한다면, 더 적절히 사용이 가능하다. 그리고 인증처에서는 사용자가 인증을 한 후 사용자의 기록을 남길 수 있도록 리스트 화하여 데이터를 날짜 별로 저장한다면, 사용자의 인증 시간까지 추적이 가능하다.

기존의 서비스에서 성인 인증 서비스의 예시로 들었으며, 해당 연구는 가상의 정부와 가상의 인증처를 구축하였고, 사설 블록체인 네트워크를 구축하였다. 만약 W3C에서 제공한 DID 표준에 맞춰 추가 개발한다면, 블록체인 네트워크에 대학증명서, 졸업증명서 등과 같은 다양한 증명서를 인증을 할 수 있으며, 다양한 서비스로 영지식 증명을 이용할 수 있다.

아직 서비스 보안적인 측면에서 부족한 점이 있다. 서비스 형태에 적절하게 추가 보안적인 보완을 시도할 예정이다.

5. 참고자료

- 자기주권 신원증명 구조 분석서 – 윤대근
- W3C DID 표준 - https://www.w3.org/TR/did-core/
- Hyperledger Indy indy-sdk Github
  - https://github.com/hyperledger/indy-sdk
- 자기주권 신원 생태계를 위한 신뢰할 수 있는 통신 방법 – 최규현,김근형

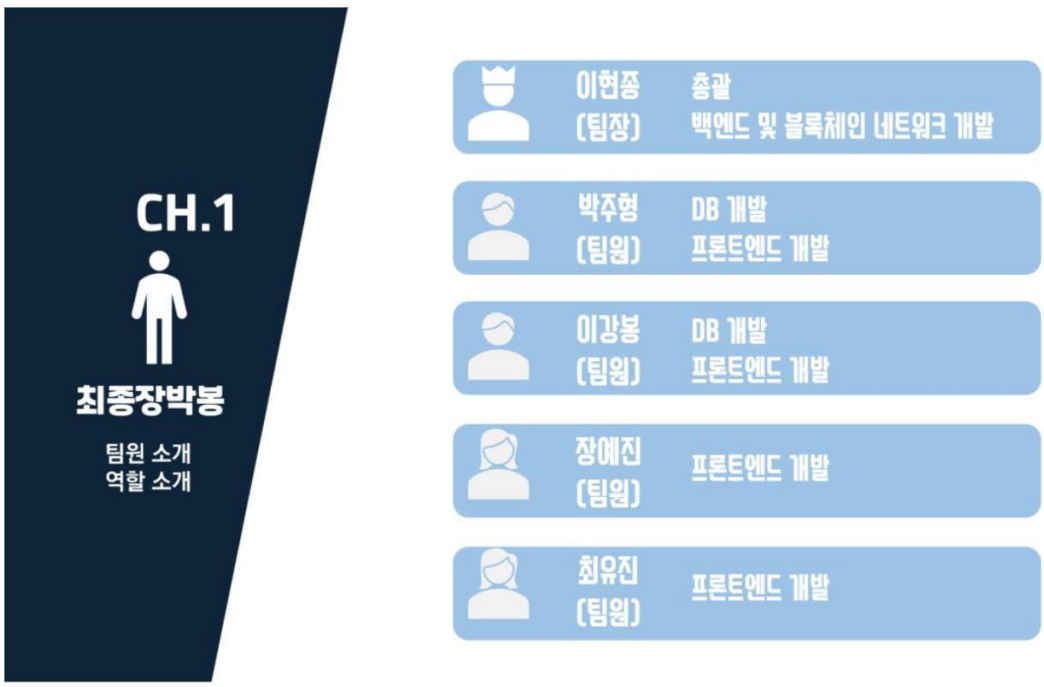
6. 별첨

6.1 깃허브 주소 - **https://github.com/ehdclr/capstonefinal.git**

6.2 웹서비스 주소 – **https://www.jpass-app.com**

6.3 발표자료


CH.1
최종장박봉
팀원 소개
역할 소개
CH.2
개발동기
CH.3
블록체인
기존 방식 문제
블록체인 소개
CH.4
본론
DID
개발환경 도구
서비스 구성도
CH.5
결론
기대효과
Q&A


CH.1
최종장박봉
팀원 소개
역할 소개
이현종 (팀장) 총괄 백엔드 및 블록체인 네트워크 개발
박주형 (팀원) DB 개발 프론트엔드 개발
이강봉 (팀원) DB 개발 프론트엔드 개발
장예진 (팀원) 프론트엔드 개발
최유진 (팀원) 프론트엔드 개발

CH.2

개발동기

J-PASS 개발 개요

현 주민등록증은 사용자의 **개인정보가 그대로 노출**되어 타인에게 쉽게 개인정보가 유출되기도 하며, 각종 첨단 장비를 통한 위 변조를 통하여 이를 방지하기 어려워지면서 이를 방지하기 위해 **탈중앙화 신원증명(Decentralized Identity, DID)**기반 모바일 신분증을 기획하게 되었다.

CH.3

블록체인

기존 방식 문제

기존 방식 문제

기존의 회원가입 방식은 중앙제어 관리 방식으로 웹사이트 자체에 개인정보를 입력하여, 회원가입을 하였다.
사용자는 각 웹사이트마다 자신의 개인정보를 입력 해야 하는 번거로움을 겪었다. 이를 극복하기 위해 기존 가입된 홈페이지를 통해 다른 웹사이트에 간편 회원가입이 가능하게 만든 SSO(Single Sign On, 통합 로그인)방식이 생기기도 하였다. (구글, NAVER 등)
이러한 방식은 중앙 관리자가 개인정보를 유출하게 될 위험성이 있다.
(실제로 페이스북과 구글은 수백만명의 개인정보를 유출한 사례가 있다.)

CH.3

블록체인

불록체인 소개

블록체인 소개

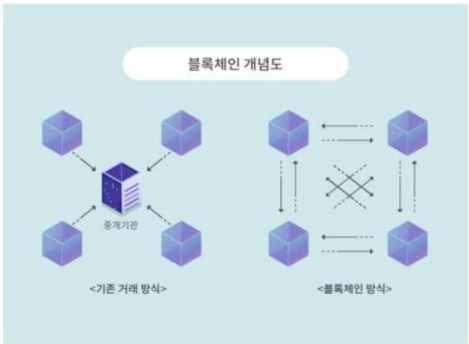
P2P(Peer to Peer) 네트워크를 통해서 관리되는 분산 데이터베이스의 형태로 분산처리와 암호화 기술을 동시에 적용하여 높은 보안성을 확보하는 한편 신속성과 투명성을 특징으로 한다.

블록 단위의 소규모 데이터들이 체인 형태로 무수히 연결되어 있어 누구도 임의로 수정할 수 없고, 누구나 변경의 결과를 열람할 수 있다는 것이 특징이다.

CH.3

블록체인

불록체인 소개



기존 중앙 기관으로 인해 관리 되었던 중앙집중 서버 방식에서 벗어나 탈중앙화 된 구조를 가지게 되어 안전한 거래를 가능하도록 만들었다.

CH.4

본론

DID

DID

블록체인을 기반으로 한 탈중앙화 신원증명 (Decentralized Identifier, DID)은 블록체인의 특장점을 활용하여 중앙 시스템에 의해 통제되던 기존 신원확인 방식과 달리 개개인이 자신의 정보에 대한 완전한 통제권을 갖도록 하는 기술이다.

SSI(Self-Sovereign Identity, 자기주권 신원증명)에 사용된다.

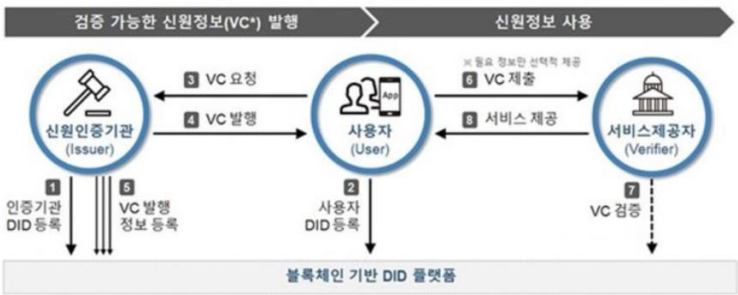
CH.4

본론

DID

SSI 구성요소

DID, DID Document, Verifiable Credential(VC),
Verifiable Presentation(VP)
VC : (보관용 ID), VP : (제출용 ID)
ZKP(Zero-Knowledge Proof)을 사용하면 속성을 알려주지 않고 검증 가능



CH.4

본론

개발환경 도구

CH.4

본론

개발환경 도구

HYPERLEDGER INDY

① 사용자 자신 외 아무도 아이덴티티를 변경/제거/삭제 할 수 없다. 자기 주권이 가능
② 인디에서 만든 아이덴티티는 여러 응용프로그램과 도메인에서 사용할 수 있고, 호환가능
③ DID를 사용하기 때문에 DID는 단일 사용자가 고유하게 소유하고 있기 때문에 ID 도용 문제 해결
④ RBFT 합의 알고리즘을 사용하여 효율적인 합의
⑤ ZKP(영지식증명)을 사용하여 다른 정보를 공개하지 않고 필요한 정보만 공개

Validation

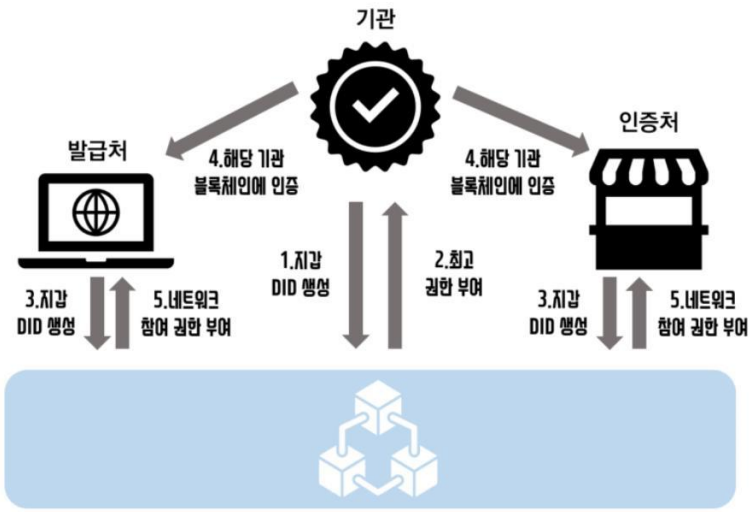
Access	Permissionless	Permissioned
Public	Bitcoin, Ethereum	Hyperledger Indy, Ripple
Private	Hollochain, LTO Network	Hyperledger Fabric, R3 Corda

CH.4

본론

서비스 구성도

초기 설정

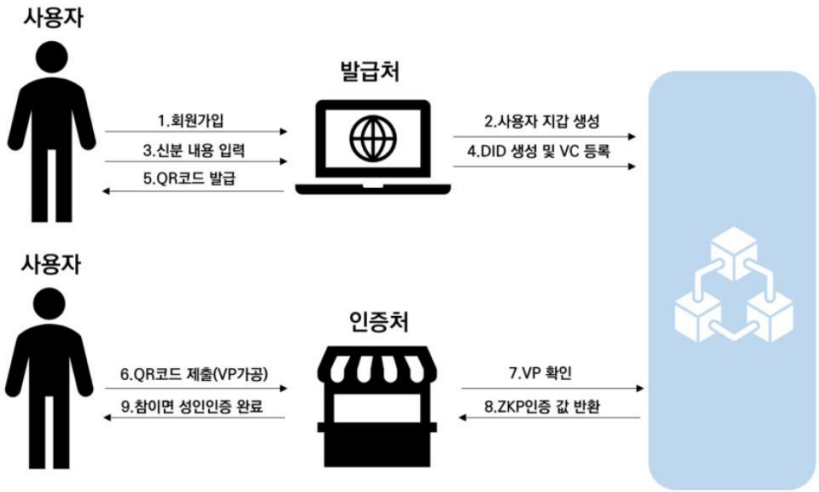


CH.4

본론

서비스 구성도

서비스 시나리오

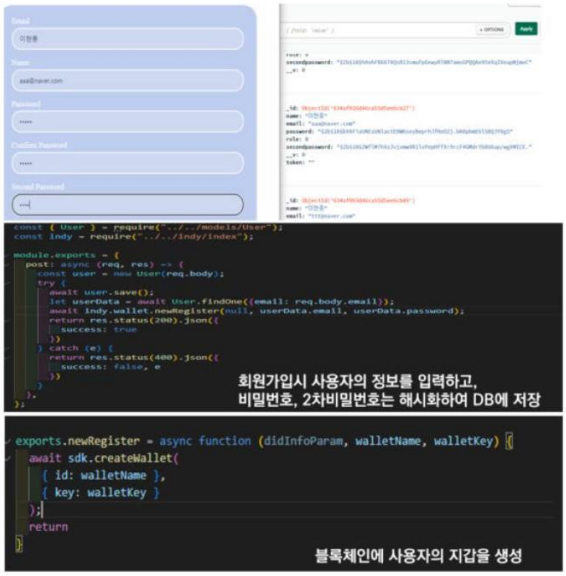


CH.4

본론

개발 내용

개발 내용 -회원가입



```
const { User } = require("../../models/User");
const indy = require("../../indy/index");

module.exports = {
  post: async (req, res) => {
    const user = new User(req.body);
    try {
      await user.save();
      let userData = await User.findOne({email: req.body.email});
      await indy.wallet.newRegister(null, userData.email, userData.password);
      return res.status(200).json({
        success: true
      })
    } catch (e) {
      return res.status(400).json({
        success: false, e
      })
    }
  },
};
```

회원가입시 사용자의 정보를 입력하고,
비밀번호, 2차비밀번호는 해시화하여 DB에 저장

```
exports.newRegister = async function (didInfoParam, walletName, walletKey) {
  await sdk.createWallet(
    { id: walletName },
    { key: walletKey }
  );
  return
}
```

블록체인에 사용자의 지갑을 생성

CH.4

본론

개발 내용

개발 내용 -로그인

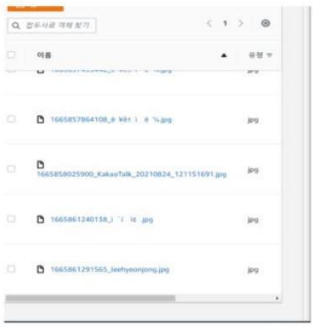
```
post: (req, res) => {
  User.findOne({ email: req.body.email }, (err, user) => {
    if (!user) {
      return res.json({
        loginSuccess: false,
        message: "제공된 이메일에 해당하는 유저가 없습니다.",
      });
    }

    user.comparePassword(req.body.password, (err, isMatch) => {
      if (!isMatch)
        return res.json({
          loginSuccess: false,
          message: "비밀번호가 틀렸습니다.",
        });

      user.generateToken((err, user) => {
        if (err) return res.status(400).send(err);

        res.cookie("x_auth", user.token).status(200).json({
          loginSuccess: true,
          userId: user._id,
          userToken: user.token,
        });
      });
    });
  });
},
```

CH.4

본론

개발 내용

개발 내용 - 발급



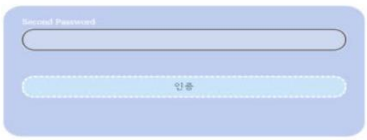
블록체인에 사용자의 DID에 대한 VC 생성

CH.4

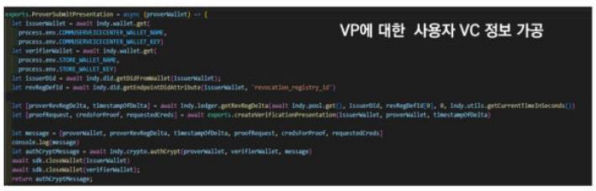
본론

개발 내용

개발 내용 - QR



VP에 대한 사용자 VC 정보 가공

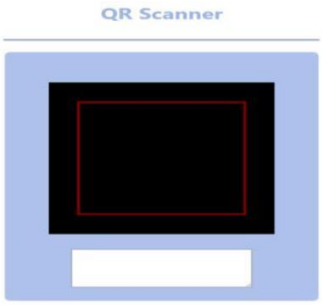


CH.4

본론

개발 내용

개발 내용 - QR




```
export verifyProof = async (encryptedMessage) => {
  let verifierWallet = await indy.wallet.get(process.env.STORE_WALLET_NAME, process.env.STORE_WALLET_KEY);
  let verifierDid = await indy.did.getDidFromWallet(verifierWallet);
  let decryptedMessage = await indy.crypto.authDecrypt(verifierWallet, encryptedMessage)
  let userData = JSON.parse(decryptedMessage["message"]);

  console.log(typeof userData);
  let proverWallet = userData[0];

  let proverDid = await indy.did.getDidFromWallet(proverWallet);

  let masterSecretId = await indy.crypto.getMasterSecretId(proverWallet);
  let [provSchemas, provCredDefs, provRevocStates] = await indy.ledger.proverGetEntitiesFromLedger(proverWallet, proverDid, userData[4], userData[1], userData[2]);

  let proof = await sdk.proverCreateProof(proverWallet, userData[3], userData[5], masterSecretId, provSchemas, provCredDefs, provRevocStates);
  let [schemas, credDefs, revRegDefs, revRegs] = await indy.ledger.verifierGetEntitiesFromLedger(verifierDid, proof["identifiers"])
  const result = await sdk.verifierVerifyProof(userData[3], proof, schemas, credDefs, revRegDefs, revRegs);

  await sdk.closeWallet(proverWallet);
  await sdk.closeWallet(verifierWallet);
  return result
}
```

인증처는 VP에 대한 사용자 인증

CH.5

결론

기대효과

기대효과

① 필요한 정보만을 제출하기 때문에 안전한 제출 가능

② 블록체인을 사용하기 때문에 무결성과 보안성을 갖춤

③ 사용자 개인이 자신의 신원 정보를 관리하기 용이

향후 계획

Q&A