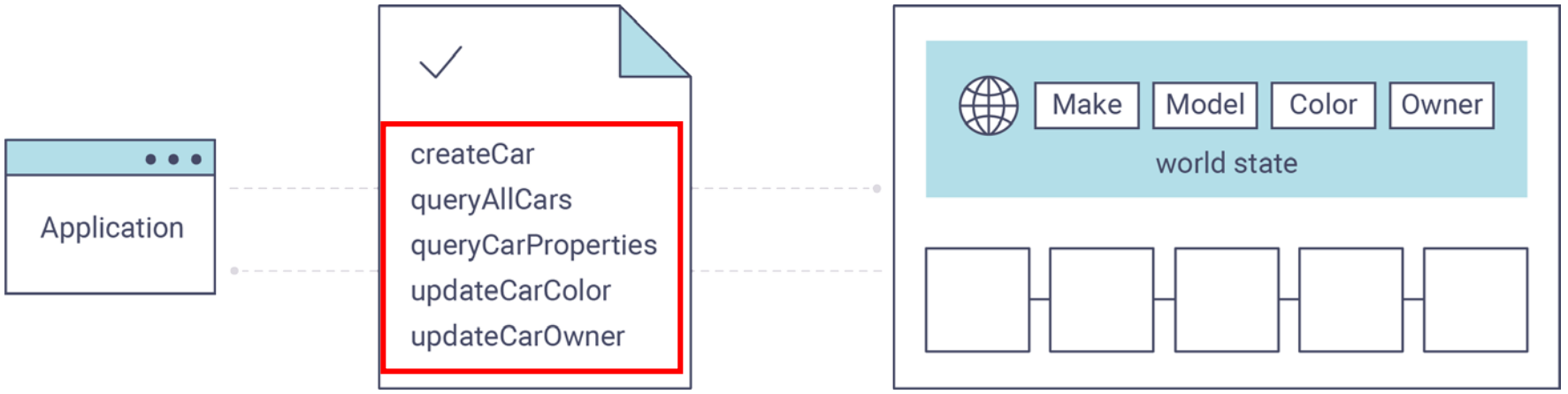
dApp 프로토타입 기획

- Dapp의 기능리스트
- Dapp에 저장할 world state
- 체인코드 이름, 함수프로토타입정의



Smart contract

기능에 따른 함수 목록 작성

ledger

World State에 저장될 데이터 목록작성

Private data가 필요할 시
Collection목록과 형식 작성

Dapp의 기능리스트

- Fabcar 의 기능
  - 자동차 정보를 추가
  - 자동차 정보를 업데이트
    - 소유주 바꾸기
  - 자동차 정보를 보여주기
    - Key를 이용하여 보여주기
    - 모두 다 보여주기
  - 가격정보를 보여주기
    - 권한이 있는 Peer만 해당 정보를 접근가능 하도록 구성

Dapp에 저장할 world state

- 항목을 구조체화 하여 실제 저장할 시에는 바이트 형식으로 저장

```
45 // Define the car structure, with 4 properties.  Structure tags are used by encoding/json library
46 type Car struct {
47         Make   string `json:"make"`
48         Model  string `json:"model"`
49         Colour string `json:"colour"`
50         Owner  string `json:"owner"`
51 }
```

Private data가 필요할 시

- private data가 필요할 시에는 collection정의 json파일을 추가로 작성
- 각 정보의 삭제되는 시점을 기준으로 구조체를 따로 작성

marbles_chaincode_private.go

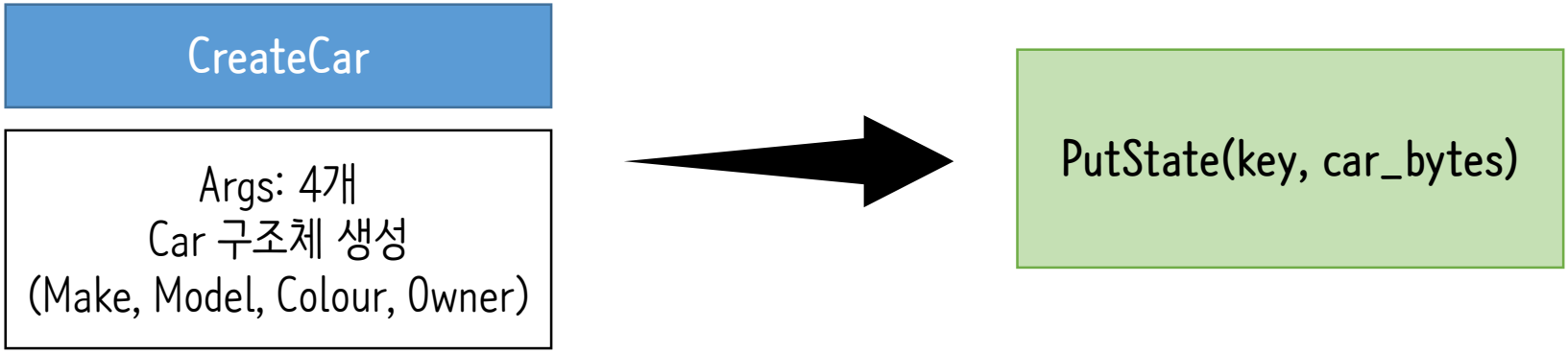
```
type marble struct {
  ObjectType string `json:"docType"`
  Name       string `json:"name"`
  Color      string `json:"color"`
  Size       int    `json:"size"`
  Owner      string `json:"owner"`
}

type marblePrivateDetails struct {
  ObjectType string `json:"docType"`
  Name       string `json:"name"`
  Price      int    `json:"price"`
}
```

collections_config.json

```
[
 {
   "name": "collectionMarbles",
   "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
   "requiredPeerCount": 0,
   "maxPeerCount": 3,
   "blockToLive":1000000,
   "memberOnlyRead": true
},
 {
   "name": "collectionMarblePrivateDetails",
   "policy": "OR('Org1MSP.member')",
   "requiredPeerCount": 0,
   "maxPeerCount": 3,
   "blockToLive":3,
   "memberOnlyRead": true
 }
]
```

체인코드 이름, 함수프로토타입정의

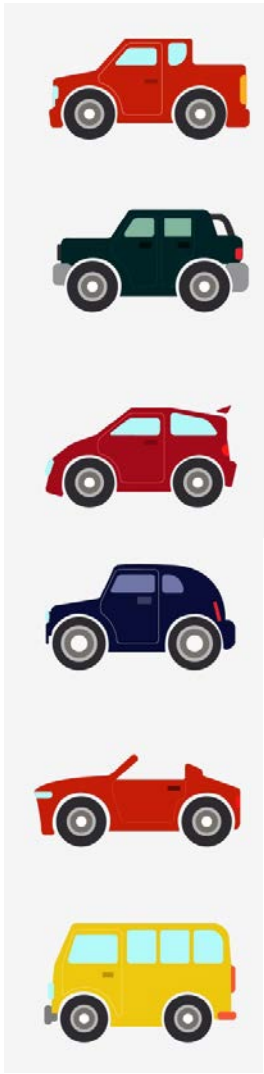
- 작성할 체인코드의 이름, 작성할 언어 선택
- 기능리스트에 따르는 함수목록 작성
- Init함수와 Invoke함수의 프로토타입과 기능 정리
- 각 기능 함수의 프로토타입을 정의 하고 함수내에서 해야 할 일을 간략히 정리
- 각 함수와 블록체인 데이터와 연관관계를 정리
- 예)



클라이언트 접속 URI설계

차정보	
make	string
model	string
color	string
owner	string

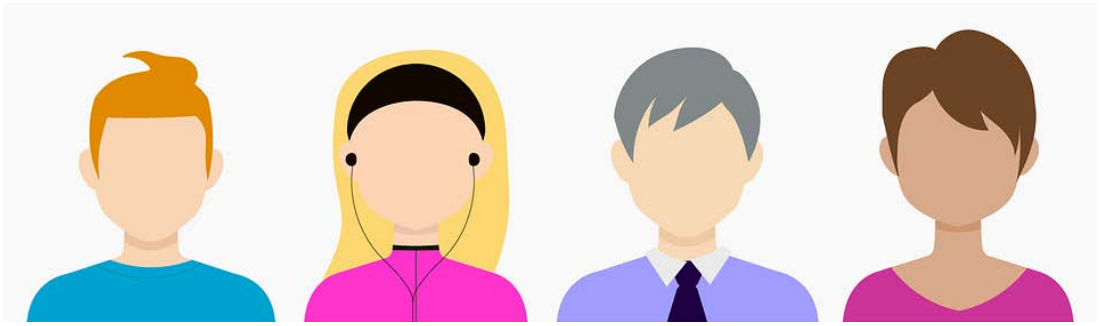
URI : /cars		
차정보 등록	post	param make model color owner
차정보 조회	get	carno
차정보 삭제	delete	carno
차정보 수정	put	carno owner



클라이언트 접속 URI설계

회원정보	
name	string

URI : /customer		
회원정보 등록	post	name
회원정보 조회	get	cusno
회원정보 삭제	delete	cusno
회원정보 수정	put	cusno name



클라이언트 접속 URI설계

거래정보	
차량 key	string
회원 key	string
가격	number

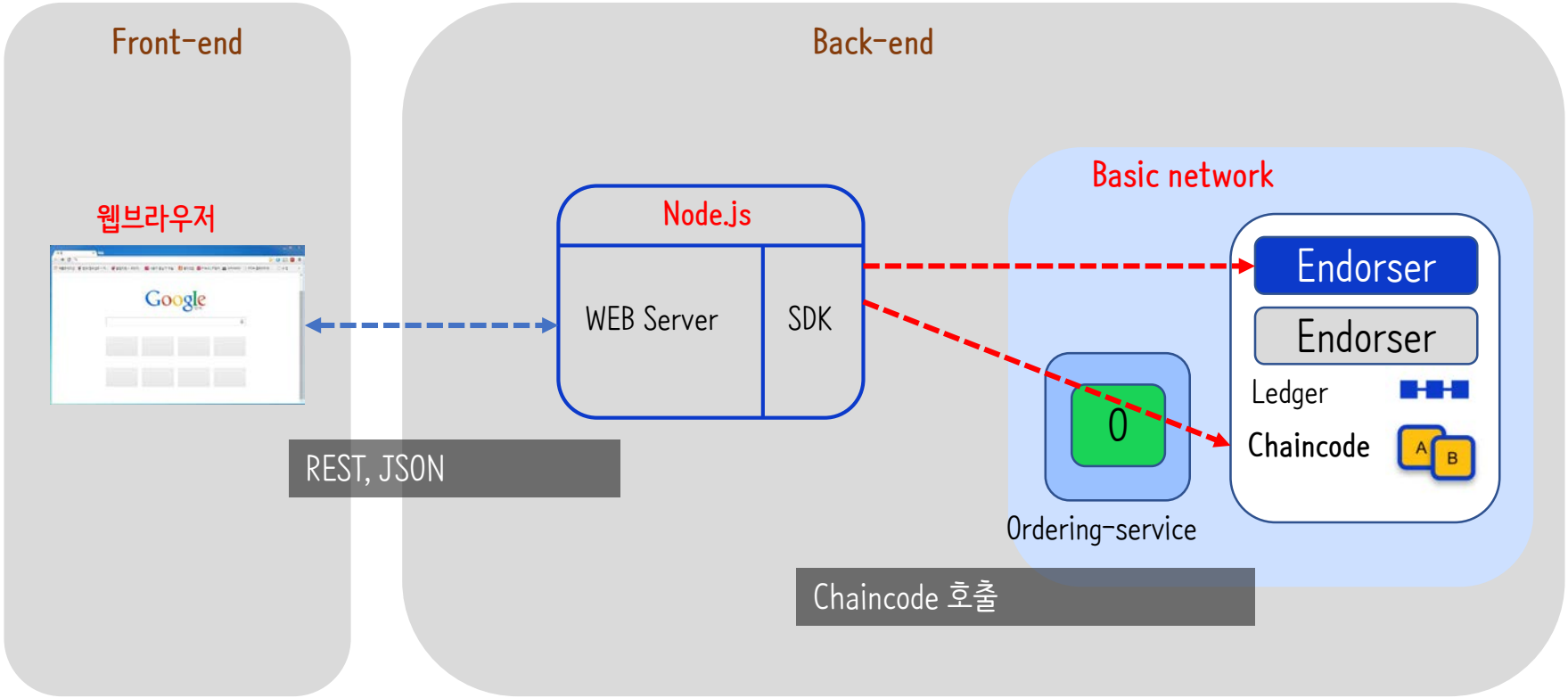
URI : /buy		
구매정보 등록	post	carno cusno price
구매정보 조회	get	buyno
구매정보 삭제	delete	buyno
구매정보 수정	put	buyno price

실습

- 문서작성
  - 기능리스트
  - 데이터
  - 체인코드 이름 및 함수리스트
- 디렉토리 생성
  - Chaincode 디렉토리 내
    - 체인코드 저장할 디렉토리 생성(myfabcar)
  - fabric-samples 디렉토리 내
    - 스크립트를 저장할 디렉토리 생성 (mynet)

front-end와 패브릭 연동

- front-end에서 사용할 인터페이스 계획



Node.js(웹서버와 체인코드 연결)

- Node.js에서 자바스크립트 형태로 체인코드의 함수 호출
- Fabric에서 제공되는 Fabric-network 유틸리티를 사용
- CA로 부터 인증 받은 인증서를 사용
- Fabric-network로 부터 채널이름을 이용하여 Gateway생성
- connection.json에 포함된 정보를 이용하여 Gateway에 연결
- 컨트랙트를 추출
- submitTransaction OR evaluateTransaction 함수를 사용하여 데이터 접근

```
const { FileSystemWallet, Gateway } = require('fabric-network');
const fs = require('fs');
const path = require('path');

const ccpPath = path.resolve(__dirname, '..', '..', 'basic-network', 'connection.json');
const ccpJSON = fs.readFileSync(ccpPath, 'utf8');
const ccp = JSON.parse(ccpJSON);

async function main() {
    try {

        // Create a new file system based wallet for managing identities.
        const walletPath = path.join(process.cwd(), 'wallet');
        const wallet = new FileSystemWallet(walletPath);
        console.log(`Wallet path: ${walletPath}`);

        // Check to see if we've already enrolled the user.
        const userExists = await wallet.exists('user1');
        if (!userExists) {
            console.log('An identity for the user "user1" does not exist in the wallet');
            console.log('Run the registerUser.js application before retrying');
            return;
        }

        // Create a new gateway for connecting to our peer node.
        const gateway = new Gateway();
        await gateway.connect(ccp, { wallet, identity: 'user1', discovery: { enabled: false } });

        // Get the network (channel) our contract is deployed to.
        const network = await gateway.getNetwork('mychannel');

        // Get the contract from the network.
        const contract = network.getContract('fabcar');

        // Submit the specified transaction.
        // createCar transaction - requires 5 argument, ex: ('createCar', 'CAR12', 'Honda', 'Accord',
        // changeCarOwner transaction - requires 2 args , ex: ('changeCarOwner', 'CAR10', 'Dave')
        await contract.submitTransaction('createCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom');
        console.log('Transaction has been submitted');

        // Disconnect from the gateway.
        await gateway.disconnect();
```

fabric-network 유틸리티를 사용하여 연결

wallet에 포함된 user1의 권한을 사용

connection.json 연결정보와 인증서를 바탕으로 Gateway에 연결

채널명을 이용하여 네트워크에 접속

체인코드 이름을 이용하여 컨트랙트 객체 가져오기

컨트랙트 객체의 submitTransaction 혹은 evaluateTransaction사용

connection.json

- 네트워크 토폴로지를 이해하는 관리자에 의해 작성
- 게이트웨이를 접근할 때 사용

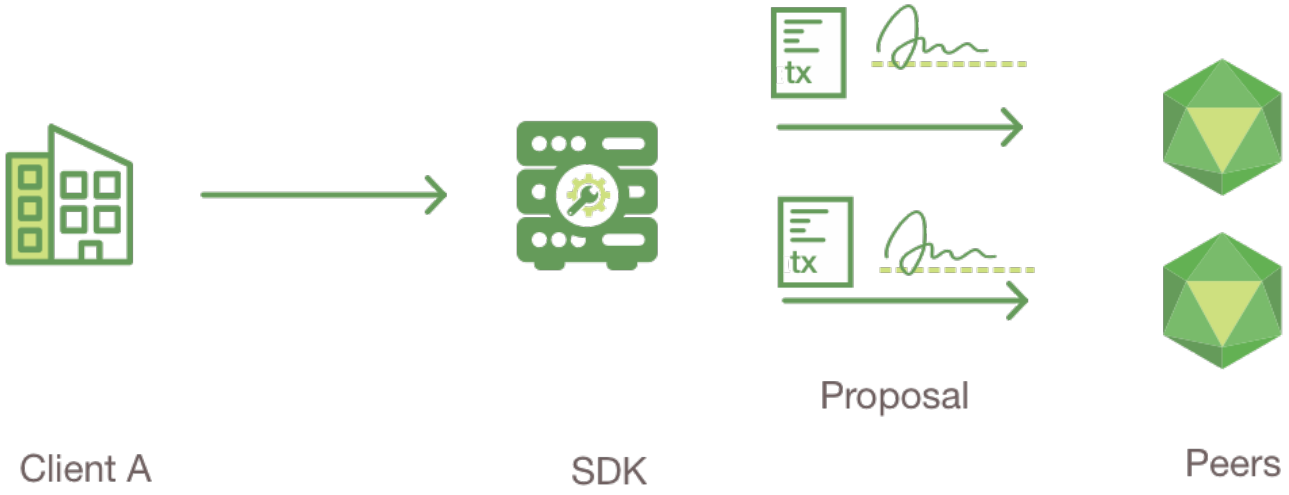
```
{
    "name": "basic-network",
    "version": "1.0.0",
    "client": {
        "organization": "Org1",
        "connection": {
            "timeout": {
                "peer": {
                    "endorser": "300"
                },
                "orderer": "300"
            }
        }
    },
    "channels": {
        "mychannel": {
            "orderers": [
                "orderer.example.com"
            ],
            "peers": {
                "peer0.org1.example.com": {}
            }
        }
    },
    "organizations": {
        "Org1": {
            "mspid": "Org1MSP",
            "peers": [
                "peer0.org1.example.com"
            ],
            "certificateAuthorities": [
                "ca.example.com"
            ]
        }
    },
    "orderers": {
        "orderer.example.com": {
            "url": "grpc://localhost:7050"
        }
    },
    "peers": {
        "peer0.org1.example.com": {
            "url": "grpc://localhost:7051"
        }
    },
    "certificateAuthorities": {
        "ca.example.com": {
            "url": "http://localhost:7054",
            "caName": "ca.example.com"
        }
    }
}
```

Transaction Flow

- 기본적인 자산교환에서의 트랜젝션 예
- 두개의 클라이언트 (A and B)

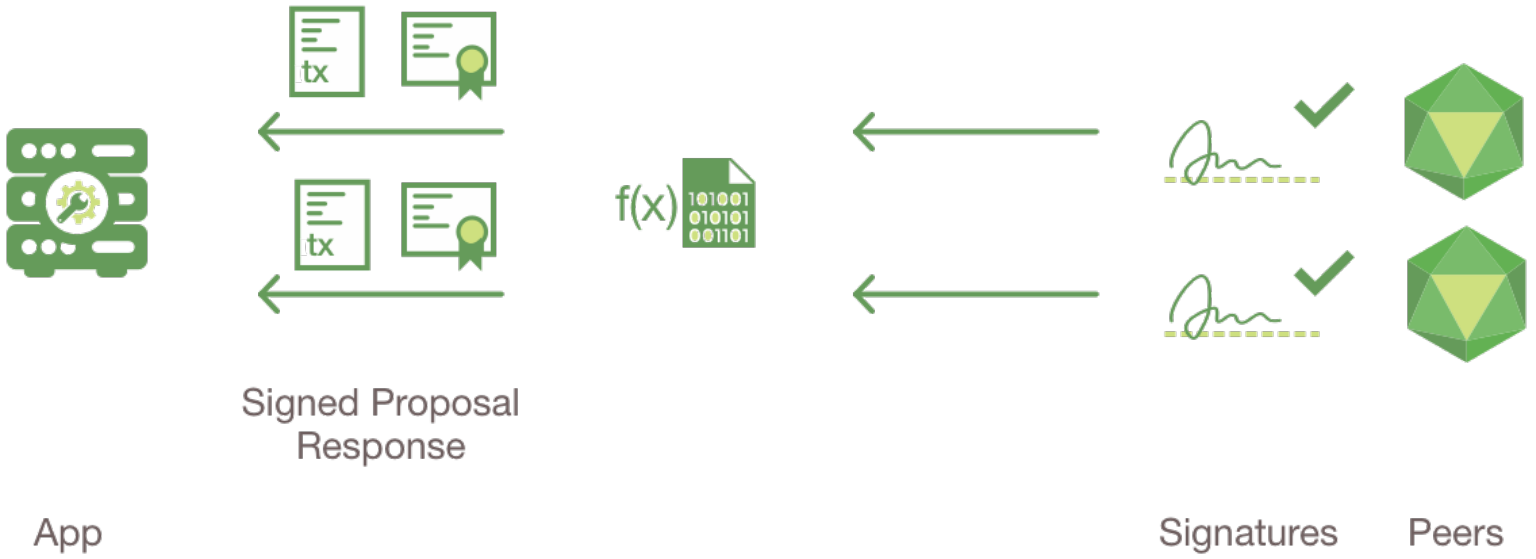
Assumptions

- 사용자등록 by ORG's CA
  - 암호정보들을 저장 (블록체인 접근 시 필요)
- 채널에 자산정보에 대한 초기정보 state가 key-value형태로 저장
- 체인코드에 거래를 위한 계약서 기능이 명시
- A와 B의 체인코드 실행을 위한 합의 정책이 정해져 있음



Client A initiates a transaction

- 클라이언트 A가 자산구매 요청을 보냄
  - PeerA, PeerB -P (PeerA and PeerB)
- 트렌젝션 프로포절 생성 (By Supported SDK-nodejs, java..)
  - 파라미터를 포함하여 체인코드 invoke 요청
- SHIM interface가 프로포절을 gRPC프로토콜을 사용하여 네트워크에 전달 (User의 인증정보 사용)

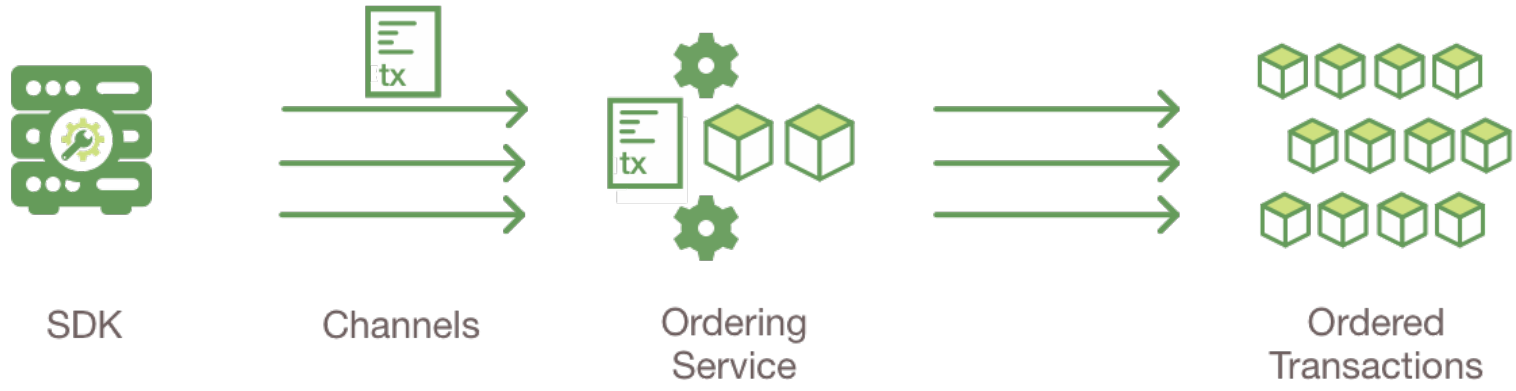


Endorsing peers verify signature & execute the transaction

- Endorsing 피어가 트렌젝션 프로포절을 검증
  - 프로포절의 오류검증
  - replay-attack 검증
  - 시그니처가 유효한지 MSP를 이용하여 검증
  - Client A가 channel에 Write 권한이 있는지 검증
- state DB에 응답에 필요한 값, readset, writeset 을 위해 실행
  - NO UPDATE
- 프로포절을 response

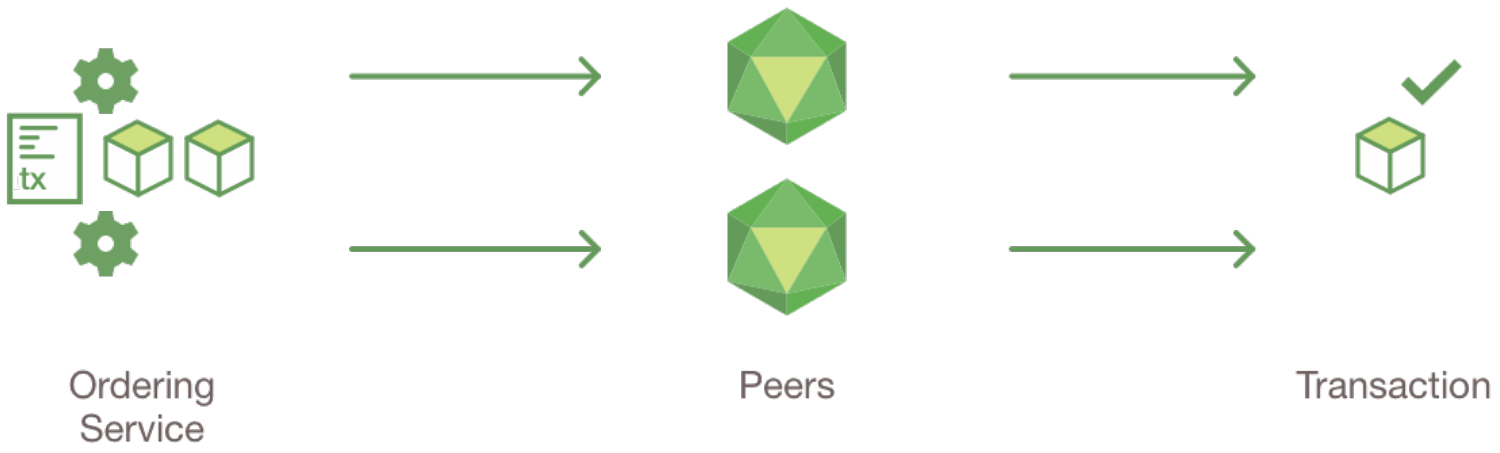
Proposal responses are inspected

- 어플리케이션이 endorsing peer의 시그니처를 검증하고 프로포절 response를 비교
- query의 경우 orderer에게 이후 트렌젝션을 보내지 않음
- 트렌젝션을 orderer에게 전달



Client assembles endorsements into a transaction

- Orderer가 해당 트렌젝션 프로포절을 Committing peer에게 BROADCAST
- 트랜젝션 : read/write sets, endorsing peers sign, ch ID
- 프로포절의 사인만 확인하고 각 피어들에게 전달



Transaction is validated and committed

- 트렌젝션의 블록이 채널의 각 피어들에게 전달
- read set/write set이 ledger의 동기화에 문제가 없으면 정보를 업데이트

Ledger updated

- 채널에 포함된 피어들의 원장에 블록을 추가
- 현재 state DB에 writesets을 업데이트
- 어플리케이션에 트렌젝션이 성사되었다고 Event를 보냄
  - 유효하거나 유효하지 않거나 모두 보냄
  - submitTransaction API는 자동으로 이벤트를 LISTEN

https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html#

Hyperledger Fabric SDK for node.js

- https://fabric-sdk-node.github.io/master/index.html

create channels

ask peer nodes to join the channel

install chaincodes in peers

instantiate chaincodes in a channel

invoke transactions by calling the chaincode

query the ledger for transactions or blocks

Features of SDK for Node.js

fabric-network

- Submitting transactions to a smart contract
- Querying a smart contract

fabric-client

- Channel operation
- Chaincode operation
- Query of channel info.
- monitoring event
- ...

fabric-ca-client

- register, enroll, revoke, customizable persistence store

Module: fabric-network

Example

```
// Obtain the smart contract with which our application wants to interact
const wallet = new FileSystemWallet(walletDirectoryPath);
const gatewayOptions: GatewayOptions = {
    identity: 'user@example.org', // Previously imported identity
    wallet,
};
const gateway = new Gateway();
await gateway.connect(commonConnectionProfile, gatewayOptions);
const network = await gateway.getNetwork(channelName);
const contract = network.getContract(chaincodeId);

// Submit transactions or evaluate queries for the smart contract
const result = await contract.createTransaction(transactionName)
    .setTransient(privateData)
    .submit(arg1, arg2);
```

fabric-network. Gateway

Example

```
const gateway = new Gateway();
const wallet = new FileSystemWallet('./WALLETS/wallet');
const ccpFile = fs.readFileSync('./network.json');
const ccp = JSON.parse(ccpFile.toString());
await gateway.connect(ccp, {
  identity: 'admin',
  wallet: wallet
});
```

실습

- mynet내 application 디렉토리 생성
  - application 내에 query.js, invoke.js 복사 (옵션)
  - package.json파일 작성
  - 모듈설치
- collection.json 복사 및 수정 (네트워크이름)
- fabric-client, fabric-ca-client, fabric-network를 설치
  - npm install fabric-client
  - npm install fabric-ca-client
  - npm install fabric-network
  - 버전확인

```
bstudent@bstudent-VirtualBox:~/fabric-samples/first-network$ fabric-ca-client version
fabric-ca-client:
 Version: 1.4.1
 Go version: go1.11.5
 OS/Arch: linux/amd64
```