껄껄껄 포팅 메뉴얼

프로젝트 사용 툴

- 이슈 관리 : JIRA
- 형상 관리 : Gitlab
- 커뮤니케이션 : Notion, Mattermost
- 디자인 : Figma
- ERD : ERD Cloud
- CI/CD : Jenkins, Docker
- Message : Firebase Cloud Messaging

개발 환경

FrontEnd

- Visual Studio Code
- **vite 5.2.0**
- **typescript 5.2.2**
- **tanstack query v5, zustand**
- prettier, eslint, husky
- **tailwind, next-ui**
- **pwa**
- **firebase messaging 8.10**
- axios

- **sockjs, stompjs**
- **msw**
- nivo
- qr-scanner

BackEnd

- IntelliJ
- Java `17`
- Spring boot `3.3.2`
- Redis

AI

- ChatGPT `4o-turbo`

Server

- ec2
- s3
- docker `24.0.7`
- mysql `8.3.0`
- jenkins `2.469`
- nginx `1.27`

BlockChain

- ipfs
- hardhat
- blocksout

포트 환경

EC2

Spring Boot	8080, 8081
Jenkins	7777
Mysql	3306
Redis	6379
Nginx	80, 443
React	3000

Other Server(BlockChain)

blockchain network	8545
ipfs-in	5001
ipfs-out	5002

설정 파일 및 환경변수

Backend

application.yml

```yaml
server:
  servlet:
    encoding:
      charset: UTF-8
    session:
      cookie:
        name: SESSIONID
        secure: true
        http-only: true
        same-site: none
        path: /


spring:
  profiles:
    include: dev, secret, blockchain
  jpa:
    show-sql: true
    open-in-view: false #default : true이기에 수정해야 함.
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        highlight_sql: true
        format_sql: true
        default_batch_fetch_size: 1000
  sql:
    init:
      mode: always
  servlet:
    multipart:
      max-file-size: 50MB
      max-request-size: 50MB

game:
  reward:
    term: 5
```

```yaml
challenge:
  reward:
    term: 60
  team:
    success:
      winner: 3
      looser: 2
    failure:
      default : 0.5
  personal:
    success: 2
    failure : 0.5
```

⚠ **git에 올리기 민감한 정보는 아래 secret, dev, blockchain 파일로 값 세팅하였음. jenkins의 credential 설정을 통해 값을 세팅**

application-secret.yml

```yaml
spring:
  mail:
    host: {smtp host}
    port: {smtp 포트}
    username: {smtp 계정}
    password: {smtp 비밀번호}
    properties:
      mail:
        transport:
          protocol: smtp
        smtp:
          auth: true
          starttls:
            enable: true
          ssl:
            trust: smtp.naver.com
            enable: true
        debug: false
cloud:
  aws:
    credentials:
      access-key: {aws s3 access-key}
      secret-key: {aws s3 secret-key}
    s3:
      bucket: {aws s3 bucket}
    region:
      static: {aws s3 region}
    stack:
      auto: false
account:
  base-url: {ssafy api base-url}
  api-key: {ssafy api key}
  institution-code: {ssafy api institution code}
  fintech-app-no: {ssafy api fintech app no}
```

```yaml
openai:
  model: {oepnai model}
  api:
    key: {openai api key}
    url: https://api.openai.com/v1/chat/completions
```

application-dev.yml

```yaml
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:{mysql 포트}/{mysql 스키마명}?serverTimezone=Asia/Seoul
    username: {db 계정}
    password: {db 패스워드}

  data:
    redis:
      host: {redis host}
      password: {redis password}
      port: {redis port}
```

application-blockchain.yml

```yaml
blockchain:
  network:
    host: {blockchain network host}
    port: {blockchain network port}
  scanner:
    host: {blockchain scanner host}
  admin:
    wallet:
      address: {wallet address}
      private-key: {wallet private key}
  smart-contract:
    address:
      token: {smart-contract token key}
      equipment-draw: {smart-contract equipment-draw key}
      equipment-nft: {smart-contract equipment-nft key}
      market: {smart-contract market key}
      application: {smart-contract application key}

ipfs:
  host: {infs host}
  port:
    api: {infs port}
    gateway: {infs gateway}
```

firebase-secret-key.json

```
{
  "type": "service_account",
  "project_id": {project_id},
  "private_key_id": {private_key_id},
  "private_key": "{private_key},
  "client_email": {client_email},
  "client_id": {client_id},
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-e
  "universe_domain": "googleapis.com"
}
```

배포

Docker

⚠ **무중단 Blue/Green 전략으로 spring boot를 구성함**

DockerFile

```
# Base image
FROM bellsoft/liberica-openjdk-alpine:17

# Set working directory
WORKDIR /app

# Copy all files to the container
COPY . .

# Run Gradle build
RUN chmod +x ./gradlew

RUN ./gradlew clean build -x test

RUN ls -al ./build/libs

RUN cp ./build/libs/application-0.0.1-SNAPSHOT.jar app.jar

RUN apk add tzdata && ln -snf /usr/share/zoneinfo/Asia/Seoul /etc/localtime

#EXPOSE 8080
```

```
cmd ["java", "-jar", "app.jar"]
```

docker-compose

```yaml
services:
  database:
    image: mysql:8.3.0
    restart: always
    ports:
      - "3306:3306"
    volumes:
      - /home/ubuntu/data/db/mysql:/var/lib/mysql/
      - /home/ubuntu/data/db/my.cnf:/etc/mysql/my.cnf
    container_name: mysql_container
    environment:
      MYSQL_DATABASE: GGUL3
      MYSQL_ROOT_HOST: "%"
      TZ: Asia/Seoul
      MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
    command: [ '--character-set-server=utf8mb4', '--collation-server=utf8mb4_unicode_ci' ]
    networks:
      - custom_network

  nginx:
    restart: always
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443" # Nginx°¡ 443 Æ÷Æ®(HTTPS ±â°» Æ÷Æ®)¿¡¼µµ ¼ö½Å ´ë±âÇÏµµ·Ï PORTS Ãß°¡
    container_name: nginx_container
    environment:
      - TZ=Asia/Seoul
    volumes:
      - /etc/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - /etc/nginx/conf.d/service-url.inc:/etc/nginx/conf.d/service-url.inc
      - /etc/letsencrypt/live/ggul3.kro.kr/fullchain.pem:/etc/nginx/certs/fullchain.pem:ro
      - /etc/letsencrypt/live/ggul3.kro.kr/privkey.pem:/etc/nginx/certs/privkey.pem:ro
    networks:
      - custom_network

  redis:
    image: redis
    ports:
      - "6379:6379"
    environment:
      - REDIS_PASSWORD=${REDIS_PASSWORD}
      - TZ=Asia/Seoul
    command: redis-server --requirepass ${REDIS_PASSWORD}
    container_name: redis_container
    networks:
      - custom_network

volumes:
```

```yaml
  mysql_data:


networks:
  custom_network:
    name: custom_network
```

docker-compose.ggul-8080.yml

```yaml
#version: "3.2"

services:
  spring:
    image: ${DOCKER_NAMESPACE}/${DOCKER_REPO}:latest
    restart: always
    ports:
      - "8080:8080"
    environment:
      - REDIS_PASSWORD=${REDIS_PASSWORD}
      - HTTP_PORT=8080
      - TZ=Asia/Seoul
      - JAVA_OPTS=-Duser.timezone=Asia/Seoul
    container_name: ggul-8080
    networks:
      - custom_network

networks:
  custom_network:
    external: true
```

docker-compose.ggul-8081.yml

```yaml
#version: "3.2"

services:
  spring:
    image: ${DOCKER_NAMESPACE}/${DOCKER_REPO}:latest
    restart: always
    ports:
      - "8081:8080"
    environment:
      - REDIS_PASSWORD=${REDIS_PASSWORD}
      - HTTP_PORT=8080
      - TZ=Asia/Seoul
      - JAVA_OPTS=-Duser.timezone=Asia/Seoul
    container_name: ggul-8081
    networks:
      - custom_network
```

```yaml
networks:
  custom_network:
    external: true
```

deploy.sh

```bash
EXIST_GGUL=$(sudo docker-compose --env-file /etc/docker/env/.env -p ggul-8080 -f /home/ubuntu/do

if [ -z "$EXIST_GGUL" ]; then
    pwd
    ls -al
    sudo docker-compose --env-file /etc/docker/env/.env -p ggul-8080 -f /home/ubuntu/docker-comp
    BEFORE_COLOR="8081"
    AFTER_COLOR="8080"
    BEFORE_PORT=8081
    AFTER_PORT=8080
else
    pwd
    ls -al
    sudo docker-compose --env-file /etc/docker/env/.env -p ggul-8081 -f /home/ubuntu/docker-comp
    BEFORE_COLOR="8080"
    AFTER_COLOR="8081"
    BEFORE_PORT=8080
    AFTER_PORT=8081
fi

echo "BEFORE_PORT: $BEFORE_PORT"
echo "AFTER_PORT: $AFTER_PORT"

for cnt in `seq 1 10`;
do
    UP=$(curl -s http://127.0.0.1:${AFTER_PORT}/health-check)
    if [ "${UP}" != "OK" ]; then
        sleep 10
        continue
    else
        break
    fi
done

if [ $cnt -eq 10 ]; then
    sudo docker rm ggul-${AFTER_COLOR} -f
    exit 1
fi


sudo sed -i "s/${BEFORE_PORT}/${AFTER_PORT}/" /etc/nginx/conf.d/service-url.inc

sudo docker restart nginx_container

sudo docker-compose -p ggul-${BEFORE_COLOR} -f /home/ubuntu/docker-compose.ggul-${BEFORE_COLOR}
sudo docker image prune -f
```

Nginx

pem키 인증서 발급

```
sudo apt-get update
sudo apt-get install software-properties-common
sudo add-apt-repository universe
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install certbot python3-certbot-nginx

// certbot을 이용해 인증서 발급
sudo certbot certonly --standalone
```

docker-compose를 통해 nginx를 설정할 때 https 설정을 위한 pem 키 volume 설정

```
- /etc/letsencrypt/live/ggul3.kro.kr/fullchain.pem:/etc/nginx/certs/fullchain.pem:ro
- /etc/letsencrypt/live/ggul3.kro.kr/privkey.pem:/etc/nginx/certs/privkey.pem:ro
```

80포트 443포트 설정

```
worker_processes  auto;
events { worker_connections 1024; }

http {
  include       mime.types;
  default_type  application/octet-stream;

  sendfile        on;
  keepalive_timeout  65;

  server {
    listen 80;
    server_name ggul3.kro.kr;
    location /{
        return 301 https://$host$request_uri;
    }

 }


  server{
        listen 443 ssl;
        server_name ggul3.kro.kr;
```

```
        ssl_certificate /etc/nginx/certs/fullchain.pem;
        ssl_certificate_key /etc/nginx/certs/privkey.pem;

        include /etc/nginx/conf.d/service-url.inc;
        location / {
                root /usr/share/nginx/html;
                index index.html;
                try_files $uri $uri/ /index.html;
        }

        location /api/ {
                if ($http_origin ~* (https://ggul3.kro.kr|http://localhost:5173|http://localhost
                        add_header 'Access-Control-Allow-Origin' $http_origin always;
                        add_header 'Access-Control-Allow-Credentials' 'true' always;
                        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, PATCH, OPTION
                        add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept,
                        add_header 'Access-Control-Expose-Headers' 'access' always;
                }
                if ($request_method = 'OPTIONS') {
                        add_header 'Access-Control-Allow-Origin' $http_origin;
                        add_header 'Access-Control-Allow-Credentials' 'true';
                        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, PATCH, OPTION
                        add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept,
                        return 204;
                }

                proxy_pass http://spring:8080/;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
        }

        location /api/stomp/connection/ {
                proxy_pass http://spring:8080/stomp/connection/;
                proxy_set_header Host $host;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;

        }
}

# j11d101.p.ssafy.io로 들어오는 요청을 ggul3.kro.kr로 리디렉션
server {
  listen 80;
  server_name j11d101.p.ssafy.io;

  # HTTP 요청을 HTTPS로 리디렉션
  return 301 https://ggul3.kro.kr$request_uri;
}

server {
```

```
    listen 443 ssl;
    server_name j11d101.p.ssafy.io;

    #ssl_certificate /etc/letsencrypt/live/j11d101.p.ssafy.io/fullchain.pem;
    #ssl_certificate_key /etc/letsencrypt/live/j11d101.p.ssafy.io/privkey.pem;

    # HTTPS 요청을 ggul3.kro.kr로 리디렉션
    return 301 https://ggul3.kro.kr$request_uri;
  }
}
```

BlockChain

> ⚠ **블록체인은 EC2가 아닌 다른 서버를 통해 배포**

기본 설정 값

- Blockscout URL : ggul-chain.kro.kr
- Blockchain Network URL : ggul-chain.kro.kr:8545
- IPFS URL
  - in : ggul-ipfs:5001
  - out : ggul-ipfs:5002

Hardhat 세팅

- Bash Shell

```
npm install --save-dev hardhat
npx hardhat
```

- hardhat.config.js

```
require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.24",
  networks: {
    'awesome-chain': {
      url: 'http://localhost:8545',
    },
    hardhat: {
      accounts: [
        {
          privateKey: "0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80",
          balance: "10000000000000000000000000"
        }
      ]
    }
```

```
  },
  etherscan: {
    apiKey: {
      'awesome-chain': 'empty'
    },
    customChains: [
      {
        network: "awesome-chain",
        chainId: 31337,
        urls: {
          apiURL: "http://localhost/api",
          browserURL: "http://localhost"
        }
      }
    ]
  }
};
```

- Bash Shell

```
npx hardhat node --network hardhat --hostname 0.0.0.0
```

Contract Upload

- hardhat/contracts에 파일 삽입
  - TokenContract.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract TokenContract is ERC20 {

    address public agent;

    mapping(address=>bool) private useTokenAllowedContractList;
    mapping(address=>bool) private transferTokenAllowedContractList;

    constructor(uint256 initialSupply) ERC20("GGUL Token", "GGUL") {
        agent = msg.sender;
        _mint(msg.sender, initialSupply);
    }

    function decimals() public view virtual override returns (uint8) {
        return 0;
    }

    function mint(uint256 _amount) public {
        require(msg.sender == agent);
        _mint(msg.sender, _amount);
    }

    function registerUseTokenAllowedContractList(address _contractAddress) public {
        require(msg.sender == agent);
```

```
        useTokenAllowedContractList[_contractAddress] = true;
    }

    function registerTransferTokenAllowedContractList(address _contractAddress) public {
        require(msg.sender == agent);
        transferTokenAllowedContractList[_contractAddress] = true;
    }

    function useToken(address _spender, uint256 _amount) public {
        require(useTokenAllowedContractList[msg.sender] || (agent == msg.sender));
        _transfer(_spender, agent, _amount);
    }

    function transferToken(address _from, address _to, uint256 _amount) public {
        require(transferTokenAllowedContractList[msg.sender] || (agent == msg.sender));
        _transfer(_from, _to, _amount);
    }
}
```

- EquipmentDrawContract.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./TokenContract.sol";

contract EquipmentDrawContract {

    uint public immutable COST;
    address private agent;

    TokenContract private tokenContract;

    event DrawResult(address indexed player, uint power, uint item);

    constructor(address _tokenContract, uint _cost) {
        tokenContract = TokenContract(_tokenContract);
        agent = msg.sender;
        COST = _cost;
    }

    function draw(address _player, uint _powerSeed, uint _itemSeed) public {
        require(agent == msg.sender);
        require(tokenContract.balanceOf(_player) >= COST);
        tokenContract.useToken(_player, COST);

        uint power = drawPower(_player, _powerSeed);
        uint item = drawItem(_player, _itemSeed);

        emit DrawResult(_player, power, item);
    }

    function drawPower(address _player, uint _seed) private view returns(uint) {
        return random(_player, _seed)%999+1;
    }

    function drawItem(address _player, uint _seed) private view returns(uint) {
```

```
        return random(_player, _seed)%20+1;
    }

    function random(address _player, uint _seed) private view returns(uint) {
        return uint(keccak256(abi.encode(_player, block.timestamp, _seed)));
    }
}
```

- EquipmentNFTContract.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract EquipmentNFTContract is ERC721{

    address private agent;
    string private baseURI;

    mapping(address=>bool) private transferAllowedContractList;

    event MintResult(address indexed owner, string ipfsCID, string nftURI);

    constructor(string memory _base) ERC721("Equipment", "NFT"){
        agent = msg.sender;
        baseURI = _base;
    }

    function mint(address _owner, string memory _ipfsCID) public {
        require(msg.sender == agent);

        _mint(_owner, _stringToUint(_ipfsCID));

        emit MintResult(_owner, _ipfsCID, nftURI(_ipfsCID));
    }

    function transfer(address _from, address _to, string memory _ipfsCID) public {
        require(transferAllowedContractList[msg.sender] || msg.sender == agent);
        _transfer(_from, _to, _stringToUint(_ipfsCID));
    }

    function nftURI(string memory _ipfsCID) public view virtual returns (string memory){
        return bytes(baseURI).length > 0 ? string.concat(baseURI, _ipfsCID) : "";
    }

    function _baseURI() internal view virtual override returns (string memory) {
        return baseURI;
    }

    function _stringToUint(string memory str) private pure returns (uint) {
        bytes memory b = bytes(str);
        uint number = 0;
        for (uint i = 0; i < b.length; i++) {
            number = number * 10 + (uint(uint8(b[i])) - 48);
        }
```

```
        return number;
    }

    function _uintToString(uint value) private pure returns (string memory) {
        return Strings.toString(value);
    }

    function ownerOf(string memory _ipfsCID) public view virtual returns (address) {
        return _ownerOf(_stringToUint(_ipfsCID));
    }

    function burn(address _owner, string memory _ipfsCID) public {
        require(msg.sender == agent);
        require(_owner == _ownerOf(_stringToUint(_ipfsCID)));
        _burn(_stringToUint(_ipfsCID));
    }

    function registerTransferAllowedContractList(address _contractAddress) public {
        require(msg.sender == agent);
        transferAllowedContractList[_contractAddress] = true;
    }
}
```

- MarketContract.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./TokenContract.sol";
import "./EquipmentNFTContract.sol";

contract MarketContract {

    enum status { PENDING, COMPLETED, CANCELED}

    struct Deal{
        address seller;
        address buyer;
        uint price;
        string ipfsCID;
        status status;
    }

    TokenContract tokenContract;
    EquipmentNFTContract equipmentNFTContract;

    address private agent;

    Deal[] private deals;

    mapping(string => bool) registeredIpfsCIDs;

    event MarketRegisterSellResult(address indexed seller, uint dealNo, uint price, string ip
    event MarketBuyResult(address indexed seller, address buyer, uint price, string ipfsCID),
    event MarketCancelResult(address indexed seller, uint dealNo, uint price, string ipfsCID

    constructor(address _tokenContract, address _equipmentNFTContract) {
```

```
        tokenContract = TokenContract(_tokenContract);
        equipmentNFTContract = EquipmentNFTContract(_equipmentNFTContract);

        agent = msg.sender;
    }

    function sell(address _seller, uint _price, string memory _ipfsCID) public {
        require(agent == msg.sender);
        require(_seller == equipmentNFTContract.ownerOf(_ipfsCID));
        require(!registeredIpfsCIDs[_ipfsCID]);

        deals.push(Deal({
            seller : _seller,
            buyer : address(0),
            price : _price,
            ipfsCID : _ipfsCID,
            status : status.PENDING
        }));

        registeredIpfsCIDs[_ipfsCID] = true;
        emit MarketRegisterSellResult(_seller, deals.length - 1, _price, _ipfsCID);
    }

    function buy(address _buyer, uint dealNo) public {
        require(agent == msg.sender);

        Deal storage deal = deals[dealNo];
        require(deal.status == status.PENDING);
        require(deal.seller != _buyer);
        require(tokenContract.balanceOf(_buyer) >= deal.price);

        tokenContract.transferToken(_buyer, deal.seller, deal.price);
        equipmentNFTContract.transfer(deal.seller, _buyer, deal.ipfsCID);

        deal.buyer = _buyer;
        deal.status = status.COMPLETED;

        registeredIpfsCIDs[deal.ipfsCID] = false;
        emit MarketBuyResult(deal.seller, deal.buyer, deal.price, deal.ipfsCID);
    }

    function cancel(address _seller, uint dealNo) public {
        require(agent == msg.sender);

        Deal storage deal = deals[dealNo];
        require(deal.seller == _seller);
        require(deal.status == status.PENDING);

        deal.status = status.CANCELED;

        registeredIpfsCIDs[deal.ipfsCID] = false;
        emit MarketCancelResult(deal.seller, dealNo, deal.price, deal.ipfsCID);
    }
}
```

- ApplicationContract.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./TokenContract.sol";

contract ApplicationContract {

    enum status { PROGRESS, COMPLETED }

    struct Application{
        address[] winners;
        uint maxWinnerCount;
        uint probabilityNume;
        uint probabilityDeno;
        uint price;
        status status;
    }

    TokenContract tokenContract;

    address private agent;

    Application[] private applications;

    event ApplicationRegisterResult(uint applicationNo, uint maxWinnerCount, uint probability
    event EnterResult(address indexed player, uint nonce, uint target, bool isSuccess, uint r

    constructor(address _tokenContract) {
        tokenContract = TokenContract(_tokenContract);
        agent = msg.sender;
    }

    function register(uint _maxWinnerCount, uint _probabilityNume, uint _probabilityDeno, uir

        require(agent == msg.sender);
        require(_maxWinnerCount > 0);
        require(_probabilityNume < _probabilityDeno);

        applications.push(Application({
            winners : new address[](0),
            maxWinnerCount : _maxWinnerCount,
            probabilityNume : _probabilityNume,
            probabilityDeno : _probabilityDeno,
            price : _price,
            status : status.PROGRESS
        }));

        emit ApplicationRegisterResult(applications.length-1, _maxWinnerCount, _probabilityNu
    }

    function enter(address _player, uint _applicationNo, uint _seed) public {
        require(agent == msg.sender);

        Application storage application = applications[_applicationNo];

        require(!_contains(application.winners, _player));
```

```
        require(application.status == status.PROGRESS);

        require(tokenContract.balanceOf(_player) >= application.price);
        tokenContract.useToken(_player, application.price);

        uint nonce = random(_player, _seed)%application.probabilityDeno;

        bool isSuccess = nonce < application.probabilityNume;

        if(isSuccess){
            application.winners.push(_player);
            if(application.winners.length == application.maxWinnerCount)
                application.status = status.COMPLETED;
        }

        emit EnterResult(_player, nonce, application.probabilityNume, isSuccess, application
    }

    function verifyWinner(address _player, uint _applicationNo) public view returns(bool){
        return _contains(applications[_applicationNo].winners, _player);
    }

    function _contains(address[] memory _array, address _player) public pure returns(bool){
        for(uint i=0; i<_array.length; i++)
            if(_array[i] == _player)
                return true;
        return false;
    }

    function random(address _player, uint _seed) private view returns(uint) {
        return uint(keccak256(abi.encode(_player, block.timestamp, _seed)));
    }
}
```

- Bash Shell

```
npx hardhat compile
```

- scripts/deploy.js

```
const hre = require("hardhat");

async function main() {
  try {
    let tx;

    /* 토큰 컨트랙트 배포 */
    const TokenContract = await ethers.getContractFactory("TokenContract");
    const initialSupply = 1000;
    const TokenContractInstance = await TokenContract.deploy(initialSupply);
    await TokenContractInstance.waitForDeployment();
    console.log("TokenContract deployed to:", await TokenContractInstance.getAddress());

    /* 장비 뽑기 컨트랙트 배포 */
    const drawCost = 10;
```

```
    const EquipmentDrawContract = await ethers.getContractFactory("EquipmentDrawContract");
    const EquipmentDrawContractInstance = await EquipmentDrawContract.deploy(TokenContractIns
    await EquipmentDrawContractInstance.waitForDeployment();
    console.log("EquipmentDrawContract deployed to:", await EquipmentDrawContractInstance.get

    /* 토큰 컨트랙트에 장비 뽑기 컨트랙트 등록 */
    tx = await TokenContractInstance.registerUseTokenAllowedContractList(await EquipmentDraw
    tx.wait();
    console.log("Register UseTokenAllowedContractList Done :: EquipmentDrawContract");

    /* 장비 NFT 컨트랙트 배포 */
    const baseURL = "http://ggul-ipfs.kro.kr:5002/ipfs/"
    const EquipmentNFTContract = await ethers.getContractFactory("EquipmentNFTContract");
    const EquipmentNFTContractInstance = await EquipmentNFTContract.deploy(baseURL);
    await EquipmentNFTContractInstance.waitForDeployment();
    console.log("EquipmentNFTContract deployed to:", await EquipmentNFTContractInstance.getA

    /* 마켓 컨트랙트 배포 */
    const MarketContract = await ethers.getContractFactory("MarketContract");
    const MarketContractInstance = await MarketContract.deploy(TokenContractInstance.getAddre
    await MarketContractInstance.waitForDeployment();
    console.log("MarketContract deployed to:", await MarketContractInstance.getAddress());

    /* 토큰 컨트랙트에 마켓 컨트랙트 등록 */
    tx = await TokenContractInstance.registerTransferTokenAllowedContractList(await MarketCo
    tx.wait();
    console.log("Register TransferTokenAllowedContractList Done :: MarketContractInstance");

    /* NFT 컨트랙트에 마켓 컨트랙트 등록 */
    tx = await EquipmentNFTContractInstance.registerTransferAllowedContractList(await Market
    tx.wait();
    console.log("Register TransferAllowedContractList Done :: MarketContractInstance");

    /* 응모 컨트랙트 배포 */
    const ApplicationContract = await ethers.getContractFactory("ApplicationContract");
    const ApplicationContractInstance = await ApplicationContract.deploy(TokenContractInstan
    await ApplicationContractInstance.waitForDeployment();
    console.log("ApplicationContract deployed to:", await ApplicationContractInstance.getAdd

    /* 응모 컨트랙트에 마켓 컨트랙트 등록 */
    tx = await TokenContractInstance.registerUseTokenAllowedContractList(await ApplicationCo
    tx.wait();
    console.log("Register UseTokenAllowedContractList Done :: ApplicationContractInstance");

  } catch (error) {
    console.error(error);
    process.exit(1);
  }
}
main();
```

- Bash Shell

```
npx hardhat run scripts/deploy.js --network localhost
```

Blockscout 세팅

- Bash Shell

```
git clone https://github.com/blockscout/blockscout.git
cd blockscout/docker-compose/envs
```

- blockscout/docker-compose/envs
  - 내부 env 파일 localhost → 실제 배포 URL로 설정
- Bash Shell

```
cd ..
docker compose -f hardhat-network.yml up -d
```

Contract Verify

- Bash Shell

```
npx hardhat verify --network awesome-chain 0x5FbDB2315678afecb367f032d93F642f64180aa3 1000
npx hardhat verify --network awesome-chain 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512 0x5FbDE
npx hardhat verify --network awesome-chain 0xCf7Ed3AccA5a467e9e704C703E8D87F634fB0Fc9 "http:/
npx hardhat verify --network awesome-chain 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9 0x5FbDE
npx hardhat verify --network awesome-chain 0xa513E6E4b8f2a923D98304ec87F64353C4D5C853 0x5FbDE
```

IPFS 세팅

- Bash Shell

```
sudo wget https://dist.ipfs.tech/go-ipfs/v0.9.1/go-ipfs_v0.9.1_linux-amd64.tar.gz
tar xzvf go-ipfs_v0.4.17_linux-amd64.tar.gz
cd go-ipfs
sudo ./install.sh
ipfs version

cd ~/.ipfs
ipfs init
ipfs daemon
```

- Bash Shell

```
vim ~/.ipfs/config
```

- ~/.ipfs/config

```
"Addresses": {
  "Swarm": [
    "/ip4/0.0.0.0/tcp/4001",
    "/ip6/::/tcp/4001",
    "/ip4/0.0.0.0/udp/4001/quic",
    "/ip6/::/udp/4001/quic"
  ],
  "Announce": [],
  "NoAnnounce": [],
  "API": "/ip4/0.0.0.0/tcp/5001",
  "Gateway": "/ip4/0.0.0.0/tcp/8080"
```

FrontEnd

환경변수 설정 (`.env` 파일 구성)

Vite 프로젝트에 필요한 환경변수 설정 파일 `.env`를 아래와 같이 구성합니다:

```
VITE_API_ENDPOINT=https://ggul3.kro.kr/api
VITE_SOCKET_CONNECT_ENDPOINT=https://www.ggul3.kro.kr/stomp/connection
VITE_BACK_HOST=https://www.ggul3.kro.kr
VITE_FIREBASE_API_KEY=AIzaSyAsbbfLiNV690Z8J3iS_Y-a5IM7aj4e_j0
VITE_FIREBASE_AUTH_DOMAIN=ggul-3.firebaseapp.com
VITE_FIREBASE_PROJECT_ID=ggul-3
VITE_FIREBASE_STORAGE_BUCKET=ggul-3.appspot.com
VITE_FIREBASE_MESSAGING_SENDER_ID=590009959234
VITE_FIREBASE_APP_ID=1:590009959234:web:ac744c45bff243d0003edb
VITE_FIREBASE_VAPID_KEY=BCENu-T5oD7Hb_U9uPbB80wbbhqKkNgPmrXDMda5B1_T9gtY_iT4y7piG1Nl3tlO8rLTN
Of-PlFCacQBuaQMwb8
```

`.env` 파일 경로

`.env` 파일은 EC2 서버에서 다음 경로에 위치해야 합니다:

```
코드 복사
/home/ubuntu/front/.env
```

`.env` 파일을 젠킨스 컨테이너로 복사

젠킨스 빌드 시 `.env` 파일을 젠킨스 컨테이너 안으로 복사해야 합니다. 아래 명령어를 사용하여 `.env` 파일을 복사합니다:

```
docker cp /home/ubuntu/front/.env jenkins:/var/jenkins_home/workspace/GGUL3-Frontend/fronten
d/ggul3/
```

젠킨스 빌드

젠킨스에서 빌드를 실행하여 프론트엔드 배포를 진행합니다. `GGUL3-Frontend` 프로젝트 빌드 시 자동으로 설정된 `.env` 파일을 활용하여 Vite 빌드를 완료합니다.

빌드 완료 후 결과물은 지정된 경로에 배포됩니다.