[포팅매뉴얼]

💡 **배포 순서**

1. be, fe에 도커파일 생성 (Git에 존재)
2. ~~빌드한 각각 be, fe 도커 이미지를 허브에 올림 => 수동배포~~
3. ~~서버에서 도커파일들 pull 받고 컨테이너로 실행시킴 (현재 프론트 3000, 백엔드 8081) => 수동배포~~
4. nginx conf에서 / -> 3000, /api -> 8081 로 리버스 프록시
5. 젠킨스 도커 이미지 pull 받아서 컨테이너로 실행시킴
6. 젠킨스 관리자 계정 생성, 필요한 플러그인 설치
7. 젠킨스 설정에서 gitlab - credentials - GitLab API token add 하고 깃랩에서 생성한 토큰 넣어줌
8. 젠킨스 파이프라인 프로젝트 생성
9. 프로젝트 설정에서 빌드 트리거 - Build when a change is push to GitLab 체크
10. 프로젝트 설정에서 빌드 트리거 - 고급에서 시크릿 토큰 생성
11. 깃랩 webhook 설정으로 가서 젠킨스 프로젝트 설정에 있는 웹훅 url이랑 시크릿 토큰 넣고 트리거 될 브랜치명 작성함
    현재 develop 브랜치 하나 써서 프론트, 백 못 나누고 하나라도 머지되면 둘 다 재배포
12. 젠킨스 Global Tool Configuration 설정에서 jdk, gradle, nodejs 설정해줌
    12-1. 서버에서 젠킨스 컨테이너에 들어감
    12-2. openjdk11 설치
    12-3. env 치면 나오는 JAVA_HOME을 젠킨스 G.T.C 설정에 적어줌 (자동 설치는 8까지 밖에 안돼서)
13. 파이프라인 작성 (git pull -> FE 빌드 -> FE 도커 빌드 -> FE 컨테이너 실행 -> BE 빌드 -> BE 도커 빌드 -> BE 컨테이너 실행)
    13-1. 서버에서 빌드하므로 도커 허브에 올리고 받을 필요없이 바로 컨테이너로 실행 가능 (젠킨스 컨테이너 실행 시 외부 도커에 접근 가능하도록 /var/run/docker.sock 를 바인딩 해두었음)
    13-1. 깃 풀 해오는 step에서 credentialsId 넣을 때 위에서 만든 GitLab API token ID는 인식이 안돼서 크레덴셜에 username&password 도 하나 추가함
    13-2. 프론트 npm build 할때 CI 옵션 false 줘야함

NGINX

`/etc/nginx/sites-available/wildworker.conf`

```
server {
        # 프론트 연결
        location /{
                proxy_pass http://localhost:3000;
        }

        # 백엔드 연결
        location /api {
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
                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_buffer_size          128k;
                proxy_buffers              4 256k;
                proxy_busy_buffers_size    256k;

                proxy_pass http://localhost:8081;
        }

        location /login {
```

```
                proxy_pass http://localhost:8081;
        }

        location ~ ^/(swagger|webjars|configuration|swagger-resources|v2|csrf) {
                proxy_pass http://localhost:8081;

                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;
        }


    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/[도메인 이름]/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/[도메인 이름]/privkey.pem;
}

server {
    # 도메인 이름을 입력
    if ($host = [도메인 이름]) {
        return 301 https://$host$request_uri;
    }

        listen 80;
        server_name j8a304.p.ssafy.io;
    return 404;
}
```

파일 생성 후 심볼릭 링크 생성

```
sudo ln -s /etc/nginx/sites-available/wildworker.conf /etc/nginx/sites-enabled
```

JENKINS

`/my-jenkins/Dockerfile`

```
FROM jenkins/jenkins:lts-jdk11

USER root

# install docker
RUN apt-get update && \
    apt-get -y install apt-transport-https \
        ca-certificates \
        curl \
        gnupg2 \
        zip \
        unzip \
        software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
    apt-get update && \
    apt-get -y install docker-ce
```

`/my-jenkins/docker-compose.yml`

```
version: '3.7'
services:
  jenkins:
    build:
      context: .
    container_name: jenkins
    user: root
    privileged: true
    ports:
      - 8080:8080
      - 50000:50000
    volumes:
```

```
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
```

Pipeline

```
pipeline {
    agent any

    environment {
        GIT_URL = "https://lab.ssafy.com/s08-blockchain-contract-sub2/S08P22A304.git"
    }

    tools {
        gradle 'gradle-7.6.1'
        nodejs 'NodeJS'
    }

    stages {
        stage('Pull Git Branch') {
            steps {
                git url: "${GIT_URL}", branch: "develop", credentialsId: 'GITLAB_AUTH', poll: true, changelog: true
            }
  post {
                failure {
                  echo 'Pull Git Branch failure !'
                }
                success {
                  echo 'Pull Git Branch success !'
                }
          }
        }

        stage('FE Build') {
            steps {
                dir('./FE/front') {
                    sh 'npm install'
                    sh 'CI=false npm run build'
                }
            }
            post {
                failure {
                  echo 'React build failure !'
                }
                success {
                  echo 'React build success !'
                }
            }
        }

        stage('FE Docker Build') {
            steps {
                dir('./FE/front') {
                    sh 'docker build -t jiyoonbyeon/wild-worker-fe .'
                }
            }
            post {
                failure {
                  echo 'Docker build failure !'
                }
                success {
                  echo 'Docker build success !'
                }
            }
        }

        stage('FE Deploy') {
            steps{
                sh 'docker ps -q -f name=^wildworker-fe$ | grep -q . && docker stop wildworker-fe && docker rm wildworker-fe'
                sh 'docker run -d --name wildworker-fe -p 3000:80 jiyoonbyeon/wild-worker-fe'
            }
        }

        stage('BE Build') {
            steps {
                dir('./BE/wild-worker') {
                    sh 'cp -r /var/jenkins_home/resources ./src/main'
                    sh 'chmod +x gradlew'
                    sh 'gradle wrap'
                    sh './gradlew clean bootJar'
                }
            }
            post {
```

```
                failure {
                  echo 'Gradle jar build failure !'
                }
                success {
                  echo 'Gradle jar build success !'
                }
            }
        }

        stage('BE Docker Build') {
            steps {
                dir('./BE/wild-worker') {
                    sh 'docker build -t jiyoonbyeon/wild-worker .'
                }
            }
            post {
                failure {
                  echo 'Docker build failure !'
                }
                success {
                  echo 'Docker build success !'
                }
            }
        }

        stage('BE Deploy') {
            steps{
                sh 'docker ps -q -f name=^wildworker$ | grep -q . && docker stop wildworker && docker rm wildworker'
                sh 'docker run -d --name wildworker --network springboot-mysql-net -p 8081:8080 -v /my-springboot/springboot_home:/roo
            }
        }
    }
}
```

MySQL

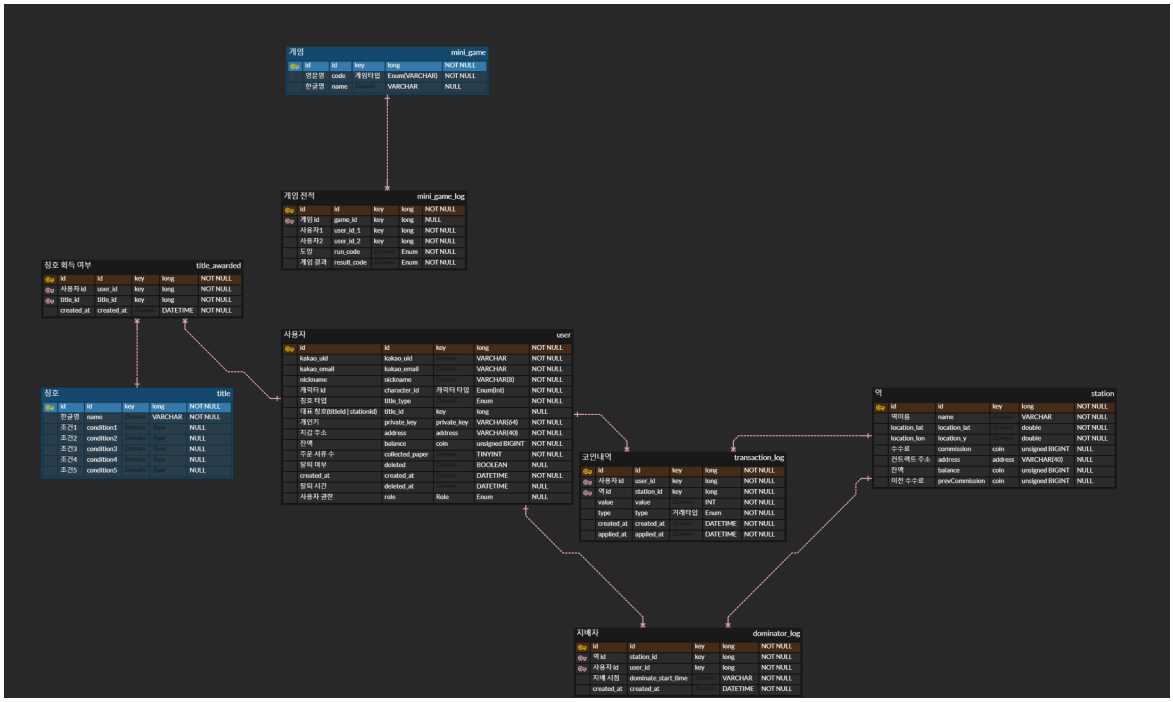
`Docker Image`

```
docker pull mysql:8.0.29
```

`Docker Run`

```
docker run -d -p 3306:3306 --network springboot-mysql-net --cap-add=sys_nice -v /my-mysql/mysql_home:/var/lib/mysql -e MYSQL_ROOT_PASS
```

- ERD



- 스키마: BE/wild-worker/src/main/resources/schema.sql
- 초기 데이터: BE/wild-worker/src/main/resources/data.sql

Redis

`Docker Image`

```
docker pull redis
```

`Docker Run`

```
docker run -d -p 6379:6379 --name redis --network springboot-mysql-net -v /my-redis/redis_home:/data -v /my-redis/redis.conf:/usr/loca
```

Node.JS

- ver. 18.13.0 LTS

`package.json`

```
{
  "name": "front",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.10.6",
    "@emotion/styled": "^11.10.6",
    "@mui/material": "^5.11.12",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.4",
    "net": "^1.0.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-iframe": "^1.8.5",
```

```
    "react-router-dom": "^6.8.2",
    "react-scripts": "5.0.1",
    "socket.io": "^4.6.1",
    "socket.io-client": "^4.6.1",
    "sockjs-client": "^1.6.1",
    "stompjs": "^2.3.3",
    "validator": "^13.9.0",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

```
// 빌드방식

npm install

npm start
```

외부 서비스

- Kakao 로그인
  - https://developers.kakao.com/
  - 앱 등록 및 Web 사이트 도메인, Redirect URI 등록
  - Spring boot - application.yml 설정

```
spring:
  security:
    user:
      name: user
      password: ssafy
    oauth2:
      client:
        registration:
          kakao:
            client-id: {앱키 - REST API 키}
            client-secret: {보안 Client Secret 코드}
            redirect-uri: {server-url}/login/oauth2/code/kakao
            client-authentication-method: POST
            authorization-grant-type: authorization_code
            scope: account_email
            client-name: kakao
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id
```

- React Material UI
  - https://mui.com/material-ui/getting-started/overview/
  - 데스크톱 웹 환경에서 모바일 웹 화면 크기 규정
  - `App.js`

```
...
// line 278 ~ 279
<Container className="app-container" maxWidth="xs">
  <Box sx={{ height: "100vh" }}>
...
```

블록체인

동작 순서

1. 스프링 서버 가동 후 동작해야 함
2. 가나슈 네트워크 가동
3. 스크립트 실행

네트워크

base 이미지

```
FROM ubuntu
FROM node:alpine
WORKDIR /app
RUN npm install --global --quiet npm truffle ganache
```

가나슈 네트워크

```
FROM qww1552/base as ganache

EXPOSE 7545

ENTRYPOINT ["ganache","--account='0x1c55e956560e05843587da7dde1a2d818170ce414cb16e11aae0294144dbfbd6, 99953972490000000000'", "-h", "0
CMD ["--logging.verbose"]
```

실행 명령어

```
docker run --rm -p 7545: 7545 --network [도커 네트워크 이름] --name ganache qww1552/ganache
```

스크립트

package.json

```
{
  "dependencies": {
    "dotenv": "^16.0.3",
    "fs": "^0.0.1-security",
    "mysql2": "^3.2.0",
    "solc": "^0.8.19",
    "web3": "^1.9.0"
  }
}
```

.env

```
BLOCKCHAIN_HOST=ganache
BLOCKCHAIN_PORT=7545
PRIVATE_KEY=0x1c55e956560e05843587da7dde1a2d818170ce414cb16e11aae0294144dbfbd6

DB_HOST= 호스트
DB_USER= DB 계정
DB_PASSWORD= DB 비밀번호
DB_DATABASE= DB 이름
```

컨트랙트

1. contracts/Won.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Won {
    string public name = "Won Token";
    string public symbol = "Won";
    uint256 public totalSupply = 10000000000;
    uint8 public decimals = 0;
    address owner;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event TransferFailed(address indexed _from, address indexed _to, uint256 _value, string reason);

    mapping(address => uint256) public balanceOf;

    modifier onlyOwner() {
        require(msg.sender == owner, "onlyOwner can call this function");
        _;
    }

    constructor() {
        balanceOf[msg.sender] = totalSupply;
        owner = msg.sender;
    }

    function transfer(
        address _from,
        address _to,
        uint256 _amount
    ) public returns (bool success) {
        if (balanceOf[_from] >= _amount) {
            balanceOf[_from] -= _amount;
            balanceOf[_to] += _amount;
            emit Transfer(_from, _to, _amount);
            return true;
        } else {
            emit TransferFailed(_from, _to, _amount, "Not enough balance");
            return false;
        }
    }

    function manualMine(address _toUser, uint _amount) public onlyOwner {
        transfer(msg.sender, _toUser, _amount);
    }

    function sendGameCost(address _toStation, uint256 _amount) public {
        transfer(msg.sender, _toStation, _amount);
    }
}
```

2. contracts/Stations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./Won.sol";

contract Station {
    uint16 public id;
    string public name;
```

```
    Won public won;
    uint256 public investmentAmount;
    address owner;

    address[] investorWallets;
    mapping(address => uint) public investors;

    constructor(Won _won, uint16 _id, string memory _name) {
        won = _won;
        id = _id;
        name = _name;
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "onlyOwner can call this function");
        _;
    }

    function payReward(
        address payable _toUser,
        uint256 _amount
    ) public onlyOwner {
        bool success = won.transfer(msg.sender, _toUser, _amount);
        require(success, "Won transfer failed");
    }

    function autoMine(address _toUser, uint256 _amount) public onlyOwner {
        bool success = won.transfer(msg.sender, _toUser, _amount);
        require(success, "Won transfer failed");
    }

    function resetInvestmentAmount() public onlyOwner {
        investmentAmount = 0;
        uint investorWalletsLength = investorWallets.length;
        for (uint i = 0; i < investorWalletsLength; i++) {
            delete investors[investorWallets[i]];
            investorWallets.pop();
        }
    }

    function countChargeEvery10Min(uint _commission) public onlyOwner {
        for (uint i = 0; i < investorWallets.length; i++) {
            uint userShare = (investors[investorWallets[i]] /
                investmentAmount) * 1000;
            uint chargeByShare = (userShare * _commission) / 1000;
            bool success = won.transfer(msg.sender, investorWallets[i], chargeByShare);
            require(success, "Won transfer failed");
        }
    }

    function recordInvestment(address _sender, uint256 _amount) public {
        bool isIn = false;
        for (uint256 i = 0; i < investorWallets.length; i++) {
            if (investorWallets[i] == _sender) {
                isIn = true;
                break;
            }
        }
        if (isIn) {
            investors[_sender] += _amount;
        } else {
            investorWallets.push(_sender);
            investors[_sender] = _amount;
        }
        investmentAmount += _amount;
        bool success = won.transfer(_sender, owner, _amount);
        require(success, "Won transfer failed");
    }
}
```

컨트랙트 배포 스크립트 /deploy-stations.js

네트워크에 컨트랙트 배포하고 주소를 DB에 저장

```
require('dotenv').config();

const fs = require("fs");
const Web3 = require("web3");
const solc = require("solc");
const mysql = require("mysql2");
```

```
const privateKey=process.env.PRIVATE_KEY;
const rpcURL = `http://${process.env.BLOCKCHAIN_HOST}:${process.env.BLOCKCHAIN_PORT}`;
const web3 = new Web3(rpcURL);

const account = web3.eth.accounts.privateKeyToAccount(privateKey);
web3.eth.accounts.wallet.add(account);

const input = {
  language: "Solidity",
  sources: {
    "Won.sol": {
      content: fs.readFileSync("./contracts/Won.sol", "utf8"),
    },
    "Station.sol": {
      content: fs.readFileSync("./contracts/Station.sol", "utf8"),
    },
  },
  settings: {
    optimizer: {
        enabled: true,
        runs: 1000,
      },
    outputSelection: {
      "*": {
        "*": ["*"],
      },
    },
  },
};

const output = JSON.parse(solc.compile(JSON.stringify(input)));

const wonBytecode =
output.contracts["Won.sol"].Won.evm.bytecode.object;
const wonAbi = output.contracts["Won.sol"].Won.abi;
console.log("won size : " +  Buffer.byteLength(wonBytecode, "utf8"));

const stationBytecode =
output.contracts["Station.sol"].Station.evm.bytecode.object;
const stationAbi = output.contracts["Station.sol"].Station.abi;

const WonContract = new web3.eth.Contract(wonAbi);

(async () => {
  const gasPrice = await web3.eth.getGasPrice();
    console.log("gas price : {}", gasPrice);

  const gasEstimate = await WonContract.deploy({ data: wonBytecode }).estimateGas();

  const wonInstance = await WonContract.deploy({ data: wonBytecode })
    .send({ from: account.address, gas: gasEstimate + Math.floor(gasEstimate * 0.3), gasPrice: gasPrice});

  console.log("Won 컨트랙트 주소:", wonInstance.options.address);
    const wonAddress =  wonInstance.options.address;
  const stationContract = new web3.eth.Contract(stationAbi);

  const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_DATABASE
  });

//   connection.connect(); // MySQL 서버에 연결

  const tableName = "station";

  for (let i = 1; i <= 51; i++) {
    // const query = `SELECT ${fieldName} FROM ${tableName} WHERE id = ${i};`;
    // const stationName = await connection.execute(query);
    const stationName = `Station ${i}`;
    console.log("stationName = " + JSON.stringify(stationName));

    const stationGasEstimate = await stationContract.deploy({ data: stationBytecode, arguments: [wonAddress, i, stationName] })
        .estimateGas();
    console.log("station estimate gas : " + stationGasEstimate);

    const stationInstance = await stationContract.deploy({ data: stationBytecode, arguments: [wonAddress, i, stationName] })
        .send({ from: account.address, gas: stationGasEstimate + Math.floor(stationGasEstimate * 0.3), gasPrice });

    console.log(`스테이션 컨트랙트 ${i} 주소:`, stationInstance.options.address);

    const updateQuery = `
    UPDATE ${tableName}
    SET address = "${stationInstance.options.address}"
    WHERE id=${i};`
```

```
  connection.query(updateQuery, (err, res) => {
    if (err) throw err;
    console.log(`스테이션 컨트랙트 ${i} 주소가 MySQL에 저장되었습니다.`);
  });
  }

  connection.end(); // MySQL 서버 연결 종료
})();
```

컨트랙트 배포 도커 이미지

- 아래 이미지 실행 시 컨트랙트를 네트워크에 배포함
- 서버 재가동 시 최초 한 번만 실행해야 함

```
FROM node:alpine

WORKDIR ./app

COPY contracts/* ./contracts/
COPY package.json ./package.json
COPY deploy/deploy-stations.js ./
COPY .env ./

RUN npm install -g npm@9.6.3
RUN npm i

ENTRYPOINT ["node", "./deploy-stations.js"]
```

실행 명령어

```
docker run --rm --network [도커 네트워크 이름] qww1552/stations
```

Sping boot

- **Application.yml**

```
spring:
  profiles:
    include: oauth, datasource, web3

  devtools:
    livereload:
      enabled: true
    restart:
      enabled: false
  freemarker:
    cache: false

  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    open-in-view: true
    hibernate:
      ddl-auto: none
      use-new-id-generator-mappings: false
    show-sql: false

server:
  servlet:
    context-path: /api/v1

logging:
  level:
    org.hibernate.SQL: error

allowed-origins: http://localhost:3000, http://localhost:8000, http://j8a304.p.ssafy.io, https://j8a304.p.ssafy.io

url:
  client: http://j8a304.p.ssafy.io
  server: http://j8a304.p.ssafy.io
```

- **Application-oauth.yml**

  외부 서비스 - 카카오 로그인 참고

```
spring:
  security:
    user:
      name: user
      password: ssafy
    oauth2:
      client:
        registration:
          kakao:
            client-id: {kakao 앱 REST 키}
            client-secret: {kakao client-secret 코드}
            redirect-uri: ${url.server}${server.servlet.context-path}/login/oauth2/code/kakao
            client-authentication-method: POST
            authorization-grant-type: authorization_code
            scope: account_email
            client-name: kakao
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id
```

- **Application-database.yml**

  최초 빌드 시 sql.init.mode ⇒ always로 빌드, 이후부터는 never로 변경하여 빌드

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://j8a304.p.ssafy.io:3306/wild_worker?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: a304
    password: a304

  ## setting for Spring Session
  main:
    allow-bean-definition-overriding: true
  redis:
    host: j8a304.p.ssafy.io
    port: 6379
    password: a304
  session:
    store-type: redis
    redis:
      flush-mode: on_save         # Sessions flush mode.
      namespace: spring:session   # Namespace for keys used to store sessions.
  sql:
    init:
      mode: never

server:
  servlet:
    session:
      timeout: 300
```

- **Application-web3.yml**

```
web3:
  url: ganache:7545
  chain-id: 1337
  root-private: "0x1c55e956560e05843587da7dde1a2d818170ce414cb16e11aae0294144dbfbd6"
  contract:
    won:
      address: "0x1aDC3a1066F0eB96D7230eab7aAe9Cad94F05BF8"
```

시연 시나리오

- 소셜로그인: 로그인 버튼을 클릭하면 카카오api를 이용해서 소셜로그인
- 전광판: 화면 상단의 전광판에 현재역과 역의 지배자가 보임. 전광판을 클릭하면 현재 보유금액, 지배자의 한마디를 볼수 있음.
- 메뉴: 전광판 아래의 메뉴. 메뉴를 클릭하면 닉네임, 칭호, 정산서가있음. 닉네임을 클릭하면 칭호와 캐릭터타입을 변경가능. 칭호를 클릭하면 현재보유하고 있는 칭호를 볼 수 있음. 칭호 토글을 클릭하면 지배자칭호로 변경가능 지배자 칭호는 자신이 보유하고 있는 가장 인기있는 역의 명칭을 백에서 보내줌. 지배자칭호는 하나만 장착 가능. 칭호를 장착하고 나간뒤 캐릭터를 클릭하면 이름 위에 칭호가 보임. 정산서기능은 최근 시간순으로 정렬되고 각 버튼을 클릭하게 되면 상세보기가 가능함.
- 수동수집: 메인페이지 바닥에 떠다니는 서류를 클릭하면 서류가방에 점수가 올라감. 서류를 30개 모으면 100원을 얻을 수 있음.
- 자동수집: gps기반으로 역에 도착하게 되면 100원을 얻을 수 있음. 오후 3시에 초기화되어 출퇴근시 한번씩 수동수집이 가능함.
  지배자 확성기: 확성기 아이콘을 클릭하게 되면 지배자의 한마디를 작성할 수 있음. 지배자 칭호를 가지고 있는 유저만 가능하고 지배자 칭호를 가진 유저가 확성기를 사용하게 되면 지배자로 되어있는 모든 역에 한마디가 변경됨.
- 투자: 메인 화면 아래에있는 지하철 아이콘을 클릭하면 노선도 페이지로 이동함
- 수수료 정산: 수수료 정산은 10분마다 진행되고 10분마다의 지분투자를 통해 다음 10분간의 지배자가 결정됨. 지분은 누적되고 일주일마다 갱신됨. 10분간 해당역에서 게임이 벌어지면 수수료를 받고 모인 수수료를 해당역에 지분율대로 지급함.
- 현재역, 투자역: 노선도 페이지에서 현재역과 투자역을 볼 수 있음
- 나의 역: 노선도 화면 아래에 나의 역 아이콘을 클릭하게 되면 현재 유저가 투자한 역을 볼 수 있음. 우측상단의 토글을 클릭하면 정렬기준을 변경 할 수 있음.
- 인기 역: 정렬 기준으로 인기역 상위 5개를 띄워 줌. 디폴트는 가장 투자가 많이 된 역.
- 역 투자: 노선도의 역을 클릭하게 되면 투자 페이지로 이동하게 됨. 투자페이지에서는 해당역의 총 투자금, 10분간 모인 수수료 등을 볼 수 있음. 해당 역 지분순위 5등까지 보여줌. 본인의 투자현황은 화면 하단에서 확인할수있음. 투자하기 버튼을 클릭해서 해당역에 투자 가능.
- Pvp: pvp는 유저가 해당 역에 2명이상일시, 메인화면에 있을 시 강제로 자동 매칭됨.
  유저는 결투와 도망이라는 분기를 선택할 수 있고 결투, 상대가 도망, 내가 도망, 내가 도망 실패라는 분기를 얻을 수 있음.
- 게임: 결투를 시작하게 되면 게임이 랜덤으로 정해짐. 게임의 종류는 두가지(계산, 두더지)게임이 있음. 회사원과 지하철이라는 스토리를 입혀서 '회식비 정산하기(계산)', '지하철 빈자리 앉기(두더지)'라는 식으로 게임을 만듦.
  계산게임은 현재 금액에서 랜덤으로 나오는 금액을 더해가는 더하기게임. 총 30초가 주어지고 더 많이 계산에 성공한 유저가 승리
  두더지 게임은 지하철의 9칸중에 비어있는 자리를 클릭해서 빨리 자리를 차지하는 게임 마찬가지로 30초가 주어지고 많이 계산에 성공한 유저가 승리
- 결과페이지: 소켓통신을 위해 잠시 대기시간을 거쳐서 승패를 알 수 있음. 승,패,무승부 세가지 경우가 있음.
- 영수증 페이지: 게임비, 도망비, 환급비를 계산해서 영수증에 보여줌.
- 지배자 강림: 특정역을 소유한 지배자가 그 역에 gps위치상으로 도착할 시 그 역에 위치한 모든 유저들에게 지배자 강림이라는 문구가 한번 뜸.
- 낮 밤 변경: 오후 5시를 기준으로 바뀜. 직장인들의 출퇴근 시간에 맞추어 5시에 낮과 밤으로 변경.
- 역별 다른 캐릭터: 2호선의 (역삼역, 신도림역, 사당역, 홍대입구역, 잠실역)에 다른 캐릭터를 메인에 나오게 함으로써 특정역에 이벤트를 생성하고 현재 자신이 그 역에 도착한 것 같은 느낌을 줌.