



# 포팅메뉴얼

팀 C102	소정현(팀장), 김대현, 노은영, 서은지, 임상은, 전윤희
담당 컨설턴트	박세영 컨설턴트
프로젝트 기간	2022.09.10 ~ 2022.10.21 [자율, 6주]

## 목차

### 목차

#### 1. 프로젝트 기술 스택

이슈 관리

형상 관리

커뮤니케이션

개발 환경

OS

IDE

Database

Server

Backend

Frontend

Deployment

#### 2. 빌드 상세

Backend

Frontend

#### 3. EC2 설정

초기 설정

timezone 설정

Docker 설치

Docker Compose 설치

Nginx 설치

SSL 인증서

/ect/nginx/sites-available/default

Jenkins 컨테이너 실행

#### 4. 배포 특이사항

[Docker-compose.yml](#)

[Backend](#)

[Jenkinsfile](#)

[Dockerfile](#)

[Frontend](#)

[Jenkinsfile](#)

[Dockerfile](#)

[5. 외부 서비스](#)

[카카오](#)

---

# 1. 프로젝트 기술 스택

## 이슈 관리

- Jira

## 형상 관리

- Gitlab

## 커뮤니케이션

- Mattermost
- Notion
- Webex

## 개발 환경

### OS

- Window 10

### IDE

- IntelliJ 7.5.1
- Visual Studio Code
- UI/UX : Figma

### Database

- MySQL 8.0.31
- Redis 7.0.5

## **Server**

- AWS EC2(Ubuntu 20.04.5 LTS)

## **Backend**

- Java 1.8
- Spring Boot 2.7.5
- Spring Data JPA 2.7.5
- Spring Security 2.7.5

## **Frontend**

- HTML5
- Scss 1.55.0
- React 18.2.0
- TypeScript 4.6.4
- react-redux 8.0.4
- react-router-dom 6.4.2
- reduxjs/toolkit 1.8.6
- vite 3.1.0

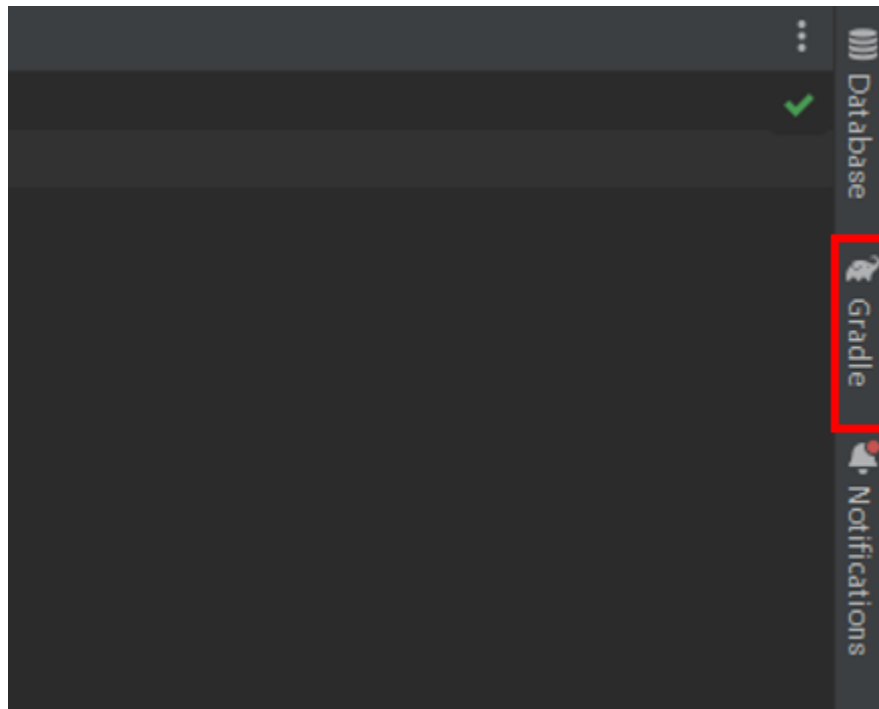
## **Deployment**

- Docker 20.10.21
- Jenkins 2.361.1
- Docker Compose 1.24.1
- Nginx 1.18.0

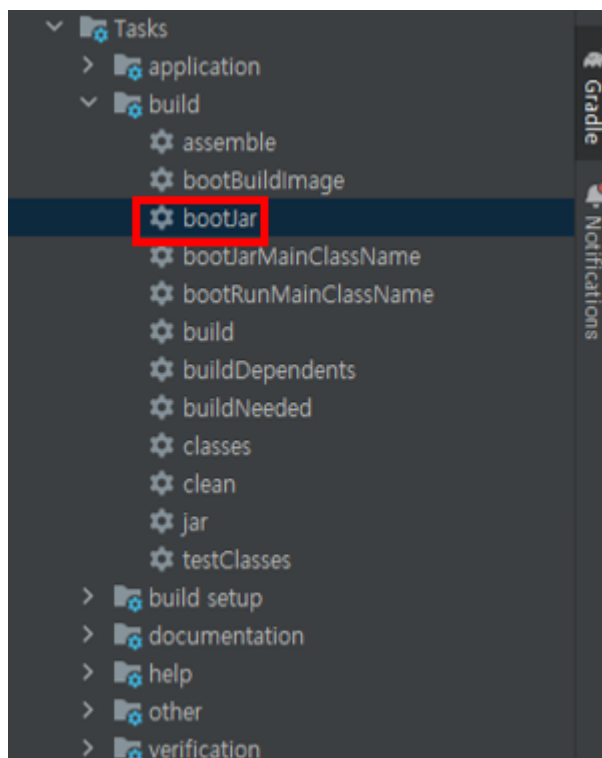
## **2. 빌드 상세**

## Backend

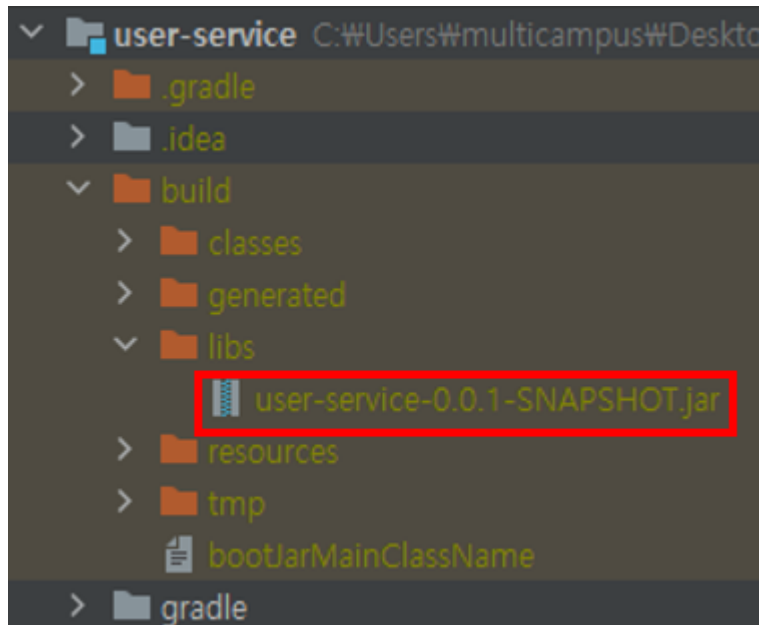
- 화면 우측 Gradle 클릭



- 프로젝트 하위 Tasks > build > bootjar 실행



- build > libs에 생성된 jar 확인



## Frontend

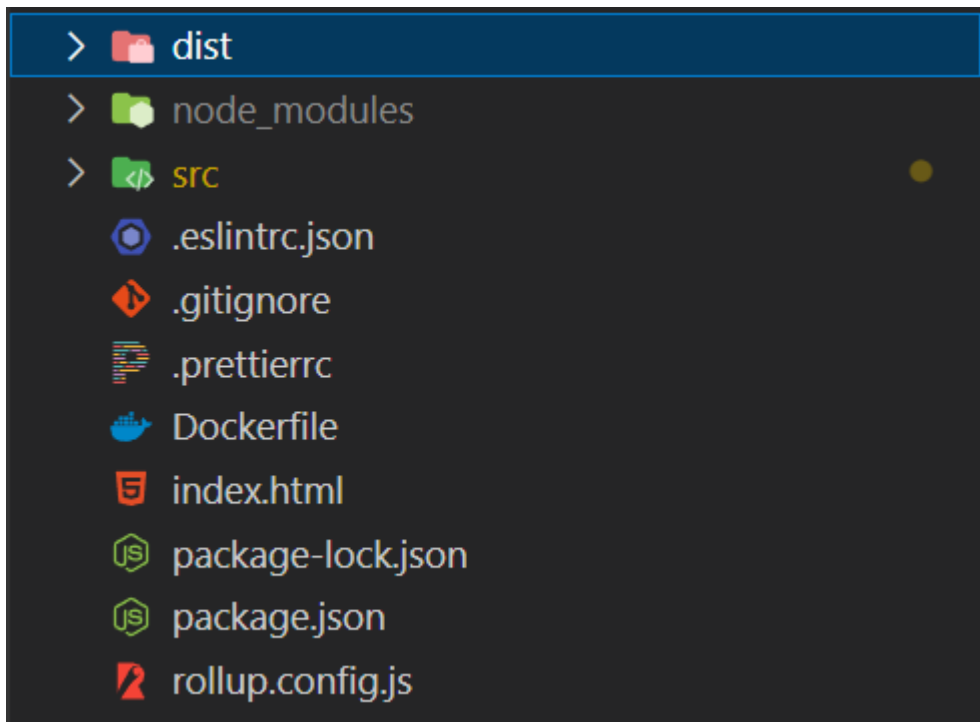
- node\_modules를 위한 install

```
npm install
```

- 빌드하기

```
npm run build
```

- root > dist에 빌드 산출물 존재



## 3. EC2 설정

### 초기 설정

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential
```

### timezone 설정

```
sudo rm /etc/localtime
sudo ln -s /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

### Docker 설치

```
# 설치
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
sudo wget -qO- https://get.docker.com/ | sh
# 서비스 실행 및 부팅 시 자동 실행
sudo systemctl start docker
sudo systemctl enable docker
```

```
# Docker 그룹에 계정 추가
sudo usermod -aG docker ${USER} # ${USER}에 ubuntu
sudo systemctl restart docker
```

## Docker Compose 설치

```
# 설치
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compos
e-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# 권한 설정
sudo chmod +x /usr/local/bin/docker-compose
# 심볼릭 링크 설정
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

## Nginx 설치

```
sudo apt update
sudo apt install nginx
```

## SSL 인증서

이메일, 서비스 약관 동의 후 https 설정 질문에 http 연결을 https로 redirect 시키도록 설정합니다.

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt install python3-certbot-nginx
sudo certbot --nginx -d oh-marking.com k7c102.p.ssafy.io
```

## /etc/nginx/sites-available/default

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;

    server_name _;

    # 모든 http 연결은 https로 리다이렉트
    location / {
        try_files $uri $uri/ =404;
```

```

    }
}

server {
    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;
    server_name oh-marking.com k7c102.p.ssafy.io; # managed by Certbot

    location / {
        charset utf-8;
        proxy_redirect off;
        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 X-NginX-Proxy true;
        client_max_body_size 10M;
        proxy_pass http://oh-marking.com:3000;
    }

    location /login {
        charset utf-8;
        proxy_redirect off;
        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 X-NginX-Proxy true;
        client_max_body_size 10M;
        proxy_pass http://oh-marking.com:8082/login;
    }

    location /logout {
        charset utf-8;
        proxy_redirect off;
        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 X-NginX-Proxy true;
        client_max_body_size 10M;
        proxy_pass http://oh-marking.com:8082/logout;
    }

    location /logic {
        charset utf-8;
        proxy_redirect off;
        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 X-NginX-Proxy true;
        client_max_body_size 10M;
        proxy_pass http://oh-marking.com:8000/logic;
    }

    location /user {
        charset utf-8;

```



```

        proxy_redirect off;
        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 X-NginX-Proxy true;
        client_max_body_size 10M;
        proxy_pass http://oh-marking.com:8000/user;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/k7c102.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/k7c102.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
server {
    if ($host = oh-marking.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    if ($host = k7c102.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80 ;
    listen [::]:80 ;
    server_name oh-marking.com k7c102.p.ssafy.io;
    return 404; # managed by Certbot
}

```

## Jenkins 컨테이너 실행

- /home/jenkins와 컨테이너의 /var/jenkins\_home 볼륨 연결
- /home/ubuntu/.ssh와 컨테이너의 /root/.ssh 볼륨 연결
- /var/run/docker.sock와 컨테이너의 /var/run/docker.sock 연결(컨테이너 바깥의 docker을 jenkins 컨테이너 속 docker에서도 사용할 수 있도록 합니다.)

```

docker pull jenkins/jenkins:lts
docker run -d --name jenkins -p 8080:8080 -p 50000:50000 -v /home/jenkins:/var/jenkins_home -v /home/ubuntu/.ssh:/root/.ssh -v /var/run/docker.sock:/var/run/docker.sock

```

## 4. 배포 특이사항

젠킨스에 프론트엔드와 백엔드 아이템을 각각 파이프라인으로 등록하고, GitLab webhook을 이용해 각 브랜치에 변동사항이 push될 때마다 자동으로 배포되도록 구성합니다. Docker Compose를 이용하여 Docker 컨테이너 관리를 용이하게 하였습니다.

### Docker-compose.yml

```
version: '3'

services:
  redis_user:
    container_name: redis_user
    image: redis
    command: redis-server --requirepass <password> --port 6380
    ports:
      - "6380:6380"
    networks:
      - back_net

  mysql_user:
    container_name: mysql_user
    image: mysql
    environment:
      MYSQL_DATABASE: ohmarking
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: <password>
      TZ: 'Asia/Seoul'
    ports:
      - "3307:3307"
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
    networks:
      - back_net

  mysql_logic:
    container_name: mysql_logic
    image: mysql
    environment:
      MYSQL_DATABASE: ohmarking
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: <password>
      TZ: 'Asia/Seoul'
    ports:
      - "3308:3308"
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
    networks:
      - back_net
```

```

discovery-service:
  container_name: discovery-service
  image: discovery-image
  ports:
    - "8761:8761"
  depends_on:
    - mysql_user
    - mysql_logic
    - redis_user
  networks:
    - back_net

gateway-service:
  container_name: gateway-service
  image: gateway-image
  ports:
    - "8000:8000"
  depends_on:
    - discovery-service
  networks:
    - back_net

user-service:
  container_name: user-service
  image: user-image
  ports:
    - "8082:8082"
  restart: always
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql_user:3307/ohmarking?useSSL=false&serverTimezone=Asia/Seoul&useUnicode=true&characterEncoding=UTF-8
    SPRING_DATASOURCE_USERNAME: "root"
    SPRING_DATASOURCE_PASSWORD: "<password>"
  depends_on:
    - gateway-service
  networks:
    - back_net

business-service:
  container_name: business-service
  image: business-image
  ports:
    - "8083:8083"
  restart: always
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql_logic:3308/ohmarking?useSSL=false&serverTimezone=Asia/Seoul&useUnicode=true&characterEncoding=UTF-8
    SPRING_DATASOURCE_USERNAME: "root"
    SPRING_DATASOURCE_PASSWORD: "<password>"
  depends_on:
    - user-service
  networks:
    - back_net

networks:
  back_net:

```

# Backend

## Jenkinsfile

```
pipeline {
    agent any

    tools {
        gradle 'gradle'
    }

    stages {
        stage('Prepare') {
            steps {
                checkout scm
            }
            post {
                success {
                    echo " prepare stage success"
                }
                failure {
                    echo "prepare stage failed"
                }
            }
        }
        stage('Backend Build') {
            steps {
                dir('discovery-service') {
                    echo "here is discovery-service"
                    // 기존에 실행중이던 컨테이너를 중지하고, 컨테이너와 이미지를 삭제합니다.
                    sh 'docker stop discovery-service || true && docker rm discovery-service || true && docker rmi discovery-image || true'
                    // 프로젝트를 빌드합니다.
                    sh 'gradle clean build -x test -b build.gradle'
                    // 도커 이미지를 빌드합니다.
                    sh 'docker build -t discovery-image .'
                }
                dir('./api-gateway'){
                    echo "here is api-gateway"
                    sh 'docker stop gateway-service || true && docker rm gateway-service || true && docker rmi gateway-image || true'
                    sh 'gradle clean build -x test -b build.gradle'
                    sh 'docker build -t gateway-image .'
                }
                dir('./user-service'){
                    echo "here is user-service"
                    sh 'docker stop user-service || true && docker rm user-service || true && docker rmi user-image || true'
                    sh 'gradle clean build -x test -b build.gradle'
                    sh 'docker build -t user-image .'
                }
                dir('./business'){
                    echo "here is business-service"
```

```

        sh 'docker stop business-service || true && docker rm business-service || true && docker rmi business-image || true'
        sh 'gradle clean build -x test -b build.gradle'
        sh 'docker build -t business-image .'
    }
}
post {
    always {
        echo "build stage complete"
    }
    failure {
        echo "build failed"
    }
    success {
        echo "build success"
    }
}
}

stage('Deploy') {
    steps {
        echo "deploy start"
        // docker-compose.yml이 있는 위치로 이동합니다.
        sh 'cd /'
        // docker-compose를 실행합니다.
        sh 'docker-compose up -d'
    }
    post {
        always {
            echo "deploy stage complete"
        }
        failure {
            echo "deploy failed"
        }
        success {
            echo "deploy success"
        }
    }
}
}
}
}
}

```

## Dockerfile

```

FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=build/libs/<프로젝트 명>-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
EXPOSE <포트번호>
ENTRYPOINT ["java", "-jar", "/app.jar"]

```

## Frontend

## Jenkinsfile

```
pipeline {
  agent any

  stages {
    stage('Prepare') {
      steps {
        checkout scm
      }
      post {
        success {
          echo "prepare stage success"
        }
        failure {
          echo "prepare stage failed"
        }
      }
    }
  }
  stage('Frontend') {
    steps {
      dir('frontend'){
        echo "here is frontend dir"
        // 기존에 실행중이던 컨테이너를 중지 및 삭제합니다.
        sh 'docker stop frontend || true && docker rm frontend || true'
        // 기존 이미지를 삭제합니다.
        sh 'docker rmi frontend-image || true'
        // 도커 이미지를 빌드합니다.
        sh 'docker build -t frontend-image .'
        // 도커 컨테이너를 실행합니다.
        sh 'docker run -d --name frontend -p 3000:3000 frontend-image'
      }
    }
    post {
      always {
        echo "frontend stage complete"
      }
      failure {
        echo "failed"
      }
      success {
        echo "success"
      }
    }
  }
}
```

## Dockerfile

```
FROM node:16.16.0-alpine

WORKDIR "/usr/src/app"
```

```
COPY package.json ./
RUN npm install
COPY ./ ./
CMD ["npm", "run", "dev"]
```

## 5. 외부 서비스

### 카카오

카카오 로그인으로 회원 가입 절차를 없애 서비스의 접근성을 높였습니다.

- 애플리케이션 추가

기본 정보		수정
앱 아이콘		
앱 이름	OhMaRking	
사업자명	김대현	

사용자가 카카오 로그인을 할 때 표시되는 정보입니다. 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다.

- 도메인 등록

## 사이트 도메인

JavaScript SDK, 카카오톡 공유, 카카오톡, 메시지 API 사용시 등록이 필요합니다.

여러개의 도메인은 줄바꿈으로 추가해주세요. 최대 10까지 등록 가능합니다. 추가 등록은 포럼(데브톡)으로 문의주세요.

예시: (O) <https://example.com> (X) <https://www.example.com>

```
https://oh-marking.com
https://k7c102.p.ssafy.io
http://oh-marking.com:8082
http://k7c102.p.ssafy.io:8082
```

## 기본 도메인

기본 도메인은 첫 번째 사이트 도메인으로, 카카오톡 공유와 카카오톡 메시지 API를 통해 발송되는 메시지의 Web 링크 기본값으로 사용됩니다.

```
https://oh-marking.com
```

- 카카오 로그인, OpenID Connect 활성화 및 Redirect URI 설정



### 활성화 설정

상태

ON

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오톡으로 서비스를 시작할 수 있습니다.  
 상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.  
 상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

### OpenID Connect 활성화 설정

상태

ON

카카오 로그인의 확장 기능인 OpenID Connect를 활성화합니다.  
 이 설정을 활성화하면 카카오 로그인 시 사용자 인증 정보가 담긴 ID 토큰을 액세스 토큰과 함께 발급받을 수 있습니다.

### Redirect URI

삭제

수정

Redirect URI	<a href="http://oh-marking.com:8082/login/oauth2/code/kakao">http://oh-marking.com:8082/login/oauth2/code/kakao</a> <a href="http://k7c102.p.ssafy.io:8082/login/oauth2/code/kakao">http://k7c102.p.ssafy.io:8082/login/oauth2/code/kakao</a> <a href="https://oh-marking.com/login/oauth2/code/kakao">https://oh-marking.com/login/oauth2/code/kakao</a> <a href="https://k7c102.p.ssafy.io/login/oauth2/code/kakao">https://k7c102.p.ssafy.io/login/oauth2/code/kakao</a>
--------------	--

- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

• 개인정보 및 접근권한 동의 항목 설정

닉네임	profile_nickname	● 필수 동의	설정
카카오계정(이메일)	account_email	● 선택 동의	설정