

# 포팅메뉴얼

≡ 태그

## 기술스택

[형상 관리](#)  
[이슈 관리](#)  
[커뮤니케이션](#)  
[IDE](#)  
[Server](#)  
[Frontend](#)  
[Backend](#)  
[Database](#)  
[Infra](#)  
[Storage](#)

## 프로젝트 구조

## 백엔드 yml 설정파일

## 사용 포트 목록

## 인텔리제이 시작 초기 세팅

- (1) 빌드 방식 변경
- (2) Lombok 설정 켜기

## 로컬에서 백엔드 서버 켜는 법

- (1) File → Open → S09P12B210 → backend → damda → src → build.gradle
- (2) 메인 시작 버튼
- (3) 서버 실행이 너무 오래 걸릴 경우(보통 5~10초 이내로 켜짐)

## 서버 연동 방법

### MySQL/Docker

[MySQL 설치 및 설정](#)  
[MySQL 외부 포트 바인딩](#)  
[혹은 Docker를 통한 볼륨 마운트](#)  
[Docker 설치](#)

### File upload / workbench

[윈도우 로컬 → EC2 파일 업로딩](#)  
[MySQL Workbench로 파일 넣기](#)

### Docker compose

[컴포즈 설치](#)  
[컴포즈 구문\(MySQL, Springboot, React\)](#)

### Jenkins

[젠킨스 컨테이너 내부에 Docker 설치, 로그 확인](#)  
[이후 과정](#)  
[PipeLine / WebHook](#)

## S3 서버 만들기

## React 프로젝트 만들기

## 카카오톡 연동

## 기술스택



프로젝트에서 사용한 버전 정보 및 스택

### 형상 관리

- GitLab

### 이슈 관리

- Jira

### 커뮤니케이션

- Mattermost

- Notion

## IDE

- IntelliJ CE 2023.1.3
- Visual Studio Code

## Server

- AWS EC2
  - Ubuntu 20.04 LTS
  - Docker 24.0.5

## Frontend

- Node.js 18.16.1 LTS
- React 18.2.0
  - React-canvas-confetti 1.4.0
  - React-cookie 4.1.1
  - react-datepicker 4.16.0
  - react-dom 18.2.0
  - react-hot-toast 2.4.1
  - react-minimal-pie-chart 8.4.0
  - react-modal 3.16.1
  - react-redux 8.1.1
  - react-router-dom 6.14.1
  - react-scripts 5.0.1
  - react-slick 0.29.0
  - react-toastify 9.1.3
  - redux-persist 6.0.0
  - redux-thunk 2.4.2
  - slick-carousel 1.8.1
  - styled-components 6.0.4
  - Redux RTK 1.9.1
- TypeScript 4.9.5
- axios 1.4.0
- email-validator 2.0.4
- event-source-polyfill 1.0.31
- html2canvas 1.4.1
- TailwindCss 3.3.3
  - TailwindCss-styled-component 2.2.0

## Backend

- Java OpenJDK 11
- SpringBoot 2.7.13
- Spring Data JPA
- Spring Security

- Spring mail 0.11.2
- Spring WebFlux
- Spring Web
- Lombok
- Jwt Token 0.11.2
- OAuth2 2.5.4
- JPQL

## Database

- MySQL

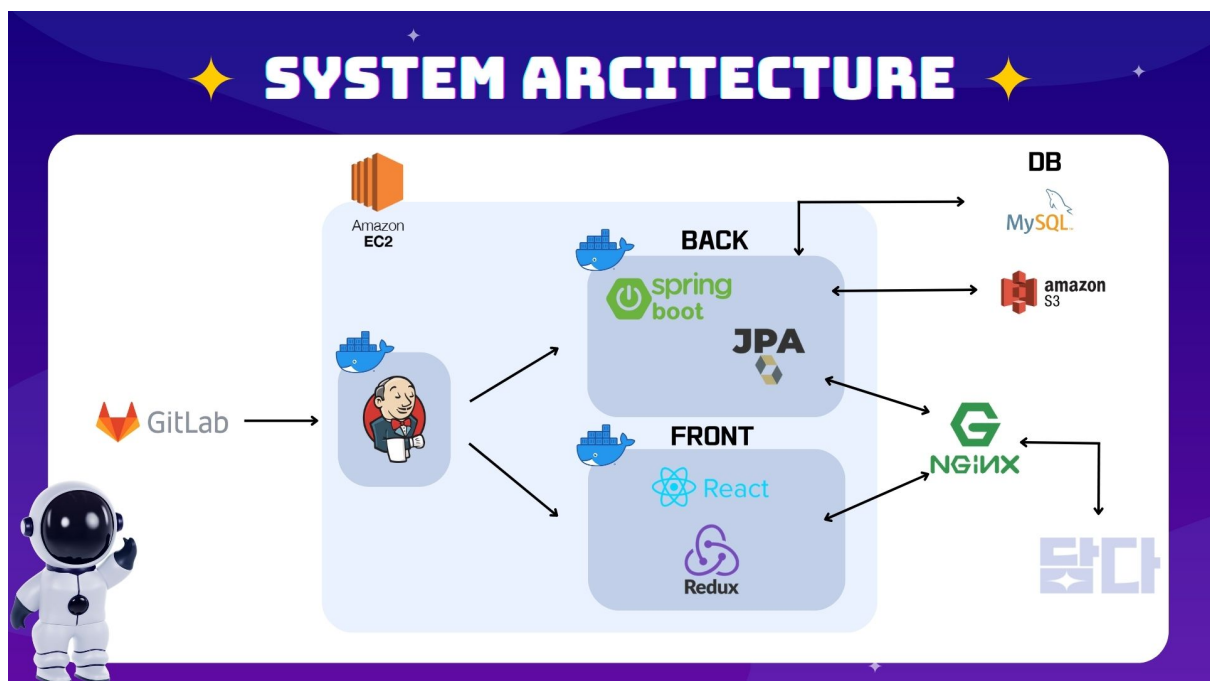
## Infra

- Jenkins 2.401.3
- docker-compose
- SSL
- CertBot

## Storage

- AWS S3

## 프로젝트 구조



## 백엔드 yml 설정파일

▼ application.yml

```
spring:
  mvc:
```

```

    static-path-pattern: /uploads/**
datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: jdbc:mysql://i9b210.p.ssafy.io:7777/damda?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
  username: root
  password: {username 비밀번호}

main:
  allow-bean-definition-overriding: true

servlet:
  multipart:
    max-file-size: 1000MB
    max-request-size: 1000MB

jpa:
  open-in-view: false
  hibernate:
    ddl-auto: none
    naming:
      physical-strategy: org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
    use-new-id-generator-mappings: false
  show-sql: false
  properties:
    hibernate.format_sql: false
    dialect: org.hibernate.dialect.MySQL8InnoDBDialect

#Personal Information Management(.ignore)
profiles:
  include: secret

AdminMail:
  id: "{메일아이디}"
  password: "{비밀번호}"

mail:
  smtp:
    auth: true
    starttls:
      required: true
      enable: true
    socketFactory:
      class: javax.net.ssl.SSLSocketFactory
      fallback: false
      port: 465
      port: 465

security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: {클라이언트 아이디}
          client-secret: {클라이언트 시크릿키}
          scope:
            - account_email
            - profile_nickname
            - profile_image
          authorization-grant-type: authorization_code
          redirect-uri: https://i9b210.p.ssafy.io/login/oauth2/code/kakao
          client-name: Kakao
          client-authentication-method: POST

      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

  web:
    resources:
      static-locations: file:/C:/new/

logging:
  level:
    com:
      amazonaws:
        util:
          EC2MetadataUtils: error
      org.hibernate.SQL: error

```

▼ application-secret.yml

```

weather:
  # 공공데이터 API KEY
  apikey: {API KEY}
  # jwt 키값 설정(어느 정도 길이를 가져야 함)
  jwt:
    secret: {토큰 KEY}

cloud:
  aws:
    s3:
      bucket: damda
    stack:
      auto: false
    region:
      static: ap-northeast-2
      auto: false
    credentials:
      accessKey: {s3 액세스 키}
      secretKey: {s3 시크릿 키}

```

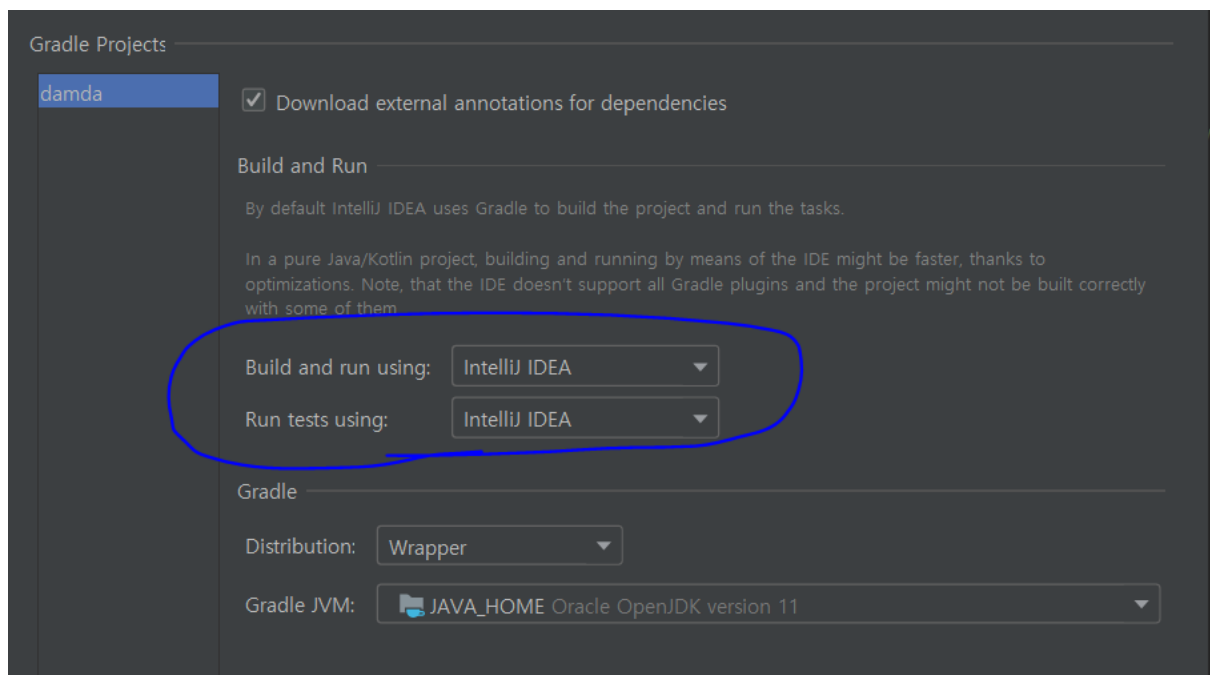
## 사용 포트 목록

이름	내부 포트	외부 포트
React	80	8160
Spring Boot	8080	1104
Jenkins	8080	9091
MySQL	3306	7777
http	80	
https	443	

## 인텔리제이 시작 초기 세팅

### (1) 빌드 방식 변경

File → Settings → Build, Execution, Deployment → Build Tools → Gradle

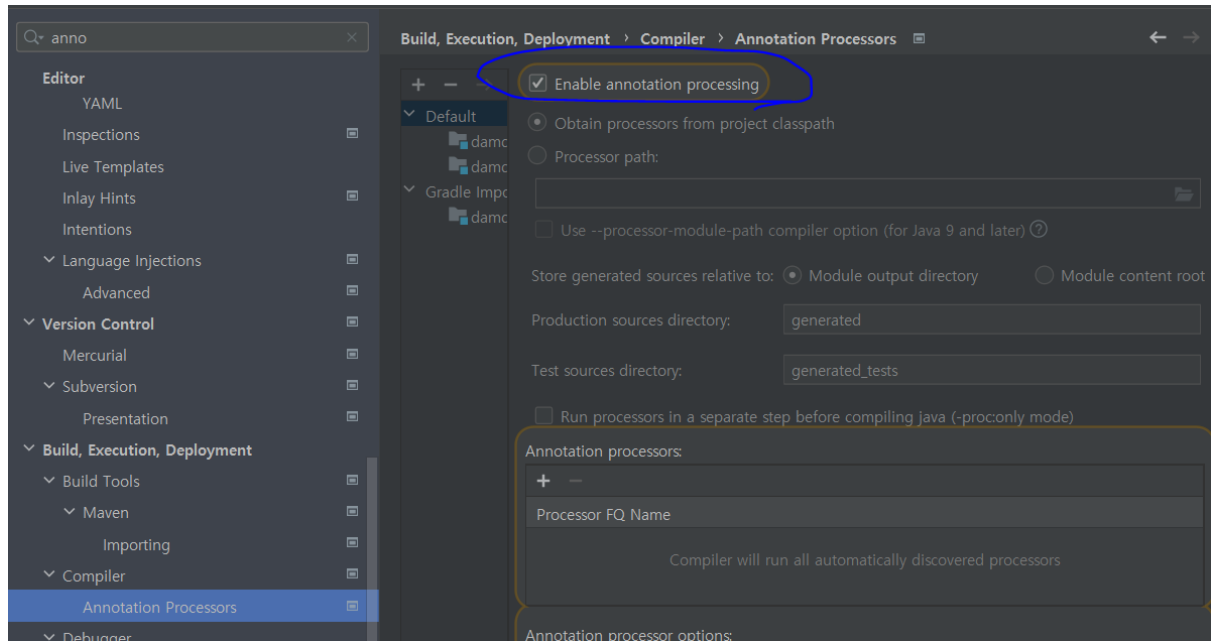


해당 두 개를 Gradle에서 IntelliJ IDEA로 바꿔준다.

- 빌드 방식을 변경. IntelliJ IDEA(중분 빌드)가 자체 인텔리제이 빌드라서 더 빠른 빌드 가능.
- 중분 빌드 방식. → 변경된 부분만 빌드하는 방식

## (2) Lombok 설정 켜기

File → Settings → Build, Execution, Deployment → Compiler → Annotation Processors



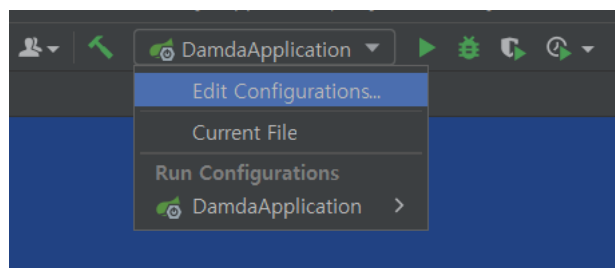
- Lombok을 사용하기 위해서 annotation을 활성화해준다.

## 로컬에서 백엔드 서버 켜는 법

(1) File → Open → S09P12B210 → backend → damda → src → build.gradle

열면 우측 하단에 실행을 위해서 라이브러리 다운로드 자동 진행.

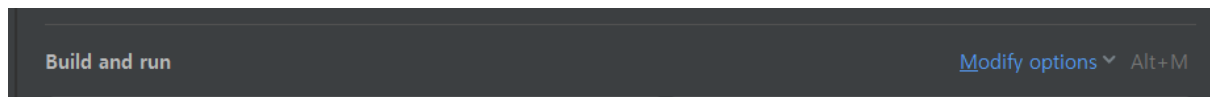
## (2) 메인 시작 버튼



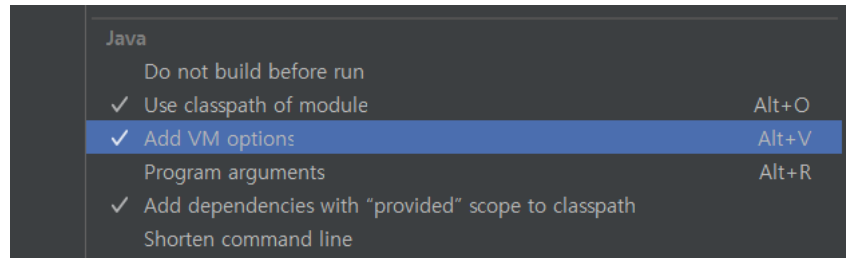
우측 상단에 DamdaApplication으로 두고 오른쪽에 시작 버튼 클릭.

## (3) 서버 실행이 너무 오래 걸릴 경우(보통 5~10초 이내로 켜짐)

Run → Edit Configurations



Modify options 클릭



Add VM options 클릭



-Dcom.amazonaws.sdk.disableEc2Metadata=true

집어넣고 저장.

- AWS SDK에게 EC2 메타데이터 조회를 비활성화 함.(서버 실행 시간이 빨라짐)

## 서버 연동 방법

### MySQL/Docker

#### ▼ MySQL 설치 및 설정

설치 생략

루트 계정 비번 변경

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '{비밀번호}';
```

#### 1. 계정 생성

```
CREATE USER 'b210'@'localhost' IDENTIFIED WITH mysql_native_password BY '{비밀번호}';
```

#### 2. 권한 부여

```
GRANT ALL PRIVILEGES ON database_name.* TO 'b210'@'localhost';
```

#### 3. 변경 사항 적용

```
FLUSH PRIVILEGES;
```

## ▼ MySQL 외부 포트 바인딩

### 1. 폴더 설정 이동

```
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

### 2. 포트 및 바인딩 주소 설정 0.0.0.0은 보안상 위험하다. 조심할 것.

```
[mysqld]
#
# * Basic Settings
#
user                = mysql
# pid-file           = /var/run/mysqld/mysqld.pid
# socket             = /var/run/mysqld/mysqld.sock
port                = 5050
# datadir            = /var/lib/mysql

# If MySQL is running as a replication slave, this should be
# changed. Ref https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_tmpdir
# tmpdir             = /tmp
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address         = 0.0.0.0
mysqlx-bind-address  = 127.0.0.1
#
# * Fine Tuning
#
key_buffer_size      = 16M
# max_allowed_packet = 64M
# thread_stack        = 256K
```

### 1.

## ▼ 혹은 Docker를 통한 볼륨 마운트

볼륨 관련 참조 링크 : <https://rondeveloper.tistory.com/92>

볼륨 생성 : mysql\_volume으로 만들었다.

```
$ docker volume create [volume 이름]
$ docker volume create mysql_volume
```

```
docker run -d \
  --name mysql \
  -e MYSQL_ROOT_PASSWORD={비밀번호} \
  -v mysql_volume:/var/lib/mysql \
  -p 7777:3306 \
  mysql:latest
```

`docker exec -it mysql(이름) mysql -u root -p` 를 통해 내부 설정으로 접속할 수 있다.

`docker exec -it [컨테이너 이름 또는 ID] /bin/bash`

`docker exec -it mysql /bin/bash`

`docker exec -it mysql mysql --password={비밀번호}`

## ▼ Docker 설치



해당 블로그 참고

<https://jeongil.tistory.com/1968>

도커 실행

```
sudo systemctl start docker
```

자동 실행되도록 설정

```
sudo systemctl enable docker
```

## File upload / workbench

### ▼ 윈도우 로컬 → EC2 파일 업로딩

```
curl -s http://169.254.169.254/latest/meta-data/public-ipv4
```

 를 통해 공개 도메인 받아오기

참조 : [https://velog.io/@\\_koiil/EC2로-파일-업다운로드](https://velog.io/@_koiil/EC2로-파일-업다운로드)

```
scp -i "I9B210T.pem" damda.sql ubuntu@13.125.238.163 :/home/ubuntu/win2ubuntu
```

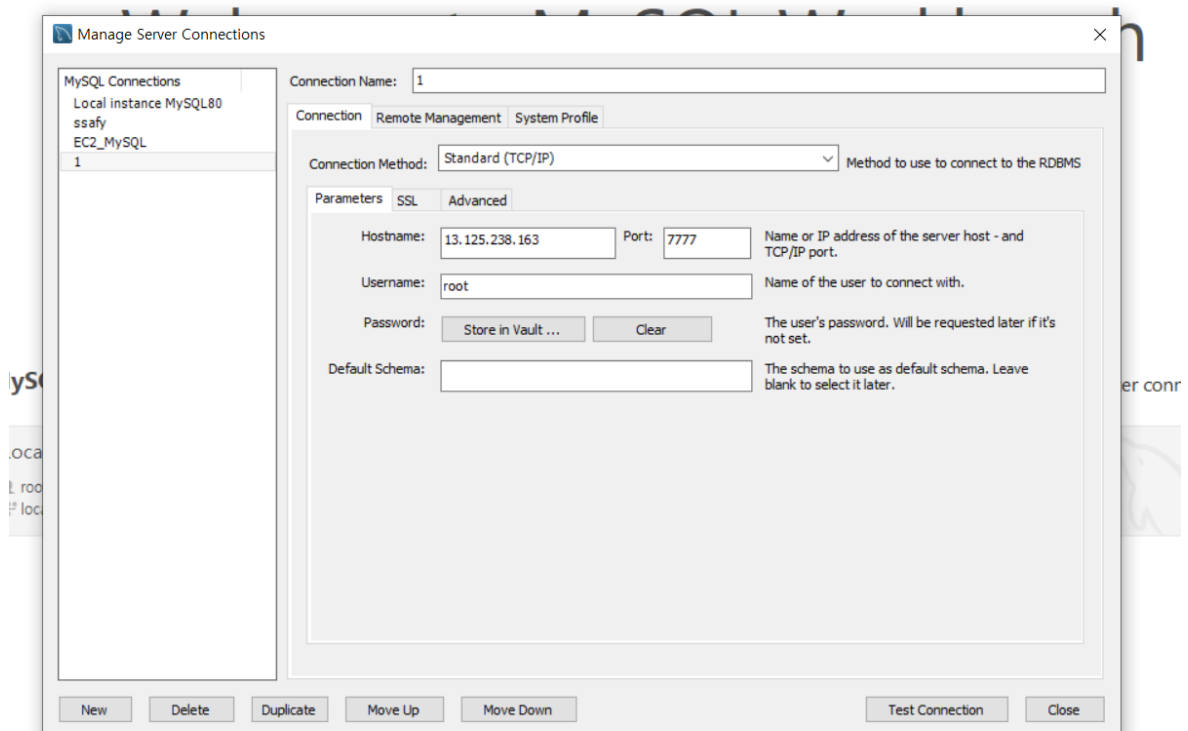
을 통해 파일을 업로드 할 수 있었다.

```
docker cp /path/to/your/localfile.sql mysql_container:/docker-entrypoint-initdb.d/
```

근데 docker cp해서 넣으니까 다 안 되던데

왜지

### ▼ MySQL Workbench로 파일 넣기



여기 연결해서 sql 파일 바꿔줬다.

## Docker compose

### ▼ 컴포즈 설치

도커 컴포즈를 별도로 설치해야 합니다. 아래 명령을 사용하여 도커 컴포즈를 설치할 수 있습니다.

최신 버전 설치:

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose 해당 경로에 권한 추가
```

특정 버전 설치:

```
sudo curl -L "https://github.com/docker/compose/releases/download/{원하는_버전}/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

**{원하는\_버전}** 자리에 원하는 도커 컴포즈 버전을 입력하세요. 예를 들어, **1.29.2** 버전을 설치하려면 해당 버전을 입력하면 됩니다.

1. 설치가 완료되었는지 확인하기 위해 다음 명령을 실행하여 도커 컴포즈 버전을 확인합니다.

```
docker-compose --version
```

도커 컴포즈 설치가 완료되면, 도커 컴포즈 파일(**docker-compose.yml**)을 작성하여 여러 컨테이너를 정의하고 실행할 수 있습니다.

### ▼ 컴포즈 구문(MySQL, Springboot, React)

```
version: '3'

services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - jenkins:/var/jenkins_home
    ports:
      - "9091:8080"
    user: root
    restart: always
    ///////////////이거 젠킨스 항상 리스타트 안되게 빼놔음. 여기는 예시용으로 씬////////

  springboot:
    image: changsigi/damda-springboot:latest
    container_name: springboot
    ports:
      - "8081:8080"
    restart: always

  react:
    image: changsigi/damda-react:latest
    container_name: react
    working_dir: /app
    volumes:
      - ./frontend:/app
    ports:
      - "3000:80"
    restart: always
```

현재 컴포즈 파일에서 사용하는 volumes 항목은

호스트의 /var/run/docker.sock을 컨테이너의 /var/run/docker.sock으로 연결하고,

호스트의 /jenkins를 컨테이너의 /var/jenkins\_home으로 연결

- ▼ 해당 public의 정적 파일을 /app/public으로 저장한다면 어떤 이점이?

`./public` 디렉토리의 파일들이 React 컨테이너 내부의 `/app/public` 디렉토리에 마운트되어 React 애플리케이션에서 접근할 수 있게 된다.

React 애플리케이션의 정적 파일을 `/app/public` 디렉토리로 저장하는 것은 다음과 같은 이점을 가질 수 있습니다:

1. **정적 파일의 분리:** `/app/public` 디렉토리는 React 애플리케이션의 정적 파일을 저장하는 공용 공간이 됩니다. 이렇게 분리된 정적 파일은 컨테이너 외부에서도 접근 가능하므로 웹 서버가 아닌 다른 방법으로도 정적 파일에 접근하고 사용할 수 있습니다.
2. **Hot Module Replacement (HMR) 등의 개발 기능 지원:** React 개발 환경에서는 정적 파일의 변경 시 자동으로 페이지를 리로드하지 않고 변경된 내용을 실시간으로 반영하는 Hot Module Replacement 기능이 있습니다. `/app/public` 디렉토리를 이용하여 정적 파일을 외부에서 변경하면 React 애플리케이션이 해당 변경사항을 실시간으로 반영하는데 유용합니다.
3. **데이터 바인딩 등의 동적 기능 지원:** 정적 파일로부터 동적으로 데이터를 불러와서 애플리케이션에 바인딩하는 경우가 있습니다. 이러한 경우, `/app/public` 디렉토리에 저장된 정적 파일로부터 데이터를 불러와서 애플리케이션에 사용할 수 있습니다.
4. **실행 가능한 환경:** React 애플리케이션을 빌드할 때 정적 파일을 생성하면, 이를 독립적인 웹 서버(예: Nginx)에 배포하여 실행 가능한 환경을 구성할 수 있습니다. 이렇게 하면 React 애플리케이션을 특정 웹 서버에 종속되지 않고 독립적으로 배포하고 실행할 수 있습니다.
5. **웹 서버가 아닌 컨테이너의 파일 시스템:** 정적 파일을 컨테이너의 `/app/public` 디렉토리에 저장하면 웹 서버가 아닌 다른 컨테이너에서도 해당 정적 파일에 접근할 수 있습니다. 이렇게 다른 컨테이너에서도 정적 파일을 공유할 수 있으며, 웹 서버와 백엔드 서버 등 여러 서비스를 별도로 구성할 때 편리합니다.

```
docker-compose -f docker_compose_folder/docker_compose.yml up -d
```

## Jenkins

참고 파일 :

[All\\_about\\_배포\\_김정윤실습코치.pdf](#)

### ▼ 젠킨스 컨테이너 내부에 Docker 설치, 로그 확인

젠킨스 컨테이너 내부에 접속

```
sudo docker exec -it jenkins /bin/bash
```

Docker 설치

```
apt-get update
apt-get install ca-certificates curl gnupg lsb-release
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) docker-ce" > /etc/apt/sources.list.d/docker.list
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-compose
```

`docker logs jenkins` 를 통해 로그를 확인할 수 있다.

### ▼ 이후 과정

1. 계정 생성한다.

b210 / {비밀번호}으로 생성하였음

▼ url 설정

damda로 추가하였음

Getting Started

# Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.401.3 Not now Save and Finish

add credentials : pdf [참고](#)

## ▼ PipeLine / WebHook

### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://i9b210.p.ssafy.io:9091/damda/project/webhook> ?
 

Enabled GitLab triggers
 
  - ☒ Push Events ?
  - ☐ Push Events in case of branch delete ?
  - ☒ Opened Merge Request Events ?
  - ☐ Build only if new commits were pushed to Merge Request ?
  - ☐ Accepted Merge Request Events ?
  - ☐ Closed Merge Request Events ?

Build when a change is pushed to GitLab. GitLab webhook URL: <http://i9b210.p.ssafy.io:9091/damda/project/webhook>

secret key 생성 : Webhook 사용 위함

{시크릿키}

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

Generate

변경 : {시크릿키}

▼ credentials

<https://dejavuqa.tistory.com/143> 참조

## S3 서버 만들기

▼ 서버 만드는 방법

S3 서버는 프리티어의 경우 5GB까지 사용이 가능하다.

**객체(object)**

파일과 파일정보로 구성된 저장단위로 그냥 파일이라 생각하면 된다.

**버킷(Bucket)**

다수의 객체를 관리하는 컨테이너로 파일시스템이라 보면된다.

AWS 콘솔 → S3 → 버킷 → 버킷 만들기

버킷 이름 설정하고 모든 액세스 차단 설정을 해제한다.

그다음 AWS에서 IAM 검색 후 들어가서

IAM → 액세스 관리 → 사용자 → 사용자 추가

사용자 이름을 입력하고 직접 정책 연결을 선택한다!

그다음 **AmazonS3FullAccess** 검색해서 선택하고 다음을 누른다.

그리고 사용자 생성을 누르면 된다!!

사용자를 생성했으면 사용자 이름을 선택 후

보안 자격 증명 → 액세스 키 만들기

액세스 키를 사용하여 AWS CLI, AWS Tools for PowerShell, AWS SDK 또는 직접 AWS API 호출을 통해 AWS에 프로그래밍 방식 호출을 전송합니다. 한 번에 최대 두 개의 액세스 키(활성 또는 비활성)를 가질 수 있습니다. [자세히 알아보기](#)

엑세스 키와 같은 장기 자격 증명을 사용하지 않는 것이 모범 사례입니다. 대신 단기 자격 증명을 제공하는 도구를 사용하세요. [자세히 알아보기](#)

14

```

import org.springframework.core.io.UrlResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class S3UploadService {

    private final AmazonS3 amazonS3;

    @Value("${cloud.aws.s3.bucket}")
    private String bucket;

    public String saveFile(MultipartFile multipartFile) throws IOException {
        String originalFilename = multipartFile.getOriginalFilename();

        String extension = originalFilename.substring(originalFilename.lastIndexOf(".")); // 파일 확장자
        String randomName = UUID.randomUUID().toString(); // 랜덤한 문자열 생성
        String newFilename = randomName + extension; // 랜덤한 문자열과 확장자를 합쳐서 새 파일명 생성

        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentLength(multipartFile.getSize());
        metadata.setContentType(multipartFile.getContentType());

        amazonS3.putObject(bucket, newFilename, multipartFile.getInputStream(), metadata);
        return amazonS3.getUrl(bucket, newFilename).toString();
    }

    public ResponseEntity<UrlResource> downloadImage(String originalFilename) {
        UrlResource urlResource = new UrlResource(amazonS3.getUrl(bucket, originalFilename));

        String contentDisposition = "attachment; filename=\"" + originalFilename + "\"";

        // header에 CONTENT_DISPOSITION 설정을 통해 클릭 시 다운로드 진행
        return ResponseEntity.ok()
            .header(HttpHeaders.CONTENT_DISPOSITION, contentDisposition)
            .body(urlResource);
    }
}

```

saveFile부터 살펴보면

파일을 받아서 원본이름, 확장자, 랜덤이름을 이용해서 새로운 이름을 만든다.

그리고 메타데이터 설정을 하고 s3서버에 업로드를 진행한다.

리턴값으로 해당 s3서버의 주소를 넘겨준다!!!

downloadImage는

아직 잘 모르겠다.

## application.yml

```

cloud:
  aws:
    s3:
      bucket: {버킷}
    stack:
      auto: false
    region:
      static: ap-northeast-2
      auto: false
    credentials:
      accessKey: {엑세스키}
      secretKey: {시크릿키}

```

yml은 이렇게 설정해두었다.

이렇게 코드는 완료시키고 이제 S3 설정을 해야한다.

기존에는 읽기가 불가능하게 설정이 돼있다.

## 정책 생성기 이용

버킷 → 권한 → 버킷 정책 → 편집 → 정책 생성기

### Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

### Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a [description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services ('\*')

Use multiple statements to add permissions for more than one service.

Actions 1 Action(s) Selected ☐ All Actions ('\*')

Amazon Resource Name (ARN)

ARN should follow the following format: arn:aws:s3:::{BucketName}/{KeyName}.  
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

Add Statement Resource field is not valid. You must enter a valid ARN.

### Step 3: Generate Policy

A policy is a document (written in the [Access Policy Language](#)) that acts as a container for one or more statements.

**Add one or more statements above to generate a policy.**

principal은 \*을 넣어주고

step1은 s3 bucket policy로 바꿔준다.

Actions는 getObject로 만들어준다.

arn은 버킷 → 속성 → 버킷개요에 있는 arn을 복사해준다.

그대로 생성하게 되면

기존에는 "Resource": "arn:aws:s3:::damda" 이렇게 정책생성기가 만들었었는데

뒤에 /\* 이거 붙여줘야 된다.

```
{
  "Version": "2012-10-17",
  "Id": "Policy1690860277051",
  "Statement": [
    {
      "Sid": "Stmt1690860274588",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::damda/*"
    }
  ]
}
```



```
}  
}
```

그대로 실행하면 아마 스프링부트에서 시간초과가 날 수 있다. aws를 연결하면 뭐가 문제가 생긴다고 했다.

```
@SpringBootApplication(exclude = {  
    SecurityAutoConfiguration.class,  
    org.springframework.cloud.aws.autoconfigure.context.ContextInstanceDataAutoConfiguration.class,  
    org.springframework.cloud.aws.autoconfigure.context.ContextStackAutoConfiguration.class,  
    org.springframework.cloud.aws.autoconfigure.context.ContextRegionProviderAutoConfiguration.class  
})
```

가장 메인 스프링부트 Application에서 exclude를 추가해줘야한다.

이거 해결하려면

Run → Edit Configurations 들어가서

Modify options → Add VM options

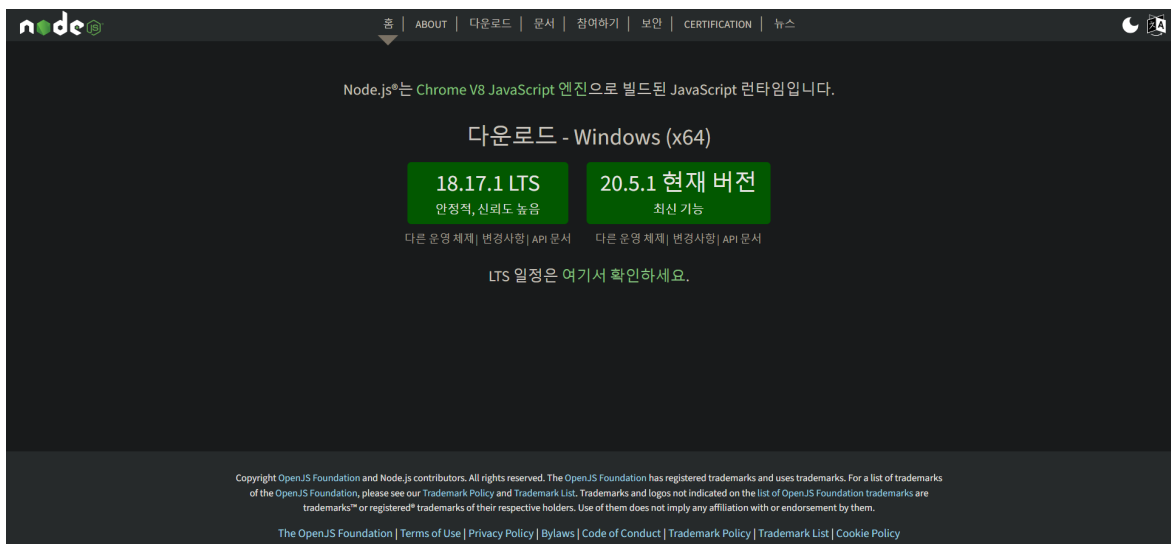
```
-Dcom.amazonaws.sdk.disableEc2Metadata=true
```

이거 추가해준다.

## React 프로젝트 만들기

▼ 프로젝트 처음 시작, 만들기

1. 먼저 node.js를 깔아야 합니다.




위에서 LTS 버전으로 다운 받습니다. 설치 파일을 다운 받고 특별한 설정 없이 그대로 설치 진행하면 됩니다.


2. vscode에서 npm 명령어로 생성하기

이 명령어로 node.js를 설치한 뒤에 npm 명령어로 react 프로젝트를 만들 수 있습니다.

반드시 처음 node.js를 설치했다면 아래 명령어를 커맨드창에서 실행해야 합니다.


 `npm install -g create-react-app`


이후, 아래 명령어를 입력하면 현재 경로에서 react 프로젝트를 생성합니다.

 `npx create-react-app <프로젝트이름>`

### 3. react 실행해보기

react를 실행하는 방법은 여러가지가 있습니다.

 `npm start` : 개발자 모드

 `npm run build` ⇒ `serve -s build` : 빌드 후 서버 실행

우리는 개발자이므로, 개발자 모드로 실행합니다.


그럼 저 명령어를 어디에서 실행해야 하느냐? react 프로젝트에 보면 `package.json`이 있는데, 해당 파일이 있는 경로 위치에서 명령어를 입력하면 됩니다.

react는 기본포트가 3000으로 되어 있고, 명령어 실행시 `localhost:3000`으로 웹페이지가 자동으로 뜹니다.

++ react 프로젝트를 생성하면 해당 프로젝트 폴더에 `.git`이 자동으로 생깁니다. 만약 이미 git remote를 해두었다면 새롭게 만들어진 `.git`과 `.gitignore`는 제거해주세요.

#### ▼ Damda-React 프로젝트 설치 및 시작

frontend/client로 경로를 들어가서 아래 명령어를 입력합니다.

 `npm install`

그러면 `package.json`에 등록되어있는 라이브러리를 자동으로 다운하고 설치해줍니다.

이후, 개발자모드 시작(`npm start`)하면 로컬로 앱을 실행할 수 있습니다.

## 카카오톡 연동

#### ▼ 카카오톡 API 시작하기

1. 메인화면에서 로그인을 진행한 후 내 애플리케이션 클릭후 애플리케이션 추가!

전체 애플리케이션 (3)

애플리케이션 이름

+

애플리케이션 추가하기

2. 앱 아이콘과 앱이름 사업자명을 작성후 동의

전체 애플리케이션

+

애플리케이션 추가하기

다

da

ID

APP

tri

ID

APP

tri

ID

애플리케이션 추가하기

앱 아이콘

이미지 업로드

파일 선택

JPG, GIF, PNG  
권장 사이즈 128px, 최대 250KB

앱 이름

내 애플리케이션 이름

사업자명

사업자 정보와 동일한 이름

- 입력된 정보는 사용자가 카카오 로그인을 할 때 표시됩니다.
- 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다.

☒

[서비스 이용이 제한되는 카테고리](#), [금지된 내용](#), [금지된 행동](#) 관련 운영정책을 위반하지 않는 앱입니다.

취소

저장

3. REST API 키 (백엔드 - 로그인 ) 와 JavaScript 키(프론트엔드 - 공유하기)를 사용할 예정 !!

플랫폼 설정하기

앱 키

네이티브 앱 키	
REST API 키	
JavaScript 키	
Admin 키	

플랫폼

설정된 플랫폼 정보가 없습니다. [플랫폼 설정하기](#)

4. 사용하려는 URL (백) 주소와 (프론트) 주소를 Web플랫폼 등록

포팅메뉴얼

19

Web 플랫폼 등록

사이트 도메인

JavaScript SDK, 카카오톡 공유, 카카오맵, 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`

http://localhost:8080

기본 도메인

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

http://localhost:8080

취소

저장

## 5. 카카오 로그인 활성화

다다

DamdaTest

ID 955633

OWNER

Web

성공적으로 반영되었습니다.

카카오 로그인

ON

동의 화면 미리보기

활성화 설정

상태

ON

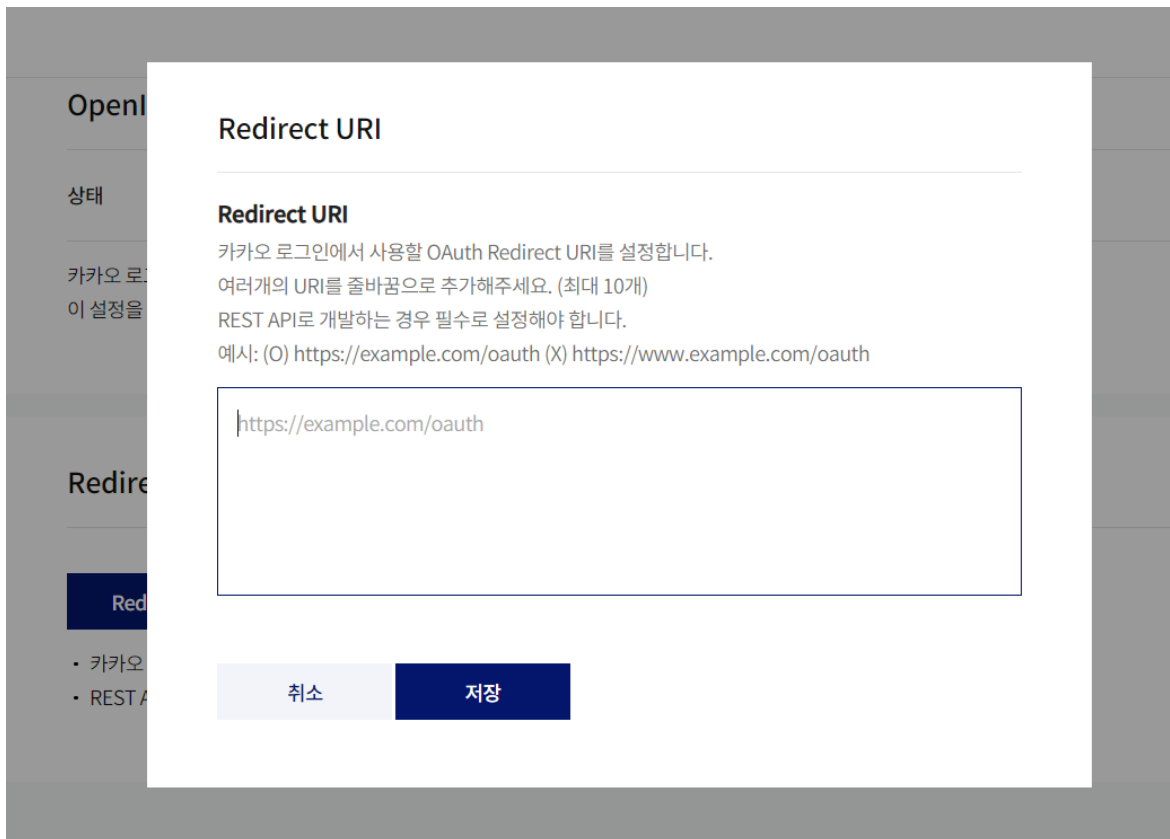
카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오톡으로 서비스를 시작할 수 있습니다.

상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.

상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

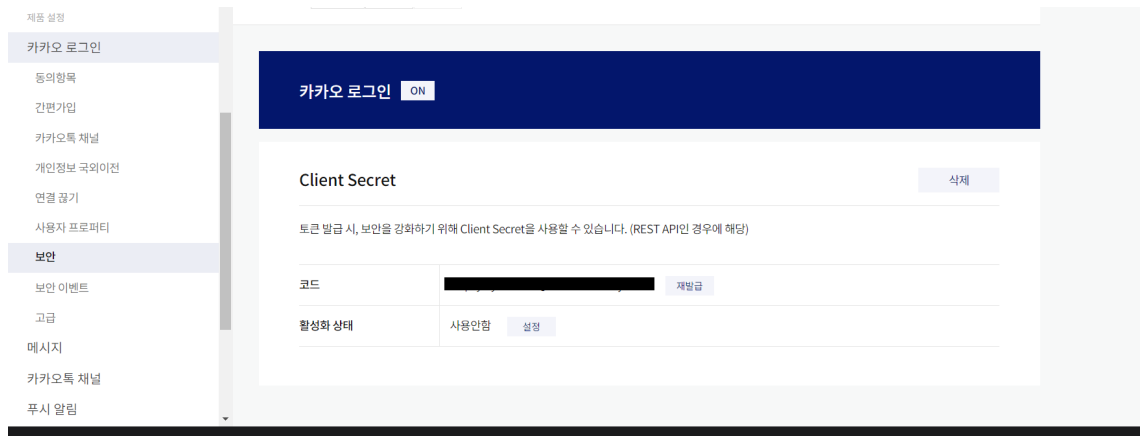
## 6. 카카오 로그인 ( Redirect URL ) 추가

→ 로그인 후.. 결과값을 받아올 백엔드 주소 !



7. 카카오 로그인 → 보안 (ON / 코드 재발급 / 활성화 설정 on)

→ 코드는 사용할 예정!!!



## ▼ 카카오톡 백엔드 적용하기

### 1. application.yml

#### 카카오톡 Security OAuth2 설정

```
security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: 카카오톡 client-id
          client-secret: 카카오톡 로그인 client-secret 값
          scope: //동의를 구할 목록
            - account_email
```

```

- profile_nickname
- profile_image
authorization-grant-type: authorization_code
redirect-uri: 로그인후 redirect로 정보를 받을 uri 주소
client-name: Kakao
client-authentication-method: POST

provider:
  kakao:
    authorization-uri: https://kauth.kakao.com/oauth/authorize // 카카오 인증 (로그인 유저 인증 코드 받음)
    token-uri: https://kauth.kakao.com/oauth/token // 카카오 토큰 (로그인 유저 토큰 발급 받음)
    user-info-uri: https://kapi.kakao.com/v2/user/me // 카카오 유저정보 (로그인 유저 정보 받음)
    user-name-attribute: id

```

## 2. JwtSuccessHandler / KakaoFallHandler

- response.sendRedirect  
url을 프론트 주소로 주시면 됩니다  
ex) "<https://localhost:3000/login>"  
ex) "<https://localhost:3000/dummykakao?code=>" + userCode

### ▼ 카카오톡 프론트엔드 적용하기

#### frontend → client → .env 파일 설정

```

REACT_APP_KAKAO_KEY = 카카오 JS 키

```