포팅메뉴얼

목차

1. 개발환경

1.1 Frontend

React 및 관련 라이브러리

- react ^18.3.1
- react-dom ^18.3.1
- react-router-dom ^6.27.0
- react-scroll ^1.9.0
- react-icons ^5.3.0

스타일링 및 UI 라이브러리

- @emotion/react ^11.13.3
- @emotion/styled ^11.13.0

유틸리티 및 데이터 라이브러리

- axios ^1.7.7
- zustand ^5.0.1

테스팅 라이브러리

- @testing-library/jest-dom ^5.17.0
- @testing-library/react ^13.4.0
- @testing-library/user-event ^13.5.0

서비스 워커 및 캐싱 관련 라이브러리

- workbox-background-sync ^6.6.0
- workbox-broadcast-update ^6.6.0
- workbox-cacheable-response ^6.6.0
- workbox-core ^6.6.0
- workbox-expiration ^6.6.0
- workbox-google-analytics ^6.6.1
- workbox-navigation-preload ^6.6.0
- workbox-precaching ^6.6.0
- workbox-range-requests ^6.6.0
- workbox-routing ^6.6.0
- workbox-strategies ^6.6.0
- workbox-streams ^6.6.0

개발 도구 및 형식화

- eslint ^8
- eslint-config-next 14.2.5
- eslint-config-prettier ^9.1.0

기타 라이브러리

- web-vitals ^2.1.4

1.2 Backend

자바

- Java OpenJDK 17
- Spring Boot 3.3.1
- Spring Dependency Management 1.1.5
    - Spring Data JPA 3.3.1
    - Spring Security 3.3.1
    - Websocket 3.3.1

    - Validation 3.3.1
    - Lombok 1.18.34
- Gradle 8.8

JSON 및 XML 처리

- jsoup 1.15.3
- org.json 20231013

JWT

- io.jsonwebtoken 0.11.5

데이터베이스

- MySQL 8.0.33

Amazon S3

- com.amazonaws 1.12.765

SpringDoc

- org.springdoc 2.6.0

1.3 Server

- Ubuntu 20.04.6 LTS
- Nginx 1.27.0
- Docker 27.1.1
- Docker Compose 1.29.2
- Jenkins 2.471

1.4 Database

- MySQL 9.0.1

1.5 IDE

- Visual Studio Code 1.90.2
- IntelliJ IDEA 2024.1.4

1.6 형상 / 이슈관리

- Gitlab
- Jira

1.7 기타 툴

- Postman 11.8.0

2. 환경변수

2.1 Backend

가독성이 좋은 application.yml 을 작성하여 환경변수를 관리

```
server:
  port: ${BACK_PORT}

fast-api:
  base-url: ${PROTOCOL}://${DOMAIN}:${FAST_API_PORT}/crawling

logging:
  level:
    root: ${ROOT_LOG_LEVEL}
    com.ssafy.a304.shortgong.domain: ${DOMAIN_LOG_LEVEL}
    com.ssafy.a304.shortgong.global: ${GLOBAL_LOG_LEVEL}

spring:
  config:
    import: optional:file:.env[.properties]

  application:
    name: shortgong

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${DOMAIN}:3306/${SCHEMA_NAME}?serverTimezone=Asia/Seoul
    username: ${SPRING_DATASOURCE_USERNAME}
    password: ${SPRING_DATASOURCE_PASSWORD}

  jpa:
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQLDialect
        jdbc:
          time_zone: Asia/Seoul
        ddl-auto: ${JPA_DDL_AUTO_OPTION}
        format_sql: true
    defer-datasource-initialization: true
    generate-ddl: true
    show-sql: ${JPA_SHOW_SQL_OPTION}
    open-in-view: false

  sql:
    init:
      mode: ${SQL_INIT_MODE}

  data:
    redis:
      host: ${DOMAIN}
```

```
      port: ${REDIS_PORT}

  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB
      max-request-size: 10MB

  naver:
    tts:
      url: https://naveropenapi.apigw.ntruss.com/tts-premium/v1/tts
      client-id: ${NAVER_SHORTGONG_CLIENT_ID}
      client-secret: ${NAVER_SHORTGONG_CLIENT_SECRET}
    ocr:
      url: ${NAVER_OCR_URL}
      secret-key: ${NAVER_OCR_SECRET}

jwt:
  secret: ${JWT_SECRET}

cors:
  allowedOrigins: ${PROTOCOL}://${DOMAIN}:${FRONT_PORT}

springdoc:
  swagger-ui:
    path: /swagger-ui.html

expire-time:
  redis-access-token: 14400
  jwt-access-token: 36000000
  jwt-refresh-token: 604800000

cloud:
  aws:
    s3:
      bucket: ${S3_BUCKET_NAME}
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}
    region:
      static: us-east-1
      auto: false
    stack:
      auto: false
    cloudfront:
      domain: ${CLOUDFRONT_DOMAIN}
      key-pair-id: ${CLOUDFRONT_KEY_PAIR_ID}
      private-key: ${CLOUDFRONT_KEY_FILE_PATH}
```

```
file:
  path:
    user-profile-folder: user
    summary-folder: summary
    upload-content-folder: upload-content

claude:
  api:
    keys:
      key-1: ${CLAUDE_API_SECRET_KEY_1}
      key-2: ${CLAUDE_API_SECRET_KEY_2}
      key-3: ${CLAUDE_API_SECRET_KEY_3}
      key-4: ${CLAUDE_API_SECRET_KEY_4}
      key-5: ${CLAUDE_API_SECRET_KEY_5}
      key-6: ${CLAUDE_API_SECRET_KEY_6}
      key-7: ${CLAUDE_API_SECRET_KEY_7}
      key-8: ${CLAUDE_API_SECRET_KEY_8}
      key-9: ${CLAUDE_API_SECRET_KEY_9}
      key-10: ${CLAUDE_API_SECRET_KEY_10}
    url: https://api.anthropic.com/v1/messages
    model: claude-3-5-sonnet-20241022
    temperature: 0.7
    max-tokens: 8192

eleven-labs:
  api:
    keys:
      key-1: ${ELEVEN_LABS_KEY_1}
      key-2: ${ELEVEN_LABS_KEY_2}
      key-3: ${ELEVEN_LABS_KEY_3}
    tts:
      url: https://api.elevenlabs.io/v1/text-to-speech/
```

2.3 민감 환경변수 관리

2.3.1 Backend

> **Jenkins Pipeline의 workspace에 위치한 프로젝트 Git Repository에서 .env 수동 저장 및 관리 (.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)**

```
# 경로
/home/ubuntu/jenkins-data/workspace/shortgong/server/shortgong

### shortgong는 Jenkins Pipeline 이름
```

3. EC2 세팅

3.1 Docker 설치

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyr
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Docker 패키지 설치(최신 버전)
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plug

# Docker Engine 설치 성공 확인
sudo docker run hello-world
```

3.2 MySQL(Docker) 설치

```
# MySQL Docker 이미지 다운로드
## 버전 명시하지 않으면 최신버전으로 다운로드
$sudo docker pull mysql

# MySQL Docker 컨테이너 생성 및 실행
$sudo docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> -d
```

3.3 Nginx 설치 및 Reverse Proxy 세팅

3.3.1 nginx 이미지 다운로드

```
$docker pull nginx:latest
```

3.3.2 nginx.conf 파일 작성

```
* ec2 인스턴스 /home/ubuntu/nginx.conf 경로에 존재

events { }

http {
        # DDoS 방어 설정
    limit_req_zone $binary_remote_addr zone=ddos_req:50m rate=20r/s;

    upstream frontend {
```

```
        server frontend:3000;
    }

    upstream backend {
        server backend:8080;
    }


    server {
        listen 80;
        server_name k11a304.p.ssafy.io;


        location /.well-known/acme-challenge/ {
              allow all;
            root /var/www/certbot;
        }
        # http 요청 https로 리다이렉션
        location / {
            return 301 https://$host$request_uri;
        }
    }


    server {
        listen 443 ssl;
        server_name k11a304.p.ssafy.io;

                # ssl 인증서 관련 부분
                # ssl 인증서 받기 전에는 이부분을 주석 처리해야 오류가 안남.
        ssl_certificate /etc/letsencrypt/live/k11a304.p.ssafy.io/fullchain.pe
        ssl_certificate_key /etc/letsencrypt/live/k11a304.p.ssafy.io/privkey.
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        # 최대 요청 크기 설정 (20MB)
        client_max_body_size 100M;

        # 타임아웃 설정 (5분 = 300초)
        client_header_timeout 300s;
        client_body_timeout 300s;
        send_timeout 300s;
        keepalive_timeout 300s;
        proxy_read_timeout 300s;
        proxy_connect_timeout 300s;


        location / {
```

```
            proxy_pass http://frontend/;
            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;
            limit_req zone=ddos_req burst=10;

        }


        location /api/ {
            proxy_pass http://backend/api/;
            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;

        }

        # Swagger 관련 리소스 필터링
        location ~ ^/(swagger|webjars|configuration|swagger-resources|v2|v3|c
            proxy_pass http://backend;
            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;
        }


    }
}
```

3.3.3 docker-compose-prod.yml 파일 작성

- docker-compose-prod.yml은 jenkins와 연동된 gitlab 레포지토리 안에 존재한다.

```
version: '3'
services:
  backend:
    container_name: back-server
    build:
      context: ./server/shortgong
      dockerfile: Dockerfile
    image: shortgong_backend:latest
    ports:
      - "8080:8080"
    env_file:
      - ./server/shortgong/.env
    volumes:
```

```
      - /home/ubuntu/jenkins-data/workspace/shortgong/server/shortgong/privat
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  crawling:
    container_name: crawling-server
    build:
      context: ./crawling_server
      dockerfile: Dockerfile
    image: shortgong_crawling:latest
    ports:
      - "5000:5000"
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  frontend:
    container_name: front-client
    build:
      context: ./front
      dockerfile: Dockerfile
    image: shortgong_frontend:latest
    ports:
      - "3000:3000"
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf
      - /home/ubuntu/data/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/data/certbot/www:/var/www/certbot
    networks:
      - shortgong

  certbot:
    image: certbot/certbot
    volumes:
      - /home/ubuntu/data/certbot/conf:/etc/letsencrypt
```

```
      - /home/ubuntu/data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep

networks:
  shortgong:
```

3.3.4 SSL 인증서 받기

```
# ec2 인스턴스

# ssl 인증서를 저장할 경로를 만들어 준다.
$ mkdir -p data/certbot/conf
$ mkdir -p data/certbot/www

$ curl -L https://raw.githubusercontent.com/wmnnd/nginx-certbot/master/init-l
$ chmod +x init-letsencrypt.sh
$ vi init-letsencrypt.sh // 도메인, 이메일, 디렉토리 수정

domains=(k11a304.p.ssafy.io)
rsa_key_size=4096
data_path="./data/certbot"
email="" # Adding a valid address is strongly recommended
staging=0 # Set to 1 if you're testing your setup to avoid hitting request li
-> 위 부분 처럼 도메인, 경로 지정 후 저장

$ sudo ./init-letsencrypt.sh // script를 실행하여 인증서 발급
```

3.5 EC2 Port

Port 번호	내용
22	SSH
80	HTTP (HTTPS로 redirect)
443	HTTPS
3306	MySQL
3000	Frontend
8080	Backend
8081	Jenkins
5000	Fast API

4. CI/CD 구축

4.1 Jenkins Dockerfile, Docker in Docker 방식

```
# ec2 인스턴스 /home/ubuntu

# jenkins dockerfile을 저장할 디렉토리 생성
```

```
$ mkdir -p jenkins-dockerfile

# /home/ubuntu/jenkins-dockerfile 루트에 Dockerfile 생성

FROM jenkins/jenkins:2.471-jdk17

USER root

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")
add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo
$(lsb_release -cs) \
stable" && \
apt-get update && \
apt-get -y install docker-ce

RUN curl -L "https://github.com/docker/compose/releases/download/1.29.2/docke
chmod +x /usr/local/bin/docker-compose

RUN groupadd -f docker
RUN usermod -aG docker jenkins

USER jenkins
```

4.2 Jenkins docker-compose.yml

```
# ec2 인스턴스 /home/ubuntu/

# Docker 볼륨 폴더 권한 설정
$ mkdir -p jenkins-data
$ sudo chown 1000 /home/ubuntu/jenkins-data/

# /home/ubuntu/jenkins-dockerfile/ 루트에 docker-compose.yml 생성

version: "3.2"
services:
    jenkins:
       container_name: jenkinscicd
       build:
          context: .
```

```
      dockerfile: Dockerfile
    restart: unless-stopped
    ports:
      - 8081:8081
      - 50000:50000
    environment:
      - JENKINS_OPTS=--httpPort=8081
      - TZ=Asia/Seoul
    volumes:
      - /home/ubuntu/jenkins-data:/var/jenkins_home
      - /home/ubuntu/jenkins/.ssh:/root/.ssh
      - /var/run/docker.sock:/var/run/docker.sock

# /home/ubuntu/jenkins
# 젠킨스 컨테이터 실행
$ sudo docker-compose up -d --build
```

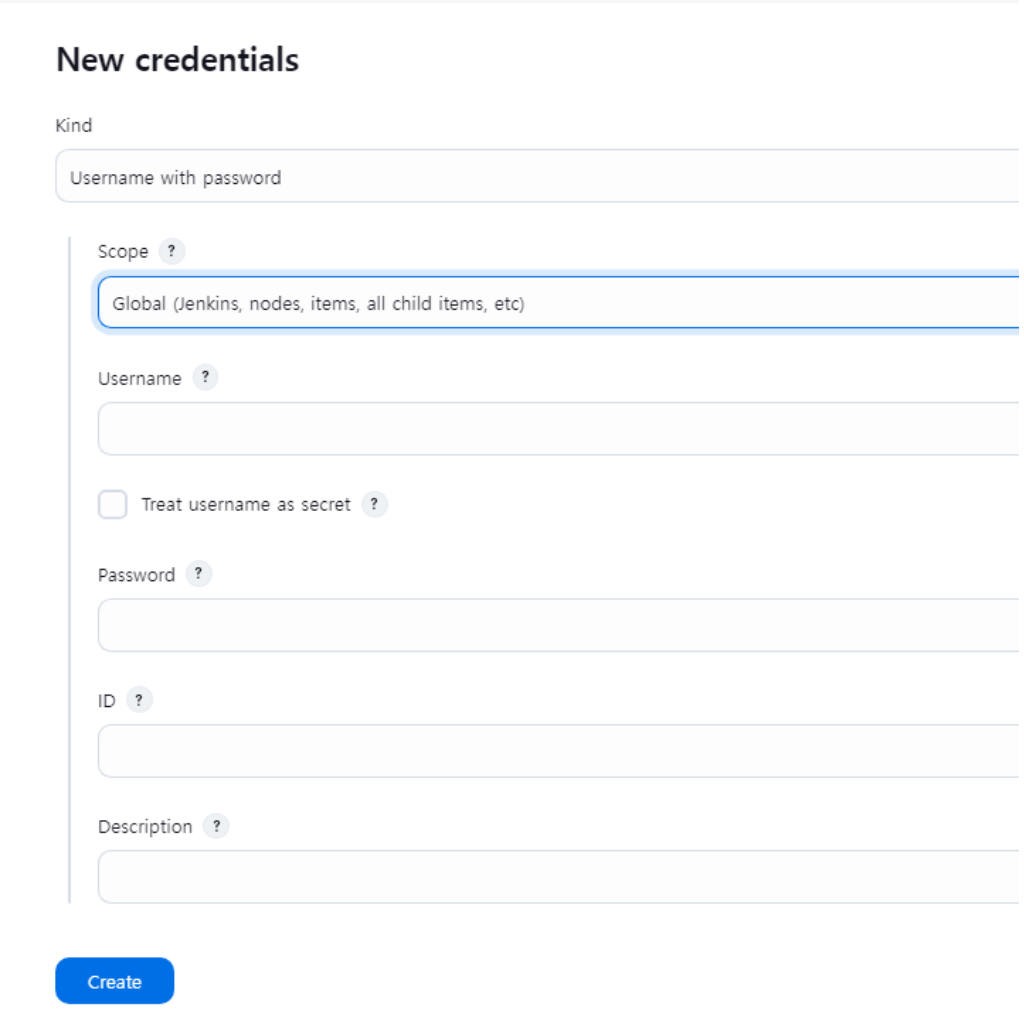
4.3 Jenkins docker 권한 설정

```
# jenkins 접속 전 /var/run/docker.sock 에 대한 권한을 설정해줘야 함.
# 초기 권한이 소유자와 그룹 모두 root였기 때문에 이제 그룹을 root에서 docker로 변경

# jenkins 컨테이너 접속
$ sudo docker exec -it -u root jenkinscicd /bin/bash

# 그룹 변경
$ sudo chown root:docker /var/run/docker.sock
$ exit
```

4.4 Jenkins 설정

4.4.1 GitLab Credentials 설정

1. 아이디 → “Credentials” 클릭
2. “Store : System” → “(global)” → “+ Add Credentials” 클릭
3. Username: GitLab ID
4. Password: GitLab Personal Access Tokens

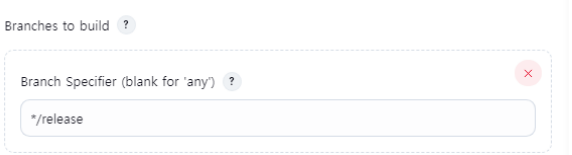


4.4.2 Jenkins Item 생성

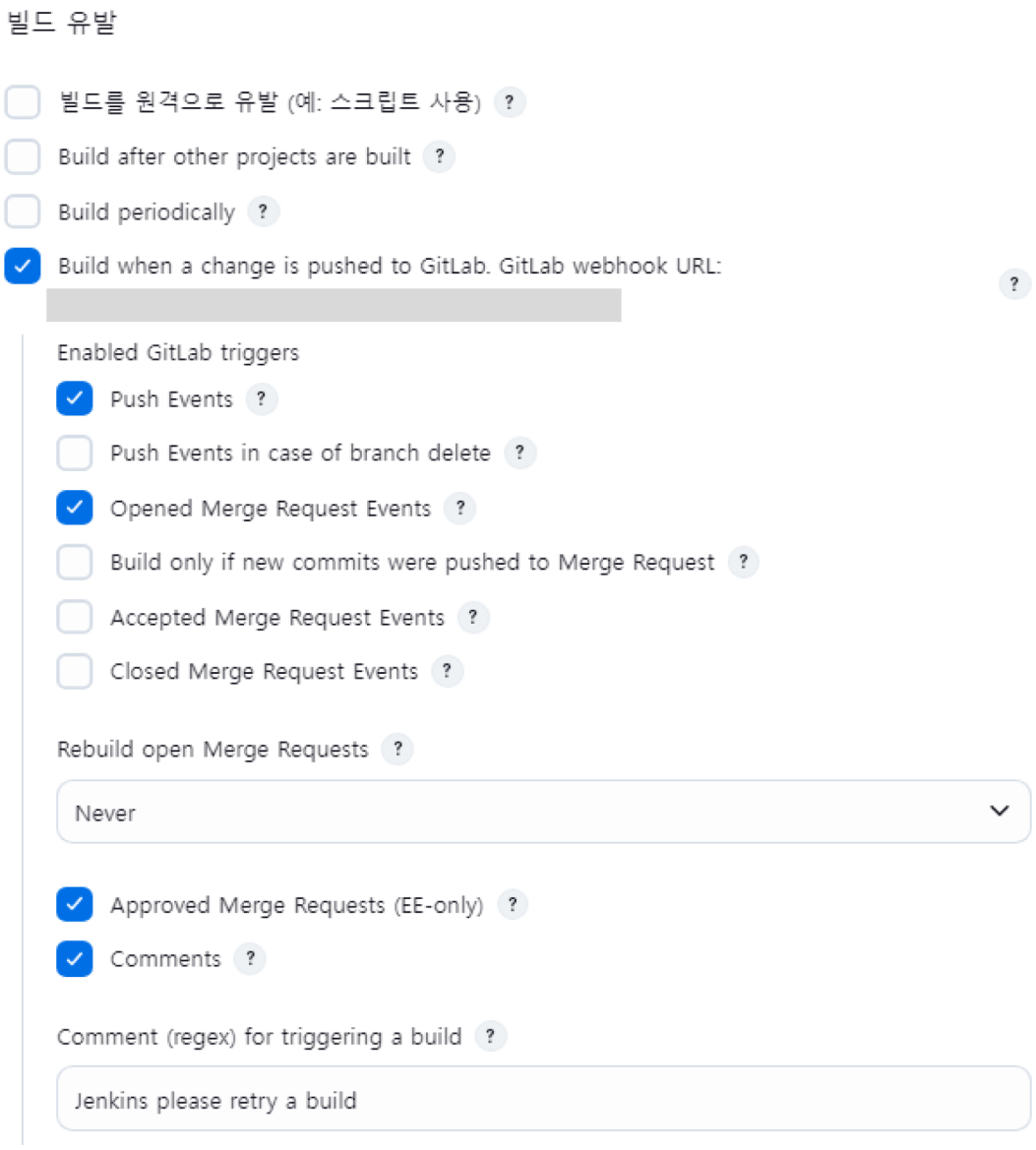
1. “새로운 Item” 클릭
2. “Enter an item name”에 임의 Item 이름 입력 → “Freestyle project” 클릭

3. “구성” 클릭
4. “소스 코드 관리”에 연동할 Repository URL 입력
5. 등록해둔 Credentials 설정

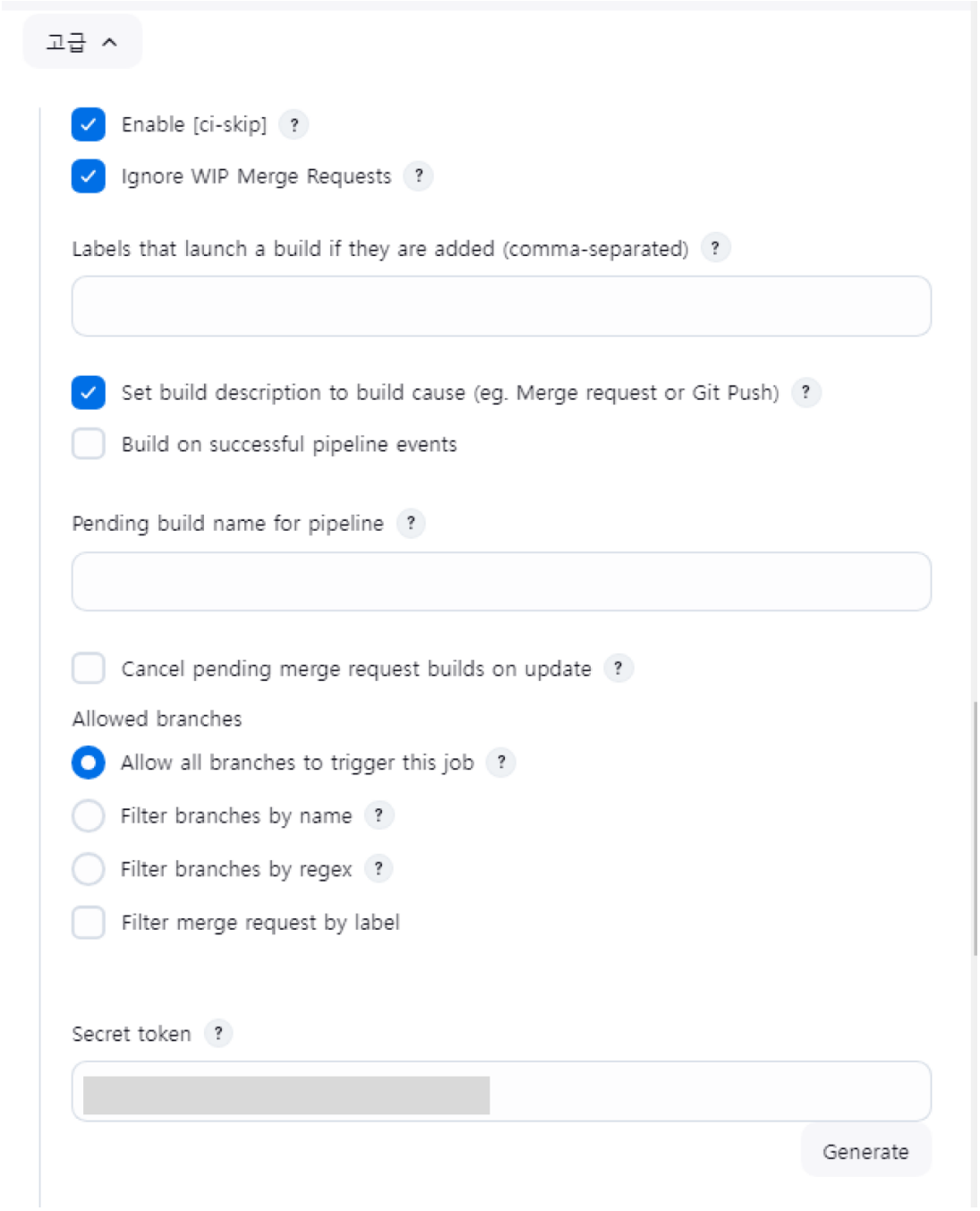
6. 연동할 브랜치 입력



7. “빌드 유발”에서 트리거 설정
8. GitLab webhook URL 기록해두기

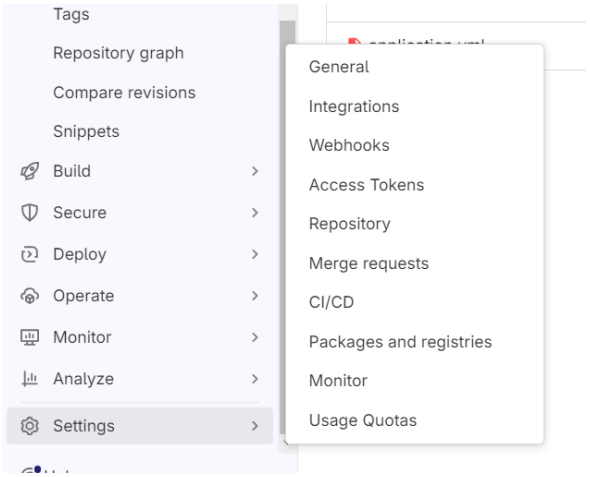


9. “빌드유발” 안에 “고급” 클릭
10. Secret Token Generate & 기록해두기

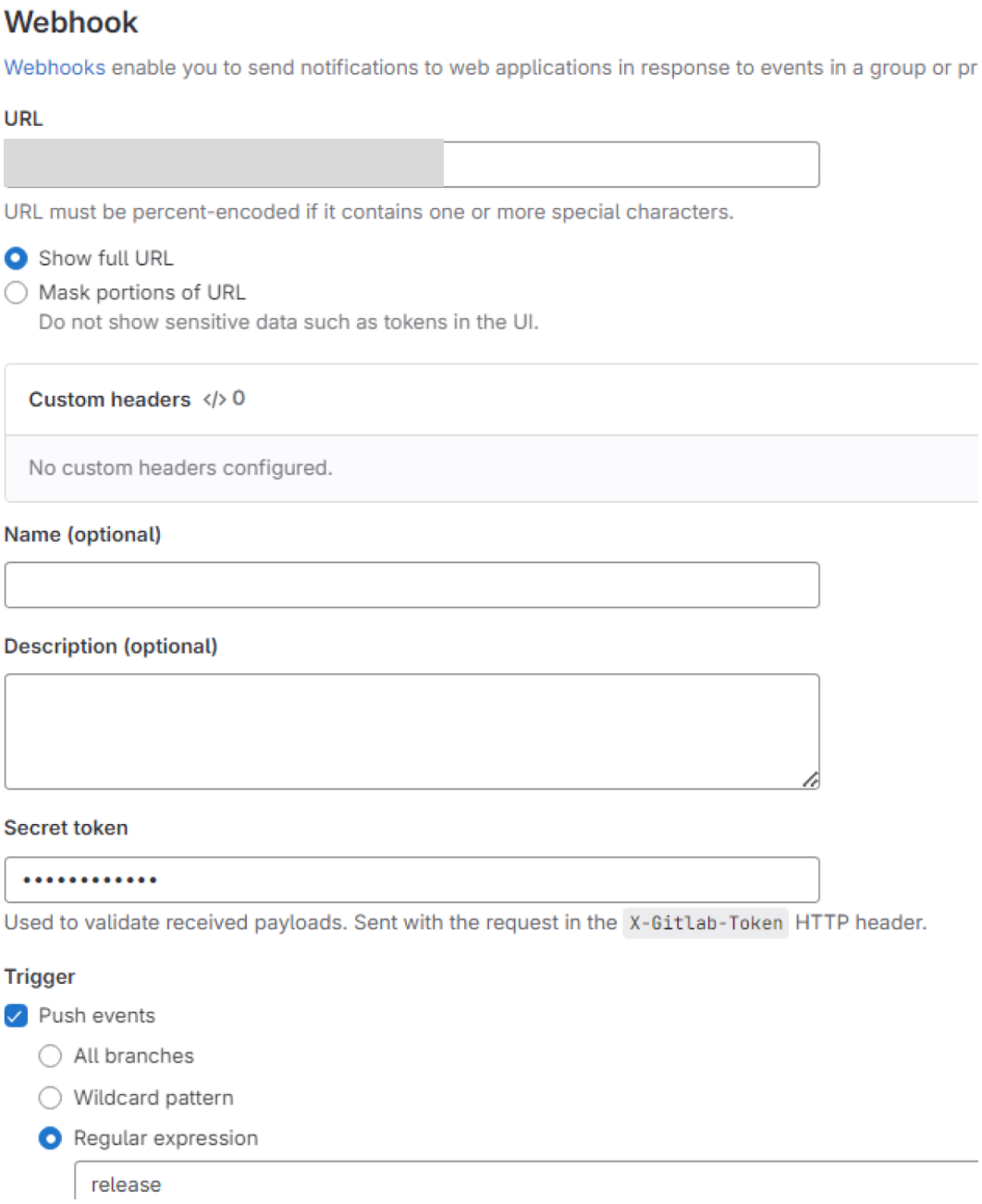


4.4.3 GitLab Webhook 설정

1. 프로젝트 GitLab → “Settings” → “Webhooks” 클릭



2. “URL”에 사전에 복사해놓은 Jenkins URL 입력
3. “Secret token”에 사전에 복사해놓은 Secret token 입력
4. “Trigger” Push events 클릭 후 “Regular expression” 에 연결 브랜치 입력



4.4.4 빌드 후 배포 명령어 설정

1. 만들어 둔 아이템 “구성” 클릭
2. “Build Steps”에서 Execute shell 작성 후 저장

4.4.5 빌드 및 배포

> **상기 WebHook 설정한 브랜치로 푸시 및 MergeRequest**

4.5 Jenkins 연동 브랜치 배포 파일 구성

4.5.1 루트 경로 파일구조

```
S11P31A304/
│
├── .idea/                          # IDE 관련 설정 디렉토리 (IntelliJ 등)
│
├── server/                        # 백엔드 소스 코드가 위치한 디렉토리
│
├── front/                      # 프론트엔드 소스 코드가 위치한 디렉토리
│
├── docker-compose-prod.yml      # 프로덕션 환경용 Docker Compose 파일
│
├── README.md                    # 프로젝트에 대한 설명과 정보를 담은 파일
│
└── start-prod.sh                # 프로덕션 환경에서 실행할 스크립트 파일
```

4.5.2 start-prod.sh 빌드 후 실행되는 파일

```
docker-compose -f docker-compose-prod.yml down

docker rmi -f $(docker images shortgong_backend:latest -q) || true
docker rmi -f $(docker images shortgong_frontend:latest -q) || true
docker rmi -f $(docker images shortgong_crawling:latest -q) || true

docker-compose -f docker-compose-prod.yml pull

COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose

docker rmi -f $(docker images -f "dangling=true" -q) || true
```

4.5.3 docker-compose-prod.yml 파일

```
version: '3'
services:
  backend:
    container_name: back-server
    build:
      context: ./server/shortgong
      dockerfile: Dockerfile
    image: shortgong_backend:latest
    ports:
      - "8080:8080"
    env_file:
      - ./server/shortgong/.env
    volumes:
      - /home/ubuntu/jenkins-data/workspace/shortgong/server/shortgong/privat
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  crawling:
    container_name: crawling-server
    build:
      context: ./crawling_server
      dockerfile: Dockerfile
    image: shortgong_crawling:latest
    ports:
      - "5000:5000"
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  frontend:
    container_name: front-client
    build:
      context: ./front
      dockerfile: Dockerfile
    image: shortgong_frontend:latest
    ports:
      - "3000:3000"
    environment:
      - TZ=Asia/Seoul
    networks:
      - shortgong

  nginx:
```

```
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf
      - /home/ubuntu/data/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/data/certbot/www:/var/www/certbot
    networks:
      - shortgong

  certbot:
    image: certbot/certbot
    volumes:
      - /home/ubuntu/data/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep

networks:
  shortgong:
```

5. Front, Back 배포 Dockerfile

5.1 Frontend

5.1.1 frontend 루트 경로 파일구조

```
front/
├── public/
├── src/
├── .dockerignore
├── .gitignore
├── .prettierrc
├── Dockerfile
├── package.json
├── package-lock.json
```

5.1.2 frontend Dockerfile

1. 경량화하여 이미지를 만들도록 설정함.
2. “package.json” 파일 수정

   window 에서는 set NODE_OPTIONS ~~~ 이지만

   Linux 환경에서는 export 명령어를 사용해야 한다.

```
"scripts": {
    "start": "export NODE_OPTIONS=--openssl-legacy-provider && react-scripts
    "build": "export NODE_OPTIONS=--openssl-legacy-provider && react-scripts
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

```
# Dockerfile

# 공식 Node.js 이미지를 베이스 이미지로 사용
FROM node:18-alpine

# 컨테이너 내 작업 디렉터리 설정
WORKDIR /app

# package.json과 package-lock.json (존재하는 경우) 복사
COPY package*.json ./

# 의존성 설치
RUN npm ci

# 프로젝트 전체를 컨테이너로 복사
COPY . .

EXPOSE 3000

# React를 위한 환경 변수 설정
ENV NODE_ENV=production

# 애플리케이션 실행
CMD ["npm", "start"]
```

5.1.3 .dockerignore 파일

```
node_modules
```

5.2 Backend

5.2.1 backend 루트 경로 파일구조

```
server/
└── shortgong/
    ├── .gradle/              # Gradle 관련 임시 파일 및 캐시 디렉토리
    ├── .idea/                # IntelliJ IDEA 프로젝트 설정 파일
    ├── build/                # 빌드 아티팩트(컴파일 결과물 등)가 저장되는 디렉토리
    ├── gradle/               # Gradle 래퍼 관련 파일
    ├── src/                  # 소스 코드 디렉토리 (주요 애플리케이션 코드 포함)
```

```
    ├── .env                  # 환경변수 설정 파일
    ├── .gitattributes        # Git 속성 파일 (주로 EOL 등 설정)
    ├── .gitignore            # Git에서 추적하지 않을 파일/디렉토리 설정
    ├── build.gradle          # Gradle 빌드 스크립트
    ├── Dockerfile            # Docker 컨테이너 생성에 사용되는 설정 파일
    ├── gradlew               # Gradle 래퍼 스크립트 (Unix 기반 시스템용)
    ├── gradlew.bat           # Gradle 래퍼 스크립트 (Windows용)
    ├── private_key.pem       # 개인 키 파일 (보안 인증에 사용, 민감 정보)
    ├── settings.gradle       # Gradle 설정 파일 (프로젝트 구성 관리)
```

5.2.2 backend Dockerfile

```
# Gradle 버전을 8.x로 변경
FROM gradle:8.2-jdk17 AS build
WORKDIR /app

# 프로젝트 전체 복사
COPY . .

# Gradlew 파일에 실행 권한 추가
RUN chmod +x ./gradlew

# Gradle 빌드 실행
RUN ./gradlew clean build -x test

FROM bellsoft/liberica-openjdk-alpine:17
LABEL authors="SSAFY"

ARG JAR_FILE=build/libs/*.jar

# 빌드된 JAR 파일 복사
COPY --from=build /app/${JAR_FILE} app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app.jar"]
```