포팅 메뉴얼

---

1) 프로젝트 및 환경 정보

백엔드

- JDK 17 (HotSpot JVM)
- Spring Boot 3.3.1
- Tomcat 10
- IntelliJ 2024.1.4

프론트

- Next 14.2.5
- Node 20.15.1
- Styled-Components 6.1.12
- Zustand 4.5.4
- TypeScript 5.5.3
- VSCode 1.92.0

AI (OCR)

- Python 3.12.4
- FastAPI 0.112.0
- VsCode 1.92.0

공통 사항

- URL
    - Server
        - https://i11a307.p.ssafy.io
    - Client
        - https://www.teamblurrr.com
        - https://teamblurrr.com
- DB
    - Driver
        - mysql
    - Host
        - i11a307.p.ssafy.io:3306
    - User
        - root
    - password
        - blurr123
- Docker
    - 26.1.4, build 5650f9b
- Docker Compose
    - version v2.27.1-desktop.1

2) 클론 후 빌드 및 실행 방법 (Ubuntu OS 사용)

해당 과정은 클론 후 해당 경로에서 진행

백엔드 빌드 및 실행 방법

💡 **포트 정보**

- 8080
  - 해당 포트 오픈 필요
- 우분투 서버에서 아래 명령어를 통해 git clone 후 루트 경로 환경 변수 지정

```
export GIT_ROOT_PATH=$(pwd)
```

1. docker & docker-compose 설치(아래 명령어 순서대로 실행)

```
sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-co

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

sudo curl -L "https://github.com/docker/compose/releases/download/1.28.2/docker-compose-$(uname

sudo chmod +x /usr/local/bin/docker-compose

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

sudo systemctl start docker
```

2. JDK 17 설치(아래 명령어 순서대로 실행)

```
SUDO apt-get install openjdk-17-jdk -y

export JAVA_HOME="/usr/lib/jvm/java-17-openjdk-amd64"

export PATH="$JAVA_HOME/bin:$PATH"
```

3. 빌드 시 필요한 yml 파일 생성 및 빌드

```
cd $GIT_ROOT_PATH/backend/blur/src/main/resources

touch application.yml application-dev.yml application-aws.yml application-jwt.yml application-ma
```

▼ application.yml

```
server:
  port: 8080

spring:
  application:
    name: blur

  profiles:
    default: local

  config:
    import: application-aws.yml, application-mail.yml, application-jwt.yml

  docker:
    compose:
      lifecycle-management: start_only
      enabled: true
      stop:
        command: down

  data:
    elasticsearch:
      repositories:
        enabled: true
      url: localhost:9200

  datasource:
    p6spy:
      enable-logging: true
      multiline: true
      logging: slf4j
      tracing:
        include-parameter-values: true

logging:
  level:
    org.hibernate.sql: debug
    org.hibernate.orm.jdbc.bind: trace
    org.hibernate.type.descriptor.sql.BasicBinder: trace

sse:
  time:
    timeout: 180000
```

▼ application-aws.yml

```
spring:
  cloud:
    aws:
      ses:
        enabled: true

aws:
```

```
  ses:
    key:
      access: AKIAU6GDV5R67BM6ZYJM
      secret: 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
    from: tkdgus1608@gmail.com

cloud:
  aws:
    credentials:
      access-key: AKIAU6GDV5R6TW3ACKGZ
      secret-key: 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
    s3:
      bucket-name: blurrr-img-bucket
    region:
      static: ap-northeast-2
    stack:
      auto: false
```

▼ application-dev.yml

```
server:
  port: 8080

spring:
  config:
    activate:
      on-profile: dev

  datasource:
    url: jdbc:mysql://localhost:3306/blur
    username: blur
    password: blurr123
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate.dialect: org.hibernate.dialect.MySQL8Dialect


  data:
    redis:
      host: localhost
      port: 6379
```

▼ application-jwt.yml

```
jwt:
  secret: dGVhbWJsdXJyLWp3dC10b2tlbi1zc2FmeTExdGgtYTMwNy1yYW5kb21SdHJpbmc6ajFENVNiMWozampzTmt
  expire:
    time:
      access: 1800000
      refresh: 604800000
```

▼ application-mail.yml

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: teamluckyvickyblurrr@gmail.com
    password: qktv hilu nspo ucjy
    properties:
      mail.smtp.debug: true
      mail.smtp.connectiontimeout: 1000 #1초
      mail.starttls.enable: true
      mail.smtp.auth: true

email:
  time:
    valid: 300000
    available: 86400000
```

4. 빌드 파일 생성

```
cd $GIT_ROOT_PATH/backend/blur

chmod +x ./gradlew

./gradlew clean build --refresh-dependencies --stacktrace --info -x test
```

5. Elastick Search 도커 컨테이너 구동

현재 위치에서 Dockerfile 생성

```
vi Dockerfile
```

아래 내용 작성 후 저장

```
FROM docker.elastic.co/elasticsearch/elasticsearch:8.6.2

RUN elasticsearch-plugin install analysis-nori
RUN bin/elasticsearch-plugin install https://github.com/netcrazy/elasticsearch-jaso-analyzer/rel
```

docker-compose.yml 파일 생성

```
vi docker-compose.yml
```

아래 내용 작성 후 저장

```
services:

    es:
      build:
        context: .
```

```
    container_name: es

    environment:
      - node.name=es-node
      - cluster.name=search-cluster
      - discovery.type=single-node
      - xpack.security.enabled=false
      - xpack.security.http.ssl.enabled=false
      - xpack.security.transport.ssl.enabled=false

    ports:
      - 9200:9200
      - 9300:9300

    networks:
      - es-bridge


  kibana:

    image: docker.elastic.co/kibana/kibana:8.6.2

    container_name: kibana

    environment:
      SERVER_NAME: kibana
      ELASTICSEARCH_HOSTS: http://es:9200

    ports:
      - 5601:5601

    # Elasticsearch Start Dependency
    depends_on:
      - es

    networks:
      - es-bridge


networks:

  es-bridge:
    driver: bridge
```

Dockderfile과 docker-compose.yml이 존재하는 경로에서 아래 명령어 실행(반드시 Dockderfile과 docker-compose.yml 파일이 같이 있어야 함)

```
docker-compose up -d —build
```

엘라스틱 서치 서버 실행 후 리그 테이블 동기화를 위해 반드시 동기화 API 호출 필요

5. 실행

```
cd $GIT_ROOT_PATH
```

```
touch deploy.sh

chmod u+x deploy.sh
```

deploy.sh에 해당 내용 작성

```
#!/bin/bash
REPOSITORY=$GIT_ROOT_PATH

echo "로그 파일 삭제"
rm -rf $REPOSITORY/nohup.out

echo "현재 구동중인 애플리케이션 pid 확인"

CURRENT_PID=$(lsof -i tcp:8080 | awk 'NR!=1 {print $2}')

echo "현재 구동중인 애플리케이션 pid: $CURRENT_PID"

if [ -z "$CURRENT_PID" ]; then
    echo "> 현재 구동중인 애플리케이션이 없으므로 종료하지 않습니다."
else
    echo "> kill -9 $CURRENT_PID"
    kill -9 $CURRENT_PID
    sleep 5
fi

echo "> 새 애플리케이션 배포"
JAR_NAME=$(ls -tr $REPOSITORY/backend/blur/build/libs/*.jar | tail -n 1)

echo "> JAR Name: $JAR_NAME"
nohup java -jar -Dspring.profiles.active=dev $JAR_NAME &

echo "> 새 애플리케이션 배포 완료"
CURRENT_PID=$(lsof -i tcp:8080 | awk 'NR!=1 {print $2}')
echo "현재 구동중인 애플리케이션 pid: $CURRENT_PID"
```

저장 후 아래 명령어 실행

```
nohup ./deploy.sh > deploy.log 2>&1 &
```

프론트엔드 빌드 및 실행 방법

💡 **포트 정보**

- 3000
  - 해당 포트 오픈 필요
- git clone 루트 경로 환경 변수 설정

```
export GIT_ROOT_PATH=$(pwd)
```

1. NPM 및 Node 설치

node 설치

```
sudo apt-get install -y curl
sudo curl -fsSL https://deb.nodesource.com/setup_18.x | bash - || { echo "Node.js setup failed";
sudo apt-get install -y nodejs
```

라이브러리 다운 및 빌드

```
cd $GIT_ROOT_PATH/frontend

npm install

export NEXT_PUBLIC_API_URL=https://i11a307.p.ssafy.io
export NEXT_PUBLIC_ACCESS_KEY=AKIAU6GDV5R6TW3ACKGZ
export NEXT_PUBLIC_SECRET_ACCESS_KEY=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
export NEXT_PUBLIC_CLOUD_FRONT_URL=https://blurrr-img-bucket.s3.ap-northeast-2.amazonaws.com/
export NEXT_PUBLIC_DASHCAM_ID=11ef5958-c65a-fb5c-b0a6-6956f2c3d7ff
export NEXT_PUBLIC_BOAST_ID=11ef595a-8521-ad41-b0a6-3fc3cc43bfe7

npm run build
```

2. Docker 컨테이너 생성 및 구동

```
cd $GIT_ROOT_PATH/frontend

touch Dockerfile docker-compose.yml
```

Dockerfile 내용

```
FROM node:18.17.0-alpine3.18

# Create app directory
WORKDIR /usr/src/app

# Copy build output files
COPY ./public ./public
COPY ./.next/standalone ./
COPY ./.next/static ./.next/static

# Running the app
ENTRYPOINT [ "node", "server.js" ]
```

docker-compose.yml 내용

```
services:
  nextjs:
    container_name: blur-client
```

```
    build:
      context: .
      dockerfile: Dockerfile

    restart: unless-stopped

    ports:
      - "3000:3000"
```

deploy.sh 파일 생성 및 작성

```
touch deploy.sh
```

```
#!/bin/bash

CONTAINER_NAME="blur-client"
PORT="3000"

# Docker 기존 컨테이너 삭제
if docker ps -q -f name=${CONTAINER_NAME}; then
    echo "Stopping and removing existing Docker container..."
    docker stop ${CONTAINER_NAME}
    docker rm ${CONTAINER_NAME}
else
    echo "No existing container found."
fi

# Docker 컨테이너 실행
echo "Running Docker container..."
sudo docker-compose up -d --build

# 완료 메시지
echo "Docker container is running. You can access it on port ${PORT}"
```

```
cd $GIT_ROOT_PATH/frontend

nohup ./deploy.sh > deploy.log 2>&1 &
```

OCR 서버 빌드 및 실행 방법

📌 OCR SERVER 위치

- gitlab repository를 기준으로 ocr_server 폴더 car_certification에 있음

Dockerfile 작성

```
#Python 베이스 이미지 사용
FROM python:3.10

# 작업 디렉토리 설정
```

```
WORKDIR /app

#필요 라이브러리 설치
COPY requirements.txt .

RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

#애플리케이션 코드 복사
COPY . .

# 컨테이너에서 사용할 포트
EXPOSE 8000

#Uvicorn을 사용해 애플리케이션 실행
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Docker-Compose 작성

```
ubuntu@ip-172-26-10-150:~/ocr/car_certification$ cat docker-compose.yml
services:
  uvicorn:
    container_name: ocr-server
    build:
      context: .
      dockerfile: Dockerfile

    ports:
      - "8000:8000"
```

3) 배포

gitlab CI/CD를 통해 진행 deploy/frontend 또는 deploy/backend 브런치에 변경 사항이 생긴다면 ci/cd 자동으로 실행한다.

정보

- Tool
  - GitLab CI/CD
- Gitlab
  - 주소
    - https://lab.ssafy.com/s11-webmobile2-sub2/S11P12A307
    - deploy 폴더에 빌드 관련 파일 존재
  - Default Branch
    - dev
    - 클론 시 dev 브런치 clone

실행 방법

1. dev 브런치 clone
2. 새로운 브런치 생성
3. 소스 코드 수정 후 push
4. 생성 브런치 → dev 브런치로 MR 요청 후 merge
5. dev → deploy/frontend 또는 deploy/backend 다시 MR 요청 후 merge
6. 자동으로 배포 실행