포팅 매뉴얼

클라이언트 구축하기

버전 (버전 작성 필요함)

- React 18.2.0
- Recoil 0.7.7
- node 18.16.1
- Type Script 4.9.5
- Tailwind CSS 9.5.1

환경 변수

.env.prod (직접 생성 필요)

```
REACT_APP_KAKAO_CLIENT_ID = <KAKAO CLIENT ID>
REACT_APP_API_URL = https://hanol.site/api
REACT_APP_API_KEY = <FIREBASE API KEY>
REACT_APP_MESSAGING_SENDER_ID= <FIREBASE MESSAGE SENDER ID>
REACT_APP_APP_ID = <FIREBASE APP ID>
REACT_APP_VAPID_KEY = <FIREBASE VAPID KEY>
REACT_APP_MEASUREMENT_ID = <FIREBASE MEASUREMENT ID>
```

.env.docker (직접 생성 필요)

```
DOCKER_USERNAME=<username>
DOCKER_PASSWORD=<password>
```

빌드 매뉴얼

nginx.conf

```
server {
  listen 3000;
  location / {
    root /usr/share/nginx/html;
    index index.html;
    try_files $uri $uri/ /index.html;
  }
}
```

Dockerfile

```
FROM node:18 AS build

WORKDIR /app

COPY package*.json ./
RUN npm install
COPY . ./
COPY .env.prod .env
RUN GENERATE_SOURCEMAP=false npm run build

FROM nginx:stable-alpine
```

```
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

참고자료

React + Nginx 를 도커 이미지로 만들어서 배포하기

react-dockerizing 폴더 밑에 nginx 폴더를 만들고 nginx.conf 파일을 아래와 같이 생성해준다.80 포트에 / 경로로 들어오면 /usr/share/nginx/html 폴더에서 index.html 을 찾는다.react-dockerizing 밑에 D

https://velog.io/@tlatjdgh3778/React-Nginx-를-도커-이미지로-만들어서-배포하기

API 서버, 데이터베이스, RabbitMQ 구축하기

버전

- Java 11
- Spring Boot 2.7.1
- Spring Security
- Spring Batch
- Spring Data JPA
- Query DSL
- Gradle
- FCM
- MariaDB 11.1.2
- Redis 7.0.12

환경 변수

application-local.yml

```
server:
  port: 8000

spring:
  datasource:
    url: jdbc:mariadb://localhost:3307/hanol?useSSL=false&characterEncoding=UTF-8&serverTimezone=UTC&allowPublicKeyRetrieval=true
    driver-class-name: org.mariadb.jdbc.Driver
    username: root
    password: ssafya205!
  redis:
    host: localhost
    port: 6379
    password: ssafya205!
  jpa:
    defer-datasource-initialization: true
    hibernate:
      ddl-auto: create-drop
    show-sql: true
    properties:
```

```
      hibernate:
        format_sql: true
        jdbc:
          time_zone: Asia/Seoul
  rabbitmq:
    host: localhost
    port: 5672
    username: admin
    password: ssafya205!
  batch:
    jdbc:
      initialize-schema: always # 운영 환경 최초 배포 시 always, 이후 배포는 never로 설정 필요
      isolation-level-for-create: default # batch 작업 격리 수준을 설정한 DB와 일치 시킴
    job:
      enabled: false
  servlet:
    multipart:
      maxFileSize: 10MB
      maxRequestSize: 10MB

  sql:
    init:
      mode: always

oauth:
  kakao:
    client-id: <KAKAO CLIENT ID>
    admin-key: <KAKAO ADMIN ID>
    issuer: https://kauth.kakao.com


jwt:
  access-key: HanolAccessKey.a205+HanolAccessKey.a205
  refresh-key: HanolRefreshKEy.a205+HanolRefreshKEy.a205
  access-expired-min: 60
  refresh-expired-day: 10

logging:
  level:
    org:
      springframework: info
  pattern:
    console: "[%d{HH:mm:ss.SSS}][%-5level][%logger.%method:line%line] - %msg%n"

cloud:
  aws:
    s3:
      bucket: ${S3_BUCKET_NAME}
    region:
      static: ${S3_REGION}
      auto: false
    stack:
      auto: false
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}

rabbitmq:
  exchange:
    name: scalp-diagnose-exchange
    type: direct
  queue:
    request:
      name: scalp-diagnose-request-queue
      routing-key: scalp-diagnose-request
    response:
      name: scalp-diagnose-response-queue
      routing-key: scalp-diagnose-response

firebase:
  sdk:
    path: firebase-sdk.json
```

firebase-sdk.json

```
{
  "type": "service_account",
  "project_id": "hanol-project",
  "private_key_id": "PRIVATE KEY ID",
  "private_key": "PRIVATE KEY",
  "client_email": "CLIENT EMAIL",
  "client_id": "CLIENT ID",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-v7lr6%40hanol-project.iam.gserviceaccount.co
  "universe_domain": "googleapis.com"
}
```

.env.docker (직접 생성 필요)

```
DOCKER_USERNAME=<username>
DOCKER_PASSWORD=<password>
```

.env (데이터베이스 docker compose에서 필요)

```
MARIADB_HOST=localhost
MARIADB_PORT=3306
MARIADB_ROOT_PASSWORD=ssafya205!
MARIADB_DATABASE=hanol
MARIADB_USER=admin
MARIADB_PASSWORD=ssafya205!

REDIS_PASSWORD=ssafya205!

RABBITMQ_USER=admin
RABBITMQ_PASS=ssafya205!
```

빌드 매뉴얼

데이터베이스 docker-compose-local.yml

```
version: "3.8"

services:
  hanol-maria:
    container_name: hanol-maria
    image: mariadb:latest
    ports:
      - '3307:3306'
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
      - --character-set-client-handshake=FALSE
      - --lower_case_table_names=1
    env_file: .env
    environment:
      - MYSQL_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
      - MYSQL_DATABASE=${MARIADB_DATABASE}
      - MYSQL_USER=${MARIADB_USER}
      - MYSQL_PASSWORD=${MARIADB_PASSWORD}
      - TZ=Asia/Seoul
    restart: always
    volumes:
      - ./db/mariadb:/var/lib/mysql
  hanol-redis:
    container_name: hanol-redis
```

```
    image: redis:7.0.12
    ports:
      - '6379:6379'
    env_file: .env
    command: redis-server --requirepass ${REDIS_PASSWORD} --port 6379
    restart: on-failure
  rabbitmq:
    container_name: hanol-rabbitmq
    image: rabbitmq:management
    env_file: .env
    environment:
      - RABBITMQ_DEFAULT_USER=${RABBITMQ_USER}
      - RABBITMQ_DEFAULT_PASS=${RABBITMQ_PASS}
    ports:
      - '5672:5672'
      - '15672:15672'
```

Dockerfile

```
FROM openjdk:11-jdk-slim AS builder
WORKDIR app

COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN apt-get update && \
    apt-get install dos2unix && \
    apt-get clean
RUN dos2unix gradlew
RUN chmod +x ./gradlew
RUN ./gradlew bootjar

FROM openjdk:11-jdk-slim

ENV TZ Asia/Seoul
RUN cp /usr/share/zoneinfo/${TZ} /etc/localtime && \
    echo "${TZ}" > /etc/timezone && \
    cat "/etc/localtime"

WORKDIR app


COPY --from=builder ./app/build/libs/*.jar app.jar

EXPOSE 8000

ENTRYPOINT ["java", "-jar", "-Duser.timezone=Asia/Seoul","-Dspring.profiles.active=${PROFILE}" ,"app.jar"]
```

AI 문진 진단 모델 서버

버전

- fastAPI : 0.103.2
- scikit-learn : 1.3.0
- python : 3.10
- uvicorn : 0.23.2

docker-compose.yml

```
version : '3.10'

services:
        fastapi-survey:
                container_name: fastapi-survey
                image: hanol/hanol-repo:server-survey-1.0
                command: uvicorn main:app --port=8001 --host='0.0.0.0'
                expose:
                        - "8001"
                networks:
                        - hanol-dev

networks:
        hanol-dev:
                external: true
```

AI 두피 진단 모델 서버

버전

- fastAPI : 0.104.0
- python : 3.10
- efficientnet-pytorch : 0.7.1
- uvicorn : 0.20.0

GPU 서버 세팅

1. 사용 서버

- AWS P2.xLarge
- `Tesla K80` 1개 탑재

2-1. Docker 설치하기

```
# docker에 필요한 패키지 다운로드
sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

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

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

# docker 설치하기
sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io

# docker 그룹에 사용자 추가하기
sudo usermod -aG docker ubuntu

# docker 재시작
sudo service docker restart
```

2-2. Docker 네트워크 생성하기

```
docker network create ai-net
```

2-3. Docker 이미지 pull

```
docker pull hanol/hanol-repo:rabbitmq-server
docker pull hnaol/hanol-repo:ai-server
```

3-1. NVIDIA Driver 설치

```
sudo apt update
sudo apt install ubuntu-drviers-common

sudo ubuntu-drivers devices
sudo apt install nvidia-driver-470
```

3-2. CUDA 12.0 및 CUDNN 설치

CUDA 12.0 설치

```
wget https://developer.download.nvidia.com/compute/cuda/12.0.0/local_installers/cuda_12.0.0_525.60.13_linux.run
sudo sh cuda_12.0.0_525.60.13_linux.run

sudo vi /etc/profile
export PATH=$PATH:/usr/local/cuda-12.0/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-12.0/lib64
export CUDADIR=/usr/local/cuda-12.0

source /etc/profile
```

CUDNN 8.9.6 설치

- https://developer.nvidia.com/rdp/cudnn-download 접속
- Local Installer for Linux x86_64 (Tar) 다운로드
- 파일 ec2 업로드

```
tar xvzf cudnn-12.0-linux-x64-v8.0.5.39.tgz
sudo cp cuda/include/cudnn* /usr/local/cuda-12.0/include
sudo cp cuda/lib64/libcudnn* /usr/local/cuda-12.0/lib64
sudo chmod a+r /usr/local/cuda-12.0/include/cudnn.h /usr/local/cuda-12.0/lib64/libcudnn*

sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_adv_train.so.8.9.6 /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudn
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_ops_infer.so.8.9.6  /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcud
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_cnn_train.so.8.9.6  /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcud
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_adv_infer.so.8.9.6  /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcud
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_ops_train.so.8.9.6  /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcud
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn_cnn_infer.so.8.9.6 /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudn
sudo ln -sf /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn.so.8.9.6  /usr/local/cuda-12.0/targets/x86_64-linux/lib/libcudnn.so.8
```

```
ldconfig
```

3-3. 설치 확인

```
nvidia-smi
nvcc -V
watch -n 0.1 nvidia-smi
```

4. rabbitmq-server docker-compose.yml

```
version : '3.8'

services:
  fastapi-ai:
    container_name: fastapi-rabbitmq
    image: rabbitmq-server
    command: uvicorn main:app --host="0.0.0.0" --port=8000
    ports:
      - 8000:8000
    expose:
      - "8000"
    volumes:
      - /home/ubuntu/fastapi-server/AI_model:/app/AI_model
    networks:
      - ai-net

networks:
  ai-net:
    external: true
```

5. ai-server docker-compose.yml

```
version : '3.8'

services:
  fastapi-ai:
    container_name: fastapi-ai
    image: ai-server
    command: uvicorn main:app --host="0.0.0.0" --port=7000
    environment:
      - NVIDIA_VISIBLE_DEVICES=0
    ports:
      - 7000:7000
    expose:
      - "7000"
    volumes:
      - /home/ubuntu/fastapi-server/AI_model:/app/AI_model
    networks:
      - ai-net
    deploy:
      resources:
        reservations:
          devices:
          - driver: nvidia
            capabilities: [gpu]
            count: 1

networks:
```

```
  ai-net:
    external: true
```

EC2 서버 세팅하기

Nignx + SSL을 이용한 HTTPS 세팅

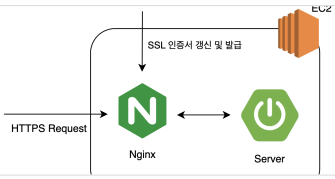
```
limit_req_zone $binary_remote_addr zone=api_rate_limit:10m rate=10r/s;

upstream frontend {
    server localhost:3000;
}
upstream backend {
    server localhost:8000;
}
server {
    server_name www.hanol.site hanol.site;
    location /api {
        limit_req zone=api_rate_limit burst=20 nodelay;
        proxy_pass http://backend;
        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;
    }
    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;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/hanol.site/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/hanol.site/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
    client_max_body_size 10M; # 클라이언트로부터 수락되는 요청의 최대 크기를 10MB로 제한

}

server {
    if ($host = www.hanol.site) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = hanol.site) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80;
    server_name www.hanol.site hanol.site;
    return 404; # managed by Certbot


}
```

참고자료

웹서버에 HTTPS 적용하기 (Let's Encrypt, Nginx, AWS EC2)

AWS에 올라간 웹서버에 HTTPS 적용하는 법을 알아보자

https://velog.io/@100journey/웹서버에-HTTPS-적용하기-Lets-Encrypt-Nginx-AWS-EC2



IoT

버전

- Python 3.11.2
- fastAPI 0.104.1
- webSockets 12.0
- Rasbian OS 32bit (lite)

카메라 서버 구축

```
sudo date MMDDHHMMYYYY (월일시분년)
apt-key list | grep -A4 "trusted.gpg$"
sudo apt-key export <apt-key> | sudo gpg --dearmor -o /tmp/raspi.gpg
file /tmp/raspi.gpg
sudo apt-key del <apt-key>
sudo mv /tmp/raspi.gpg /etc/apt/trusted.gpg.d/

sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED

pip3 install websockets
pip3 install fastAPI
sudo apt update
sudo apt upgrade
sudo apt install -y python3-picamera2 --no-install-recommends
```

ssh 접속(raspi 전원 on 한 후)

```
ssh song@192.168.162.252
password: hanol
```

스트리밍 서버 실행

```
python3 main.py
```

SSL 인증

```
https://192.168.162.252:8888
```

배포하기

Gitlab Runner 세팅하기

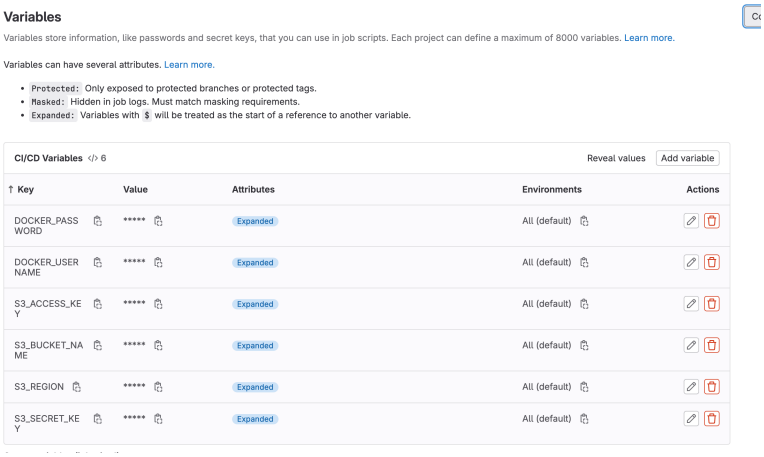
gitlab runner 설치하기

Gitlab runner 란?, AWS EC2 인스턴스 Gitlab Runner 등록하기

Gitlab Runner 란? Gitlab Runner 동작과정 AWS EC2 인스턴스를 생성해 Gitlab Specific Runner 등록하기 Gitlab Runner 란? Gitlab CI를 실행하기 위한 별도의 프로세스 Shared, Group, Specific 의 3가지 러너로 구성 가능 사용량에 따라 Runner의 Configration을 변경하여 사용 Gitlab이 구성되어 있는 Instance에는 Runner 구성 권장하지 않음

https://nearhome.tistory.com/140

환경변수 저장하기



빌드 시 필요한 파일을 EC2에 저장

EC2의 /home/gitlab-runner/config 위치에 application-dev.yml 등 환경변수 파일을 옮겨서 이를 활용함

.gitlab-ci.yml

```
stages:
  - build

variables:
  DOCKER_USERNAME: "$DOCKER_USERNAME"
  DOCKER_PASSWORD: "$DOCKER_PASSWORD"
  S3_ACCESS_KEY: "$S3_ACCESS_KEY"
  S3_SECRET_KEY: "$S3_SECRET_KEY"
  S3_BUCKET_NAME: "$S3_BUCKET_NAME"
  S3_REGION: "$S3_REGION"


# hanol api 서버 배포
```

```
build_server:
  stage: build
  script:
    - |
      # 버전 관리
      echo "Using downloaded application-dev.yml for the build..."  # Secure Files을 사용한 빌드 작업 수행
      echo "Deploying version $CI_COMMIT_TAG"

      # 파일 복사
      echo "스프링 서버 빌드에 필요한 파일 복사"  # Secure Files을 사용한 빌드 작업 수행
      cp ~/config/server/application-dev.yml ./BE/hanol/src/main/resources
      cp ~/config/server/firebase-sdk.json ./BE/hanol/src/main/resources
      ls -l BE/hanol/src/main/resources/

      # 도커 로그인
      echo "DOCKER USERNAME = $DOCKER_USERNAME"
      echo "DOCKER PASSWORD = $DOCKER_PASSWORD"
      echo "Login to Docker using username, password"
      docker login --username $DOCKER_USERNAME --password $DOCKER_PASSWORD

      container_name="hanol-server"
      image_name="hanol-server"

      # 빌드 시작
      echo "hanol server 이미지 빌드 시작"
      docker build --no-cache -t ${image_name}:latest --platform linux/amd64 ./BE/hanol/
      echo "hanol server 이미지 빌드 성공"

      echo "BE server container 시작"
      echo "$S3_BUCKET_NAME"
      echo "$S3_REGION"

      # container 실행 여부 확인
      if docker ps -a --format "{{.Names}}" | grep -q "$container_name"; then
        echo "'$container_name'가 실행중입니다. 컨테이너를 삭제합니다."
        docker stop ${container_name}
        docker rm ${container_name}
        sleep 2
      else
        echo "'$container_name'가 실행중이지 않습니다."
      fi

      echo "==================== 컨테이너 빌드 시작 ====================="
      docker run -d --name ${container_name} --network hanol-dev -p 8000:8000 -e PROFILE=dev -e S3_BUCKET_NAME=${S3_BUCKET_NAME} -e S3_REGI

      echo "==================== 새로운 서버 생성 성공 => 사용하지 않는 이미지 삭제====================="
      # yes | docker image prune -a

  tags:
    - hanol
  only:
    - release/server

# 클라이언트 배포
build_client:
  stage: build
  script:
    - |
      echo "Client test 배포 시작"

      # 파일 복사
      echo "스프링 서버 빌드에 필요한 파일 복사"  # Secure Files을 사용한 빌드 작업 수행
      cp ~/config/client/.env.prod ./FE/hanol/
      cat FE/hanol/.env.prod

      # 도커 로그인
      echo "DOCKER USERNAME = $DOCKER_USERNAME"
      echo "DOCKER PASSWORD = $DOCKER_PASSWORD"
      echo "Login to Docker using username, password"
      docker login --username $DOCKER_USERNAME --password $DOCKER_PASSWORD


      container_name="hanol-client"
      image_name="hanol-client"

      # Dockerfile을 사용하여 이미지 빌드
      docker build -t ${image_name}:latest --platform linux/amd64 ./FE/hanol
```

```
    # container 실행 여부 확인
    if docker ps --format "{{.Names}}" | grep -q "$container_name"; then
            echo "'$container_name'가 실행중입니다."
            docker stop ${container_name}
            docker rm ${container_name}
    else
            echo "'$container_name'가 실행중이지 않습니다."
    fi

    echo "==================== 컨테이너 빌드 시작 ====================="
    docker run -d --name ${container_name} -p 3000:3000 --network hanol-dev ${image_name}


tags:
  - hanol
only:
  - release/client
```