포팅 매뉴얼

목차

1. 사용 도구

- 형상 관리 : Gitlab, Git
- 커뮤니케이션 : Notion, MatterMost
- 와이어프레임 : Figma
- 이슈 관리 : Jira
- CI/CD : Docker, Jenkins, Docker-compose

2. 개발 도구

- IntelliJ IDEA: 2023.3.8
- Visual Studio Code : 1.97.2

3. 개발 환경

OS

- Ubuntu : 22.04.5

BackEnd

- Java OpenJDK : 23.0.2

- Spring Boot : 3.4.2
- Spring Security : 6.3.1.1
- Spring Data JPA : 3.4.2
- Gradle : 8.12.1
- JWT : 0.11.5
- Python : 3.10.12
- Apache Kafka : 3.6.1
- Fastapi : 0.115.8
- Rembg : 2.0.62
- Uvicorn : 0.34.0
- Spring WebSocket : 6.2.2
- Lombok : 1.18.36

FrontEnd

- JavaScript (ES6+)
- Node.js : 22.13.0
- React : 18.3.1
- Vite : 6.1.0
- Html2canvas : 1.4.1
- Stomp/stompjs (WebSocket) : 7.0.0
- Sockjs-client (WebSocket) : 1.6.1
- Axios : 1.7.9
- Chart.js : 4.4.7
- Swiper : 11.2.4
- React DnD : 18.0.1

Database

- MySQL : 8.0

- MongoDB : 6.0
- Redis : 7.0
- AWS S3

Infra

- AWS EC2
- Gitlab Webhook
- Docker : 26.1.3
- Docker-compose : 2.24.1
- Jenkins : 2.479.3
- Nginx : 1.18.0
- Certbot : 1.21.0

포트 정보

BackEnd 8081
FrontEnd 80
Jenkins 8080
Fastapi 8000
MongoDB 27017
MySQL 3306
Redis 6379
Apache Kafka 9092

4. 환경 변수

- **application.yml** (.gitignore로 보안 관리)

```
spring:
  application:
    name: pop4u
  datasource:
    url: jdbc:mysql://i12d105.p.ssafy.io:3306/pop4u?useSSL=false&serverTi
    username: newuser
    password: newpassword
```

```
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false
    properties:
      hibernate:
        format_sql: true
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: 320038735950-mv4hq6ainjih9mp1dansdsjsflulr45o.apps
            client-secret: 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
            scope:
              - email
              - profile
  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB
      max-request-size: 50MB

  data:
    redis:
      host: i12d105.p.ssafy.io
      port: 6379
    mongodb:
      uri: mongodb://pop4u:pop4u123@i12d105.p.ssafy.io:27017/pop4u

  kafka:
    bootstrap-servers: i12d105.p.ssafy.io:9092
    consumer:
      group-id: game-completion-group
      auto-offset-reset: latest
      key-deserializer: org.apache.kafka.common.serialization.StringDeseria
      value-deserializer: org.springframework.kafka.support.serializer.JsonD
```

```
      properties:
        spring.json.trusted.packages: "*"
    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.springframework.kafka.support.serializer.JsonSer



jwt:
  issuer: pop4u
  secretKey: d105_pop4ud105_pop4ud105_pop4ud105_pop4ud105_pop4u

cloud:
  aws:
    s3:
      bucket: pop4u
    stack.auto: false
    region.static: us-east-1
    credentials:
      accessKey: AKIAXZ2CK3QV4WQ7TVW5
      secretKey: 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

springdoc:
  swagger-ui:
    path: /api-test
    groups-order: DESC
    tags-sorter: alpha
    operations-sorter: method

server:
  port: 8081

logging:
  level:
    root: INFO
    org.springframework: WARN
    org.hibernate.SQL: ERROR
    org.hibernate: ERROR
```

```
    org.apache.kafka: DEBUG
    org.springframework.kafka: DEBUG
```

- **.env**

```
# GitLab 설정
GITLAB_TOKEN=xoMPfjRkdtS_oUKdVR1y

# EC2 배포 정보
EC2_HOST=i12d105.p.ssafy.io
EC2_USER=ubuntu

# Jenkins 설정
JENKINS_URL=http://i12d105.p.ssafy.io:8080
JENKINS_USER=admin
JENKINS_TOKEN=14e870936128d502012064bf01f28b1b

# MySQL
MYSQL_ROOT_PASSWORD=newpassword
MYSQL_DATABASE=pop4u
MYSQL_USER=newuser
MYSQL_PASSWORD=newpassword

# MongoDB
MONGO_INITDB_ROOT_USERNAME=pop4u
MONGO_INITDB_ROOT_PASSWORD=pop4u123
MONGO_INITDB_DATABASE=pop4u

# Spring
SPRING_PROFILES_ACTIVE=prod

# S3
AWS_ACCESS_KEY_ID=AKIAXZ2CK3QV4WQ7TVW5
AWS_SECRET_ACCESS_KEY=Rc5fGV+0LEmKIggmAQvNKQkLTOfv/wGX2r
AWS_DEFAULT_REGION=us-east-1
S3_BUCKET_NAME=pop4u
```

5. CI/CD 구축

기본 설정

AWS EC2 접속

- pem키 필요
- pem키 있는 디렉토리에서 터미널 실행

```
# ssh -i [pem키] [접속 계정]@[접속 도메인]
ssh -i I12D105T.pem ubuntu@i12d105.p.ssafy.io
```

업데이트

```
sudo apt update # 설치 가능한 패키지의 최신 목록 업데이트

sudo apt upgrade -y # 설치된 패키지를 최신 버전으로 업그레이드

sudo apt install -y build-essential # 필수 개발 도구 설치

# 패키지 관리자 업데이트
sudo apt-get update
sudo apt-get upgrade -y
```

ufw 포트 설정

```
sudo ufw status # ufw 상태 확인

sudo ufw allow [포트 번호] # 사용할 포트 허용

sudo ufw show added # 등록 포트 조회
```

Docker & Docker-compose

설치

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

# Docker 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

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

# Docker 엔진 설치
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# Docker 서비스 시작
sudo systemctl start docker
sudo systemctl enable docker

# 현재 사용자를 docker 그룹에 추가 (sudo 없이 docker 명령어 실행 가능)
sudo usermod -aG docker $USER

# Docker Compose 설치
sudo curl -L "https://github.com/docker/compose/releases/latest/download/d
sudo chmod +x /usr/local/bin/docker-compose

# 설치 확인
docker --version
docker-compose --version
```

```
# Docker 테스트
docker run hello-world
```

Docker-compose.yml

```
version: '3.8'

services:
  backend:
    build:
      context: ./backend/pop4u
      dockerfile: Dockerfile
    container_name: spring-backend
    ports:
      - "8081:8081"
    environment:
      - SPRING_PROFILES_ACTIVE=prod
    networks:
      - app-network
    depends_on:
      - mysql
      - redis
      - mongodb
      - fastapi

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: react-frontend
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt:ro
    networks:
      - app-network
```

```
fastapi:
  build:
    context: ./ai-server/life-four-cuts
    dockerfile: Dockerfile
  container_name: life-four-cuts
  ports:
    - "8000:8000"
  env_file:
    - ./ai-server/life-four-cuts/.env
  networks:
    - app-network

mysql:
  image: mysql:8.0
  container_name: mysql
  ports:
    - "3306:3306"
  environment:
    - MYSQL_ROOT_PASSWORD=newpassword
    - MYSQL_DATABASE=pop4u
    - MYSQL_USER=newuser
    - MYSQL_PASSWORD=newpassword
  volumes:
    - mysql-data:/var/lib/mysql
  networks:
    - app-network

redis:
  image: redis:alpine
  container_name: redis
  ports:
    - "6379:6379"
  volumes:
    - redis-data:/data
  networks:
    - app-network

mongodb:
```

```
    image: mongo:latest
    container_name: mongodb
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=pop4u
      - MONGO_INITDB_ROOT_PASSWORD=pop4u123
    volumes:
      - mongo-data:/data/db
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

volumes:
  mysql-data:
  redis-data:
  mongo-data:
```

Dockerfile

- **BackEnd**

```
# Java 23 버전
FROM eclipse-temurin:23-jdk-alpine

ENV JAVA_HOME /opt/java/openjdk
ENV PATH $PATH:$JAVA_HOME/bin
# 작업 디렉토리 설정
WORKDIR /app

# 환경 변수 설정
ENV JAVA_HOME=/opt/java/openjdk
ENV PATH $PATH:$JAVA_HOME/bin

# Gradle 빌드 파일 복사
```

```
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .

# 소스 코드 복사
COPY src ./src

# Gradle 빌드
RUN chmod +x gradlew
RUN ./gradlew build -x test

# JAR 파일을 실행
ENTRYPOINT ["java", "-jar", "/app/build/libs/pop4u-0.0.1-SNAPSHOT.jar"]
```

- **FrontEnd**

```
# Build Stage
FROM node:20-alpine AS build

WORKDIR /app

# package.json과 package-lock.json 복사
COPY package*.json ./

# 의존성 설치
RUN npm install

# 소스 코드 복사
COPY . .

# React 프로젝트 빌드
RUN npm run build

# Production Stage
FROM nginx:alpine

# React 라우팅을 위한 Nginx 설정
```

```
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf

# 빌드된 React 파일들을 Nginx로 복사
COPY --from=build /app/dist /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

- **FastAPI**

```
FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Jenkins 설정

Jenkins 컨테이너 실행

```
# Jenkins 볼륨 생성
docker volume create jenkins_home

# Jenkins 컨테이너 실행
docker run -d \
  --name jenkins-docker \
  --restart=unless-stopped \
  -p 8080:8080 \
  -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
```

```
  jenkins/jenkins:lts

# 초기 비밀번호 확인
docker exec jenkins-docker cat /var/jenkins_home/secrets/initialAdminPassw
```

Jenkins 초기 설정

- http://i12d105.p.ssafy.io:8080 접속
- 초기 관리자 비밀번호 입력
- Suggested Plugins 설치 선택
- 관리자 계정 생성

  ```
  ID : admin
  password : wjdrlahdla
  ```

- 필요한 플러그인 설치
  - Jenkins 관리 → Plugins → Available plugins

Credentials 설정

- `Jenkins 관리` → `Credentials` → `System` → `Global credentials` → `Add Credentials`

1. **GitLab Access Token**
   - GitLab에서 Access Token 생성
     - GitLab → User Settings → Access Tokens
     - Name : jenkins-access
     - Expiration date 설정
     - Scopes: api, read_api, read_repository, write_repository
     - Token 생성 후 값 복사 (생성 직후에만 볼 수 있음)
   - Jenkins에서 GitLab Token 등록
     - Kind : GitLab API token
     - Scope : Global

    - Token : GitLab에서 생성한 token 값 입력
    - ID : gitlab-token
    - Description : gitlab access

2. **EC2 SSH Key 등록**
    - Kind : SSH Username with private key
    - ID : ec2-ssh-key
    - Username : ubuntu
    - Private Key : I12D105T.pem (pem키)
    - Description : EC2 SSH Key
3. **환경 설정 파일**
    - application.yml
        - Kind : Secret file
        - ID : application-yml
        - File : application.yml 파일 (변경 시마다 업데이트)
        - Description : application.yml
    - .env
        - Kind : Secret file
        - ID : .env
        - File : .env 파일
        - Description : envfile

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	gitlab-token	GitLab API token (gitlab access)
		System	(global)	gitlab-credentials	jjmm1123/****** (gitlab access)
		System	(global)	ec2-ssh-key	ubuntu (EC2 SSH Key)
		System	(global)	application-yml	application.yml (application.yml)
		System	(global)	.env	.env (envfile)

Pipeline 생성

- `Jenkins` → `새로운 Item` → `Pipeline` → `구성`

1. **Github project**
   - Project url : https://lab.ssafy.com/s12-webmobile2-sub1/S12P11D105.git/
2. **Triggers**
   - GitLab webhook URL : http://i12d105.p.ssafy.io:8080/project/S12P11D105
   - Push Events, Opened Merge Request Events 설정
3. **Pipeline**
   - Definition : Pipeline script from SCM
   - SCM : Git
   - Repositories
     - Repository URL : https://lab.ssafy.com/s12-webmobile2-sub1/S12P11D105.git
     - Credentials : gitlab API token 선택
   - Branches to build
     - Branch Specifier (blank for 'any') :
       - */back_develop
       - */front_develop
       - BackEnd, FrontEnd 따로 빌드하기 위함
   - Script Path : Jenkinsfile

Jenkinsfile

```
pipeline {
    agent any

    environment {
        EC2_HOST = "i12d105.p.ssafy.io"
        WORKSPACE_PATH = "/var/jenkins_home/workspace/S12P11D105"
    }
```

```
stages {
    stage('Checkout') {
        steps {
            checkout scm
        }
    }

    stage('credentials download') {
        when {
            expression { env.GIT_BRANCH == 'origin/back_develop' }
        }
        steps {
            withCredentials([file(credentialsId: 'application-yml', variable: 'dbCo
                script {
                    sh 'cp -f $dbConfigFile backend/pop4u/src/main/resources/app
                }
            }
            withCredentials([file(credentialsId: '.env', variable: 'envFile')]) {
                script {
                    // 각 디렉토리에 .env 파일 복사
                    sh "cp -f $envFile backend/pop4u/.env"
                    sh "cp -f $envFile ai-server/life-four-cuts/.env"
                }
            }
        }
    }

    stage('Deploy') {
        steps {
            script {
                def deployBranch = ""

                if (env.GIT_BRANCH == 'origin/back_develop') {
                    deployBranch = 'backend'
                    containerName = 'spring-backend'

                } else if (env.GIT_BRANCH == 'origin/front_develop') {
```

```
        deployBranch = 'frontend'
        containerName = 'react-frontend'
    }

    // 디버깅
    echo "Current branch: ${env.GIT_BRANCH}"
    echo "Deploy branch: ${deployBranch}"
    echo "Current workspace: ${WORKSPACE}"

    if (deployBranch == 'frontend') {
        sshagent(['ec2-ssh-key']) {
            sh """
                ssh -o StrictHostKeyChecking=no ubuntu@\${EC2_HOST}
                    cd ~
                    rm -rf ${deployBranch} || true
                '
                scp -o StrictHostKeyChecking=no -r ${WORKSPACE}/front
                ssh -o StrictHostKeyChecking=no ubuntu@\${EC2_HOST}
                    cd ~
                    docker-compose stop ${deployBranch} || true
                    docker rm -f ${containerName} || true
                    docker-compose build --no-cache ${deployBranch}
                    docker-compose up -d --build ${deployBranch}
                '
            """
        }
    } else if (deployBranch == 'backend') {
        sshagent(['ec2-ssh-key']) {
            sh """
                ssh -o StrictHostKeyChecking=no ubuntu@\${EC2_HOST}
                    cd ~
                    rm -rf ${deployBranch} || true
                '
                scp -o StrictHostKeyChecking=no -r ${WORKSPACE}/* ub
                ssh -o StrictHostKeyChecking=no ubuntu@\${EC2_HOST}
                    cd ~
                    docker-compose stop ${deployBranch} fastapi || true
                    docker rm -f ${containerName} life-four-cuts || true
```

```
                        docker-compose build --no-cache ${deployBranch} fast
                        docker-compose up -d --build ${deployBranch} fastapi
                    '
                """
                }
            }
        }
    }
  }

  post {
    success {
      script {
        def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout:
        def Author_Name = sh(script: "git show -s --pretty=%ae", returnStd
        def Name = Author_ID.substring(1)
        mattermostSend (color: 'good',
        message: "${env.JOB_NAME}의 Jenkins ${env.BUILD_NUMBER}번째
        endpoint: 'https://meeting.ssafy.com/hooks/ciw46xyw1td98yepnryh
        channel: 'd105-ci-cd-alert'
        )
      }
    }
    failure {
      script {
        def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout:
        def Author_Name = sh(script: "git show -s --pretty=%ae", returnStd
        def Name = Author_ID.substring(1)
        mattermostSend (color: 'danger',
        message: "${env.JOB_NAME}의 Jenkins ${env.BUILD_NUMBER}번째
        endpoint: 'https://meeting.ssafy.com/hooks/ciw46xyw1td98yepnryh
        channel: 'd105-ci-cd-alert'
        )
      }
    }
  }
}
```

빌드 과정

1. GitLab의 back_develop, front_develop 브랜치에서 푸시/머지 될 때마다 젠킨스가 실행됩니다.
2. 각 브랜치별로 빌드가 각각 다르게 실행됩니다.
    - front_develop : docker-compose에서 FrontEnd Dockerfile 실행
    - back_develop : Jenkins credentials의 환경 변수 복사, docker-compose에서 BackEnd Dockerfile 실행 (의존성 있는 컨테이너도 동시 실행)
3. 빌드 완료 후 MatterMost로 빌드의 성공/실패를 알립니다.

Nginx 설정

- FrontEnd 디렉토리 내부에 위치하여 Dockerfile에서 함께 빌드되도록 함.

SSL 인증서 발급 (Certbot)

```
sudo apt-get update
sudo apt-get upgrade

# certbot 설치
sudo apt-get install python3-certbot-nginx

# SSL 인증서 발급
sudo certbot certonly --nginx -d i12d105.p.ssafy.io
```

- /etc/letsencrypt/renewal/i12d105.p.ssafy.io.conf 5개의 파일 확인
    - 4개의 pem, 1개의 README

```
ubuntu@ip-172-26-14-6:/etc/letsencrypt/renewal$ cat i12d105.p.ssafy.io.conf
# renew_before_expiry = 30 days
version = 1.21.0
archive_dir = /etc/letsencrypt/archive/i12d105.p.ssafy.io
cert = /etc/letsencrypt/live/i12d105.p.ssafy.io/cert.pem
privkey = /etc/letsencrypt/live/i12d105.p.ssafy.io/privkey.pem
chain = /etc/letsencrypt/live/i12d105.p.ssafy.io/chain.pem
fullchain = /etc/letsencrypt/live/i12d105.p.ssafy.io/fullchain.pem
```

Nginx.conf

```
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
```

```
}
server {
    listen 80;
    server_name i12d105.p.ssafy.io;
    return 301 https://$server_name$request_uri;
}

# HTTPS 서버 블록
server {
    listen 443 ssl;
    server_name i12d105.p.ssafy.io;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/i12d105.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i12d105.p.ssafy.io/privkey.pem;

    # SSL 설정 최적화
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    location / {
        root /usr/share/nginx/html;
        index index.html;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        rewrite ^/api/(.*) /$1 break;
        proxy_pass http://spring-backend:8081;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        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 /ws/ {
        proxy_pass http://spring-backend:8081/ws/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        proxy_set_header Host $host;
        proxy_hide_header X-Frame-Options;
        proxy_buffering off;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
}

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

  listen 80;
  server_name example.com;
    return 404; # managed by Certbot
}
```

https://i12d105.p.ssafy.io

Pop4U

6. DB 덤프

MySQL

Dump_pop4u.sql

MongoDB

pop4u.chat_messages.json

7. 외부 서비스 사용

- GPT API
- rembg API
- API 테스트 : Swagger, Postman