포팅매뉴얼

소유자	성문 성문
태그	

개발환경

Backend

- JVM : Liberica 17.0.13
- SpringBoot : 3.2.2
- IDE : IntelliJ IDEA
- Gradle

application-scret.yml

```
DB_URL: jdbc:mysql://{MySQL 컨테이너 이름}:{포트 번호}/{DB 명}?useSSL=fal
se&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
DB_PASSWORD: {DB 비밀번호}
REDIS_DEFAULT_PASSWORD: {Redis 비밀번호1}
REDIS_WEBSOCKET_PASSWORD: {Redis 비밀번호2}

DB_NAME: {DB 이름}
DB_USERNAME: {DB 유저명}
JWT_SECRETKEY: {JWT 토큰 생성 비밀키}

FRONT_URL: {서비스 프론트 URL}

SSAFY_CLIENT_ID: {싸피 클라이언트 아이디}
SSAFY_REDIRECT_URI: {싸피 리다이렉트 URL}
SSAFY_SECRET: {싸피 비밀키}
```

```
KAKAO_CLIENT_ID: {카카오 클라이언트 아이디}
KAKAO_REDIRECT_URI: {카카오 리다이렉트 URL}

PORTONE_API_KEY: {결제 포트원 API 키}
PORTONE_API_SECRET_KEY: {결제 포트원 비밀키}
```

Dockerfile

```
FROM openjdk:17-jdk-alpine

RUN apk add --no-cache tzdata

ENV TZ=Asia/Seoul

WORKDIR /app

COPY . .

RUN ./gradlew clean build -x test

RUN find /app -name "*SNAPSHOT.jar" -exec cp {} /app/app.jar \;

ENV SPRING_PROFILES_ACTIVE=prod

ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "app.jar"]
```

Frontend

- node : 22.13.0
- react : 19
- Vite: 6.2.0
- ESLint : 8
- Prettier : 3.4.2
- tailwindcss : 3.4.1
- typescript : 5

- Mock Service Worker : 2.7.3
- web socket + Stomp.js : 7.1.0
- IDE : Visual Studio Code

.env

```
VITE_SERVER_URL={백엔드 API 요청 서버 URL}
VITE_WEBSOCKET_URL={백엔드 소켓 서버 URL}

VITE_SSAFY_CLIENT_ID={싸피 클라이언트 아이디}
VITE_KAKAO_CLIENT_ID={카카오 클라이언트 아이디}

VITE_SSAFY_REDIRECT_URI={싸피 리다이렉트 URL}
VITE_KAKAO_REDIRECT_URI={카카오 리다이렉트 URL}

VITE_PORTONE_CODE={포트원 코드}
VITE_PORTONE_CHANNEL={포트원 채널}
```

Dockerfile

```
FROM node:22

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build

CMD ["npm", "run", "preview", "--", "--host", "0.0.0.0"]
```

AI

- python : 3.12.8
- IDE : Visual Studio Code

.env

```
OUTPUT_DIR =  "comfyUI 결과물 path"
WORKFLOW_PATH = "workflow.json path"
COMFYUI_IP = "comfyUI IP주소"
OPEN_AI_API_KEY = "OpenAI API 키"
```

- Dockerfile

```
FROM python:3.12.8

WORKDIR /code

RUN pip install --upgrade pip

COPY . /code
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
EXPOSE 8000

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

배포 시 특이사항

- MySQL과 Redis 미리 컨테이너로 실행되어 있고 같은 도커 네트워크에 있도록 설정
- 배포 시 도커와 도커 컴포즈를 활용해서 배포를 진행

```
services:
  frontend:
    container_name: grande-horse
    build:
      context: ./frontend/web
      dockerfile: Dockerfile
    ports:
      - "4173:4173"
    networks:
      - app-network
```

```
  backend:
    container_name: grande-horse-server
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      SPRING_PROFILES_ACTIVE: prod
    networks:
      - app-network

networks:
  app-network:
    external: true
    name: app-network
```

- Nginx를 활용하여 리버스 프록시를 구성합니다.

```
worker_processes auto;

events {
    worker_connections 1024;
}

http {
    server {
        listen 80;
        server_name j12a606.p.ssafy.io;
        return 301 https://$host$request_uri;
    }

    server {
        listen 443 ssl;
        server_name j12a606.p.ssafy.io;

        ssl_certificate /etc/letsencrypt/live/p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/p.ssafy.io/privkey.pem;
```

```
location /jenkins-server {
    proxy_pass http://localhost:9000;
    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;
    proxy_redirect off;
}

location /api/v1 {
    proxy_pass http://localhost:8080;
    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;
    proxy_redirect off;
}

location /api/v1/ws {
    proxy_pass http://localhost:8080;
    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;
    proxy_redirect off;
}


location / {
    proxy_pass http://localhost:4173;
    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;
        proxy_redirect off;
    }
  }
}
```

- Jenkins를 활용하여 CI/CD를 구축해 GitLab의 master 브랜치에 push가 되면 GitLabWebhook을 통해 자동으로 Jenkins 파이프라인을 거쳐 배포

```
pipeline {
  agent any

  environment {
    TARGET_BRANCH = 'master'
  }

  stages {
    stage('Check Branch') {
      when {
        expression {
          return env.BRANCH_NAME == "${TARGET_BRANCH}"
        }
      }
      steps {
        echo "Triggered by push to branch: ${env.BRANCH_NAME}"
      }
    }

    stage('Clone Repository') {
      steps {
        git branch: "${TARGET_BRANCH}",
          credentialsId: 'ACCESS_TOKEN',
          url: 'https://lab.ssafy.com/s12-bigdata-dist-sub1/S12P21A606.gi
t'
      }
    }

    stage('Prepare Credentials') {
```

```
        steps {
            parallel(
                BackendCredentials: {
                    withCredentials([file(credentialsId: 'BACKEND_SECRET_YML',
variable: 'BACKEND_SECRET')]) {
                        sh 'cp $BACKEND_SECRET backend/src/main/resources/a
pplication-secret.yml'
                    }
                },
                FrontendCredentials: {
                    withCredentials([file(credentialsId: 'FRONTEND_ENV', variabl
e: 'FRONTEND_ENV_FILE')]) {
                        sh 'cp $FRONTEND_ENV_FILE frontend/web/.env'
                    }
                }
            )
        }
    }

    stage('Docker Build & Push') {
        when {
            branch 'master'
        }
        steps {
            withCredentials([usernamePassword(credentialsId: 'DockerHub_L
ogin', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PA
SS')]) {
                sh """
                    echo \$DOCKER_PASS | docker login -u \$DOCKER_USER --
password-stdin
                    docker build --no-cache -t imkm/grandehorse:backend-lates
t ./backend
                    docker push imkm/grandehorse:backend-latest

                    docker build --no-cache -t imkm/grandehorse:frontend-lates
t ./frontend/web
                    docker push imkm/grandehorse:frontend-latest
```

```
                    docker logout
                """
            }
        }
    }

    stage('Deploy with Docker Compose') {
        steps {
            script {
                sh 'chmod +x ./backend/gradlew'
                sh 'docker-compose -f docker-compose.yml down || true'
                sh 'docker-compose -f docker-compose.yml build --no-cache'
                sh 'docker-compose -f docker-compose.yml up -d --remove-o
rphans'
            }
        }
    }
  }
}
```

DB 초기 설정

- JPA를 통해 DB 테이블 생성

```
INSERT INTO `grande_horse_db`.`cash_product` (
   `name`,
   `price`,
   `acquired_coin`,
   `selling`
)
VALUES
   ( '500코인',   500,   500, 1),
   ( '2500코인', 2500,  2500, 1),
   ( '5000코인', 5000,  5000, 1);

-- 데일리 카드팩 (1)
INSERT INTO cardpack_probability (cardpack_id, card_rank, probability) VA
LUES
```

```
(1, 'normal', 89.5),
(1, 'rare', 9.5),
(1, 'epic', 0.989),
(1, 'unique', 0.01),
(1, 'legend', 0.001);

-- 노말 카드팩 (2)
INSERT INTO cardpack_probability (cardpack_id, card_rank, probability) VA
LUES
(2, 'normal', 100.0);

-- 레어 카드팩 (3)
INSERT INTO cardpack_probability (cardpack_id, card_rank, probability) VA
LUES
(3, 'normal', 60.0),
(3, 'rare',   40.0);


-- 에픽 카드팩 (4)
INSERT INTO cardpack_probability (cardpack_id, card_rank, probability) VA
LUES
(4, 'normal', 30.0),
(4, 'rare',   40.0),
(4, 'epic',   30.0);


-- 유니크 카드팩 (5)
INSERT INTO cardpack_probability (cardpack_id, card_rank, probability) VA
LUES
(5, 'normal', 35.0),
(5, 'rare',   30.0),
(5, 'epic',   30.0),
(5, 'unique', 5.0);

INSERT INTO cardpack (name, card_count) VALUES
('데일리 카드팩', 6),
('노말 카드팩', 6),
('레어 카드팩', 6),
```

```
('에픽 카드팩', 6),
('유니크 카드팩', 6);

insert into product(type, type_id,price,selling) value('CARDPACK',1,0,1);
insert into product(type, type_id,price,selling) value('CARDPACK',2,100,1);
insert into product(type, type_id,price,selling) value('CARDPACK',3,300,1);
insert into product(type, type_id,price,selling) value('CARDPACK',4,500,1);
insert into product(type, type_id,price,selling) value('CARDPACK',5,1000,1);
```

프로젝트 시 사용한 외부 서비스

1. 카카오 소셜 로그인
2. 싸피 소셜 로그인
3. 포트원 결제
4. 공공 데이터 포탈 마사회 경기 조회 API