포팅 매뉴얼

사용 프로그램 버전

💡 **프로젝트에서 사용한 프로그램의 버전을 정리합니다.**

Backend

- Development Tool
    - JDK : 17.0.11
    - gradle : 8.8
- Framework
    - Spring boot : 3.3.2
        - Spring data JPA
        - Validation
    - Spring Doc: 2.0.2
    - OpenAI: 1.0.0-M1
    - Spring Cloud AWS: 2.2.6.RELEASE
- Security
    - JWT 0.12.3
    - Spring Security
    - OAuth2 Client
- Database
    - MySQL: 8.0.36
    - Spring Data Redis: 3.3.2

- Networking and Sockets
  - Netty-SocketIO: 2.0.11
- Utilities
  - Lombok: 1.18.34
  - OkHttp: 4.9.3
- Infra
  - Ubuntu: 20.04.6 LTS
  - Docker: 27.1.1
  - Docker Compose : 2.29.1
  - Nginx: 1.27.0
  - certbot: 0.40.0

Frontend

- axios: 1.7.2
- flowbite-react: 0.10.1
- ldrs: 1.0.2
- react: 18.3.1
- react-dom: 18.3.1
- react-modal: 3.16.1
- react-router-dom: 6.25.1
- react-speech-recognition: 3.10.0
- react-youtube: 10.1.0
- recoil: 0.7.7
- regenerator-runtime: 0.14.1
- socket.io-client: 4.7.5
- sweetalert2: 11.6.13
- swiper: 11.1.9

포트번호

💡 **서버구성에 사용된 포트번호를 정리합니다.**

SSH

- **포트 번호:** 22
- **역할:** Remote server management and access

MySQL (mysql-container)

- **포트 번호:** 3306
- **역할:** MySQL Database Server

Redis (redis-container)

- **포트 번호:** 6379
- **역할:** Redis Database Server

Backend (backend-container)

- **포트 번호:** 8080
- **역할:** Spring Boot Application Server
- **포트 번호:** 9093
- **역할:** Netty Socket Server

Frontend (frontend-container)

- **포트 번호:** 5173
- **역할:** React Front Application

Nginx (nginx-container)

- **포트 번호:** 80, 443
- **역할:** HTTP, HTTPS Web Server, Reverse Proxy

```
To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
8989                       ALLOW       Anywhere
443                        ALLOW       Anywhere
80                         ALLOW       Anywhere
8080/tcp                   ALLOW       Anywhere
5173/tcp                   ALLOW       Anywhere
3306                       ALLOW       Anywhere
9093                       ALLOW       Anywhere
9093/tcp                   ALLOW       Anywhere
6379                       ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
8989 (v6)                  ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
8080/tcp (v6)              ALLOW       Anywhere (v6)
5173/tcp (v6)              ALLOW       Anywhere (v6)
3306 (v6)                  ALLOW       Anywhere (v6)
9093 (v6)                  ALLOW       Anywhere (v6)
9093/tcp (v6)              ALLOW       Anywhere (v6)
6379 (v6)                  ALLOW       Anywhere (v6)
```

배포환경

💡 **서버구성에 사용된 코드를 정리합니다.**

네트워크 구성

- public IP adress

```
curl ifconfig.me
3.34.43.113
```

- private IP address

```
hostname -I
172.26.3.85
```

- Docker bridge network interface IP address

```
docker network inspect s11p12c106_dreamong-network

[
    {
        "Name": "s11p12c106_dreamong-network",
        "Id": "c61030aa74a34fb19ba3831214e850e9355dc553f4857718c1baa39e6bfe8e53",
        "Created": "2024-08-15T18:08:12.965302466Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "06c1436dadd639efb89038d5c666b8300ab3b5b9d312a1168784141e32c8f26a": {
                "Name": "nginx-container",
                "EndpointID": "6936f3f7b01def5ad25339fe9e77dbd5a3c3c09cc7c473b4d8611dc46ed903
                "MacAddress": "02:42:ac:12:00:06",
                "IPv4Address": "172.18.0.6/16",
                "IPv6Address": ""
            },
            "27692e20ed9a8a68d7efa6922288d3c283043451be1bb85cc6ceedf53b521542": {
                "Name": "frontend-container",
                "EndpointID": "f0f74de3c963a1d12d0220d99e1caf7db4bf964ce34388a0e968f6b12d624c
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "6d9154b42cc5b4dce4d256768b3d5e40e26d94e62ab83a10b4a035bd218840d6": {
                "Name": "redis-container",
                "EndpointID": "3cd2d4b4a664b29bc0c44661419284ecdd62dc05b45c2f10bacde651f407e8
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
            },
            "82b9bc86f5fa91a2818c50e3e6f627974642fcbc4fa71918683c4530b817f205": {
                "Name": "mysql-container",
                "EndpointID": "2082b93a91d2a0389bd8418046fab3f664e80b115bd9767e4b112bf19241e6
```

```
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "956519ef9045ed6f64cbb5b6c91c6f0b0e33dd490487e0e4fe6a33b3530337e2": {
                "Name": "backend-container",
                "EndpointID": "7790aa5e0b1c11a5973b2d1d5239743667de23ae42901f5fedf5c3e22d3c5b
                "MacAddress": "02:42:ac:12:00:05",
                "IPv4Address": "172.18.0.5/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "dreamong-network",
            "com.docker.compose.project": "s11p12c106",
            "com.docker.compose.version": "2.29.1"
        }
    }
]
```

서버환경 설정

서버 방화벽 설정

```
# ufw 설치명령
sudo apt-get install ufw

# ufw 상태확인 명령
sudo ufw status verbose
sudo ufw status

sudo ufw allow 22 # ssh
sudo ufw enable

# ...
sudo ufw status
```

Docker 설정

1. 기존 Docker 설치 제거 (이미 설치된 경우)

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

2. Docker 설치에 필요한 패키지를 설치

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

3. Docker 공식 GPG 키 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/k
eyrings/docker.gpg
```

4. Docker 리포지토리 설정

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://
download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Docker 설치

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-com
pose-plugin
```

6. Docker 버전 확인

```
docker --version
```

- etc) Docker를 `sudo` 없이 실행하기 위해 현재 사용자를 `docker` 그룹에 추가

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

- etc) 이후, 시스템을 재부팅하거나 다음 명령어로 그룹 변경을 반영

```
exec $SHELL
```

Docker Compose 설치

1. Docker Compose 다운로드

```
sudo curl -L "https://github.com/docker/compose/releases/download/v2.29.1/docker-compose
-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Docker Compose에 실행 권한 부여

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

3. Docker Compose 버전 확인

```
docker-compose --version
```

Backend 설정

1. JDK 설치

```
# JDK 17 설치
sudo apt-get update
sudo apt-get install openjdk-17-jdk
```

```
# JDK 버전 확인
java -version
```

2. Gradle 설치

```
# Gradle 설치
sudo apt-get update
sudo apt-get install gradle

# Gradle 버전 확인
gradle -v
```

3. 프로젝트 디렉토리로 이동하여 의존성 설치 및 빌드

```
cd ~/S11P12C106/backend/dreamong
./gradlew clean build
```

Frontend 설정

1. nvm 설치

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
source ~/.bashrc
```

2. Node.js 20.15 버전 설치 및 사용

```
nvm install 20.15.0
nvm use 20.15.0

npm install -g npm@latest
```

3. 프로젝트 디렉토리로 이동하여 의존성 설치 및 빌드

```
cd ~/S11P12C106/frontend/dreamong
npm install
npm run build
```

배포방법

```
docker compose down -v
docker compose build --no-cache
docker compose up -d
```

설정 파일

nginx

- nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
```

```
events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        ##

        gzip on;

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/xml app

        ##
        # Virtual Host Configs
        ##
        include /etc/nginx/sites-available/*;
        include /etc/nginx/conf.d/*.conf;
```

```
        #include /etc/nginx/sites-enabled/*;
}
```

- default

```
resolver 127.0.0.11 valid=30s;

# HTTP 요청을 HTTPS로 리디렉션하는 서버 블록
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name i11c106.p.ssafy.io;

    # 모든 HTTP 요청을 HTTPS로 리디렉션
    return 301 https://$host$request_uri;
}

# SSL을 사용한 HTTPS 서버 블록
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name i11c106.p.ssafy.io;

    # SSL 인증서 경로 및 키
    ssl_certificate /etc/letsencrypt/live/i11c106.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i11c106.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location = /firebase-messaging-sw.js {
        root /app/dist;  # 빌드된 정적 파일이 위치한 경로
        default_type application/javascript;
        types { }
        add_header Service-Worker-Allowed "/";
        add_header Cache-Control "no-cache";
#       try_files $uri =404;
    }

    # 백엔드 애플리케이션 프록시 설정
    location /api {
        proxy_pass http://backend-container:8080;
        proxy_http_version 1.1;
        proxy_set_header Host $http_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_buffer_size          128k;
        proxy_buffers              4 256k;
        proxy_busy_buffers_size    256k;
    }

    # 프론트엔드 애플리케이션 프록시 설정
    location / {
        proxy_pass http://frontend-container:5173;
        proxy_http_version 1.1;
        proxy_set_header Host $http_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;
#       try_files $uri $uri/ /index.html;  # 프론트엔드 SPA 지원

        proxy_buffer_size          128k;
        proxy_buffers              4 256k;
        proxy_busy_buffers_size    256k;

    }

    # WebSocket 프록시 설정
    location /socket.io/ {
                proxy_pass http://backend-container:9093;
        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;

        # WebSocket에 대한 버퍼 설정
        proxy_buffer_size 128k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;

        # WebSocket 연결 시 timeout 설정
        proxy_read_timeout 60s;
        proxy_send_timeout 60s;
        proxy_connect_timeout 60s;

        # WebSocket 특화 설정 추가
        proxy_buffering off;  # WebSocket 연결시 버퍼 종료
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Ssl on;
     }


    # 추가 보안 헤더
    add_header X-Frame-Options DENY;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
}
```

YML 설정파일

- application.yml

```
spring:
  application:
    name: # 애플리케이션의 이름

  sql:
    init:
      mode: # 데이터베이스 초기화 모드
```

```
    continue-on-error: # SQL 스크립트 오류 시 계속할지 여부

jpa:
  hibernate:
    ddl-auto: # 데이터베이스 스키마 자동 생성/업데이트 모드
    naming:
      physical-strategy: # JPA 물리적 네이밍 전략 클래스
  properties:
    hibernate:
      format_sql: # SQL 포맷팅 여부
    default_batch_fetch_size: # 기본 배치 페치 사이즈
    open-in-view: # Open EntityManager in View 설정

jwt:
  secret: # JWT 서명에 사용되는 비밀 키
  access-token-expiration: # 액세스 토큰의 만료 시간
  refresh-token-expiration: # 리프레시 토큰의 만료 시간

  ai:
    openai:
      api-key: # OpenAI API 키
      chat:
        options:
          model: # OpenAI 모델 이름

cloud:
  aws:
    s3:
      bucket: # AWS S3 버킷 이름
    credentials:
      access-key: # AWS 액세스 키
      secret-key: # AWS 시크릿 키
    region:
      static: # 고정된 리전 이름
      auto: # 자동으로 리전 선택 여부
    stack:
      auto: # 자동 스택 관리 여부

novita:
  api:
    key: # novita API key

clova:
  api:
    url: # 클로바 API URL
    secret: # 클로바 API 비밀 키

deepl:
  api:
    url: # Deepl API URL
    key: # Deepl API key

logging.level:
  org.hibernate.SQL: # Hibernate SQL 로그 레벨
  org.springframework.web: # Spring Web 로그 레벨
```

- application-local.yml

```yaml
socketio:
  server:
    hostname: # 소켓 서버의 호스트명
    port: # 소켓 서버의 포트 번호

login:
  callback-url: # 로그인 콜백 URL

spring:
  datasource:
    url: # 데이터베이스 연결 URL
    username: # 데이터베이스 사용자명
    password: # 데이터베이스 비밀번호
    driver-class-name: # 데이터베이스 드라이버 클래스 이름

  data:
    redis:
    host: # Redis 서버 호스트명
    port: # Redis 서버 포트 번호
    repositories:
      enabled: # Redis 리포지토리 활성화 여부

security:
  oauth2:
    client:
      registration:
        naver:
          scope: # 네이버 OAuth2 스코프
          client-name: # 클라이언트 이름
          client-id: # 네이버 클라이언트 ID
          client-secret: # 네이버 클라이언트 시크릿
          redirect-uri: # 네이버 리다이렉트 URI
          authorization-grant-type: # 권한 부여 유형
        google:
          scope: # 구글 OAuth2 스코프
          client-name: # 클라이언트 이름
          client-id: # 구글 클라이언트 ID
          client-secret: # 구글 클라이언트 시크릿
          redirect-uri: # 구글 리다이렉트 URI
          authorization-grant-type: # 권한 부여 유형
        kakao:
          scope: # 카카오 OAuth2 스코프
          client-name: # 클라이언트 이름
          client-id: # 카카오 클라이언트 ID
          # client-secret: # 카카오 클라이언트 시크릿 (카카오는 선택 사항)
          redirect-uri: # 카카오 리다이렉트 URI
          authorization-grant-type: # 권한 부여 유형

      provider:
        kakao:
          user-name-attribute: # 사용자 이름 속성
          user-info-uri: # 사용자 정보 URI
          authorization-uri: # 권한 부여
          token-uri: # 토큰 URI
        naver:
           user-name-attribute: # 사용자 이름 속성
          user-info-uri: # 사용자 정보 URI
```

```
            authorization-uri: # 권한 부여
            token-uri: # 토큰 URI
```

- application-prod.yml

```
server:
  servlet:
    context-path: # 애플리케이션의 컨텍스트 경로

socketio:
  server:
    hostname: # 소켓 서버의 호스트명
    port: # 소켓 서버의 포트 번호

login:
  callback-url: # 로그인 콜백 URL

spring:
  datasource:
    url: # 데이터베이스 연결 URL
    username: # 데이터베이스 사용자명
    password: # 데이터베이스 비밀번호
    driver-class-name: # 데이터베이스 드라이버 클래스 이름

  data:
    redis:
      host: # Redis 서버 호스트명
      port: # Redis 서버 포트 번호
      repositories:
        enabled: # Redis 리포지토리 활성화 여부

  security:
    oauth2:
      client:
        registration:
          naver:
            scope: # 네이버 OAuth2 스코프
            client-name: # 클라이언트 이름
            client-id: # 네이버 클라이언트 ID
            client-secret: # 네이버 클라이언트 시크릿
            redirect-uri: # 네이버 리다이렉트 URI
            authorization-grant-type: # 권한 부여 유형
          google:
            scope: # 구글 OAuth2 스코프
            client-name: # 클라이언트 이름
            client-id: # 구글 클라이언트 ID
            client-secret: # 구글 클라이언트 시크릿
            redirect-uri: # 구글 리다이렉트 URI
            authorization-grant-type: # 권한 부여 유형
          kakao:
            scope: # 카카오 OAuth2 스코프
            client-name: # 클라이언트 이름
            client-id: # 카카오 클라이언트 ID
            # client-secret: # 카카오 클라이언트 시크릿 (카카오는 선택 사항)
            redirect-uri: # 카카오 리다이렉트 URI
            authorization-grant-type: # 권한 부여 유형

        provider:
```

```
            kakao:
              user-name-attribute: # 사용자 이름 속성
              user-info-uri: # 사용자 정보 URI
              authorization-uri: # 권한 부여
              token-uri: # 토큰 URI
            naver:
               user-name-attribute: # 사용자 이름 속성
              user-info-uri: # 사용자 정보 URI
              authorization-uri: # 권한 부여
              token-uri: # 토큰 URI
```

- .env

```
MYSQL_ROOT_PASSWORD=<your_mysql_root_password>
MYSQL_DATABASE=<your_database_name>
MYSQL_USER=<your_database_user>
MYSQL_PASSWORD=<your_database_password>
```

- Dokcerfile - BE

```
# 베이스 이미지 선택
FROM openjdk:17-jdk-slim

# 작업 디렉토리 설정
WORKDIR /app

# 소스코드 복사
COPY . .

# 빌드 실행
RUN ./gradlew clean build

# 빌드된 JAR 파일 복사
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar

# 환경 변수 설정
ENV SPRING_PROFILES_ACTIVE=prod

# 애플리케이션 실행
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- Dockerfile - FE

```
# 1단계: React 앱 빌드
FROM node:20.15-alpine AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json 및 package-lock.json 파일 복사
COPY package*.json ./

# 종속성 설치
RUN npm install
```

```
# 애플리케이션 코드 복사
COPY . .

# React 앱 빌드
RUN npm run build

# dist 디렉토리의 내용 확인
RUN ls -la /app/dist

# Stage 2: Node.js 서버로 React 앱 제공
FROM node:20.15-alpine

# 작업 디렉토리 설정
WORKDIR /app

# serve 패키지 전역 설치
RUN npm install -g serve

# 이전 단계에서 빌드된 출력물 복사
COPY --from=build /app/dist /app/dist

# 포트 5173 노출
EXPOSE 5173

# 서버 시작
CMD ["serve", "-s", "dist", "-l", "5173"]
```

- docker-compose.yml

```
services:
  mysql:
    image: mysql:latest
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - dreamong-network
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
      interval: 30s
      timeout: 10s
      retries: 5

  redis:
    image: redis:latest
    container_name: redis-container
    ports:
      - "6379:6379"
    networks:
      - dreamong-network
```

```
    volumes:
      - redis-data:/data
    restart: always

  backend:
    build:
      context: ./backend/dreamong
    container_name: backend-container
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql-container:3306/${MYSQL_DATABASE}?useSSL=false
      SPRING_DATASOURCE_USERNAME: ${MYSQL_USER}
      SPRING_DATASOURCE_PASSWORD: ${MYSQL_PASSWORD}
    depends_on:
      - mysql
      - redis
    ports:
      - "8080:8080"
      - "9093:9093"
    networks:
      - dreamong-network
    restart: on-failure

  frontend:
    build:
      context: ./frontend/dreamong
    container_name: frontend-container
    volumes:
      - frontend-dist:/app/dist
    ports:
      - "5173:80"
    networks:
      - dreamong-network

  nginx:
    image: nginx:latest
    container_name: nginx-container
    depends_on:
      - backend
      - frontend
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /etc/nginx/nginx.conf:/etc/nginx/nginx.conf
      - /etc/nginx/sites-available:/etc/nginx/sites-available
      - /etc/nginx/sites-enabled:/etc/nginx/sites-enabled
      - /etc/letsencrypt:/etc/letsencrypt
      - frontend-dist:/app/dist
    networks:
      - dreamong-network
    restart: always

volumes:
  mysql-data:
  redis-data:
  frontend-dist:
```

```
networks:
  dreamong-network:
```

- schema.sql

```
-- 외래 키 제약 조건을 비활성화
SET FOREIGN_KEY_CHECKS = 0;

-- 테이블 삭제
DROP TABLE IF EXISTS dream_category;
DROP TABLE IF EXISTS comment_like;
DROP TABLE IF EXISTS comment;
DROP TABLE IF EXISTS notification;
DROP TABLE IF EXISTS room;
DROP TABLE IF EXISTS dream;
DROP TABLE IF EXISTS category;
DROP TABLE IF EXISTS users;

-- 테이블 생성
CREATE TABLE category (
                          category_id INTEGER NOT NULL AUTO_INCREMENT,
                          word VARCHAR(255),
                          type ENUM('character', 'dreamType', 'location', 'mood', 'objects'),
                          PRIMARY KEY (category_id)
) ENGINE=InnoDB;

CREATE TABLE users (
                       user_id INTEGER NOT NULL AUTO_INCREMENT,
                       created_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                       last_modified_date DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_T
                       email VARCHAR(100) NOT NULL,
                       fcm_token VARCHAR(255),
                       name VARCHAR(255) NOT NULL,
                       nickname VARCHAR(255),
                       provider_user_id VARCHAR(255) NOT NULL,
                       refresh_token VARCHAR(255),
                       role ENUM('ADMIN', 'MEMBER') NOT NULL,
                       PRIMARY KEY (user_id)
) ENGINE=InnoDB;

CREATE TABLE dream (
                       dream_id INTEGER NOT NULL AUTO_INCREMENT,
                       is_shared BOOLEAN,
                       user_id INTEGER,
                       created_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                       last_modified_date DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_T
                       image VARCHAR(255),
                       summary VARCHAR(255) NOT NULL,
                       write_time VARCHAR(255) NOT NULL,
                       content TEXT NOT NULL,
                       interpretation TEXT,
                       PRIMARY KEY (dream_id)
) ENGINE=InnoDB;

CREATE TABLE comment (
```

```
                            comment_id INTEGER NOT NULL AUTO_INCREMENT,
                            dream_id INTEGER,
                            likes_count INTEGER,
                            user_id INTEGER,
                            content VARCHAR(255),
                            PRIMARY KEY (comment_id)
) ENGINE=InnoDB;

CREATE TABLE comment_like (
                                comment_id INTEGER,
                                comment_like_id INTEGER NOT NULL AUTO_INCREMENT,
                                user_id INTEGER,
                                PRIMARY KEY (comment_like_id)
) ENGINE=InnoDB;

CREATE TABLE dream_category (
                                  category_id INTEGER,
                                  dream_category_id INTEGER NOT NULL AUTO_INCREMENT,
                                  dream_id INTEGER,
                                  PRIMARY KEY (dream_category_id)
) ENGINE=InnoDB;

CREATE TABLE room (
                        room_id INTEGER NOT NULL AUTO_INCREMENT,
                        user_id INTEGER,
                        created_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                        last_modified_date DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TI
                        youtube_link VARCHAR(1000) NOT NULL,
                        thumbnail VARCHAR(255) NOT NULL,
                        title VARCHAR(255) NOT NULL,
                        PRIMARY KEY (room_id)
) ENGINE=InnoDB;

CREATE TABLE notification (
                                id INTEGER NOT NULL AUTO_INCREMENT,
                                sent BOOLEAN,
                                user_id INTEGER,
                                created_date DATETIME DEFAULT CURRENT_TIMESTAMP,
                                last_modified_date DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CU
                                schedule_time DATETIME,
                                notification_type ENUM('MORNING_WAKEUP_REMINDER', 'SLEEP_REMINDER
                                PRIMARY KEY (id)
) ENGINE=InnoDB;

-- 외래 키 제약 조건 추가 및 인덱스 생성
ALTER TABLE users
    ADD CONSTRAINT UK_provider_user_id UNIQUE (provider_user_id);

ALTER TABLE dream
    ADD CONSTRAINT FK_dream_user_id
        FOREIGN KEY (user_id) REFERENCES users (user_id)
            ON DELETE CASCADE;

ALTER TABLE comment
    ADD CONSTRAINT FK_comment_dream_id
        FOREIGN KEY (dream_id) REFERENCES dream (dream_id)
            ON DELETE CASCADE,
    ADD CONSTRAINT FK_comment_user_id
```

```
        FOREIGN KEY (user_id) REFERENCES users (user_id)
            ON DELETE CASCADE;

ALTER TABLE comment_like
    ADD CONSTRAINT FK_comment_like_comment_id
        FOREIGN KEY (comment_id) REFERENCES comment (comment_id)
            ON DELETE CASCADE,
    ADD CONSTRAINT FK_comment_like_user_id
        FOREIGN KEY (user_id) REFERENCES users (user_id)
            ON DELETE CASCADE;

ALTER TABLE dream_category
    ADD CONSTRAINT FK_dream_category_category_id
        FOREIGN KEY (category_id) REFERENCES category (category_id),
    ADD CONSTRAINT FK_dream_category_dream_id
        FOREIGN KEY (dream_id) REFERENCES dream (dream_id)
        ON DELETE CASCADE;

ALTER TABLE notification
    ADD CONSTRAINT FK_notification_user_id
        FOREIGN KEY (user_id) REFERENCES users (user_id);

ALTER TABLE room
    ADD CONSTRAINT FK_room_user_id
        FOREIGN KEY (user_id) REFERENCES users (user_id);

-- 외래 키 제약 조건을 다시 활성화
SET FOREIGN_KEY_CHECKS = 1;
```