



포팅 메뉴얼

1. 개발 환경 및 기술 스택
2. 설정 파일 및 환경 변수 정보
3. 빌드 및 배포
 - 1) Docker + Docker compose 설치
 - 2) Jenkins 컨테이너 실행
 - 3) docker compose를 통한 실행
 - 5) NGINX 설정
4. 외부 서비스 및 정보
5. DB dump 설명

1. 개발 환경 및 기술 스택

- Backend
 - JAVA 17 (17.0.9 2023-10-17 LTS)
 - Spring boot 3.2.3
 - Lombok
 - openAPI (Swagger 3.0)
 - OAUTH 2.0
 - JWT
 - Spring Security
 - Spring Data JPA
 - QueryDSL
 - WebFlux
 - WebSocket
 - MySQL 8.0.36
 - Redis

- Firebase Storage
- RabbitMQ
- Jasypt (Java Simplified Encryption)
- Java Mail Sender
- Flyway
- OpenAI API (GPT)
- Python 3.8.10
- FastAPI 0.110.0
- Frontend
 - React
 - Typescript
 - axios
 - zustand
 - React Query
 - MUI
 - Styled Components
 - react-router-dom
 - AWS Polly
 - react-speech-recognition
 - STOMP
 - ApexCharts
- Infra
 - Ubuntu 20.04.6 LTS
 - Ngin
 - Jenkins 2.448
 - docker 25.0.4
- Data
 - selenium 4.19.0

- Hadoop 3.3.6
- Crontab
- Mecab-python3
- scikit-learn 1.4.1
- webdriver-manager 4.0.1
- googletrans 4.0.0rc1
- requests 2.31.0
- paramiko 3.4.0
- pytz 2024.1
- unidic-lite
- SQLAlchemy 2.0.28
- SQLAlchemy_Utils 0.41.1
- IDE
 - IntelliJ Ultimate 2023.3.6
 - Vscode 1.85.1

2. 설정 파일 및 환경 변수 정보

1. Spring boot

- application-local.yml (로컬용)

```
server:
  base-url: localhost

spring:
  servlet:
    # file 업로드 관련 세팅 (명시적으로 설정 안할 시 Spring boot는
    multipart:
      max-file-size: 10MB # 최대 파일 크기
      max-request-size: 10MB # 최대 요청 크기

  jpa:
    open-in-view: false
```

```

defer-datasource-initialization: false # flyway 관련
generate-ddl: false
hibernate:
    ddl-auto: none # ddl 자동 작성 여부
properties:
    hibernate:
        format_sql: true # 하이버네이트가 실행한 SQL
        use_sql_comments: true
        show_sql: true # 하이버네이트가 실행한 SQL
    jdbc:
        batch_size: 100 # insert/update 쿼리
        default_batch_fetch_size: 100

datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver # DB 드라이버
    url: jdbc:mysql://${server.base-url}:3306/talkydoki?useSSL=false
    username: ENC(oIucHTMgwwzUiBqlBzyB9g==)
    password: ENC(nA/sTfx2u8plMWgVt5MQ3A==)

# data_테이블명.sql 관련 실행 setting
sql:
    init:
        mode: always
        data-locations:
            - 'classpath:/FOREIGN_KEY_CHECKS_0.sql' # 외래키 생성
            - 'classpath:/db_backup.sql' # DB 백업
            - 'classpath:/FOREIGN_KEY_CHECKS_1.sql' # 외래키 삭제

# NoSQL setting
data:
    # Redis setting
    redis:
        host: ${server.base-url}
        port: 6379

# rabbitMQ setting
rabbitmq:

```

```

    host: ${server.base-url}
    port: 15672
    username: guest
    password: guest

# Java Mail Sender setting (Google Mail)
mail:
    host: smtp.gmail.com
    port: 587
    username: ENC(9zcJxzMJjdfsV9707BJQvAtUCYHyAV12FmzBkDqp
    password: ENC(bad9dzuo4Yv1McadA8E0Sn13JEE1/6UeyvsknQfC
    properties:
        mail:
            smtp:
                auth: true
                starttls:
                    enable: true

flyway:
    enabled: false # Flyway 활성화

# jwt setting
jwt:
    accessKey: ENC(s0vsxAuXm1zr6DjkUVLr1IS+MdTML5ar0QyKWar2q
    refreshKey: ENC(gE+DTAeZ7HD32Rd6u5HcfJu+bg4kgdULucqZWVJC
    accessExpiration: PT420M # 420분 (PT420M)
    refreshExpiration: PT10080M # 10080분 (7일) (PT10080M)

# log 관리
logging:
    level:
        org.hibernate:
            type.descriptor.sql: trace
            org.hibernate.SQLQuery: debug

# OAUTH2.0 Setting
oauth:
    kakao:

```

```

client-id: ENC(BWjHj+A/SJQBzhtK+vutMYiLy0sKAo2lQwFqW2P
client-secret: ENC(BWjHj+A/SJQBzhtK+vutMYiLy0sKAo2lQwF
redirect-uri: http://${server.base-url}:5173/member/lo
scope:
  - profile_nickname
  - profile_image
  - account_email
  - name

naver:
  client-id: ENC(yyBV/ueEBgJtvkDlLenjkPwTcYy1AYXQuIDYlVx
  client-secret: ENC(4zsZi9kyz8gJf75VByLo4PtdGuq9iqfS)
  redirect_uri: http://${server.base-url}:5173/member/lo
  scope:
    - nickname
    - name
    - email
    - profile_image

google:
  client_id: ENC(rg6zRGzhH2As0ap38BB/G74FEw8/Q1aH0PqsEdA
  client_secret: ENC(i1901JCsABZV+KTXIz0B5CT4Vu03YMEPd7J
  redirect_uri: http://${server.base-url}:5173/member/lo
  scope:
    - profile
    - email

# firebase setting
app:
  firebase-configuration-file: classpath:serviceAccountKey
  firebase-bucket: ENC(web/1dqCEyq3z3sEPNyoxYLJJD87LW/MD7n

# gpt-api setting
openai:
  key: ENC(ERakHHzDWQNw7FjB5uddrBvQOp1I3lkdduXwNNo6NEzVAc8

```

- application-dev.yml (배포용)

```

server:
  base-url: j10c107.p.ssafy.io
  https-url: https://j10c107.p.ssafy.io

spring:
  servlet:
    # file 업로드 관련 세팅 (명시적으로 설정 안할 시 Spring boot는
    multipart:
      max-file-size: 10MB # 최대 파일 크기
      max-request-size: 10MB # 최대 요청 크기

  jpa:
    open-in-view: false
    defer-datasource-initialization: false # flyway 관련
    generate-ddl: false
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        format_sql: true # 하이버네이트가 실행한 SQL
        use_sql_comments: true
        show_sql: true # 하이버네이트가 실행한 SQL
      jdbc:
        batch_size: 100 # insert/update 쿼리
        default_batch_fetch_size: 100

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver # DB 드라이버
    url: jdbc:mysql://${server.base-url}:3306/talkydoki?useSSL=false
    username: ENC(f0r0VaK09iSvSITZnTanow==)
    password: ENC(pcq3fN0xzVJNEAPYy/U+mQ==)

# data_테이블명.sql 관련 실행 setting
# sql:
#   init:
#     mode: never
#     data-locations:
#       - 'classpath:/data_vocabulary.sql'

```

```

#         - 'classpath:/data_news.sql'
#         - 'classpath:/data_member.sql'
#         - 'classpath:/data_keyword.sql'
#         - 'classpath:/data_news_keyword_mapping.sql'

# NoSQL setting
data:
  # Redis setting
  redis:
    host: ${server.base-url}
    port: 6379

# rabbitMQ setting
rabbitmq:
  host: ${server.base-url}
  port: 15672
  username: guest
  password: guest

# Java Mail Sender setting (Google Mail)
mail:
  host: smtp.gmail.com
  port: 587
  username: ENC(9zcJxzMJjdfsV9707BJQvAtUCYHyAV12FmzBkDqp
  password: ENC(bad9dzuo4Yv1McadA8E0Sn13JEE1/6UeyvsknQfC
  properties:
    mail:
      smtp:
        auth: true
        starttls:
          enable: true

flyway:
  enabled: true
  locations: classpath:db/migration
  baseline-on-migrate: true
  out-of-order: false

```



```

# jwt setting
jwt:
  accessKey: ENC(V6FVgNoVoFGu9ToZlWz18KKTUV11tnkhj0b7CsXow
  refreshKey: ENC(Ddc9rzvz2gZKRrXxLIROcj9UJEEmARcByz/2wru
  accessExpiration: PT420M # 60분 (PT420M)
  refreshExpiration: PT10080M # 10080분 (7일) (PT10080M)

# log 관리
logging:
  level:
    org.hibernate:
      type.descriptor.sql: trace
      org.hibernate.SQLQuery: debug

# OAUTH2.0 Setting
oauth:
  kakao:
    client-id: ENC(E3l0nslHCdphnWGsPFa1Hm5QFIrmkY1zPXzHgHU.
    client-secret: ENC(TYqE7s5TCJUb8/g+bxw/RWGWNJCGdmkVZbk
    redirect-uri: ${server.https-url}/member/loading/kakao
    scope:
      - profile_nickname
      - profile_image
      - account_email
#      - name

    naver:
      client-id: ENC(fgl1U6jmkfqSW5Lbet7BdnTta/ycQfTAwj3mVFH
      client-secret: ENC(MNZuMPNyaH+s/6Fx8g0KJ2bKC+7tFMMu)
      redirect_uri: ${server.https-url}/member/loading/naver
      scope:
        - nickname
        - name
        - email
        - profile_image

    google:
      client_id: ENC(QKPQqGiQ64z9Aqor6E4+iqkn9coyITJcZJpnGof

```

```

client_secret: ENC(9s47E9126PeDwGGEQ3CE0xgisvSvZEF8rG
redirect_uri: ${server.https-url}/member/loading/googl
scope:
  - profile
  - email

# firebase setting
app:
  firebase-configuration-file: classpath:serviceAccountKey
  firebase-bucket: ENC(SLVtdlieYMj0aGKJbbtcNZ7jA0k5GmHVdu8

# gpt-api setting
openai:
  key: ENC(QrUxpVFQUVlQx9zGMdFKUh4+HMsOPwGBmyzriWx3o/UydA9

```

2. React

- .env(로컬용)

```

# API URL settings for PJT
VITE_REACT_API_URL=http://localhost:8080/api/v1

# Firebase Setting

# 기본 URL 설정
BASE_URL=/

# Websocket URL
VITE_REACT_WS_URL=ws://localhost:8080/ws

# AWS POLLY 인증키(STT)
VITE_AWS_ACCESS_KEY_ID=AKIAYPGOVRI32DL4WUMZ
VITE_AWS_ACCESS_KEY_ID_APP_AWS_SECRET_ACCESS_KEY=muIN13CJS

```

- .env-dev(배포용)

```

# API URL settings for PJT
VITE_REACT_API_URL=https://j10c107.p.ssafy.io/api/v1

# Firebase Setting

# 기본 URL 설정
BASE_URL=/

# Websocket URL
VITE_REACT_WS_URL=wss://j10c107.p.ssafy.io/ws

# AWS POLLY 인증키 (STT)
VITE_AWS_ACCESS_KEY_ID=AKIAYPGOVRI32DL4WUMZ
VITE_AWS_ACCESS_KEY_ID_APP_AWS_SECRET_ACCESS_KEY=muIN13CJS

```

3. 빌드 및 배포

1) Docker + Docker compose 설치

```

#!/bin/bash

# 기존 도커 패키지 제거 (이전 버전이 설치된 경우)
sudo apt-get remove docker docker-engine docker.io containerd

# 필수 패키지 설치
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates c

# 도커 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sud

# 도커 저장소 추가
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-ar

# 도커 설치

```

```

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

# 도커 컴포즈 최신 버전 다운로드
sudo curl -L "https://github.com/docker/compose/releases/late

# 실행 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 도커 사용자 그룹에 현재 사용자 추가
sudo usermod -aG docker $USER
newgrp docker
sudo service docker restart

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

# 권한 설정
chmod +x installDocker.sh

# 실행
./installDocker.sh

```

2) Jenkins 컨테이너 실행

```

#!/bin/bash

# Jenkins 폴더 생성
JENKINS_DIR="./jenkins"
if [ ! -d "$JENKINS_DIR" ]; then
    mkdir "$JENKINS_DIR"
fi

# Docker Compose 실행
docker-compose up -d

# Jenkins 컨테이너가 완전히 실행될 때까지 대기

```

```

sudo sleep 60

# Jenkins 폴더로 이동
cd ./jenkins

# Jenkins 폴더가 완전히 생성될 때까지 대기
sudo sleep 60

# update center에 필요한 CA 파일 다운로드
UPDATE_CENTER_DIR="./update-center-rootCAs"
if [ ! -d "$UPDATE_CENTER_DIR" ]; then
    mkdir "$UPDATE_CENTER_DIR"
fi

sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-c

# Jenkins 설정 파일 수정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#

# Jenkins 재시작 (필수)
docker restart jenkins

# 현재 폴더 확인
pwd

# /home/ubuntu/develop/CICD 확인
chmod +x ./Install/installJenkins.sh

# 실행
./Install/installJenkins.sh

```

3) docker compose를 통한 실행

- Jenkins를 통해 docker-compose.yml파일의 구성을 실행시킨다.

docker-compose.yml

```

version: "0.0.0"

services:
  talkydoki_frontend:
    container_name: talkydoki_frontend
    build:
      context: ./FrontEnd
      dockerfile: Dockerfile
    image: talkydoki_frontend_img
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /etc/letsencrypt:/etc/nginx/ssl
    networks:
      - talkydoki_net

  talkydoki_backend_springboot:
    container_name: talkydoki_backend_springboot
    build:
      context: ./BackEnd/SpringBootTest
      dockerfile: Dockerfile
    image: talkydoki_backend_springboot_img
    restart: always
    ports:
      - "8080:8080"
    environment:
      - jasypt.encryptor.key=ssafy
    networks:
      - talkydoki_net

networks:
  talkydoki_net:

```

docker-compose-fastapi.yml

```

version: "0.0.0"

services:
  talkydoki_backend_fastapi:
    container_name: talkydoki_backend_fastapi
    build:
      context: ./BackEnd/FastApiServer
      dockerfile: Dockerfile
    image: talkydoki_backend_fastapi_img
    restart: always
    ports:
      - "8000:8000"
    networks:
      - talkydoki_net

  talkydoki_backend_dataprocessing:
    container_name: talkydoki_backend_dataprocessing
    build:
      context: ./BackEnd/DataProcessing
      dockerfile: Dockerfile
    image: talkydoki_backend_dataprocessing_img
    restart: always
    volumes:
      - /home/ubuntu/newsData:/usr/src/app/Hadoop/data
    networks:
      - talkydoki_net

networks:
  talkydoki_net:

```

BE Jenkinsfile

```

pipeline {
  agent any

  stages {
    stage('Deploy with Docker Compose') {
      steps {

```

```

        script {
            // 이전 실행에서 사용된 컨테이너 및 네트워크 정리
            sh "docker-compose down --volumes"

            // 새로운 푸시에 대한 스크립트 실행
            sh "docker-compose up --build -d"
        }
    }
}
}
}

```

FastAPI Jenkinsfile

```

pipeline {
    agent any

    stages {
        stage('Deploy with Docker Compose') {
            steps {
                script {
                    // 이전 실행에서 사용된 컨테이너 및 네트워크 정리
                    sh "docker-compose down --volumes"

                    // 새로운 푸시에 대한 스크립트 실행
                    sh "docker-compose -f docker-compose-fastapi up --build -d"

                    // FastAPI 재실행
                    sh "docker restart talkydoki_backend_fastapi"
                }
            }
        }
    }
}

```

BE Dockerfile


```

# OpenJDK 17 이미지를 베이스로 사용
FROM openjdk:17-jdk-slim

# 애플리케이션을 빌드할 소스 코드 및 리소스 복사
COPY . /app

# 작업 디렉토리 설정
WORKDIR /app

# Gradle Wrapper에 실행 권한 부여
RUN chmod +x ./gradlew

# Spring Boot 애플리케이션 빌드
RUN ./gradlew clean bootJar

# JAR 파일을 /app 디렉토리로 복사
RUN cp build/libs/*.jar /app/app.jar

RUN mkdir /app/videos

# Spring Boot 애플리케이션 실행을 위한 명령 설정
ENTRYPOINT ["java", "-Dspring.profiles.active=dev", "-jar", ".

```

FastAPI Dockerfile

```

# 공식 Python 런타임 이미지를 사용합니다
FROM python:3.10-slim

# 컨테이너 내에서 작업 디렉토리를 설정합니다
WORKDIR /app

# 현재 디렉토리의 내용을 컨테이너 내의 /app 디렉토리로 복사합니다
COPY . /app

# requirements.txt에 명시된 필요한 패키지를 설치합니다
RUN pip install --no-cache-dir -r requirements.txt

# 앱이 실행되는 포트를 노출합니다

```

```
EXPOSE 8000
```

```
# 환경 변수를 정의합니다
```

```
ENV FASTAPI_ENV production
```

```
# 애플리케이션을 실행하는 명령어를 정의합니다
```

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

DataProcessing Dockerfile

```
# Python 이미지를 베이스로 사용
```

```
FROM python:3
```

```
# 작업 디렉토리 생성 및 설정
```

```
WORKDIR /usr/src/app
```

```
# 필요한 경우 requirements.txt 파일을 사용하여 패키지를 설치합니다.
```

```
COPY requirements.txt .
```

```
RUN pip3 install --no-cache-dir -r requirements.txt
```

```
RUN pip3 install selenium
```

```
# MeCab 및 관련 패키지 설치
```

```
RUN apt-get update && apt-get -q -y install file mecab libmecab
```

```
RUN pip3 install mecab-python3
```

```
RUN pip3 install unidic-lite
```

```
RUN pip3 install paramiko
```

```
RUN pip3 install pytz
```

```
# Chrome 및 ChromeDriver 설치
```

```
RUN apt-get update && apt-get install -y wget
```

```
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub
```

```
RUN sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main"
```

```
RUN apt-get update && apt-get install -y google-chrome-stable
```

```
# ChromeDriver 다운로드 및 설치
```

```
RUN python -c "from webdriver_manager.chrome import ChromeDriverManager; ChromeDriverManager().install()"
```

```
# 크론 서비스 설치 및 크론 작업 추가
```

```

RUN apt-get -y install cron
COPY crontab /etc/cron.d/cronjob
RUN chmod 0644 /etc/cron.d/cronjob
RUN crontab /etc/cron.d/cronjob

# 모든 파일을 컨테이너 내부로 복사
COPY . .

# 스크립트 실행 권한 부여
RUN chmod +x /usr/src/app/DataProcessing.sh

# 크론 작업 실행 명령어를 추가합니다.
CMD ["cron", "-f"]

```

FE Nginx 멀티 스테이지 빌드(Multi-Stage Build) Dockerfile

```

# 기본 이미지로 Node.js 버전 20.11.1 사용
FROM node:20.11.1 as build

# 작업 디렉토리 설정
WORKDIR /usr/src/app

# package.json 및 package-lock.json을 복사하여 종속성 설치
COPY package*.json ./

# 종속성 설치
RUN npm install

# 나머지 애플리케이션 코드 복사
COPY . .

# .env 파일 변경
COPY .env-dev .env

# 프론트엔드 코드 빌드
RUN npm run build

```

```

# NGINX 이미지 생성
FROM nginx:latest

# NGINX에서 작업 디렉토리 설정
WORKDIR /usr/share/nginx/html

# 기본 NGINX 정적 콘텐츠 제거
RUN rm -rf ./*

# Node.js 빌드 단계에서 빌드된 프론트엔드 코드 복사
COPY --from=build /usr/src/app/dist/ .

# 추가 NGINX 구성 파일 복사
COPY nginx.conf /etc/nginx/conf.d/default.conf

# NGINX를 시작
CMD ["nginx", "-g", "daemon off;"]

```

5) NGINX 설정

nginx.conf

```

# HTTP 리다이렉션을 HTTPS로
server {
    listen 80;
    server_name j10c107.p.ssafy.io;

    return 301 https://$host$request_uri;
}

# HTTPS 서버 설정
server {
    listen 443 ssl;
    server_name j10c107.p.ssafy.io;

    ssl_certificate /etc/nginx/ssl/live/j10c107.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/live/j10c107.p.ssafy.io/private.pem;
    ssl_trusted_certificate /etc/nginx/ssl/live/j10c107.p.ssafy.io/intermediate.pem;
}

```

```

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

# 백엔드 프록시
location /api {
    proxy_pass http://j10c107.p.ssafy.io:8080; # 백엔드 서
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forward
    proxy_set_header X-Forwarded-Proto $scheme;
}

# 웹 소켓 프록시
location /ws {
    proxy_pass http://j10c107.p.ssafy.io:8080;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
}

```

4. 외부 서비스 및 정보

- 카카오 소셜 로그인 정보



카카오 로그인 ON

[동의 화면 미리보기](#)

동의항목

카카오 로그인으로 서비스를 시작할 때 동의받는 항목을 설정합니다. 미리보기를 통해 사용자에게 보여질 화면을 확인할 수 있습니다.
권한이 필요한 동의항목은 심사 신청 후 사용할 수 있습니다.

개인정보 동의항목 심사 신청

개인정보

항목 이름	ID	상태	
닉네임	profile_nickname	● 필수 동의	설정
프로필 사진	profile_image	● 필수 동의	설정
카카오계정(이메일)	account_email	● 필수 동의 [수집]	설정

- 네이버 소셜 로그인 정보

애플리케이션 이름 ↗

Talkydoki

- 네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요.
- 40자 이내의 영문, 한글, 숫자, 공백문자, 쉼표(,), "/" , "-" , "_" , 만 입력 가능합니다.

카테고리

기타 ▼

사용 API ↗

선택하세요. ▼

네이버 로그인

제공 정보 선택(이용자 식별자는 기본 정보로 제공) ?

필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보만을 선택해야 합니다.

권한	필수	추가
회원이름	<input checked="" type="checkbox"/>	<input type="checkbox"/>
연락처 이메일 주소	<input checked="" type="checkbox"/>	<input type="checkbox"/>
별명	<input checked="" type="checkbox"/>	<input type="checkbox"/>
프로필 사진	<input checked="" type="checkbox"/>	<input type="checkbox"/>
성별	<input type="checkbox"/>	<input type="checkbox"/>
생일	<input type="checkbox"/>	<input type="checkbox"/>
연령대	<input type="checkbox"/>	<input type="checkbox"/>
출생연도	<input type="checkbox"/>	<input type="checkbox"/>
휴대전화번호	<input type="checkbox"/>	<input type="checkbox"/>

- 구글 소셜 로그인 정보

사용자 인증 정보

+ 사용자 인증 정보 만들기

삭제

삭제된 사용자 인증 정보 복원

사용 설정한 API에 액세스하려면 사용자 인증 정보를 만드세요. 자세히 알아보기

API 키

<input type="checkbox"/> 이름	생성일 ↓	제한사항	작업
<input type="checkbox"/> Browser key (auto created by Firebase)	2024. 3. 11.	없음	키 표시

OAuth 2.0 클라이언트 ID

<input type="checkbox"/> 이름	생성일 ↓	유형	클라이언트 ID	작업
<input type="checkbox"/> Talkydoki Social Login	2024. 3. 11.	웹 애플리케이션	998604407617-m2gg...	<div> <div></div> <div></div> <div></div> </div>

서비스 계정

<input type="checkbox"/> 이메일	이름 ↑	작업
<input type="checkbox"/> talkydoki@appspot.gserviceaccount.com	App Engine default service account	<div> <div></div> <div></div> </div>
<input type="checkbox"/> firebase-adminsdk-9uq4q@talkydoki.iam.gserviceaccount.com	firebase-adminsdk	<div> <div></div> <div></div> </div>

이름 *

Talkydoki Social Login

OAuth 2.0 클라이언트의 이름입니다. 이 이름은 콘솔에서 클라이언트를 식별하는 용도로만 사용되며 최종 사용자에게 표시되지 않습니다.

5. DB dump 설명

- 개발 초기 ~ 중기 단계에서는 JPA를 이용하여 테이블들을 자동으로 DB와 매핑하여 Spring boot 서버 내에서 자동으로 생성 되게끔 설정하였습니다.
 - 사용된 DB dump 파일 (BackEnd/SpringBootTest/src/main/resources 폴더내에 저장)
 - data_member.sql (회원 데이터)
 - data_vocabulary.sql (단어장 데이터)
 - data_news.sql (뉴스 데이터)
 - data_news_keyword_mapping.sql (뉴스 키워드 매핑 데이터)
 - data_keyword (뉴스안 키워드 데이터)
- 개발 후기 단계 및 프로덕션 상태에서는 JPA를 이용한 DDL 자동 생성 및 삭제를 사용하지 않고 flyway를 사용한 DB 마이그레이션 작업을 실시하였습니다.
 - 사용된 DB dump 파일
(BackEnd/SpringBootTest/src/main/resources/db/migration 폴더내에 저장)
 - V1__Initial_schema.sql (배포용 초기 DB 테이블 및 연관관계 세팅)
 - V2__Add_hotel_check_in_to_ai_chat_room.sql (이후 추가된 컬럼에서 ENUM 값 세팅)
 - 이후에 기능 추가 및 테이블 및 연관 관계 추가 시 버전 업그레이드 하면서 관리할 예정