포팅 매뉴얼

I. 기술 스택 & 개발 환경

[사용 도구]

- 이슈 관리 : JIRA
- 형상 관리 : GitLab
- 커뮤니케이션 : Mattermost, Notion, Discord
- 디자인 : Figma
- UCC : Movavi
- CI/CD : EC2, Docker, Jenkins

[개발 환경]

- Front-end
  - 주요 프레임워크 및 빌드 도구
    - Node.js : 22.13.1
    - React : 18.3.1
    - Vite : 6.0.5
    - Recoil : 0.7.7 (전역 상태 관리)
  - 주요 라이브러리
    - React Router : 7.1.5 (라우팅)
    - Axios : 1.7.9 (HTTP 요청)
    - React Icons : 5.4.0 (아이콘)
    - SweetAlert2 : 11.15.0 (알림 모달)
    - Framer Motion : 12.4.2 (애니메이션)
    - Chart.js : 4.4.7 (차트)
    - PrimeReact : 10.9.2 (UI 컴포넌트)
    - Tailwind CSS : 3.4.17 (스타일링)
    - Date-fns : 4.1.0 (날짜 처리)
  - WebRTC & 실시간 기능
    - OpenVidu Browser : 2.31.0 (WebRTC 화상통화)
    - Simple-Peer : 9.11.1 (P2P WebRTC)
    - Socket.IO : 4.8.1 (실시간 소켓 통신)
  - 머신러닝 & AI 모델
    - Mediapipe Hands : 0.4.1675469240 (손 동작 인식)
    - TensorFlow.js : 4.22.0 (딥러닝 라이브러리)
    - Pose Detection : 2.1.3 (자세 추적)
  - 에디터 & 텍스트 관련
    - React Quill : 2.0.0 (리치 텍스트 에디터)
    - Dompurify : 3.2.4 (XSS 방지)
    - Xeger Quill Image Actions : 0.7.2 (이미지 확장 기능)
    - Xeger Quill Image Formats : 0.7.2 (이미지 포맷 확장)
  - 캘린더 & 날짜 관련
    - React Calendar : 5.1.0
  - 개발 환경 및 도구
    - ESLint : 9.17.0
    - Prettier : 3.5.1
    - PostCSS : 8.5.1
    - Autoprefixer : 10.4.20

        - @vitejs/plugin-react : 4.3.4

- Back-end
    - 주요 개발 환경
        - Java : OpenJDK 17
        - Spring Boot : 3.4.1
        - Spring Dependency Management : 1.1.7
        - Build Tool : Gradle
        - IDE : IntelliJ IDEA
    - 주요 라이브러리
        - Spring Boot Core
            - Spring Boot Starter Web, Security, WebSocket
            - Spring Boot Starter Data JPA, Redis
            - Spring Boot Starter Mail, Validation, WebFlux
        - Spring Cloud
            - Spring Cloud AWS: 2.2.6.RELEASE
        - OAuth 및 인증 관련
            - Spring Boot Starter OAuth2 Client
            - JJWT (JSON Web Token) : 0.11.5
        - Swagger (API 문서)
            - Swagger Annotations: 2.2.25
        - WebRTC 및 실시간 기능
            - OpenVidu Java Client: 2.25.0
        - Elasticsearch
            - Spring Boot Starter Data Elasticsearch
            - Elasticsearch Java 클라이언트: 8.17.0
        - 데이터베이스 및 Jackson
            - MySQL Connector
            - Jackson Datatype JSR310: 2.18.2
        - HTTP 클라이언트
            - Apache HttpClient: 4.5.13
        - Google Cloud API
            - Google Cloud Speech
            - Protobuf Java: 3.23.4
        - HTML 파싱
            - Jsoup: 1.15.4

- DB
    - Database : MySQL 8.0.40
    - Driver : MySQL Connector
    - Port : 3306Database Name : dream_moa

- Infra (버전 정보 추가 부탁)
    - 서버
        - AWS EC2 : Ubuntu 22.04.4 LTS
    - 컨테이너
        - Docker : 27.5.1
    - CI/CD
        - Jenkins : 2.497
    - 웹 서버
        - Nginx : 1.27.4

[외부 서비스]

- Google STT API

[gitignore]

```
[Back]

# Gradle 관련
.gradle/
build/

# IDE 설정 파일
.idea/
.vscode/

# OS별 파일
.DS_Store
Thumbs.db

# 로그 및 환경 파일
*.log
.env
application.properties

C:/SSAFY/uploads

# 컴파일 산출물
bin/
out/
*.class
```

```
[Front]

# Logs
logs
*.log
npm-debug.log*

# Node.js 관련
node_modules
dist

# IDE 관련
.vscode/
.idea/
```

[환경변수(Back-end)]

```
# 서버 설정
SERVER_PORT={백엔드 서버 포트}

# DB 서버 연결 설정
DB_URL=jdbc:mysql://{도메인}:{DB 포트}/{DB 이름}
DB_USERNAME={DB 아이디}
DB_PASSWORD={DB 패스워드}

# OAuth (Google, Naver, Kakao)
GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GOOGLE_REDIRECT_URI
NAVER_CLIENT_ID, NAVER_CLIENT_SECRET, NAVER_REDIRECT_URI
KAKAO_CLIENT_ID, KAKAO_REDIRECT_URI

# Redis 설정
REDIS_HOST=localhost
REDIS_PORT=6380
REDIS_TIMEOUT=2000ms

# WebSocket 설정
```

```
WEBSOCKET_ENABLED=true
WEBSOCKET_ALLOW_ORIGIN=*

# OpenVidu (WebRTC)
OPENVIDU_URL=https://dreammoa.duckdns.org:8443/
OPENVIDU_SECRET=DREAMMOA

# OpenAI (GPT API)
OPENAI_API_KEY={OpenAI API Key}
OPENAI_API_URL=https://api.openai.com/v1/chat/completions
OPENAI_MODEL=gpt-3.5-turbo-1106

# Elasticsearch 설정
ELASTICSEARCH_HOST=localhost
ELASTICSEARCH_USERNAME={USER ID}
ELASTICSEARCH_PASSWORD={USER PASSWORD}
ELASTICSEARCH_URI=http://localhost:9200

# 파일 업로드 설정
FILE_UPLOAD_DIR=C:/SSAFY/uploads/
MAX_FILE_SIZE=500MB

# AWS S3 설정
AWS_S3_BUCKET=dream-moa
AWS_REGION=ap-northeast-2
AWS_ACCESS_KEY={AWS 액세스 키}
AWS_SECRET_KEY={AWS 시크릿 키}

# SMTP
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USERNAME=Gmail 계정
SMTP_PASSWORD=Gmail 앱 비밀번호
SMTP_AUTH=true
SMTP_STARTTLS_ENABLE=true
SMTP_CONNECTION_TIMEOUT=5000ms
MAIL_AUTH_CODE_EXPIRATION=1800000ms (30분)
```

II. 빌드 및 배포

[개발 환경에서 직접 빌드 (로컬 빌드)]

- Front-end
  - 의존성 설치 npm install
  - 개발 서버 실행 npm run dev
- Back-end
  - 프로젝트 빌드 `./gradlew build`
  - 빌드된 JAR 파일 실행 `java -jar build/libs/{프로젝트명}.jar`

[배포 시 빌드 (Jenkins 파이프라인)]

- jenkins 파이프라인

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE_BE = 'backend'
        DOCKER_IMAGE_FE = 'frontend'
        DOCKER_IMAGE_FASTAPI = 'fastapi'
        DOCKER_REGISTRY = 'hanju2725/dreammoa'
        GIT_REPO = 'https://lab.ssafy.com/s12-webmobile1-sub1/S12P11C106.git'
    }

    tools {
        gradle 'gradle'
```

```
    nodejs 'nodeJS'
}

stages {
//     stage('Clone Repository') {
//         steps {
//             script {
//                 sh 'rm -rf S12P11C106 || true'

//                 withCredentials([string(credentialsId: 'gitlabtoken', variable: 'GITLAB_TOKEN')]) {
//                     sh '''
//                     git clone -b deploy https://oauth2:${GITLAB_TOKEN}@lab.ssafy.com/s12-webmobile1-sub1/S12P11C106.git
//                     cd S12P11C106
//                     '''
//                 }
//             }
//         }
//     }

    // stage('Add Env') {
    //     steps {
    //         dir('S12P11C106/BE') {
    //             withCredentials([file(credentialsId: 'application', variable: 'application')]) {
    //                 sh 'cp ${application} src/main/resources/application.properties'
    //             }
    //         }
    //     }
    // }

//     stage('Build Backend') {
//         steps {
//             dir('S12P11C106/BE') {
//                 sh '''
//                 chmod +x gradlew
//                 ./gradlew clean build -x test
//                 '''
//             }
//         }
//     }

//     stage('Build Frontend') {
//         steps {
//             dir('S12P11C106/FE') {
//                 sh 'npm install'
//                 sh 'npm run build'
//             }
//         }
//     }

    stage('Docker Login') {
        steps {
            script {
                withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PA
                    sh "echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin"
                }
            }
        }
    }

    // stage('Docker Build & Push') {
    //     steps {
    //         script {
    //             sh "docker build --no-cache -t $DOCKER_REGISTRY:$DOCKER_IMAGE_BE-latest S12P11C106/BE"
    //             sh "docker push $DOCKER_REGISTRY:$DOCKER_IMAGE_BE-latest"
    //             sh "docker build --no-cache -t $DOCKER_REGISTRY:$DOCKER_IMAGE_FE-latest S12P11C106/FE"
    //             sh "docker push $DOCKER_REGISTRY:$DOCKER_IMAGE_FE-latest"
    //             sh "docker build --no-cache -t $DOCKER_REGISTRY:$DOCKER_IMAGE_FASTAPI-latest S12P11C106/PYTHON"
    //             sh "docker push $DOCKER_REGISTRY:$DOCKER_IMAGE_FASTAPI-latest"
    //         }
    //     }
    // }
```

```
        stage('Deploy') {
            steps {
                script {
                    // sh 'docker-compose -f /var/jenkins_home/docker-compose/docker-compose.yml down'
                    // sh 'docker rmi hanju2725/dreammoa:backend-latest'
                    // sh 'docker rmi hanju2725/dreammoa:frontend-latest'
                    // sh 'docker rmi hanju2725/dreammoa:fastapi-latest'
                    // sh 'docker image prune -f'
                    // sh 'docker-compose -f /var/jenkins_home/docker-compose/docker-compose.yml up --build -d'

                    // 네트워크를 삭제하지 않고 특정 컨테이너만 종료
                    sh 'docker-compose -f /var/jenkins_home/docker-compose/docker-compose.yml stop backend frontend fastapi'
                    sh 'docker-compose -f /var/jenkins_home/docker-compose/docker-compose.yml rm -f backend frontend fastapi'

                    // // 기존 이미지를 삭제
                    sh 'docker rmi hanju2725/dreammoa:backend-latest || true'
                    sh 'docker rmi hanju2725/dreammoa:frontend-latest || true'
                    sh 'docker rmi hanju2725/dreammoa:fastapi-latest || true'

                    // 필요 없는 이미지 정리
                    sh 'docker image prune -f'

                    // // 백엔드, 프론트엔드, FastAPI 컨테이너만 다시 실행
                    sh 'docker-compose -f /var/jenkins_home/docker-compose/docker-compose.yml up --build -d backend frontend fastapi'
                }
            }
        }

        stage('Notification') {
            steps {
                echo 'jenkins notification!'
            }
            post {
                always {
                    script {
                        def isGitRepo = sh(script: "[ -d S12P11C106/.git ] && echo true || echo false", returnStdout: true).trim()
                        if (isGitRepo == "true") {
                            dir('S12P11C106') {
                                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()

                                def status = currentBuild.result ?: 'SUCCESS'
                                def color = (status == 'SUCCESS') ? 'good' : 'danger'
                                def message = "빌드 ${status}: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.BUILD_URL}|Details>)"

                                mattermostSend(color: color,
                                    message: message,
                                    endpoint: 'https://meeting.ssafy.com/hooks/6s9owcjm478bjjzqhy5mbsxoyo',
                                    channel: 'C106_jenkins'
                                )
                            }
                        } else {
                            echo "Git repository not found, skipping notification."
                        }
                    }
                }
            }
        }
    }
}
```

- Frontend Dockerfile

```
# 1단계: Node.js 기반 이미지 사용
FROM node:22.13.0-alpine AS build

# 리액트 애플리케이션 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json 파일 복사
```

```
COPY package*.json ./

# 의존성 설치
RUN npm install

# 현재 디렉토리의 모든 파일을 도커 컨테이너의 작업 디렉토리에 복사
COPY . .

# 리액트 애플리케이션 빌드
RUN npm run build

# 2단계: Nginx를 사용해 빌드된 파일을 서빙
FROM nginx:alpine

# 이전 빌드 단계에서 빌드한 결과물을 /usr/share/nginx/html 으로 복사한다.
COPY --from=build /app/dist /usr/share/nginx/html

# 기본 nginx 설정 파일을 삭제한다. (custom 설정과 충돌 방지)
RUN rm /etc/nginx/nginx.conf

# 포트 80을 외부로 노출
EXPOSE 80 433

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

- Backend Dockerfile

```
# Use a base image with JDK 17
FROM openjdk:17-jdk-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the built JAR file into the container (assumes the JAR file is built in the `build/libs` folder)
COPY build/libs/*.jar app.jar

# Expose the port the application will run on
EXPOSE 8080

# Run the Spring Boot application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- Nginx 설정 파일

```
upstream.conf

# upstream은 nginx가 받은 요청을 넘겨 줄 서버 지시자
# keepalive는 upstream 접속에 사용될 connection 수

upstream backend {
    server backend:8080;
#    keepalive 10;
}

upstream fastapi {
    server fastapi:8000;
#    keepalive 10;
}
```

```
nginx.conf

worker_processes  auto;

## Worker Process가 수용할 수 있는 Connection 개수
events {
    worker_connections  1024;
```

```
}

http {
    include       conf.d/mime.types;              # 해당 http 블럭에서 처리할 mime type 모음, 기본으로 octet_stream을 사용함
    default_type  application/octet-stream; # octet_stream은 file이나 data의 content-type을 식별하기 어려운 경우에 사용할 수 있는 기>본 타입임

    sendfile       on;                    # 디스크에서 네트워크로 파일을 전송할 때 sendfile()이라는 system call을 사용할지 여부를 >결정함
    tcp_nopush     on;                    # TCP 패킷을 가능한 빨리 전송할 지(on) 또는 조금 기다렸다가 모아서 전송할 지(off)를 결정>함

    keepalive_timeout  65;                # keep-alive connection는 유휴 상태의 keep-alive 커넥션이 얼마나 오래 유지될지를 결정함(default 75)
    keepalive_requests 100;               # keep-alive requests는 단일 keep-alive 커넥션이 최대 몇 개의 요청을 전송할 수 있는지를 >결정함(default 100)
    gzip           on;                    # gzip 압축을 활성화함

    send_timeout       15s;               # send_timeout는 nginx가 WAS로 요청을 보내고, 응답을 기다리는 시간임(default 60s)
    resolver_timeout   5s;                # resolver_timeout는 DNS를 찾는 것을 완료하기까지 기다리는 시간임(default 30s)


    large_client_header_buffers 20 32k;    # 요청 헤더를 읽을 때 사용되는 버퍼의 최대 개수와 크기(default 4, 8k), 초과 시 414 에러 >반환
    client_header_buffer_size   8k;        # 요청 헤더를 저장하기 위한 버퍼 크기(default 1k), 초과 시 414 에러 반환 ex)쿠키 크기가 >커질 경우
    client_max_body_size        100M;      # 요청 바디의 최대 크기(default 1M), 초과 시 413 에러 반환
    client_body_buffer_size     1M;        # 요청 바디를 저장하기 위한 버퍼 크기(default 8k), 초과 시 버퍼 데이터를 메모리에서 디스>크로 저장함
    output_buffers 20 32k;                # 응답 바디를 저장하기 위한 버퍼 크기(default 1, 32k), 32K짜리 버퍼 1개를 응답에 사용하겠다는 의미임

    ignore_invalid_headers off;           # 요청 헤더에 유효하지 않은 헤더가 있을 경우 무시할지 결정함
    server_tokens off;                    # 응답 헤더에 nginx 버전 정보를 넣을지 결정함
    autoindex off;                        # 요청 경로 대상이 디렉토리인 경우, 파일 리스팅을 할지 여부(off 시 403 에러 반환)

    # 요청을 다른 서버로 프록시할 때, 헤더를 세팅해줌
    include conf.d/header.conf;

    # nginx 로그 포맷
    log_format  main  '$remote_addr $remote_user "$request" '
                      '$status $body_bytes_sent "$http_referer" "$request_time" '
                      '"$http_user_agent" ';

    include conf.d/upstream.conf;      # upstream(서버 그룹)을 모아둔 설정 파일
    include conf.d/http-server.conf;   # Http 요청(80번 포트)로 들어오는 경우 처리하는 서버 블럭
    include conf.d/ssl-server.conf;    # SSL 요청(443 포트)로 들어오는 경우 처리하는 서버 블럭

}
```

http-server.conf

```
# http 80으로 들어온 요청에 대한 proxy pass 연결 부분
server {

        listen 80;                      # 80번 요청에 대한 listen
        server_name dreammoa.duckdns.org; # 이 server 블럭에서 처리할 도메인명들

    # ~*은 정규 표현식을 사용하여 요청된 URI와 지정된 패턴을 비교하는 정규식 매칭 연산자
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}
```

ssl-server.conf

```
# https 443으로 들어온 요청에 대한 proxy pass 연결 부분
server {
    listen 443 ssl;
    server_name dreammoa.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/dreammoa.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dreammoa.duckdns.org/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
```

```
# 로그 파일 경로 설정
access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;

# 프론트엔드 React 앱 서빙
location / {
    root /usr/share/nginx/html;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
    }
    location ~ /\.env {
        deny all;
    }
    location ~* ^/(cgi-bin|manager|boaform) {
        return 403;
    }
# 정적 파일 MIME 타입 설정 및 캐싱 최적화
location /assets/ {
    root /usr/share/nginx/html;
    expires 1y;
    add_header Cache-Control "public, max-age=31536000, immutable";
    access_log off;
}

# /actuator 접근 제한 (SpringBoot Actuator 보안)
location ^~ /actuator {
    return 404;
}
# 📌 OpenVidu 프록시 설정 (8443 포트 사용)
location /openvidu/ {
     proxy_pass https://dreammoa.duckdns.org:8443/;
     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_ssl_verify off;  # OpenVidu가 자체 SSL을 사용하므로 검증 비활성화
}
# Spring Boot API 요청을 백엔드로 프록시
location /api/ {
    proxy_pass http://backend/;  # backend 컨테이너로 요청 전달
    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_buffering off;
        proxy_cache off;
        proxy_read_timeout 3600s;
}
# 서버 상태 확인을 위한 엔드포인트 예시
location /health {
    proxy_pass http://backend/health;  # 백엔드의 헬스 체크 URL
}
    # 📌 FastAPI 요청을 프록시 (예: `/fastapi/` 경로 사용)
location /fastapi/ {
    proxy_pass http://fastapi/;  # fastapi 컨테이너로 요청 전달
    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;

    # WebSocket 업그레이드 처리 추가
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_read_timeout 86400;
        proxy_send_timeout 86400;
        send_timeout 86400;
    # 연결 끊김 방지
        proxy_connect_timeout 600s;
        keepalive_timeout 600s;
```

```
        # WebSocket 관련 버퍼 설정
            proxy_buffering off;
            proxy_cache off;
    }
}
```