

포팅 매뉴얼

👤 작성자	👤 재환 🍷 정찬호
🕒 최종 수정일	@2025년 2월 20일 오후 11:49

1. Directory Structure

2. Tech Stack Version

01. Web

001. System Configuration

002. Primary Packages

02. Jetson

001. System Configuration

002. Primary Packages

003. Models

3. Dockerfile & docker-compose.yml

00. Web docker-compose.yml

01. Django

02. React

03. FastAPI

04. chromadb

4. How to Run

01. Web

02. Jetson

001. 환경변수 설명 (Jetson/fastapi/.env)

5. 시연 시나리오

1. Directory Structure

- Jetson Orin Nano를 사용하는 프로젝트 구조 상 프로젝트 루트 디렉토리에 **Web/** 과 **Jetson/** 디렉토리를 구분하여 사용
- **Web/** 은 다시 **frontend/** 와 **backend/** 로 구분
- **Jetson/** 은 **chromaDB/** 와 **fastapi/** 로 구분

```
📁 project-root
├── 📁 Web
│   ├── 📁 backend
│   │   ├── 📁 MariaDB
│   │   │   └── 📄 .sql
│   │   └── 📁 django
│   │       └── 📁 app
│   │           ├── 📄 Dockerfile
│   │           ├── 📄 manage.py
│   │           ├── 📄 requirements.txt
│   │           ├── 📁 accounts
│   │           ├── 📁 ai203
│   │           ├── 📁 meetingroom
│   │           ├── 📁 meetings
│   │           ├── 📁 projects
│   │           └── 📄 wait-for-it.sh
│   └── 📄 docker-compose.yml
├── 📁 frontend
│   └── 📁 react
│       └── 📁 app
│           ├── 📄 Dockerfile
│           └── 📄 README.md
```

```
|   ├── package.json
|   ├── vite.config.js
|   ├── public
|   └── src
├── Jetson
|   ├── chromaDB
|   |   ├── data
|   |   └── Dockerfile
|   ├── fastapi
|   |   ├── .llm-model-caches
|   |   ├── app
|   |   |   ├── api
|   |   |   ├── core
|   |   |   ├── schemes
|   |   |   ├── services
|   |   |   ├── utils
|   |   |   ├── dependencies.py
|   |   |   └── main.py
|   |   ├── Dockerfile
|   |   ├── README.md
|   |   ├── requirements-jetson.txt
|   |   ├── requirements-macos.txt
|   |   └── requirements-windows.txt
|   └── docker-compose.yml
```

2. Tech Stack Version

01. Web

! EC2 미사용

→ 네트워크 구조 상 외부에서 Jetson Orin Nano로의 데이터 전송이 불가하여 SSAFY 내 사설망에서 진행

001. System Configuration

- **OS:** Windows 11
- **Docker Desktop:** 4.37.1
- **Docker Engine:** 27.4.0
- **Docker Client:** 27.4.0

002. Primary Packages

- **celery:** 5.4.0 → 분산 작업 큐
 - **Django:** 5.1.5 → 웹 프레임워크
 - **django-rest-framework:** 3.15.2 → Django용 REST API 라이브러리
 - **mysqlclient:** 2.2.7 → MySQL 데이터베이스 드라이버
 - **python-docx:** 1.1.2 → Word 문서 처리 라이브러리
 - **redis:** 5.2.1 → Redis 클라이언트 라이브러리
 - **uvicorn:** 0.34.0 → ASGI 서버
-
- **reduxjs/toolkit:** 2.5.1 → Redux 상태 관리를 쉽게 해주는 공식 툴킷
 - **react-big-calendar:** 1.17.1 → 일정 및 이벤트 캘린더 UI 컴포넌트
 - **react-calendar:** 5.1.0 → 간단한 날짜 선택 및 달력 UI 제공
 - **react-day-picker:** 9.5.1 → 날짜 선택을 위한 커스텀 가능한 React 컴포넌트
 - **react-dom:** 18.3.1 → React 요소를 실제 DOM에 렌더링하는 핵심 라이브러리
 - **react-icons:** 5.4.0 → 다양한 아이콘을 쉽게 사용할 수 있는 라이브러리
 - **react-redux:** 9.2.0 → Redux를 React에서 쉽게 사용할 수 있도록 도와주는 라이브러리
 - **react-router-dom:** 7.1.4 → React 애플리케이션에서 클라이언트 사이드 라우팅 지원
 - **styled-components:** 6.1.14 → CSS-in-JS 방식으로 스타일을 관리
 - **@restart/hooks:** 0.4.16 → React의 훅을 보완하는 유틸리티 라이브러리
 - **react-overlays:** 5.2.1 → 모달, 툴팁 등의 UI 오버레이를 쉽게 구현할 수 있는 라이브러리
 - **uncontrollable:** 7.2.1 → React 컴포넌트의 `controlled` 및 `uncontrolled` 상태 관리
 - **use-sync-external-store:** 1.4.0 → 외부 스토어(예: Redux)의 상태를 동기적으로 구독하는 React 훅

02. Jetson

001. System Configuration

- **Device:** Jetson Orin Nano Developer Kit
- **Jetpack:** Jetpack 6.2 <https://developer.nvidia.com/embedded/jetpack>
- **OS:** Jetson Linux 36.4 (Ubuntu 22.04)
- **CUDA:** 12.6

002. Primary Packages

<https://pypi.jetson-ai-lab.dev/jp6/cu126>

- **PyTorch:** 2.5.0 → 딥러닝 프레임워크
- **transformers:** 4.48.2 → Pre-trained NLP 모델 라이브러리
- **sentence-transformers:** 3.4.1 → 문장 임베딩 생성
- **llama-cpp-python:** 0.3.7 → Llama 모델 실행 지원
- **bitsandbytes:** 0.45.0 → 저비트 연산 최적화
- **ctranslate2:** 4.5.0 → 경량 번역 및 추론 엔진
- **faster_whisper:** 1.1.1 → Whisper STT 모델 가속
- **SpeechRecognition:** 3.14.1 → 음성 인식 라이브러리
- **fastapi:** 0.115.8 → 고성능 웹 API 프레임워크

003. Models

- **Embedding :** [nlpai-lab/KoE5](#)
- **RAG:** [LGAI/EXAONE-3.5-2.4B-Instruct-Q5_K_M](#)
- **Summary:** [LGAI/EXAONE-3.5-2.4B-Instruct-Q6_K_L](#)

<https://huggingface.co/nlpai-lab/KoE5>

<https://huggingface.co/LGAI-EXAONE>

3. Dockerfile & docker-compose.yml

00. Web docker-compose.yml

```
services:
  backend:
    build:
      context: ./backend/django/app
      dockerfile: Dockerfile
    env_file:
      - ./backend/django/app/.env
    container_name: backend
    ports:
      - "8000:8000"
    depends_on:
      - db
      - redis

  celery:
    build:
      context: ./backend/django/app
      dockerfile: Dockerfile
    env_file:
      - ./backend/django/app/.env
    container_name: celery
```

```

command: celery -A ai203 worker --loglevel=info --pool=solo
depends_on:
  - db
  - redis

frontend:
  build:
    context: ./frontend/react/app
    dockerfile: Dockerfile
  env_file:
    - ./frontend/react/app/.env
  container_name: frontend
  ports:
    - "5173:5173"

db:
  image: mariadb:latest
  container_name: mariadb
  environment:
    MYSQL_ROOT_PASSWORD: root_pw
    MYSQL_DATABASE: test_maria
    MYSQL_USER: admin
    MYSQL_PASSWORD: admin
  ports:
    - "3306:3306"
  volumes:
    - ".sql 파일 경로::컨테이너 시작 시 실행될 initdb 경로"

redis:
  image: redis:latest
  container_name: redis
  ports:
    - "6379:6379"

```

01. Django

```

# django Dockerfile
# Base image
FROM python:3.12.8

# Set working directory
WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy project files
COPY . .

# wait-for-it.sh 다운로드 + 실행 권한 부여
# COPY wait-for-it.sh /app/wait-for-it.sh
RUN chmod +x ./wait-for-it.sh

```

```
# 환경 변수 설정
ENV CELERY_BROKER_URL=redis://redis:6379/0
ENV CELERY_BACKEND=redis://redis:6379/0

# Expose port
EXPOSE 8000

# Run the server
CMD ["sh", "-c", "./wait-for-it.sh db:3306 -t 30 && python manage.py makemigrations && python manage.py migrate"]
```

02. React

```
# react Dockerfile
ARG NODE_VERSION=22.13.0-alpine
FROM node:${NODE_VERSION}

# Set working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package.json package-lock.json ./
RUN npm install

# Copy the rest of the application
COPY . .

# Expose port
EXPOSE 5173

# Run React app
CMD ["npm", "run", "dev"]
```

03. FastAPI

- FROM nvcr.io/nvidia/l4t-jetpack:r36.4.0 : Nvidia에서 제공하는 Jetpack Image 사용
- WORKDIR /fastapi

```
# https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-jetpack
# nvidia jetpack image
FROM nvcr.io/nvidia/l4t-jetpack:r36.4.0

WORKDIR /fastapi

# 필수 패키지 설치
RUN apt-get update && apt-get install -y \
    zsh \
    curl \
    wget \
    git \
    python3-pip \
    sudo \
    ninja-build \
    libopenblas-dev \
```

```

language-pack-en \
libomp-dev \
libasound2-dev \
alsa-utils \
portaudio19-dev \
cmake \
build-essential \
zlib1g-dev \
&& apt-get clean

# zsh를 기본 셸로 설정
RUN chsh -s $(which zsh)

# zsh를 자동으로 실행하도록 bash를 수정
RUN echo "exec zsh" >> ~/.bashrc

# zsh 및 플러그인 설정
RUN sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)" || true &
git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins
git clone https://github.com/zsh-users/zsh-syntax-highlighting.git ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/p
echo "\nplugins=(git zsh-autosuggestions zsh-syntax-highlighting)\n" >> ~/.zshrc

# 코드 복사
COPY . /fastapi

# FastAPI 및 라이브러리 설치
RUN pip install --no-cache-dir --ignore-installed numpy
RUN pip install --no-cache-dir --use-feature=in-tree-build \
./llama_cpp_python-0.3.7-cp310-cp310-linux_aarch64.whl \
./torch-2.5.0-cp310-cp310-linux_aarch64.whl \
./bitsandbytes-0.45.0-cp310-cp310-linux_aarch64.whl \
./ctranslate2-4.5.0-cp310-cp310-linux_aarch64.whl \
-r /fastapi/requirements-jetson.txt

# 기존에 설치되어 있을 수 있는 ctranslate2 제거 (오류 방지를 위해)
RUN pip uninstall -y ctranslate2 || true

# CTranslate2 소스코드 클론 후 빌드 및 설치
# RUN git clone --recursive https://github.com/OpenNMT/CTranslate2.git /opt/CTranslate2 && \
# cd /opt/CTranslate2 && \
# mkdir build && cd build && \
# cmake -DWITH_MKL=OFF -DWITH_INTEL_OPENMP=OFF -DOPENMP_RUNTIME=NONE -DWITH_CUDA=ON && \
# make -j$(nproc) && \
# make install && \
# ldconfig && \
# cd ../python && \
# pip install -r install_requirements.txt && \
# python setup.py bdist_wheel && \
# pip install dist/*.whl

# Expose port for FastAPI
EXPOSE 8000

```

04. chromadb

```
# ChromaDB 공식 이미지 사용
FROM chromadb/chroma:latest
```

```
# 작업 디렉토리 설정
WORKDIR /chroma
```

```
# 포트 노출
EXPOSE 8001
```

4. How to Run

01. Web

1. Clone project
2. Web 디렉토리 이동 (cd Web)
3. DB 파일 MariaDB 경로에 위치하고 있는지 확인
4. .env 파일 수정
 - a. django .env :
 - i. REDIS_BASE_URL=local url:**6379**
 - ii. **FASTAPI_BASE_URL=Jetson** local url:8000
 - iii. **DJANGO_BASE_URL**=local url:8000
 - b. react .env :
 - i. **VITE_APP_BASEURL**=local url:8000
5. 컴포즈 빌드 및 실행 (docker compose up —build -d)

02. Jetson

1. Clone project
2. Jetson 디렉토리 이동 (cd Jetson)
3. 컴포즈 빌드 및 실행 (docker compose up —build -d)
4. fastapi 디렉토리 이동 (cd fastapi)
5. .env 파일 수정
 - a. SYSTEM_OS=linux
 - b. etc..
6. 모델 저장소 생성 (mkdir .llm-model-caches) 및 이동 (cd .llm-model-caches)
7. huggingface-caches 및 llamacpp-caches 생성
(mkdir huggingface-caches llamacpp-caches)
8. RAG 및 Summary 모델 다운로드
 - a. \$ cd llamacpp-caches
 - b. \$ wget https://huggingface.co/bartowski/EXAONE-3.5-2.4B-Instruct-GGUF/blob/main/EXAONE-3.5-2.4B-Instruct-Q5_K_M.gguf
 - c. \$ wget https://huggingface.co/bartowski/EXAONE-3.5-2.4B-Instruct-GGUF/blob/main/EXAONE-3.5-2.4B-Instruct-Q6_K_L.gguf

001. 환경변수 설명 (Jetson/fastapi/.env)

환경 변수	설정 값	설명
SYSTEM_OS	linux	운영 체제 설정 (linux, macos, windows 중 선택)
DJANGO_URL		Django STT 데이터를 전송할 URL
CHROMADB_PERSISTENT_CLIENT_PATH	../chromaDB/data	ChromaDB 영구 저장 경로
CHROMADB_HTTP_CLIENT_HOST	chromadb-server	ChromaDB HTTP 클라이언트 호스트 (docker-compose.yml에서 설정한 이름)
CHROMADB_HTTP_CLIENT_PORT	8001	ChromaDB HTTP 클라이언트 포트
CHROMADB_HTTP_CLIENT_SSL	False	ChromaDB HTTP 클라이언트 SSL 사용 여부
LLM_MODEL_CACHES_DIR	./llm-model-caches/	LLM 모델 캐시 디렉토리
HUGGINGFACE_CACHE_DIR	./llm-model-caches/huggingface-caches/	HuggingFace 캐시 디렉토리
LLAMACPP_CACHE_DIR	./llm-model-caches/llamacpp-cache/	LlamaCPP 캐시 디렉토리
EMBEDDING_MODEL_NAME	nlpai-lab/KoE5	임베딩 모델 이름
EMBEDDING_MODEL_LOW_CPU_USAGE	True	임베딩 모델 저사양 사용 여부
EMBEDDING_MODEL_LOAD_IN_4BIT	True	4-bit 양자화 사용 여부
EMBEDDING_MODEL_BBN_4BIT_COMPUTE_DTYPE	torch.float16	4-bit 양자화 사용 시 데이터 타입
RAG_MODEL_NAME	EXAONE-3.5-2.4B-Instruct-Q4_K_M.gguf	RAG 모델 이름
RAG_MODEL_N_CTX	2048	RAG 모델 입력 프롬프트 최대 토큰 수
RAG_MODEL_TEMPERATURE	0.0	RAG 모델 온도 (텍스트 생성 무작위성 조절)
RAG_MODEL_N_GPU_LAYERS	-1	RAG 모델 GPU 레이어 수 (-1: 모든 레이어 사용)
SUMMARY_MODEL_NAME	EXAONE-3.5-2.4B-Instruct-Q6_K_L.gguf	요약 모델 이름

환경 변수	설정 값	설명
SUMMARY_MODEL_N_CTX	4096	요약 모델 입력 프롬프트 최대 토큰 수
SUMMARY_MODEL_TEMPERATURE	0.0	요약 모델 온도 (텍스트 생성 무작위성 조절)
SUMMARY_MODEL_N_GPU_LAYERS	-1	요약 모델 GPU 레이어 수 (-1: 모든 레이어 사용)

5. 시연 시나리오

- django의 .env에 입력한 PC IP:5173 에 접속하면, '로그인 페이지' 위치
- '123000 + (1 ~ 9)'를 아이디로 입력하고, 'qwe123``' 를 입력하여
유저로 로그인 > '대시보드 페이지' 접속
- 대시보드 페이지에서 좌측 탭의 Project 버튼을 눌러 '프로젝트 생성 및 관리 페이지' 접속
- 프로젝트 생성 후 좌측 탭에서 Meetingroom 버튼 눌러 '회의 예약 페이지'로 이동.
- 회의실 1, 2중 하나 선택하여 원하는 날짜, 시간의 선택한 프로젝트 관련 회의 생성
- 회의 생성 이후, 표에 있는 해당 회의 버튼 눌러 회의 대기 버튼 클릭
- 회의 대기가 끝나면, 회의 시작 버튼이 나오고, 회의 시작 버튼을 누르면
생성한 회의의 첫번째 안건부터 실시간 회의 시작
- 실시간 회의가 시작되면, 마이크로 들어간 음성이 문장별로 화면에 출력되고,
우측에는 해당 안건과 유사도가 높은 문서들이 내림차순으로 정렬되어 출력됨.
- 실시간 회의 도중, '아리' 가 들어간 음성이 마이크로 들어가면,
해당 문장에 들어있는 유저의 질문에 대해 RAG를 통해 생성된 답변을 출력해주고,
가장 관련이 높은 문서를 우측에 팝업해줌.
- '다음 안건' 버튼을 누르면 다음 안건의 회의를 진행 가능하고,
해당 안건과 유사한 문서를 또다시 우측에 정렬해 출력시킴.
- 회의를 종료시킬 때는 좌측 하단의 '회의 종료' 버튼을 이용하여 종료하고,
회의가 종료된 이후에는 해당 프로젝트 탭에 진행한 회의의 '회의록' 이 저장됨.
- STT의 부정확성이 있을 수 있어,
Project 탭으로 이동해 방금 진행한 회의의 회의록을 수정 가능.
- 수정 버튼을 누르면 기록된 회의록이 위치하고, 해당 회의록의 내용을 수정한 이후
우측 하단의 버튼을 누르면 Jetson 내부의 ChromaDB에
임베딩 벡터로 저장 및 요약본을 만들어 Web에 송신.
- 요약이 완료된 후, 다시 프로젝트 탭으로 나와 해당 프로젝트의 상세보기를 보면,
요약된 회의록을 볼 수 있음.