

# 포팅메뉴얼

## 1. 개발환경

- 1.1 Frontend
- 1.2 Backend
- 1.3 Jetson Orin Nano 8GB
- 1.4 EC2 Server
- 1.5 GPU Server
- 1.6 Database
- 1.7 IDE
- 1.8 형상/이슈관리
- 1.9 애자일 도구
- 1.10 UI/UX

## 2. Jetson Orin Nano

- 2.1 부팅 불량 해결
- 2.2 JetPack 버전 호환성 참고사항
- 2.3 하드웨어 사양
- 2.4 소프트웨어 사양
- 2.5 성능 최적화

## 3. EC2 Server

- 3.1 Docker 접속
- 3.2 Docker 설치
- 3.3 Nginx 설치
- 3.4 Let's Encrypt 인증서 발급 - SSL인증서
- 3.5 ufw 포트 설정
- 3.6 nginx.conf - SSL 적용
- 3.7 EC2 Port

## 4. CI/CD 구축

- 4.1 Jenkinsfile
- 4.2 Frontend - Dockerfile
- 4.3 Backend - Dockerfile
- 4.4 Middleware - Dockerfile
- 4.5 **deploy\_frontend.sh** - 배포 스크립트 - 치료사 전용 웹
- 4.6 **deploy\_backend.sh** - 배포 스크립트 - 치료사 전용 웹
- 4.7 **deploy\_middle.sh** - 배포 스크립트 - Middleware
- 4.8 Jenkins 설정
- 4.9 Gitlab 연동
- 4.10 Mattermost 연동

## 5. GPU Server

- 5-1. Anaconda 설치

5-2 가상 환경 생성 및 사용

5-3. CUDA 지원되는 PyTorch 설치

## 6. 외부 서비스 이용

6.1 허깅페이스 세팅

6.2 Stable Diffusion 3 medium 모델 사용

6.3 TypecastAPI

6.4 WhisperAPI

6.5 OPENCV 및 DEEPFACE, MTCNN

6.6 COCO-SSD

# 1. 개발환경

## 1.1 Frontend

- HTML5, CSS3
- TypeScript
- React 18
- node 22

## 1.2 Backend

- Java 17, Python 3.10
- SpringBoot 3
  - Mybatis
  - JWT
  - Spring Security
  - Lombok
- Postman
- Gradle
- Flask 3.1
  - requirements.txt

```
absl-py==2.1.0
aiohappyeyeballs==2.4.6
```

aiohttp==3.11.12  
aiosignal==1.3.2  
annotated-types==0.7.0  
anyio==4.8.0  
astunparse==1.6.3  
async-timeout==5.0.1  
attrs==25.1.0  
bcrypt==4.2.1  
beautifulsoup4==4.13.1  
bidict==0.23.1  
blinker==1.9.0  
BTrees==6.1  
certifi==2025.1.31  
cffi==1.17.1  
charset-normalizer==3.4.1  
click==8.1.8  
colorama==0.4.6  
config==0.5.1  
contourpy==1.3.0  
cryptography==44.0.1  
cyclers==0.12.1  
deepface==0.0.93  
distro==1.9.0  
dnspython==2.7.0  
eventlet==0.39.0  
exceptiongroup==1.2.2  
filelock==3.17.0  
fire==0.7.0  
Flask==3.1.0  
Flask-Cors==5.0.0  
Flask-SocketIO==5.5.1  
Flask-SQLAlchemy==3.1.1  
flatbuffers==25.1.24  
fonttools==4.55.8  
frozenlist==1.5.0  
fsspec==2025.2.0  
gast==0.6.0  
gdown==5.2.0

```
gevent==24.11.1
gevent-websocket==0.10.1
google-pasta==0.2.0
greenlet==3.1.1
grpcio==1.70.0
gTTS==2.5.4
gunicorn==23.0.0
h11==0.14.0
h5py==3.12.1
httpcore==1.0.7
httpx==0.28.1
idna==3.10
importlib_metadata==8.6.1
importlib_resources==6.5.2
itsdangerous==2.2.0
Jinja2==3.1.5
jiter==0.8.2
keras==3.8.0
kiwisolver==1.4.7
libclang==18.1.1
llvmlite==0.44.0
Markdown==3.7
markdown-it-py==3.0.0
MarkupSafe==3.0.2
matplotlib==3.9.4
mdurl==0.1.2
ml-dtypes==0.4.1
more-itertools==10.6.0
mpmath==1.3.0
mtcnn==0.1.1
multidict==6.1.0
namex==0.0.8
networkx==3.4.2
numba==0.61.0
numpy==1.26.4
openai==1.63.0
openai-whisper==20240930
opencv-python==4.11.0.86
```

opt\_einsum==3.4.0  
optree==0.14.0  
packaging==24.2  
pandas==2.2.3  
persistent==6.1  
pillow==11.1.0  
propcache==0.2.1  
protobuf==5.29.3  
psutil==6.1.1  
py-cpuinfo==9.0.0  
PyAudio==0.2.14  
pycparser==2.22  
pydantic==2.10.6  
pydantic\_core==2.27.2  
pydub==0.25.1  
Pygments==2.19.1  
PyMySQL==1.1.1  
pyparsing==3.2.1  
pyscard==2.2.1  
PySocks==1.7.1  
python-dateutil==2.9.0.post0  
python-dotenv==1.0.1  
python-engineio==4.11.2  
python-socketio==5.12.1  
pytz==2025.1  
PyYAML==6.0.2  
regex==2024.11.6  
requests==2.32.3  
retina-face==0.0.17  
rich==13.9.4  
scipy==1.13.1  
seaborn==0.13.2  
simple-websocket==1.1.0  
six==1.17.0  
smartcard==0.3  
sniffio==1.3.1  
soupsieve==2.6  
SQLAlchemy==2.0.38

```
sympy==1.13.1
tensorboard==2.18.0
tensorboard-data-server==0.7.2
tensorflow==2.18.0
tensorflow-io-gcs-filesystem==0.31.0
termcolor==2.5.0
tf_keras==2.18.0
tiktoken==0.8.0
torch==2.6.0
torchvision==0.21.0
tqdm==4.67.1
transaction==5.0
typing_extensions==4.12.2
tzdata==2025.1
ultralitics==8.3.75
ultralitics-thop==2.0.14
urllib3==2.3.0
Werkzeug==3.1.3
whisper==1.1.10
wrapt==1.17.2
wsproto==1.2.0
yarl==1.18.3
zc.lockfile==3.0.post1
ZConfig==4.2
zipp==3.21.0
ZODB==6.0
zodbpickle==4.1.1
zope.deferredimport==5.0
zope.event==5.0
zope.interface==7.2
zope.proxy==6.1
```

## 1.3 Jetson Orin Nano 8GB

AI Performance	40 TOPS
GPU	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores
GPU Max Frequency	625 MHz
CPU	6-core Arm Cortex-A78AE v8.2 64-bit CPU 1.5MB L2 + 4MB L3
CPU Max Frequency	1.5 GHz
Memory	8GB 128-bit LPDDR5, 68 GB/s
Storage	- (SD Card Slot & external NVMe via M.2 Key M)
Video Encode	1080p30 supported by 1-2 CPU cores
Video Decode	1x 4K60 (H.265) / 2x 4K30 (H.265) 5x 1080p60 (H.265) / 11x 1080p30 (H.265)
CSI Camera	2x MIPI CSI-2 22-pin Camera Connectors
PCIe*	M.2 Key M slot with x4 PCIe Gen3 M.2 Key M slot with x2 PCIe Gen3 M.2 Key E slot
USB*	USB Type-A Connector: 4x USB 3.2 Gen2 USB Type-C Connector for UFP
Networking*	1xGbE Connector
Display	1x DisplayPort 1.2 (+MST) connector
Other I/O	40-Pin Expansion Header(UART, SPI, I2S, I2C, GPIO) 12-pin button header / 4-pin fan header microSD Slot / DC power jack
Power	7W - 15W
Mechanical	100 mm x 79 mm x 21 mm (Height includes feet, carrier board, module, and thermal solution)

- Ubuntu 22.04 ARM64(aarch64)
- JetPack 6.2
- Driver 540.4.0
- CUDA 12.6
- tokenizers 0.21.0
- transformers 4.48.2
- python 3.10
- torch 2.5.0
- torchvision 0.20.0
- SQLite3

## 1.4 EC2 Server

- Ubuntu 22.04 AMD64(x86\_64)
- Nginx

## 1.5 GPU Server

- Ubuntu 20.04 AMD64(x86\_64)
- Jupyter Notebook
- rootless
- JetPack 6.1
- Driver 535.183.01
- CUDA 12.2
- tokenizers 0.21.0
- transformers 4.48.2
- python 3.10
- torch 2.5.0
- torchvision 0.20.0

## 1.6 Database

- Mysql 8
- SQLite3

## 1.7 IDE

- IntelliJ, Datagrip, MysqlWorkbench, VSCode, Pycharm
- Git, Gitlab

## 1.8 형상/이슈관리

- Git
- Gitlab



## 1.9 애자일 도구

- Jira
- Mattermoset
- Notion

## 1.10 UI/UX

- Figma
- PowerPoint

# 2. Jetson Orin Nano

## 2.1 부팅 불량 해결



### 문제점

보드에 전원을 연결하면 켜지고 NVIDIA 화면이 나오고 화면이 꺼지면서 팬이 멈춘다.

→ Jetson Orin의 잦은 부팅 불량 문제

준비물 : Rufus(Etcher), SD Card Formatter, 점퍼케이블(Force Recovery Mode 실행하기 위함)

참고로 이미지 굽는 것은 최소 20~40분정도 소요됩니다.

### 참고 사이트

<https://developer.nvidia.com/embedded/learn/jetson-orin-nano-devkit-user-guide/howto.html#force-recovery-mode>

<https://operationcoding.tistory.com/217>

<https://seozero00.tistory.com/8>

<https://m.blog.naver.com/zeta0807/223169248857>

### 해결방법

1. usb 8기가 이상 하나 준비해서 데스크탑에서 포맷
2. sd 카드에는 jetpack 5.1.3 버전을 Rufus를 이용해 이미지를 굽습니다.
3. ubuntu 22.0.4 LTS 버전 ios 파일 다운 (공홈 참고)
  - a. 22.0.4 선택한 이유는 jetpack 6.2 버전 사용하기 위해서 입니다.
  - b. jetpack 6.2가 기존의 명세서 5.1.3 보다 성능적인 측면에서 우월하기 때문에 6.2로 업그레이드 해야 했습니다.
  - c. 참고로 UEFI 번호가 36.0 보다 작으면 jetpack 5.1.3 먼저 설치하고 6.x 버전을 설치할 수 있다고 합니다. 확인하는 방법은 jetson 부팅할 때 BIOS 모드로 켜서 들어가면 알 수 있어요.
4. Rufus 또는 Etcher를 활용해서 ubuntu 이미지를 usb에 굽습니다.
5. 데스크탑에 이미지를 구운 usb를 꼽고 부팅하면서 F2 연타하여 BIOS 모드에 들어가서 부팅 순서를 Ubuntu를 최상단위 위치 시켜두고 저장하고 나가서 실행합니다.

6. 기본적인 우분투 설정을 한 다음에, 구글에 nvidia sdk manager 검색하면 공홈에서 sdk manager deb 파일을 받을 수 있습니다.
7. 실행하면 nvidia 계정이 필요해서 회원가입 해야 합니다
8. sdk manager가 실행됐다면, jetson의 9, 10 핀(REC, GND)을 점퍼케이블로 연결하고 전원 케이블을 연결하여 전원을 켜면 Force Recovery Mode에 진입하게 되면서 sdk manager가 보드를 인식할 수 있게 됩니다.
9. 그리고 필요한 부분 체크해 가면서 설치하면 됩니다.
10. 설치가 됐다면 보호모드 해제하고 실행시키면 부팅문제를 해결된 모습을 볼 수 있습니다.

## 2.2 JetPack 버전 호환성 참고사항

NVIDIA SDK	SDK Version	Supported Host Operating System							
		Ubuntu					CentOS	RHEL	Debian
		16.04	18.04	20.04	22.04	24.04	8.2	8.2	10.8
JetPack	JetPack 4.x	✓	✓						
	JetPack 5.x		✓	✓					
	JetPack 6.x			✓	✓	✓ <sup>1</sup>			

## 2.3 하드웨어 사양

- Jetson Orin Nano Developer Kit 8GB
- MicroSD 카드 (64GB 이상 권장)
- USB 키보드/마우스
- HDMI 또는 DP 모니터
- USB-C 전원 어댑터 (5V 4A 권장)
- 이더넷 또는 Wi-Fi Dongle
- 모니터 입력(DP)
- USB C 타입

## 2.4 소프트웨어 사양

- Ubuntu 20.04 (Host PC)

- NVIDIA SDK Manager
- JetPack (Jetson Linux + CUDA + cuDNN + TensorRT 포함)

## 2.5 성능 최적화

```
sudo nvpmodel -m 0 # 최대 성능 모드  
sudo jetson_clocks # 클럭 최적화
```

## 3. EC2 Server

### 3.1 Docker 접속

```
ssh -i your-key.pem ubuntu@your-ec2-ip
```

### 3.2 Docker 설치

```
# 패키지 목록 업데이트  
sudo apt update  
  
# 필수 패키지 설치  
sudo apt install -y ca-certificates curl gnupg  
  
# 테스트  
docker ps
```

### 3.3 Nginx 설치

```
sudo apt update && sudo apt upgrade -y  
  
sudo apt install -y docker.io nginx certbot python3-certbot-nginx
```

### 3.4 Let's Encrypt 인증서 발급 - SSL인증서

```
sudo certbot --nginx -d 도메인이름
```

### 3.5 ufw 포트 설정

```
# EC2 포트 상태 확인  
sudo ufw status
```

```
# 해당 포트 개방  
sudo ufw allow 22  
sudo ufw allow 80
```

```
# firewall 활성화 상태 확인  
sudo ufw enable  
sudo ufw status verbose
```

```
sudo ufw allow 80/tcp # HTTP (필요한 경우만 허용)  
sudo ufw allow 443/tcp # HTTPS (SSL 사용)  
sudo ufw allow ssh # SSH 접근 허용 (안 하면 서버 접근 불가)  
sudo ufw deny 7001/tcp # ⚠ 백엔드 직접 접근 차단 (프록시만 사용)  
sudo ufw enable # 방화벽 활성화
```

### 3.6 nginx.conf - SSL 적용

```
worker_processes 1;  
  
events {  
    worker_connections 1024;  
}  
  
http {  
    include /etc/nginx/mime.types;  
    default_type application/octet-stream;  
    sendfile on;  
    keepalive_timeout 65;  
  
    server {
```

```

listen 80;
listen 443 ssl;
server_name 도메인이름;

ssl_certificate /etc/letsencrypt/live/도메인이름/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/도메인이름/privkey.pem;

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

error_page 404 /index.html;

# 🚀 백엔드 API 프록시 설정 (경로 유지)
location /api/ {
    rewrite ^/api/(.*)$ /$1 break; # ✅ /api/ 제거하고 백엔드로 전달
    proxy_pass http://manage-children:7001/;
    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;
}
}

```

## 3.7 EC2 Port

Port 번호	내용
80	HTTP
443	HTTPS
7001	manage-children 스프링서버
7260	GPU Server 의 API 요청받는 포트
9090	Jenkins

## 4. CI/CD 구축

### 4.1 Jenkinsfile

```

pipeline {
    agent any

    environment {
        IMAGE_NAME = "suhwany/aitalk" // ✅ 하나의 Docker Hub 레포지토리
        IMAGE_TAG_BACKEND = "backend-latest" // 백엔드 태그
        IMAGE_TAG_FRONTEND = "frontend-latest" // 프론트엔드 태그
        IMAGE_TAG_MIDDLE = "middle-latest" // 미들웨어 태그

        MATTERMOST_WEBHOOK = "https://meeting.ssafy.com/hooks/9p1665jgnjdn3msh9piyg1zpme" // ✅ Mattermost 웹훅 URL
    }

    stages {
        stage('Notify GitLab') {
            steps {
                echo 'Notify GitLab'
                updateGitlabCommitStatus name: 'build', state: 'pending'
            }
        }

        stage('Checkout') {
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'gitlabId', usernameVariable: 'GIT_USERNAME', passwordVariable: 'GIT_TOKEN')]) {
                        git branch: 'develop', url: "https://${GIT_USERNAME}:${GIT_TOKEN}@lab.ssafy.com/s12-webmobile3-sub1/S12P11E102.git"
                    }
                }
            }
        }
    }
}

```

```

}

// ✅ Backend 빌드 및 배포
stage('Build & Push Backend') {
    steps {
        script {
            withCredentials([usernamePassword(credentialsId: 'DockerId',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')])
{
                sh '''
                cd backend
                chmod +x ./gradlew
                ./gradlew clean build -x test "-Dorg.gradle.jvmargs=-Xmx2g"
                docker build -t ${IMAGE_NAME}:${IMAGE_TAG_BACKEND} .
                echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER"
--password-stdin
                docker push ${IMAGE_NAME}:${IMAGE_TAG_BACKEND}
                '''
            }
        }
    }
}

stage('Deploy Backend') {
    steps {
        sh '''
        chmod +x jenkins/scripts/deploy_backend.sh
        ./jenkins/scripts/deploy_backend.sh
        '''
    }
}

// ✅ Frontend 빌드 및 배포
stage('Build & Push Frontend') {
    steps {
        script {
            withCredentials([usernamePassword(credentialsId: 'DockerId',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')])


```



```

{
    sh '''
    cd frontend
    rm -rf node_modules package-lock.json dist
    npm install
    chmod +x node_modules/.bin/tsc
    npm run build
    docker build -t ${IMAGE_NAME}:${IMAGE_TAG_FRONTEND}
    .
    echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER"
--password-stdin
    docker push ${IMAGE_NAME}:${IMAGE_TAG_FRONTEND}
    '''
}
}
}
}

stage('Deploy Frontend') {
    steps {
        sh '''
        chmod +x jenkins/scripts/deploy_frontend.sh
        ./jenkins/scripts/deploy_frontend.sh
        '''
    }
}

//  Middle-tier 빌드 및 배포
stage('Build & Push Middleware') {
    steps {
        script {
            withCredentials([usernamePassword(credentialsId: 'DockerId',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')])
{
                sh '''
                cd middle
                docker build -t ${IMAGE_NAME}:${IMAGE_TAG_MIDDLE} .
                echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER"

```

```

--password-stdin
    docker push ${IMAGE_NAME}:${IMAGE_TAG_MIDDLE}
    '''
    }
  }
}

stage('Deploy Middleware') {
  steps {
    sh '''
    chmod +x jenkins/scripts/deploy_middle.sh
    ./jenkins/scripts/deploy_middle.sh
    '''
  }
}

post {
  success {
    script {
      sh '''
      curl -X POST -H 'Content-Type: application/json' --data '{
        "text": "✅ Jenkins Build & Deployment Success! 🎉"
      }' ${MATTERMOST_WEBHOOK}
      '''
    }
    updateGitlabCommitStatus name: 'build', state: 'success'
  }
  failure {
    script {
      sh '''
      curl -X POST -H 'Content-Type: application/json' --data '{
        "text": "❌ Jenkins Build Failed! 🔥"
      }' ${MATTERMOST_WEBHOOK}
      '''
    }
    updateGitlabCommitStatus name: 'build', state: 'failed'
  }
}

```

```
}  
}  
}
```

## 4.2 Frontend - Dockerfile

```
# 1 Node.js에서 Vite 빌드  
FROM node:22 AS builder  
WORKDIR /app  
  
# ✔ package.json & package-lock.json 복사 후 종속성 설치  
COPY package.json package-lock.json ./  
RUN npm install  
  
# ✔ 앱 코드 복사 & 빌드 실행  
COPY . .  
RUN npm run build  
  
# 2 Nginx로 배포  
FROM nginx:alpine  
WORKDIR /usr/share/nginx/html  
  
# ✔ 기존 HTML 파일 제거 후 빌드된 파일 복사  
RUN rm -rf /usr/share/nginx/html/*  
COPY --from=builder /app/dist /usr/share/nginx/html  
  
# ✔ Nginx 설정 복사  
COPY nginx.conf /etc/nginx/nginx.conf  
  
# ✔ 포트 개방  
EXPOSE 80  
  
# ✔ Nginx 실행  
CMD ["nginx", "-g", "daemon off;"]
```

## 4.3 Backend - Dockerfile

```
# 1. JDK 17 기반 이미지 사용 (빌드용)
FROM eclipse-temurin:17-jdk as builder

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 프로젝트 파일 복사
COPY . .

# 4. 실행 권한 추가 (chmod +x gradlew)
RUN chmod +x ./gradlew

# 5. Gradle 빌드 실행 (JAR 파일 생성)
RUN ./gradlew clean build -x test

# 6. 실행 환경 설정 (JDK 17)
FROM eclipse-temurin:17-jdk as runtime
WORKDIR /app

# 7. 빌드된 JAR 파일 복사
COPY --from=builder /app/build/libs/*.jar app.jar

# 8. 컨테이너 실행 시 JAR 실행
CMD ["java", "-jar", "app.jar"]
```

## 4.4 Middleware - Dockerfile

```
# 1. JDK 17 기반 이미지 사용 (빌드용)
FROM eclipse-temurin:17-jdk as builder

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 프로젝트 파일 복사
COPY . .

# 4. 실행 권한 추가 (chmod +x gradlew)
```

```
RUN chmod +x ./gradlew
```

```
# 5. Gradle 빌드 실행 (JAR 파일 생성)
```

```
RUN ./gradlew clean build -x test
```

```
# 6. 실행 환경 설정 (JDK 17)
```

```
FROM eclipse-temurin:17-jdk as runtime
```

```
WORKDIR /app
```

```
# 7. 빌드된 JAR 파일 복사
```

```
COPY --from=builder /app/build/libs/*.jar app.jar
```

```
# 8. 컨테이너 실행 시 JAR 실행
```

```
CMD ["java", "-jar", "app.jar"]
```

## 4.5 deploy\_frontend.sh - 배포 스크립트 - 치료사 전용 웹

```
#!/bin/bash
```

```
echo "🚀 EC2 Frontend 배포 시작..."
```

```
# 최신 Docker 이미지 가져오기
```

```
docker pull suhwany/aitalk:frontend-latest
```

```
# 기존 컨테이너 중지 및 삭제
```

```
docker stop manage-children-front || true
```

```
docker rm manage-children-front || true
```

```
# ✅ 네트워크 존재 여부 확인 후 생성
```

```
if ! docker network inspect my_network > /dev/null 2>&1; then
```

```
    echo "🔗 my_network 네트워크 생성"
```

```
    docker network create my_network
```

```
fi
```

```
# ✅ 프론트 컨테이너 실행 (Nginx 사용)
```

```
docker run -d --name manage-children-front \
```

```
    --network my_network \
```

```
-p 80:80 -p 443:443 \  
-v /etc/letsencrypt:/etc/letsencrypt:ro \  
suhwany/aitalk:frontend-latest
```

```
# 사용하지 않는 Docker 이미지 정리  
docker image prune -a -f
```

```
echo "✅ EC2 Frontend 배포 완료!"
```

## 4.6 deploy\_backend.sh - 배포 스크립트 - 치료사 전용 웹

```
#!/bin/bash
```

```
echo "🚀 EC2 Backend 배포 시작..."
```

```
# 최신 Docker 이미지 가져오기  
docker pull suhwany/aitalk:backend-latest
```

```
# 기존 컨테이너 중지 및 삭제  
docker stop manage-children || true  
docker rm manage-children || true
```

```
# ✅ 네트워크 존재 여부 확인 후 생성  
if ! docker network inspect my_network > /dev/null 2>&1; then  
  echo "🔗 my_network 네트워크 생성"  
  docker network create my_network  
fi
```

```
# ✅ 백엔드 컨테이너 실행 (네트워크 포함)  
docker run -d --name manage-children \  
  --network my_network \  
  -p 7001:7001 \  
  suhwany/aitalk:backend-latest
```

```
# 사용하지 않는 Docker 이미지 정리  
docker image prune -a -f
```

```
echo "✅ EC2 Backend 배포 완료!"
```

## 4.7 deploy\_middle.sh - 배포 스크립트 - Middleware

```
#!/bin/bash
echo "🚀 EC2 Middleware 배포 시작..."

docker pull suhwany/aitalk:middle-latest

docker stop jetson-middle || true
docker rm jetson-middle || true

docker run -d --name jetson-middle \
  -p 7260:7260 \
  -v /home/ubuntu/images:/home/ubuntu/images \
  suhwany/aitalk:middle-latest

docker image prune -a -f
echo "✅ EC2 Middleware 배포 완료!"
```

## 4.8 Jenkins 설정

### 1. docker 방식 설치 시

#### 1) jenkins container 생성 및 구동

```
cd /home/ubuntu && mkdir jenkins-data
```

```
sudo ufw allow *9090*/tcp
```

```
sudo ufw reload
```

```
sudo ufw status
```

```
sudo docker run -d -p 9090:9090 -v /home/ubuntu/jenkins-data:/var/jenkins_home --name jenkins jenkins/jenkins:its
```

```
sudo docker logs jenkins
```

```
sudo docker stop jenkins
```

```
sudo docker ps -a
```

#### 2) 환경 설정 변경 (매우 중요)

```
cd /home/ubuntu/jenkins-data
```

```
mkdir update-center-rootCAs
```

```
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O ./update-center-rootCAs/update-center.crt
```

```
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tenant/update-center.json#' ./hudson.model.UpdateCenter.xml
```

```
sudo docker restart jenkins
```

### 3) 주요 명령어

```
sudo docker start jenkins
```

```
sudo docker stop jenkins
```

```
sudo docker logs jenkins
```

```
sudo docker logs -f jenkins
```

### 4) config 보안 설정 확인(매우 중요)

```
vi /home/ubuntu/jenkins-data/config.xml
```

```
<useSecurity>true</useSecurity>
```

```
...(중략)
```

```
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
```

```
<disableSignup>true</disableSignup>
```

```
# 젠킨스 접속(url)
```

```
ubuntu@도메인이름:9090
```

```
# 초기 비밀번호 모를 때 찾는 방법
```


```
sudo docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

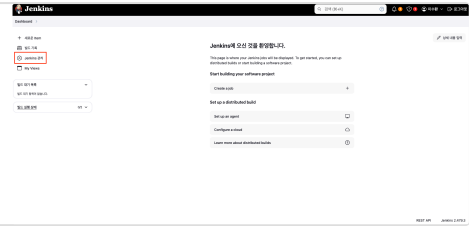
## 4.9 Gitlab 연동



## Jenkins Gitlab 연동

Jenkins와 Gitlab 연동을 빠르게 해보자!

 <https://velog.io/@getbrave/Jenkins-Gitlab-%EC%97%B0%EB%8F%99>



## 4.10 Mattermost 연동

Jenkins관리

→ plugin 에서 Mattermost Notification Plugin 설치

→ system

→ Global Mattermost Notifier Settings

# Endpoint

mattermost → 통합 -> 전체 Incoming Webhook → Incoming Webhook 추가 -> 제목, 설명 작성 -> 채널(알림을 보내고 싶은 채널) -> 저장

생성된 웹훅에서 URL 주소가 Endpoint

# Channel

jenkins

# Build Server URL

jenkins url

## 5. GPU Server

### 5-1. Anaconda 설치

# 공식사이트에서 설치

wget [https://repo.anaconda.com/archive/Anaconda3-2024.10-1-Linux-x86\\_64.sh](https://repo.anaconda.com/archive/Anaconda3-2024.10-1-Linux-x86_64.sh)

# 사용자 홈 디렉토리 (~/.anaconda3/) 에 설치

bash Anaconda3-2024.10-1-Linux-x86\_64.sh

```
# 아나콘다 활성화
source ~/.bashrc
conda --version
```

## 5-2 가상 환경 생성 및 사용

```
# 새로운 가상 환경 생성
conda create --name myenv python=3.10

# 생성한 가상 환경 활성화
conda activate myenv

# conda 또는 pip을 사용해 설치 (pip 추천)
pip install numpy pandas matplotlib
conda install pytorch torchvision torchaudio -c pytorch
```

## 5-3. CUDA 지원되는 PyTorch 설치

```
# gpu 및 cuda 확인
nvidia-smi

# cuda 버전에 맞는 pytorch 설치
https://pytorch.org/

# gpu 사용 여부 테스트
import torch
print(torch.cuda.is_available()) # True가 나오면 GPU 사용 가능
print(torch.cuda.device_count()) # 사용 가능한 GPU 개수 출력
```

# 6. 외부 서비스 이용

## 6.1 허깅페이스 세팅

```
# 파이썬 버전 확인 (3.8 이상 사용)
python --version
```

```
# 가상환경에서 세팅
conda create 가상환경명 python=3.10
conda activate 가상환경명

# 허깅페이스 cli 설치
pip install --upgrade huggingface_hub
pip install --upgrade diffusers transformers accelerate safetensors

# 허깅페이스 로그인
huggingface-cli login
```

## 6.2 Stable Diffusion 3 medium 모델 사용

```
# app.py

from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
import model # model.py에서 함수 호출
import os

app = Flask(__name__)
CORS(app) # 모든 도메인에서 요청 허용

# 생성된 이미지를 접근 가능하게 설정
IMAGE_DIR = "/home/디렉토리명/"
os.makedirs(IMAGE_DIR, exist_ok=True)

@app.route("/generate", methods=["POST"])
def generate():
    data = request.get_json()
    prompt = data.get("prompt")

    if not prompt:
        return jsonify({"error": "No prompt provided"}), 400

    # 이미지 생성 요청 -> model.py에서 처리
```

```

result = model.generate_image(prompt)

if isinstance(result, str) and result.startswith(IMAGE_DIR):
    # 생성된 이미지 경로를 URL로 변환
    filename = os.path.basename(result)
    image_url = f"http://<ec2 ip>:포트/images/{filename}"
    return jsonify({"image_url": image_url, "filepath": result})
else:
    return jsonify({"error": result}), 500

# 이미지 폴더를 정적 파일로 제공
@app.route("/images/<filename>")
def serve_image(filename):
    return send_from_directory(IMAGE_DIR, filename)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5220)

# model.py
import os
import torch
import unicodedata # ✅ import 추가
import re # ✅ 추가
from diffusers import StableDiffusion3Pipeline

# GPU 메모리 캐시 비우기 (메모리 누수 방지)
torch.cuda.empty_cache()

# 모델 로드 (GPU에서 실행)
pipe = StableDiffusion3Pipeline.from_pretrained(
    "stabilityai/stable-diffusion-3-medium-diffusers",
    torch_dtype=torch.float16
).to("cuda")

# 이미지 저장 디렉토리
IMAGE_DIR = "/home/디렉토리명/"
os.makedirs(IMAGE_DIR, exist_ok=True)

```

```

# ✅ **파일명 정규화 함수**
def normalize_filename(prompt):
    """파일명으로 사용할 수 있도록 프롬프트를 정규화"""
    filename = unicodedata.normalize('NFKD', prompt).encode('ascii', 'ignore').decode('utf-8') # 유니코드 정규화
    filename = filename.replace(" ", "_") # 공백을 "_"로 변경
    filename = re.sub(r'[^a-zA-Z0-9_-]', '', filename) # 영문자, 숫자, `_`, `-`만 허용
    return filename

# ✅ **이미지 생성 함수**
def generate_image(prompt):
    filename = f"{normalize_filename(prompt)}.png"
    filepath = os.path.join(IMAGE_DIR, filename)

    try:
        # Stable Diffusion을 사용해 이미지 생성
        image = pipe(
            prompt,
            negative_prompt="blurry, distorted, low resolution, extra objects, multiple subjects, cluttered background, surreal, abstract, overexposed, underexposed, unnatural lighting, unrealistic proportions, deformed faces, extra limbs, low contrast, bad anatomy, poorly drawn, grainy, watermark, text, caption, cropped, mutated, glitch, nsfw",
            num_inference_steps=28,
            guidance_scale=7.0
        ).images[0]

        # 이미지 저장
        image.save(filepath)

        return filepath # 생성된 이미지 파일 경로 반환

    except Exception as e:
        return str(e) # 오류 메시지 반환

```

## 6.3 TypecastAPI

## 1. 조직 생성



Organization / Overview

### 조직 관리

조직에 등록된 캐릭터와 API 정보를 관리합니다.

API 정보

API 키명	만료 시간	내용 제한
__plt8sj....ELsv	30	100

캐릭터



아찌  
한국어  
66596206b7bd6e89c3a2c54e

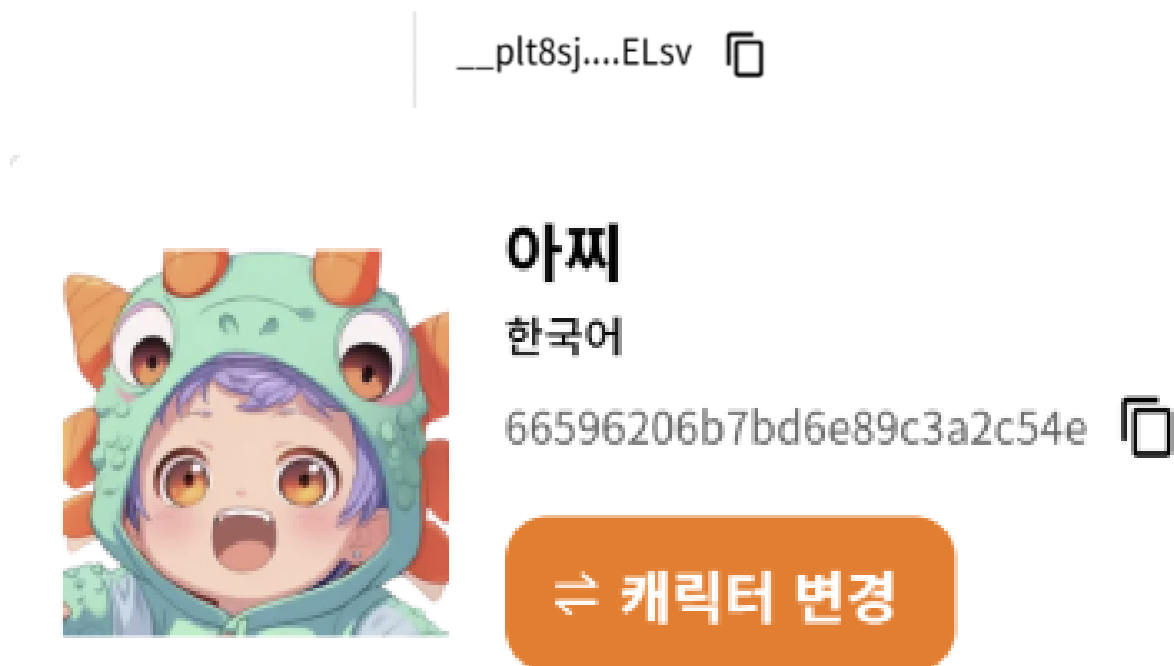
캐릭터 변경



다들 온 캐릭터를  
만들어보고 싶으세요?

캐릭터 변경

## 2. API KEY, VOICE KEY 복사



\_\_plt8sj....ELsv

아찌  
한국어  
66596206b7bd6e89c3a2c54e

캐릭터 변경

.env

```
TYPECAST_API_KEY=__plt8sjAaMiSnGrWXLukjDdGV279pRX4kkSLGdHnELsv
TYPECAST_VOICE_ID=66596206b7bd6e89c3a2c54e
```

speech\_service

```

from dotenv import load_dotenv
load_dotenv()
TYPECAST_API_KEY = os.getenv("TYPECAST_API_KEY")
TYPECAST_ACTOR_ID = os.getenv("TYPECAST_VOICE_ID")

def text_to_speech(text):
    """
    Typecast API를 사용하여 텍스트를 음성으로 변환한 후, Base64 MP3로 변환하여 번
    Polling 방식을 이용하여 음성 합성이 완료될 때까지 대기한 후, 결과를 다운로드한다.
    """
    if not TYPECAST_API_KEY or not TYPECAST_ACTOR_ID:
        logging.error("❌ Typecast API Key 또는 Actor ID가 설정되지 않았습니다.")
        return None

    api_url = "https://typecast.ai/api/speak"
    headers = {
        "Authorization": f"Bearer {TYPECAST_API_KEY}",
        "Content-Type": "application/json"
    }
    payload = json.dumps({
        "actor_id": TYPECAST_ACTOR_ID,
        "text": text,
        "lang": "auto",
        "xapi_audio_format": "mp3"
    })

    try:
        # ✅ Typecast API 요청 (음성 생성)
        response = requests.post(api_url, headers=headers, data=payload)
        if response.status_code != 200:
            logging.error(f"❌ Typecast API 오류: {response.status_code} - {response}")
            return None

        # ✅ 응답에서 Polling URL 추출
        result = response.json()
        speak_v2_url = result["result"]["speak_v2_url"]
        logging.info(f"🔊 Polling 시작... URL: {speak_v2_url}")

```

```

# ✅ Polling (최대 60초 동안 대기)
for _ in range(60):
    time.sleep(1) # 1초 간격으로 상태 확인
    poll_response = requests.get(speak_v2_url, headers=headers)
    poll_result = poll_response.json()["result"]

    if poll_result["status"] == "done":
        audio_download_url = poll_result["audio_download_url"]
        logging.info(f"🎉 음성 합성 완료! 다운로드 URL: {audio_download_url}")
        break
    else:
        logging.info(f"⌚ 음성 처리 중... (현재 상태: {poll_result['status']})")
else:
    logging.error("❌ 음성 합성 시간이 초과되었습니다.")
    return None

# ✅ 오디오 파일 다운로드
audio_response = requests.get(audio_download_url)
if audio_response.status_code != 200:
    logging.error(f"❌ 음성 파일 다운로드 실패: {audio_response.status_code}")
    return None

# ✅ Base64로 변환 후 반환
audio_base64 = base64.b64encode(audio_response.content).decode('utf-8')
return audio_base64

except Exception as e:
    logging.error(f"❌ Typecast API 요청 중 오류 발생: {e}")
    return None

```

## 6.4 WhisperAPI

### 1. OpenAI 키 발급



**Create new secret key**

Owned by

**You** Service account

This API key is tied to your user and can make requests against the selected project. If you are removed from the organization or project, this key will be disabled.

Name Optional

My Test Key

Project

Select project...

Permissions

**All** Restricted Read only

Cancel Create secret key

## 2. 필요 라이브러리

```
openai==1.63.0
openai-whisper==20240930
```

.env

```
OPENAI_API_KEY=sk-proj-OuK51CIExu6gAmX3Giu1UQQAuDc-n0_W34fkt4d2
```

speech\_service



```
try:
    wav_io = BytesIO()
    with wave.open(wav_io, "wb") as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(p.get_sample_size(FORMAT))
        wf.setframerate(RATE)
        wf.writeframes(full_audio)
    wav_io.seek(0)


    # 🔥 임시 WAV 파일 저장 후 OpenAI Whisper 사용
```


```

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as tmp:
    tmp.write(wav_io.getvalue())
    tmp_path = tmp.name

with open(tmp_path, "rb") as audio_file:
    result = openai.audio.transcriptions.create(
        model="whisper-1",
        file=audio_file,
        language="ko",
        temperature=0.2
    )
text = result.text.strip()

if text:
    logging.info(f" 변환된 텍스트: {text}")
    Thread(target=get_gpt_response, args=(text, child_id), daemon=True).start()
else:
    logging.warning(f" 음성에서 텍스트를 추출하지 못함")
    is_tts_playing = False

except Exception as e:
    logging.error(f" 텍스트 변환 오류: {e}")
    is_tts_playing = False

finally:
    #  임시 파일 삭제
    if os.path.exists(tmp_path):
        os.remove(tmp_path)

```

## 6.5 OPENCV 및 DEEPPFACE, MTCNN

홈페이지에서 어떻게 사용하는지 가입, api사용법, 코드, 필요한 라이브러리, 설치 파일 등

### 1. 환경설정`

- 필수 패키지 설치

아래 패키지를 설치해야함.

```
pip install opencv-python numpy pandas deepface mtcnn
```

또한, **OpenCV** 가 카메라를 인식할 수 있도록 **V4L2** 백엔드를 활성화.

- 필요 라이브러리 확인

```
import cv2
import numpy as np
import pandas as pd
import os
from deepface import DeepFace
from mtcnn import MTCNN
```

---

## 2. 프로젝트 구조

```
face_recognition_project/
|— main.py          # 메인 실행 파일
|— face_db.csv      # 얼굴 임베딩 데이터 저장 파일
└— requirements.txt # 필요한 패키지 목록
```

---

## 3. 주요기능

- 카메라 확인

```
get_available_camera()
```

사용 가능한 카메라를 검

- 치료사 얼굴 등록

```
register_user_face(therapist_id: int, therapist_name: str)
```

치료사의 얼굴을 등록하고, `face_db.csv`에 저장.

- 치료사 얼굴 인증

```
verify_user_face()
```

실시간으로 치료사의 얼굴을 감지하고 인증을 시도.

#### 4. 실행 방법

- 얼굴 등록

```
register_user_face(therapist_id=1, therapist_name="홍길동")
```

치료사의 얼굴을 촬영하여 등록.

- 얼굴 인증

```
verify_user_face()
```

등록된 얼굴과 비교하여 인증.

#### 5. API 사용법

- API 엔드포인트

메서드	엔드포인트	설명
POST	<code>/register_face</code>	치료사 얼굴 등록
POST	<code>/verify_face</code>	얼굴 인증

- 요청 및 응답 예시
  - 치료사 얼굴 등록

**요청:**

```
{
  "therapist_id": 1,
  "therapist_name": "홍길동"
}
```

응답:

```
{
  "status": 201,
  "message": "✅ 얼굴 등록 완료!",
  "data": {
    "therapist_id": 1,
    "therapist_name": "홍길동"
  }
}
```

- 치료사 얼굴 인증

응답:

```
{
  "status": 200,
  "message": "✅ 인증 성공!",
  "therapist_id": 1,
  "therapist_name": "홍길동",
  "similarity": 0.82
}
```

user\_face\_recognition.py

```
import cv2
import numpy as np
import pandas as pd
import os
from deepface import DeepFace
from mtcnn import MTCNN
```

```

DB_PATH = "face_db.csv" # 치료사 얼굴 DB 파일
detector = MTCNN() # MTCNN 사용
THRESHOLD = 0.7 # 인증 임계값 (테스트에 따라 조정)
SECONDARY_THRESHOLD = 0.65 # 보조 임계값 (필요 시 활용)

def get_available_camera():
    """사용 가능한 카메라 찾기"""
    for i in range(5): # 최대 5개 카메라 검사
        cap = cv2.VideoCapture(i, cv2.CAP_V4L2) # V4L2 백엔드 사용
        if cap.isOpened():
            print(f"✅ 사용 가능한 카메라: {i}번")
            cap.release()
            return i
    print("🚫 사용 가능한 카메라가 없습니다!")
    return None

def warmup_camera(cap):
    """카메라 워밍업 (3프레임 스킵)"""
    for _ in range(3):
        ret, _ = cap.read()
        if ret:
            print("[INFO] 카메라 워밍업 완료")
            break

def align_face(frame, face):
    """
    얼굴 정렬 함수.
    MTCNN의 검출 결과(face dict)에서 'box'와 'keypoints'를 사용하여,
    좌우 눈의 각도를 계산하고, 해당 각도로 얼굴 영역을 회전시킵니다.
    """
    box = face['box'] # [x, y, w, h]
    keypoints = face['keypoints']
    x, y, w, h = box
    # 원본 frame에서 얼굴 영역 crop
    face_img = frame[y:y + h, x:x + w]
    # 좌우 눈 좌표 (얼굴 영역 기준으로 조정)

```

```

left_eye = (keypoints['left_eye'][0] - x, keypoints['left_eye'][1] - y)
right_eye = (keypoints['right_eye'][0] - x, keypoints['right_eye'][1] - y)
dx = right_eye[0] - left_eye[0]
dy = right_eye[1] - left_eye[1]
angle = np.degrees(np.arctan2(dy, dx))
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, angle, 1)
aligned_face = cv2.warpAffine(face_img, M, (w, h))
return aligned_face

def preprocess_face(face_img):
    """얼굴 전처리: BGR→RGB 변환 후 160x160 리사이즈"""
    face_img = cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB)
    face_img = cv2.resize(face_img, (160, 160))
    return face_img

def extract_embedding(processed_face, model="ArcFace"):
    """DeepFace를 이용해 얼굴 임베딩 추출 (512차원)"""
    try:
        analysis = DeepFace.represent(
            processed_face,
            model_name=model,
            detector_backend="mtcnn",
            enforce_detection=False
        )
        if not analysis or "embedding" not in analysis[0]:
            return None
        embedding = np.array(analysis[0]['embedding'], dtype=float)
        # 128차원 벡터가 나오면 512차원으로 패딩
        if embedding.shape[0] == 128:
            embedding = np.pad(embedding, (0, 384), mode="constant")
        return embedding
    except Exception as e:
        print(f"[ERROR] 얼굴 벡터 추출 오류: {e}")
        return None

```

```

def register_user_face(therapist_id: int, therapist_name: str):
    """
    치료사 얼굴 등록:
    - 사용 가능한 카메라에서 최대 15회 시도 중 5개 품질 좋은 샘플 수집
    - 얼굴 정렬 및 전처리 후 임베딩 추출하여 CSV 파일에 저장
    """
    camera_index = get_available_camera()
    if camera_index is None:
        return {"status": 503, "message": "사용 가능한 카메라가 없습니다."}

    cap = cv2.VideoCapture(camera_index, cv2.CAP_V4L2)
    if not cap.isOpened():
        return {"status": 503, "message": "카메라 연결 오류."}

    warmup_camera(cap)

    # DB 파일 로드 (없으면 새로 생성)
    if not os.path.exists(DB_PATH):
        df = pd.DataFrame(columns=["therapist_id", "therapist_name"] + [f"v{i}"
        else:
        df = pd.read_csv(DB_PATH, dtype={'therapist_id': int, 'therapist_name': s
    # 동일 therapist_id가 있으면 제거
    df = df[df["therapist_id"] != therapist_id]

    samples = []
    attempts = 0
    while len(samples) < 5 and attempts < 15:
        ret, frame = cap.read()
        attempts += 1
        if not ret:
            print("[ERROR] 이미지 캡처 실패. 다시 시도 중...")
            continue
        frame = cv2.flip(frame, 1)
        faces = detector.detect_faces(frame)
        if not faces:
            print("[WARN] 얼굴이 감지되지 않음. 다시 시도 중...")
            continue

```



```

for face in faces:
    if face.get('confidence', 0) < 0.90:
        print("[WARN] 얼굴 감지 신뢰도 낮음. 스킵.")
        continue
    # 얼굴 정렬 적용
    aligned_face = align_face(frame, face)
    processed_face = preprocess_face(aligned_face)
    embedding = extract_embedding(processed_face)
    if embedding is None or embedding.shape[0] != 512:
        print("[ERROR] 임베딩 오류 또는 크기 불일치! 다시 시도 중...")
        continue
    samples.append([therapist_id, therapist_name] + embedding.tolist())
    print(f"[INFO] 샘플 {len(samples)} 수집됨.")
    if len(samples) == 5:
        break

```

```

cap.release()
cv2.destroyAllWindows()

```

```

if samples:
    new_data = pd.DataFrame(samples, columns=df.columns)
    df = pd.concat([df, new_data], ignore_index=True)
    df.to_csv(DB_PATH, index=False, float_format='%.6f')
    return {"status": 201, "message": f"✅ 얼굴 등록 완료!", "data": {
        "therapist_id": therapist_id,
        "therapist_name": therapist_name
    }}
return {"status": 400, "message": "❌ 얼굴 등록 실패. 다시 시도해주세요."}

```

```

def verify_user_face():

```

```

    """

```

```

    치료사 얼굴 인증:

```

- 사용 가능한 카메라에서 10프레임 캡처 후, 얼굴 정렬 및 전처리하여 임베딩 추출
- 품질이 높은 상위 3개 프레임의 평균 임베딩을 계산하고, 등록된 각 치료사의 평균 임베딩과 비교

```

    """

```

```

    camera_index = get_available_camera()

```

```

    if camera_index is None:

```

```

    return {"status": 503, "message": "사용 가능한 카메라가 없습니다."}

cap = cv2.VideoCapture(camera_index, cv2.CAP_V4L2)
if not cap.isOpened():
    return {"status": 503, "message": "카메라 연결 오류."}

warmup_camera(cap)

if not os.path.exists(DB_PATH):
    return {"status": 401, "message": "등록된 얼굴 데이터가 없습니다."}

df = pd.read_csv(DB_PATH, dtype={'therapist_id': int, 'therapist_name': str})
if df.empty:
    return {"status": 401, "message": "등록된 얼굴이 없습니다."}

# 각 therapist_id별 평균 임베딩 계산
grouped = df.groupby("therapist_id")
avg_embeddings = {}
names = {}
for therapist_id, group in grouped:
    names[therapist_id] = group["therapist_name"].iloc[0]
    emb_array = group.iloc[:, 2:].values.astype(float)
    avg_embeddings[therapist_id] = np.mean(emb_array, axis=0)

collected_embeddings = []
quality_scores = []
# 10 프레임 캡처
for _ in range(10):
    ret, frame = cap.read()
    if not ret:
        continue
    frame = cv2.flip(frame, 1)
    faces = detector.detect_faces(frame)
    if not faces:
        continue
    face = faces[0]
    if face.get('confidence', 0) < 0.90:
        continue

```

```

aligned_face = align_face(frame, face)
processed_face = preprocess_face(aligned_face)
embedding = extract_embedding(processed_face)
if embedding is not None and embedding.shape[0] == 512:
    collected_embeddings.append(embedding)
    quality_scores.append(face.get('confidence', 0))

cap.release()
cv2.destroyAllWindows()

if not collected_embeddings:
    return {"status": 401, "message": "얼굴을 감지하지 못했습니다."}

collected_embeddings = np.array(collected_embeddings)
if len(collected_embeddings) > 3:
    # 상위 3개의 신뢰도 높은 프레임 선택
    indices = np.argsort(quality_scores)[::-1][:3]
    selected_embeddings = collected_embeddings[indices]
else:
    selected_embeddings = collected_embeddings
live_embedding = np.mean(selected_embeddings, axis=0)

best_match = None
best_similarity = 0
for therapist_id, avg_emb in avg_embeddings.items():
    similarity = np.dot(live_embedding, avg_emb) / (np.linalg.norm(live_embe
    if similarity > best_similarity:
        best_similarity = similarity
        best_match = therapist_id

if best_match and best_similarity > THRESHOLD:
    return {
        "status": 200,
        "message": "✅ 인증 성공!",
        "therapist_id": int(best_match),
        "therapist_name": names[best_match],
        "similarity": round(best_similarity, 3)
    }

```

```
return {"status": 401, "message": "❌ 인증 실패. 등록된 얼굴과 일치하지 않습니다"}
```

## 6.6 COCO-SSD

### 1. tensorflow 설치 (Node.js)

```
npm install @tensorflow/tfjs @tensorflow-models/coco-ssd
```

### 2. 사용

import

```
import * as cocoSsd from '@tensorflow-models/coco-ssd';  
import * as tf from '@tensorflow/tfjs';
```

CameraScanPage.tsx

```
import { useEffect, useRef, useState } from 'react';  
import { useNavigate, useLocation } from 'react-router-dom';  
import CurrentUserText from '../components/Texts/CurrentUserText';  
import LogoutButton from '../components/Buttons/LogoutButton';  
import NavbarContainer from '../components/Common/NavbarContainer';  
import BackPlaySelectButton from '../components/Common/BackPlaySelectButton';  
import * as cocoSsd from '@tensorflow-models/coco-ssd';  
import * as tf from '@tensorflow/tfjs';  
import { HStack } from '@chakra-ui/react';  
import './CameraScanPage.css';  
import { useSelector } from 'react-redux';  
import { RootState } from '../feature/store';  
  
// ✅ 직접 DetectedObject 타입 정의  
type DetectedObject = {  
  bbox: [number, number, number, number]; // [x, y, width, height]  
  class: string;  
  score: number;  
}
```

```

};

export default function CameraScanPage() {
  const currentUser = useSelector((state: RootState) ⇒ state.user.currentUser);
  const navigate = useNavigate();
  const location = useLocation();
  const scheduleId = location.state?.scheduleId; // PlaySelectPage에서 전달받은
  const streamRef = useRef<MediaStream | null>(null);
  const videoRef = useRef<HTMLVideoElement | null>(null);
  const canvasRef = useRef<HTMLCanvasElement | null>(null);
  const [model, setModel] = useState<ReturnType<typeof cocoSsd.load> | null>(
    null,
  );
  const [isDetecting, setIsDetecting] = useState(false);
  const isDataSentRef = useRef(false); // ✅ 중복 실행 방지용 useRef
  const isDetectingRef = useRef(true);

  // ✅ 추가된 변수 (객체 인식 유지 시간 체크)
  const CONFIDENCE_THRESHOLD = 0.7; // 최소 확률 임계값
  const DETECTION_DURATION = 1000; // 유지해야 하는 시간(ms)

  let detectedObject: string | null = null;
  let detectedStartTime: number | null = null;

  useEffect(() ⇒ {
    // ✅ 모델 로드
    const loadModel = async () ⇒ {
      await tf.ready();
      const loadedModel = await cocoSsd.load();
      setModel(loadedModel);
      console.log('✅ COCO-SSD 모델 로드 완료');
    };

    loadModel();
  }, []);

  useEffect(() ⇒ {
    // startCamera 함수가 비디오 엘리먼트에 연결한 스트림을 streamRef에 저장하도록

```

```

const startCamera = async () => {
  try {
    const stream = await navigator.mediaDevices.getUserMedia({
      video: { facingMode: 'user' },
    });
    // stream을 별도의 ref에 저장
    streamRef.current = stream;
    if (videoRef.current) {
      videoRef.current.srcObject = stream;
    }
  } catch (error) {
    console.error('❌ 카메라 접근 오류:', error);
  }
};

startCamera();

// ✅ cleanup: 컴포넌트 언마운트 시 camera 스트림 종료
return () => {
  if (streamRef.current) {
    streamRef.current.getTracks().forEach((track) => track.stop());
    console.log('✅ 카메라 스트림 종료 완료');
  }
};
}, []);

useEffect(() => {
  let animationId: number;

  const detectObjects = async () => {
    if (!model || !videoRef.current || !canvasRef.current) return;
    if (isDataSentRef.current || !isDetectingRef.current) return; // ✅ 데이터가 전

    const video = videoRef.current;
    if (video.readyState !== 4) {
      requestAnimationFrame(detectObjects);
      return;
    }
  }

```

```

const canvas = canvasRef.current;
const ctx = canvas.getContext('2d');
if (!ctx) return;

canvas.width = video.videoWidth || video.clientWidth;
canvas.height = video.videoHeight || video.clientHeight;
ctx.clearRect(0, 0, canvas.width, canvas.height);

const predictions: DetectedObject[] = await model.detect(video);

// ✅ 'person' 객체는 아예 필터링 (화면에 표시 X)
const filteredPredictions = predictions.filter(
  (prediction) ⇒ prediction.class !== 'person',
);

// ✅ 모든 감지된 객체를 화면에 표시 (person 제외)
filteredPredictions.forEach((prediction) ⇒ {
  const [x, y, width, height] = prediction.bbox;
  ctx.strokeStyle = 'rgba(0, 255, 0, 0.8)';
  ctx.lineWidth = 7;
  ctx.strokeRect(x, y, width, height);

  ctx.fillStyle = 'rgba(0, 255, 0, 0.8)';
  ctx.font = '18px Arial';
  ctx.fillText(
    `${prediction.class} (${(prediction.score * 100).toFixed(1)}%)`,
    x,
    y - 5,
  );
});

// ✅ 특정 객체(person 제외) + 확률 0.7 이상인 경우만 유지 시간 체크
const highConfidencePredictions = filteredPredictions.filter(
  (prediction) ⇒ prediction.score >= CONFIDENCE_THRESHOLD,
);

if (highConfidencePredictions.length > 0) {

```

```

const bestPrediction = highConfidencePredictions[0]; // 확률이 가장 높은 객체
const objectName = bestPrediction.class;
setIsDetecting(true); // 감지 중 상태 설정

if (detectedObject === objectName) {
  // 같은 객체가 계속 감지되는 경우
  if (
    detectedStartTime &&
    Date.now() - detectedStartTime >= DETECTION_DURATION
  ) {
    sendToBackend(objectName);
    detectedObject = null; // 한 번 보냈으면 초기화
    detectedStartTime = null;
  }
} else {
  // 새로운 객체가 감지됨 → 타이머 초기화
  detectedObject = objectName;
  detectedStartTime = Date.now();
}
} else {
  // 감지된 객체가 없으면 초기화
  setIsDetecting(false);
  detectedObject = null;
  detectedStartTime = null;
}

animationId = requestAnimationFrame(detectObjects);
};

if (videoRef.current) {
  videoRef.current.onloadedmetadata = () => {
    detectObjects();
  };
}

return () => cancelAnimationFrame(animationId);
}, [model]);

```



```

// ✅ 백엔드 전송 함수 수정 (데이터 전송 후 state와 함께 페이지 이동)
const sendToBackend = async (objectName: string) => {
  if (!scheduleId) {
    console.error('❌ scheduleId가 없습니다.');
```

return;

}

```

  if (isDataSentRef.current) {
    console.log('⚠️ 이미 데이터가 전송됨. 중복 전송 방지');
    isDetectingRef.current = false;
    return;
  }

  const data = {
    schedule_id: scheduleId,
    word: objectName,
  };

  console.log('📡 백엔드로 데이터 전송:', data);
  isDataSentRef.current = true;
  isDetectingRef.current = false;

  // ✅ 백엔드 응답 없이 먼저 페이지 이동 (백엔드 응답을 기다리지 않음)
  navigate('/camera-img-generate', { state: { data } });

  try {
    const response = await fetch('http://localhost:5000/play/camera-scan', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      throw new Error('❌ 서버 응답 오류: ${response.status}');
    }
  }

```

```

console.log('✅ 데이터 전송 성공!');
const imageData = await response.json();

// ✅ 백엔드 응답을 다음 페이지에서 업데이트하도록 설정
window.dispatchEvent(
  new CustomEvent('backendResponse', { detail: imageData }),
);
} catch (error) {
  console.error('❌ 데이터 전송 실패:', error);
  isDataSentRef.current = false;
  isDetectingRef.current = true;
}
};

return (
  <div className="BackgroundContainer">
    <div className="BackgroundImage"></div>
    <NavbarContainer>
      <HStack gap={315}>
        <BackPlaySelectButton className="CustomMarginTop" />
        {/* 로그인 한 경우에만 치료사의 이름이 렌더링되도록 함함 */}
        {currentUser && (
          <HStack gap={10}>
            <CurrentUserText />
            <LogoutButton />
          </HStack>
        )}
      </HStack>
    </NavbarContainer>
    <div className="CameraScanContainer">
      <div className="WebCamContainer">
        <p className="CameraScanTextContainer">
          물건을 화면의 <span className="highlight">중앙에</span> 맞춰서
          보여주세요 !
        </p>
        {/* ✅ 웹캠 화면 출력 */}
        <video
          ref={videoRef}

```

```
    autoPlay
    playsInline
    muted
    className="CameraFeed"
  ></video>

  { /* ✅ 객체 감지 캔버스 */}
  <canvas
    ref={canvasRef}
    className={`ObjectDetectionCanvas ${isDetecting ? 'active' : ''}`}
  ></canvas>
</div>
</div>
</div>
);
}
```