


포팅 메뉴얼

☰ 태그	 자료함
🕒 생성 일시	@2025년 5월 21일 오전 10:38

1. 개발환경

- [1.1 Frontend](#)
- [1.2 Backend](#)
- [1.3 AI](#)
- [1.4 Server](#)
- [1.5 Database](#)
- [1.6 UI/UX](#)
- [1.7 Mobile](#)
- [1.8 IDE](#)
- [1.9 형상 / 이슈 관리](#)

2. 인프라 세팅

- [2.1 서버 세팅 공통](#)
 - [2.1.0 서버 시간대 통일 / 깃 정보 등록](#)
 - [2.1.1 Zulu 17 설치](#)
 - [2.1.2 NGINX 설치](#)
 - [2.1.3 SSL 인증서 등록 \(Jenkins 제외\)](#)
 - [2.1.4 Docker 설치](#)
- [2.2 Jenkins 서버 세팅](#)
 - [2.2.1 Jenkins 설치](#)
 - [2.2.2 NGINX 설정](#)
 - [2.2.3 젠킨스 서버 방화벽 설정](#)
- [2.3 백엔드 배포 서버 세팅](#)
 - [2.3.2 Redis 설치 & 추가](#)
 - [2.3.3 MySQL 설치 / 설정](#)
 - [2.3.6 BE 배포 NGINX 세팅](#)
 - [2.3.7 FE 배포 NGINX 세팅](#)
 - [2.3.8 도메인 포워딩](#)
 - [2.3.9 방화벽 설정](#)
- [2.4 GPU서버 세팅](#)
 - [2.4.1 nvidia driver 설치](#)
 - [2.4.1 CUDA 설치](#)
 - [2.4.2 CuDNN 설치](#)
 - [2.4.3 CUDA 등록](#)
 - [2.4.3 CUDA 등록](#)

2.4.4 도커 Nvidia container toolkit 설치

2.4.5 python 설치

2.4.6 WireGuard 설치

2.4.7 WireGaurd 세팅

2.4.8 WireGuard 적용

2.4.9 Qdrant 설치

2.4.10 GPU서버 NGINX 세팅

2.3.9 방화벽 설정

3. Jenkins Pipeline

BE/FE 자동 배포

4. Front 세팅

4.1 키오스크 시작 세팅

1 vite + react + ts 설치

2 tailwindcss + styled-components + twin.macro 설치

파일 수정

test 코드

3 prettier, eslint 설치

1. Prettier와 ESLint 관련 플러그인을 설치

2. **eslint.config.js** 수정

2. **.prettierrignore** 파일 생성

3. **.prettierrc** 파일 생성

4. VS code extension 에서 아래 설치

5. VS Code 에서 ctrl + shift + p → settings.json 에 추가

6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음

4 zustand 설치

5 라우터 설치

6 axios 설치

7 Shadcn 설치 (UI 라이브러리)

8 절대 경로 설정

1. vite.config.ts

2. tsconfig.json

3. tsconfig.app.json

★전체창 모드 (키오스크)

1. 키오스크 모드(주소창 없이 전체화면)

2. 키오스크 모드 설정하는 방법

3. 주소창은 제거 되었으나, 탭이 뜰

카메라 + 얼굴 인식

4.2 앱 시작 세팅

*npm 패키지 이름 규칙

1 vite + react + ts 설치

2 pwa 설치

3 tailwindcss + styled-components + twin.macro 설치

파일 수정

test 코드

4 zustand 설치

5 TanStack Query 설치

6 prettier, eslint 설치

1. Prettier와 ESLint 관련 플러그인을 설치

2. **eslint.config.js** 수정

2. **.prettierrc** 파일 생성

3. **.prettierrc** 파일 생성

4. VS code extension 에서 아래 설치

5. VS Code 에서 ctrl + shift + p → settings.json 에 추가

6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음

7 ngrok 설치 - 로컬에서 http → https 변환용 (배포 전)

다운받은 파일의 압축을 해제하고 ngrok.exe를 **관리자 권한으로 실행**

1. Ngrok 계정 생성

2. 인증 토큰 확인

3. 로컬 환경에 Ngrok 인증 등록

4. Ngrok 다시 실행

*기존 인증 토큰 재설정

5. **vite.config.js** 수정

8 실행

9 절대 경로 설정

1. vite.config.ts

2. tsconfig.json

3. tsconfig.app.json

10 라우터 설치

1 1 Shadcn 설치 (UI 라이브러리)

1 2 axios 설치

1 3 react cookie 설치

4.3. env

4. 백엔드 설정

5. AI 세팅

5.1. 키오스크 세팅

Insightface 환경 세팅

1. Windows의 경우 Visual Studio 2019 dev tools 설치해야 c 컴파일러 사용 가능

2. Qdrant 설치

3. CUDA & CuDNN 설치

4. Python 환경 설치

1. 개발환경

1.1 Frontend

- React 18.3.1
- zustand 5.0.3
- TailWind CSS 3.4.17
- TypeScript 5.7.2
- Vite 6.2.0
- 라이브러리
 - twin.macro(Tailwind CSS v3 + @emotion)
 - react router dom
 - react-icons, iconoir-react
 - eslint
 - prettier
 - postcss

1.2 Backend

- Java
 - Azul Zulu 17.0.14
 - Spring Boot 3.4.2
 - JWT
 - Gradle
- Python
 - Python 3.10.11
 - Fast API
 - Pyglet

1.3 AI

- Insightface
- Mediapipe
- Arcface
- UTKFace Dataset
- CUDA 12.1.1
- CuDNN 9.1.1

1.4 Server

Ubuntu 22.04

Docker

Let'sEncrypt (SSL)

1.5 Database

- MySQL 8.0.41
- Redis 7.4.2
- Qdrant 1.14.0 * 4
- AWS S3

1.6 UI/UX

- Figma, Canva

1.7 Mobile

- PWA

1.8 IDE

- VS Code
- IntelliJ IDEA

1.9 형상 / 이슈 관리

- GitLab

- Jira

2. 인프라 세팅

2.1 서버 세팅 공통

2.1.0 서버 시간대 통일 / 깃 정보 등록

```
# 서버 시간대 확인
timedatectl

# 서울이 아닐 경우
sudo timedatectl set-timezone Asia/Seoul

git config --global user.name ""
git config --global user.email ""
```

2.1.1 Zulu 17 설치

```
sudo apt install gnupg ca-certificates curl
# 키 등록
curl -s https://repos.azul.com/azul-repo.key | sudo gpg --dearmor -o /usr/
share/keyrings/azul.gpg
echo "deb [signed-by=/usr/share/keyrings/azul.gpg] https://repos.azul.co
m/zulu/deb stable main" | sudo tee /etc/apt/sources.list.d/zulu.list

sudo apt-get update
sudo apt-get install -y zulu17-jdk

# 설치된 자바 폴더 위치 확인 (zulu 단독 설치 시)
sudo update-alternatives --config java
# 나온 경로 복사하여 JAVA_HOME 환경변수 설정해주기
# /usr/lib/jvm/zulu17/bin/java
```

```
sudo nano /etc/environment
PATH=":/usr/lib/jvm/zulu17/bin" # 기존에 추가
JAVA_HOME="/usr/lib/jvm/zulu17"
# Ctrl + O , Ctrl + X 로 저장 후 나가기

# 변경사항 적용
source /etc/environment
# 적용 확인
echo $JAVA_HOME
```

2.1.2 NGINX 설치

```
sudo apt-get update && sudo apt-get install nginx -y

sudo systemctl enable nginx
sudo systemctl start nginx
```

2.1.3 SSL 인증서 등록 (Jenkins 제외)

```
sudo apt-get update
sudo apt-get install -y certbot python3-certbot-nginx

sudo systemctl stop nginx

# 적용할 도메인 주소와 이메일 입력해주기
sudo certbot certonly --nginx -d {도메인주소}

# 생성 후 nginx에 키 등록
sudo nano /etc/nginx/sites-available/default

server {
    listen 80 default_server;
    server_name _;

    location / {
        return 200 '서비스 준비 중입니다.';
    }
}
```

```

    add_header Content-Type text/plain;
}
}

server {
    if ($host = {도메인 주소}) {
        return 301 https://$host$request_uri;
    }
    listen 80;
    server_name {도메인 주소};
    return 404;
}
server {
    listen 443 ssl;
    server_name {도메인 주소};

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 65;

    ssl_certificate /etc/letsencrypt/live/{도메인주소}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/{도메인주소}/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
    ssl_prefer_server_ciphers on;

    이후 세팅...
...

```

2.1.4 Docker 설치

```

sudo apt-get update
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/
keyrings/docker.asc

```



```

sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/doc
ker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-pl
ugin docker-compose-plugin

# 일반 사용자 권한 설정
sudo usermod -aG docker $USER

# 등록 && 시작
sudo systemctl enable docker
sudo systemctl start docker

```

2.2 Jenkins 서버 세팅

2.2.1 Jenkins 설치

```

# jenkins 설치
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins -y

# 권한 재확인
sudo chown -R jenkins:jenkins /var/lib/jenkins
sudo chown -R jenkins:jenkins /var/cache/jenkins

```

```
sudo chown -R jenkins:jenkins /var/log/jenkins
```

```
# 서비스 등록 및 실행
```

```
sudo systemctl enable jenkins
```

```
sudo systemctl start jenkins
```

2.2.2 NGINX 설정

```
sudo nano /etc/nginx/sites-available/jenkins
```

```
sudo ln -s /etc/nginx/sites-available/jenkins /etc/nginx/sites-enabled/
```

```
sudo rm /etc/nginx/sites-enabled/default
```

```
sudo systemctl restart nginx
```

앞단에 프록시 서버가 하나 있어서 거기에 맞게 NGINX 설정 - 80 포트로만 받게 설정됨

```
# /etc/nginx/sites-available/jenkins
```

```
upstream jenkins {
```

```
    server 127.0.0.1:8080;
```

```
    keepalive 32; # 커넥션 유지
```

```
}
```

```
server {
```

```
    listen 80;
```

```
    server_name localhost;
```

```
    ignore_invalid_headers off;
```

```
    location / {
```

```
        proxy_pass http://jenkins;
```

```
        # 프록시 헤더 설정
```

```
        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 https; # 앞단의 HTTPS를 위해
```

```
        proxy_set_header X-Forwarded-Host $host;
```

```
        proxy_set_header X-Forwarded-Port 443; # HTTPS 포트
```

```
# 리다이렉션 설정
proxy_redirect http://jenkins https://$host;

proxy_http_version 1.1;
proxy_request_buffering off;
proxy_buffering off;

# 타임아웃 설정
proxy_connect_timeout 150;
proxy_send_timeout 100;
proxy_read_timeout 100;
}
}
```

2.2.3 젠킨스 서버 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw enable
```

2.3 백엔드 배포 서버 세팅

2.3.2 Redis 설치 & 추가

```
sudo apt-get update
sudo apt-get install -y redis-server

# 서비스 등록 및 실행
sudo systemctl start redis
sudo systemctl enable redis

# 접속
redis-cli
```

```

# 키 조회
keys *
# get "조회할 키 이름"

# 서버 하나 더 열기
sudo cp /etc/redis/redis.conf /etc/redis/redis2.conf
sudo nano /etc/redis/redis2.conf
# 수정
port 6380
pidfile /run/redis/redis-server2.pid
logfile /var/log/redis/redis-server2.log
dbfilename dump2.rdb
dir /var/lib/redis2

cp /lib/systemd/system/redis-server.service /lib/systemd/system/redis-server2.service
sudo nano /lib/systemd/system/redis-server2.service
# 수정
[Unit]
Description=Advanced key-value store second
ExecStart=/usr/bin/redis-server /etc/redis/redis2.conf
ReadWriteDirectories=-/var/lib/redis2

[Install]
WantedBy=multi-user.target
Alias=redis2.service

sudo systemctl daemon-reload
sudo systemctl enable redis-server2.service
sudo systemctl start redis-server2.service
sudo systemctl status redis-server2.service

# 접속
redis-cli -p 6380

```

2.3.3 MySQL 설치 / 설정

```

# 설치
sudo apt-get update && sudo apt-get install -y mysql-server
# 포트 허용
sudo ufw allow mysql
# 시작 등록 및 시작
sudo systemctl enable mysql
sudo systemctl start mysql

# mysql 접속
sudo mysql

# 루트유저 비밀번호 생성
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '비밀번호';
# 권한 갱신
FLUSH PRIVILEGES;

# 유저 생성
CREATE USER '유저명'@'localhost' IDENTIFIED BY '유저비번';

# 데이터베이스 생성 후
CREATE DATABASE 새로운데이터베이스;
SHOW DATABASES;

# 데이터베이스 권한 주기
GRANT ALL PRIVILEGES ON 생성한DB.* TO '권한 줄 유저명'@'localhost';
FLUSH PRIVILEGES; # 권한 갱신

EXIT;

# 이후엔 mysql -u 유저명 -p 로 sql 접속

```

2.3.6 BE 배포 NGINX 세팅

```

# api
upstream api {
    keepalive 32;

```

```

server 127.0.0.1:8080;
}
map $http_upgrade $connection_upgrade {
    default upgrade;
    "" close;
}
server {
    if ($host = <api 배포 도메인>) {
        return 301 https://$host$request_uri;
    }
    listen 80;
    server_name <api 배포 도메인>;
    return 404;
}
server {
    listen 443 ssl;
    server_name <api 배포 도메인>;

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 120;

    proxy_read_timeout 300;
    proxy_connect_timeout 300;
    proxy_send_timeout 300;

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

    ssl_certificate /etc/letsencrypt/live/<api 배포 도메인>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<api 배포 도메인>/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
    ssl_prefer_server_ciphers on;

```

```

location / {
    proxy_pass http://api;
    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_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}

location = /50x.html {
    root /usr/share/nginx/html;
}
}

```

2.3.7 FE 배포 NGINX 세팅

```

# user-app
server {
    if ($host = <앱 배포 도메인>) {
        return 301 https://$host$request_uri;
    }
    listen 80;
    server_name <앱 배포 도메인>;
    return 404;
}

server {
    listen 443 ssl;
    server_name <앱 배포 도메인>;

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 120;

    ssl_certificate /etc/letsencrypt/live/<앱 배포 도메인>/fullchain.pem;

```

```

ssl_certificate_key /etc/letsencrypt/live/<앱 배포 도메인>/privkey.pem;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
ssl_prefer_server_ciphers on;

sub_filter_once on;
sub_filter_types text/html;

sub_filter '<head>' '<head>
<meta name="description" content="주문, 결제 페이스로 한번에">
<meta name="keywords" content="키오스크, 추천, 얼굴인식">
<meta property="og:title" content="커피페이스 COFFACE">
<meta property="og:description" content="주문, 결제 페이스로 한번에">
<meta property="og:image" content="https://s3.ap-northeast-2.amazon
aws.com/order.me/banner.png">
<meta property="og:image:width" content="1200">
<meta property="og:image:height" content="630">';

location / {
    root /var/www/userapp;
    index index.html;
    try_files $uri /index.html;
}

location = /50x.html {
    root /usr/share/nginx/html;
}
}

# kiosk
server {
    if ($host = <키오스크 배포 도메인>) {
        return 301 https://$host$request_uri;
    }

    listen 80;
    server_name <키오스크 배포 도메인>;

```



```

    return 404;
}
server {
    listen 443 ssl;
    server_name <키오스크 배포 도메인>;

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 65;

    ssl_certificate /etc/letsencrypt/live/<키오스크 배포 도메인>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<키오스크 배포 도메인>/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
    ssl_prefer_server_ciphers on;

    sub_filter_once on;
    sub_filter_types text/html;

    sub_filter '<head>' '<head>
    <meta name="description" content="주문, 결제 페이스로 한번에">
    <meta name="keywords" content="키오스크, 추천, 얼굴인식">
    <meta property="og:title" content="커피페이스 COFFACE">
    <meta property="og:description" content="주문, 결제 페이스로 한번에">
    <meta property="og:image" content="https://s3.ap-northeast-2.amazonaws.com/order.me/banner.png">
    <meta property="og:image:width" content="1200">
    <meta property="og:image:height" content="630">';

    location / {
        root /var/www/kiosk;
        index index.html;
        try_files $uri /index.html;
    }

    location = /50x.html {

```

```
    root /usr/share/nginx/html;
}
}
```

2.3.8 도메인 포워딩

```
# ssafy domain
server {
    if ($host = k12e202.p.ssafy.io) {
        return 301 https://<app 배포 도메인>$request_uri;
    }
    listen 80;
    listen 443;

    ssl_certificate /etc/letsencrypt/live/k12e202.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k12e202.p.ssafy.io/privkey.pem;

    server_name k12e202.p.ssafy.io;
    return 404;
}
```

2.3.9 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw allow 8080 # spring boot
sudo ufw allow 51820 # WireGuard
sudo ufw enable
```

2.4 GPU서버 세팅

2.4.1 nvidia driver 설치

```
sudo apt install git build-essential dkms net-tools htop curl wget tmux bmon vim nano -y
sudo apt-get install -y ubuntu-drivers-common
sudo ubuntu-drivers autoinstall
```

2.4.1 CUDA 설치

```
wget https://developer.download.nvidia.com/compute/cuda/12.1.1/local_installers/cuda_12.1.1_530.30.02_linux.run
sudo sh cuda_12.1.1_530.30.02_linux.run
```

2.4.2 CuDNN 설치

```
# 설치
wget https://developer.download.nvidia.com/compute/cudnn/9.1.1/local_installers/cudnn-local-repo-ubuntu2204-9.1.1_1.0-1_amd64.deb
sudo dpkg -i cudnn-local-repo-ubuntu2204-9.1.1_1.0-1_amd64.deb
sudo cp /var/cudnn-local-repo-ubuntu2204-9.1.1/cudnn-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cudnn
```

```
# 적용, 권한 주기
sudo cp -a /usr/include/cudnn*.h /usr/local/cuda-12.1/include/
sudo cp -a /usr/lib/x86_64-linux-gnu/libcudnn* /usr/local/cuda-12.1/lib64/
sudo chmod a+r /usr/local/cuda-12.1/include/cudnn*.h /usr/local/cuda-12.1/lib64/libcudnn*
```

2.4.3 CUDA 등록

```
nano ~/.bashrc # .zshrc

# 쿠다 환경변수 기본값
export CUDA_HOME=/usr/local/cuda-12.1
export PATH=/usr/local/cuda-12.1/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-12.1/lib64:$LD_LIBRARY_PATH
```

```
export CPATH=/usr/local/cuda-12.1/include:$CPATH
```

```
source ~/.bashrc
```

2.4.3 CUDA 등록

```
nano ~/.bashrc # .zshrc
```

```
# 쿠다 환경변수 기본값
```

```
export CUDA_HOME=/usr/local/cuda-12.1
```

```
export PATH=/usr/local/cuda-12.1/bin:$PATH
```

```
export LD_LIBRARY_PATH=/usr/local/cuda-12.1/lib64:$LD_LIBRARY_PATH
```

```
export CPATH=/usr/local/cuda-12.1/include:$CPATH
```

```
source ~/.bashrc
```

2.4.4 도커 Nvidia container toolkit 설치

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --d  
earmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \  
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-  
container-toolkit.list | \  
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-containe  
r-toolkit-keyring.gpg] https://#g' | \  
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

```
sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
```

```
# 검증
```

```
nvidia-ctk --version
```

```
> NVIDIA Container Toolkit CLI version 1.17.6
```

2.4.5 python 설치

```
sudo apt-get install python3 python3-pip python3-dev -y # 3.10.12 설치됨
```

```
# python path 설정
```

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.1
```

```
0 1
# pip path 설정
sudo nano /etc/environment

# PATH 맨 앞에
PATH="<유저홈 절대경로>/local/bin: ~~~ "

source /etc/environment
```

2.4.6 WireGuard 설치

```
# 사용할 모든 서버에 설치
sudo apt-get update && sudo apt-get install wireguard -y

# 각 서버에서 키 발급
wg genkey | tee privatekey | wg pubkey > publickey
```

2.4.7 WireGaurd 세팅

```
sudo nano /etc/wireguard/wg0.conf

# /etc/wireguard/wg0.conf
[Interface]
PrivateKey = <1번 노드에서 발급한 개인키>
Address = 10.0.0.1/24
ListenPort = 51820

[Peer]
PublicKey = <2번 노드에서 발급한 공개키>
AllowedIPs = 10.0.0.2/32
Endpoint = <2번 노드 (공인) IP 또는 도메인>:51820
PersistentKeepalive = 25

[Peer]
PublicKey = <3번 노드에서 발급한 공개키>
AllowedIPs = 10.0.0.3/32
Endpoint = <3번 노드 (공인) IP 또는 도메인>:51820
```

```
PersistentKeepalive = 25
```

```
[Peer]
```

```
PublicKey = <4번 노드에서 발급한 공개키>
```

```
AllowedIPs = 10.0.0.4/32
```

```
Endpoint = <4번 노드 (공인) IP 또는 도메인>:51820
```

```
PersistentKeepalive = 25
```

2.4.8 WireGuard 적용

```
# 모든 서버에서 WireGuard 시작
```

```
sudo systemctl enable wg-quick@wg0
```

```
sudo systemctl start wg-quick@wg0
```

```
# 연결 테스트
```

```
ping 10.0.0.2
```

```
# qdrant 암호화 키 발급
```

```
openssl rand -hex 32
```

2.4.9 Qdrant 설치

물리적으로 분리된 4개의 서버에 각각 세팅 후 WireGuard를 통해 연결

Leader 노드 (노드 1)

```
docker run -d --name qdrant \
```

```
-p 6333:6333 -p 6334:6334 -p 6335:6335 \ # 6333(REST API), 6334(gRPC), 6335(P2P 통신)
```

```
-v $(pwd)/qdrant_storage:/qdrant/storage \ # 데이터를 저장할 로컬 디렉토리, 도커에 마운트
```

```
-e QDRANT_CLUSTER_ENABLED=true \ # 클러스터 모드 활성화
```

```
-e QDRANT_CLUSTER_DISCOVERY_METHOD=static \ # 정적 피어 목록을 사용
```

```
-e QDRANT_CLUSTER_P2P_PORT=6335 \ # P2P 통신에 사용할 포트 번호
```

```
-e QDRANT_CLUSTER_STATIC_PEERS=10.0.0.1:6335 \ # 클러스터 피어 목록
```

```
-e QDRANT_SERVICE_API_KEY="API 키" \ # REST API 요청 인증용
```

```
qdrant/qdrant \ # 사용할 도커 이미지 - 최신 1.14.0
```

```
./qdrant --uri http://10.0.0.1:6335 # 이 노드의 URI 주소 지정 (클러스터 내에서 이  
노드를 식별하는 주소)
```

Member 노드 (노드 2)

```
docker run -d --name qdrant \ # 같은 서버 인스턴스에서 여러 개의 도커로 할 경  
우 이름 다르게.  
-p 6333:6333 -p 6334:6334 -p 6335:6335 \ # 6333(REST API), 6334(gRP  
C), 6335(P2P 통신)  
-v $(pwd)/qdrant_storage:/qdrant/storage \  
-e QDRANT__CLUSTER__ENABLED=true \  
-e QDRANT__CLUSTER__DISCOVERY_METHOD=static \  
-e QDRANT__CLUSTER__P2P__PORT=6335 \  
-e QDRANT__CLUSTER__STATIC__PEERS=10.0.0.1:6335,10.0.0.2:6335 \ # 클  
러스터 피어 목록 갱신  
-e QDRANT__SERVICE__API_KEY="API 키" \  
qdrant/qdrant \  
./qdrant --bootstrap http://10.0.0.1:6335 --uri http://10.0.0.2:6335 # 첫 번째  
노드에 연결(bootstrap)하고 자신의 URI 지정
```

Member 노드 (노드 3)

```
docker run -d --name qdrant \ # 같은 서버 인스턴스에서 여러 개의 도커로 할 경  
우 이름 다르게.  
-p 6333:6333 -p 6334:6334 -p 6335:6335 \  
-v $(pwd)/qdrant_storage:/qdrant/storage \  
-e QDRANT__CLUSTER__ENABLED=true \  
-e QDRANT__CLUSTER__DISCOVERY_METHOD=static \  
-e QDRANT__CLUSTER__P2P__PORT=6335 \  
-e QDRANT__CLUSTER__STATIC__PEERS=10.0.0.1:6335,10.0.0.2:6335,10.0.  
0.3:6335 \ # 클러스터 피어 목록 갱신  
-e QDRANT__SERVICE__API_KEY="API 키" \  
qdrant/qdrant \  
./qdrant --bootstrap http://10.0.0.1:6335 --uri http://10.0.0.3:6335 # 첫 번째  
노드에 연결(bootstrap)하고 자신의 URI 지정
```

Member 노드 (노드 4)

```
docker run -d --name qdrant \  
-p 6333:6333 -p 6334:6334 -p 6335:6335 \  
-v $(pwd)/qdrant_storage:/qdrant/storage \  
-e QDRANT_CLUSTER_ENABLED=true \  
-e QDRANT_CLUSTER_DISCOVERY_METHOD=static \  
-e QDRANT_CLUSTER_P2P_PORT=6335 \  
-e QDRANT_CLUSTER_STATIC_PEERS=10.0.0.1:6335,10.0.0.2:6335,10.0.0.3:6335,10.0.0.4:6335 \  
-e QDRANT_SERVICE_API_KEY="API 키" \  
qdrant/qdrant \  
./qdrant --bootstrap http://10.0.0.1:6335 --uri http://10.0.0.4:6335
```

설치 확인

```
curl -H "api-key: API 키" http://localhost:6333/cluster
```

```
# 응답  
{  
  "result":{  
    "status":"enabled",  
    "peer_id":2688715334810444,  
    "peers":{  
      "3338640081498217":{"uri":"http://10.0.0.4:6335/"},  
      "5094325737698394":{"uri":"http://10.0.0.2:6335/"},  
      "2688715334810444":{"uri":"http://10.0.0.1:6335/"},  
      "517720741566645":{"uri":"http://10.0.0.3:6335/"},  
    },  
    "raft_info":{  
      "term":2,  
      "commit":17,  
      "pending_operations":0,  
      "leader":2688715334810444,  
      "role":"Leader",  
      "is_voter":true  
    },  
  },  
}
```



```

"consensus_thread_status":{
  "consensus_thread_status":"working",
  "last_update":"2025-05-20T18:23:22.788794737Z"
},
"message_send_failures":{}
},
"status":"ok",
"time":5.066e-6
}

```

2.4.10 GPU서버 NGINX 세팅

```

upstream face {
    keepalive 32;
    server 127.0.0.1:8000;
}

map $http_upgrade $connection_upgrade {
    default upgrade;
    "" close;
}

server {
    if ($host = <얼굴 인식 api 배포 도메인>) {
        return 301 https://$host$request_uri;
    }
    listen 80;
    server_name <얼굴 인식 api 배포 도메인>;
    return 404;
}

server {
    listen 443 ssl;
    server_name <얼굴 인식 api 배포 도메인>;

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 65;
}

```

```

ssl_certificate /etc/letsencrypt/live/<얼굴 인식 api 배포 도메인>/fullchain.p
em;
ssl_certificate_key /etc/letsencrypt/live/<얼굴 인식 api 배포 도메인>/privke
y.pem;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
ssl_prefer_server_ciphers on;

location / {
    proxy_pass http://face;
    proxy_redirect default;
    proxy_http_version 1.1;

    proxy_set_header Connection $connection_upgrade;
    proxy_set_header Upgrade $http_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;
}

location = /50x.html {
    root /usr/share/nginx/html;
}
}

```

2.3.9 방화벽 설정

```

sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw allow 8000 # Fast API
sudo ufw allow 51820 # WireGuard
sudo ufw enable

```

3. Jenkins Pipeline

BE/FE 자동 배포

```
import groovy.json.JsonOutput

pipeline {
    agent any
    tools {
        gradle 'Default Gradle'
        jdk 'Zulu17'
        git 'Default Git'
    }
    environment {
        // 프로젝트 구조 변수
        BACKEND_DIR = "BE/orderme"
        KIOSK_DIR = "FE/kiosk"
        APP_DIR = "FE/user-app"
        STAGE_NAME = ""

        // Docker 설정
        DOCKER_USER = 'team2room'
        IMAGE_NAME = 'cofface'
        GIT_COMMIT_SHORT = ""
        DOCKER_TAG = ""

        // 깃 정보
        COMMIT_MSG = ""
        COMMIT_HASH = ""
        AUTHOR = ""
        BRANCH_NAME = 'develop'
        EXCLUDE_BRANCH = 'master,documents,feature'
        ERROR_MSG = "false"

        // 서버 정보
```

```

PROD_SERVER = 'api.cofface.store'
JIRA_BASE_URL = 'https://ssafy.atlassian.net'
GITLAB_BASE_URL = 'https://lab.ssafy.com/s12-final/S12P31E202'
}
stages {
    // 공통으로 실행
    stage('Checkout and Update') {
        steps {
            script {
                echo "Branch: ${env.GIT_BRANCH}"
                echo "Commit: ${env.GIT_COMMIT}"
                BRANCH_NAME = (env.GIT_BRANCH ?: "develop").replaceFirst
("refs/heads/", "")
                STAGE_NAME = "Checkout and Update (1/7)"
                def notAllowedBranches = EXCLUDE_BRANCH.split(',')
                def repoExists = fileExists('.git')
                if (repoExists) {
                    echo "Repository exists. Updating..."
                    try {
                        checkout([
                            $class: 'GitSCM',
                            branches: [[name: '*/develop']],
                            userRemoteConfigs: [[
                                url: "${GITLAB_BASE_URL}.git",
                                credentialsId: 'gitlab-credentials'
                            ]],
                            extensions: [
                                [$class: 'CleanBeforeCheckout'],
                                [$class: 'PruneStaleBranch']
                            ]
                        ])
                        withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials')]) {
                            sh """
                                git fetch --all --prune
                                git checkout -B ${BRANCH_NAME} origin/${BRANCH
_NAME} --force
                                git pull origin ${BRANCH_NAME}

```

```

"""

    if (notAllowedBranches.contains(BRANCH_NAME)) {
        BRANCH_NAME='develop'
        echo "허용되지 않은 브랜치이므로 develop으로 빌드합니
다."

        sh "git checkout -B develop origin/develop --force"
    }

    GIT_COMMIT_SHORT = sh(script: "git rev-parse --short
HEAD", returnStdout: true).trim()
    DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"

    echo DOCKER_TAG
}
} catch (Exception e) {
    echo "Error during update: ${e.message}"
    ERROR_MSG = "Failed to update repository"
    error ERROR_MSG
}
} else {
    echo "Repository does not exist. Cloning..."
    try {
        withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials')]) {
            sh "git clone ${GITLAB_BASE_URL}.git ."
            sh "git checkout ${BRANCH_NAME}"

            if (notAllowedBranches.contains(BRANCH_NAME)) {
                BRANCH_NAME='develop'
                echo "허용되지 않은 브랜치이므로 develop으로 빌드합니
다."

                sh "git checkout -B develop origin/develop --force"
            }

            GIT_COMMIT_SHORT = sh(script: "git rev-parse --short
HEAD", returnStdout: true).trim()
            DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"

```

```

T_SHORT}"

        echo DOCKER_TAG
    }
} catch (Exception e) {
    echo "Error during clone: ${e.message}"
    ERROR_MSG = "Failed to clone repository"
    error ERROR_MSG
}
}
AUTHOR = sh(script: "git log -1 --pretty=format:%an", returnStdout: true).trim()
echo AUTHOR
COMMIT_MSG = sh(script: 'git log -1 --pretty=%B | tr "\\n" " "', returnStdout: true).trim()
echo COMMIT_MSG
COMMIT_HASH = sh(script: "git log -1 --pretty=format:%H", returnStdout: true).trim()
}
}
}
// be 세팅
stage('BE Inject Config') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[be]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/be')))
        }
    }
    steps {
        script {
            STAGE_NAME = "BE Inject Config (2/7)"
        }
        withCredentials([
            file(credentialsId: 'orderme-config', variable: 'CONFIG_FILE')
        ]) {
            sh """
                cp \\$CONFIG_FILE ${BACKEND_DIR}/src/main/resources/ap

```

```

plication.yml
    """
    }
    withCredentials([
        file(credentialsId: 'firebase-file', variable: 'JSON_FILE')
    ]) {
        sh """
            mkdir -p ${BACKEND_DIR}/src/main/resources/firebase
            cp \${JSON_FILE} ${BACKEND_DIR}/src/main/resources/fireb
ase/orderme-9ec2c-firebase-adminsdk-fbsvc-badfbf41fa.json
        """
    }
}
// fe 세팅
stage('FE Inject Config') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMM
IT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase
().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE Inject Config (2/7)"
        }
        withCredentials([
            file(credentialsId: 'app-config', variable: 'CONFIG_FILE')
        ]) {
            sh """
                rm ${APP_DIR}/.env || true
                rm ${KIOSK_DIR}/.env || true
                cp \${CONFIG_FILE} ${APP_DIR}/.env
                cp \${CONFIG_FILE} ${KIOSK_DIR}/.env
            """
        }
    }
}

```

```

}
// be 빌드
stage('BE Build') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[be]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/be')))
        }
    }
    steps {
        script {
            STAGE_NAME = "BE Build (3/7)"
        }
        dir(BACKEND_DIR) {
            sh "chmod +x gradlew"
            script{
                try {
                    sh "./gradlew clean build -x test"
                } catch(Exception e) {
                    ERROR_MSG = e.getMessage()
                    error ERROR_MSG
                }
            }
        }
    }
    post {
        failure {
            cleanWs()
            script {
                ERROR_MSG += "\nBuild failed"
                error ERROR_MSG
            }
        }
    }
}
// 키오스크 빌드
stage('FE KIOSK build') {

```



```

    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMM
IT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase
().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE KIOSK Build (3/7)"
        }
        dir(KIOSK_DIR) {
            script{
                try {
                    echo "build"
                    sh """"#!/bin/bash
                        source ~/.bashrc
                        nvm use --lts
                        npm install
                        npm run build
                        tar -czvf dist.tar.gz dist/
                    """"
                } catch(Exception e) {
                    ERROR_MSG = e.getMessage()
                    error ERROR_MSG
                }
            }
        }
    }
}
// be 도커 묶기
stage('Docker Build') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[be]') || (COM
MIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCas
e().contains('feature/be')))
        }
    }
}

```

```

    }
    steps {
        script {
            STAGE_NAME = "Docker Build (4/7)"
            IMAGE_NAME += (BRANCH_NAME == 'develop') ? "-dev" : "-sub"

            def jarFile = sh(script: "ls ${BACKEND_DIR}/build/libs/*SNAPSHOT.jar", returnStdout: true).trim()
            try {
                // docker.build("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}-test", "--no-cache --build-arg JAR_FILE=${jarFile} .")
                docker.build("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}", "-f ${BACKEND_DIR}/Dockerfile --no-cache --build-arg JAR_FILE=${jarFile} .")
                // sh "docker save ${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}-test | gzip > image.tar.gz"
                docker.withRegistry('https://index.docker.io/v1/', 'keywi-docker') {
                    docker.image("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}").push()
                }
            } catch (Exception e) {
                ERROR_MSG = e.getMessage()
                error ERROR_MSG
            }
        }
    }
    post {
        failure {
            cleanWs()
            script {
                ERROR_MSG += "\nDocker Build failed"
                error ERROR_MSG
            }
        }
    }
}
// 키오스크 배포

```

```

stage('FE KIOSK Deploy') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE KIOSK Deploy (4/7)"
        }
        dir(KIOSK_DIR) {
            script{
                try {
                    echo "deploy"
                    sshagent(['orderme']) {
                        sh """
                            set -e
                            rsync -av --progress -e 'ssh -o StrictHostKeyChecking=no' -W dist.tar.gz ubuntu@${PROD_SERVER}:/tmp/

                            sleep 1

                            local_size=$(stat -c%s dist.tar.gz)
                            remote_size=$(ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "stat -c%s /tmp/dist.tar.gz")

                            if [ "$local_size" -ne "$remote_size" ]; then
                                echo "ERROR: File size mismatch (Local: $local_size, Remote: $remote_size)"
                                exit 1
                            fi

                            sleep 1

                            ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "

```

```

        sudo rm -rf /var/www/kiosk
        sudo mkdir -p /var/www/kiosk
        cd /tmp
        tar -xzvf dist.tar.gz
        sudo mv dist/* /var/www/kiosk/
        rm -rf dist dist.tar.gz
        sudo systemctl restart nginx
    "

    rm -rf dist dist.tar.gz
    """
    }
    } catch(Exception e) {
        ERROR_MSG = e.getMessage()
        error ERROR_MSG
    }
    }
    }
    }
    }
    // be 배포
    stage('Deploy to Prod') {
        when {
            expression {
                return (COMMIT_MSG.toLowerCase().contains('[be]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/be')))
            }
        }
        steps {
            script {
                STAGE_NAME = "Deploy to Prod (5/7)"
                def currentBranch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).trim()
                sshagent(['ec2-ssafy']) {
                    sh """
                        ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "

```

```

        docker pull ${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}

        docker stop ${IMAGE_NAME} || true
        docker rm ${IMAGE_NAME} || true
        docker run -d --network host --name ${IMAGE_NAME}
        ${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}
    "
    ""
}
echo "Success Production deployment."
}
}
post {
    failure {
        script {
            ERROR_MSG = "Production deployment failed"
            error ERROR_MSG
        }
    }
}
}
// 앱 빌드
stage('FE APP build') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE APP Build (5/7)"
        }
        dir(APP_DIR) {
            script{
                try {
                    echo "build"

```

```

        sh ""#!/bin/bash
        source ~/.bashrc
        nvm use --lts
        npm install
        npm run build
        tar -czvf dist.tar.gz dist/
        ""
    } catch (Exception e) {
        ERROR_MSG = e.getMessage()
        error ERROR_MSG
    }
}
}
}
}
// be 배포 테스트
stage('Prod Health Check') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[be]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/be')))
        }
    }
    steps {
        script {
            STAGE_NAME = "Prod Health Check (6/7)"
            def maxRetries = 5
            def timeout = 10
            def success = false

            for (int i = 0; i < maxRetries; i++) {
                sleep(timeout)
                try {
                    def response = httpRequest "https://${PROD_SERVER}/actor/health"

                    if (response.status == 200) {
                        success = true

```

```

        break
    }
} catch(e) {
    echo "Health check attempt ${i+1} failed"
}
}
if (!success) {
    ERROR_MSG = "Health check failed after ${maxRetries} attempts"
    error ERROR_MSG
}
}
}
// 앱 배포
stage('FE APP Deploy') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE APP Deploy (6/7)"
        }
        dir(APP_DIR) {
            script {
                try {
                    echo "deploy"
                    sshagent(['orderme']) {
                        sh """
                            set -e
                            rsync -av --progress -e 'ssh -o StrictHostKeyChecking=no' -W dist.tar.gz ubuntu@${PROD_SERVER}:/tmp/

                            sleep 1

```

```

        local_size=\$(stat -c%s dist.tar.gz)
        remote_size=\$(ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "stat -c%s /tmp/dist.tar.gz")

        if [ "\$local_size" -ne "\$remote_size" ]; then
            echo "ERROR: File size mismatch (Local: \$local_size, Remote: \$remote_size)"
            exit 1
        fi

        sleep 1

        ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "
            sudo rm -rf /var/www/userapp
            sudo mkdir -p /var/www/userapp
            cd /tmp
            tar -xzf dist.tar.gz
            sudo mv dist/* /var/www/userapp/
            rm -rf dist dist.tar.gz
            sudo systemctl restart nginx
        "

        rm -rf dist dist.tar.gz
    ""
}
} catch(Exception e) {
    ERROR_MSG = e.getMessage()
    error ERROR_MSG
}
}
}
}
}
// be 배포 완료
stage('BE Deploy Complete') {
    when {

```



```

        expression {
            return (COMMIT_MSG.toLowerCase().contains('[be]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/be')))
        }
    }
    steps {
        script {
            STAGE_NAME = "Deploy Complete (7/7)"
        }
    }
}
// fe 배포 완료
stage('FE Deploy Complete') {
    when {
        expression {
            return (COMMIT_MSG.toLowerCase().contains('[fe]') || (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/fe')))
        }
    }
    steps {
        script {
            STAGE_NAME = "FE Deploy Complete (7/7)"
        }
    }
}
// mm 발송
post {
    always {
        script {
            def issueKeyPattern = /\[#(S12P31E202-\d+)\]/
            def issueKey = (COMMIT_MSG =~ /S12P31E202-\d+/) ? (COMMIT_MSG =~ /S12P31E202-\d+/)[0] : null
            def cleanedMessage = issueKey ? COMMIT_MSG.replaceFirst(issueKeyPattern, '').trim() : COMMIT_MSG
            def jiraLink = issueKey ? "${JIRA_BASE_URL}/jira/software/c/proj

```

```

ects/S12P31E202/boards/8270?selectedIssue=${issueKey}" : ""

    def message = "${env.JOB_NAME} - #${env.BUILD_NUMBER}\n"
+ "- 결과: ${currentBuild.currentResult}\n" +
        "- 브랜치: ${BRANCH_NAME}\n- 커밋: " +
        (issueKey ? "[${issueKey}] " : "") +
        "[${cleanedMessage}](${GITLAB_BASE_URL}/-/commit/
${COMMIT_HASH}) (${GIT_COMMIT_SHORT}) [${AUTHOR}]\n" +
        "- 실행 시간: ${currentBuild.durationString}\n" +
        "- 최종 실행된 스테이지 : ${STAGE_NAME}\n" +
        ((ERROR_MSG!="false") ? "- ERROR :\n`${ERROR_MSG}`
\n" : "")

    if (issueKey) {
        try {
            def requestBody = [body: message]
            def response = httpRequest authentication: 'jira-credentials',
                contentType: 'APPLICATION_JSON',
                httpMode: 'POST',
                requestBody: groovy.json.JsonOutput.toJson(requestBod
y),
                url: "${JIRA_BASE_URL}/rest/api/2/issue/${issueKey}/com
ment"

            echo "JIRA comment added successfully. Status: ${respons
e.status}"
        } catch(e) {
            echo "JIRA 코멘트 추가 실패: ${e.message}"
        }
    }

    message += (currentBuild.currentResult == 'ABORTED' ? "- **사용
자 취소**\n" : "")

    message += "- 상세: " + (currentBuild.currentResult == 'SUCCE
S' ? ":jenkins7:" : (currentBuild.currentResult == 'ABORTED' ? ":jenkins_cut
e_flip:" : ":jenkins5:")) + " [Jenkins](${env.BUILD_URL})"

    message += jiraLink ? " | :jira: [Jira](${jiraLink}) " : (cleanedMessa
ge.contains('Merge') ? " | :jira6: [Jira](${JIRA_BASE_URL}/jira/software/c/p
rojects/S12P31E202/boards/8270)" : " | :jira3:")

    message += "\n\n`${env.BUILD_TIMESTAMP}`"

```

```

        mattermostSend color: currentBuild.currentResult == 'SUCCESS'
? 'good' : (currentBuild.currentResult == 'ABORTED' ? 'warning' : 'danger'),
message: message
    }
}
// 실패시 모두 삭제
failure {
    script {
        // def message = "${env.JOB_NAME} - #${env.BUILD_NUMBER}
Failed:\n" +
        // "- 파이프라인 실행 중 오류가 발생했습니다.\n" +
        // "- 최종 실행된 스테이지 : ${STAGE_NAME}\n\n" +
        // "`${env.BUILD_TIMESTAMP}`"

        // mattermostSend color: 'danger', message: message

        cleanWs(cleanWhenNotBuilt: false,
            deleteDirs: true,
            disableDeferredWipeout: true,
            notFailBuild: true)
    }
}
}
}
}

```

4. Front 세팅

4.1 키오스크 시작 세팅

1 vite + react + ts 설치

```
npm create vite@latest kiosk --template react-ts
```

```
cd kiosk
```

```
npm install
```

```
npm run dev
```



react버전 변경

확인 : `npm list react`

변경 : `npm install react@18 react-dom@18`

2 tailwindcss + styled-components + twin.macro 설치

```
# v3 버전 설치 방법
```

```
# Tailwind CSS v3.x 설치
```

```
npm i -D tailwindcss@3 postcss autoprefixer
```

```
npx tailwind init -p
```

```
# styled-components 설치 (TypeScript 지원 포함)
```

```
npm install styled-components
```

```
npm install -D @types/styled-components
```

```
# twin.macro 설치
```

```
npm install twin.macro
```

```
npm install vite-plugin-babel-macros
```

```
npm install -D @emotion/react @emotion/styled
```

```
npm install tailwindcss-animate
```

파일 수정

public 폴더 아래 → icons 폴더 생성 → 아래 파일들 넣기

favicomatic.zip

```
// vite.config.ts

import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import macrosPlugin from 'vite-plugin-babel-macros'

// https://vite.dev/config/
export default defineConfig({
  plugins: [
    react(),
    macrosPlugin(),
  ],
})
```

```
// tailwind.config.js 수정

/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.js,ts,jsx,tsx",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

```
// postcss.config.js 수정

export default {
  plugins: {
```

```
'postcss-import': {},
tailwindcss: {},
autoprefixer: {},
},
}
```

test 코드

```
/* index.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

/* 경고 표시 뜰 때 */

✓ 해결 방법 1: VSCode Tailwind CSS 확장 프로그램 설치

✓ 해결 방법 2: settings.json 수정

- VSCode에서 Ctrl + Shift + P (⌘ + Shift + P on Mac)

- "Preferences: Open Settings (JSON)" 검색해서 선택

```
{
  "css.lint.unknownAtRules": "ignore"
}
```

```
// App.tsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

import tw from "tailwindcss";
import styled from "styled-components";

// Tailwind 클래스를 styled-components처럼 사용 가능
const Container = styled.div`
  ${tw`flex flex-col items-center justify-center h-screen bg-gray-100`}
`;
```

```

const Title = styled.h1`
  ${tw`text-3xl font-bold text-blue-600`}
`;

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>

      <Title>Welcome to My PWA</Title>

      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App

```

3 prettier, eslint 설치

1. Prettier와 ESLint 관련 플러그인을 설치

```
npm install --save-dev prettier eslint-config-prettier eslint-plugin-prettier
```

설치된 패키지 설명:

- **prettier**: 코드 포매팅 도구
- **eslint-config-prettier**: ESLint와 Prettier 간의 충돌을 방지
- **eslint-plugin-prettier**: ESLint에서 Prettier 규칙을 실행

2. **eslint.config.js** 수정

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'
import prettierConfig from 'eslint-config-prettier'
import prettierPlugin from 'eslint-plugin-prettier'

export default tseslint.config(
  { ignores: ['**/*', '!src', '!src/'] }, // .prettiignore 과 동일하게 설정
  {
    extends: [
      js.configs.recommended,
      ...tseslint.configs.recommended,
      prettierConfig, // Prettier 설정 추가 (충돌 방지)
    ],
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
  },
)
```



```

    prettier: prettierPlugin, // Prettier 플러그인 추가
  },
  rules: {
    ...reactHooks.configs.recommended.rules,
    'react-refresh/only-export-components': [
      'warn',
      { allowConstantExport: true },
    ],
    'prettier/prettier': 'error', // Prettier 규칙을 ESLint에서 에러로 표시
  },
},
)

```

2. `.prettierignore` 파일 생성

```

**

!src/
!src/**

```

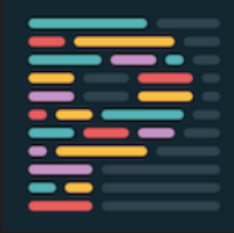
3. `.prettierrc` 파일 생성


```


{
  "semi": false,
  "singleQuote": true,
  "trailingComma": "all",
  "tabWidth": 2,
  "printWidth": 80,
  "arrowParens": "always",
  "endOfLine": "auto",
  "bracketSpacing": true
}


```

4. VS code extension 에서 아래 설치



Prettier - Code formatter
 Prettier  prettier.io | 55,252,877 | ★★★★★ (476) | ❤️ 스폰서
 Code formatter using prettier
 사용 안 함 ▼ 제거 ▼ ☒ 자동 업데이트 ⚙️



ESLint
 Microsoft  microsoft.com | 41,716,748 | ★★★★★ (241)
 Integrates ESLint JavaScript into VS Code.
 사용 안 함 ▼ 제거 ▼ 시험판 버전으로 전환 ☒ 자동 업데이트 ⚙️

5. VS Code 에서 ctrl + shift + p → settings.json 에 추가

```
// settings.json

"editor.defaultFormatter": "esbenp.prettier-vscode",
"editor.formatOnSave": true,
"editor.codeActionsOnSave": {
  "source.fixAll.eslint": "explicit"
},
```

6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음

4 zustand 설치

```
npm install zustand
```

5 라우터 설치

```
npm install react-router-dom
```

6 axios 설치

```
npm i axios
```

7 Shadcn 설치 (UI 라이브러리)

```
npx shadcn@latest init
```

```
newyork - neutral
```

(사용할 UI 설치)

```
npx shadcn@latest add drawer
```

아이콘 설치

```
npm install react-icons --save
```

8 절대 경로 설정

```
npm install --save-dev @types/node
```

1. vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: [
      { find: '@', replacement: path.resolve(__dirname, 'src') }
    ],
  },
});
```

2. tsconfig.json

```
{
  "files": [],
  "references": [
    { "path": "./tsconfig.node.json" },
    { "path": "./tsconfig.app.json" },
  ],
  "compilerOptions": {
    "baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정
    "paths": {
      "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑
    }
  }
}
```

3. tsconfig.app.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
```

```

"noFallthroughCasesInSwitch": true,

/* Path */
"baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정
"paths": {
  "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑
},
},
"include": ["src"]
}

```

★전체창 모드 (키오스크)

1. 키오스크 모드(주소창 없이 전체화면)

- (A) 브라우저를 키오스크 모드로 실행
 - 크롬 기준: `-kiosk` 옵션을 줘서 실행 가능
 - 명령어 예시 (Windows 기준):

```

"C:\Program Files\Google\Chrome\Application\chrome.exe" --kiosk
http://localhost:5173

```

- (B) 아예 Electron으로 감싸서 실행 (진짜 앱처럼)
 - 이렇게 하면 처음부터 주소창 없이 네이티브 앱처럼 돌아감. (별도 설치 필요)

2. 키오스크 모드 설정하는 방법

- Vite (`npm run dev`)는 "로컬 서버"만 띄워주는 역할, 브라우저를 띄우는 건 **OS / 브라우저 실행 옵션**
- 개발 서버에서는 `npm run dev` 로 서버 켜고, 별도로 `크롬 --kiosk` 로 띄우기

// package.json 수정 ("브라우저 자동 실행" 추가)

```

"scripts": {
  "dev": "vite",

```

```
// Chrome이 PATH에 등록되어 있다면 (start는 윈도우 CMD 기준)
"dev:kiosk": "start chrome --kiosk --app=http://localhost:5173"

// 없다면
"dev:kiosk": "\"C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe\" --kiosk http://localhost:5173"
}
```

```
npm run dev
```

```
npm run dev:kiosk
```



환경 변수 설정 (윈도우에서 PATH 추가 방법)

1. 시작 메뉴에서 "환경 변수" 검색 → "시스템 환경 변수 편집" 클릭
2. 하단의 "환경 변수(N)..." 버튼 클릭
3. 아래쪽에서 "시스템 변수" 영역 → **Path** 선택 → "편집" 클릭
4. "새로 만들기" 클릭 → 아래 경로 입력:

```
C:\Program Files\Google\Chrome\Application
```

5. 확인하기 (터미널(CMD) 또는 PowerShell)

```
chrome --version
```

```
where chrome
```

결과 나오면 성공:

```
Google Chrome 123.0.xxxxx
```

```
C:\Program Files\Google\Chrome\Application\chrome.exe
```

3. 주소창은 제거 되었으나, 탭이 뜸

기본적으로 크롬은 기존 사용자 프로필을 사용할 때:

- `-app` 또는 `-kiosk` 옵션이 일부 제대로 적용되지 않는 경우가 있음
(
`-app=URL` 만 쓰면 주소창은 숨지만 탭은 뜰 수 있음)
- `-kiosk` 는 전체 화면이지만 이전 브라우징 기록이나 창 상태(탭 포함)에 영향을 받음
(캐시나 사용자 설정이 걸리면 정상 동작 안 할 수 있음)
⇒ 이전에 열었던 방식대로 다시 열리기도 함 → 탭이 뜨는 이유

✓ 해결 방법

`-user-data-dir` 로 새 환경을 강제하는 게 키오스크에선 거의 필수 (항상 깨끗한 프로필로 실행)

- 임시 사용자 데이터 디렉토리를 만들기 때문에, 항상 새로운 상태에서 완전한 키오스크 모드가 보장됨
- 정상적으로 전체화면 (탭 없이) 실행됨

⇒ 캐시 없이 실행 (임시 유저 디렉토리를 만들어 캐시 없이 실행)

```
// package.json
```

```
start chrome --kiosk --app=http://localhost:5173 --user-data-dir=%TEMP%\kiosk-data
```

```
"scripts": {  
  "dev:kiosk": "start chrome --kiosk --app=http://localhost:5173 --user-data-dir=%TEMP%\\kiosk-data"  
}
```

```
-----  
-----
```

```
// 아래 코드로 자동으로 Vite 서버도 같이(npm run dev) 실행 가능  
npm install concurrently wait-on --save-dev
```

```
// package.json
```

```
"scripts": {  
  "dev:kiosk": "concurrently \"npm run dev\" \"wait-on http://localhost:5173 && start chrome --kiosk --app=http://localhost:5173 --user-data-dir=%TE
```

```
MP%\\kiosk-data\""
```

```
}
```

```
FE > kiosk > {} package.json > {} dependencies
1  {
2    "name": "kiosk",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    > 디버그
7    "scripts": {
8      "dev": "vite",
9      "build": "tsc -b && vite build",
10     "lint": "eslint .",
11     "preview": "vite preview",
12     "dev:kiosk": "start chrome --kiosk --app=http://localhost:5173",
13     "kiosk": "start chrome --kiosk --app=http://localhost:5173 --user-data-dir=%TEMP%\\kiosk-data\"\"",
14   },
15   "dependencies": {}
```

방법	동작	단축키 가능 여부
<code>--kiosk</code>	완전 키오스크	❌ 거의 다 막힘 (<code>Alt+F4</code> 만 가능)
<code>--app</code>	간이 키오스크	○ 일부 단축키 가능 (<code>Esc</code> , <code>Ctrl+W</code>)
<code>window.close()</code>	JS 종료	❌ 시스템 창에는 미작동
AutoHotKey	강제 키 매핑	○ 가능

카메라 + 얼굴 인식

- 키오스크에 **외부 USB 카메라**를 꽂으면 웹 브라우저에서도 인식할 수 있어.
- 얼굴 인식은 브라우저 안에서 JavaScript로도 가능해.
 - 대표 라이브러리:
 - `face-api.js` (Tensorflow.js 기반, 쉬움)
 - `MediaPipe Face Detection` (구글 제공, 빠르고 가벼움)

단계적으로 진행하면:

- `navigator.mediaDevices.getUserMedia()` 로 카메라 연결

- `face-api.js` 같은 걸로 얼굴 인식

👉 이 부분은 프로젝트 기본 세팅 끝나면 구체적으로 같이 구현해보자.

4.2 앱 시작 세팅

*npm 패키지 이름 규칙

1. 소문자만 사용 가능
2. 공백, 특수 문자 금지
3. 대시(-), 밑줄(_), 숫자 사용 가능
4. 예약어 사용 금지 (`test` , `npm` , `react` 같은 예약어 ❌)

1 vite + react + ts 설치

```
npm create vite@latest user-app --template react-ts  
→ React, TypeScript 선택
```

```
cd 폴더
```

```
npm install
```

2 pwa 설치

```
npm install vite-plugin-pwa
```

3 tailwindcss + styled-components + twin.macro 설치

```
# v3 버전 설치 방법
```

```
# Tailwind CSS v3.x 설치
```

```
npm i -D tailwindcss@3 postcss autoprefixer
```

```
npx tailwind init -p
```

```
# styled-components 설치 (TypeScript 지원 포함)
```

```
npm install styled-components
npm install -D @types/styled-components
```

```
# twin.macro 설치
npm install twin.macro
npm install vite-plugin-babel-macros
```

```
npm install -D @emotion/react @emotion/styled
```

파일 수정

public 폴더 아래 → icons 폴더 생성 → 아래 파일들 넣기

[favicomatic \(1\).zip](#)

```
// vite.config.ts

import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { VitePWA } from 'vite-plugin-pwa';
import macrosPlugin from 'vite-plugin-babel-macros'

// https://vite.dev/config/
export default defineConfig({
  plugins: [
    react(),
    VitePWA({
      registerType: 'autoUpdate',
      devOptions: {
        enabled: true, // 개발환경에서 pwa 기능활성화
      },
      manifest: {
        name: 'keywi',
        short_name: '키위',
        description: '1:1 키보드 견적 서비스, 나만의 키보드를 맞추고 뽑내보세요.',
        start_url: '/',
      },
    }),
  ],
});
```

```
display: 'standalone', // 네이티브앱처럼 화면 전체를 채움
background_color: '#ffffff',
theme_color: '#ffffff',
lang: 'ko',
"icons": [
  {
    "src": "icons/apple-touch-icon-57×57.png",
    "sizes": "57×57",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-60×60.png",
    "sizes": "60×60",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-72×72.png",
    "sizes": "72×72",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-114×114.png",
    "sizes": "114×114",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-120×120.png",
    "sizes": "120×120",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-144×144.png",
    "sizes": "144×144",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-152×152.png",
    "sizes": "152×152",
```

```

        "type": "image/png"
    },
    {
        "src": "icons/apple-touch-icon-180×180.png",
        "sizes": "180×180",
        "type": "image/png"
    },
    {
        "src": "icons/favicon-32×32.png",
        "sizes": "32×32",
        "type": "image/png"
    },
    {
        "src": "icons/favicon-96×96.png",
        "sizes": "96×96",
        "type": "image/png"
    },
    {
        "src": "icons/favicon-16×16.png",
        "sizes": "16×16",
        "type": "image/png"
    },
    {
        "src": "icons/logo192.png",
        "sizes": "192×192",
        "type": "image/png"
    },
    {
        "src": "icons/logo512.png",
        "sizes": "512×512",
        "type": "image/png"
    }
],
},
}),
macrosPlugin(),
],
})

```

```
// tailwind.config.js 수정

/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.js,ts,jsx,tsx",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

```
// postcss.config.js 수정

export default {
  plugins: {
    'postcss-import': {},
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

test 코드

```
/* index.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

/* 경고 표시 뜰 때 */

- ✓ 해결 방법 1: VSCode Tailwind CSS 확장 프로그램 설치
- ✓ 해결 방법 2: settings.json 수정
 - VSCode에서 Ctrl + Shift + P (⌘ + Shift + P on Mac)

- "Preferences: Open Settings (JSON)" 검색해서 선택

```
{  
  "css.lint.unknownAtRules": "ignore"  
}
```

```
// App.tsx  
import { useState } from 'react'  
import reactLogo from './assets/react.svg'  
import viteLogo from '/vite.svg'  
import './App.css'  
  
import tw from "twind.macro";  
import styled from "styled-components";  
  
// Tailwind 클래스를 styled-components처럼 사용 가능  
const Container = styled.div`  
  ${tw`flex flex-col items-center justify-center h-screen bg-gray-100`}  
`;  
  
const Title = styled.h1`  
  ${tw`text-3xl font-bold text-blue-600`}  
`;  
  
function App() {  
  const [count, setCount] = useState(0)  
  
  return (  
    <>  
    <div>  
      <a href="https://vite.dev" target="_blank">  
        <img src={viteLogo} className="logo" alt="Vite logo" />  
      </a>  
      <a href="https://react.dev" target="_blank">  
        <img src={reactLogo} className="logo react" alt="React logo" />  
      </a>  
    </div>  
    <h1>Vite + React</h1>  
  )  
}
```

```

<Title>Welcome to My PWA</Title>

<div className="card">
  <button onClick={() => setCount((count) => count + 1)}>
    count is {count}
  </button>
  <p>
    Edit <code>src/App.tsx</code> and save to test HMR
  </p>
</div>
<p className="read-the-docs">
  Click on the Vite and React logos to learn more
</p>
</>
)
}

export default App

```

4 zustand 설치

```
npm install zustand
```

5 TanStack Query 설치

```
npm install @tanstack/react-query
```

6 prettier, eslint 설치

1. Prettier와 ESLint 관련 플러그인을 설치

```
npm install --save-dev prettier eslint-config-prettier eslint-plugin-prettier
```

설치된 패키지 설명:

- **prettier**: 코드 포매팅 도구

- **eslint-config-prettier**: ESLint와 Prettier 간의 충돌을 방지
- **eslint-plugin-prettier**: ESLint에서 Prettier 규칙을 실행

2. **eslint.config.js** 수정

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'
import prettierConfig from 'eslint-config-prettier'
import prettierPlugin from 'eslint-plugin-prettier'

export default tseslint.config(
  { ignores: ['**/*', '!src', '!src/'] }, // .prettiignore 과 동일하게 설정
  {
    extends: [
      js.configs.recommended,
      ...tseslint.configs.recommended,
      prettierConfig, // Prettier 설정 추가 (충돌 방지)
    ],
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
      prettier: prettierPlugin, // Prettier 플러그인 추가
    },
    rules: {
      ...reactHooks.configs.recommended.rules,
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
)
```



```
'prettier/prettier': 'error', // Prettier 규칙을 ESLint에서 에러로 표시
},
},
)
```

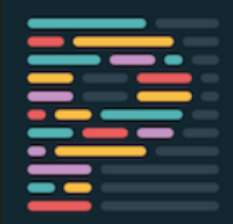
2. **.prettierignore** 파일 생성

```
**
!src/
!src/**
```


3. **.prettierrc** 파일 생성

```
{
  "semi": false,
  "singleQuote": true,
  "trailingComma": "all",
  "tabWidth": 2,
  "printWidth": 80,
  "arrowParens": "always",
  "endOfLine": "auto",
  "bracketSpacing": true
}
```


4. VS code extension 에서 아래 설치

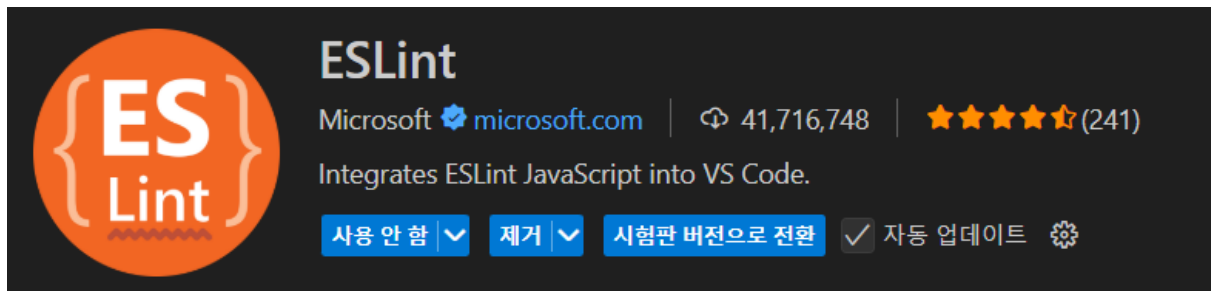


Prettier - Code formatter

Prettier  prettier.io | 55,252,877 | ★★★★★ (476) | ❤️ 스폰서

Code formatter using prettier

사용 안 함 제거 ☒ 자동 업데이트 



5. VS Code 에서 ctrl + shift + p → settings.json 에 추가

```
// settings.json

"editor.defaultFormatter": "esbenp.prettier-vscode",
"editor.formatOnSave": true,
"editor.codeActionsOnSave": {
  "source.fixAll.eslint": "explicit"
},
```

6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음

7 ngrok 설치 - 로컬에서 http → https 변환용 (배포 전)



npm을 이용한 간단한 설치 방법

폴더 → `npm install -g ngrok` → 버전 확인 `ngrok -v` → 실행 `ngrok http 포트번호`

<https://ngrok.com/downloads/windows?tab=download>

다운받은 파일의 압축을 해제하고 ngrok.exe를 관리자 권한으로 실행

1. Ngrok 계정 생성

1. Ngrok 공식 웹사이트로 이동
2. 계정을 만들고 로그인

2. 인증 토큰 확인

1. 로그인 후 Ngrok 인증 토큰 페이지로 이동

2. `authtoken` 을 복사

3. 로컬 환경에 Ngrok 인증 등록

터미널에서 다음 명령어 실행 (복사한 `authtoken` 을 붙여넣기)

```
ngrok config add-authtoken <YOUR_AUTH_TOKEN>
```

4. Ngrok 다시 실행

```
ngrok http 3000 # 예제 (3000번 포트 노출)
```

*기존 인증 토큰 재설정

만약 이전에 등록한 인증 토큰이 잘못되었거나 만료된 경우, 아래 명령어로 다시 설정

```
ngrok authtoken <NEW_AUTH_TOKEN>
```

5. `vite.config.js` 수정

`vite.config.js` 파일을 열고, `server.allowedHosts` 옵션을 추가하여 ngrok 도메인을 허용합니다.

```
javascriptimport { defineConfig } from 'vite'

export default defineConfig({
  server: {
    allowedHosts: ['2623-59-20-195-127.ngrok-free.app'], // ngrok 주소를 명시적
  },
})
```

만약 ngrok 주소가 매번 바뀌어 관리가 번거롭다면, 모든 호스트를 허용하도록 설정할 수도 있습니다. 하지만 이는 보안상 권장되지 않습니다.

```
javascriptexport default defineConfig({
  server: {
    allowedHosts: true, // 모든 호스트를 허용 (DNS 리바인딩 공격에 취약할 수 있음)
  },
})
```

8 실행

```
npm run dev
```

```
npm run build
```

```
npm run preview
```

9 절대 경로 설정

```
npm install --save-dev @types/node
```

1. vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: [
      { find: '@', replacement: path.resolve(__dirname, 'src') }
    ],
  }
});
```

2. tsconfig.json

```
{
  "files": [],
  "references": [
    { "path": "./tsconfig.node.json" },
    { "path": "./tsconfig.app.json" },
  ],
  "compilerOptions": {
    "baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정
    "paths": {
```

```

    "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑
  }
}
}

```

3. tsconfig.app.json

```

{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "isolatedModules": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,

    /* Path */
    "baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정
    "paths": {
      "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑
    }
  },
  "include": ["src"]
}

```

10 라우터 설치

```
npm install react-router-dom
```

```
// App.tsx

import './App.css'
import Fonts from './styles/fonts'
import { Route, Routes } from 'react-router-dom'

function App() {
  return (
    <>
      <Fonts />
      <Routes>
        <Route path="/" element={<h1>Home</h1>} />
      </Routes>
    </>
  )
}

export default App
```

```
// main.tsx

import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'
import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </StrictMode>
)
```

```
</StrictMode>,  
)
```

1 1 Shadcn 설치 (UI 라이브러리)

```
npx shadcn@latest init
```

```
newyork - neutral
```

(사용할 UI 설치)

```
npx shadcn@latest add drawer
```

1 2 axios 설치

```
npm i axios
```

1 3 react cookie 설치

```
npm install react-cookie
```

4.3. env

```
// .env  
VITE_BASE_URL= api 배포 주소  
VITE_FACE_URL= 안면인식 gpu 서버 배포주소  
VITE_VAPID_KEY=FCM 토큰 키
```

4. 백엔드 설정

```
# application.yml - 기본 설정 파일  
server:
```

```
port: 8080
servlet:
  context-path: /

tomcat:
  connection-timeout: 180000
  keep-alive-timeout: 120000
  max-keep-alive-requests: 200
  max-connections: 8192
  accept-count: 100
  threads:
    max: 200
    min-spare: 10

spring:
  application:
    name: orderme
  datasource:
    url: jdbc:mysql://localhost:3306/orderme?useSSL=false&serverTimezone=Asia/Seoul&allowPublicKeyRetrieval=true&createDatabaseIfNotExist=true
    username: 유저명
    password: 유저 비밀번호
    driver-class-name: com.mysql.cj.jdbc.Driver
  hikari:
    maximum-pool-size: 10
    connection-timeout: 30000
# Redis 설정 (세션 또는 캐시 저장소)
redis:
  host: localhost
  port: 6379

mvc:
  async:
    request-timeout: 300000

# API 키 / 나중에 뺄 예정
openweathermap:
```



```
api:
  key: "None"

geocoding:
  api:
    key: "None"

# MyBatis 설정
mybatis:
  mapper-locations: classpath:mappers/**/*.xml
  configuration:
    map-underscore-to-camel-case: true
    default-fetch-size: 100
    default-statement-timeout: 30
  type-aliases-package: com.ssafy.orderme.user.model,com.ssafy.orderme.kiosk.model,com.ssafy.orderme.recommendation.model

# JWT 설정
jwt:
  secret: jwt 시크릿키
  token-validity-in-seconds: 86400 # 24시간

# CoolSMS 설정
coolsms:
  api:
    key: coolSMS api키
    secret: 시크릿키
  sender:
    number: 발송자 번호

# 토스 페이먼트 API 설정
toss:
  client-key: 토스 테스트용 클라이언트 키
  secret-key: 토스 테스트용 시크릿 키
  success-url: http://localhost:8080/api/payments/success
  fail-url: http://localhost:8080/api/payments/fail

# 로깅 설정
```

logging:
 level:
 root: INFO
 com.cafe.api: DEBUG
 org.springframework.security: INFO
 org.mybatis: DEBUG
 org.apache.coyote.http11: DEBUG
 org.apache.tomcat: INFO

menu:
 limit: 3

fcm:
 connection-timeout: 30000
 read-timeout: 30000
 write-timeout: 30000
 retry:
 max-attempts: 3
 backoff-initial-interval: 1000
 backoff-multiplier: 2.0

management:
 endpoints:
 web:
 exposure:
 include: health,info
 endpoint:
 health:
 show-details: never
 probes:
 enabled: true
 group:
 liveness:
 include: livenessState
 readiness:
 include: readinessState

5. AI 세팅

5.1. 키오스크 세팅

CUDA 12.1.1과 CuDNN 9.1.1 이 설치되어 있다고 가정.

파이썬은 3.10.XX 버전 설치 필요.

Insightface 환경 세팅

1. Windows의 경우 Visual Studio 2019 dev tools 설치해야 c 컴파일러 사용 가능

VS BuildTools 16.10.4

C++를 사용한 데스크톱 개발 을 선택하여 설치

- 이때 Cmake, Windows SDK, MSVC v142 필수 설치

2. Qdrant 설치

wsl에서 불러오기 or EC2에서 불러오고 포트 열어서 테스트

```
# Docker 사용하여 Qdrant 설치
docker run -d --name qdrant -p 6333:6333 -v $(pwd)/qdrant_storage:/qdrant/storage qdrant/qdrant
# python 클라이언트 설치
pip install qdrant-client
```

3. CUDA & CuDNN 설치

CUDA 12.1.1

CuDNN 9.1.1

설치 완료 후 `nvcc -V` 명령어를 사용해서 설치 확인

4. Python 환경 설치

1. 레포지토리의 AI/BE 폴더에서 깃배시를 통해 install.sh를 실행하여 환경 설치.

2. 설치가 완료되면 run.sh를 실행