프로젝트 포팅 메뉴얼

노션 링크

(1) 개요

- **서비스명:** NAMUH
- **팀명:** 부리부리몬(S13P31E108)
- **주요 구성:**
    - **Frontend:** React 기반의 PWA 앱 및 대시보드
    - **Backend:** Spring Boot(메인 API), FastAPI(AI)
    - **AI:** Object Detection, Reinforcement Learning
    - **Embedded:** Raspberry Pi, ESP32, STM32 제어 코드
    - **Database:** MySQL, Redis
    - **Infra:** AWS EC2, S3, Lambda, Docker, Jenkins, Nginx, n8n

(2) 개발 및 서버 환경

2.1. 서버 환경

- **OS:** Ubuntu 22.04 LTS (Jammy)
- **주요 사용 도구:**
    - **형상 관리:** GitLab
    - **CI/CD:** Jenkins, Docker
    - **웹 서버:** Nginx (Reverse Proxy)
    - **통신:** Mattermost (GitLab 웹훅 연동)

2.2. UFW 및 포트 설정

- **💻 웹 서비스 및 개발 환경 포트**

포트	서비스	내부 포트	비고
22	SSH	-	EC2 원격 접속
80	HTTP	-	Nginx (HTTPS로 리다이렉트)
443	HTTPS	-	Nginx (메인 서비스 프록시)
-	backend-spring	8080	Nginx를 통해 api.buriburi.monster/spring 로 프록시
-	backend-fastapi	8081	Nginx를 통해 api.buriburi.monster/fastapi 로 프록시
-	frontend-app	3000	Nginx를 통해 app.buriburi.monster으로 프록시
-	frontend-dashboard	3001	Nginx를 통해 buriburi.monster으로 프록시
20080/20443	OpenVidu (HTTP/HTTPS)		OpenVidu 서비스 포트

- **💾 데이터베이스 및 캐시 포트**

포트	서비스명	비고
33066(3066)	Mysql	개발 네트워크(devnet) 내부에서 사용하는 포트는 3066
6381(6379)	Redis	개발 네트워크(devnet) 내부에서는 6379, 외부 TCP 접속은 6381 사용
5540	Redis Insight	Redis 데이터 시각화/관리 도구

- **📡 기타 서비스 및 프로토콜 포트**

포트	서비스명	비고
5678	n8n	워크플로우 자동화 도구 (GitLab Merge Request에 대한 챗봇 기반 코드 리뷰 시스템 구축)
8883/9001	MQTT (SSL/WSS)	8883은 백엔드 MQTTS (보안 MQTT), 9001은 프론트 브라우저 접근용 WSS (보안

포트	서비스명	비고
		WebSocket)
3478	STUN 프로토콜 (NAT 통과 지원)	라즈베리파이 OpenVidu stun 서버 연결

3.3. 개발 환경

- **Backend - FastAPI**

구분	사용 기술
Language	Python 3.12
IDE	Visual Studio Code 1.106.2
Framework	FastAPI 0.120.3
Library	Pydantic, SQLAlchemy, PyJWT, dependency-injector, aiomqtt, boto3, OpenAI
Runtime	Uvicorn (ASGI Server)
Features	STT/TTS (OpenAI Whisper), MQTT Messaging, S3 File Management, JWT Auth

- **Backend - Spring Boot**

구분	사용 기술
Language	Java 17
IDE	IntelliJ IDEA 2025.2.4 (Ultimate Edition)
Framework	Spring Boot 3.5.6
Library	Spring Security, Spring Data JPA, OAuth2, JWT
Build Tool	Gradle
Features	Member Management, OAuth2 Social Login, Channel & Media Management, JWT Auth

- **Frontend**

구분	사용 기술
Language	TypeScript v5
Runtime Environment	Node.js v22
IDE	Visual Studio Code 1.106.2, WebStorm 2025.2.4
Framework	React v18/19
Library	React Router DOM, React Three Fiber, @react-three/drei, MQTT.js, @stomp/stompjs, Axios, JWT-decode, @lottiefiles/dotlottie-react
Build Tool	Vite v5
Styling	TailwindCSS v4
3D Graphics	Three.js, React Three Fiber, Postprocessing
PWA	vite-plugin-pwa, Workbox
Features	Mobile PWA App, 3D Robot Visualization Dashboard, Real-time MQTT/WebSocket Communication

- **Embedded & IoT**

구분	사용 기술
Languages	C/C++, Python
IDE	Visual Studio Code 1.106.2, Arduino IDE
Microcontroller	STM32 (Cortex-M), ESP32 (Dual-core Xtensa LX6, WiFi 802.11 b/g/n)
SBC	Raspberry Pi (Python 3.11, Python 3.13, Picamera2, libcamera)
Hardware & Robotics	DOFBOT 6-DOF Robot Arm x2, Servo Motors (ESP32Servo, PWM Control)
Development Tools	Visual Studio Code, Arduino IDE, **Isaac Sim, Isaac Lab**
Arduino Libs	FastLED, PubSubClient, ArduinoJson, ESP32Servo
Python Libs	OpenCV, Arm_Lib, PySerial
Communication	MQTT over WiFi (JSON payload), Serial (UART/USB)
Features	Servo Control, LED Facial Expression, Dual Robot Arm Control, Face Tracking

- **AI & CV**

구분	사용 기술
Language	Python 3.12
Vision	OpenCV (cv2), MediaPipe
AI API	OpenAI Whisper (STT), OpenAI TTS, Porcupine (Wake Word Detection)
Video Processing	aiortc (WebRTC), PyAV (FFmpeg Binding)
Detection	Haar Cascade Face Detection, MediaPipe Face Landmarker, Gesture Recognition
Features	Real-time Face Tracking, Gesture Recognition, Speech-to-Text, Text-to-Speech, Wake Word Detection
Runpod	GPU: RTX 4500 (20GB VRAM) RAM: 54GB CPU: 12 vCPU SSD: 80GB

- **DevOps & Infra**

구분	사용 기술
Intance Type	T2.XLARGE
CPU	4 vCPUs
RAM	16GB
Storage (Disk)	SSD: 310 GB
OS	Ubuntu 22.04.5 LTS
Kernel	Linux 6.8.0-1040-aws x86_64
Docker	v28.5.1
Docker Compose	v2.40.2

구분	사용 기술
Jenkins	2.528.1
nginx	nginx/1.27

- **Database & Storage**

구분	사용 기술
RDBMS	MySQL 8.4.6
Cache	Redis 7.4.5 (jemalloc-5.3.0)
Storage	AWS S3

- **Communication Protocols**

구분	사용 기술
Protocols	HTTP/HTTPS, WebSocket, MQTT (QoS 0), WebRTC
Data Formats	JSON, Base64, JPEG, MP4/WebM
Security	JWT, OAuth 2.0, CORS, TLS/HTTPS

- **Control Systems**

구분	사용 기술
PID Controller	Face Tracking Servo Control, Stable Position Control
Easing Functions	Smooth Motion Animation, Natural Movement
State Machine	Robot Action Flow Control, Command Preemption
Async Processing	Python asyncio, aiomqtt, Threading, Subprocess

(3) 사전 설치 및 도구 버전

- 모든 서비스는 Docker 컨테이너 위에서 동작하므로, EC2에는 Docker와 Docker Compose만 설치하면 됩니다.
- **3.1. Backend**
    - `backend-spring`
        - Java: OpenJDK 17
        - Framework: Spring Boot 3.5.6
        - Build Tool: Gradle
    - `backend-fastapi`
        - Python: 3.12
        - Framework: FastAPI 0.120.3, SQLAlchemy
        - ASGI Server: Uvicorn
- **3.2. Frontend**
    - `frontend-app` & `frontend-dashboard`
        - Runtime: Node.js 22
        - Framwork: React 19
        - Build Tool: Vite
- **3.3. Database**
    - MySQL: 8.4.6 for Linux on x86_64
    - Redis: 7.4.5
- **3.4. Docker, Docker Compose 설치**

```
# 1. 패키지 목록 업데이트 및 필수 패키지 설치
sudo apt-get update
sudo apt-get install ca-certificates curl

# 2. Docker 공식 GPG 키 저장 디렉토리 생성
# apt에 GPG 키를 안전하게 보관할 디렉토리 생성 및 권한 755 설정
sudo install -m 0755 -d /etc/apt/keyrings

# 3. Docker 공식 GPG 키 다운로드 및 저장
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/do
cker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# 4. docker 패키지를 받아올 apt 저장소 등록
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] http
```

```
s://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stab
le" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 5. 저장소 등록 후 패키지 목록 다시 업데이트
sudo apt-get update
```

```
# docker package 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker
-compose-plugin -y
```

```
# docker 설치 확인 후 후속 설정
# 1. Docker 실행 테스트
sudo docker run hello-world

# 2. sudo 없이 docker 명령어 사용 설정
# 현재 사용자를 docker 그룹에 추가합니다.
sudo usermod -aG docker $USER
# 변경 사항을 현재 세션에 반영
newgrp docker

# 3. 시스템 부팅 시 Docker 자동 시작 설정
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

(4) EC2 프로젝트 구조

🌳 **홈 디렉토리 (`/home/ubuntu/`)**

```
.
├── certbot_data/          # SSL 인증서 (LetsEncrypt) 데이터
│   └── letsencrypt/
│       ├── live/
│       └── ...
│
├── dockerfile/             # Jenkins Docker 설정 파일
│   ├── jenkins-docker-compose.yaml
│   └── jenkins.dockerfile
│
├── jenkins_home/            # Jenkins 서버 데이터
│   ├── jobs/              # Jenkins 파이프라인 Job 목록
│   │   ├── dev-backend-fastapi
│   │   ├── dev-backend-spring
│   │   ├── dev-frontend-app
│   │   ├── dev-frontend-dashboard
│   │   ├── develop-main
│   │   └── global-mr-guardian
│   │
│   ├── workspace/           # Jenkins 빌드 작업 공간
│   │   ├── dev-backend-fastapi
│   │   ├── dev-backend-spring
│   │   ├── dev-frontend-app
│   │   ├── dev-frontend-dashboard
│   │   └── develop-main
│   │
│   ├── plugins/            # (설치된 Jenkins 플러그인 다수)
│   │   └── ...
│   │
│   ├── credentials.xml       # Jenkins 인증 정보
│   ├── config.xml           # Jenkins 메인 설정
│   └── ... (기타 Jenkins 설정 파일들)
│
├── logs/                  # 애플리케이션 로그
│   ├── fastapi/
│   └── spring/
│
└── oily/                  # 서비스 운영을 위한 Docker 볼륨 및 데이터
    ├── edge/
    │   ├── mysql_dev/       # MySQL 데이터
    │   ├── n8n_data/        # n8n 워크플로우 데이터
    │   ├── nginx_data/       # Nginx 설정 파일 (conf.d/, nginx.conf)
    │   ├── redis_dev/        # Redis 데이터
    │   └── ...
    └── ...
```

🌳 **OpenVidu 디렉토리 (`/opt/openvidu/`)**

```
/opt/openvidu/
├── docker-compose.yml        # OpenVidu 메인 Docker Compose 파일
├── docker-compose.override.yml
├── .env                      # OpenVidu 환경변수 파일
│
├── certificates/             # RTC 서비스용 SSL 인증서
│   └── live/
│       └── rtc.buriburi.monster/
│
├── recordings/               # 화상 통화 녹화본 저장 위치
│
├── kurento-logs/             # Kurento 미디어 서버 로그
│   └── ... (날짜별 로그 파일 다수)
│
└── ... (기타 OpenVidu 구성 요소)
```

(5) 프로젝트 다운로드

```
# GitLab 프로젝트 클론
git clone https://lab.ssafy.com/s13-final/S13P31E108.git

# 디렉터리 이동
cd S13P31E108
```

(6) 환경 변수 설정

- 배포는 Jenkins를 통해 자동화되어 있으며, Jenkins의 Credentials 플러그인에 `dev-env_*` 형태의 ID로 환경변수 파일 (`.env`)들이 등록되어 있습니다.
- 수동으로 배포하거나 로컬에서 테스트할 경우, 각 프로젝트 루트에 아래와 같은 형식의 `.env` 파일을 생성해야 합니다.
    - `backend-spring`
        - `application.yaml`

```
server:
  port: ${SERVER_PORT}
  servlet:
    context-path: ${SERVER_CONTEXT_PATH}

swagger:
  uri: ${SWAGGER_URI}

spring:
  application:
    name: ${SPRING_APPLICATION_NAME}
  data:
    redis:
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}
      password: ${REDIS_PASSWORD}
  datasource:
    url: ${DB_URL}
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
  mqtt:
    broker-url: ${MQTT_BROKEN_URL}
    username: ${MQTT_USERNAME}
    password: ${MQTT_PASSWORD}
    topic: ${MQTT_TOPIC}
  jpa:
    hibernate:
      ddl-auto: ${JPA_HIBERNATE_DDL}
    properties:
      hibernate:
        format_sql: ${JPA_FORMAT_SQL}
        dialect: ${JPA_DIALECT}
    show-sql: ${JPA_SHOW_SQL}

  cloud:
    aws:
      credentials:
        access-key: ${AWS_ACCESS_KEY}
        secret-key: ${AWS_SECRET_KEY}
      region:
        static: ${AWS_REGION_STATIC}
      s3:
        bucket: ${AWS_S3_BUCKET}

  security:
    oauth2:
      client:
        registration:
          google:
            client-id: ${OAUTH2_GOOGLE_CLIENT_ID}
            client-secret: ${OAUTH2_GOOGLE_CLIENT_SECRET}
            scope:
              - email
              - profile
            redirect-uri: "${OAUTH2_REDIRECT_URI}"
      authorize-uri: ${SECURITY_OAUTH2_AUTHORIZE_URI}
      redirect-uri: ${SECURITY_OAUTH2_REDIRECT_URI}
      client-redirect-uri: ${SECURITY_OAUTH2_CLIENT_REDIRECT_URI}

openvidu:
  url: ${OPENVIDU_URL}
  secret: ${OPENVIDU_SECRET}
  session-prefix: ${OPENVIDU_SESSION_PREFIX}

security:
  jwt:
    secret-key: ${SECURITY_JWT_SECRET_KEY}
    expire-time:
      access-token: ${SECURITY_JWT_EXPIRE_TIME_ACCESS}
      refresh-token: ${SECURITY_JWT_EXPIRE_TIME_REFRESH}
  whitelist:
    GET: ${SECURITY_WHITELIST_GET}
    POST: ${SECURITY_WHITELIST_POST}
    PUT: ${SECURITY_WHITELIST_PUT}
    DELETE: ${SECURITY_WHITELIST_DELETE}
    PATCH: ${SECURITY_WHITELIST_PATCH}
    OPTIONS: ${SECURITY_WHITELIST_OPTION}
  cors:
    allowed-origins: ${SECURITY_CORS_ALLOWED_ORIGINS}
    allowed-methods: ${SECURITY_CORS_ALLOWED_METHODS}
    allowed-headers: ${SECURITY_CORS_ALLOWED_HEADERS}
    allow-credentials: ${SECURITY_CORS_ALLOW_CREDENTIALS}
    exposed-headers: ${SECURITY_CORS_EXPOSED_HEADERS}
    max-age: ${SECURITY_CORS_MAX_AGE}
  role:
    admin: ${SECURITY_ROLE_ADMIN}
expire-time:
  sign-up-expire-time: ${SING_UP_EXPIRE_TIME}

robot:
  scheduler:
    cron:
      good-morning: ${SCHEDULE_GOOD_MORNING}
      good-night: ${SCHEDULE_GOOD_NIGHT}
      ate-all: ${SCHEDULE_ATE_ALL}
      hungry: ${SCHEDULE_HUNGRY}
```

- `.env`

```
# MR 52 기준
# Server
SERVER_PORT=${SERVER_PORT}
SWAGGER_URI=/
SPRING_APPLICATION_NAME=oily-dev
SERVER_CONTEXT_PATH=/spring

# MySQL
DB_URL=${DB_URL}
DB_USERNAME=${DB_USERNAME}
DB_PASSWORD=${DB_PASSWORD}

# Redis
REDIS_HOST=${REDIS_HOST}
REDIS_PORT=${REDIS_PORT}
REDIS_PASSWORD=${REDIS_PASSWORD}

# Jpa
JPA_HIBERNATE_DDL=update
JPA_FORMAT_SQL=true
JPA_DIALECT=org.hibernate.dialect.MySQLDialect
JPA_SHOW_SQL=true

# Oauth
OAUTH2_GOOGLE_CLIENT_ID=${OAUTH2_GOOGLE_CLIENT_ID}
OAUTH2_GOOGLE_CLIENT_SECRET=${OAUTH2_GOOGLE_CLIENT_SECRET}
OAUTH2_REDIRECT_URI={baseUrl}/v1/login/oauth2/code/{registrationId}
# OAuth2 Custom
SECURITY_OAUTH2_AUTHORIZE_URI=/v1/oauth2/authorization
SECURITY_OAUTH2_REDIRECT_URI=/v1/login/oauth2/code/*
SECURITY_OAUTH2_CLIENT_REDIRECT_URI=https://app.buriburi.monster/callback
```

```
# JWT
JWT_SECRET=${JWT_SECRET}

# MQTT
MQTT_BROKEN_URL=ssl://buriburi.monster:${MQTT_BROKEN_PORT}
MQTT_TOPIC=buriburi/robot/all/command
MQTT_USERNAME=${MQTT_USERNAME}
MQTT_PASSWORD=${MQTT_PASSWORD}

# OpenVidu
OPENVIDU_URL=${OPENVIDU_URL}
OPENVIDU_SECRET=${OPENVIDU_SECRET}
OPENVIDU_SESSION_PREFIX=${OPENVIDU_SESSION_PREFIX}

# AWS S3
AWS_ACCESS_KEY=${AWS_ACCESS_KEY}
AWS_SECRET_KEY=${AWS_SECRET_KEY}
AWS_REGION_STATIC=${AWS_REGION_STATIC}
AWS_S3_BUCKET=${AWS_S3_BUCKET}

# CORS
ALLOWED_ORIGINS=http://localhost:3001,http://localhost:3000,https://buriburi.
monster,https://app.buriburi.monster
ALLOWED_METHODS=GET,POST,PUT,DELETE,PATCH,OPTIONS
ALLOWED_HEADERS=Authorization,Content-Type,Accept
ALLOWED_ALLOW_CREDENTIALS=true
ALLOWED_EXPOSED_HEADERS=Set-Cookie,Authorization
ALLOWED_MAX_AGE=3600

# Security
SING_UP_EXPIRE_TIME=20m
# JWT
SECURITY_JWT_SECRET_KEY=${SECURITY_JWT_SECRET_KEY}
SECURITY_JWT_EXPIRE_TIME_ACCESS=1h
SECURITY_JWT_EXPIRE_TIME_REFRESH=14d
# Whitelist
SECURITY_WHITELIST_GET=/swagger-ui/**,/v3/api-docs/**,/swagger-resource
s/**,/webjars/**,/v1/auth/redirect,/v1/oauth2/authorization/**,/v1/login/oauth2/c
ode/**,/health-check,/v1/media/smile-videos/**,/v1/channels/**,/ws-stomp/**
SECURITY_WHITELIST_POST=/v1/auth/refresh,/v1/channels/**
SECURITY_WHITELIST_PUT=
SECURITY_WHITELIST_DELETE=
SECURITY_WHITELIST_PATCH=
SECURITY_WHITELIST_OPTION=
# CORS
SECURITY_CORS_ALLOWED_ORIGINS=http://localhost:3001,http://localhost:30
00,https://buriburi.monster,https://app.buriburi.monster,https://api.buriburi.mon
ster,http://localhost:8081,http://localhost:8080
```

```
SECURITY_CORS_ALLOWED_METHODS=GET,POST,PUT,DELETE,PATCH,OPTIO
NS
SECURITY_CORS_ALLOWED_HEADERS=Authorization,Content-Type,Accept
SECURITY_CORS_ALLOW_CREDENTIALS=true
SECURITY_CORS_EXPOSED_HEADERS=Set-Cookie,Authorization
SECURITY_CORS_MAX_AGE=3600
# role
SECURITY_ROLE_ADMIN=/v1/admin/**

# Robot Schedule
SCHEDULE_GOOD_MORNING=0 0 8 * * *
SCHEDULE_GOOD_NIGHT=0 0 21 * * *
SCHEDULE_ATE_ALL=0 0 13 * * *
SCHEDULE_HUNGRY=0 0 12 * * *
```

- `backend-fastapi/`
    - `.env.base`

```
# Application
APP_NAME=backend-fastapi
APP_VERSION=v1
APP_DOCS_URL=swagger-ui/index.html
APP_REDOC_URL=redoc/index.html
APP_OPENAPI_URL=openapi.json
APP_ROOT_PATH=/fastapi

# JWT
APP_JWT_SECRET_KEY=${APP_JWT_SECRET_KEY}
APP_JWT_ALGORITHM=${APP_JWT_ALGORITHM}

# GMS
APP_GMS_API_KEY=${YOUR_GMS_KEY}
APP_GMS_API_URL=${YOUR_GMS_URL}

# MySQL
APP_MYSQL_URL=${YOUR_DB_URL}
APP_MYSQL_USERNAME=${YOUR_DB_USERNAME}
APP_MYSQL_PASSWORD=${YOUR_DB_PASSWORD}
APP_MYSQL_DB_NAME=${YOUR_DB_NAME}
APP_MYSQL_PORT=${YOUR_DB_PORT}

# Redis
APP_REDIS_HOST=${YOUR_REDIS_HOST}
APP_REDIS_PORT=${YOUR_REDIS_PORT}
APP_REDIS_PASSWORD=${YOUR_REDIS_PASSWORD}
APP_REDIS_TOPIC_KEY=SUBSCRIBE_TOPIC_LIST

# S3
```

```
APP_S3_ACCESS_KEY=${YOUR_S3_ACCESS_KEY}
APP_S3_SECRET_KEY=${YOUR_S3_SECRET_KEY}
APP_S3_REGION_STATIC=${YOUR_S3_REGION}
APP_S3_BUCKET=${YOUR_S3_BUCKET_NAME}

# OpenVidu
APP_OPENVIDU_URL=${APP_OPENVIDU_URL}
APP_OPENVIDU_SECRET=${APP_OPENVIDU_SECRET}

# MQTT
APP_MQTT_HOST=${APP_MQTT_HOST}
APP_MQTT_PORT=${APP_MQTT_PORT}
APP_MQTT_USERNAME=${APP_MQTT_USERNAME}
APP_MQTT_PASSWORD=${APP_MQTT_PASSWORD}
```

- `.env.prod`

```
# MR 35 기준
APP_NAME=backend-fastapi-production
APP_CORS_ALLOW_ORIGINS=https://k13e108.p.ssafy.io
APP_PROFILE=production
APP_OPENVIDU_SESSION_PREFIX=${APP_OPENVIDU_SESSION_PREFIX}
```

- `frontend-app/`
  - `.env.build`

```
VITE_API_BASE_URL=https://api.buriburi.monster/spring/v1
```

  - `pakcage,json`

```
{
  "name": "frontend-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "vite",
    "start": "serve -s dist -l 3000",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "@lottiefiles/dotlottie-react": "^0.17.6",
    "axios": "^1.13.2",
    "openvidu-browser": "^2.31.0",
    "jwt-decode": "^4.0.0",
    "react": "^19.2.0",
    "react-dom": "^19.2.0",
    "react-router-dom": "^7.9.5",
    "serve": "^14.2.5"
  },
  "devDependencies": {
    "@tailwindcss/postcss": "^4.1.16",
    "@types/node": "^24.10.0",
    "@types/react": "^19.2.2",
    "@types/react-dom": "^19.2.2",
    "@types/tailwindcss": "^3.0.11",
    "@vitejs/plugin-react": "^5.1.0",
    "autoprefixer": "^10.4.21",
    "postcss": "^8.5.6",
    "tailwindcss": "^4.1.16",
    "typescript": "^5.9.3",
    "vite": "^7.1.12",
    "vite-plugin-pwa": "^1.1.0",
    "workbox-window": "^7.3.0"
  }
}
```

- `frontend-dashboard/`
  - `.env.build`

```
#MQTT
VITE_MQTT_BROKER_URL=wss://buriburi.monster:9001
VITE_MQTT_USERNAME=${VITE_MQTT_USERNAME}
VITE_MQTT_PASSWORD=${VITE_MQTT_PASSWORD}

# WEBSOCKET
VITE_WS_URL=https://api.buriburi.monster/spring
```

  - `package.json`

```
{
  "name": "frontend-dashboard",
  "version": "1.0.0",
  "description": "React PWA Dashboard Application",
  "main": "index.js",
  "scripts": {
    "dev": "vite",
    "start": "serve -s dist -l 3001",
    "build": "tsc && vite build",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warning
```

```
s 0",
    "preview": "vite preview",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "react",
    "pwa",
    "dashboard",
    "vite"
  ],
  "author": "",
  "license": "ISC",
  "type": "module",
  "dependencies": {
    "@react-three/drei": "^9.122.0",
    "@react-three/fiber": "^8.18.0",
    "@react-three/postprocessing": "^2.19.1",
    "@stomp/stompjs": "^7.2.1",
    "mqtt": "^5.14.1",
    "openvidu-browser": "^2.31.0",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "serve": "^14.2.5",
    "sockjs-client": "^1.6.1",
    "three": "^0.164.1"
  },
  "devDependencies": {
    "@tailwindcss/postcss": "^4.1.16",
    "@types/react": "^18.3.12",
    "@types/react-dom": "^18.3.1",
    "@types/sockjs-client": "^1.5.4",
    "@types/tailwindcss": "^3.0.11",
    "@types/three": "^0.180.0",
    "@typescript-eslint/eslint-plugin": "^6.14.0",
    "@typescript-eslint/parser": "^6.14.0",
    "@vitejs/plugin-react": "^4.3.3",
    "autoprefixer": "^10.4.21",
    "eslint": "^8.55.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.5",
    "postcss": "^8.5.6",
    "tailwindcss": "^4.1.16",
    "typescript": "^5.6.3",
    "vite": "^6.0.1",
    "vite-plugin-pwa": "^0.21.1",
    "workbox-window": "^7.2.0"
  }
}
```

(7) 빌드 및 배포

- 배포는 Jenkins 파이프라인을 통해 자동화되어 있습니다. GitLab `develop` 브랜치에 MR 또는 Push가 발생하면 각 서비스별 Jenkins Job이 실행되어 자동으로 빌드 및 배포가 진행됩니다.
- `develop-main.groovy`

```
pipeline {
    agent any

    environment {
        // --- Git Repository Settings ---
        GIT_URL           = "https://lab.ssafy.com/s13-final/S13P31E108.git"
        GIT_CREDENTIAL_ID = "gitlab_token_username_with_password"  // 젠킨스에 등록한
Credential ID

        // --- Downstream Job Names ---
        BACKEND_SPRING_JOB_NAME      = "dev-backend-spring"
        BACKEND_FASTAPI_JOB_NAME     = "dev-backend-fastapi"
        FRONTEND_DASHBOARD_JOB_NAME  = "dev-frontend-dashboard"
        FRONTEND_APP_JOB_NAME        = "dev-frontend-app"

        // --- Internal Flags ---
        //DEPLOY_BACKEND    = 'false'
        //DEPLOY_FRONTEND   = 'false'
    }

    // 수동 실행 시 브랜치 이름을 입력받을 UI를 생성
    parameters {
        string(name: 'BRANCH_TO_BUILD', defaultValue: 'develop', description: 'Enter the
branch name to build (e.g., develop, master)')
    }

    stages {
        stage('Initialize & Check Changes') {
            steps {
                // 플래그 변수 초기화
                script {
                    env.DEPLOY_BACKEND_SPRING      = 'false'
                    env.DEPLOY_BACKEND_FASTAPI     = 'false'
                    env.DEPLOY_FRONTEND_DASHBOARD  = 'false'
                    env.DEPLOY_FRONTEND_APP        = 'false'

                    // 자동 실행(gitlabBranch)과 수동 실행(BRANCH_TO_BUILD)을 모두 고려하여 현
재 브랜치 결정
                    def currentBranch = env.gitlabBranch ?: params.BRANCH_TO_BUILD
                    echo env.gitlabBranch ? "🚀 Build triggered by GitLab webhook on branch
```

```
'${currentBranch}'"
                    : "👤 Build triggered manually for branch '${currentBranch}'"

                git url: env.GIT_URL,
                    branch: currentBranch,
                    credentialsId: env.GIT_CREDENTIAL_ID

                def changedFiles = ''
                if (env.gitlabBefore && env.gitlabAfter) {
                    // Webhook 실행 시, 정확한 commit hash로 비교
                    echo "Webhook detected. Comparing commits ${env.gitlabBefore}..${en
v.gitlabAfter}"
                    changedFiles = sh(script: "git diff --name-only ${env.gitlabBefore} ${en
v.gitlabAfter}", returnStdout: true).trim()
                } else {
                    // 수동 실행 시, 이전 방식으로 비교
                    echo "Manual build detected. Comparing HEAD~1..HEAD"
                    sh 'git fetch --unshallow || true'
                    changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD", returnStd
out: true).trim()
                }

                echo "Detected changed files:\n${changedFiles}"

                if (currentBranch != 'develop' && currentBranch != 'master') {
                    error "This test will only run on 'develop' or 'master' branches."
                }

                def changedList = changedFiles.split('\\r?\\n').findAll { it?.trim() }

                // 각 디렉터리별로 변경 사항 감지
                def backendSpringChanged       = changedList.any { it.startsWith('backend
-spring/') }
                def backendFastapiChanged      = changedList.any { it.startsWith('backend
-fastapi/') }
                def frontendDashBoardChanged   = changedList.any { it.startsWith('fronte
nd-dashboard/') }
                def frontendAppChanged         = changedList.any { it.startsWith('frontend-
app/') }

                // 감지된 변경 사항에 따라 플래그 설정
                if (backendSpringChanged)   env.DEPLOY_BACKEND_SPRING = 'true'
                if (backendFastapiChanged)  env.DEPLOY_BACKEND_FASTAPI = 'true'
                if (frontendDashBoardChanged)   env.DEPLOY_FRONTEND_DASHBOARD =
'true'
                if (frontendAppChanged)     env.DEPLOY_FRONTEND_APP = 'true'

                echo "Backend Spring deployment needed: ${env.DEPLOY_BACKEND_SPRI
```

```
NG}"
                echo "Backend FastAPI deployment needed: ${env.DEPLOY_BACKEND_FA
STAPI}"
                echo "Frontend DashBoard deployment needed: ${env.DEPLOY_FRONTEN
D_DASHBOARD}"
                echo "Frontend App deployment needed: ${env.DEPLOY_FRONTEND_AP
P}"

                env.BUILD_TITLE_PARAM        = env.gitlabMergeRequestTitle ?: env.gitlabC
ommitTitle ?: "Commit on ${currentBranch}"
                env.BUILD_URL_PARAM          = env.gitlabMergeRequestUrl ?: "" // Push의
경우 URL이 없을 수 있음
                env.BUILD_AUTHOR_PARAM       = env.gitlabUserName ?: "Unknown"
                env.BUILD_TARGET_BRANCH_PARAM = env.gitlabMergeRequestTargetBran
ch ?: currentBranch
                env.CHANGED_FILES_LIST_PARAM  = changedFiles // 위에서 계산한 변경 파일
목록
            }
        }
    }

    stage('Trigger Deployments') {
        // 4개의 플래그 중 하나라도 true이면 스테이지 실행
        when {
            expression {
                env.DEPLOY_BACKEND_SPRING == 'true' ||
                env.DEPLOY_BACKEND_FASTAPI == 'true' ||
                env.DEPLOY_FRONTEND_DASHBOARD == 'true' ||
                env.DEPLOY_FRONTEND_APP == 'true'
            }
        }
        steps {
            script {
                def branchToBuild = env.gitlabBranch ?: params.BRANCH_TO_BUILD

                // 4개의 잡을 병렬로 실행
                parallel (
                    'backend-spring': {
                        if (env.DEPLOY_BACKEND_SPRING == 'true') {
                            echo "Triggering Backend Spring build..."
                            build job: env.BACKEND_SPRING_JOB_NAME,
                                parameters: [
                                    string(name: 'BRANCH', value: branchToBuild),
                                    string(name: 'BUILD_TITLE', value: env.BUILD_TITLE_PARAM),
                                    string(name: 'BUILD_URL', value: env.BUILD_URL_PARAM),
                                    string(name: 'BUILD_AUTHOR', value: env.BUILD_AUTHOR_PAR
AM),
                                    string(name: 'BUILD_TARGET_BRANCH', value: env.BUILD_TAR
GET_BRANCH_PARAM),
                    string(name: 'CHANGED_FILES_LIST', value: env.CHANGED_FIL
ES_LIST_PARAM)
                  ],
                  wait: true
              }
            },
            'backend-fastapi': {
              if (env.DEPLOY_BACKEND_FASTAPI == 'true') {
                echo "Triggering Backend FastAPI build..."
                build job: env.BACKEND_FASTAPI_JOB_NAME,
                  parameters: [
                    string(name: 'BRANCH', value: branchToBuild),
                    string(name: 'BUILD_TITLE', value: env.BUILD_TITLE_PARAM),
                    string(name: 'BUILD_URL', value: env.BUILD_URL_PARAM),
                    string(name: 'BUILD_AUTHOR', value: env.BUILD_AUTHOR_PAR
AM),
                    string(name: 'BUILD_TARGET_BRANCH', value: env.BUILD_TAR
GET_BRANCH_PARAM),
                    string(name: 'CHANGED_FILES_LIST', value: env.CHANGED_FIL
ES_LIST_PARAM)
                  ],
                  wait: true
              }
            },
            'frontend-dashboard': {
              if (env.DEPLOY_FRONTEND_DASHBOARD == 'true') {
                echo "Triggering Frontend DashBoard build..."
                build job: env.FRONTEND_DASHBOARD_JOB_NAME,
                  parameters: [
                    string(name: 'BRANCH', value: branchToBuild),
                    string(name: 'BUILD_TITLE', value: env.BUILD_TITLE_PARAM),
                    string(name: 'BUILD_URL', value: env.BUILD_URL_PARAM),
                    string(name: 'BUILD_AUTHOR', value: env.BUILD_AUTHOR_PAR
AM),
                    string(name: 'BUILD_TARGET_BRANCH', value: env.BUILD_TAR
GET_BRANCH_PARAM),
                    string(name: 'CHANGED_FILES_LIST', value: env.CHANGED_FIL
ES_LIST_PARAM)
                  ],
                  wait: true
              }
            },
            'frontend-app': {
              if (env.DEPLOY_FRONTEND_APP == 'true') {
                echo "Triggering Frontend App build..."
                build job: env.FRONTEND_APP_JOB_NAME,
                  parameters: [
                                        string(name: 'BRANCH', value: branchToBuild),
                                        string(name: 'BUILD_TITLE', value: env.BUILD_TITLE_PARAM),
                                        string(name: 'BUILD_URL', value: env.BUILD_URL_PARAM),
                                        string(name: 'BUILD_AUTHOR', value: env.BUILD_AUTHOR_PAR
AM),
                                        string(name: 'BUILD_TARGET_BRANCH', value: env.BUILD_TAR
GET_BRANCH_PARAM),
                                        string(name: 'CHANGED_FILES_LIST', value: env.CHANGED_FIL
ES_LIST_PARAM)
                                    ],
                                    wait: true
                            }
                        }
                    )
                }
            }
        }
    }
    post {
        always {
            echo 'Main dispatcher pipeline finished.'
        }
    }
}
```

- `develop-backend-fastapi.groovy`

```
import groovy.json.JsonOutput

final def MM_USER_MAP = [
    "dahxtq1": "dahxtq1",
    "me_in_u": "me_in_u",
    "dhnn1536": "dhnn1536",
    "phangmin03": "phangmin03",
    "ryongseong.dev": "ryongseong.dev",
    "doriconi": "doriconi"
]

pipeline {
  agent any

  environment {
    // ===== Git =====
    GIT_URL           = "https://lab.ssafy.com/s13-final/S13P31E108.git"
    GIT_CREDENTIAL_ID = "gitlab_token_username_with_password"
    BRANCH            = "${params.BRANCH ?: 'develop'}"

    // ===== App / Docker =====
    IMAGE_NAME           = "dev-backend-fastapi"
    IMAGE_TAG            = "latest"
    DOCKER_CONTAINER_NAME = "dev-backend-fastapi"
    NETWORK_NAME         = "devnet"

    // ===== Ports (Dockerfile.production 기준) =====
    INTERNAL_PORT = "8081" // Dockerfile의 EXPOSE 포트

    // ===== Python / Project =====
    PROJECT_DIR = "backend-fastapi"

    // ===== Secrets (Credentials IDs) =====
    ENV_FILE_BASE_CRED_ID = "dev_env_fastapi_base"
    ENV_FILE_PROD_CRED_ID = "dev_env_fastapi_prod"

    // ===== Mattermost =====
    MATTERMOST_ENDPOINT = "https://meeting.ssafy.com/hooks/w74iy4erapnzixx37o
uiawi9ye"
    MATTERMOST_CHANNEL  = "e108-release-notification"

    // ===== Health Check =====
    HEALTH_CHECK_URL = "https://api.buriburi.monster/fastapi/v1/health"
    SWAGGER_URL = "https://api.buriburi.monster/fastapi/swagger-ui/index.html"

    SKIP_BUILD = "false"
  }

  // 메인 잡에서 MR/Commit 정보를 받기 위한 파라미터
  parameters {
    string(name: 'BRANCH', defaultValue: 'develop', description: '빌드할 브랜치')
    string(name: 'BUILD_TITLE', defaultValue: '', description: 'MR/Commit Title from main
job')
    string(name: 'BUILD_URL', defaultValue: '', description: 'MR/Commit URL from main jo
b')
    string(name: 'BUILD_AUTHOR', defaultValue: 'Unknown', description: 'MR/Commit A
uthor from main job')
    string(name: 'BUILD_TARGET_BRANCH', defaultValue: 'develop', description: 'MR/Co
mmit Target Branch from main job')
    string(name: 'CHANGED_FILES_LIST', defaultValue: '', description: 'List of changed fi
les from main job')
  }

  stages {
    stage('Checkout') {
      steps {
        cleanWs()
        echo "WORKSPACE: ${env.WORKSPACE}"
        sh 'pwd && ls -al'
        git url: env.GIT_URL, branch: env.BRANCH, credentialsId: env.GIT_CREDENTIAL_ID
        sh '''
          echo "== After git clone =="; pwd && ls -al
          echo "== ${PROJECT_DIR} (backend-fastapi) listing =="
          [ -d ${PROJECT_DIR} ] && ls -al ${PROJECT_DIR} || echo "${PROJECT_DIR} not f
ound"
        '''
      }
    }

    stage('Detect Changes in backend-fastapi/') {
      steps {
        script {
          sh 'git fetch --unshallow || true'
          def changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", retu
rnStdout: true).trim()
          echo "Changed files:\n${changedFiles}"

          def finalChangedFiles = params.CHANGED_FILES_LIST
          if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
              echo "No changed files list from main job, using manual diff."
              finalChangedFiles = changedFiles
          } else {
              echo "Using changed files list from main job."
          }

          if (!finalChangedFiles.contains("${PROJECT_DIR}/")) {
              echo "No changes found in ${PROJECT_DIR}. Setting SKIP_BUILD to true."
              env.SKIP_BUILD = "true"
          }

          if (env.SKIP_BUILD == "true") echo "⏭ ${PROJECT_DIR}/ 변경 없음 → 빌드/배포 스
킵"
        }
      }
    }

    stage('Place .env file') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        withCredentials([
          file(credentialsId: "${ENV_FILE_BASE_CRED_ID}", variable: 'ENV_BASE_TMP'),
          file(credentialsId: "${ENV_FILE_PROD_CRED_ID}", variable: 'ENV_PROD_TMP')
        ]) {
          sh '''
            set -e
            mkdir -p ${PROJECT_DIR}
        cp "$ENV_BASE_TMP" ${PROJECT_DIR}/.env.base
        cp "$ENV_PROD_TMP" ${PROJECT_DIR}/.env.prod

        chmod 600 ${PROJECT_DIR}/.env.base || true
        chmod 600 ${PROJECT_DIR}/.env.prod || true

        echo "== placed .env file for deployment =="
        ls -al ${PROJECT_DIR}/.env.base || true
        ls -al ${PROJECT_DIR}/.env.prod || true
       '''
      }
    }
  }

  stage('Docker Build') {
    when { environment name: 'SKIP_BUILD', value: 'false' }
    steps {
      dir(env.PROJECT_DIR) {
       sh '''
        echo "== Docker build context =="; pwd && ls -al
        echo "== Dockerfile.production 내용 (상위 40줄) =="; sed -n '1,40p' Dockerfile.pr
oduction || true
        echo "== docker build (using Dockerfile.production) =="
        docker build -f Dockerfile.production -t ${IMAGE_NAME}:${IMAGE_TAG} .

        echo "== built images =="; docker images | grep ${IMAGE_NAME} || true
       '''
      }
    }
  }

  stage('Ensure Docker Network') {
    when { environment name: 'SKIP_BUILD', value: 'false' }
    steps {
      sh '''
       docker network inspect ${NETWORK_NAME} >/dev/null 2>&1 || docker network
create ${NETWORK_NAME}
       echo "== networks =="; docker network ls | grep ${NETWORK_NAME} || true
      '''
    }
  }

  // 이 스테이지의 docker run 명령어 맨 끝에 덮어쓰기 명령 추가
  stage('Deploy Container') {
    when { environment name: 'SKIP_BUILD', value: 'false' }
    steps {
      sh '''
       set -e
```

```
          echo "== stop & remove old container =="; docker stop ${DOCKER_CONTAINER_
NAME} || true; docker rm ${DOCKER_CONTAINER_NAME} || true

          echo "== run new container =="
          docker run -d \
            --name ${DOCKER_CONTAINER_NAME} \
            --network ${NETWORK_NAME} \
            --network-alias ${DOCKER_CONTAINER_NAME} \
            --restart unless-stopped \
            --env-file ${WORKSPACE}/${PROJECT_DIR}/.env.base \
            --env-file ${WORKSPACE}/${PROJECT_DIR}/.env.prod \
            -e TZ=Asia/Seoul \
            -v /home/ubuntu/logs/fastapi:/app/logs \
            ${IMAGE_NAME}:${IMAGE_TAG} \
            python main.py

          echo "== 추가 네트워크 연결: edge =="
          docker network connect edge ${DOCKER_CONTAINER_NAME} || true

          echo "== container inspect (name/networks) =="
          docker inspect --format 'Name: {{.Name}}, Networks: {{range $k, $v := .Network
Settings.Networks}}{{$k}}(IP: {{$v.IPAddress}}) {{end}}' ${DOCKER_CONTAINER_NAM
E}
          echo "== port mapping (Nginx 프록시를 사용하므로 호스트 포트 없음) =="
          docker ps --filter name=${DOCKER_CONTAINER_NAME} --format "table {{.Name
s}}\t{{.Ports}}\t{{.Status}}"
        '''
      }
    }

    stage('Health Check') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        script {
          try {
            // 총 2분(120초)의 타임아웃 설정
            timeout(time: 2, unit: 'MINUTES') {
              boolean success = false

              // 성공할 때까지 반복하는 while 루프
              while (!success) {
                try {
                  echo "Attempting health check on ${env.HEALTH_CHECK_URL}..."

                  // -f: 4xx/5xx 에러 시 실패
                  // -L: 리다이렉션 따르기
                  // --max-time 10: 10초 타임아웃
                  sh "curl -fL --max-time 10 ${env.HEALTH_CHECK_URL}"
```

```
                              // 위 curl이 성공하면(오류 코드가 없으면)
                              // success가 true가 되고 루프가 종료됨
                              success = true
                              echo "✅ Health check passed!"

                          } catch (Exception e) {
                              // curl이 실패하면(앱이 아직 안 떴으면) catch로 빠짐
                              echo "Health check failed (server not ready?). Retrying in 10 second
s..."

                              // 10초 대기 후 while 루프의 처음으로 돌아가 다시 시도
                              sleep(10)
                          }
                      }
                  }

              } catch (org.jenkinsci.plugins.workflow.steps.FlowInterruptedException e) {
                  // timeout(2) 만료 시
                  echo "Health check timed out after 2 minutes."
                  // error 스텝을 호출하여 빌드를 'FAILURE'로 강제 변경
                  error('Health check timed out and failed.')
              }
          }
      }
    }
  }

  post {
    always {
      sh '''
        echo "== workspace listing =="; pwd && ls -al
        echo "== ${PROJECT_DIR} listing =="; [ -d ${PROJECT_DIR} ] && ls -al ${PROJEC
T_DIR} || true
        if [ "${SKIP_BUILD}" = "false" ]; then
          echo "== Pruning docker images =="
          docker image prune -f || true
        fi
      '''
    }

    success {
      script {
        def finalChangedFiles = params.CHANGED_FILES_LIST
        if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
            echo "No changed files passed from main job. Running manual diff as fallback."
            finalChangedFiles = sh(
                script: "git diff --name-only HEAD~1 HEAD || echo ''",
            returnStdout: true
          ).trim()
        }

        def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AUTH
OR

        if (env.SKIP_BUILD == "false") {
          // --- 1. 빌드 및 배포 성공 ---
          echo '🎉 FastAPI 배포 및 헬스체크가 성공적으로 완료되었습니다!'

          def finalMessage = ""

          if (params.BUILD_URL && params.BUILD_URL != "") {
            finalMessage = """
#### :shinchan_dance: Jenkins Pipeline Success :shinchan_dance:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏼 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기**

[:fastapi: FastAPI **Dev Server**](${env.SWAGGER_URL})
"""
          } else {
            finalMessage = """
#### :shinchan_dance2: Jenkins Pipeline Success :shinchan_dance2:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`
📝 **Changed Files**:

```
${finalChangedFiles}
```
💬 **수동 빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기** :shin_hyeong_man:

- [:fastapi: FastAPI **Dev Server**](${env.SWAGGER_URL})
"""
```

```
        }

        mattermostSend(
          // ❗ [수정] endpoint를 명시적으로 전달
          endpoint: "${MATTERMOST_ENDPOINT}",
          channel: "${MATTERMOST_CHANNEL}",
          color: '#36a64f', // 초록색
          message: finalMessage.stripIndent()
        )

      } else {
        // --- 2. 빌드 스킵 ---
        echo "⏭️ ${PROJECT_DIR} 디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다."

        def finalMessage = ""

        if (params.BUILD_URL && params.BUILD_URL != "") {
          finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏼 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
        } else {
          finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

💬 **디렉토리 변경사항이 없어서 수동 빌드를 건너뛰었습니다.**
"""
        }

        mattermostSend(
          // ❗ [수정] endpoint를 명시적으로 전달
          endpoint: "${MATTERMOST_ENDPOINT}",
          channel: "${MATTERMOST_CHANNEL}",
          color: '#ffaa00', // 주황색
          message: finalMessage.stripIndent()
        )
      }
    }
    }

    failure {
        script {
            def logOutput = ""
            try {
                def logLines = currentBuild.rawBuild.getLog(100)
                logOutput = logLines.join('\n')
                                     .replaceAll('\\$', '\\\\\\$')
                                     .replaceAll('`', '\\\\`')
            } catch (Exception e) {
                logOutput = "Could not retrieve build log: ${e.message}"
            }

            def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AU
THOR

            def finalMessage = ""
            def failedStage = env.STAGE_NAME ?: "알 수 없음"

            if (params.BUILD_URL && params.BUILD_URL != "") {
                finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:

:alert_siren: **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏼 **Author**: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

##### 📝 Error Log (Failed Stage: ${failedStage})
```
${logOutput}
```
**전체 로그 확인하기**: ${env.BUILD_URL}console
"""
            } else {
                finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:
:alert_siren: **DEV FastAPI 수동 배포 실패!**

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

##### 📝 Error Log (Failed Stage: ${failedStage})
```
${logOutput}
```
```

```
"""
          }

          mattermostSend(
             endpoint: "${MATTERMOST_ENDPOINT}",
             channel: "${MATTERMOST_CHANNEL}",
             color: '#ff0000',  // 빨간색
             message: finalMessage.stripIndent()
          )
       }
    }
  }
}
```

- `develop-backend-spring.groovy`

```
import groovy.json.JsonOutput

// ❗ GitLab ID를 Mattermost ID로 매핑 (FastAPI와 동일)
final def MM_USER_MAP = [
   "dahxtq1": "dahxtq1",
   "me_in_u": "me_in_u",
   "dhnn1536": "dhnn1536",
   "phangmin03": "phangmin03",
   "ryongseong.dev": "ryongseong.dev",
   "doriconi": "doriconi"
]

pipeline {
  agent any

  environment {
    // ===== Git (E108 프로젝트) =====
    GIT_URL          = "https://lab.ssafy.com/s13-final/S13P31E108.git"
    GIT_CREDENTIAL_ID = "gitlab_token_username_with_password"
    BRANCH           = "${params.BRANCH ?: 'develop'}"

    // ===== App / Docker (Spring) =====
    IMAGE_NAME           = "dev-backend-spring" // 빌드할 이미지 이름
    IMAGE_TAG            = "latest"
    // Nginx가 바라보는 컨테이너 이름 (dev-backend)
    DOCKER_CONTAINER_NAME = "dev-backend-spring"
    NETWORK_NAME         = "devnet"

    // ===== Ports (Dockerfile 기준) =====
    INTERNAL_PORT = "8080" // Dockerfile의 EXPOSE 포트
    // ===== Java / Project (Spring) =====
    PROJECT_DIR = "backend-spring"

    // ===== Secrets (Credentials IDs) =====
    // Spring용 .env 파일 Credential ID
    ENV_FILE_CRED_ID = "dev_env_spring"

    // ===== Mattermost (FastAPI와 동일) =====
    MATTERMOST_ENDPOINT = "https://meeting.ssafy.com/hooks/w74iy4erapnzixx37o
uiawi9ye"
    MATTERMOST_CHANNEL  = "e108-release-notification"

    // ===== Health Check (Spring) =====
    // Nginx 경로(/spring/)와 Spring 내부 경로(/api/v1/health) 조합
    HEALTH_CHECK_URL = "https://api.buriburi.monster/spring/health-check"
    SWAGGER_URL = "https://api.buriburi.monster/spring/swagger-ui/index.html"

    SKIP_BUILD = "false"
  }

  // 메인 잡에서 MR/Commit 정보를 받기 위한 파라미터 (FastAPI와 동일)
  parameters {
    string(name: 'BRANCH', defaultValue: 'develop', description: '빌드할 브랜치')
    string(name: 'BUILD_TITLE', defaultValue: '', description: 'MR/Commit Title from main
job')
    string(name: 'BUILD_URL', defaultValue: '', description: 'MR/Commit URL from main jo
b')
    string(name: 'BUILD_AUTHOR', defaultValue: 'Unknown', description: 'MR/Commit A
uthor from main job')
    string(name: 'BUILD_TARGET_BRANCH', defaultValue: 'develop', description: 'MR/Co
mmit Target Branch from main job')
    string(name: 'CHANGED_FILES_LIST', defaultValue: '', description: 'List of changed fi
les from main job')
  }

  stages {
    stage('Checkout') {
      steps {
        cleanWs()
        echo "WORKSPACE: ${env.WORKSPACE}"
        sh 'pwd && ls -al'
        git url: env.GIT_URL, branch: env.BRANCH, credentialsId: env.GIT_CREDENTIAL_ID
        sh '''
          echo "== After git clone =="; pwd && ls -al
          echo "== ${PROJECT_DIR} (backend-spring) listing =="
          [ -d ${PROJECT_DIR} ] && ls -al ${PROJECT_DIR} || echo "${PROJECT_DIR} not f
ound"
        '''
      }
    }

    stage('Detect Changes in backend-spring/') {
      steps {
        script {
          sh 'git fetch --unshallow || true'
          def changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", retu
rnStdout: true).trim()
          echo "Changed files:\n${changedFiles}"

          def finalChangedFiles = params.CHANGED_FILES_LIST
          if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
            echo "No changed files list from main job, using manual diff."
            finalChangedFiles = changedFiles
          } else {
            echo "Using changed files list from main job."
          }

          // ❗ Spring 프로젝트 디렉터리 감지
          if (!finalChangedFiles.contains("${PROJECT_DIR}/")) {
            echo "No changes found in ${PROJECT_DIR}. Setting SKIP_BUILD to true."
            env.SKIP_BUILD = "true"
          }

          if (env.SKIP_BUILD == "true") echo "⏭ ${PROJECT_DIR}/ 변경 없음 → 빌드/배포 스
킵"
        }
      }
    }

    stage('Place .env file') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        withCredentials([
          // ❗ Spring용 .env 파일 1개만 사용
          file(credentialsId: "${ENV_FILE_CRED_ID}", variable: 'ENV_TMP')
        ]) {
          sh '''
            set -e
            mkdir -p ${PROJECT_DIR}

            cp "$ENV_TMP" ${PROJECT_DIR}/.env
            chmod 600 ${PROJECT_DIR}/.env || true

            echo "== placed .env file for deployment =="
            ls -al ${PROJECT_DIR}/.env || true
          '''
        }
      }
    }

    // ❗ [수정] Java/Gradle 설치 대신 Dockerfile을 이용한 빌드
    stage('Docker Build (Multi-stage)') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        dir(env.PROJECT_DIR) { // Spring 프로젝트 디렉터리로 이동
          sh '''
            echo "== Docker build context =="; pwd && ls -al
            echo "== Dockerfile 내용 (상위 40줄) =="; sed -n '1,40p' Dockerfile || true
            echo "== docker build (using Dockerfile) =="
            # ❗ Dockerfile의 ARG SKIP_TESTS=true를 활용하여 테스트 스킵
            docker build --build-arg SKIP_TESTS=true -t ${IMAGE_NAME}:${IMAGE_TAG} .

            echo "== built images =="; docker images | grep ${IMAGE_NAME} || true
          '''
        }
      }
    }

    stage('Ensure Docker Network') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          docker network inspect ${NETWORK_NAME} >/dev/null 2>&1 || docker network
create ${NETWORK_NAME}
          echo "== networks =="; docker network ls | grep ${NETWORK_NAME} || true
        '''
      }
    }

    stage('Deploy Container') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          set -e
          echo "== stop & remove old container =="; docker stop ${DOCKER_CONTAINER_
NAME} || true; docker rm ${DOCKER_CONTAINER_NAME} || true

          echo "== run new container =="
          docker run -d \
            --name ${DOCKER_CONTAINER_NAME} \
            --network ${NETWORK_NAME} \
            --network-alias ${DOCKER_CONTAINER_NAME} \
            --restart unless-stopped \
            --env-file ${WORKSPACE}/${PROJECT_DIR}/.env \
          -e TZ=Asia/Seoul \
          -e SPRING_PROFILES_ACTIVE=dev \
          -v /home/ubuntu/logs/spring:/app/logs \
          ${IMAGE_NAME}:${IMAGE_TAG}

         echo "== 추가 네트워크 연결: edge =="
         docker network connect edge ${DOCKER_CONTAINER_NAME} || true

         echo "== container inspect (name/networks) =="
         docker inspect --format 'Name: {{.Name}}, Networks: {{range $k, $v := .Network
Settings.Networks}}{{$k}}(IP: {{$v.IPAddress}}) {{end}}' ${DOCKER_CONTAINER_NAM
E}
         echo "== port mapping (Nginx 프록시를 사용하므로 호스트 포트 없음) =="
         docker ps --filter name=${DOCKER_CONTAINER_NAME} --format "table {{.Name
s}}\t{{.Ports}}\t{{.Status}}"
        '''
      }
    }

    stage('Health Check') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        script {
          try {
            // 총 2분(120초)의 타임아웃 설정
            timeout(time: 2, unit: 'MINUTES') {
              boolean success = false

              // 성공할 때까지 반복하는 while 루프
              while (!success) {
                try {
                  echo "Attempting health check on ${env.HEALTH_CHECK_URL}..."

                  // -f: 4xx/5xx 에러 시 실패
                  // -L: 리다이렉션 따르기
                  // --max-time 10: 10초 타임아웃
                  sh "curl -fL --max-time 10 ${env.HEALTH_CHECK_URL}"

                  // 위 curl이 성공하면(오류 코드가 없으면)
                  // success가 true가 되고 루프가 종료됨
                  success = true
                  echo "✅ Health check passed!"

                } catch (Exception e) {
                  // curl이 실패하면(앱이 아직 안 떴으면) catch로 빠짐
                  echo "Health check failed (server not ready?). Retrying in 10 second
s..."
```

```
                // 10초 대기 후 while 루프의 처음으로 돌아가 다시 시도
                sleep(10)
              }
            }
          }

        } catch (org.jenkinsci.plugins.workflow.steps.FlowInterruptedException e) {
          // timeout(2) 만료 시
          echo "Health check timed out after 2 minutes."
          // error 스텝을 호출하여 빌드를 'FAILURE'로 강제 변경
          error('Health check timed out and failed.')
        }
      }
    }
  }
}

post {
  always {
    sh '''
      echo "== workspace listing =="; pwd && ls -al
      echo "== ${PROJECT_DIR} listing =="; [ -d ${PROJECT_DIR} ] && ls -al ${PROJEC
T_DIR} || true
      if [ "${SKIP_BUILD}" = "false" ]; then
        echo "== Pruning docker images =="
        docker image prune -f || true
      fi
    '''
  }

  success {
    script {
      def finalChangedFiles = params.CHANGED_FILES_LIST
      if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
          finalChangedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", re
turnStdout: true).trim()
      }
      def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AUTH
OR

      if (env.SKIP_BUILD == "false") {
          echo '🎉 Spring 배포 및 헬스체크가 성공적으로 완료되었습니다!'
          def finalMessage = ""
          if (params.BUILD_URL && params.BUILD_URL != "") {
              finalMessage = """
#### :shinchan_dance: Jenkins Pipeline Success :shinchan_dance:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**
```

```
✍🏻 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기**

- [:springboot: Spring **Dev Server**](${env.SWAGGER_URL})
"""
            } else {
                finalMessage = """
#### :shinchan_dance2: Jenkins Pipeline Success :shinchan_dance2:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`
📝 **Changed Files**:
```
${finalChangedFiles}
```
💬 **수동 빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기** :shin_hyeong_man:

- [:springboot: Spring **Dev Server**](${env.SWAGGER_URL})
"""
            }
            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#36a64f',
                message: finalMessage.stripIndent()
            )
        } else {
            echo "⏭️ ${PROJECT_DIR} 디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다."
            def finalMessage = ""
            if (params.BUILD_URL && params.BUILD_URL != "") {
                finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏻 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`
```

```
💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
            } else {
                finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

💬 **디렉토리 변경사항이 없어서 수동 빌드를 건너뛰었습니다.**
"""
            }
            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#ffaa00',
                message: finalMessage.stripIndent()
            )
        }
      }
    }

    failure {
        script {
            def logOutput = ""
            try {
                // Java 스택 트레이스를 위해 로그 150줄로 늘림
                def logLines = currentBuild.rawBuild.getLog(150)
                logOutput = logLines.join('\n')
                                    .replaceAll('\\$', '\\\\\\$')
                                    .replaceAll('`', '\\\\`')
            } catch (Exception e) {
                logOutput = "Could not retrieve build log: ${e.message}"
            }
            def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AU
THOR
            def finalMessage = ""
            def failedStage = env.STAGE_NAME ?: "알 수 없음"

            if (params.BUILD_URL && params.BUILD_URL != "") {
                finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:

:alert_siren: **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏻 **Author**: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`
```

```
##### 📝 Error Log (Failed Stage: ${failedStage})
```
${logOutput}
```
**전체 로그 확인하기**: ${env.BUILD_URL}console
"""
            } else {
                finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:
:alert_siren: **DEV Spring 수동 배포 실패!**

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

##### 📝 Error Log (Failed Stage: ${failedStage})
```
${logOutput}
```
"""
            }

            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#ff0000',
                message: finalMessage.stripIndent()
            )
        }
    }
  }
}
```

- `develop-frontend-app.groovy`

```
import groovy.json.JsonOutput

final def MM_USER_MAP = [
    "dahxtq1": "dahxtq1",
    "me_in_u": "me_in_u",
    "dhnn1536": "dhnn1536",
    "phangmin03": "phangmin03",
    "ryongseong.dev": "ryongseong.dev",
    "doriconi": "doriconi"
]

pipeline {
  agent any

  environment {
    GIT_URL          = "https://lab.ssafy.com/s13-final/S13P31E108.git"
    GIT_CREDENTIAL_ID = "gitlab_token_username_with_password"
    BRANCH           = "${params.BRANCH ?: 'develop'}"

    IMAGE_NAME           = "dev-frontend-app"
    IMAGE_TAG            = "latest"

    DOCKER_CONTAINER_NAME = "dev-frontend-app"
    NETWORK_NAME         = "devnet"

    INTERNAL_PORT = "3000"

    PROJECT_DIR = "frontend-app"

    ENV_FILE_CRED_ID = "dev_env_frontend_app"

    MATTERMOST_ENDPOINT = "https://meeting.ssafy.com/hooks/w74iy4erapnzixx37o
uiawi9ye"
    MATTERMOST_CHANNEL  = "e108-release-notification"

    HEALTH_CHECK_URL = "https://app.buriburi.monster"
    SERVICE_URL = "https://app.buriburi.monster"

    SKIP_BUILD = "false"
  }

  parameters {
    string(name: 'BRANCH', defaultValue: 'develop', description: '빌드할 브랜치')
    string(name: 'BUILD_TITLE', defaultValue: '', description: 'MR/Commit Title from main
job')
    string(name: 'BUILD_URL', defaultValue: '', description: 'MR/Commit URL from main jo
b')
    string(name: 'BUILD_AUTHOR', defaultValue: 'Unknown', description: 'MR/Commit A
uthor from main job')
    string(name: 'BUILD_TARGET_BRANCH', defaultValue: 'develop', description: 'MR/Co
mmit Target Branch from main job')
    string(name: 'CHANGED_FILES_LIST', defaultValue: '', description: 'List of changed fi
les from main job')
  }

  stages {
    stage('Checkout') {
      steps {
        cleanWs()
        echo "WORKSPACE: ${env.WORKSPACE}"
        sh 'pwd && ls -al'
        git url: env.GIT_URL, branch: env.BRANCH, credentialsId: env.GIT_CREDENTIAL_ID
        sh '''
          echo "== After git clone =="; pwd && ls -al
          echo "== ${PROJECT_DIR} (frontend-app) listing =="
          [ -d ${PROJECT_DIR} ] && ls -al ${PROJECT_DIR} || echo "${PROJECT_DIR} not f
ound"
        '''
      }
    }
    stage('Detect Changes in frontend-app/') {
        steps {
            script {
                sh 'git fetch --unshallow || true'
                def changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", r
eturnStdout: true).trim()
                echo "Changed files:\n${changedFiles}"
                def finalChangedFiles = params.CHANGED_FILES_LIST
                if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
                    echo "No changed files list from main job, using manual diff."
                    finalChangedFiles = changedFiles
                } else {
                    echo "Using changed files list from main job."
                }
                if (!finalChangedFiles.contains("${PROJECT_DIR}/")) {
                    echo "No changes found in ${PROJECT_DIR}. Setting SKIP_BUILD to true."
                    env.SKIP_BUILD = "true"
                }
                if (env.SKIP_BUILD == "true") echo "⏭ ${PROJECT_DIR}/ 변경 없음 → 빌드/배포
스킵"
            }
        }
    }

    stage('Place .env file') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        withCredentials([
          file(credentialsId: "${ENV_FILE_CRED_ID}", variable: 'ENV_TMP')
        ]) {
          sh '''
            set -e
            mkdir -p ${PROJECT_DIR}

            cp "$ENV_TMP" ${PROJECT_DIR}/.env.build
            chmod 600 ${PROJECT_DIR}/.env.build || true

            echo "== placed .env files for build =="
            ls -al ${PROJECT_DIR}/.env.build || true
          '''
        }
      }
    }

    // --build-arg 없이 단순 빌드로 변경 (Dockerfile이 .env.build를 읽음)
    stage('Docker Build') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        dir(env.PROJECT_DIR) { // PWA 프로젝트 디렉터리로 이동
          sh '''
            echo "== Docker build context =="; pwd && ls -al
            echo "== Dockerfile 내용 (상위 40줄) =="; sed -n '1,40p' Dockerfile || true
            echo "== docker build (using Dockerfile) =="

            # ❗ 단순 Docker 빌드 (Dockerfile이 알아서 .env.build를 읽음)
            docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .

            echo "== built images =="; docker images | grep ${IMAGE_NAME} || true
          '''
        }
      }
    }

    stage('Ensure Docker Network') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          docker network inspect ${NETWORK_NAME} >/dev/null 2>&1 || docker network
create ${NETWORK_NAME}
          echo "== networks =="; docker network ls | grep ${NETWORK_NAME} || true
        '''
      }
    }
    stage('Deploy Container') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          set -e
          echo "== stop & remove old container =="; docker stop ${DOCKER_CONTAINER_
NAME} || true; docker rm ${DOCKER_CONTAINER_NAME} || true

          echo "== run new container =="
          docker run -d \
            --name ${DOCKER_CONTAINER_NAME} \
            --network ${NETWORK_NAME} \
            --network-alias ${DOCKER_CONTAINER_NAME} \
            --restart unless-stopped \
            --env-file ${WORKSPACE}/${PROJECT_DIR}/.env.build \
            -e TZ=Asia/Seoul \
            -e NODE_ENV=production \
            -e PORT=${INTERNAL_PORT} \
            ${IMAGE_NAME}:${IMAGE_TAG}

          echo "== 추가 네트워크 연결: edge =="
          docker network connect edge ${DOCKER_CONTAINER_NAME} || true
          echo "== container inspect (name/networks) =="
          docker inspect --format 'Name: {{.Name}}, Networks: {{range $k, $v := .Network
Settings.Networks}}{{$k}}(IP: {{$v.IPAddress}}) {{end}}' ${DOCKER_CONTAINER_NAM
E}
          echo "== port mapping (Nginx 프록시를 사용하므로 호스트 포트 없음) =="
          docker ps --filter name=${DOCKER_CONTAINER_NAME} --format "table {{.Name
s}}\t{{.Ports}}\t{{.Status}}"
        '''
      }
    }
    stage('Health Check') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
          script {
          try {
            timeout(time: 2, unit: 'MINUTES') {
              boolean success = false
              echo "Waiting 10 seconds for container to start..."
              sleep(10)
              while (!success) {
                try {
                  echo "Attempting health check on ${env.HEALTH_CHECK_URL}..."
                  sh "curl -fL --max-time 10 -k ${env.HEALTH_CHECK_URL}"
                  success = true
                  echo "✅ Health check passed!"
                } catch (Exception e) {
                  echo "Health check failed (server not ready?). Retrying in 10 second
s..."
                  sleep(10)
                }
              }
            }
          } catch (org.jenkinsci.plugins.workflow.steps.FlowInterruptedException e) {
            echo "Health check timed out after 2 minutes."
            error('Health check timed out and failed.')
          }
          }
      }
    }
  }

  post {
    always {
      sh '''
        echo "== workspace listing =="; pwd && ls -al
        echo "== ${PROJECT_DIR} listing =="; [ -d ${PROJECT_DIR} ] && ls -al ${PROJEC
T_DIR} || true
        # ❗ [수정] 빌드용 임시 파일(.env.build) 삭제
        if [ -f "${PROJECT_DIR}/.env.build" ]; then
            echo "== Cleaning up temporary build file =="
            rm -f "${PROJECT_DIR}/.env.build"
        fi
        if [ "${SKIP_BUILD}" = "false" ]; then
          echo "== Pruning docker images =="
          docker image prune -f || true
        fi
      '''
    }

    success {
      script {
        def finalChangedFiles = params.CHANGED_FILES_LIST
        if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
            finalChangedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", re
turnStdout: true).trim()
        }
        def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AUTH
OR
        if (env.SKIP_BUILD == "false") {
            echo '🎉 Frontend App (PWA) 배포 및 헬스체크가 성공적으로 완료되었습니다!'
            def finalMessage = ""
            if (params.BUILD_URL && params.BUILD_URL != "") {
                finalMessage = """
#### :shinchan_dance: Jenkins Pipeline Success :shinchan_dance:
🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**
✍🏼 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기**
- [:react: Frontend App (PWA) **Dev Server**](${env.SERVICE_URL})
"""
            } else {
                finalMessage = """
#### :shinchan_dance2: Jenkins Pipeline Success :shinchan_dance2:
```

```
🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`
📝 **Changed Files**:
```
${finalChangedFiles}
```
💬 **빌드 체크가 성공적으로 완료되었습니다.**

----
:shin_hyeong_man: **서비스 점검하러 가기** :shin_hyeong_man:
- [:react: Frontend App (PWA) **Dev Server**](${env.SERVICE_URL})
"""
            }
            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#36a64f',
                message: finalMessage.stripIndent()
            )
        } else {
            echo "⏭️ ${PROJECT_DIR} 디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다."
            def finalMessage = ""
            if (params.BUILD_URL && params.BUILD_URL != "") {
                finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**
✍️ Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`
💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
            } else {
                finalMessage = """
#### :shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`
💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
            }
            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#ffaa00',
                message: finalMessage.stripIndent()
```

~~~
      )
    }
   }
  }
  failure {
    script {
      def logOutput = ""
      try {
        def logLines = currentBuild.rawBuild.getLog(150)
        logOutput = logLines.join('\n')
                  .replaceAll('\\$', '\\\\\\$')
                  .replaceAll('`', '\\\\`')
      } catch (Exception e) {
        logOutput = "Could not retrieve build log: ${e.message}"
      }
      def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AU
THOR
      def finalMessage = ""
      def failedStage = env.STAGE_NAME ?: "알 수 없음"
      if (params.BUILD_URL && params.BUILD_URL != "") {
        finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:
:alert_siren: **[${params.BUILD_TITLE}](${params.BUILD_URL})**

✍🏼 **Author**: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

##### 📝 Error Log (Failed Stage: ${failedStage})

```
${logOutput}
```

**전체 로그 확인하기**: ${env.BUILD_URL}console
"""
      } else {
        finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:

:alert_siren: **DEV Frontend App (PWA) 배포 실패!**

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

##### 📝 Error Log (Failed Stage: ${failedStage})

```
~~~

```
${logOutput}
```
"""
            }
            mattermostSend(
                endpoint: "${MATTERMOST_ENDPOINT}",
                channel: "${MATTERMOST_CHANNEL}",
                color: '#ff0000',
                message: finalMessage.stripIndent()
)
        }
    }
  }
}
```

- `develop-frontend-dashboard.groovy`

```
import groovy.json.JsonOutput

final def MM_USER_MAP = [
    "dahxtq1": "dahxtq1",
    "me_in_u": "me_in_u",
    "dhnn1536": "dhnn1536",
    "phangmin03": "phangmin03",
    "ryongseong.dev": "ryongseong.dev",
    "doriconi": "doriconi"
]

pipeline {
  agent any

  environment {

    GIT_URL           = "https://lab.ssafy.com/s13-final/S13P31E108.git"
    GIT_CREDENTIAL_ID = "gitlab_token_username_with_password"
    BRANCH            = "${params.BRANCH ?: 'develop'}"

    IMAGE_NAME            = "dev-frontend-dashboard"
    IMAGE_TAG             = "latest"

    DOCKER_CONTAINER_NAME = "dev-frontend-dashboard"
    NETWORK_NAME          = "devnet"

    INTERNAL_PORT = "3001"

    PROJECT_DIR = "frontend-dashboard"
```
```

```
    ENV_FILE_CRED_ID = "dev_env_frontend_dashboard"

    MATTERMOST_ENDPOINT = "https://meeting.ssafy.com/hooks/w74iy4erapnzixx37o
uiawi9ye"
    MATTERMOST_CHANNEL = "e108-release-notification"

    HEALTH_CHECK_URL = "https://buriburi.monster"
    SERVICE_URL = "https://buriburi.monster"

    SKIP_BUILD = "false"
  }

  parameters {
    string(name: 'BRANCH', defaultValue: 'develop', description: '빌드할 브랜치')
    string(name: 'BUILD_TITLE', defaultValue: '', description: 'MR/Commit Title from main
job')
    string(name: 'BUILD_URL', defaultValue: '', description: 'MR/Commit URL from main jo
b')
    string(name: 'BUILD_AUTHOR', defaultValue: 'Unknown', description: 'MR/Commit A
uthor from main job')
    string(name: 'BUILD_TARGET_BRANCH', defaultValue: 'develop', description: 'MR/Co
mmit Target Branch from main job')
    string(name: 'CHANGED_FILES_LIST', defaultValue: '', description: 'List of changed fi
les from main job')
  }

  stages {
    stage('Checkout') {
      steps {
        cleanWs()
        echo "WORKSPACE: ${env.WORKSPACE}"
        sh 'pwd && ls -al'
        git url: env.GIT_URL, branch: env.BRANCH, credentialsId: env.GIT_CREDENTIAL_ID
        sh '''
          echo "== After git clone =="; pwd && ls -al
          echo "== ${PROJECT_DIR} (frontend-dashboard) listing =="
          [-d ${PROJECT_DIR}] && ls -al ${PROJECT_DIR} || echo "${PROJECT_DIR} not f
ound"
        '''
      }
    }
    stage('Detect Changes in frontend-dashboard/') {
        steps {
            script {
                sh 'git fetch --unshallow || true'
                def changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", r
eturnStdout: true).trim()
                echo "Changed files:\n${changedFiles}"
                def finalChangedFiles = params.CHANGED_FILES_LIST
                if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
                  echo "No changed files list from main job, using manual diff."
                  finalChangedFiles = changedFiles
                } else {
                  echo "Using changed files list from main job."
                }
                if (!finalChangedFiles.contains("${PROJECT_DIR}/")) {
                  echo "No changes found in ${PROJECT_DIR}. Setting SKIP_BUILD to true."
                  env.SKIP_BUILD = "true"
                }
                if (env.SKIP_BUILD == "true") echo "⏭ ${PROJECT_DIR}/ 변경 없음 → 빌드/배포
스킵"
            }
        }
    }

    stage('Place .env file') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        withCredentials([
          file(credentialsId: "${ENV_FILE_CRED_ID}", variable: 'ENV_TMP')
        ]) {
          sh '''
            set -e
            mkdir -p ${PROJECT_DIR}

            cp "$ENV_TMP" ${PROJECT_DIR}/.env.build
            chmod 600 ${PROJECT_DIR}/.env.build || true

            echo "== placed .env files for build and runtime =="
            ls -al ${PROJECT_DIR}/.env.build || true
          '''
        }
      }
    }

    // 단순 빌드로 변경 (Dockerfile이 .env.build를 읽음)
    stage('Docker Build') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        dir(env.PROJECT_DIR) { // 대시보드 프로젝트 디렉터리로 이동
          sh '''
            echo "== Docker build context =="; pwd && ls -al
            echo "== Dockerfile 내용 (상위 40줄) =="; sed -n '1,40p' Dockerfile || true
            echo "== docker build (using Dockerfile) =="

            docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .
            echo "== built images =="; docker images | grep ${IMAGE_NAME} || true
          '''
        }
      }
    }

    stage('Ensure Docker Network') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          docker network inspect ${NETWORK_NAME} >/dev/null 2>&1 || docker network
create ${NETWORK_NAME}
          echo "== networks =="; docker network ls | grep ${NETWORK_NAME} || true
        '''
      }
    }
    stage('Deploy Container') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        sh '''
          set -e
          echo "== stop & remove old container =="; docker stop ${DOCKER_CONTAINER_
NAME} || true; docker rm ${DOCKER_CONTAINER_NAME} || true

          echo "== run new container =="
          docker run -d \
            --name ${DOCKER_CONTAINER_NAME} \
            --network ${NETWORK_NAME} \
            --network-alias ${DOCKER_CONTAINER_NAME} \
            --restart unless-stopped \
            --env-file ${WORKSPACE}/${PROJECT_DIR}/.env.build \
            -e TZ=Asia/Seoul \
            -e NODE_ENV=production \
            -e PORT=${INTERNAL_PORT} \
            ${IMAGE_NAME}:${IMAGE_TAG}

          echo "== 추가 네트워크 연결: edge =="
          docker network connect edge ${DOCKER_CONTAINER_NAME} || true
          echo "== container inspect (name/networks) =="
          docker inspect --format 'Name: {{.Name}}, Networks: {{range $k, $v := .Network
Settings.Networks}}{{$k}}(IP: {{$v.IPAddress}}) {{end}}' ${DOCKER_CONTAINER_NAM
E}
          echo "== port mapping (Nginx 프록시를 사용하므로 호스트 포트 없음) =="
          docker ps --filter name=${DOCKER_CONTAINER_NAME} --format "table {{.Name
s}}\t{{.Ports}}\t{{.Status}}"
        '''
      }
    }
    stage('Health Check') {
      when { environment name: 'SKIP_BUILD', value: 'false' }
      steps {
        script {
          try {
            timeout(time: 2, unit: 'MINUTES') {
              boolean success = false
              echo "Waiting 10 seconds for container to start..."
              sleep(10)
              while (!success) {
                try {
                  echo "Attempting health check on ${env.HEALTH_CHECK_URL}..."
                  sh "curl -fL --max-time 10 -k ${env.HEALTH_CHECK_URL}"
                  success = true
                  echo "✅ Health check passed!"
                } catch (Exception e) {
                  echo "Health check failed (server not ready?). Retrying in 10 second
s..."
                  sleep(10)
                }
              }
            }
          } catch (org.jenkinsci.plugins.workflow.steps.FlowInterruptedException e) {
            echo "Health check timed out after 2 minutes."
            error('Health check timed out and failed.')
          }
        }
      }
    }
  }

  post {
    always {
      sh '''
        echo "== workspace listing =="; pwd && ls -al
        echo "== ${PROJECT_DIR} listing =="; [-d ${PROJECT_DIR}] && ls -al ${PROJEC
T_DIR} || true
        # ❗ [수정] 빌드용 임시 파일(.env.build) 삭제
        if [-f "${PROJECT_DIR}/.env.build"]; then
          echo "== Cleaning up temporary build file =="
          rm -f "${PROJECT_DIR}/.env.build"
        fi
        if ["${SKIP_BUILD}" = "false"]; then
          echo "== Pruning docker images =="
          docker image prune -f || true
        fi
      '''
  }

  // ... (success, failure 블록은 동일, 메시지만 'Web Dashboard'로 수정) ...
  success {
   script {
    def finalChangedFiles = params.CHANGED_FILES_LIST
    if (finalChangedFiles == null || finalChangedFiles.trim() == "") {
      finalChangedFiles = sh(script: "git diff --name-only HEAD~1 HEAD || echo ''", re
turnStdout: true).trim()
    }
    def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AUTH
OR
    if (env.SKIP_BUILD == "false") {
      echo '🎉 Frontend Dashboard 배포 및 헬스체크가 성공적으로 완료되었습니다!'
      def finalMessage = ""
      if (params.BUILD_URL && params.BUILD_URL != "") {
        finalMessage = """
:shinchan_dance: Jenkins Pipeline Success :shinchan_dance:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**
✍️ Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **빌드 체크가 성공적으로 완료되었습니다.**

:shin_hyeong_man: **서비스 점검하러 가기**
- [:desktop_computer: Web Dashboard **Dev Server**](${env.SERVICE_URL})
"""
      } else {
        finalMessage = """
:shinchan_dance2: Jenkins Pipeline Success :shinchan_dance2:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`
📝 **Changed Files**:
```
${finalChangedFiles}
```

💬 **빌드 체크가 성공적으로 완료되었습니다.**

:shin_hyeong_man: **서비스 점검하러 가기** :shin_hyeong_man:
- [:desktop_computer: Web Dashboard **Dev Server**](${env.SERVICE_URL})
"""
      }
      mattermostSend(
        endpoint: "${MATTERMOST_ENDPOINT}",
        channel: "${MATTERMOST_CHANNEL}",
        color: '#36a64f',
        message: finalMessage.stripIndent()
)
    } else {
      echo "⏭️ ${PROJECT_DIR} 디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다."
      def finalMessage = ""
      if (params.BUILD_URL && params.BUILD_URL != "") {
        finalMessage = """
:shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🔀 **[${params.BUILD_TITLE}](${params.BUILD_URL})**
✍🏼 Author: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
      } else {
        finalMessage = """
:shinchan_walking: Jenkins Pipeline Skipped :shinchan_walking:

🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

💬 **디렉토리 변경사항이 없어서 빌드를 건너뛰었습니다.**
"""
      }
      mattermostSend(
        endpoint: "${MATTERMOST_ENDPOINT}",
        channel: "${MATTERMOST_CHANNEL}",
        color: '#ffaa00',
        message: finalMessage.stripIndent()
)
    }
   }
  }
  failure {
    script {
      def logOutput = ""
      try {
        def logLines = currentBuild.rawBuild.getLog(150)
        logOutput = logLines.join('\n')
                    .replaceAll('\\$', '\\\\\\$')
                    .replaceAll('`', '\\\\`')
      } catch (Exception e) {
            logOutput = "Could not retrieve build log: ${e.message}"
          }
          def mmAuthor = MM_USER_MAP[params.BUILD_AUTHOR] ?: params.BUILD_AU
THOR
          def finalMessage = ""
          def failedStage = env.STAGE_NAME ?: "알 수 없음"
          if (params.BUILD_URL && params.BUILD_URL != "") {
            finalMessage = """
:shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:

:alert_siren: **[${params.BUILD_TITLE}](${params.BUILD_URL})**
✍🏼 **Author**: @${mmAuthor}
🎯 **Target Branch**: `${params.BUILD_TARGET_BRANCH}`

📝 Error Log (Failed Stage: ${failedStage})

```
${logOutput}
```

전체 로그 확인하기: ${env.BUILD_URL}console
"""
          } else {
            finalMessage = """
#### :shoke-shin-chang: Jenkins Pipeline Failure :shoke-shin-chang:

:alert_siren: **DEV Web Dashboard 배포 실패!**
🎯 **Target Branch**: `${env.BRANCH}`
🔗 **Build Number**: #${env.BUILD_NUMBER}
✨ **Project**: `${env.JOB_NAME}`

##### 📝 Error Log (Failed Stage: ${failedStage})

```
${logOutput}
```
"""
          }
          mattermostSend(
            endpoint: "${MATTERMOST_ENDPOINT}",
            channel: "${MATTERMOST_CHANNEL}",
            color: '#ff0000',
            message: finalMessage.stripIndent()
)
        }
      }
    }
```

```
}
```

(8) Nginx 설정

nginx_data/conf.d/

- api.conf

```
server {
    listen 80;
    server_name api.buriburi.monster;

    # HTTP → HTTPS 리디렉션
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    http2 on;
    server_name api.buriburi.monster;

    # --- SSL 설정 ---
    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # --- 보안 헤더 ---
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; pre
load" always;

    # --- 업로드 및 타임아웃 ---
    client_max_body_size 50m;
    proxy_read_timeout   300;
    proxy_send_timeout   300;
    proxy_connect_timeout 60;

    # --- 공통 프록시 헤더 ---
    proxy_http_version 1.1;

    proxy_set_header Upgrade           $http_upgrade;
    proxy_set_header Connection        $connection_upgrade;

    # 쿠키 및 인증 헤더
    proxy_set_header Cookie            $http_cookie;
    proxy_set_header Authorization      $http_authorization;

    # 호스트 및 클라이언트 정보 헤더
    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_set_header X-Forwarded-Port   $server_port;
    proxy_set_header X-Forwarded-Host   $host;

    # CORS 관련 헤더 전달
    proxy_set_header Origin            $http_origin;
    proxy_set_header Access-Control-Request-Method $http_access_control_request_m
ethod;
    proxy_set_header Access-Control-Request-Headers $http_access_control_request_h
eaders;

    # --- 1. Spring 부트 라우팅 ---
    # https://api.buriburi.monster/spring/ 로 오는 모든 요청
    location /spring/ {
        # 'edge' 네트워크의 'dev-backend-spring' 컨테이너 8080 포트로 전달
        proxy_pass http://dev-backend-spring:8080/spring/;
    }

    # --- 2. FastAPI 라우팅 ---
    # https://api.buriburi.monster/fastapi/ 로 오는 모든 요청
    location /fastapi/ {
        # 'edge' 네트워크의 'dev-backend-fastapi' 컨테이너 8081 포트로 전달
        proxy_pass http://dev-backend-fastapi:8081/fastapi/;
    }

    # --- Spring Boot Swagger API Docs 라우팅 ---
    # Swagger UI 페이지가 내부적으로 /v3/api-docs/ 경로로 API 명세를 요청하면 백엔드로 전달
    # location /v3/api-docs {
    #     proxy_pass http://dev-backend-spring:8080/v3/api-docs;
    # }

    # --- 루트 경로 처리 ---
    # https://api.buriburi.monster/ 로 직접 접근 시
    location / {
        return 404;
    }
}
```

- app.conf

```
server {
    listen 80;
    server_name app.buriburi.monster;

    # HTTP → HTTPS 리디렉션
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    http2 on;
    server_name app.buriburi.monster;

    # --- SSL 설정 ---
    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # --- 보안 헤더 ---
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; pre
load" always;

    # --- 업로드 및 타임아웃 ---
    client_max_body_size 50m;
    proxy_read_timeout   300;
    proxy_send_timeout   300;
    proxy_connect_timeout 60;

    # --- 메인 PWA 라우팅 ---
    location / {
        # 'edge' 네트워크의 'dev-frontend-app' 컨테이너 3000 포트로 전달
        proxy_pass http://dev-frontend-app:3000;

        # --- 공통 프록시 헤더 (WebSocket 지원 포함) ---
        proxy_http_version 1.1;
        proxy_set_header Upgrade           $http_upgrade;
        proxy_set_header Connection        $connection_upgrade;

        proxy_set_header Cookie            $http_cookie;
        proxy_set_header Authorization     $http_authorization;
        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_set_header X-Forwarded-Port  $server_port;
        proxy_set_header X-Forwarded-Host  $host;
    }
}
```

- `jenkins.conf`

```
server {
    listen 80;
    server_name jenkins.buriburi.monster;

    # HTTP → HTTPS 리디렉션
    location / {
        return 301 https://$host$request_uri;
    }
}

# 80번 포트(HTTP) -> 443 포트(HTTPS)로 리디렉트
server {
    listen 443 ssl;
    http2 on;
    server_name jenkins.buriburi.monster;

    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    client_max_body_size 200m;
    proxy_read_timeout   3600;
    proxy_send_timeout   3600;
    proxy_connect_timeout 60;
    proxy_buffering off;
    proxy_request_buffering off;

    location / {
        proxy_pass http://jenkins:8080;     # edge 네트워크의 jenkins 서비스명
        proxy_http_version 1.1;

        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "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;
    }
}
```

- `k13e108.conf`

```
# 80번 포트(HTTP) -> 443 포트(HTTPS)로 리디렉트
server {
    listen 80;
    server_name k13e108.p.ssafy.io;

     # cerbot 갱신 경로
    location /.well-known/acme-challenge/ {
     root /var/www/certbot;
      }

    # 모든 80포트 요청은 메인 도메인(HTTPS)로 리디렉트
    location / {
            return 301 https://buriburi.monster$request_uri;
      }
}

server {
    listen 443 ssl;
    http2 on;
    server_name k13e108.p.ssafy.io;

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

    # 모든 443 포트 요청은 메인 도메인(HTTPS)로 리디렉트
    location / {
        return 301 https://buriburi.monster$request_uri;
    }
}
```

- `n8n.conf`

```
server {
  listen 80;
  listen [::]:80;
  server_name n8n.buriburi.monster;

  # HTTP → HTTPS 리디렉션
  location / {
    return 301 https://$host$request_uri;
  }
}

# 80번 포트(HTTP) -> 443 포트(HTTPS)로 리디렉트
server {
  listen 443 ssl;
  http2 on;
  server_name n8n.buriburi.monster;

  # Use your existing wildcard certificate
  ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;

  location / {
    proxy_pass http://n8n:5678;
    proxy_http_version 1.1;

    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "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;
  }
}
```

- redis-dev.conf

```
# HTTP → HTTPS 리다이렉트
server {
    listen 80;
    listen [::]:80;
    server_name redisin-dev.buriburi.monster;
    return 301 https://$host$request_uri;
}

# HTTPS 프록시
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    http2 on;
    server_name redisin-dev.buriburi.monster;

    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;

    # (선택) 보안 헤더
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;

    add_header X-VHost "edge-redisin-dev v1" always;

    # RedisInsight UI는 루트로 노출
    location / {
        # 프록시 대상 컨테이너 이름
        proxy_pass http://redisinsight-dev:5540;

        # WebSocket & 일반 프록시 공통
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_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;

        proxy_read_timeout 300;
        proxy_send_timeout 300;

        # (선택) 버퍼링 해제시 실시간 로그/스트림에 유리
        proxy_buffering off;
    }
}

# map 블록은 http {} 범위에 정의되어 있어야 함 (nginx.conf의 http {} 안이나 별도 파일)
# WebSocket 업그레이드 처리
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}
```

- `root.conf`

```
server {
  listen 80;
  listen [::]:80;

  # www도 함께 처리
  server_name buriburi.monster www.buriburi.monster;

  # Certbot 갱신용 챌린지 경로
  location /.well-known/acme-challenge/ {
    root /var/www/certbot;
  }
  # 나머지 모든 요청은 HTTPS로 강제 리디렉트
  location / {
    return 301 https://$host$request_uri;
  }
}

server {
    listen 443 ssl;
    http2 on;
    server_name buriburi.monster www.buriburi.monster;

    # --- SSL 설정 ---
    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # --- 보안 헤더 ---
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; pre
load" always;

    # --- 업로드 및 타임아웃 ---
    client_max_body_size 50m;
    proxy_read_timeout   300;
    proxy_send_timeout   300;
    proxy_connect_timeout 60;

    # --- 메인 대시보드 라우팅 ---
    location / {
        # 'edge' 네트워크의 'dev-frontend-dashboard' 컨테이너 3001 포트로 전달
        proxy_pass http://dev-frontend-dashboard:3001;

        # --- 공통 프록시 헤더 (WebSocket 지원 포함) ---
        proxy_http_version 1.1;
        proxy_set_header Upgrade           $http_upgrade;
        proxy_set_header Connection        $connection_upgrade;

        proxy_set_header Cookie            $http_cookie;
        proxy_set_header Authorization     $http_authorization;

        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_set_header X-Forwarded-Port   $server_port;
        proxy_set_header X-Forwarded-Host   $host;
    }

    # 정적 파일 캐시(선택)
    location ~* \.(?:js|css|png|jpg|jpeg|gif|webp|ico|svg|woff2?)$ {
        proxy_pass http://dev-frontend-dashboard:3001;

        proxy_http_version 1.1;
        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;

        # 캐시 설정
        expires 7d;
        add_header Cache-Control "public, max-age=604800" always;
    }
}
```

- `rtc.conf`

```
# 80(HTTP) → 443(HTTPS) 리디렉트
server {
    listen 80;
    server_name rtc.buriburi.monster;
    location / {
        return 301 https://$host$request_uri;
    }
}

# 443(HTTPS) 요청을 OpenVidu로 프록시
server {
    listen 443 ssl;
    http2 on;
    server_name rtc.buriburi.monster;

    ssl_certificate     /etc/letsencrypt/live/buriburi.monster/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/buriburi.monster/privkey.pem;

    # 모든 요청(HTTP + WebSocket)을 OpenVidu 스택으로 전달
    location / {
          # 'https'와 '20443' 포트로 연결
          proxy_pass https://host.docker.internal:20443;
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto $scheme;
```

```
            # --- WebSocket 업그레이드 헤더 추가 ---
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "Upgrade";
            proxy_read_timeout 86400; # 타임아웃 설정

            # OpenVidu는 자체 서명 인증서(selfsigned)를 쓰므로, 오류 무시
            proxy_ssl_verify off;
            proxy_ssl_session_reuse on;
    }
}
```

`nginx_data/stream.d/`

- `redis.stream.conf`

```
server {
  listen 6381;
  proxy_connect_timeout 10s;
  proxy_timeout 5m;
  proxy_pass redis-dev:6379;

  # Redis 연결 유지
  proxy_socket_keepalive on;

  # 팀 IP 화이트리스트 (필요시 활성화)
  # allow 192.168.0.0/16;  # 로컬 네트워크
  # allow 172.16.0.0/12;   # Docker 네트워크
  # allow 10.0.0.0/8;      # 프라이빗 네트워크
  # deny all;
}
```

`nginx_data/`

- `nginx.conf`

```
user  nginx;
worker_processes  auto;

events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile on;
    keepalive_timeout 65;

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

    server_tokens off;

    include /etc/nginx/conf.d/*.conf;
}

stream {
  # Docker 내장 DNS (컨테이너 이름 재해석용)
  resolver 127.0.0.11 ipv6=off valid=30s;

  # stream 설정을 분리 관리
  include /etc/nginx/stream.d/*.conf;

  # include /etc/nginx/stream.d/dev/*.conf;
  # include /etc/nginx/stream.d/edge/*.conf;
  # include /etc/nginx/stream.d/prod/*.conf;
}
```

- 설정 변경 후 Nginx 재시작

```
sudo systemctl restart nginx
```

(9) 테스트 명령어

Frontend (App & Dashboard)

```
# 개발 서버 실행
npm run dev

# 프로덕션 빌드
npm run build

# 빌드 결과물 실행
npm start
```

Backend (Spring)

```
# 프로젝트 빌드 (테스트 포함)
./gradlew build

# 테스트 스킵하고 빌드
./gradlew build -x test
```

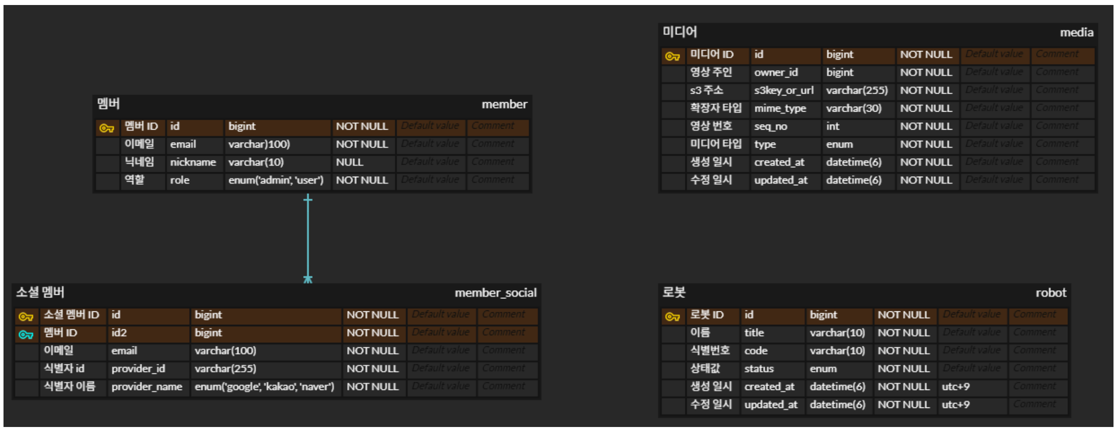
Backend (FastAPI)

```
python3 -m venv venv
source venv/bin/activate # In windows command: source venv/Scripts/activate
pip install -r requirements/local.txt
python main.py
```

(10) DataBase

ERD

- 🔗 ERD Cloud 이동



멤버 member

🔑	멤버 ID	id	bigint	NOT NULL	Default value	Comment
	이메일	email	varchar)100)	NOT NULL	Default value	Comment
	닉네임	nickname	varchar(10)	NULL	Default value	Comment
	역할	role	enum('admin', 'user')	NOT NULL	Default value	Comment

소셜 멤버 member_social

🔑	소셜 멤버 ID	id	bigint	NOT NULL	Default value	Comment
🔑	멤버 ID	id2	bigint	NOT NULL	Default value	Comment
	이메일	email	varchar(100)	NOT NULL	Default value	Comment
	식별자 id	provider_id	varchar(255)	NOT NULL	Default value	Comment
	식별자 이름	provider_name	enum('google', 'kakao', 'naver')	NOT NULL	Default value	Comment

미디어 media

🔑	미디어 ID	id	bigint	NOT NULL	Default value	Comment
	영상 주인	owner_id	bigint	NOT NULL	Default value	Comment
	s3 주소	s3key_or_url	varchar(255)	NOT NULL	Default value	Comment
	확장자 타입	mime_type	varchar(30)	NOT NULL	Default value	Comment
	영상 번호	seq_no	int	NOT NULL	Default value	Comment
	미디어 타입	type	enum	NOT NULL	Default value	Comment
	생성 일시	created_at	datetime(6)	NOT NULL	Default value	Comment
	수정 일시	updated_at	datetime(6)	NOT NULL	Default value	Comment

로봇 robot

🔑	로봇 ID	id	bigint	NOT NULL	Default value	Comment
	이름	title	varchar(10)	NOT NULL	Default value	Comment
	식별번호	code	varchar(10)	NOT NULL	Default value	Comment
	상태값	status	enum	NOT NULL	Default value	Comment
	생성 일시	created_at	datetime(6)	NOT NULL	utc+9	Comment
	수정 일시	updated_at	datetime(6)	NOT NULL	utc+9	Comment