포팅 메뉴얼

1. 사용 도구

- 이슈 관리 : Jira
- 형상 관리 : GitLab
- 커뮤니케이션 : MatterMost, Notion, Google sheet
- 디자인 : Figma
- CI/CD : 젠킨스

2. 개발 도구

Frontend

- **프레임워크**: Next.js, React
- **언어** : TypeScript, JavaScript
- 라이브러리 및 의존성 모음

```
"@egjs/react-flicking": "^4.12.0",
  "@emotion/cache": "^11.14.0",
  "@emotion/react": "^11.14.0",
  "@emotion/styled": "^11.14.0",
  "@floating-ui/react": "^0.27.5",
  "@fullcalendar/core": "^6.1.17",
  "@fullcalendar/daygrid": "^6.1.17",
  "@fullcalendar/interaction": "^6.1.17",
  "@fullcalendar/react": "^6.1.17",
  "@headlessui/react": "^2.2.0",
  "@mui/material": "^7.0.1",
  "@mui/x-date-pickers": "^7.28.2",
  "@radix-ui/react-dialog": "^1.1.6",
  "@react-spring/web": "^9.7.5",
  "@reduxjs/toolkit": "^2.6.1",
  "@types/d3": "^7.4.3",
  "@use-gesture/react": "^10.3.1",
```

```
  "axios": "^1.8.4",
  "d3": "^7.9.0",
  "date-fns": "^2.30.0",
  "date-fns-tz": "^2.0.0",
  "eslint-config-prettier": "^10.1.1",
  "eslint-plugin-prettier": "^5.2.4",
  "firebase": "^11.5.0",
  "next": "15.2.3",
  "next-redux-wrapper": "^8.1.0",
  "prettier": "^3.5.3",
  "qs": "^6.14.0",
  "react": "18.2.0",
  "react-calendar": "^5.1.0",
  "react-clock": "^5.1.0",
  "react-datepicker": "^8.2.1",
  "react-dom": "18.2.0",
  "react-hook-form": "^7.54.2",
  "react-i18next": "^15.4.1",
  "react-ios-time-picker": "^0.2.2",
  "react-mobile-picker": "^1.1.0",
  "react-redux": "^9.2.0",
  "redux-persist": "^6.0.0"
```

- 개발 도구 및 환경 구성

```
"@eslint/eslintrc": "^3",
  "@trivago/prettier-plugin-sort-imports": "^5.2.2",
  "@types/node": "^20",
  "@types/qs": "^6.9.18",
  "@types/react": "^19",
  "@types/react-dom": "^19",
  "@types/react-time-picker": "^4.0.3",
  "eslint": "^9",
  "eslint-config-next": "15.2.3",
  "eslint-plugin-react-refresh": "^0.4.19",
  "next-pwa": "^5.6.0",
  "redux-logger": "^3.0.6",
  "typescript": "^5"
```

- **기타 환경 정보**
  - 상태관리: `Redux Toolkit`, `redux-persist`
  - 코드 품질 관리: `ESLint`, `Prettier`
  - 번역: `react-i18next`
  - 날짜 및 시간: `date-fns`, `react-datepicker`, `react-ios-time-picker`

Backend

프레임워크 : Spring Boot 3.4.3

언어 : Java 21

IDE : IntelliJ IDEA 2024.1.1

빌드 도구 : Gradle

주요 설정

- Java Toolchain 사용: `Java 21`
- Dependency 관리: `io.spring.dependency-management` 플러그인 사용
- 테스트 플랫폼: `JUnit Platform`

라이브러리 및 의존성 목록

🔷 Spring Core

- `spring-boot-starter-web` : REST API 개발
- `spring-boot-starter-data-jpa` : JPA & Hibernate
- `spring-boot-starter-security` : Spring Security
- `spring-boot-starter-oauth2-client` : OAuth2 인증
- `spring-boot-starter-batch` : Spring Batch
- `hibernate-validator` : Bean Validation

🔷 데이터베이스

- `mysql-connector-j` : MySQL JDBC
- `h2` : 인메모리 테스트용 DB
- `redis` : 서버간 세션 공유 DB

- `spring-boot-starter-data-jpa` : ORM
- `p6spy-spring-boot-starter` : SQL 로그 확인

🔷 테스트

- `spring-boot-starter-test` : Spring Test
- `spring-security-test` : 시큐리티 테스트
- `spring-batch-test` : 배치 테스트
- `junit-platform-launcher` : JUnit 런타임
- `h2` (`testImplementation`): 테스트용 인메모리 DB

🔷 JWT

- `jjwt-api`, `jjwt-impl`, `jjwt-jackson`: JWT 토큰 처리

🔷 Firebase

- `firebase-admin` : Firebase Admin SDK

🔷 AWS

- `spring-cloud-aws-starter-parameter-store` : AWS SSM 파라미터 스토어
- `aws-java-sdk-s3` : S3 연동

🔷 API 문서화

- `springdoc-openapi-starter-webmvc-ui` : Swagger UI (OpenAPI)

🔷 PDF 생성

- `itext7-core` : PDF 생성
- `openhtmltopdf-pdfbox`, `openhtmltopdf-slf4j` : HTML → PDF 렌더링

🔷 기타 유틸

- `jackson-databind` : JSON 직렬화/역직렬화
- `guava` : Google 유틸리티
- `org.json:json` : 간단한 JSON 처리
- `okhttp` : HTTP 클라이언트

- `lombok` : 보일러플레이트 코드 제거 (compileOnly + annotationProcessor)

🔷 Web3

- `web3j-core` : 블록체인 연동

레포지토리 설정

```
repositories {
    mavenCentral()
    maven {
        url "https://oss.sonatype.org/content/repositories/snapshots/"
    }
}
```

Spring Cloud AWS BOM

```
dependencyManagement {
    imports {
        mavenBom "io.awspring.cloud:spring-cloud-aws-dependencies:3.0.0"
    }
}
```

개발 환경

Frontend

- JavaScript
  - Node.js 20.13.1
  - React 18.2.0
  - Next.js 15.2.3
  - TypeScript 5

Backend

- Java 21
- Spring Boot 3.4.3
- Spring Cloud AWS 3.0.0

- MySQL 8.4.4
- H2
- Firebase Admin SDK 9.3.0
- JWT jjwt 0.12.6
- Web3j 4.12.0
- PDF 생성 iText 7.2.4, OpenHtmlToPdf 1.0.10

Infra

- `Docker` : 28.0.1
- `Nginx` : 1.18.0 (ubuntu)
- `Redis` : 7.4.3
- `MySQL` : 8.4.3
- `ELK` → `docker compose 9.0` 버전
  - `Elastic Search`
  - `LogStash`
  - `Kibana`
- `FileBeat` : 8.5.1
- `Grafana-Prometheus` : `latest`
- `AWS`
  - `ELB - ALB`
  - `R53`
  - `SSM - Parameter Store`
  - `EC2`

---

4. 환경변수

Backend

- `application.yml` → 로컬용

```
spring:
  cloud:
    aws:
      region:
        static: ${AWS_REGION}
      credentials:
        access-key: ${AWS_ACCESS_KEY_ID}
        secret-key: ${AWS_SECRET_ACCESS_KEY}
  application:
    name:
  config:
    import:
      - optional:file:.env[.properties] #ENV ?? ??
      - aws-parameterstore:/chaing/

  datasource:
    url: ${DATASOURCE_URL}
    username: ${DATASOURCE_USERNAME}
    password: ${DATASOURCE_PASSWORD}
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQLDialect

  security:
    oauth2:
      client:
        registration:
          google:
            client-name: google
            client-id: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            redirect-uri: ${GOOGLE_REDIRECT_URI}
```

```
            authorization-grant-type: authorization_code
            scope:
              - email
              - profile
  profiles:
    active: dev

  servlet:
    multipart:
      max-file-size: 20MB
      max-request-size: 50MB
      enabled: true

  mail:
    host: smtp.gmail.com
    port: 587
    username: ${GOOGLE_MAIL_USERNAME}
    password: ${GOOGLE_MAIL_PASSWORD}
    protocol: smtp
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
            required: true

  batch:
    jdbc:
      initialize-schema: always

  lifecycle:
    timeout-per-shutdown-phase: 60s

server:
  shutdown: graceful

application:
```

```
  security:
    jwt:
      secret-key: ${APPLICATION_SECURITY_JWT_SECRET_KEY}
      access-token-expiration: ${APPLICATION_SECURITY_JWT_ACCESS_T
      refresh-token-expiration: ${APPLICATION_SECURITY_JWT_REFRESH_

firebase:
  project-name: ${FIREBASE_PROJECT_NAME}
  service-account-base64: ${FIREBASE_SERVICE_ACCOUNT}

app:
  cors:
    allow-hosts:
      - http://localhost:8080
      - http://localhost:3000
  frontend:
    url: http://localhost:3000
springdoc:
  api-docs:
    path: /v3/api-docs
    version: openapi_3_0


api:
  naver:
    client-id: ${API_NAVER_CLIENT_ID}
    client-secret: ${API_NAVER_CLIENT_SECRET}

cloud:
  aws:
    s3:
      bucket: ${AWS_BUCKET_NAME}

web3j:
  client-address: ${web3j.client-address}
  fallback-client-address: ${web3j.fallback-client-address}
  connection-timeout: 15
  contract-wallet-private-key: ${web3j.contract-wallet-private-key}
```

```
  rent-wallet-private-key: ${web3j.rent-wallet-private-key}
  utility-wallet-private-key: ${web3j.utility-wallet-private-key}
  contract-address: ${web3j.contract-address} # DB 에 넣어야 함.
  rent-contract-address: ${web3j.utility-contract-address}
  utility-contract-address: ${web3j.rent-contract-address}
  chain-id: ${web3j.chain-id}

ssafy:
  fintech:
    api-key: ${FINTECH_API_KEY}
    user-key: ${FINTECH_USER_KEY}
    card-unique-no: ${FINTECH_CARD_UNIQUE_NO}
    base-url: ${FINTECH_BASEURL}

openai:
  api:
    key: ${GPT_API_KEY}
    url: ${GPT_URL}
  model: ${GPT_MODEL}
```

- `application-deploy-1.yml` → 배포 서버 1

```
spring:
  application:
    name: vibeEditorProject
  cloud:
    aws:
      region:
        static: ${AWS_REGION}
      credentials:
        access-key: ${AWS_ACCESS_KEY_ID}
        secret-key: ${AWS_SECRET_ACCESS_KEY}
  config:
    import:
      - optional:file:.env[.properties]
      - aws-parameterstore:/vibeeditor/

  profiles:
```

```
  active: prod

session:
  redis:
    namespace: vibe-editor

datasource:
  url: ${datasource.url}
  username: ${datasource.username}
  password: ${datasource.password}
  driver-class-name: com.mysql.cj.jdbc.Driver

jpa:
  hibernate:
    ddl-auto: validate
  show-sql: false
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQLDialect
data:
  mongodb:
    uri: ${mongodb.url}
  redis:
    host: ${redis.host1}
    port: ${redis.port}
    password: ${redis.password}

security:
  user:
    name: ${security.user-name}
    password: ${security.user-password}
    roles: ${security.user-role}

  cors:
    allowed-origins: ${security.cors-allow-origin}

  jwt:
    secret-key: ${security.jwt-secret-key}
```

```
  access-token-expiration: ${security.jwt-access.expiration}
  refresh-token-expiration: ${security.jwt-refresh.expiration}
  pass-urls: ${security.jwt-pass-urls}
  admin-urls: ${security.jwt-admin-urls}
oauth2:
  client:
    registration:
      google:
        client-name: google
        client-id: ${google.client-id}
        client-secret: ${google.client-secret}
        redirect-uri: ${google.redirect-url}
        authorization-grant-type: authorization_code
        scope:
          - email
          - profile
      github:
        client-name: Github
        client-id: ${github.client-id}
        client-secret: ${github.client-secret}
        redirect-uri: ${github.redirect-url}
        authorization-grant-type: authorization_code
        scope:
          - read:user
          - user:email

    provider:
      google:
        authorization-uri: https://accounts.google.com/o/oauth2/auth
        token-uri: https://oauth2.googleapis.com/token
        user-info-uri: https://openidconnect.googleapis.com/v1/userinfo
        user-name-attribute: sub
      github:
        authorization-uri: https://github.com/login/oauth/authorize
        token-uri: https://github.com/login/oauth/access_token
        user-info-uri: https://api.github.com/user
        user-name-attribute: id
```

```
  servlet:
    multipart:
      enabled: true
      max-file-size: 20MB
      max-request-size: 50MB

  ai:
    openai:
      api-key: ${gpt.api-key}
      base-url: ${gpt.api-url}
      model: ${gpt.model}

    anthropic:
      api-key: ${claude.api-key}
      chat:
        options:
          temperature: ${claude.temperature}
          model: ${claude.model}
          max-tokens: ${claude.max-token}
      base-url: ${claude.baseurl}

  lifecycle:
    timeout-per-shutdown-phase: 60s

server:
  shutdown: graceful

app:
  cors:
    allow-hosts:
      - ${allow-host.front}
      - ${allow-host.local}

springdoc:
  api-docs:
    path: /v3/api-docs
    version: openapi_3_0
```

```
logging:
  file:
    name: /var/log/vibe/vibe.log
  level:
    root: INFO

decorator:
  datasource:
    p6spy:
      enable-logging: false
notion:
  api:
    version: ${notion.version}
    base_url: ${notion.baseurl}

management:
  endpoints:
    access:
      default: none
    web:
      exposure:
        include: health,info,prometheus
      base-path: /api
  endpoint:
    health:
      show-details: always
      access: unrestricted
    prometheus:
      access: unrestricted

aes-algo: ${security.aes-algorithm}
aes-secret-key: ${security.aes-key}

ssafy:
  client-id: ${ssafy.client-id}
  secret-key: ${ssafy.secret-key}
  base_url: ${ssafy.base-url}
  redirect_url: ${ssafy.redirect-url}
```

- `application-deploy-2.yml` → 배포 서버 2

```
spring:
  application:
    name: vibeEditorProject
  cloud:
    aws:
      region:
        static: ${AWS_REGION}
      credentials:
        access-key: ${AWS_ACCESS_KEY_ID}
        secret-key: ${AWS_SECRET_ACCESS_KEY}
  config:
    import:
      - optional:file:.env[.properties]
      - aws-parameterstore:/vibeeditor/

  profiles:
    active: prod

  session:
    redis:
      namespace: vibe-editor

  datasource:
    url: ${datasource.url-2}
    username: ${datasource.username}
    password: ${datasource.password}
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: false
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQLDialect
  data:
```

```
  mongodb:
    uri: ${mongodb.url-2}
  redis:
    host: ${redis.host2}
    port: ${redis.port}
    password: ${redis.password}

security:
  user:
    name: ${security.user-name}
    password: ${security.user-password}
    roles: ${security.user-role}

  cors:
    allowed-origins: ${security.cors-allow-origin}

  jwt:
    secret-key: ${security.jwt-secret-key}
    access-token-expiration: ${security.jwt-access.expiration}
    refresh-token-expiration: ${security.jwt-refresh.expiration}
    pass-urls: ${security.jwt-pass-urls}
    admin-urls: ${security.jwt-admin-urls}

  oauth2:
    client:
      registration:
        google:
          client-name: google
          client-id: ${google.client-id}
          client-secret: ${google.client-secret}
          redirect-uri: ${google.redirect-url}
          authorization-grant-type: authorization_code
          scope:
            - email
            - profile
        github:
          client-name: Github
          client-id: ${github.client-id}
```

```
        client-secret: ${github.client-secret}
        redirect-uri: ${github.redirect-url}
        authorization-grant-type: authorization_code
        scope:
          - read:user
          - user:email

      provider:
        google:
          authorization-uri: https://accounts.google.com/o/oauth2/auth
          token-uri: https://oauth2.googleapis.com/token
          user-info-uri: https://openidconnect.googleapis.com/v1/userinfo
          user-name-attribute: sub
        github:
          authorization-uri: https://github.com/login/oauth/authorize
          token-uri: https://github.com/login/oauth/access_token
          user-info-uri: https://api.github.com/user
          user-name-attribute: id

servlet:
  multipart:
    enabled: true
    max-file-size: 20MB
    max-request-size: 50MB

ai:
  openai:
    api-key: ${gpt.api-key}
    base-url: ${gpt.api-url}
    model: ${gpt.model}

  anthropic:
    api-key: ${claude.api-key}
    chat:
      options:
        temperature: ${claude.temperature}
        model: ${claude.model}
        max-tokens: ${claude.max-token}
```

```
      base-url: ${claude.baseurl}

  lifecycle:
    timeout-per-shutdown-phase: 60s

server:
  shutdown: graceful

app:
  cors:
    allow-hosts:
      - ${allow-host.front}
      - ${allow-host.local}

springdoc:
  api-docs:
    path: /v3/api-docs
    version: openapi_3_0

logging:
  file:
    name: /var/log/vibe/vibe.log
  level:
    root: INFO

decorator:
  datasource:
    p6spy:
      enable-logging: false
notion:
  api:
    version: ${notion.version}
    base_url: ${notion.baseurl}

management:
  endpoints:
    access:
      default: none
```

```
    web:
      exposure:
        include: health,info,prometheus
      base-path: /api
  endpoint:
    health:
      show-details: always
      access: unrestricted
    prometheus:
      access: unrestricted

aes-algo: ${security.aes-algorithm}
aes-secret-key: ${security.aes-key}

ssafy:
  client-id: ${ssafy.client-id}
  secret-key: ${ssafy.secret-key}
  base_url: ${ssafy.base-url}
  redirect_url: ${ssafy.redirect-url}
```

5. CI/CD 및 배포

AWS EC2 -1

Nginx : 80/443

- 포트 번호
  - Jenkins : 9090
  - Backend : 8080
  - Frontend : 3000

AWS EC2 -2

- Nginx : 80/443
- 포트 번호
  - Mysql : 3306
  - Backend : 8080

    - Kibana : 5601
    - logstash : 9600
    - ElasticSearch : 9200
    - Prometheus : 9091
    - Grafana : 9092
    - Node - Exporter : 9100
    - Frontend : 3000
- Dokcer
    - Frontend

```
# ⚡ 1. React(Next.js) 빌드 단계
FROM node:18-alpine AS builder

WORKDIR /app

# package.json과 package-lock.json만 복사 후 의존성 설치 (최적화)
COPY package.json package-lock.json ./
RUN npm ci --production

# Next.js 소스 코드 복사 후 빌드 실행
COPY . .
RUN npm run build

# 🏃 2. 배포용 컨테이너 (Next.js 서버 실행)
FROM node:18-alpine

WORKDIR /app

# 빌드된 Next.js 파일과 필요한 패키지만 복사
COPY --from=builder /app/package.json ./
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/public ./public

EXPOSE 3000
```

```
# Next.js 프로덕션 실행
CMD ["npx", "next", "start"]
```

- Backend

```
FROM openjdk:21-slim AS builder

RUN apt-get update && apt-get install -y \
    unzip \
 && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY gradlew settings.gradle build.gradle ./
COPY gradle/ gradle/

RUN chmod +x gradlew

COPY . .

RUN ./gradlew --no-daemon clean build -x test

FROM openjdk:21-slim AS runner

WORKDIR /app

COPY --from=builder /app/build/libs/*.jar app.jar
COPY src/main/resources/application-deploy.yml /app/application-de

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app/app.jar", "--spring.config.location=
```

- Jenkins

```
FROM jenkins/jenkins:lts-jdk21
```

```
USER root

# 패키지 리스트 업데이트 및 Docker CLI, AWS CLI 설치
RUN apt-get update && apt-get install -y docker.io wget curl unzip \
    && curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip
    && unzip awscliv2.zip \
    && ./aws/install \
    && rm -rf awscliv2.zip aws

# Jenkins 실행 사용자로 변경
USER jenkins

# 기본 실행 명령어
CMD ["/usr/local/bin/jenkins.sh"]
```

도커 명령어 관련

- 도커 빌드 명령어

```
docker build -t main-container .
# main-container 라는 태그 지정.
```

- 프로젝트 스프링 이미지 실행 명령어

```
docker run -p 8080:8080 --env-file .env main-container
```

- 젠킨스 컨테이너 실행 명령어

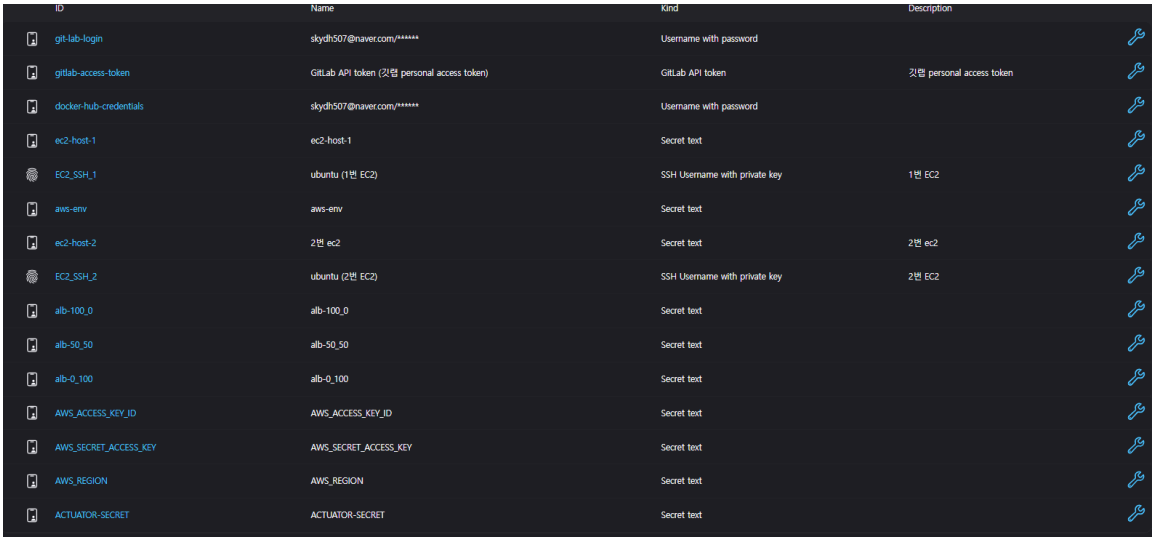
```
docker run -d -p 9090:8080 -e TZ=Asia/Seoul -e JENKINS_OPTS="--pref
docker exec -u root -it custom-jenkins bash
apt-get install -y docker.io
```

- mysql 실행 명령어

```
### Mysql 도커 컨테이너
docker run -d   --name mysql-container   -e MYSQL_ROOT_PASSWORD=p
docker exec -it container mysql -u root -p
```

Jenkins 세팅

- Credentials



ID	Name	Kind	Description
git-lab-login	skydh507@naver.com/******	Username with password	
gitlab-access-token	GitLab API token (깃랩 personal access token)	GitLab API token	깃랩 personal access token
docker-hub-credentials	skydh507@naver.com/******	Username with password	
ec2-host-1	ec2-host-1	Secret text	
EC2_SSH_1	ubuntu (1번 EC2)	SSH Username with private key	1번 EC2
aws-env	aws-env	Secret text	
ec2-host-2	2번 ec2	Secret text	2번 ec2
EC2_SSH_2	ubuntu (2번 EC2)	SSH Username with private key	2번 EC2
alb-100_0	alb-100_0	Secret text	
alb-50_50	alb-50_50	Secret text	
alb-0_100	alb-0_100	Secret text	
AWS_ACCESS_KEY_ID	AWS_ACCESS_KEY_ID	Secret text	
AWS_SECRET_ACCESS_KEY	AWS_SECRET_ACCESS_KEY	Secret text	
AWS_REGION	AWS_REGION	Secret text	
ACTUATOR-SECRET	ACTUATOR-SECRET	Secret text	

- CI 파이프라인

```
pipeline {
    agent any

    environment {
        CREDENTIALS_ID = 'git-lab-login'
        PROJECT_NAME = 'S12P31A503'
        CONFLICT_DETECTED = "false"
    }

    tools {
        jdk 'jdk21'
        nodejs 'node'
    }

    stages {
        stage('Initialize GitLab Status') {
            steps {
                script {
                    updateGitlabCommitStatus name: 'Jenkins Pipeline', state: 'pen
                }
            }
```

```
        }
        stage('Clean Workspace') {
          steps {
            echo "🧹 Cleaning workspace..."
            sh 'git reset --hard && git clean -fdx'
          }
        }

        stage('Detect MR Branches') {
          steps {
            script {
              updateGitlabCommitStatus name: 'Detect MR Branches', state

              env.SOURCE_BRANCH = env.gitlabSourceBranch ?: env.GIT_B

              if (env.gitlabTargetBranch) {
                env.TARGET_BRANCH = env.gitlabTargetBranch
              } else {
                if (env.SOURCE_BRANCH == "dev-be" || env.SOURCE_BRAI
                  env.TARGET_BRANCH = env.SOURCE_BRANCH
                } else {
                  env.TARGET_BRANCH = "dev-be"
                }
              }

              echo "🔍 Detected Source Branch: ${env.SOURCE_BRANCH}"
              echo "🔍 Detected Target Branch: ${env.TARGET_BRANCH}"

              if (!env.SOURCE_BRANCH || !env.TARGET_BRANCH) {
                error("❌ 소스/대상 브랜치가 감지되지 않았습니다.")
              }

              updateGitlabCommitStatus name: 'Detect MR Branches', state
            }
          }
          post {
            failure {
              script {
```

```
                updateGitlabCommitStatus name: 'Detect MR Branches', sta
                mattermostSend(
                    message: """\
❌ *Detect MR Branches Failed!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
"""
                )
            }
        }
    }

    stage('Clean Conflicting Branches') {
        steps {
            echo "🧹 Cleaning up stale local refs that may cause branch con
            sh 'git remote prune origin || true'
        }
    }

    stage('Merge Conflict Check') {
        when {
            expression { env.gitlabSourceBranch }
        }
        steps {
            script {
                updateGitlabCommitStatus name: 'Merge Conflict Check', stat

                echo "🔄 Checking out repository for merge conflict check..."
                git branch: env.SOURCE_BRANCH,
                    credentialsId: env.CREDENTIALS_ID,
                    url: 'https://lab.ssafy.com/s12-final/S12P31A503.git'

                echo "🔄 Checking for merge conflicts: ${env.SOURCE_BRAN

                withCredentials([usernamePassword(
```

```
                    credentialsId: env.CREDENTIALS_ID,
                    usernameVariable: 'GIT_USER',
                    passwordVariable: 'GIT_PASS'
                )]) {
                    sh """
git config credential.helper "!f() { echo username=\\"$GIT_USER\\"; echo p
git config user.email "jenkins@example.com"
git config user.name "Jenkins"
git fetch https://lab.ssafy.com/s12-final/S12P31A503.git ${env.TARGET_BR
git merge --no-commit --no-ff origin/${env.TARGET_BRANCH} || echo 'CO
"""
                }

                def mergeResult = sh(script: "git diff --name-only --diff-filter=
                if (mergeResult) {
                    echo "❌ Merge conflict detected!"
                    env.CONFLICT_DETECTED = "true"
                    updateGitlabCommitStatus name: 'Merge Conflict Check', st
                    currentBuild.result = 'FAILURE'
                    error("❌ Merge conflict detected! Resolve conflicts before
                }

                updateGitlabCommitStatus name: 'Merge Conflict Check', stat
            }
        }
        post {
            failure {
                script {
                    mattermostSend(
                        message: """\
❌ *Merge Conflict Detected!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
👉 Please resolve merge conflicts and try again.
"""
                    )
```

```
            }
          }
        }
      }

      stage('Build & Unit Test (Backend)') {
        when {
          expression {
            env.CONFLICT_DETECTED == "false" && env.TARGET_BRANC
          }
        }
        steps {
          script {
            updateGitlabCommitStatus name: 'Backend Build', state: 'penc

            echo "🛠️ Running backend (Spring) build & unit tests..."
            dir('backend') {
              sh '''
chmod +x ./gradlew
./gradlew build -x test
'''
            }

            updateGitlabCommitStatus name: 'Backend Build', state: 'succ
          }
        }
        post {
          failure {
            script {
              updateGitlabCommitStatus name: 'Backend Build', state: 'fai
              mattermostSend(
                message: """\
❌ *Backend Build & Unit Test Failed!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
"""
```

```
                )
            }
        }
    }

    stage('Build & Unit Test (Frontend)') {
        when {
            expression {
                env.CONFLICT_DETECTED == "false" && env.TARGET_BRANC
            }
        }
        steps {
            script {
                updateGitlabCommitStatus name: 'Frontend Build', state: 'penc

                echo "🛠️ Running frontend (npm) build & unit tests..."
                dir('extension/vibe-editor') {
                    sh '''
npm install
npm run compile
'''
                }

                updateGitlabCommitStatus name: 'Frontend Build', state: 'succ
            }
        }
        post {
            failure {
                script {
                    updateGitlabCommitStatus name: 'Frontend Build', state: 'fa
                    mattermostSend(
                        message: """\
❌ *Frontend Build & Unit Test Failed!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
```

```
"""
                )
            }
        }
    }
  }

  post {
    success {
      script {
        updateGitlabCommitStatus name: 'Jenkins Pipeline', state: 'succe
        echo "✅ Build & Tests Passed!"
        mattermostSend(
          message: """\
✅ *Jenkins Build & Test Success!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
"""
        )
      }
    }
    failure {
      script {
        updateGitlabCommitStatus name: 'Jenkins Pipeline', state: 'failed
        echo "❌ Build or Test Failed."
        mattermostSend(
          message: """\
❌ *Jenkins Build & Test Failed!*
Source: ${env.SOURCE_BRANCH}
Target: ${env.TARGET_BRANCH}
Project: ${env.PROJECT_NAME}
Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
"""
        )
      }
```

```
        }
        aborted {
            script {
                echo "⚠️ Build Aborted."
            }
        }
    }
}
```

- CD 파이프라인
  - Frontend

```
pipeline {
    agent any

    tools {
        jdk 'jdk21'
        nodejs 'node'
    }

    environment {
        CREDENTIALS_ID = 'git-lab-login'
        DOCKER_CREDENTIALS_ID = 'docker-hub-credentials'
        DOCKERHUB_REPO = "stussyhunter"
        FRONTEND_IMAGE = "vibeeditor-frontend"
        PROJECT_NAME = 'S12P31A503'
    }

    stages {
        stage('Cleanup old refs') {
            steps {
                script {
                    echo "🧹 오래된 Git 레퍼런스 정리"
                    sh 'rm -rf .git || true'
                    sh 'git init'
                    sh 'git remote add origin https://lab.ssafy.com/s12-final/S1
                    sh 'git remote prune origin || true'
                }
```

```
    }
}

stage('Checkout') {
    steps {
        git branch: 'dev-fe',
            credentialsId: "${CREDENTIALS_ID}",
            url: 'https://lab.ssafy.com/s12-final/S12P31A503.git'
    }
}
stage('Clean Frontend Cache') {
    steps {
        script {
            dir('frontend') {
                sh 'rm -rf .next .turbo node_modules/.cache dist'
            }
        }
    }
}
stage('Test Build') {
    steps {
        script {
            echo "🛠️ Next.js 빌드 테스트"
            dir('frontend') {
                sh "npm install"
                sh "npm run build"
            }
        }
    }
}

stage('Test Docker Hub Login') {
    steps {
        script {
            echo "✅ Docker Hub 로그인 테스트"
            docker.withRegistry('https://index.docker.io/v1/', "${DOCK
                echo "🔑 Docker Hub 로그인 성공!"
            }
```

```
        }
    }
}

stage('Build & Push Test Image') {
    steps {
        script {
            def testImage = "${DOCKERHUB_REPO}/${FRONTEND_IM
            echo "✅ 테스트용 Docker 이미지 빌드 & 푸시"
            docker.withRegistry('https://index.docker.io/v1/', "${DOCK
                def builtImage = docker.build("${testImage}", "-f fronter
                builtImage.push('latest')
            }
        }
    }
}

stage('Deploy to EC2s') {
    parallel {
        stage('Deploy to EC2-1') {
            steps {
                script {
                    try {
                        deployToEC2('EC2_SSH_1', 'ec2-host-1', 'EC2-1')
                    } catch (Exception e) {
                        mattermostSend(
                            message: """\
                                ❌ *EC2-1 배포 실패!*
                                Project: ${PROJECT_NAME}
                                Reason: ${e.getMessage()}
                                Time: ${new Date().format("yyyy-MM-dd HH:r
                            """.stripIndent()
                        )
                        error("EC2-1 배포 실패")
                    }
                }
            }
        }
```

```
            stage('Deploy to EC2-2') {
                steps {
                    script {
                        try {
                            deployToEC2('EC2_SSH_2', 'ec2-host-2', 'EC2-2')
                        } catch (Exception e) {
                            mattermostSend(
                                message: """\
                                    ❌ *EC2-2 배포 실패!*
                                    Project: ${PROJECT_NAME}
                                    Reason: ${e.getMessage()}
                                    Time: ${new Date().format("yyyy-MM-dd HH:r
                                """.stripIndent()
                            )
                            error("EC2-2 배포 실패")
                        }
                    }
                }
            }
        }
    }

    post {
        success {
            script {
                mattermostSend(
                    message: """\
                        ✅ *전체 배포 성공!*
                        Project: ${PROJECT_NAME}
                        Image: ${DOCKERHUB_REPO}/${FRONTEND_IMAGE}:la
                        Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
                    """.stripIndent()
                )
            }
        }
    }
}
```

```
def deployToEC2(sshKeyId, hostId, displayName) {
    withCredentials([
        sshUserPrivateKey(credentialsId: sshKeyId, keyFileVariable: 'KEY
        string(credentialsId: hostId, variable: 'HOST')
    ]) {
        echo "🚀 ${displayName} 배포 시작:"
        echo "🔑 SSH 키 권한 설정 중..."
        def keyPath = "${KEY}"
        def hostName = "${HOST}"
        sh """
            chmod 400 "${keyPath}"
            ssh -i "${keyPath}" -o StrictHostKeyChecking=no ubuntu@${h
                echo "[EC2] 🐳 Docker 컨테이너 재시작 중..."
                docker pull ${DOCKERHUB_REPO}/${FRONTEND_IMAGE}:lat
                docker stop -t 70 ${FRONTEND_IMAGE} || true
                docker rm ${FRONTEND_IMAGE} || true
                docker run -d --name ${FRONTEND_IMAGE} \\
                    -p 3000:3000 \\
                    ${DOCKERHUB_REPO}/${FRONTEND_IMAGE}:latest
                echo "⏳ 컨테이너 초기화 대기 (10초)"
                sleep 3
                echo "🔍 실행 중인 컨테이너 확인"
                docker ps | grep ${FRONTEND_IMAGE} > /dev/null 2>&1
                if [ \$? -eq 0 ]; then
                    echo "✅ 프론트엔드 컨테이너가 정상적으로 실행 중입니다."
                else
                    echo "❌ 프론트엔드 컨테이너 실행 실패: 배포에 문제가 발생했습
                exit 1
                fi
EOF
        """.stripIndent()

        echo "✅ ${displayName} 배포 완료"
    }
}
```

- Backend

```
pipeline {
    agent any

    tools {
        jdk 'jdk21'
    }

    environment {
        CREDENTIALS_ID = 'git-lab-login'
        DOCKER_CREDENTIALS_ID = 'docker-hub-credentials'
        DOCKERHUB_REPO = "stussyhunter"
        TEST_IMAGE_NAME = "vibeeditor-backend"
        PROJECT_NAME = 'S12P31A503'
        AWS_ACCESS_KEY_ID = credentials('AWS_ACCESS_KEY_ID')
        AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCES
        AWS_REGION = credentials('AWS_REGION')
        ALB_100_0 = credentials('alb-100_0')
        ALB_50_50 = credentials('alb-50_50')
        ALB_0_100 = credentials('alb-0_100')
        EC2_2_IP = credentials('ec2-host-2')
        EC2_1_IP = credentials('ec2-host-1')
        // actuator_secret = credentials('ACTUATOR_SECRET')
    }

    stages {
        stage('Git Cleanup & Checkout') {
            steps {
                script {
                    sh 'rm -rf .git || true'
                    git branch: 'dev-be',
                        credentialsId: "${CREDENTIALS_ID}",
                        url: 'https://lab.ssafy.com/s12-final/S12P31A503.git'
                }
            }
        }

        stage('Build Project') {
            steps {
```

```
            dir('backend') {
                withCredentials([string(credentialsId: 'aws-env', variable: '
                    sh '''
                        echo "$ENV_CONTENT" > .env
                        export $(grep -v '^#' .env | xargs)
                        chmod +x ./gradlew
                        ./gradlew clean build -x test -Dspring.profiles.active=
                    '''
                }
            }
        }
    }

    stage('Test Docker Hub Login') {
        steps {
            script {
                echo "✅ Docker Hub 로그인 테스트"
                docker.withRegistry('https://index.docker.io/v1/', "${DOCK
                    echo "🔑 Docker Hub 로그인 성공!"
                }
            }
        }
    }

    stage('Docker Build & Push') {
        steps {
            script {
                def image = "${DOCKERHUB_REPO}/${TEST_IMAGE_NAM
                docker.withRegistry('https://index.docker.io/v1/', "${DOCK
                    def builtImage = docker.build("${image}", "-f backend/D
                    builtImage.push('latest')
                }
            }
        }
    }

    stage('Use AWS CLI with Secret Text') {
        steps {
```

```
            withCredentials([string(credentialsId: 'aws-env', variable: 'EN
                sh '''
                echo "$ENV_AWS" > aws-env.sh
                bash -c "
                set -e
                source aws-env.sh
                aws sts get-caller-identity
                "
                '''
            }
        }
    }

    stage('All traffic to ec2-1') {
        steps {
            sh '''
            echo "$AWS_ENV" > aws-env.sh

            bash -c "
            source aws-env.sh
            echo $ALB_100_0 | base64 -d > alb-100-0.json
            "
            '''
            echo '==== JSON 내용 ===='

            sh '''
            cat alb-100-0.json
            aws elbv2 modify-listener \
            --listener-arn arn:aws:elasticloadbalancing:ap-northeast-2:9
            --cli-input-json file://alb-100-0.json
            '''
        }

    }

    stage('Deploy to EC2-2') {
        steps {
            script {
```

```
                deployToEC2('EC2_SSH_2', 'ec2-host-2', 'EC2-2')
            }
        }
    }

    stage('Health Check EC2-2') {
        steps {
            script {
                withCredentials([string(credentialsId: 'ACTUATOR-SECRET
                    def MAX_RETRIES = 12
                    def HEALTH_URL = "http://${EC2_2_IP}/api/health"
                    def success = false

                    for (int i = 0; i < MAX_RETRIES; i++) {
                        echo "[$i/$MAX_RETRIES] Checking health at ec2-2:

                        def responseCode = sh(
                            script: "curl -u admin:${ACTUATOR_SECRET} --ma
                            returnStdout: true
                        ).trim()

                        echo "Response code: ${responseCode}"

                        if (responseCode == '200') {
                            echo "✅ Health check succeeded."
                            success = true
                            break
                        }

                        sleep(time: 5, unit: 'SECONDS')
                    }

                    if (!success) {
                        error "❌ Health check failed after ${MAX_RETRIES} a
                    }
                }
            }
        }
```

```
        }


        stage('All traffic to ec2-2') {
            steps {
                sh '''
                echo "$AWS_ENV" > aws-env.sh

                bash -c "
                source aws-env.sh
                echo $ALB_0_100 | base64 -d > alb-0-100.json
                "
                '''
                echo '==== JSON 내용 ===='

                sh '''
                cat alb-0-100.json
                aws elbv2 modify-listener \
                --listener-arn arn:aws:elasticloadbalancing:ap-northeast-2:9
                --cli-input-json file://alb-0-100.json
                '''
            }
        }

        stage('Deploy to EC2-1') {
            steps {
                script {
                    deployToEC2('EC2_SSH_1', 'ec2-host-1', 'EC2-1')
                }
            }
        }


        stage('Health Check EC2-1') {
            steps {
                script {
                    withCredentials([string(credentialsId: 'ACTUATOR-SECRET
                        def MAX_RETRIES = 12
```

```
                def HEALTH_URL = "http://${EC2_1_IP}/api/health"
                def success = false

                for (int i = 0; i < MAX_RETRIES; i++) {
                    echo "[$i/$MAX_RETRIES] Checking health at ec2-1:

                    def responseCode = sh(
                        script: "curl -u admin:${ACTUATOR_SECRET} --ma
                        returnStdout: true
                    ).trim()

                    echo "Response code: ${responseCode}"

                    if (responseCode == '200') {
                        echo "✅ Health check succeeded."
                        success = true
                        break
                    }

                    sleep(time: 5, unit: 'SECONDS')
                }

                if (!success) {
                    error "❌ Health check failed after ${MAX_RETRIES} a
                }
            }
        }
    }
}


stage('Traffic split 5:5') {
    steps {
        sh '''
        echo "$AWS_ENV" > aws-env.sh

        bash -c "
        source aws-env.sh
```

```
            echo $ALB_50_50 | base64 -d > alb-50-50.json
            "
            '''
            echo '==== JSON 내용 ===='

            sh '''
            cat alb-50-50.json
            aws elbv2 modify-listener \
            --listener-arn arn:aws:elasticloadbalancing:ap-northeast-2:9
            --cli-input-json file://alb-50-50.json
            '''
        }
    }

}

post {
    success {
        script {
            mattermostSend(
                message: """\
                    ✅ *무중단 배포 성공!*
                    Project: ${PROJECT_NAME}
                    Image: ${DOCKERHUB_REPO}/${TEST_IMAGE_NAME}:l
                    Time: ${new Date().format("yyyy-MM-dd HH:mm:ss")}
                """.stripIndent()
            )
        }
    }
}
}

def deployToEC2(sshKeyId, hostId, displayName) {
    withCredentials([
        sshUserPrivateKey(credentialsId: sshKeyId, keyFileVariable: 'SSH
        string(credentialsId: hostId, variable: 'EC2_HOST')
    ]) {
        def result = sh(
```

```
            script: """
                echo "🚀 ${displayName} 배포 시작"
                chmod 400 \$SSH_KEY_PATH
                ssh -i \$SSH_KEY_PATH -o StrictHostKeyChecking=no ubunt
                docker ps -a
docker pull ${DOCKERHUB_REPO}/${TEST_IMAGE_NAME}:latest
docker stop -t 70 ${TEST_IMAGE_NAME} || true
docker rm ${TEST_IMAGE_NAME} || true
docker ps -a
docker run -d --name ${TEST_IMAGE_NAME} \\
  -p 8080:8080 \\
  --env-file .env \\
  -v /home/ubuntu/application-deploy.yml:/app/application-deploy.ym
  -v /var/log/vibe:/var/log/vibe \\
  ${DOCKERHUB_REPO}/${TEST_IMAGE_NAME}:latest
EOF
            """,
            returnStatus: true
        )
        if (result != 0) {
            error("${displayName} 컨테이너 실행 실패")
        }
    }
}

def waitForHealthCheck(instanceName) {
    def retries = 10
    def delay = 6
    echo "🔍 ${instanceName} 헬스체크 확인 시작..."
    for (int i = 0; i < retries; i++) {
        def status = sh(
            script: "curl -s -o /dev/null -w '%{http_code}' ${HEALTH_CHEC
            returnStdout: true
        ).trim()
        if (status == "200") {
            echo "✅ ${instanceName} 헬스체크 통과"
            return
        }
```

```
        echo "⏳ ${instanceName} 아직 준비 안됨. 다시 시도 (${i + 1}/${retr
        sleep delay
    }
    error("❌ ${instanceName} 헬스체크 실패")
}
```