포팅 메뉴얼

목차

1. 사용 도구

- 이슈 관리 : Jira
- 형상 관리 : GitLab
- 커뮤니케이션 : Notion, MatterMost, discord
- 디자인 : Figma
- CI/CD : Jenkins

2. 개발 환경

- SrpingBoot : 3.3.6
- JVM : 22.0.1
- react : 18.3.1
- axios : 1.7.7
- Visual Studio Code : 1.91.1
- Intellij : 2024.1.4
- EC2 Server : Ubuntu 20.04.6 LTS
- DB : mySQL 8.0.38
- Reddis : 7.2.5

3. 환경변수

ssenbi.env

rds.database=ssenbidb
rds.host=ssenbi-rds-mysql.cfmuckwc4t6p.ap-northeast-2.rds.amazonaws.com
rds.username=ssenbi
rds.pwd=ssenbi1234

# maria db
test.mariadb.database=S11P31A109
test.mariadb.host=stg-yswa-kr-practice-db-master.mariadb.database.azure.com
test.mariadb.username=S11P31A109@stg-yswa-kr-practice-db-master
test.mariadb.pwd=dtdqmOXoxz

# local
local.database=bobidb
local.host=localhost
local.username=root
local.pwd=root

# redis
redis.host=redis
redis.port=6379

# coolsms
COOLSMS_API_KEY=NCS2Z7U5ZVFBV0KX
COOLSMS_API_SECRET=HNPQE4ERCYPCYZFXSENT5PFRJZVCO846
COOLSMS_SENDER=01054621615

# openai
OPENAI_API_KEY=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
OPENAI_API_URL=https://api.openai.com/v1
OPENAI_PROMPT_ROLE=너는 이제부터 메시지 템플릿 제작자야

OPENAI_PROMPT_ROLE2=너의 설명은 필요 없어 메시지만 제공해줘
OPENAI_PROMPT_ROLE3=이름을 템플릿에 삽입할 경우 [[고객명]] 이런 형식으로 삽입해줘
OPENAI_PROMPT_ROLE4=공통으로 반드시 한개의 메시지만을 만들어줘 개별적으로
메시지를 만들면 안돼

application.yaml

```
server:
  servlet:
    context-path: /api/v1/ssenbi

spring:
  profiles:
    active: dev
  config:
    import: optional:file:.env[.properties]

springdoc:
  packages-to-scan: com.haneolenae.bobi
  default-consumes-media-type: application/json;charset=UTF-8
  default-produces-media-type: application/json;charset=UTF-8
  swagger-ui:
    path: /swagger-ui.html
    disable-swagger-default-url: true
    display-request-duration: true
    operations-sorter: alpha
```

4. 배포

개요

docker container 는 8 개로 관리하고 있습니다.

- docker container 들은 my-network-bridge 내부 네트워크로 통신합니다.
- jenkins, nginx, frontend, backend-ssenbii, backend-cardcompany, mysql-ssenbi, redis
  GitLab 의 2 개의 브랜치를 추적하여, CI/CD 를 구축하였습니다.
- frontend, backend

springboot

- jenkins pipeline 에서 env 파일을 주입하며, dockerfile 을 통해 docker image 를 생성합니다.
- dockerhub 에 docker image 를 업로드하며, 해당 image 를 이용하여 컨테이너를 생성합니다.

react

- jenkins pipeline 에서 env 파일을 주입하며, dockerfile 을 통해 docker image 를 생성합니다.
- react-container 에는 웹서버로 nginx 를 사용하고 있습니다.

database

- docker-compose.yml 파일을 이용하여 mysql 과 redis 를 docker 로 관리하고 있습니다
  - ex) docker-compose up -d redis

docker-compose.yml

```
version: '3'

services:
  db:
    image: mysql:latest
    container_name: milli_db
    environment:
      MYSQL_DATABASE: 'millidb'
      MYSQL_USER: 'ssafy'
      MYSQL_PASSWORD: '1234'
      MYSQL_ROOT_PASSWORD: '1234'
    ports:
      - '3307:3306'
    volumes:
      - 'mysqldata:/var/lib/mysql'
        # - './paymilli_init.sql:/docker-entrypoint-initdb.d/init.sql'
```

```
  networks:
    - my-bridge-network

db2:
  image: mysql:latest
  container_name: cardcompany_db
  environment:
    MYSQL_DATABASE: 'cardcompanydb'
    MYSQL_USER: 'ssafy'
    MYSQL_PASSWORD: '1234'
    MYSQL_ROOT_PASSWORD: '1234'
  ports:
    - '3308:3306'
  volumes:
    - 'companydata:/var/lib/mysql'
      # - './cardcompany_init.sql:/docker-entrypoint-initdb.d/init.sql'
  networks:
    - my-bridge-network

redis:
  image: redis:latest
  container_name: redis-container
  volumes:
    - redisdata:/data
  networks:
    - my-bridge-network

nginx:
  image: nginx-image:latest
  container_name: nginx-container
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt
```

```
    environment:
      - TZ=Asia/Seoul
    networks:
      - my-bridge-network

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins-container
    ports:
      - "9090:8080"
      - "50000:50000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - jenkins_home:/var/jenkins_home
    user: root
    environment:
      - JENKINS_OPTS=--httpPort=8080
      - TZ=Asia/Seoul

volumes:
  mysqldata:
  companydata:
  redisdata:
  jenkins_home:

networks:
  my-bridge-network:
    external: true

master:
  image: redis:latest
  container_name: master
  volumes:
    - /etc/redis/master.conf:/usr/local/etc/redis/redis.conf
  command: redis-server /usr/local/etc/redis/redis.conf
```

```
    ports:
      - "6379:6379"
      - "6380:6380"
      - "6381:6381"
      - "5000:5000"
      - "5001:5001"
      - "5002:5002"
  redis-1:
    image: redis:latest
    network_mode: "service:master"
    container_name: slave-1
    volumes:
      - /etc/redis/slave:/slave
    command: redis-server /slave/slave-1.conf

  redis-2:
    network_mode: "service:master"
    image: redis:latest
    container_name: slave-2
    volumes:
      - /etc/redis/slave:/slave
    command: redis-server /slave/slave-2.conf

  sentinel-1:
    network_mode: "service:master"
    image: redis:latest
    container_name: sentinel-1
    volumes:
      - /etc/redis/sentinel:/sentinel
    command: redis-server /sentinel/sentinel-1.conf --sentinel
    depends_on:
      - master
  sentinel-2:
    network_mode: "service:master"
    image: redis:latest
```

```
    container_name: sentinel-2
    volumes:
      - /etc/redis/sentinel:/sentinel
    command: redis-server /sentinel/sentinel-2.conf --sentinel
    depends_on:
      - master
  sentinel-3:
    network_mode: "service:master"
    image: redis:latest
    container_name: sentinel-3
    volumes:
      - /etc/redis/sentinel:/sentinel
    command: redis-server /sentinel/sentinel-3.conf --sentinel
    depends_on:
      - master
```

Nginx 설정

nginx.conf

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
```

```
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

conf.d/default.conf

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

    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root    /usr/share/nginx/html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;
```

```
    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ ₩.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ ₩.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /₩.ht {
    #    deny  all;
    #}
}
```

site-available/ssenbi.conf

```
server {
```

```
    listen 9090;
    listen [::]:9090;

    server_name k11a109.p.ssafy.io;

  # gitlab to jenkins webhook
    location /project/ {
        proxy_pass http://k11a109.p.ssafy.io:9090/project/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}



server {
    listen 80;
    listen [::]:80;
    # server_name k11a109.p.ssafy.io;
    server_name www.ssenbi.co.kr;

    access_log /var/log/nginx/access.log main;
    error_log /var/log/nginx/error.log warn;

    # ssenbi
    location /api/v1/ssenbi/ {
        proxy_pass http://ssenbi-backend:8080;  # Proxy to Spring Boot container
        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;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

5. CI/CD 구축

jenkins 설정

backend 파이프라인

```
pipeline {
    agent any

    environment {
        // credentials
        ENV_CREDENTIALS = credentials('ssenbi-env')



        // git
        GIT_URL = "https://lab.ssafy.com/s11-final/S11P31A109.git"
        TRACKING_BRANCH = "backend"
        PROJ_DIR = "backend/Bobi"

        // docker
        IMAGE_NAME = "haneolenae/ssenbi:1.0"
        CONTAINER_NAME = "ssenbi-backend"
        DOCKER_BRIDGE = "ssenbi-bridge"
    }

    stages {



        stage('Clean Workspace') {
            steps {
                cleanWs()
            }
        }

        stage('Checkout Application Git Branch') {
            steps {
```

```
            echo "Checkout Application Git Branch
==================================================="
            git credentialsId: 'gitlab-cred',
            url:"${GIT_URL}",
            branch: "${TRACKING_BRANCH}"
        }
    }


    stage('Remove Old Docker Images') {
        steps {
            echo "Removing Old Docker Images
==================================================="
            sh '''
                docker image prune -a -f
            '''
        }
    }


    stage('.env file setting') {
        steps{
            echo ".env file setting
==================================================="
            dir(path: "${PROJ_DIR}") {
                sh '''
                    chmod -R 755 .
                    cp $ENV_CREDENTIALS .env
                  '''
            }
        }
    }


    stage('BE-Build') {
```

```
        steps {
            echo "BE-Build
==================================================="
            script {
                // 작업 디렉토리가 존재하는지 확인
                if (fileExists("${PROJ_DIR}")) {
                    dir("${PROJ_DIR}") {
                        // gradlew 가 실행 가능한지 확인하고 빌드
                        sh 'chmod +x gradlew'
                        sh './gradlew clean build -x test'
                    }
                } else {
                    error "Directory ${PROJ_DIR} does not exist."
                }
            }
        }
    }


    stage('Stop and Remove Existing Container') {
        steps {
            echo "Stopping and Removing Existing Container
==================================================="
            sh '''
                if [ $(docker ps -aq -f name=${CONTAINER_NAME}) ]; then
                    echo "Stopping and removing existing container"
                    docker stop ${CONTAINER_NAME} || true
                    docker rm ${CONTAINER_NAME} || true
                fi
            '''
        }
    }
```

```
        stage('Docker Image Build') {
            steps {
                echo "Docker Image Build
==================================================="
                script {
                    if (fileExists("${PROJ_DIR}")) {
                        dir("${PROJ_DIR}") {
                            sh 'docker build -t $IMAGE_NAME .'
                        }
                    } else {
                        error "Directory '${PROJ_DIR}' does not exist."
                    }
                }
            }
        }



        stage('Deploy Docker Container') {
            steps {
                echo "Deploy Docker Container
==================================================="
                script {
                    // 컨테이너가 존재하면 중지 및 삭제
                    sh '''
                        if [ $(docker ps -aq -f name=${CONTAINER_NAME}) ]; then
                            echo "Stopping and removing existing container"
                            docker stop ${CONTAINER_NAME} || true
                            docker rm ${CONTAINER_NAME} || true
                        fi
                    '''

                    // 컨테이너 실행
                    sh '''
```

```
                    docker run -d --name ${CONTAINER_NAME} ₩
                    -e TZ=Asia/Seoul ₩
                    --net ${DOCKER_BRIDGE} ₩
                    -p 8080:8080 ₩
                    $IMAGE_NAME
                '''
            }
        }
    }


}
```

frontend 파이프라인

```
pipeline {
    agent any

    tools {
        nodejs 'nodejs22'
    }

    environment {
        // git
        GIT_URL = "https://lab.ssafy.com/s11-final/S11P31A109.git"
        TRACKING_BRANCH = "frontend"
        PROJ_DIR = "."

        // docker
        IMAGE_NAME = "haneolenae/ssenbi-frontend:1.0"
        CONTAINER_NAME = "ssenbi-frontend"
        DOCKER_BRIDGE = "ssenbi-bridge"
    }

    stages {
        stage('Clean Workspace') {
```

```
            steps {
                cleanWs()
            }
        }

        stage('Checkout Application Git Branch') {
            steps {
                echo "Checkout Application Git Branch
==================================================="
                git credentialsId: 'gitlab-cred',
                    url: "${GIT_URL}",
                    branch: "${TRACKING_BRANCH}"
            }
        }

        stage('Capture Author Info') {
            steps {
                script {
                    AUTHOR_ID = sh(script: "git show -s --pretty=%an", returnStdout:
true).trim()
                    AUTHOR_NAME = sh(script: "git show -s --pretty=%ae", returnStdout:
true).trim()
                }
            }
        }

        stage('.env.local 파일 설정') {
            steps {
                echo ".env.local 파일 설정
==================================================="
                sh '''
                    cat > .env.local << EOF
                    NEXT_PUBLIC_API_END_POINT=https://www.ssenbi.co.kr/api/v1/ssenbi
                    NEXT_PUBLIC_FOO=bar
                    EOF
```

```
                '''
            }
        }

        stage('Write Dockerfile') {
            steps {
                echo "Dockerfile 생성
==================================================="
                writeFile file: 'Dockerfile', text: '''
                # 1. 빌드 스테이지: 애플리케이션 빌드
                FROM node:22-alpine AS builder

                # 작업 디렉토리 설정
                WORKDIR /app

                # 패키지 파일 복사 및 의존성 설치
                COPY package*.json ./
                RUN npm install

                # 소스 코드 복사
                COPY . .

                # Next.js 애플리케이션 빌드
                RUN npm run build

                # 2. 프로덕션 스테이지: 경량 이미지 생성
                FROM node:22-alpine

                # 작업 디렉토리 설정
                WORKDIR /app

                # 프로덕션 의존성 설치
                COPY package*.json ./
                RUN npm install --only=production
```

```
            # 빌드된 파일과 정적 파일 복사
            COPY --from=builder /app/.next .next
            COPY --from=builder /app/public public
            COPY --from=builder /app/next.config.mjs ./

            # 환경 변수 파일 복사 (.env.local 은 빌드 스크립트에서 생성됨)
            COPY --from=builder /app/.env.local ./

            # 애플리케이션 포트 노출
            EXPOSE 3000

            # 애플리케이션 시작 명령어
            CMD ["npm", "start"]
            '''
        }
    }

    stage('Install Dependencies') {
        steps {
            echo "Install Dependencies
==================================================="
            sh 'npm install'
        }
    }

    stage('Build Application') {
        steps {
            echo "Build Application
==================================================="
            sh 'npm run build'
        }
    }

    stage('Remove Old Docker Images') {
        steps {
```

```
            echo "Removing Old Docker Images
==================================================="
            sh '''
                docker image prune -a -f || true
            '''
        }
    }

    stage('Docker Image Build') {
        steps {
            echo "Docker Image Build
==================================================="
            sh 'docker build -t $IMAGE_NAME .'
        }
    }

    stage('Stop and Remove Existing Container') {
        steps {
            echo "Stopping and Removing Existing Container
==================================================="
            sh '''
                if [ $(docker ps -aq -f name=${CONTAINER_NAME}) ]; then
                    echo "Stopping and removing existing container"
                    docker stop ${CONTAINER_NAME} || true
                    docker rm ${CONTAINER_NAME} || true
                fi
            '''
        }
    }

    stage('Deploy Docker Container') {
        steps {
            echo "Deploy Docker Container
==================================================="
            sh '''
```

```
                    docker run -d --name ${CONTAINER_NAME} ₩
                    -e TZ=Asia/Seoul ₩
                    -p 3000:3000 ₩
                    --net ${DOCKER_BRIDGE} ₩
                    $IMAGE_NAME
                '''
            }
        }
    }


    post {
      always {
          script {
              def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
              def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout:
true).trim()
              def buildStatus = currentBuild.result ?: 'SUCCESS'
              def color = buildStatus == 'SUCCESS' ? 'good' : 'danger'
              def message = buildStatus == 'SUCCESS' ? "빌드 성공" : "빌드 실패"

              mattermostSend(color: color,
                message: "${message}: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Name})₩n(<${env.BUILD_URL}|Details>)",
                endpoint: 'https://meeting.ssafy.com/hooks/bywuihufjbfnpfqchikh3r8u8r',
                channel: 'front_build_result'
              )
          }
      }
    }
}
```