포팅매뉴얼

돈줌

1. 사용 도구

이슈 관리	Jira
형상 관리	GitLab
CI/CD	Docker, Jenkins, SonarQube
디자인	Figma
협업 툴	MatterMost, Notion

2. 개발환경

Server	Ubuntu 20.04.6 LTS
JDK	Amazon Corretto 17
Nginx	1.27.1
MySQL	8.0.39
Redis	6.2.14
IntelliJ	2024.1
Visual Studio Code	1.94.9
Node.js	v18.19.0
Python	3.9.20

3. 설정파일 및 환경변수 정보

EC2 포트 번호

Backend	8081
Frontend	3000
MySQL	3306
Redis	6379
Nginx	8080/443
FastAPI	8082

외부 서비스

- Kakao, Naver, Google API: OAuth
- 금융 API
- Open AI API
- Firebase API

Backend

- **Spring Boot**

  pem 키 파일은 별도 설정이 필요합니다.

  ▼ **application.properties**

  /src/main/resources, 또는 classPath에 위치해야 합니다.

```
cd /home/ubuntu/jenkins-data/pipeline/backend/src/main/resources

vim application.properties
```

```
spring.application.name=
server.port=
server.servlet.context-path=

frontend.uri=http://j11a108.p.ssafy.io:3000
backend.uri=http://j11a108.p.ssafy.io:8081

# Spring DataSource (MySQL)
spring.datasource.driver-class-name=
spring.datasource.url=
spring.datasource.username=
spring.datasource.password=

# Spring JPA
spring.jpa.database=
spring.jpa.hibernate.ddl-auto=
spring.jpa.defer-datasource-initialization=
spring.jpa.generate-ddl=
spring.jpa.show-sql=
spring.jpa.properties.hibernate.format_sql=
spring.sql.init.mode=

# 파일 저장소
file.upload-dir=

# 정적 자원 제공 경로 설정
spring.mvc.static-path-pattern=
spring.web.resources.static-locations=

#File Upload size Setting
spring.servlet.multipart.max-file-size=25MB
spring.servlet.multipart.max-request-size=25MB

# JWT setting
jwt.salt=
# 60 * 60 * 1000 (ms) = 1 hour
jwt.oauth.token.expireTime=3600000
```

```
# 2 * 60 * 60 * 1000 (ms) = 2 hour
jwt.accessToken.expireTime=72000000
# 7 * 24 * 60 * 60 * 1000 (ms) = 7 day
jwt.refreshToken.expireTime=604800000

spring.jwt.secret=
# JWT
spring.security.oauth2.resourceserver.jwt.issuer-uri=
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=

# Redis
spring.redis.host=
spring.redis.port=

# 금융 API
fin.api-key="금융API Key"
fin.base-url="금융API Base URL"
fin.generate-api-key-url=
fin.generate-user-key-url=
fin.user-info-url=
fin.inquire-bank-code-url=
fin.generate-demand-deposit-url=
fin.inquire-demand-deposit-url=
fin.create-demand-deposit-account-url=
fin.inquire-demand-deposit-account-list-url=
fin.inquire-demand-deposit-account-url=
fin.inquire-demand-deposit-account-holder-url=
fin.inquire-demand-deposit-account-balance-url=
fin.update-demand-deposit-account-withdrawal-url=
fin.update-demand-deposit-account-deposit-url=
fin.update-demand-deposit-account-transfer-url=
fin.inquire-transaction-history-url=
fin.inquire-transaction-history-list-url=
fin.inquire-category-list-url=
fin.create-merchant-url=
fin.inquire-card-issuer-codes-url=
fin.create-credit-card-product-url=
fin.inquire-credit-card-list-url=
fin.create-credit-card-url=
fin.inquire-sign-up-credit-card-list=
fin.inquire-merchant-list-url=
fin.create-credit-card-transaction-url=
fin.inquire-credit-card-transaction-list-url=
fin.delete-transaction-url=

#Kakao
spring.security.oauth2.client.provider.kakao.authorization-uri=
spring.security.oauth2.client.provider.kakao.user-name-attribute=
spring.security.oauth2.client.provider.kakao.token-uri=
spring.security.oauth2.client.provider.kakao.user-info-uri=

spring.security.oauth2.client.registration.kakao.client-name=
spring.security.oauth2.client.registration.kakao.authorization-grant-type=
spring.security.oauth2.client.registration.kakao.redirect_uri=
spring.security.oauth2.client.registration.kakao.client-id=
```

```
spring.security.oauth2.client.registration.kakao.client-secret=
spring.security.oauth2.client.registration.kakao.client-authentication-method=
spring.security.oauth2.client.registration.kakao.scope=

#Naver
spring.security.oauth2.client.provider.naver.authorization-uri=
spring.security.oauth2.client.provider.naver.token-uri=
spring.security.oauth2.client.provider.naver.user-info-uri=
spring.security.oauth2.client.provider.naver.user-name-attribute=

spring.security.oauth2.client.registration.naver.client-name=
spring.security.oauth2.client.registration.naver.authorization-grant-type=
spring.security.oauth2.client.registration.naver.redirect-uri=
spring.security.oauth2.client.registration.naver.client-id=
spring.security.oauth2.client.registration.naver.client-secret=
spring.security.oauth2.client.registration.naver.scope=

#Google
google.client.id=
spring.security.oauth2.client.registration.google.client-name=
spring.security.oauth2.client.registration.google.authorization-grant-type=
spring.security.oauth2.client.registration.google.redirect-uri=
spring.security.oauth2.client.registration.google.client-id=
spring.security.oauth2.client.registration.google.client-secret=
spring.security.oauth2.client.registration.google.scope=

#Ticket Price
ticketPrice = 5
```

▼ 환경변수 (.env)

root 디렉토리에 위치해야 합니다.

```
MYSQL_USER=
MYSQL_PASSWORD=
MYSQL_DATABASE=
MYSQL_ROOT_PASSWORD=
REDIS_HOST=
REDIS_PORT=
```

▼ Firebase 환경변수 (firebase-service-account.json)

/src/main/resources, 또는 classPath에 위치해야 합니다.

```
{
  "type": "service_account",
  "project_id": "donzoom",
  "private_key_id": "서비스 Key ID",
  "private_key": "",
  "client_email": "",
  "client_id": "",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "",
```

```
  "universe_domain": "googleapis.com"
}
```

- **FastAPI**
  - ▼ **.env**

    root 디렉토리에 위치해야 합니다.

    ```
    SPRING_BOOT_URL=
    STOCK_PATH=/stock
    NEWS_PATH=/news
    REPORT_PATH=/report
    ```

  - ▼ **requirements.txt**

    root 디렉토리에 위치해야 합니다.

    ```
    fastapi
    uvicorn
    apscheduler
    yfinance
    pytz
    requests
    python-dotenv
    selenium
    webdriver-manager
    ```

Frontend

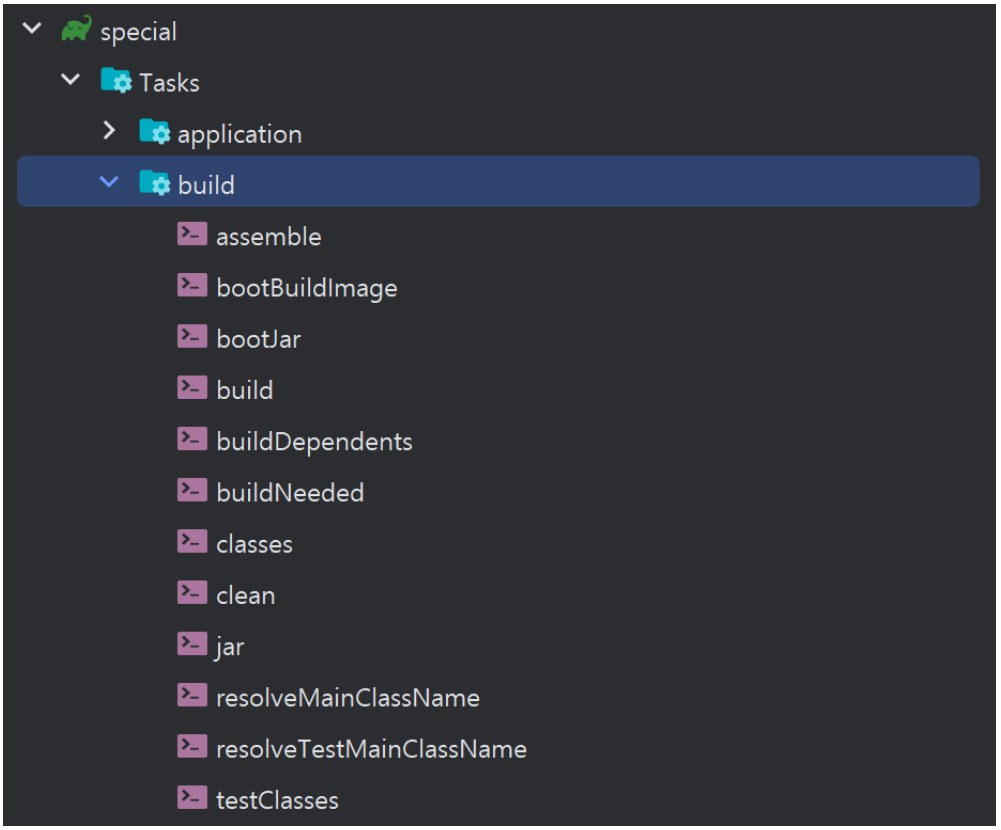
- **React Native**
  - ▼ **.env**

    ```
    BASE_URL=
    EC2=
    GOOGLE_WEB_CLIENT_ID=
    NAVER_WEB_CLIENT_ID=
    NAVER_CONSUMER_SECRET=
    NAVER_APP_NAME=돈줌
    ```

4. 빌드 방법

4-1. Spring Boot

- IntelliJ 우측 코끼리 모양 (Gradle) 클릭
- `clean` → `build`



4-2. FastAPI

```
# root 디렉토리 내에서 실행
# 라이브러리 설치
pip install -r requirements.txt

# 앱 실행 명령어
uvicorn main:app --reload --port 8082
```

4-3. Frontend

```
# 라이브러리 설치
npm install

# android 실행
npm run android

# android 내에서 app을 실행
a
```

```
cd frontend/android
chmod +x gradlew  # gradlew 파일에 실행 권한 부여
./gradlew assembleRelease  # APK 빌드
```

5. EC2 Setting

Docker 설치

- EC2 서버 접속

```
ssh -i {KEY_PATH} {USER}@{SERVER_IP}
```

  - KEY_PATH: EC2서버 쪽에서 인증에 사용될 키페어(.pem)파일의 경로
  - USER: 접속한 서버에서 사용할 User계정, Ubuntu운영체제를 선택할 경우 기본으로 ubuntu계정 사용
  - SERVER_IP: 접속하고자 하는 서버의 IP주소(EC2서버에 부여한 탄력적 IP주소)
- 패키지 업데이트

```
sudo apt update
```

- https 관련 패키지 설치 → SSL

```
sudo apt install apt-transport-https ca-certificates curl software-properties-commo
n
```

- docker repository 접근을 위한 gpg 키 설정

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- docker repository 등록

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
focal stable"
```

- 다시 패키지 업데이트

```
sudo apt update
```

- docker 설치

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- docker 설치 확인

```
sudo docker --version
sudo docker run hello-world
```

Docker-compose 설치

- 현재 사용자를 docker group에 추가

```
sudo usermod -aG docker ubuntu
```

- 터미널 재시작 후, 결과 확인

```
# Docker가 있으면 됨
id -nG
```

- Compose standalone 설치

```
curl -SL https://github.com/docker/compose/releases/download/v2.20.0/docker-compose
-linux-x86_64 -o /usr/local/bin/docker-compose
```

- 설치한 파일에 실행권한 추가

```
chmod +x /usr/local/bin/docker-compose
```

- docker-compose 설치 확인

```
docker-compose -v
```

DockerFile 설정

▼ **Spring Boot**

```
# Step 1: Amazon Corretto 17을 베이스 이미지로 선택
FROM amazoncorretto:17

# Step 2: 작업 디렉토리 설정
WORKDIR /app

# Step 3: 빌드된 JAR 파일을 Docker 이미지에 복사
COPY build/libs/donzoom.jar /app/donzoom.jar

# Step 4: 컨테이너 포트 노출
EXPOSE 8081

# Step 5: config 파일을 위한 볼륨 마운트 설정
VOLUME /config

# Step 6: 컨테이너가 실행될 때 JAR 파일을 실행하고 /config 경로의 설정 파일을 사용하도록 설정
ENTRYPOINT ["java", "-jar", "/app/donzoom.jar", "--spring.config.location=/config/a
pplication.properties"]
```

▼ **FastAPI**

```
# 베이스 이미지로 Python 3.9를 사용
FROM python:3.9-slim

# 작업 디렉토리 생성
WORKDIR /app

# 필요한 패키지 설치를 위해 필요한 종속성 설치 및 Google Chrome 레포지토리 추가
RUN apt-get update && apt-get install -y \
    wget \
    unzip \
    curl \
    gnupg \
    libnss3 \
    libgconf-2-4 \
    libxi6 \
    libxrender1 \
    libxtst6 \
    libxss1 \
    libxrandr2 \
    libasound2 \
    fonts-liberation \
    libappindicator3-1 \
```

```
    xdg-utils \
    && wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | apt-key add
- \
    && sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable
main" >> /etc/apt/sources.list.d/google-chrome.list' \
    && apt-get update \
    && apt-get install -y google-chrome-stable \
    && rm -rf /var/lib/apt/lists/*

# 필요한 의존성 파일을 컨테이너에 복사
COPY requirements.txt .

# 필요한 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# 모든 애플리케이션 파일을 컨테이너에 복사
COPY . .

# 컨테이너의 8082 포트 노출
EXPOSE 8082

# uvicorn을 사용하여 FastAPI 서버 실행
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8082", "--reload"]
```

Docker Compose 설정

▼ **docker-compose.yml**

redis, mysql, springboot, fastapi

- 볼륨 설정: `mysql`, `sprinbgboot`

```
version: '3.8'
services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    environment:
      TZ: Asia/Seoul
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
    ports:
      - "3306:3306"
    volumes:
      - ./mysql_data:/var/lib/mysql

  springboot:
    image: backend
    build: ./backend  # backend 디렉토리에서 Dockerfile을 사용해 이미지 빌드
    container_name: donzoom
    environment:
      TZ: Asia/Seoul
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/${MYSQL_DATABASE}
      SPRING_DATASOURCE_USERNAME: ${MYSQL_USER}
```

```
      SPRING_DATASOURCE_PASSWORD: ${MYSQL_PASSWORD}
      SPRING_REDIS_HOST: redis
      SPRING_REDIS_PORT: 6379
    ports:
      - "8081:8081"
    volumes:
      - /home/ubuntu/jenkins-data/workspace/pipeline/backend/src/main/resources/application.properties:/config/application.properties
      - /home/ubuntu/uploads:/uploads  # EC2의 /home/ubuntu/uploads 디렉토리를 컨테이너의 /uploads로 마운트

    depends_on:
      - mysql
      - redis

  redis:
    image: redis:6.2
    container_name: redis
    environment:
      TZ: Asia/Seoul
    ports:
      - "6379:6379"  # 호스트와 컨테이너 간의 포트 매핑
    command: ["redis-server", "--bind", "0.0.0.0", "--protected-mode", "no"]  # 모든 IP에서 접근 허용 및 보호 모드 비활성화
    restart: always  # 컨테이너가 종료되면 자동 재시작

  fastapi:
    build: ./python  # fastapi 디렉토리에서 Dockerfile을 사용해 이미지 빌드
    container_name: fastapi
    environment:
      TZ: Asia/Seoul
    ports:
      - "8082:8082"  # FastAPI 앱의 포트
    depends_on:
      - mysql
      - redis
```

Nginx 도커 컨테이너 띄우기

▼ docker에 nginx 이미지 다운로드

```
docker pull nginx:latest
```

▼ docker 컨테이너 실행

```
docker run --name nginx -p 8080:80 -d nginx:latest
docker ps # 제대로 설치되었는지 확인
```

Nginx 설정

▼ **/etc/nginx/conf.d/default.conf**

managed by certbot ssl 발급받는 과정에서 자동으로 적용됩니다.

```
server {
    server_name j11a108.p.ssafy.io;
```

```
        ### Spring 서버 ###
    location /api/ {
        proxy_pass http://j11a108.p.ssafy.io:8081;
        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;

        # CORS 설정
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELET
E';
        add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Conten
t-Type, Accept';

        # OPTIONS 메서드에 대한 CORS Preflight 처리
        if ($request_method = OPTIONS ) {
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DEL
ETE';
            add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Co
ntent-Type, Accept';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Length' 0;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            return 204;
        }
    }

        ### WebSocket 연결 ###
    location /websocket/ {
        proxy_pass https://j11a108.p.ssafy.io;  # 백엔드 서버와 일관되게 HTTP 사용
        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;

        # CORS 설정 추가
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELET
E';
        add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Conten
t-Type, Accept';

        if ($request_method = OPTIONS ) {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Credentials' 'true';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DEL
ETE';
            add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Co
ntent-Type, Accept';
            add_header 'Access-Control-Max-Age' 1728000;
```

```
            add_header 'Content-Length' 0;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            return 204;
        }
    }

        ### FastAPI 서버 ###
    location /fastapi/ {
        proxy_pass http://j11a108.p.ssafy.io:8082;
        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;

        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELET
E';
        add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Conten
t-Type, Accept';
    }

    location / {
        proxy_pass http://j11a108.p.ssafy.io:8080;
        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;

        # CORS 설정 추가
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELET
E';
        add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Conten
t-Type, Accept';

        if ($request_method = OPTIONS ) {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Credentials' 'true';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DEL
ETE';
            add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Co
ntent-Type, Accept';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Length' 0;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            return 204;
        }
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j11a108.p.ssafy.io/fullchain.pem; # manag
ed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/j11a108.p.ssafy.io/privkey.pem; # man
```

```
aged by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = j11a108.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name j11a108.p.ssafy.io;
    return 404; # managed by Certbot
}
```

Jenkins 도커 컨테이너 띄우기

▼ docker에 jenkins 이미지 다운로드

```
 docker pull jenkins/jenkins
```

▼ docker 컨테이너 실행

```
docker run -d -v jenkins_home:/var/jenkins_home -p 8088:8080 -p 50000:50000 --resta
rt=on-failure --name jenkins-server jenkins/jenkins:lts-jdk11
```

JenkinsFile 설정

GitLab Webhooks 기반

▼ **Backend**

- Repository: https://lab.ssafy.com/s11-fintech-finance-sub1/S11P21A108.git / *backend*
- URL: http://j11a108.p.ssafy.io:8080/project/pipeline

```
pipeline {
    agent any

    stages {
        stage('Checkout SCM') {
            steps {
                echo 'Checking out SCM...'
                checkout scm
            }
        }

        ### SonarQube 설정 ###
        stage('Prepare SonarQube Analysis') {
            steps {
                echo 'Setting executable permissions for gradlew...'
                sh '''
                    cd backend
                    chmod +x gradlew  # gradlew 파일에 실행 권한 부여
                '''
            }
        }
```

```
                ### SonarQube 실행 ###
        stage('SonarQube Analysis') {
            steps {
                echo 'Running SonarQube analysis...'
                withSonarQubeEnv('SonarQube'){  // SonarQube 인스턴스 이름을 사용
                    sh '''
                        cd backend
                        ./gradlew sonar
                    '''
                }
            }
        }

                ### Spring JAR 파일 빌드 ###
        stage('Build JAR') {
            steps {
                echo 'Building JAR file...'
                sh '''
                    cd backend
                    chmod +x gradlew  # gradlew 파일에 실행 권한 부여
                    ./gradlew clean bootJar --no-build-cache -x test
                '''
            }
        }

                ### Docker Compose로 빌드 ###
        stage('Deploy with Docker Compose') {
            steps {
                echo 'Deploying with Docker Compose...'
                sh '''
                    # 새로운 이미지를 빌드하고 모든 컨테이너 실행
                    docker-compose up -d --build
                '''
            }
        }
    }

    post {
        always {
            echo 'Pipeline finished.'
        }
        success {
            echo 'Pipeline completed successfully.'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

▼ **Frontend**

- Repository: https://lab.ssafy.com/s11-fintech-finance-sub1/S11P21A108.git / *frontend*
- URL: http://j11a108.p.ssafy.io:8080/project/frontend

```
pipeline {
    agent any

    environment {
        APK_OUTPUT_DIR = 'frontend/android/app/build/outputs/apk/release/'  // APK
파일 위치
        APK_FINAL_NAME = 'DONZOOM.apk'  // APK 파일 최종 이름
    }

    stages {
        stage('Checkout SCM') {
            steps {
                echo 'Checking out frontend SCM...'
                checkout scm  // Git에서 소스 코드 체크아웃
            }
        }

                ### 필요한 Node.js 라이브러리 빌드 ###
        stage('Install Node Modules') {
            steps {
                echo 'Installing node modules...'
                sh '''
                    cd frontend
                    npm install  # 필요한 Node 모듈 설치
                    npm install @svgr/plugin-svgo --save-dev  # 누락된 모듈 추가 설치
                '''
            }
        }

                ### APK 파일로 빌드 ###
        stage('Build APK') {
            steps {
                echo 'Building APK for frontend...'
                sh '''
                    cd frontend/android
                    chmod +x gradlew  # gradlew 파일에 실행 권한 부여
                    ./gradlew assembleRelease  # APK 빌드
                '''
            }
        }

        stage('Find and Rename APK') {
            steps {
                echo 'Finding and renaming APK...'
                script {
                    def apkPath = sh(script: "find ${APK_OUTPUT_DIR} -name 'app-rel
ease.apk'", returnStdout: true).trim()
                    if (apkPath) {
                        echo "APK found at: ${apkPath}"
                        sh "mv ${apkPath} ${APK_OUTPUT_DIR}${APK_FINAL_NAME}"
                    } else {
                        error "APK 파일을 찾을 수 없습니다."
                    }
                }
```

```
            }
        }

                ### 다운로드 가능한 URL로 아카이브 ###
        stage('Archive APK') {
            steps {
                echo 'Archiving DONZOOM.apk...'
                archiveArtifacts allowEmptyArchive: false, artifacts: "${APK_OUTPUT
_DIR}${APK_FINAL_NAME}"  // 상대 경로로 아카이브
            }
        }
    }

    post {
        always {
            echo 'Frontend pipeline finished.'
        }
        success {
            echo 'Frontend pipeline completed successfully.'
        }
        failure {
            echo 'Frontend pipeline failed.'
        }
    }
}
```

Jenkins CI/CD 파이프라인

- **Backend**
  - `New Item` → `Pipeline`
  - `Build Trigger` → `GitLab webhook` → `Push Events`
  - Pipeline Definition
    - Definition: `Pipeline script from SCM`
    - SCM: `Git`
    - Repository URL: `https://lab.ssafy.com/s11-fintech-finance-sub1/S11P21A108.git`
    - Credentials: `"저장소 접근을 위한 자격증명"`
  - Branch Specifier (blank for 'any'): `*/backend`
  - ScriptPath: `Jenkinsfile`
- **Frontend**
  - `New Item` → `Pipeline`
  - `Build Trigger` → `GitLab webhook` → `Push Events`
  - Pipeline Definition
    - Definition: `Pipeline script from SCM`
    - SCM: `Git`
    - Repository URL: `https://lab.ssafy.com/s11-fintech-finance-sub1/S11P21A108.git`
    - Credentials: `"저장소 접근을 위한 자격증명"`
  - Branch Specifier (blank for 'any'): `*/frontend`
  - ScriptPath: `Jenkinsfile`