포팅 메뉴얼

태그	마무리

1. 개발환경

1.1 Frontend(Mobile, Watch)

- Kotlin 2.0.0 (Language)
- Jetpack Compose (UI Framework)
- AGP 8.8.0 (Android Gradle Plugin)

1.2 Backend

- Java
  - Azul Zulu 17.0.14
  - Spring Boot 3.4.2
    - Spring Dpendency Management 1.1.7
    - Spring Security
    - Spring Data JPA
    - Spring Web MVC
    - Lombok
    - JWT
    - Oauth 2.0
    - Socket.io
  - Gradle 8.12.1

1.3 Mobile (Galaxy)

- Android 14
- Retrofit 2.9.0 (http)
- Coil 3.0.4 (gif)
- Socket.io 2.0.0 (웹소켓통신-레이드)

- Capturable (캐릭터 캡쳐)
- Mapbox 11.9.0 (GPX 지도)
- Google Play services 19.0.0 (워치 데이터 송수신)
- ComposeCharts 0.1.1 (통계 차트)

1.4 Watch (Galaxy)

- Wear OS 5
- play Services Wearable : 19.0.0
- Gson : 2.9.0

1.5 Database

- MySQL 8.0.41
- MongoDB 6.0.20
- Redis 7.4.2
- AWS S3

1.6 Server

Ubuntu 22.04

Docker

Let’sEncrypt (SSL)

- Jenkins 2.492.1
  - Contabo
    - NGINX
- Test Server
  - Google Computing Engine
- Production Server
  - EC2
    - NGINX

1.7 UI/UX

- Figma, Canva

- Kirta, ClipStudio, LibreSprite

1.8 IDE

- Visual Studio Code
- IntelliJ IDEA
- Android Studio

1.9 형상 / 이슈관리

- GitLab
- Jira

1.10 기타 툴

- Postman
- Mattermost
- Notion
- Discord
- OBS Studio
- scrcpy
- Termius
- puTTY

2. 인프라 세팅

2.1 서버 세팅 공통

2.1.0 서버 시간대 통일 / 깃 정보 등록

```
# 서버 시간대 확인
timedatectl

# 서울이 아닐 경우
sudo tiemdatectl set-timezone Asia/Seoul
```

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

2.1.1 Zulu 17 설치

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

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

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

sudo nano /etc/environment
# 맨 마지막줄에 환경변수 작성
JAVA_HOME="/usr/lib/jvm/zulu17/bin/java"
# Ctrl + O , Ctrl + X 로 저장 후 나가기

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

2.1.2 Redis 설치

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

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

# 접속
redis-cli

# 키 조회
keys *
# get "조회할 키 이름"
```

2.1.3 MySQL 설치 / 설정

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

# mysql 접속
sudo mysql

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

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

# 데이터베이스 생성 후 보기
CREATE DATABASE 새로운데이터베이스;
SHOW DATABASES;

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

EXIT;

# 이후엔 mysql -u 유저명 -p 로 sql 접속
```

2.1.4 MongoDB 설치 / 세팅

```
# MongoDB 공개 GPG키 가져오기
sudo apt-get install gnupg curl
curl -fsSL https://www.mongodb.org/static/pgp/server-6.0.asc | \
   sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg \
   --dearmor

# 사용중인 우분투 버전에 맞게 목록파일 생성 - 22.04(jammy)
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg ] https:/

# 최신 릴리즈 설치
sudo apt-get update && sudo apt-get install -y mongodb-org

# 등록 && 실행
sudo systemctl enable mongod
```

```
sudo systemctl start mongod

# MongoDB 접속
mongosh

# 데이터베이스 생성
use 데이터베이스

# 테스트 데이터 생성
db.test.insertOne({ "name": "test" })

# db 목록 확인
show dbs

# collections 확인
show collections

# collection 내부 데이터 확인
db.[collection 이름].find()

# 데이터 입력
db.[collection 이름].insert<Many-여러개, One-1개>(데이터)


# 데이터 삭제
db.[collection 이름].delete<Many-여러개, One-1개, All-전부>(데이터)
```

2.1.5 Docker 설치

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

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

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

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

# 등록 && 시작
```

```
sudo systemctl enable docker
sudo systemctl start docker
```

2.1.6 SSL 인증서 등록

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

# 적용할 도메인 주소와 이메일 입력해주기
sudo certbot certonly --standalone
# 생성한 키를 PKCS12 형식으로 변환
sudo openssl pkcs12 -export -in /etc/letsencrypt/live/<도메인주소>/fullchain.pem -inkey /etc/letser
# 비밀번호 설정함. 이 비밀번호로 스프링부트 SSL 설정
```

2.1.7 NGINX 설치

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

sudo systemctl enable nginx
sudo systemctl start nginx
```

2.1.8 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80  # HTTP
sudo ufw allow 443  # HTTPS
sudo ufw allow mysql
sudo ufw allow 27017  # MongoDB
sudo ufw allow redis
sudo ufw allow 6380  # sub redis
sudo ufw allow 8080  # springboot
sudo ufw allow 8081  # springboot2
sudo ufw allow 9092  # springboot socket.io
sudo ufw allow 9093  # springboot socket.io2
sudo ufw enable
```

2.2 Jenkins 서버 세팅

2.2.1 Jenkins 설치

```

# jenkins 설치
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
```

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

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

# 서비스 등록 및 실행
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

2.2.2 NGINX 설정

```
sudo nano /etc/nginx/sites-available/jenkins
sudo ln -s /etc/nginx/sites-available/jenkins /etc/nginx/sites-enabled/
sudo rm /etc/nginx/sites-enabled/default
sudo systemctl restart nginx
```

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

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

upstream jenkins {
    server 127.0.0.1:8080;
    keepalive 32;  # 커넥션 유지
}

server {
    listen 80;
    server_name localhost;

    ignore_invalid_headers off;

    location / {
        proxy_pass http://jenkins;

        # 프록시 헤더 설정
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;  # 앞단의 HTTPS를 위해
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port 443;     # HTTPS 포트
```

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

        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off;

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

2.3 Production Server 세팅

2.3.1 NGINX 설정

```
sudo nano /etc/nginx/nginx.conf

# /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 1024;
}

http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

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

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
```

```
gzip on;
gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_types text/plain text/css application/json application/javascript text/xml application/xml ap

client_body_buffer_size 10K;
client_header_buffer_size 1k;
client_max_body_size 8m;
large_client_header_buffers 2 1k;

upstream springboot_8080 {
    server localhost:8080;
}

upstream springboot_9092 {
    server localhost:9092;
}


upstream springboot_8081 {
    server localhost:8081;
}

upstream springboot_9093 {
    server localhost:9093;
}

server {
    listen 80;
    server_name i12e205.p.ssafy.io;

    location / {
        proxy_pass http://springboot_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;
    }

    location /api {
        proxy_pass http://springboot_9092;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /test {
        proxy_pass http://springboot_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;
    }

    location /test/api {
        proxy_pass http://springboot_9093;
        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;
    }
}

server {
    listen 443 ssl;
    server_name i12e205.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/i12e205.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i12e205.p.ssafy.io/privkey.pem;

    location / {
        proxy_pass https://springboot_8080;
        proxy_ssl_verify off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        proxy_pass https://springboot_9092;
        proxy_ssl_verify off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /test {
        proxy_pass https://springboot_8081;
        proxy_ssl_verify off;
```

```
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /test/api {
            proxy_pass https://springboot_9093;
            proxy_ssl_verify off;
            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;
        }
    }
}
```

3. CI/CD 구축

3.1 SpringBoot Health Check

- 요청주소 : API주소:포트/actuator/health

```
// build.gradle
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
}
```

health package - SimpleHealthIndicator class

```
# health/SimpleHealthIndicator.java
package com.ssafy.roCatRun.health;

import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class SimpleHealthIndicator implements HealthIndicator {
    @Override
    public Health health() {
        return Health.up().build();
```

```
    }
}
```

application.yml

```
// application.yml
management:
  endpoint:
    health:
      show-details: always
      show-components: NEVER
  health:
    defaults:
      enabled: false
    simple:
      enabled: true
```

SpringSecurity

```
// config/SecurityConfig.java
...
public class SecurityConfig {

    ...

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            ...

            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/actuator/health").permitAll()
                .requestMatchers("/actuator/**").hasRole("ADMIN")

                 ...

                .anyRequest().authenticated()
            )
            ...

            .httpBasic(Customizer.withDefaults())
            .csrf(csrf -> csrf.ignoringRequestMatchers("/actuator/**"));
        return http.build();
    }
}
...
```

- 응답 - 200 OK

```
{status : 'up'}
```

3.2 Jenkins

3.2.1 세팅

1. 플러그인 설치

기본적으로 추천 플러그인 설치된 상태.

Dashboard > Jenkins 관리 > Plugins > Available plugins

- GitLab
  - 깃랩 연동
- SSH Agent
  - SSH로 배포 서버 연결
- Docker
  - Docker 사용
- Docker Pipeline
  - Docker 작업 간소화
- Pipeline: Stage View
  - 파이프라인 단계별 상태 확인
- AnsiColor
  - 콘솔 출력 컬러 지원
- Monitoring
  - Jenkins 서버 상태 모니터링
- Jira
  - Jira 이슈 연동
- Mattermost Notification
  - Mattermost 알림 연동

이외에 기본적으로 설치되는 플러그인

- Git
  - 깃 레포지토리
- Pipeline
  - Jenkinsfile 기반 파이프라인 작성
- Credentials Binding
  - 환경변수 설정

- Email Extension
  - 이메일 알림

2. Credentials 설정

Dashboard > Jenkins 관리 > Credentials > System > Global credentials > Add Credentials

- GitLab API token 등록
  - Kind - GitLab API token
- Jira API token 등록
  - Kind - Username with password
  - Username : Jira email
  - Password : token
- Docker-Hub
  1. docker-hub에서 우측 상단의 프로필 누르고 Account settings 들어가기
  2. Personal access tokens 들어가서 생성 - 권한 Read, Write, Delete 선택
  3. 생성된 키 Jenkins에 등록 - Credentials > System > Global credentials
  4. Docker-Hub username이랑 생성한 엑세스 토큰 등록 (Kind: Username with password)
- 서버 SSH
  1. Jenkins Credentials 등록의 Kind에서 SSH 선택
  2. ID 작성하고 SSH 접속할 때 사용하는 username 작성 (ec2 우분투 인스턴스는 ubuntu)
  3. Private Key 작성 - Enter directly 체크 후 Key Add 눌러서 pem 키 내용 전부 복붙
- application.yml
- SSL Key file

3. Mattermost Webhook 설정

1. Mattermost의 통합 > 전체 Incoming Webhook > 추가하기
2. 제목과 채널 설정
3. 생성된 Webhook 링크 복사
4. 알림 보낼 채널에서 링크 복사하고 채널 ID 찾기
   - https://meeting.ssafy.com/s12p10e2/channels/2-5 에서는 2-5가 채널 ID
5. Jenkins System 설정에서 Mattermost Notifier 설정
   a. Endpoint에 복사한 Webhook 링크 넣기
   b. Channel에 복사한 채널ID 넣기
   c. Test Connection 버튼 눌러서 전송 잘 되는지 확인
   d. Success 뜨면 저장

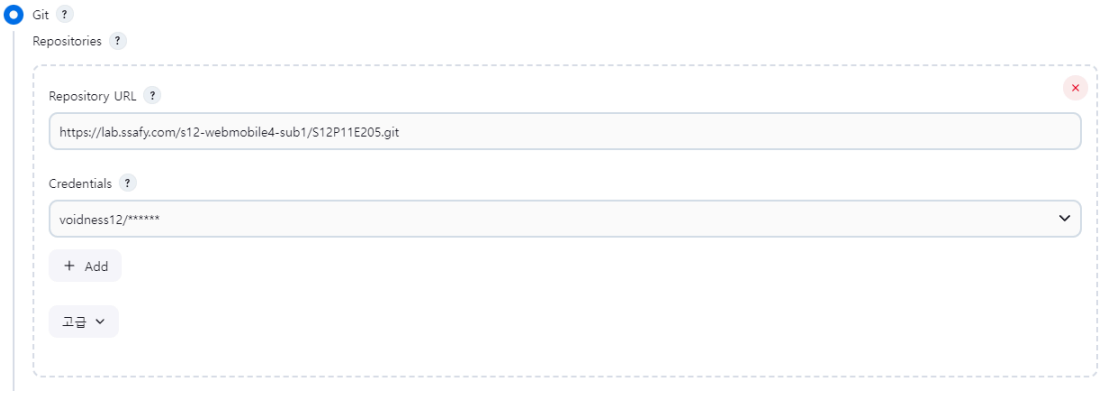
웹훅 링크 테스트(unix)

- curl -X POST -H 'Content-Type: application/json' -d '{"text": "Test message from webhook"}' 웹훅링크

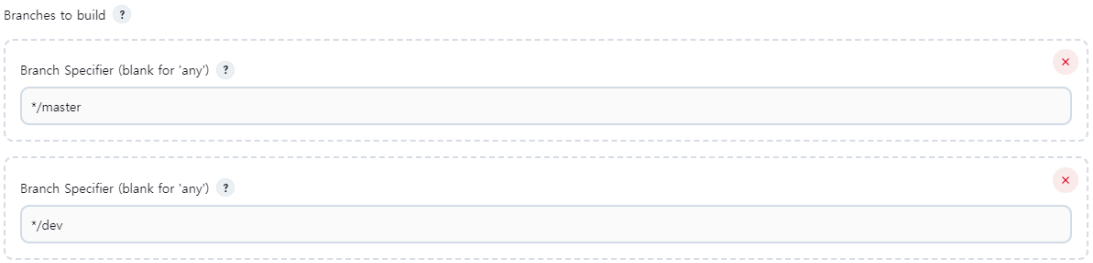
3.2.2 깃랩 웹훅 설정

깃랩 웹훅 설정 - 브랜치별 조건 다르게

1. 소스 코드 관리 - Git
    - 깃 레포지토리 설정

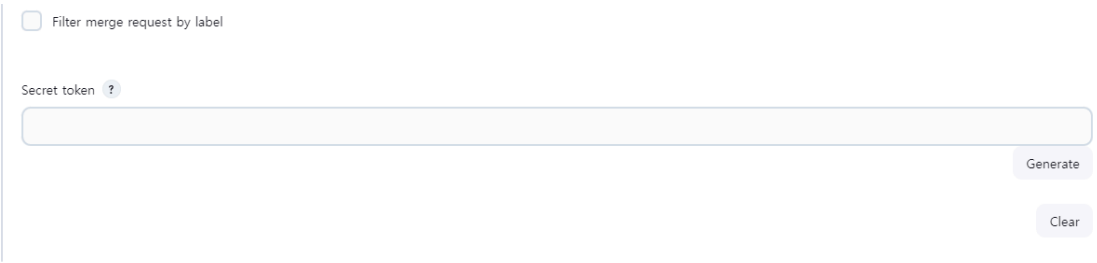


    - 브랜치 설정

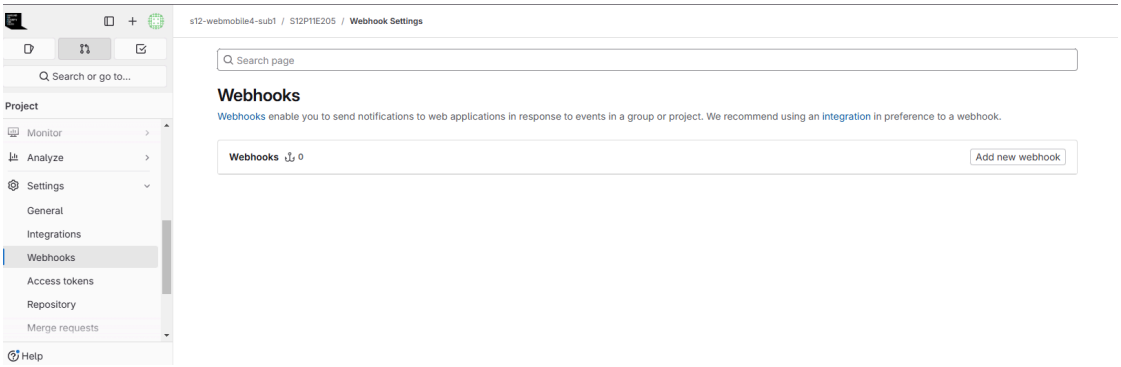


2. 빌드 트리거
    - 깃랩 푸시 감지 - 웹훅 사용

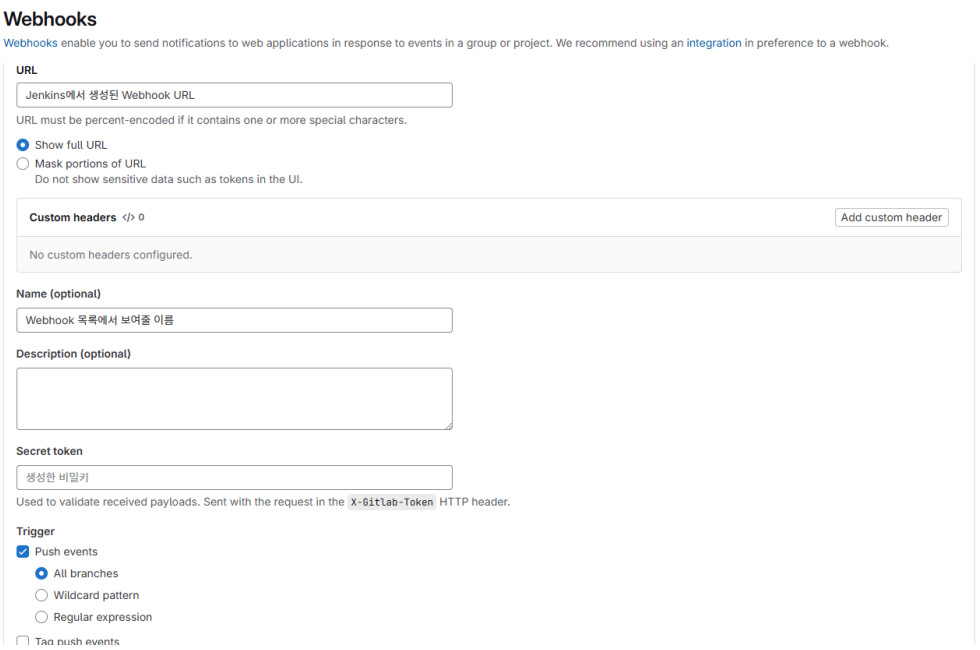
    - 고급에서 비밀키 생성



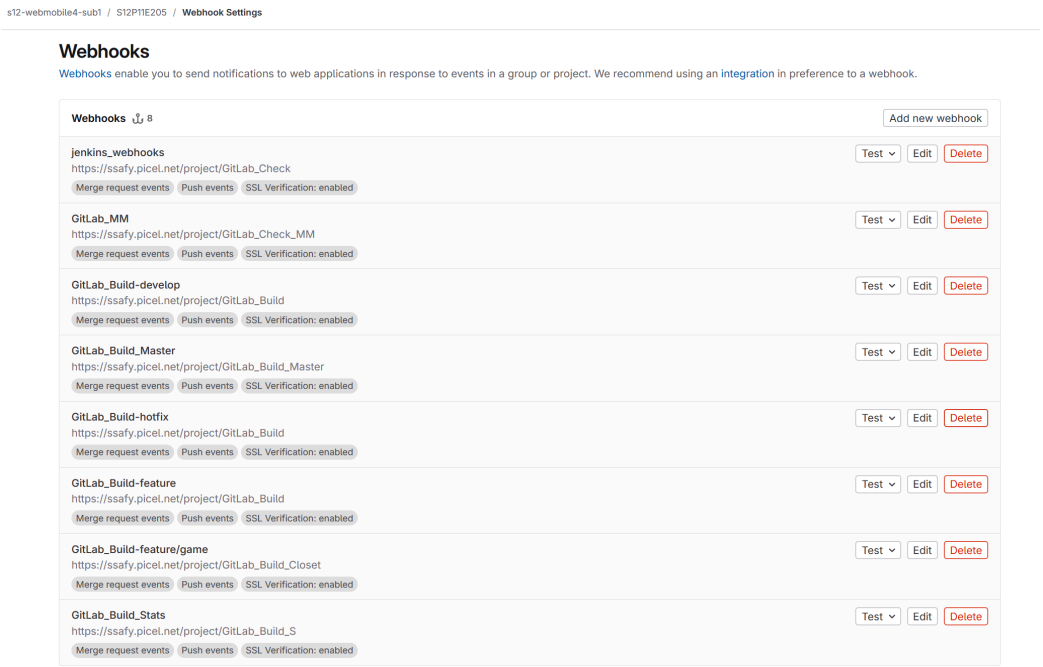
3. 생성한 비밀키로 GitLab Webhook 생성
    - 감시할 GitLab Repo의 Settings - Webhooks 들어가서 Add new webhook 버튼



    - 위에서 누른 Build when a change is pushed to GitLab. 옵션 맨 끝에 딸린 Webhook 주소와 고급탭에서 생성한 비밀키로 웹훅 작성



    - 트리거 설정하고 저장
        - Push, MR 이벤트 설정



3.2.3 Pipeline 작성

```
import groovy.json.JsonOutput

pipeline {
    agent any
    tools {
        gradle 'Default Gradle'
        jdk 'Zulu17'
        git 'Default Git'
    }
    environment {
        // 프로젝트 구조 변수
        ROOT = "GitLab_Build"
        BACKEND_DIR = "Back"
        DOCKERFILE_PATH = "Dockerfile"
        STAGE_NAME = ''

        // Docker 설정
        DOCKER_USER = 'rocatrun'
        IMAGE_NAME = 'rocatrun'
        GIT_COMMIT_SHORT = ''
        DOCKER_TAG = ''

        // 깃 정보
        COMMIT_MSG = ''
        COMMIT_HASH = ''
```

```
    AUTHOR = ''
    BRANCH_NAME = ''
    EXCLUDE_BRANCH = 'master,feature/game,feature/mobile,feature/mypage,feature/home,fea
    ERROR_MSG = "false"

    // 서버 정보
    TEST_SERVER = 'ssafy-gcloudtest.kro.kr'
    PROD_SERVER = 'i12e205.p.ssafy.io'
    JIRA_BASE_URL = 'https://ssafy.atlassian.net'
    GITLAB_BASE_URL = 'https://lab.ssafy.com/s12-webmobile4-sub1/S12P11E205'
}
stages {
    stage('Checkout and Update') {
        steps {
            script {
                STAGE_NAME = "Checkout and Update (1/9)"
                def notAllowedBranches = EXCLUDE_BRANCH.split(',')
                def repoExists = fileExists('.git')
                if (repoExists) {
                    echo "Repository exists. Updating..."
                    try {
                        checkout([
                            $class: 'GitSCM',
                            branches: [[name: '*/develop']],
                            userRemoteConfigs: [[
                                url: "${GITLAB_BASE_URL}.git",
                                credentialsId: 'gitlab-credentials'
                            ]],
                            extensions: [
                                [$class: 'CleanBeforeCheckout'],
                                [$class: 'PruneStaleBranch']
                            ]
                        ])
                        withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials', gitTool
                            sh """
                                git fetch --all --prune
                                git remote update --prune
                                git gc --prune=now
                            """

                            BRANCH_NAME = sh(script: """
                                git ls-remote --sort=-committerdate origin 'refs/heads/*' |
                                awk -F'/' '{print substr(\$0, index(\$0,\$3))}' |
                                head -n 1
                            """, returnStdout: true).trim()
                            echo "Latest branch: ${BRANCH_NAME}"

                            if (notAllowedBranches.contains(BRANCH_NAME)) {
```

```
                    BRANCH_NAME='develop'
                    echo "허용되지 않은 브랜치이므로 develop으로 빌드합니다."
                }

                sh "git checkout -B ${BRANCH_NAME} origin/${BRANCH_NAME} --force"

                GIT_COMMIT_SHORT = sh(script: "git rev-parse --short HEAD", returnStdout: tr
                DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"
                echo DOCKER_TAG
            }
        } catch (Exception e) {
            echo "Error during update: ${e.message}"
            ERROR_MSG = "Failed to update repository"
            error ERROR_MSG
        }
    } else {
        echo "Repository does not exist. Cloning..."
        try {
            withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials', gitTool
                sh "git clone ${GITLAB_BASE_URL}.git ."
                sh "git checkout develop"

                sh """
                    git fetch --all --prune
                    git remote update --prune
                    git gc --prune=now
                """

                BRANCH_NAME = sh(script: """
                    git ls-remote --sort=-committerdate origin 'refs/heads/*' |
                    awk -F'/' '{print substr(\$0, index(\$0,\$3))}' |
                    head -n 1
                """, returnStdout: true).trim()
                echo "Latest branch: ${BRANCH_NAME}"

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

                sh "git checkout -B ${BRANCH_NAME} origin/${BRANCH_NAME} --force"

                GIT_COMMIT_SHORT = sh(script: "git rev-parse --short HEAD", returnStdout: tr
                DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"
                echo DOCKER_TAG
            }
        } catch (Exception e) {
            echo "Error during clone: ${e.message}"
```

```
                    ERROR_MSG = "Failed to clone repository"
                    error ERROR_MSG
                }
            }
            AUTHOR = sh(script: "git log -1 --pretty=format:%an", returnStdout: true).trim()
            COMMIT_MSG = sh(script: 'git log -1 --pretty=%B', returnStdout: true).trim()
            COMMIT_HASH = sh(script: "git log -1 --pretty=format:%H", returnStdout: true).trim()
        }
        script {
            if (sh(
                script: "git ls-tree -d origin/${BRANCH_NAME} Back",
                returnStatus: true
            ) != 0) {
                currentBuild.result = 'ABORTED'
                ERROR_MSG = "Back 디렉토리가 원격 브랜치에 존재하지 않음"
                error ERROR_MSG
            }

            if (!fileExists("Back")) {
                ERROR_MSG = "Back 디렉토리가 로컬에 존재하지 않음"
                error ERROR_MSG
            }

            if (COMMIT_MSG.toLowerCase().contains('[fe]')) {
                ERROR_MSG = "FE 커밋으로 빌드 중단"
                error ERROR_MSG
            }
        }
    }
}
stage('Inject Config') {
    steps {
        script {
            STAGE_NAME = "Inject Config (2/9)"
        }
        withCredentials([
            file(credentialsId: 'spring-config', variable: 'CONFIG_FILE'),
            file(credentialsId: 'test-server-cert', variable: 'TEST_CERT'),
            file(credentialsId: 'prod-server-cert', variable: 'PROD_CERT')
        ]) {
            sh """
                mkdir -p ${BACKEND_DIR}/src/main/resources
                cp \$CONFIG_FILE ${BACKEND_DIR}/src/main/resources/application.yml
                cp \$TEST_CERT ${WORKSPACE}/test-cert.p12
                cp \$PROD_CERT ${WORKSPACE}/prod-cert.p12
            """
        }
    }
```

```
        }
        stage('Build') {
            steps {
                script {
                    STAGE_NAME = "Build (3/9)"
                }
                dir(BACKEND_DIR) {
                    sh "chmod +x gradlew"
                    script{
                        try {
                            sh "./gradlew clean build -x test"
                        } catch(Exception e) {
                            ERROR_MSG = e.getMessage()
                            error ERROR_MSG
                        }
                    }
                }
            }
            post {
                failure {
                    cleanWs()
                    script {
                        ERROR_MSG += "\nBuild failed"
                        error ERROR_MSG
                    }
                }
            }
        }
        stage('Docker Build') {
            steps {
                script {
                    STAGE_NAME = "Docker Build (4/9)"
                    IMAGE_NAME += (BRANCH_NAME == 'develop') ? "-dev" : "-sub"
                    def jarFile = sh(script: "ls Back/build/libs/*.jar", returnStdout: true).trim()
                    try {
                        docker.build("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}-test", "--no-ca
                        docker.build("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}", "--no-cache -
                        sh "docker save ${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}-test | gzip >
                    } catch(Exception e) {
                        ERROR_MSG = e.getMessage()
                        error ERROR_MSG
                    }
                }
            }
            post {
                failure {
                    cleanWs()
                    script {
```

```
                ERROR_MSG += "\nDocker Build failed"
                error ERROR_MSG
            }
        }
    }
}
stage('Deploy to Test') {
    steps {
        script {
            STAGE_NAME = "Deploy to Test (5/9)"
            def currentBranch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).tr
            sshagent(['gcloud-ssafy']) {
                sh """
                    set -e
                    rsync -av --progress -e 'ssh -o StrictHostKeyChecking=no' -W image.tar.gz plak

                    sleep 1

                    local_size=\$(stat -c%s image.tar.gz)
                    remote_size=\$(ssh -o StrictHostKeyChecking=no plaksharp@${TEST_SERVER} '

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

                    sleep 1

                    ssh -o StrictHostKeyChecking=no plaksharp@${TEST_SERVER} "
                        set -e
                        gunzip -c image.tar.gz | docker load
                        docker stop backend || true
                        docker rm backend || true
                        docker run -d --network host --name backend -e SERVER_SSL_KEY_STORE=/a
                        rm image.tar.gz
                    "
                """
            }
            echo "Success Test deployment."
        }
    }
    post {
        failure {
            cleanWs()
            script {
                ERROR_MSG = "Test deployment failed"
                error ERROR_MSG
            }
```

```
            }
        }
    }
    stage('Test Health Check') {
        steps {
            script {
                STAGE_NAME = "Test Health Check (6/9)"
                def maxRetries = 5
                def timeout = 10
                def success = false

                for (int i = 0; i < maxRetries; i++) {
                    sleep(timeout)
                    try {
                        def response = httpRequest "https://${TEST_SERVER}:8080/actuator/health"
                        if (response.status == 200) {
                            success = true
                            break
                        }
                    } catch(e) {
                        echo "Health check attempt ${i+1} failed"
                    }
                }
                if (!success) {
                    ERROR_MSG = "Health check failed after ${maxRetries} attempts"
                    error ERROR_MSG
                } else {
                    docker.withRegistry('https://index.docker.io/v1/', 'docker-credentials') {
                        docker.image("${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}").push()
                    }
                }
            }
        }
    }
    stage('Deploy to Prod') {
        steps {
            script {
                STAGE_NAME = "Deploy to Prod (7/9)"
                def currentBranch = sh(script: 'git rev-parse --abbrev-ref HEAD', returnStdout: true).tr
                sshagent(['ec2-ssafy']) {
                    sh """
                        ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "
                            docker pull ${DOCKER_USER}/${IMAGE_NAME}:${DOCKER_TAG}
                            docker stop backend || true
                            docker rm backend || true
                            docker run -d --network host --name backend -e SERVER_SSL_KEY_STORE=/a
                        "
                    """
```

```
                }
                echo "Success Production deployment."
            }
        }
        post {
            failure {
                script {
                    ERROR_MSG = "Production deployment failed"
                    error ERROR_MSG
                }
            }
        }
    }
    stage('Prod Health Check') {
        steps {
            script {
                STAGE_NAME = "Prod Health Check (8/9)"
                def maxRetries = 5
                def timeout = 10
                def success = false

                for (int i = 0; i < maxRetries; i++) {
                    sleep(timeout)
                    try {
                        def response = httpRequest "https://${PROD_SERVER}:8080/actuator/health"
                        if (response.status == 200) {
                            success = true
                            break
                        }
                    } catch(e) {
                        echo "Health check attempt ${i+1} failed"
                    }
                }
                if (!success) {
                    ERROR_MSG = "Health check failed after ${maxRetries} attempts"
                    error ERROR_MSG
                }
            }
        }
    }
    stage('Complete') {
        steps {
            script {
                STAGE_NAME = "Complete (9/9)"
            }
        }
    }
}
```

```
    post {
        always {
            script {
                def issueKeyPattern = /\[#(S12P11E205-\d+)]/
                def issueKey = (COMMIT_MSG =~ /S12P11E205-\d+/) ? (COMMIT_MSG =~ /S12P11E205-
                def cleanedMessage = issueKey ? COMMIT_MSG.replaceFirst(issueKeyPattern, '').trim()
                def jiraLink = issueKey ? "${JIRA_BASE_URL}/jira/software/c/projects/S12P11E205/board

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

                if (issueKey) {
                    try {
                        def requestBody = [body: message]
                        def response = httpRequest authentication: 'jira-credentials',
                            contentType: 'APPLICATION_JSON',
                            httpMode: 'POST',
                            requestBody: groovy.json.JsonOutput.toJson(requestBody),
                            url: "${JIRA_BASE_URL}/rest/api/2/issue/${issueKey}/comment"
                        echo "JIRA comment added successfully. Status: ${response.status}"
                    } catch(e) {
                        echo "JIRA 코멘트 추가 실패: ${e.message}"
                    }
                }
                message += (currentBuild.currentResult == 'ABORTED' ? "- **사용자 취소**\n" : "")
                message += "- 상세: " + (currentBuild.currentResult == 'SUCCESS' ? ":jenkins7:" : (curre
                message += jiraLink ? " | :jira: [Jira](${jiraLink}) " : (cleanedMessage.contains('Merge') ?
                message += "\n\n`${env.BUILD_TIMESTAMP}`"

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

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

                cleanWs(cleanWhenNotBuilt: false,
```

```
                deleteDirs: true,
                disableDeferredWipeout: true,
                notFailBuild: true)
        }
    }
  }
}
```

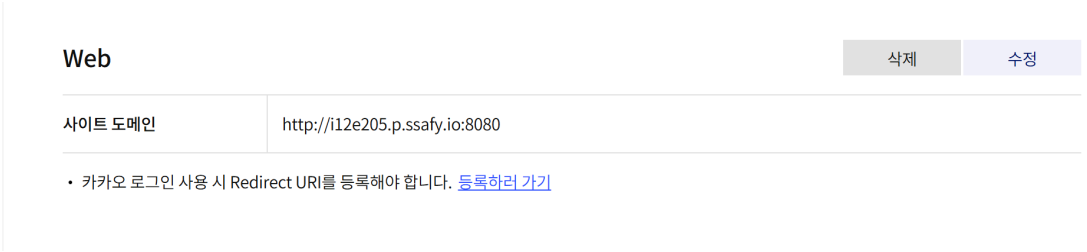
3.3 Dockerfile

```
FROM azul/zulu-openjdk:17.0.14-17.56
WORKDIR /app
# 필요한 패키지 설치
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
# 빌드된 jar 파일 복사
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
# SSL 인증서 파일 복사 및 권한 설정
ARG CERT_FILE
COPY ${CERT_FILE} cert.p12
RUN chown 1001:1001 cert.p12 && chmod 644 cert.p12
EXPOSE 8080
EXPOSE 9092
# 작업 디렉토리 권한 설정
RUN chown -R 1001:1001 /app
# 비루트 사용자로 전환
USER 1001
ENV SERVER_SSL_KEY_STORE=/app/cert.p12
# 헬스체크 설정
HEALTHCHECK --interval=30s --timeout=5s --start-period=60s --retries=3 \
  CMD curl -k -f https://localhost:8080/actuator/health || exit 1
# 애플리케이션 실행
CMD java -jar app.jar
```

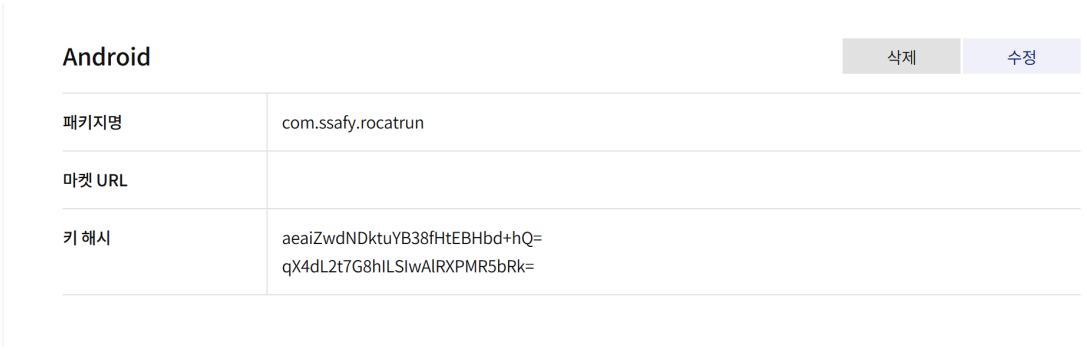
4. 외부 서비스

4.1 소셜 로그인 - Kakao

4.1.1 사이트 도메인등록



4.1.2 안드로이드 등록

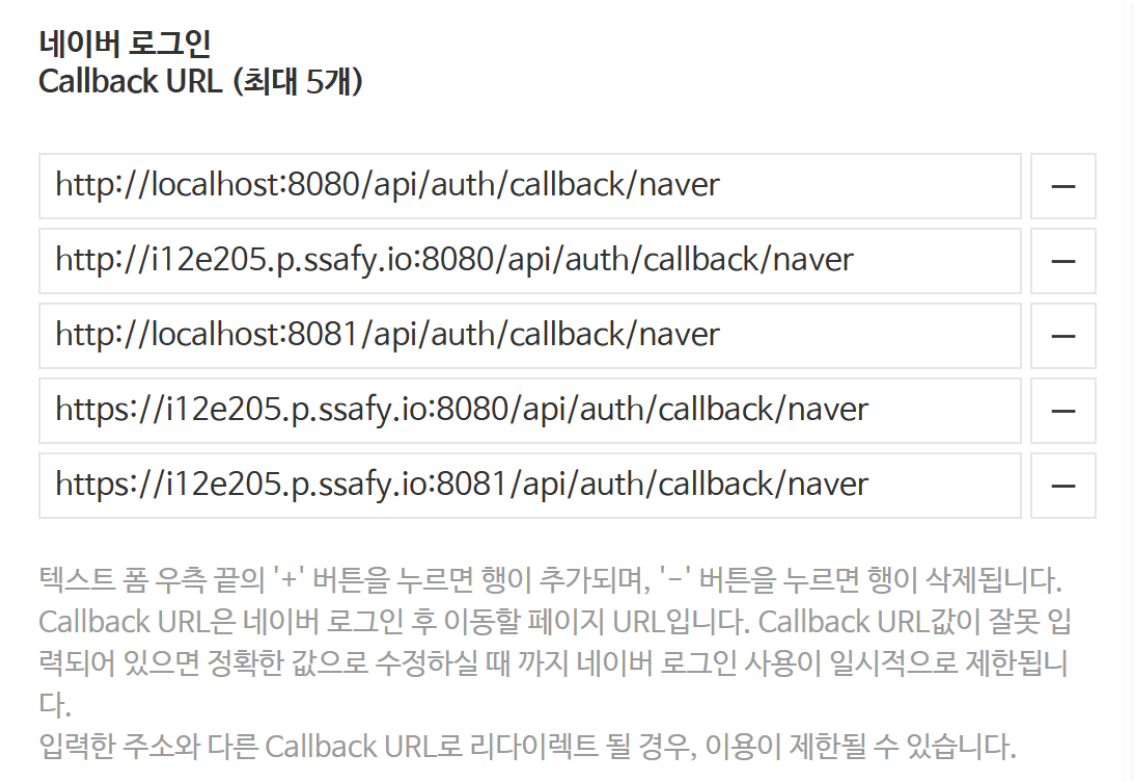


4.1.3 카카오 로그인 활성화

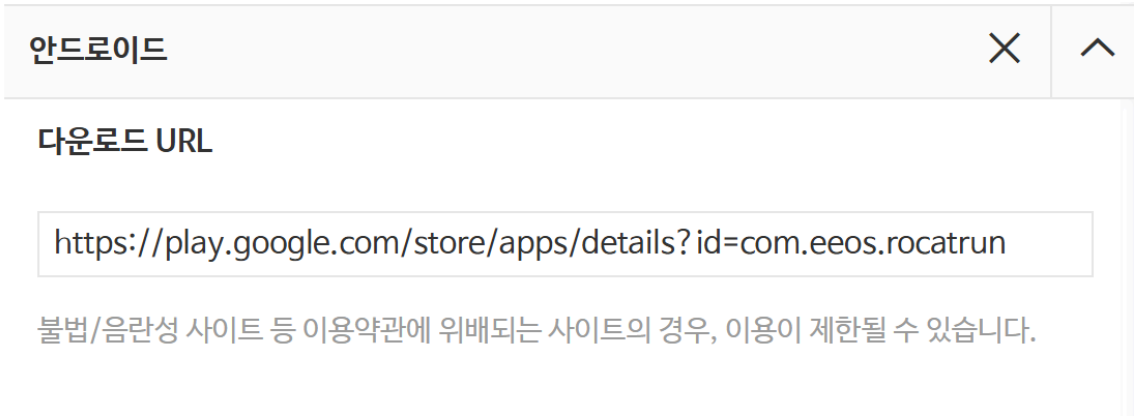
4.1.4 리다이렉트 UR설정

4.2 소셜 로그인 - Naver

4.2.1 리다이렉트 URI 지정



4.2.2 안드로이드 URL 지정



4.2.3 로고 이미지 설정

로고 이미지

파일선택

네이버 로그인 연동 과정에서 사용자에게 보여지는 이미지이므로 서비스를 대표할 수 있는 이미지로 설정해주세요.
권장 크기는 140X140 사이즈이며 500KB 이하의 jpg, png, gif만 등록 가능합니다.

4.3 소셜 로그인 - Google

4.3.1 리다이렉트 URI 설정

승인된 리디렉션 URI

웹 서버의 요청에 사용

URI 1 *
http://localhost:8080/api/auth/callback/google

URI 2 *
http://i12e205.p.ssafy.io:8080/api/auth/callback/google

URI 3 *
http://localhost:8081/api/auth/callback/google

URI 4 *
https://i12e205.p.ssafy.io:8080/api/auth/callback/google

URI 5 *
https://i12e205.p.ssafy.io:8081/api/auth/callback/google

+ URI 추가

yml

```
# OAuth2 소셜 로그인 설정
oauth2:
  kakao:
    client_id: { 카카오 id }
    redirect_uri: { 리다이렉트 주소 }
  naver:
    client_id: { 네이버 id }
    client_secret: { 네이버 시크릿키 }
```

```
    redirect_uri: { 리다이렉트 주소 }
  google:
    client_id: { 구글 id }
    client_secret: { 구글 시크릿키 }
    redirect_uri: { 리다이렉트 주소 }
```

요청 흐름

1. 프론트엔드 로그인 요청 주소

[카카오]

https://kauth.kakao.com/oauth/authorize?client_id={카카오 클라이언트 아이디}&redirect_uri={리다이렉트 uri}

[네이버]

https://nid.naver.com/oauth2.0/authorize?response_type=code&client_id={네이버 클라이언트 아이디}&redirect_uri={리다이렉트 uri}

[구글]

https://accounts.google.com/o/oauth2/v2/auth?client_id={네이버 클라이언트 아이디}&redirect_uri={리다이렉트 uri}

2. 로그인 요청 주소로 인가코드를 요청
3. 프론트엔드에서 리다이렉트 uri 로 넘어온 code = {authCode} 를 통해서 백엔드 요청을 보냄
4. 백엔드에서 넘어온 authCode 를 이용하여 카카오, 네이버, 구글에 정보를 요청해여 설정한 정보들을 받아옴
5. 정보들을 통해 db에 해당 유저가 없으면 유저를 생성해주고, 회원가입을 진행함
6. JWT accessToken을 가지고 redis에 저장한 정보와 일치하는지 확인하여 우리 서비스의 로그인을 진행함
7. 백엔드에서 넘겨준 JWT accessToken으로 우리 앱에서 로그인 및 회원가입 진행

5. 갤럭시 워치

무선 디버깅 연결

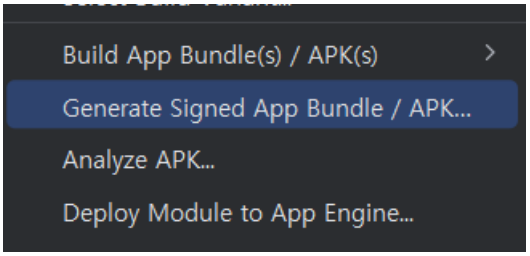
1. 워치 개발자 옵션 켜기
   - 설정 - 워치 정보 - 소프트웨어 정보 - 소프트웨어 버전 연타
2. 무선 디버깅 켜기
   - 먼저 와이파이 연결 필수
3. platform-tools 폴더의 adb로 페어링
   a. 워치 무선디버깅에서 새 기기 등록 눌러서 페어링 코드 확인
   b. 안드로이드 스튜디오의 File - Project Structure - SDK Location의 주소 복사
      - C:\Users\<사용자명>\AppData\Local\Android\Sdk\platform-tools
   c. 해당 위치로 이동 후 주소창에 cmd쳐서 명령 프롬프트 실행

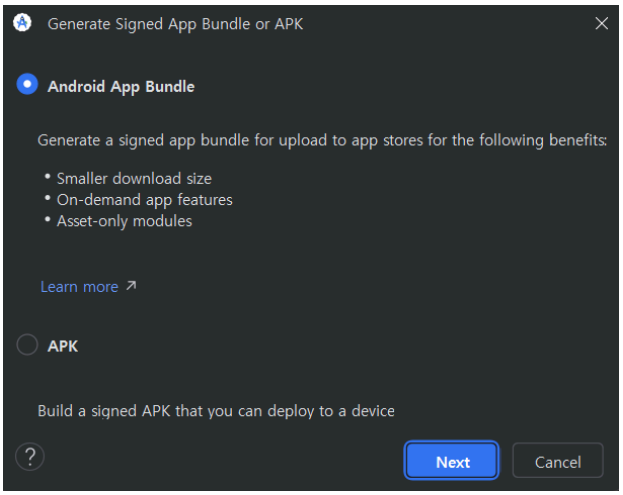
d. 또는 해당주소 시스템 환경변수로 등록
    - 시스템 → 고급 시스템 설정 → 환경변수
e. adb pair <워치 무선 디버깅 IP>:<페어링 포트 주소> <페어링 코드>

4. 페어링 성공이 떴다면 커넥트
    - adb connect <워치 무선 디버깅 IP>:<디버깅 포트 주소>
5. 안드로이드 스튜디오 Device Manager에 워치 추가해서 미러링 열기

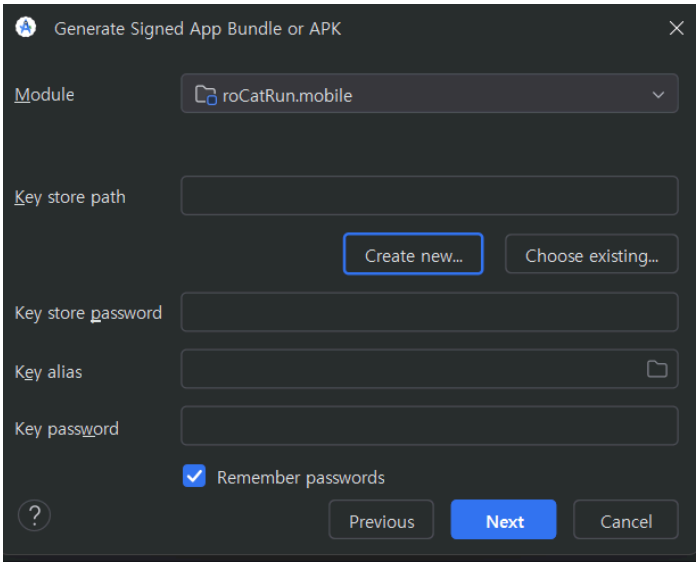
6. 앱 빌드

6.1 안드로이드 스튜디오 메뉴에서 서명된 앱 생성하기

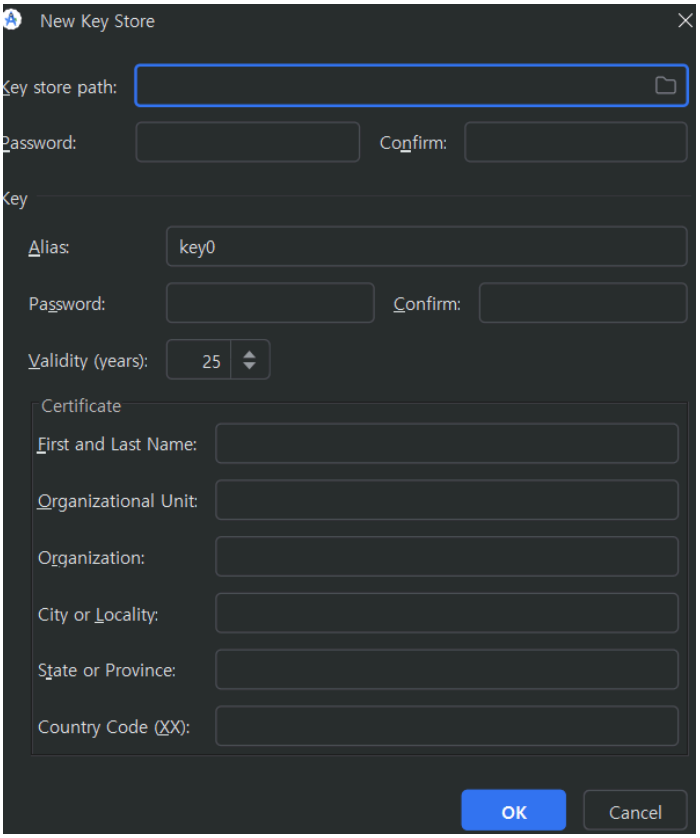




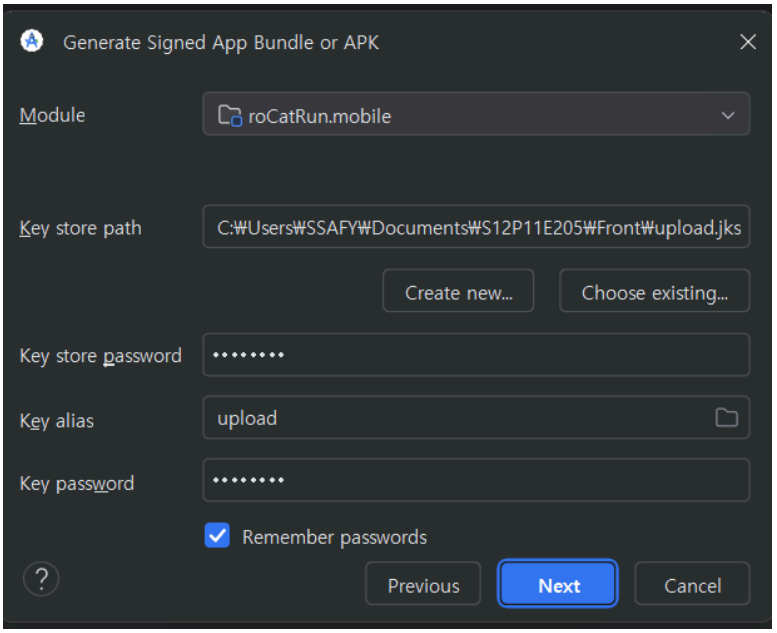
- 새로운 키 생성



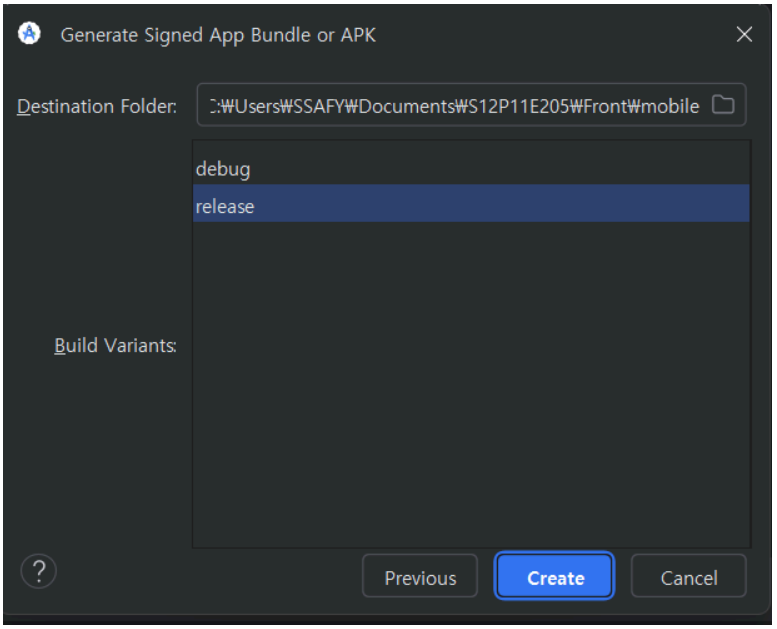
- 저장경로, 암호, 키 별칭, 키 암호(암호와 동일) 그리고 이름과 주소 작성



생성한 키 적용



릴리즈 버전으로 빌드



6.2 gradlew로 생성하기

- keystore.properties <프로젝트 루트에 저장>

```
storePassword=<암호>
keyPassword=<키 암호>
keyAlias=<키 별칭>
storeFile=<저장경로-keystore.properties 기준 상대 경로>
```

- build.gradle.kts에서 호출

```
import java.util.Properties
import java.io.FileInputStream
val keystorePropertiesFile = rootProject.file("keystore.properties")
val keystoreProperties = Properties()
keystoreProperties.load(FileInputStream(keystorePropertiesFile))
```

- 빌드 실행

```
./gradlew clean build
```

빌드 완료된 app bundle 구글 play console 사용해서 올리기