SISEON 포팅 매뉴얼

▼ 목차

---

1. 개발환경

1.1. Frontend

- frontend:
    - basics:
        - flutter: stable
        - dart: ">=3.8.1"
        - android_sdk_platform: 34
        - jdk: 11
        - ndk: "27.0.12077973"
        - namespace: "com.hmh.siseon.siseon2"
        - minSdk: 24
        - compileSdk: 34
- dependencies:
    - core:
        - flutter
        - firebase_core: "^3.4.0”

        - firebase_messaging: "^15.0.0"
        - shared_preferences: "^2.2.3"
        - http: "^0.13.0"
        - intl: "^0.20.2"
    - io_connectivity:
        - flutter_blue_plus: "^1.35.5"
        - mqtt_client: "^10.10.0"
        - permission_handler: "^11.0.1"
        - flutter_foreground_task: "^9.1.0"
    - ui_charts_ux:
        - fl_chart: "^1.0.0"
        - flutter_spinkit: "^5.2.0"
        - flutter_joystick: "^0.2.2"
    - auth:
        - google_sign_in: "^6.2.1"
    - config_utils:
        - flutter_dotenv: "^5.0.2"
        - jwt_decoder: "^2.0.1"

1.2. Backend

- Java
    - Java OpenJDK 17
    - Spring Boot 3.5.3
        - Spring Data JPA 3.5.3
        - Spring Security 3.5.3
        - Spring Batch 3.5.3
        - Spring Oauth2
            - oauth2-jose 6.4.3
            - oauth2-authorization-server 1.3.3
            - oauth2-resource-server 3.5.3
        - Spring MQTT 6.4.3
        - Lombok 1.18.34
    - External Libraries
        - paho-mqtt 1.2.5
        - springdoc-openapi 2.2.0
        - firebase-admin 9.2.0
- Database
    - Mysql 8.0.36
- MQTT
    - EMQX 5.8.6

- CI/CD
  - Docker 28.3.2
  - Jenkins 2.520

1.3. IoT

- OS
  - Ubuntu 24.04
- C++
  - g++ 13.3.0
- ROS2 Jazzy Jalisco
- Librairies
  - paho-mqtt-cpp
  - nlohmann_json
  - rclcpp
  - std_msgs
  - geometry_msgs
  - sensor_msgs
  - xacro
  - robot_state_publisher
  - joint_state_publisher
  - tf2_ros
  - dynamixel_sdk

1.4. AI

- OS
  - Ubuntu 22.04
- Python 3.10.12
  - torch 2.8.0
  - torchvision 0.23.0
  - ultralytics 8.3.174
  - roboflow 1.2.3
  - opencv-python 4.9.0.80

1.5. IDE

- Frontend
  - android studio
- Backend
  - Intellij Ultimate
  - Vscode
- AI
- IoT

- Cursor

1.6. 형상관리 / 이슈관리

- Jira
- GitLab

2. 환경변수

2.1. Frontend

```
====.env====
# MQTT
MQTT_HOST={MQTT 주소}
MQTT_PORT={MQTT TLS 포트, default=8883}
MQTT_USERNAME={MQTT 아이디}
MQTT_PASSWORD={MQTT 비밀번호}
MQTT_CERT={TLS 인증용 cert.pem 파일, assets/certs/cert.pem}

====android/app/google-services.json====
# Firebase FCM 알림용 json 파일(프론트)
{
  "project_info": {
    "project_number": "...",
    "project_id": "...",
    "storage_bucket": "..."
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "...",
        "android_client_info": {
          "package_name": "..."
        }
      },
      "oauth_client": [],
      "api_key": [
        {
          "current_key": "..."
        }
      ],
      "services": {
        "appinvite_service": {
          "other_platform_oauth_client": []
        }
      }
    }
  ],
  "configuration_version": "..."
}

====Flutter run====
- git clone <YOUR_REPO_URL>
- cd frontend
- flutter clean
```

```
- flutter pub get
- flutter run
====Flutter build====
- flutter build apk --release
```

2.2. Backend

```
====.env====
# Main DB
DB_HOST={도커 DB 컨테이너의 service명}
DB_NAME={메인 DB명}
DB_USER={메인 DB 아이디}
DB_PASS={메인 DB 비밀번호}

# MQTT
MQTT_PORT={MQTT TCP 포트, default=1883} // 내부망 MQTT라 TLS 포트 사용 x
MQTT_USER={MQTT 아이디}
MQTT_PASS={MQTT 비밀번호}

# Batch DB
BATCH_DB_HOST={도커 DB 컨테이너의 service명}
BATCH_DB_NAME={Batch DB명}
BATCH_DB_USER={Batch DB 아이디}
BATCH_DB_PASS={Batch DB 비밀번호}

# JWT 시크릿
APP_JWT_SECRET={BASE64URL로 인코딩된 키 값}

====resources/firebase/serviceAccountKey.json====
# Firebase FCM 알림용 json 파일(백엔드, *프론트와 동일 계정에서 발급한 json)
{
  "type": "...",
  "project_id": "...",
  "private_key_id": "...",
  "private_key": "...",
  "client_email": "...",
  "client_id": "...",
  "auth_uri": "...",
  "token_uri": "...",
  "auth_provider_x509_cert_url": "...",
  "client_x509_cert_url": "...",
  "universe_domain": "..."
}
```

2.3. AI

```
====siseon-chatbot폴더의 .env====

# JWT 서명, Spring 서버에서 설정한 값 불러오기
SPRING_BASE_URL=http://{서버 도메인 이름}:{Spring 서버 포트번호, default=8080}
JWT_SECRET_KEY={BASE64URL로 인코딩된 키 값}
JWT_ALGORITHM=HS256
JWT_ISSUER={Spring서버에서 설정한 ISSUER, 현재 설정 값=siseon}
```

```
# OpenAI
OPENAI_API_KEY={OPENAPI의 API KEY, 현재는 SSAFY GMS의 KEY 사용}
OPENAI_API_BASE={OPENAPI의 응답 BASE_URL을 사용, 현재는 SSAFY GMS의 BASE_URL을 사용}

# LLM 모델 설정
GMS_MODEL=gpt-4.1
TEMPERATURE=0.0
MANUAL_PATH=/app/manuals/siseon_manual.pdf

# Database
DB_HOST={도커 DB 컨테이너의 service명}
DB_PORT={도커 DB 컨테이너의 포트번호, default=3306}
DB_NAME={챗봇 DB명}
DB_USER={챗봇 DB 아이디}
DB_PASSWORD={챗봇 DB 비밀번호}


====AI 디렉토리의 configs/default.yaml====

# 공통 설정
common:
  seed: 42
  gpu_id: 0
  paths:
    output_root: 'runs'

# 포즈 추정 설정
pose:
  dataset_yaml: 'pose_estimation/data/FLIC_yolo/flic_upperbody.yaml'
  weights: 'yolo11n-pose.pt'
  epochs: 200
  imgsz: 640
  batch: 16
  freeze: 10
  optimizer: 'AdamW'
  project_name: 'pose'
  run_name: 'flic_upperbody_y11n'

# 시선 추적(학습/사전학습) 설정
gaze:
  data:
    api_key: '...'
    workspace: 'eye-annotations-yolo-to-voc'
    project_name: 'eye-detection-kso3d'
    version: 4
    data_fmt: 'yolov11'
  pretrain:
    api_key: 'MF8Wd7JxbRUZSmQTC9fw'
    workspace: 'iseeds-workspace'
    project_name: 'iseeds'
    version: 8
    data_fmt: 'yolov11'
    img_size: 640
    batch_size: 16
    workers: 4
```

```
    epochs: 500

# 모델 설정
model:
  name: 'yolo11n.pt'
  epochs: 500
  img_size: 416
  batch_size: 16
  workers: 4

# 추론/인퍼런스 설정
inference:
  conf_thresh: 0.25          # gaze 모델 confidence threshold
  webcam_index: 0             # 사용할 웹캠 인덱스
  cam_width: 640              # 캡처 프레임 너비
  cam_height: 480             # 캡처 프레임 높이
  weights: 'weights/best_gaze.pt'      # gaze 모델 weights 위치
  pose_weights: 'weights/best_pose.pt' # pose 모델 weights 위치

# 하이퍼파라미터 최적화
hpo:
  n_trials: 20
  timeout: 3600
  trial_epochs: 1
  pruner:
    type: median
    warmup_steps: 5
```

2.4. IoT

```
# 디바이스 정보
export DEVICE_ID={디바이스의 시리얼 번호}

# MQTT 관련 설정
export MQTT_HOST={서버 도메인 이름}
export MQTT_PORT={MQTT TLS 포트, default=8883}
export MQTT_PROTOCOL="mqtts"
export MQTT_USER={MQTT 아이디}
export MQTT_PASSWD={MQTT 비밀번호}
export MQTT_CA_CERT={TLS 인증용 cert.pem 파일, /home/b101/cert.pem}
export MQTT_KEEP_ALIVE=60
```

3. EC2 settings

```
ubuntu@ip-172-26-7-184:~$ sudo docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED             STATUS                  PORTS

          NAMES
2f878cff848f   app_backend       "java -jar /app/app.…"   About an hour ago   Up About an hour        0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp

          siseon-was
fe416df16d62   mysql:8.0.36      "docker-entrypoint.s…"   About an hour ago   Up About an hour        0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp

          siseon-db
409a31db0d1b   app_chatbot       "uvicorn app:app --h…"   44 hours ago        Up 44 hours             0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp

          chatbot
4ca9e7b782fe   ci-jenkins        "/usr/bin/tini -- /u…"   5 days ago          Up 5 days               0.0.0.0:50000->50000/tcp, [::]:50000->50000/tcp, 0.0.0.
0:13570->8080/tcp, [::]:13570->8080/tcp
          jenkins
7670d0bdd762   emqx/emqx:5.8.6   "/usr/bin/docker-ent…"   2 weeks ago         Up 2 weeks (unhealthy)   4370/tcp, 0.0.0.0:1883->1883/tcp, [::]:1883->1883/tcp,
0.0.0.0:8083-8084->8083-8084/tcp, [::]:8083-8084->8083-8084/tcp, 0.0.0.0:8883->8883/tcp, [::]:8883->8883/tcp, 0.0.0.0:18083->18083/tcp, [::]:18083->18083/tcp,
 5369/tcp   emqx
```

3.1. Jenkins

- CI/CD 컨테이너 배포

```
# Jenkins Dockerfile
FROM jenkins/jenkins:jdk17

USER root
RUN apt-get update &&\
    apt-get upgrade -y &&\
    apt-get install -y openssh-client
```

```
# Jenkins docker-compose.yml
version: "3.9"
services:
  jenkins:
    container_name: jenkins
    build:
      context: jenkins-dockerfile
      dockerfile: Dockerfile
    restart: unless-stopped
    user: root
    ports:
      - "13570:8080"
      - "50000:50000"
    environment:
      - JENKINS_OPTS=--prefix=/jenkins
    volumes:
      - /var/jenkins_home:/var/jenkins_home
      - /home/ubuntu/siseon/.ssh:/root/.ssh
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/bin/docker
```

"Jenkins 컨테이너 내부에서 호스트의 Docker CLI와 Docker Daemon을 직접 사용하여 `docker compose` 명령어를 실행할 수 있도록 하는 볼륨 마운트 설정"

- /var/run/docker.sock:/var/run/docker.sock
- /usr/bin/docker:/usr/bin/docker

“environment에 JENKINS_OPTS는 Nginx(리버스 프록시)를 통해 https 접속을 위한 prefix를 설정”

- JENKINS_OPTS=--prefix=/jenkins

```
/var/jenkins_home/secrets/initialAdminPassword
```

→ 젠킨스 컨테이너를 띄운 후 초기에 설정된 비밀번호를 확인하는 경로

3.2. WAS, DB

- Spring WAS & Mysql DB

```
# Dockerfile

# ┌── 빌드 스테이지: Gradle + JDK17
FROM gradle:7.6-jdk17 AS builder
WORKDIR /home/gradle/project

# 의존성 캐시
COPY build.gradle settings.gradle gradlew ./
COPY gradle gradle

# gradlew 실행 권한 추가
RUN chmod +x ./gradlew

# 의존성 다운로드 및 빌드 준비
RUN ./gradlew --no-daemon clean assemble

# 실제 소스 복사 후 JAR 생성
COPY src src
RUN ./gradlew --no-daemon bootJar -x test

# └── 런타임 스테이지: OpenJDK17 슬림
FROM openjdk:17-jdk-slim
WORKDIR /app

# 빌드 결과물만 복사
COPY --from=builder /home/gradle/project/build/libs/*.jar app.jar

ENTRYPOINT ["java", "-jar", "/app/app.jar"]
```

```
#docker-compose.yml

version: '3.9'

services:
  db:
    image: mysql:8.0.36
    container_name: siseon-db
    restart: unless-stopped
    environment:
      TZ: Asia/Seoul
      JAVA_TOOL_OPTIONS: "-Duser.timezone=Asia/Seoul"
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}
    command: --default-time-zone=Asia/Seoul
    volumes:
      - db_data:/var/lib/mysql
```

```
    networks:
      - siseon-net
    ports:
      - "3306:3306"

  backend:
    build: .
    container_name: siseon-was
    restart: unless-stopped
    env_file:
      - .env
    environment:
      TZ: Asia/Seoul
    depends_on:
      - db
    networks:
      - siseon-net
    ports:
      - "8080:8080"

volumes:
  db_data:

networks:
  siseon-net:
    external: true
```

networks는 `siseon-net` 으로 Spring 서버와 DB를 연결

```
environment:
  TZ: Asia/Seoul
  JAVA_TOOL_OPTIONS: "-Duser.timezone=Asia/Seoul"
```

→ 타임 존을 **UTC(Default)**에서 KST로 변경해야 서비스 시간대에 맞춰서 데이터를 관리할 수 있음

```
# Jenkinsfile

pipeline {
  agent any

  environment {
    COMPOSE_IMAGE = 'docker/compose:1.29.2'
    SERVICE_DIR   = "${WORKSPACE}/siseon-backend"
  }

  stages {
    stage('Checkout') {
      steps { checkout scm }
    }

    stage('Build & Deploy') {
      steps {
        sh """
          # Compose down
          docker run --rm \\
            -v /var/run/docker.sock:/var/run/docker.sock \\
```

```
          -v "${SERVICE_DIR}":/app \\
          -w /app \\
          ${COMPOSE_IMAGE} down

        # Compose up
        docker run --rm \\
          -v /var/run/docker.sock:/var/run/docker.sock \\
          -v "${SERVICE_DIR}":/app \\
          -w /app \\
          ${COMPOSE_IMAGE} up -d --build
      """
    }
  }
 }

 post {
   success { echo '✅ 배포 완료!' }
   failure { echo '❌ 배포 실패...' }
 }
}
```

→ GitLab에서 Jenkins파일을 상위 디렉토리에 두고,

새로운 commit이 발생하면 WAS서버 및 DB업데이트 가능

- FAST API

```
# Dockerfile

FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 타임존 설정 추가 (KST)
RUN apt-get update && apt-get install -y --no-install-recommends tzdata \
 && ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime \
 && echo "Asia/Seoul" > /etc/timezone \
 && rm -rf /var/lib/apt/lists/*

# 매뉴얼 포함 및 존재 검증 (없으면 빌드 실패)
RUN mkdir -p /app/manuals
COPY manuals/siseon_manual.pdf /app/manuals/siseon_manual.pdf
RUN test -f /app/manuals/siseon_manual.pdf || (echo "❌ Missing manuals/siseon_manual.pdf" && exit 1)
ENV MANUAL_PATH=/app/manuals/siseon_manual.pdf

# 앱 소스 복사
COPY . /app

EXPOSE 8000
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

→ FAST API 역시 유저가 챗봇과의 실시간 채팅이 필요하므로, 한국(KST) 시간대 설정이 필요

```
#docker-compose.yml

version: '3.9'

services:
  chatbot:
    build:
      context: ./chatbot
      dockerfile: Dockerfile
    container_name: chatbot
    env_file:
      - ./chatbot/.env
    environment:
      UVICORN_LOG_LEVEL: info
      LOGLEVEL: INFO
      PYTHONUNBUFFERED: "1"
      TZ: Asia/Seoul
    ports:
      - "8000:8000"
    networks:
      - siseon-net
    restart: unless-stopped
    volumes:
      - hf_cache:/root/.cache/huggingface

networks:
  siseon-net:
    external: true

volumes:
  hf_cache:
```

```
# Jenkinsfile

pipeline {
  agent any

  environment {
    COMPOSE_IMAGE = 'docker/compose:1.29.2'
    SERVICE_DIR   = "${WORKSPACE}"
  }

  stages {
    stage('Checkout') {
      steps { checkout scm }
    }

    stage('Deploy (Compose)') {
      steps {
        sh """
          # compose down
```

```
        docker run --rm \
          -v /var/run/docker.sock:/var/run/docker.sock \
          -v "${SERVICE_DIR}":/app \
          -w /app \
          ${COMPOSE_IMAGE} down

        # compose up --build
        docker run --rm \
          -v /var/run/docker.sock:/var/run/docker.sock \
          -v "${SERVICE_DIR}":/app \
          -w /app \
          ${COMPOSE_IMAGE} up -d --build
      """
    }
  }
 }

 post {
  success { echo '✅ FastAPI 배포 완료' }
  failure { echo '❌ FastAPI 배포 실패' }
 }
}
```

3.3. Nginx(Reverse Proxy, TLS)

```
# 서비스 포팅 정보

# jenkins 서버
https://{EC2 도메인}/jenkins

# 백엔드 WAS 서버
https://{EC2 도메인}/siseon

# 챗봇 API
https://{EC2 도메인}/ai

# MQTT
https://{EC2 도메인}/emqx

"1883:1883"    # MQTT (TCP)
"8883:8883"    # MQTT over TLS
"18083:18083"  # EMQXDashboard
```

```
# MQTT over TLS 포트인 8883번을 제외하고 모두 닫기
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
```

```
# localhost에 띄워져있는 Docker Container 포트 정보
services:
 spring:
  ports:
   - "127.0.0.1:8080:8080" # Spring API
 fastapi:
```

```
    ports:
      - "127.0.0.1:8000:8000" # Chabot API
  jenkins:
    ports:
      - "127.0.0.1:13570:13570" # Jenkins API
  emqx:
    ports:
      - "127.0.0.1:18083:18083" # MQTT Dashboard
```

```
# /etc/nginx/conf.d/siseon.conf

upstream spring_app  { server 127.0.0.1:8080; }
upstream fastapi_app { server 127.0.0.1:8000; }
upstream jenkins_app { server 127.0.0.1:13570; }
upstream emqx_app    { server 127.0.0.1:18083; }

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

server {
  listen 80;
  server_name {EC2 도메인};

  location /.well-known/acme-challenge/ { root /var/www/certbot; }
  location / { return 301 https://$host$request_uri; }
}

server {
  listen 443 ssl http2;
  server_name {EC2 도메인};
    # TLS 버전 1.3 (Certbot)

  ssl_certificate     /etc/letsencrypt/live/i13b101.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/i13b101.p.ssafy.io/privkey.pem;

  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 /siseon/ {
    proxy_http_version 1.1;
    proxy_read_timeout 60s;
    rewrite ^/siseon/?(.*)$ /$1 break;
    proxy_pass http://spring_app;
  }

  location /chat/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade    $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_read_timeout 60s;
    rewrite ^/chat/?(.*)$ /$1 break;
    proxy_pass http://fastapi_app;
  }
```

```
  location /jenkins/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade    $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_read_timeout 3600s;
    proxy_buffering off;
    client_max_body_size 100m;
    proxy_pass http://jenkins_app/jenkins/;
  }

  location /emqx/ {
    proxy_http_version 1.1;
    proxy_read_timeout 60s;
    rewrite ^/emqx/?(.*)$ /$1 break;
    proxy_pass http://emqx_app;
  }
}
```

443 서버블록(SSL) 잠깐 비활성화

```
/etc/nginx/conf.d/siseon.conf
```

에서 **443 서버블록 전체**를 주석 처리하거나 삭제하고, **80 포트 블록만** 남기기

```
sudo nginx -t
sudo systemctl reload nginx
```

certbot로 인증서 발급(nginx 플러그인 사용)

```
sudo apt-get install -y certbot python3-certbot-nginx
sudo certbot --nginx -d i13b101.p.ssafy.io --redirect
```

- 이 명령이 **인증서를 발급**하고, 자동으로 **443 서버블록과 ssl_certificate 경로**를 conf에 추가/갱신.
- 발급이 끝나면 `nginx -t && systemctl reload nginx`

갱신 리허설 (90일 만료 후 TLS)

```
sudo certbot renew --dry-run
```

3.4. UTF Settings

```
# UTF Settings
$ sudo ufw show added
Added user rules (see 'ufw status' for running firewall):
ufw allow 22 # SSH Port
ufw allow 80 # HTTP
ufw allow 8883 # MQTT over TLS
ufw allow 443 # HTTPS
```

4. 외부서비스

```
1) Google Cloud / Firebase에 프로젝트 생성
2) Firebase 콘솔에서 Android 앱 등록 → google-services.json 다운로드 → android/app/google-services.json에
```

배치
3) Google OAuth: Google Cloud Console (APIs & Services)에서 OAuth 동의화면 구성 → OAuth 클라이언트(Android) 생성(패키지명 + SHA-1) → (앱은 google_sign_in으로 accessToken 획득)
4) Firebase: Project settings → Service accounts → Generate new private key → serviceAccountKey.json 다운로드 → 백엔드에서 안전하게 보관(Secret Manager 권장)
5) 앱은 google_sign_in으로 얻은 accessToken을 `{ "accessToken": "<...>" }` 로 `POST /siseon/api/auth/google` 전송.

4.1. Google 계정 / OAuth 설정

> 목적: Flutter(Android)에서 Google SDK(google_sign_in)로 토큰을 얻어 서버로 전달하려면 Google Cloud에서 OAuth 클라이언트를 만들어야 합니다.

1. Google Cloud Console 접속
   - URL: https://console.developers.google.com (또는 https://console.cloud.google.com)
2. 새 프로젝트 생성 또는 기존 프로젝트 선택
   - (프로젝트 이름 자유)
3. **OAuth 동의화면 구성**
   - 메뉴: APIs & Services → OAuth consent screen
   - User Type: Internal(조직 내) / External(외부 사용자) 선택
   - App name, Support email 등 필수항목 입력
   - Scopes: 기본(e.g. email, profile, openid)면 OK
   - Test users: External이고 인증 전이라면 테스트 사용자로 개발자 계정 추가
   - 저장(검증이 필요한 범위가 없으면 간단)
4. **OAuth 클라이언트 생성 (Android)**
   - 메뉴: APIs & Services → Credentials → Create Credentials → OAuth client ID
   - Application type: Android
   - 입력: Name, Package name (앱의 `applicationId`), SHA-1 (아래에서 얻는 값)
   - 생성 후 생성된 **Client ID(안드로이드용)**를 메모(필요시 사용)
5. (선택) Google Sign-In을 앱에서 `google_sign_in` 플러그인으로 사용하면, `google-services.json`을 통해 대부분 자동 설정되지만 OAuth Client(앱용)가 없으면 동작 안 됨 → 반드시 위에서 Android OAuth client 생성 + SHA-1 등록 필요.
6. **SHA-1 설정하기**

- 디버그 키 (로컬 개발)

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

- 릴리즈 키 (배포용)

```
keytool -list -v -keystore path/to/your-release.jks -alias your_alias
```

- 얻은 SHA-1 값을 OAuth 클라이언트에 등록하거나 Firebase 앱 설정에 등록하세요.

4.2. Firebase 프로젝트 & FCM 설정

> 목적: 앱에서 FCM 토큰을 받고, 백엔드가 Firebase Admin SDK로 푸시를 보내게 하기

1. Firebase Console 접속
   - URL: https://console.firebase.google.com
2. 새 프로젝트 생성 또는 기존 프로젝트 선택
3. **Android 앱 추가** (Project Overview → Add app → Android)
   - Android package name (ex. `com.company.siseon`) 입력
   - App nickname (선택), Debug signing certificate SHA-1 등록(권장)
   - 등록 완료 시 **google-services.json** 파일 다운로드
     - **파일명:** `google-services.json`
     - **앱에 배치할 위치:** `android/app/google-services.json`
4. **Firebase Authentication → Sign-in method**
   - Google provider **Enable** (필수: 구글 로그인 허용)
   - (옵션) 웹 client id 필요할 경우 표시되는 값 사용
5. **Firebase Cloud Messaging(FCM)**
   - 기본적으로 프로젝트에 포함되어 있음 → 설정 확인
6. **서비스 계정 키(백엔드용) 생성**
   - Firebase Console → Project settings → Service accounts → Generate new private key
   - **파일명:** (다운로드된 파일) 보통 `serviceAccountKey.json`
   - **중요:** 이 파일은 **서버 전용**입니다. 절대 클라이언트(앱)이나 VCS(깃)에 업로드하지 마세요.
   - **저장:** 로컬 개발: 안전한 위치(예: `~/.secrets/`), 배포: Secret Manager/GCP Secret/OS Vault/k8s Secret 등으로 주입

4.3. 필요한 파일 설정

파일	어디서 얻나	앱에 넣는 위치	백엔드에 넣는 위치(개발)	
`google-services.json`	Firebase Console → Android app 등록 후 다운로드	`android/app/google-services.json`	없음(백엔드 불필요)	
OAuth Android Client (Client ID)	Google Cloud Console → APIs & Services → Credentials	(값만 사용)	백엔드 토큰 검증용 설정에 필요 시 보관	
`serviceAccountKey.json`	Firebase Console → Project settings → Service accounts → Generate new private key	절대 넣지 마세요	개발: `src/main/resources/firebase/serviceAccountKey.json` (현재 repo 위치) → **운영에서는 제거 필요**	

4.4. Flutter 설정

1. `android/app/google-services.json` 파일 추가
2. `android/build.gradle`에 `classpath 'com.google.gms:google-services:...'` (plugin) 적용, `android/app/build.gradle` 하단에 `apply plugin: 'com.google.gms.google-services'` 추가 (일반 Firebase 세팅)
3. `pubspec.yaml`에

```
dependencies:
  google_sign_in: ^...
```

```
firebase_core: ^...
firebase_messaging: ^...
```

4. AndroidManifest / Gradle에 권한/설정(플러그인 문서 참고) — 보통 `firebase_messaging` 설치 가이드에 맞춰 설정하면 됨

---

1. Google Sign-In:

```
final account = await GoogleSignIn(scopes: ['email','profile']).signIn();
final auth = await account!.authentication;
final accessToken = auth?.accessToken;
```

4.5. Spring 설정

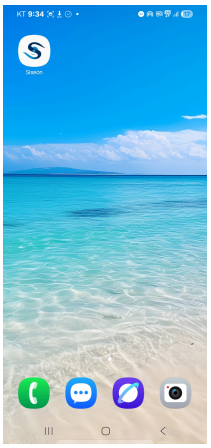
- **개발환경**: 로컬에서 `serviceAccountKey.json`

  (예: `src/main/resources/firebase/serviceAccountKey.json`)
- **운영환경**: 절대 파일을 이미지/레포에 포함하지 말고 다음 중 하나 사용:
  - CI: CI secret에 업로드 → 배포 시 런타임에 주입
- `application.yml` 예시(현재 repo 참고):

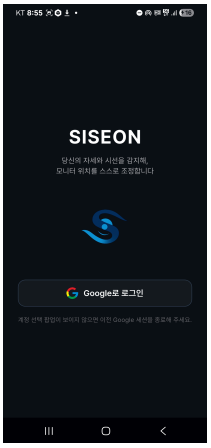
```
firebase:
  service-account-file: classpath:firebase/serviceAccountKey.json
```

5. 시연시나리오

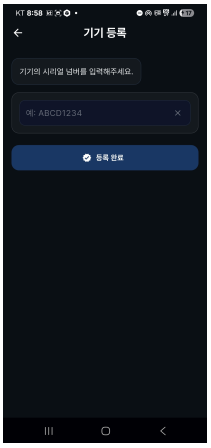
1. 앱 클릭



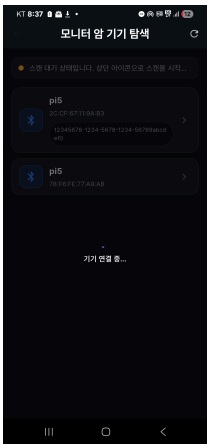
2. GOOGLE 로그인



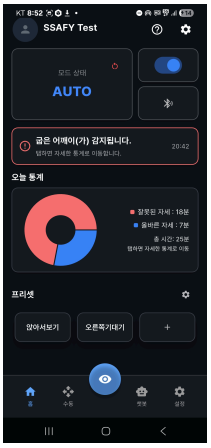
3. 기기등록



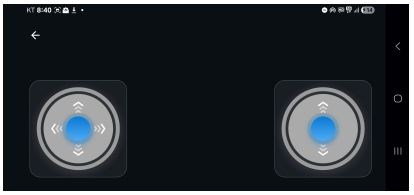
4. 블루투스 연결



5. Auto 모드

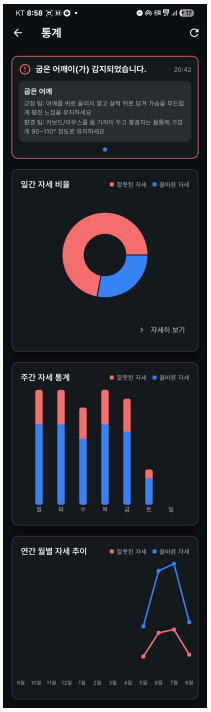


6. Manual 모드



7. 프리셋 모드

8. 통계 화면



9. 참가자 모집

10. 참가자 시연