**모바일 프로그래밍 최종 프로젝트**



|  |  |
| --- | --- |
| **과목 명** | **모바일 프로그래밍** |
| **교수 명** | **강동현 교수님** |
| **프로젝트 명** | **보행안전 지킴이** |
| **이름** | **권철현** |
| **학번** | **20173041** |
| **제출 날짜** |  |

**목차**

[1. 서론 5](#_Toc88156159)

[1.1 어플리케이션 설계의 목적 5](#_Toc88156160)

[1.2 유사 어플리케이션 5](#_Toc88156161)

[1.3 어플리케이션의 기대 효과 5](#_Toc88156162)

[2. 사용자 요구사항 정의 6](#_Toc88156163)

[2.1 제공하는 기능 6](#_Toc88156164)

[2.2 준수해야 하는 프로세스 6](#_Toc88156165)

[2.3 Use Case Diagram 6](#_Toc88156166)

[2.4 Use Case 명세 6](#_Toc88156167)

[3. 시스템 아키텍처 10](#_Toc88156168)

[3.1 Class Diagram 10](#_Toc88156169)

[3.2 Class Diagram 명세 10](#_Toc88156170)

**그림 목차**

[그림 1 : 스마트폰 사용이 보행안전에 미치는 위험성 연구 1](file:///C:\Study\3rd_year_2nd_semester\MobilePrograming\최종%20프로젝트%20문서\모바일%20프로그래밍%20최종%20프로젝트.docx#_Toc88234976)

[그림 2: 보행안전 지킴이 실행화면1 2](#_Toc88234977)

[그림 3: 보행안전 지킴이 실행화면2 3](#_Toc88234978)

[그림 4: Use Case Diagram 4](file:///C:\Study\3rd_year_2nd_semester\MobilePrograming\최종%20프로젝트%20문서\모바일%20프로그래밍%20최종%20프로젝트.docx#_Toc88234979)

[그림 5: Class Diagram 8](file:///C:\Study\3rd_year_2nd_semester\MobilePrograming\최종%20프로젝트%20문서\모바일%20프로그래밍%20최종%20프로젝트.docx#_Toc88234980)

[그림 6: 인터넷 권한 허용 13](#_Toc88234981)

[그림 7: 의존성 추가 13](#_Toc88234982)

[그림 8: LocationAPI 13](#_Toc88234983)

[그림 9: LocationInterceptor 14](#_Toc88234984)

[그림 10: Retrofit 객체 생성 14](#_Toc88234985)

[그림 11: LocationAPI 인스턴스 할당 15](#_Toc88234986)

[그림 12: download 메소드 15](#_Toc88234987)

[그림 13: Room 의존성 추가 16](#_Toc88234988)

[그림 14: 데이터베이스 테이블, Coordinate 클래스 17](#_Toc88234989)

[그림 15: CoordinateDatabase 추상 클래스 17](#_Toc88234990)

[그림 16: CoordinateDAO 인터페이스 17](#_Toc88234991)

[그림 17: Coordinate Repository 18](#_Toc88234992)

[그림 18: 싱글톤을 위한 동반객체 19](#_Toc88234993)

**표 목차**

[표 1: U1. 어플리케이션 최초 실행 4](#_Toc88234957)

[표 2: U2. 데이터 파싱 4](#_Toc88234958)

[표 3: U3. 데이터 저장 5](#_Toc88234959)

[표 4: U4. 지역 선택 5](#_Toc88234960)

[표 5: U5. 포그라운드 실행 5](#_Toc88234961)

[표 6: U6. 알림 발생 6](#_Toc88234962)

[표 7: U7. 위치 비교 6](#_Toc88234963)

[표 8: U8. 데이터 가져오기 6](#_Toc88234964)

[표 9: U9. 사용자 위치 가져오기 7](#_Toc88234965)

[표 10: MainActivity 명세 8](#_Toc88234966)

[표 11: Coordinate 명세 9](#_Toc88234967)

[표 12: MyService 명세 9](#_Toc88234968)

[표 13: MyService.ServiceThread 명세 10](#_Toc88234969)

[표 14: MyService.ServicHandler 명세 11](#_Toc88234970)

[표 15: CoordinateDatabase 명세 11](#_Toc88234971)

[표 16: CoordinateDatabase 명세 11](#_Toc88234972)

[표 17: CoordinateRepository 명세 12](#_Toc88234973)

[표 18: LocationAPI 명세 12](#_Toc88234974)

[표 19: LocationInterceptor 명세 12](#_Toc88234975)

# 1. 서론

## 1.1 어플리케이션 설계의 목적

스마트폰 사용의 증가로 인해 보행 중 스마트폰을 사용하는 사례가 점차 늘고 있다. 보행 중 스마트폰을 사용할 경우 시야폭이 56%정도 감소하고, 전방 주시 정도도 85%가량 감소한다. **그림 1**은 교통안전공단에서 실시한 ‘스마트폰 사용이 보행안전에 미치는 위험성 연구’의 결과로, 설문 응답자 중 95.7%가 보행 중 스마트폰을 사용했다고 답했다. 또한, 응답자의 5명 중 1명은 보행 중 사고가 발생할 뻔한 경험이 있다고 답했다.



출처: https://news.sktelecom.com/127208

그림 1 : 스마트폰 사용이 보행안전에 미치는 위험성 연구

한국교통안전공단에서 조사한 결과에 따르면, 스마트폰 사용 중 교통사고의 수는 2017년보다 2019년에 27% 증가했다.

이에 본 프로젝트에서는 스마트폰 사용 중 보행자 교통사고를 줄이기 위하여 보행자 사고 다발지역에서 경고 알림을 발생시키는 어플리케이션 ‘보행안전 지킴이’를 기획했다.

## 1.2 유사 어플리케이션

운전자의 경우, 네비게이션에서 교통사고 다발지역을 알려주지만, 보행자의 경우 교통사고 다발지역을 알려주는 유사한 서비스가 존재하지 않아서 본 프로젝트를 기획했다. 본 프로젝트에서는 마치 네비게이션에서 교통사고 다발지역을 알려주는 것처럼 사용자가 교통사고 다발지역에 접근하면 경고 알림을 발생시킨다.

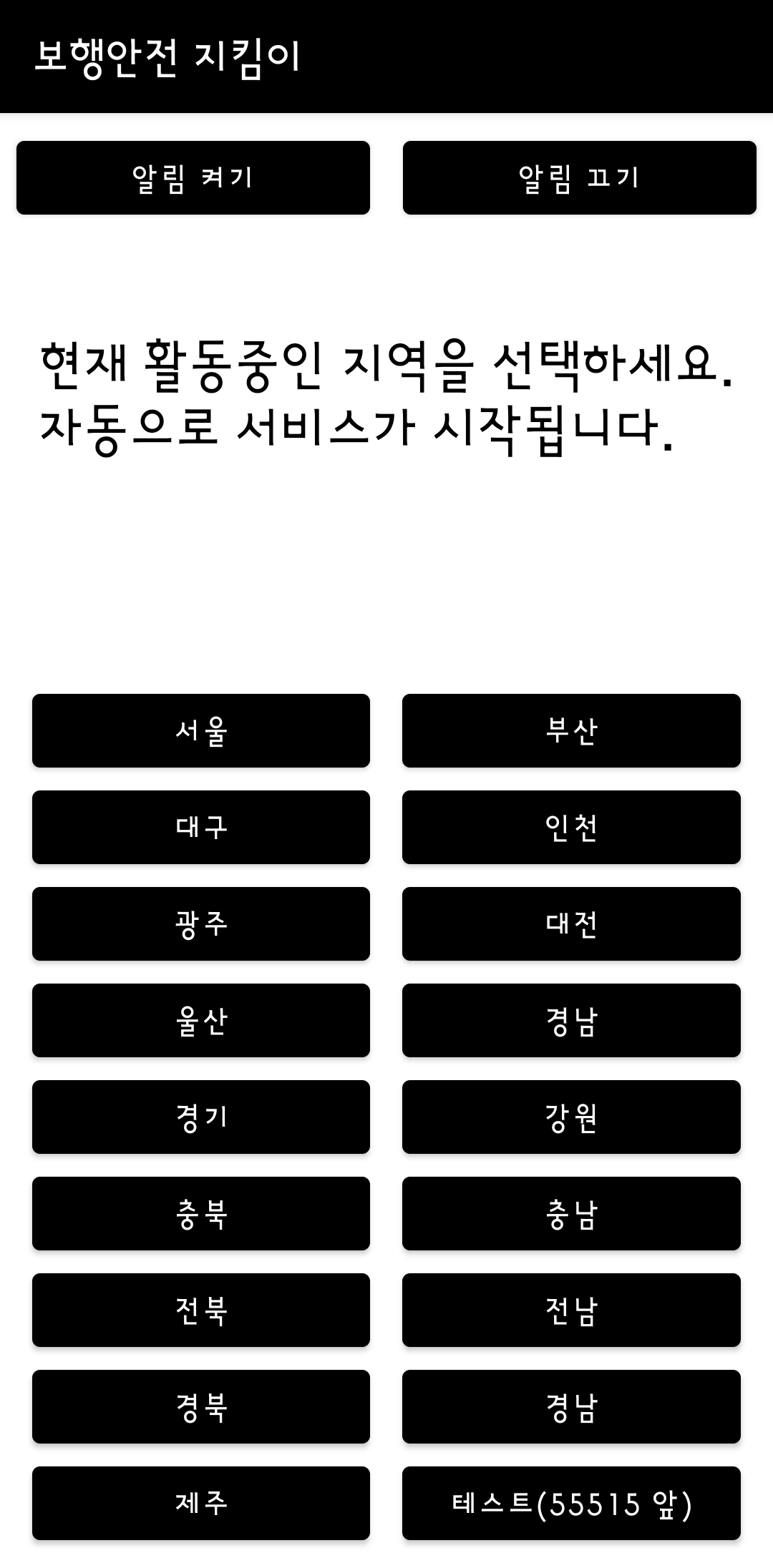
## 1.3 어플리케이션의 기대 효과

본 어플리케이션의 보행자 사고 다발지역 알림 기능을 통해 보행자 교통사고 발생률을 낮출 수 있을 것이라고 예상한다. 특히 아동의 경우, 성인에 비해 외부 자극을 감지하는 능력이 떨어지기 때문에 스마트폰으로 인한 보행 중 교통사고 위험이 더 크다. 본 어플리케이션을 통해 아동 보행사고 발생률 또한 크게 감소시킬 수 있을 것이라 기대한다.

# 2. 사용자 요구사항 정의

## 2.1 기능적 요구사항

본 어플리케이션이 제공하는 주요한 기능은 사용자가 교통사고 다발지역 인근에 접근했을 때 경고 알림을 발생시키는 것이다. 알림 발생 기능을 끄고 켤 수 있는 기능도 존재한다. 자세한 내용은 아래 그림을 참고하며 설명한다. **그림 2**는 보행안전 지킴이의 실행 화면이다.



(a)

(b)

(c)

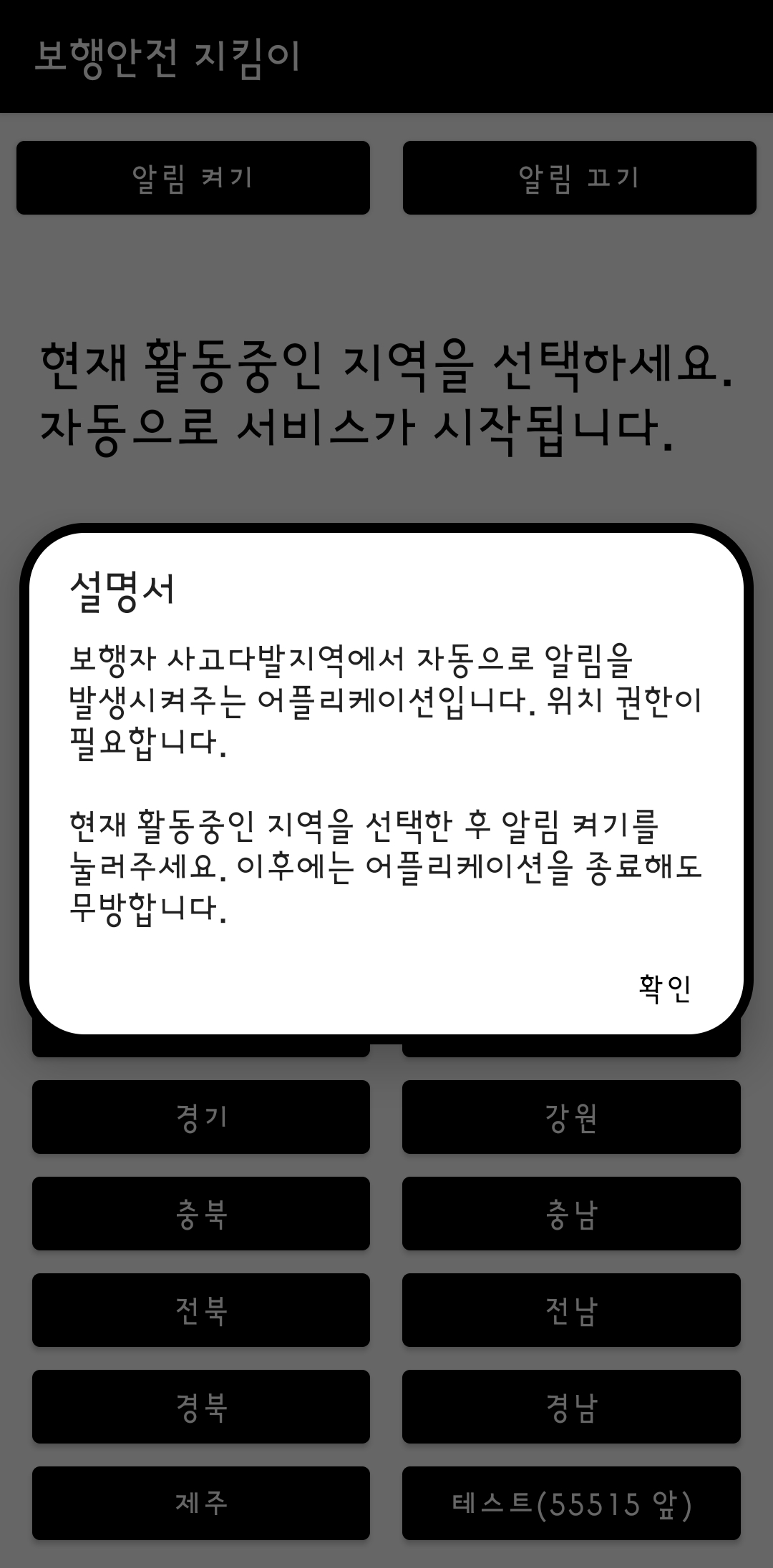
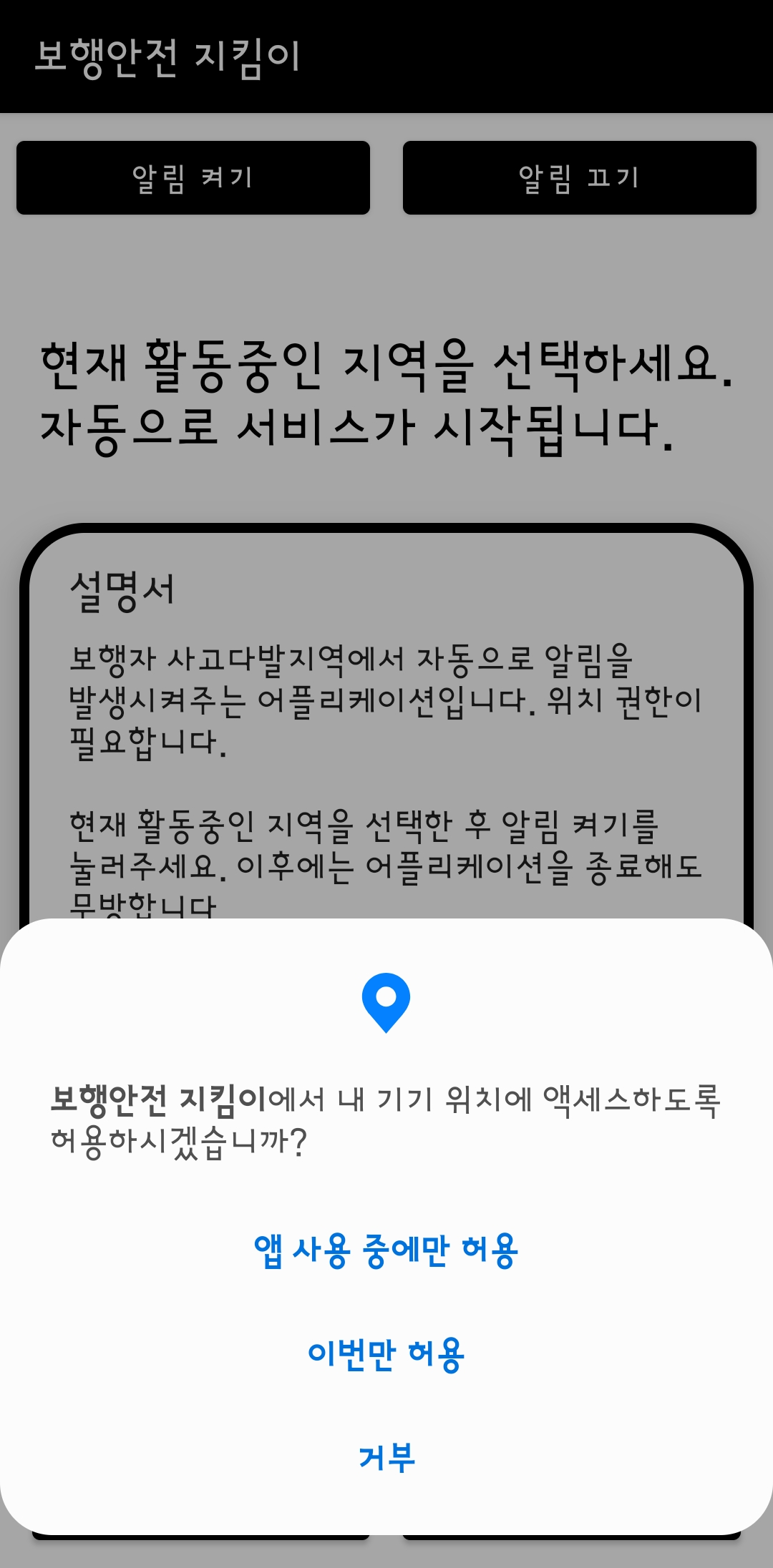


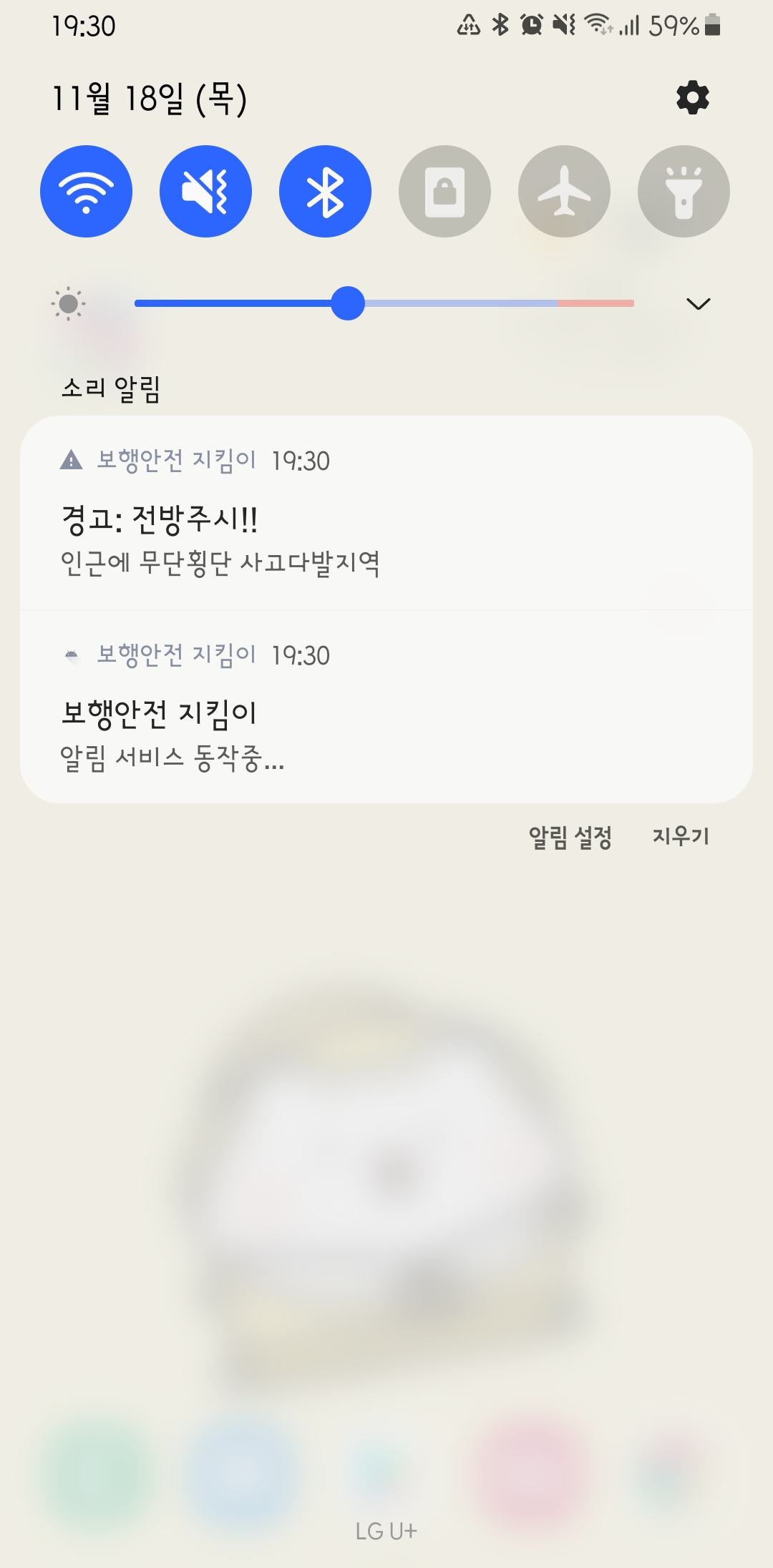
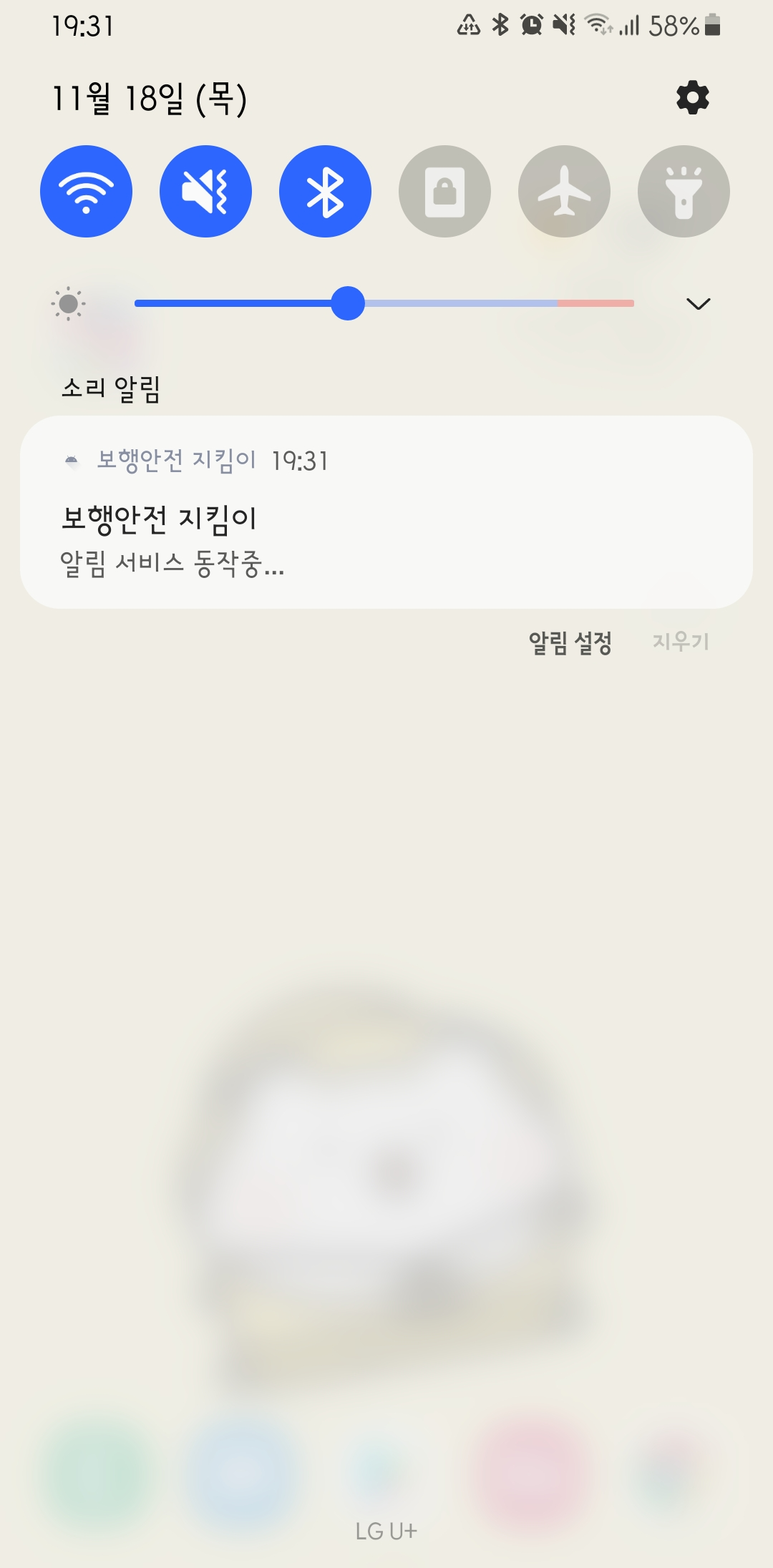
그림 2: 보행안전 지킴이 실행화면1

• 최초에 보행안전 지킴이를 실행하면 **그림 2**의 **(b)**와 같이 사용 방법을 나타낸다

• 사용자가 현재 위치하는 시, 도를 선택할 수 있다. 이는 어플리케이션의 메인 화면 **그림 2 (c)**의 하단에서 설정 가능하다.

• 사용자가 알림 기능을 켜거나 끌 수 있다. 이는 어플리케이션의 메인 화면 **그림 2 (c)**의 상단에서 설정 가능하다.

• 사용자가 교통사고 다발지역 근처에 접근하면 **그림 3**의 **(b)**와 같이 알림을 발생시킨다.



(a)

(b)

그림 3: 보행안전 지킴이 실행화면2

## 2.2 비기능적 요구사항

본 어플리케이션의 비기능적 요구사항은 아래와 같다.

• 본 어플리케이션은 안드로이드 스튜디오를 사용하고 안드로이드 기반 스마트폰 시장을 목표로 Kotlin을 사용해서 구현한다.

• 최소 API 버전은 21: Android5.1(Lollipop)이다.

• GPS를 사용하기 위해서 그림 2 (a)와 같이 사용자의 위치 사용 권한을 요청해야 한다. 권한을 거부하면 어플리케이션을 종료한다.

• GPS를 사용함으로, AVD(Android Virtual Machine) 환경이 아닌 실제 안드로이드 기기에서 동작해야 한다.

• 사용자가 어플리케이션을 종료해도 동작해야 하기 때문에 포그라운드 서비스로 구현한다.

• 포그라운드 서비스를 사용 중일때는 **그림 3 (a)**와 같이 알림을 구현해야 한다.

## 2.3 Use Case Diagram

**그림 4**는 본 어플리케이션의 Use Case Diagram이다. Use Case Diagram은 사용자(Actor)와 전체적인 어플리케이션의 기능을 나타내는 기능(Use Case)으로 나타낸다.

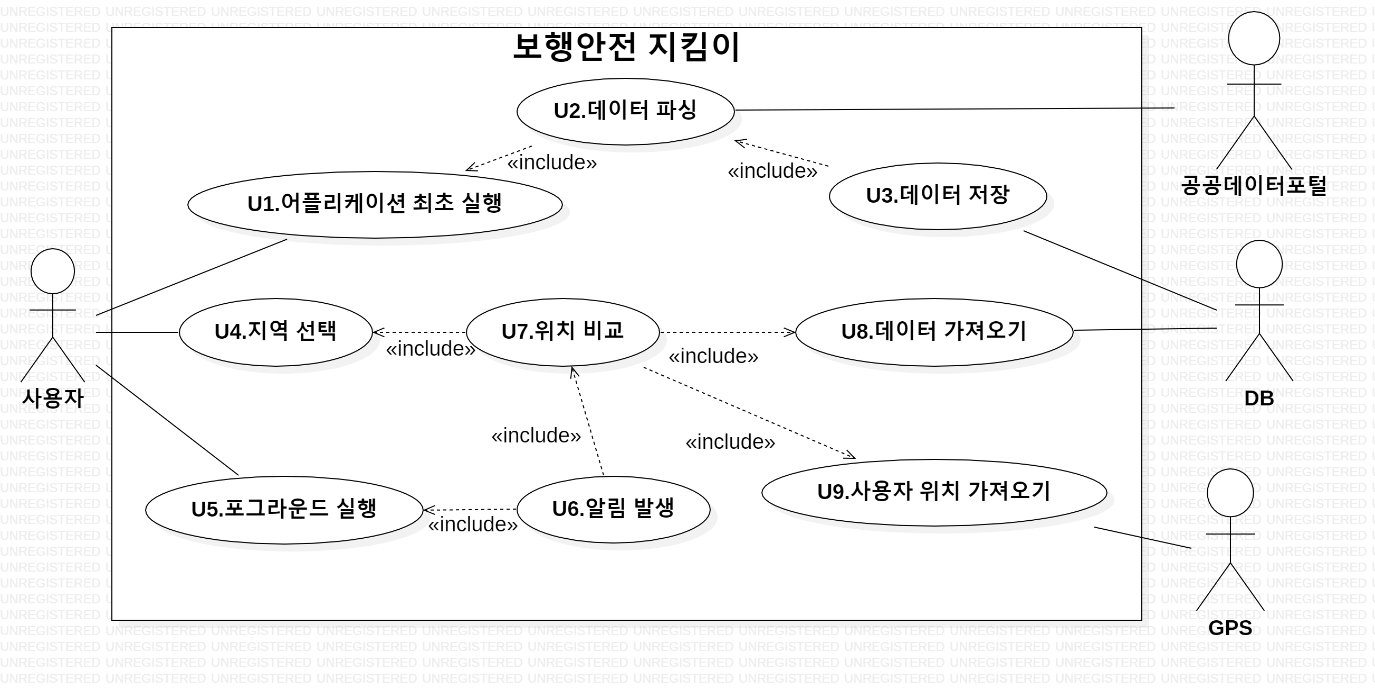


그림 4: Use Case Diagram

## 2.4 Use Case 명세

**표 1** ~ **표 9**는 앞서 나타낸 Use Case Diagram의 상세 명세이다. 각 기능이 어떤 역할을 하는지를 상세히 나타낸다.

표 1: U1. 어플리케이션 최초 실행

|  |  |
| --- | --- |
| **U1. 어플리케이션 최초 실행** | |
| **액터** | 사용자 |
| **설명** | 어플리케이션을 최초로 실행했을 때 사용 설명을 보여주고, 위치 정보 사용에 대한 권한을 요청한다. |
| **데이터** | 공공데이터 포털[1]의 주소 |
| **자극** | 사용자가 어플리케이션을 최초로 실행 |
| **반응** | 사용자의 위치정보 권한을 요청하고, 백그라운드 쓰레드에서 공공데이터 포털[1]에서 제공하는 교통사고 다발지역 데이터 다운로드를 진행한다. |
| **주석** | 사용자가 위치정보 사용을 거절하면 어플리케이션을 종료한다. |

표 2: U2. 데이터 파싱

|  |  |
| --- | --- |
| **U2. 데이터 파싱** | |
| **액터** | 공공데이터 포털[1] |
| **설명** | 공공데이터 포털[1]에서 제공하는 무단횡단 교통사고 다발지역 정보를 다운로드하는 기능이다. |
| **데이터** | 공공데이터 포털[1]의 주소 |
| **자극** | 사용자가 어플리케이션을 최초로 실행 |
| **반응** | 백그라운드 쓰레드에서 공공데이터 포털[1]에서 제공하는 교통사고 다발지역 데이터를 다운로드하고, 곧바로 어플리케이션에 내장된 데이터베이스에 좌표 정보만 골라 저장한다. API에서 여러 시, 도, 구, 군을 분리하여 데이터를 제공하기 때문에 동시에 여러 쓰레드를 실행시켜 데이터를 다운로드한다. |
| **주석** | 2019년의 데이터를 사용한다. |

표 3: U3. 데이터 저장

|  |  |
| --- | --- |
| **U3. 데이터 저장** | |
| **액터** | 공공데이터 포털[1], DB |
| **설명** | 다운받은 보행자 사고다발지역의 데이터를 기기의 데이터베이스에 저장하는 기능이다. |
| **데이터** | 보행자 사고다발지역의 좌표 |
| **자극** | 보행자 사고다발지역 데이터 다운로드 |
| **반응** | 다운로드 받은 보행자 사고다발지역 데이터 중에 위도와 경도, 시도 코드 정보를 데이터베이스에 저장한다. |
| **주석** | 테스트를 위한 좌표 하나를 추가로 저장한다. |

표 4: U4. 지역 선택

|  |  |
| --- | --- |
| **U4. 지역 선택** | |
| **액터** | 사용자 |
| **설명** | 사용자의 현재 행동 반경을 시, 도 단위로 입력 받는다. |
| **데이터** | 시도 코드 |
| **자극** | 사용자의 버튼 클릭 |
| **반응** | 사용자의 버튼입력을 통해 사용자의 현재 시, 도 코드를 공유 프리퍼런스에 저장한다. |
| **주석** | 테스트 좌표의 시도 코드는 100으로 저장한다. |

표 5: U5. 포그라운드 실행

|  |  |
| --- | --- |
| **U5. 포그라운드 실행** | |
| **액터** | 사용자, GPS, DB |
| **설명** | 사용자가 알림 발생 버튼을 누르면 포그라운드 서비스를 실행하고, 쓰레드를 반복 실행한다. |
| **데이터** | 시도 코드, 사용자 위치, 보행자 사고다발지역 좌표 |
| **자극** | 사용자의 버튼 클릭 |
| **반응** | 포그라운드 서비스를 실행하고, 포그라운드 서비스 실행 중을 나타내는 알림을 생성한다. 서비스 내에서는 쓰레드를 반복 실행한다. 쓰레드 내부의 동작은 U6에서 다룬다. |
| **주석** | 사용자가 알림 끄기 버튼을 누르면 쓰레드를 종료한다. |

표 6: U6. 알림 발생

|  |  |
| --- | --- |
| **U6. 알림 발생** | |
| **액터** | GPS, DB |
| **설명** | DB에서 가져온 사고다발지역 좌표와 사용자의 위치를 비교해서 사용자가 사고다발지역 근처에 있다면 알림을 발생시킨다. |
| **데이터** | 사용자 위치, 보행자 사고다발지역 좌표 |
| **자극** | 서비스를 실행시킨다. |
| **반응** | 쓰레드가 실행되면 사용자의 위치를 비교해 알림 발생 명령을 수행한다 20초 간격으로 쓰레드가 종료되기 전까지 무한 반복한다. 이 때 알림을 발생시키기 위한 위치 비교는 U7에서 다룬다. |
| **주석** | 사용자가 알림 끄기 버튼을 누르면 쓰레드를 종료한다. |

표 7: U7. 위치 비교

|  |  |
| --- | --- |
| **U7. 위치 비교** | |
| **액터** | GPS, DB |
| **설명** | 사용자의 현재 위치를 가져오고, 시도 코드를 활용해서 데이터베이스에서 사고다발지역의 좌표를 가져온다. 사고다발지역의 좌표에서 +-0.0003 이내에 사용자가 위치할 때 알림을 발생시킨다. |
| **데이터** | 사용자 위치, 보행자 사고다발지역 좌표 |
| **자극** | 쓰레드를 실행시킨다. |
| **반응** | 사용자의 위치와 사고다발지역의 좌표에서 위도 + 0.0003, 경도 + 0.0003 ~ 위도 – 0.0003, 경도 -0.0003 이내에 사용자의 위치가 존재하는지 비교한다. 사용자의 위치와 데이터를 가져오는 과정은 각각 U8, U9에서 설명한다. |
| **주석** | 사고다발지역이 여러 곳에 존재함으로, 루프를 통해 반복 실행한다. |

표 8: U8. 데이터 가져오기

|  |  |
| --- | --- |
| **U8. 데이터 가져오기** | |
| **액터** | DB |
| **설명** | 사용자가 입력한 시도 코드를 통해 데이터베이스에서 데이터를 검색한다. 여러 데이터를 List 형태로 받아온다. |
| **데이터** | 시도 코드 |
| **자극** | 쓰레드를 실행시킨다. |
| **반응** | 위치 비교를 위해 데이터베이스에 시도 코드를 통해 위치를 검색한다. LiveData를 통해 별도의 백그라운드 쓰레드에서 쿼리 검색을 실행한다. |
| **주석** | 데이터베이스에 대한 접근은 항상 별도의 쓰레드에서 동작해야한다. |

표 9: U9. 사용자 위치 가져오기

|  |  |
| --- | --- |
| **U9. 사용자 위치 가져오기** | |
| **액터** | GPS |
| **설명** | GPS를 통해 사용자의 위치정보를 가져온다. |
| **데이터** | 사용자 위치 |
| **자극** | 쓰레드를 실행시킨다. |
| **반응** | 위치 비교를 위해 사용자의 GPS 정보를 받아온다. PRIORITY\_HIGH\_ACCURACY 옵션을 통해 정확한 위치를 가져온다. |
| **주석** | 위치권한이 없다면 사용자의 위치정보를 가져올 수 없다. |

# 3. 시스템 아키텍처

## 3.1 Class Diagram

본 단락에서는 어플리케이션의 구조를 나타내기 위해 클래스 다이어그램을 포함한다. 클래스 다이어그램은 클래스의 속성, 동작 방식, 객체 간 관계를 표시하며 시스템의 구조를 나타낸다. **그림 5**는 해당 클래스 다이어그램이다.

## 3.2 Class Diagram 명세

본 단락에서는 3.1에서 제시한 Class Diagram에 대한 명세를 나타낸다. 아래 **표 10 ~ 20**은 해당 명세이다.

**표 10**은 MainActivity 클래스에 대한 명세이다. 해당 클래스는 안드로이드 프로젝트를 생성하면 자동으로 생성되며, AppCompatActivity()를 자동 상속한다. 해당 다이어그램에서는 이 상속 관계에 대한 내용은 나타내지 않았다. 어플리케이션의 생명 주기에 따라 전반적인 동작에 대한 내용을 포함한다.

텍스트, 점수판이(가) 표시된 사진

자동 생성된 설명

그림 5: Class Diagram

표 10: MainActivity 명세

|  |  |  |
| --- | --- | --- |
| **MainActivity** | | |
| **Attribute** | -btnStart: Button | 알림 켜기 버튼 |
| -btnStop: Button | 알림 끄기 버튼 |
| -btn1 ~ btn 18 | 각 지역을 나타내는 버튼, 총 18개의 버튼 객체이지만 편의상 묶어서 표현했음. |
| -serviceIntent: Intet | 서비스를 할당할 Intent 객체 |
| -locationAPI: LocationAPI | 데이터 파싱을 위해 생성한 retrofit2 에노테이션을 포함한 인터페이스 |
| -client: OkHttpClient | Http 통신을 위한 클라이언트, LocationInterceptor 객체가 추가되어있음. |
| -gson: Gson | Java 객체와 Json 형식 사이의 변환을 지원하는 객체 |
| -retrofit: Retrofit | 서버와 클라이언트간 Http 통신을 위한 인터페이스 |
| -repository: CoordinateRepository | 데이터베이스 연결을 위한 repository 인스턴스 |
| -locCodeArr: Array<Array<Int>> | 시도 코드와 구군 코드를 저장한 2차원 배열 |
| **Operation** | +onCreate (Bundle) | 어플리케이션의 생명주기 중 하나이다. 어플리케이션이 실행되면 동작한다. GPS 권한을 요청하고, 버튼 객체들에 대한 레이아웃 소스 할당, 최초 실행 시 설명서를 보여주고 데이터를 다운로드 하는 등 전반적인 기능을 수행하도록 하는 메소드이다. |
| -download (Int, Int) | 시도 코드, 구군 코드를 사용해서 LocationAPI를 호출하여 데이터를 다운로드하는 메소드이다. |

**표 11**은 데이터 클래스 Coordinate의 명세이다. 데이터 클래스는 Kotlin에서 지원하는 형식이며 컴파일 시에 getter, setter, 생성자를 자동으로 생성함으로 따로 정의할 필요가 없다. 교통사고 다발지역의 좌표와 시도 코드를 포함한다. 데이터베이스에서 엔티티로 사용하기 위해 @Entitiy 에노테이션을 포함한다.

표 11: Coordinate 명세

|  |  |  |
| --- | --- | --- |
| **Coordinate** | | |
| **Attribute** | -sido: Int | 시도코드 |
| -latitude: Double | 위도 |
| -longitude: Double | 경도 |

**표 12**는 포그라운드 서비스를 실행하기 위한 MyService 클래스이다. 포그라운드 서비스는 어플리케이션이 종료되어도 계속 동작하는 서비스를 제공한다. 포그라운드 서비스가 작동 중이면 필수적으로 알림을 발생시켜야 한다. 해당 클래스에서 사용자가 보행자 사고다발지역 근처에 접근했을 때 알림을 발생시킨다.

표 12: MyService 명세

|  |  |  |
| --- | --- | --- |
| **MyService** | | |
| **Attribute** | -fusedLocationProvider: FusedLocationProviderClient | 사용자 위치를 제공해주는 객체 |
| -repository: CoordinateRepository | 데이터베이스 연결을 위한 repository 인스턴스 |
| -coordinateList: LiveData<List<Coordinate>> | 데이터베이스에서 읽은 좌표 데이터를 저장할 LiveData |
| -myPosition: Coordinate | 사용자의 위치를 저장할 Coordinate 객체 |
| -thread: ServiceThread | 서비스를 반복 동작하게 만들기 위한 Thread 객체 |
| **Operation** | +onStartCommand (Intent?, Int, Int): Int | Activity의 onCreate와 같이 서비스가 시작되었을 때 동작하는 서비스의 생명주기, 포그라운드 서비스를 실행하고 쓰레드를 작동시킨다. |
| +onBind (Intent?): IBinder? | 다른 안드로이드 컴포넌트와 통신할 수 있는 서비스를 구현하기 위한 메소드, 본 어플리케이션에서는 사용하지 않기 때문에 null을 반환한다. |
| -processCommand() | 쓰레드 내부에서 실행할 작업, 사용자의 위치를 가져오고 데이터베이스에서 사고 다발지역 데이터를 가져온 뒤 비교해서 알림을 발생시키는 작업을 하는 메소드이다. |

**표 13**, **14**는 각각 MyService 내부의 inner class인 ServiceThread 클래스와 MyServiceHandler 클래스에 대한 명세이다. 코틀린에서 inner class로 클래스를 선언하면 내부 클래스로 선언되고, 외부 클래스를 참조할 수 있다. ServiceThread와 MyServiceHandler는 각각 Thread와 Handler을 상속받는다.

표 13: MyService.ServiceThread 명세

|  |  |  |
| --- | --- | --- |
| **MyService.ServiceThread** | | |
| **Attribute** | -handler: Handler | 생성자로 입력받을 Handler 객체 |
| -boolean: isRun = true | 쓰레드의 반복을 위한 Boolean 객체, 기본적으로 true 값을 가진다. |
| **Operation** | +run () | Thread 클래스에서 상속받은 메소드, isRun이 true 일 때 handler에 명령 처리 메소드를 실행한 후 20초 대기하는 동작을 무한히 반복한다. |
| +stopNow () | isRun을 false로 설정한다. |

표 14: MyService.ServicHandler 명세

|  |  |  |
| --- | --- | --- |
| **MyService.ServiceHandler** | | |
| **Operation** | +handleMessage (Message) | Handler 클래스에서 상속받은 메소드이다. MyService의 processCommand() 메소드를 실행한다. |

**표 15**는 CoordinateDatabase 추상 클래스에 대한 명세이다. Room을 활용해 안드로이드 내부 데이터베이스를 사용하기 위해 생성한다. Room을 활용하여 데이터베이스를 생성하고 사용하기 위해서는 @Entitiy 에노테이션을 포함한 클래스, @Dao(Database Access Object: 데이터베이스에 접근을 위한 객체) 에노테이션을 포함한 인터페이스를 포함해야한다. 해당 클래스에서는 Coordinate 클래스를 엔티티로 사용한다.

표 15: CoordinateDatabase 명세

|  |  |  |
| --- | --- | --- |
| ***CoordinateDatabse*** | | |
| **Operation** | +*CoordinateDAO(): CoordinateDAO* | Dao를 구현해야 함을 명시하기 위한 메소드이다. |

**표 16**은 @Dao 에노테이션을 포함하는 CoordinateDAO 인터페이스에 대한 명세이다.

표 16: CoordinateDatabase 명세

|  |  |  |
| --- | --- | --- |
| <<Interface>>  **CoordinateDatabse** | | |
| **Operation** | +getCoordinates (Int): LiveData<List<Coordinate>> | 데이터베이스에서 Coordinate를 LiveData<List<Coordinate>> 형식으로 가져오기 위한 쿼리, @Query(“SELECT \* FROM Coordinate WHERE siDo=(:siDo)” 에노테이션을 통해 쿼리의 기능을 수행한다. |
| +insertCoordinate (Coordinate) | 데이터베이스에 Coordinate를 입력하기 위한 쿼리, @Insert 에노테이션을 통해 쿼리의 기능을 수행한다. |

**표 17**은 데이터베이스와 Dao를 실체화해서 데이터베이스를 구현하고 접근할 수 있도록 하는 CoordinateRepository 클래스에 대한 명세이다. 이는 데이터베이스를 구현하는 단계의 클래스임으로, 싱글톤으로 구현한다. 싱글톤이란 어플리케이션 내에서 하나의 인스턴스만 가지는 구조를 의미한다. 싱글톤에 대한 자세한 내용은 [**4.2 데이터베이스 관련 코드**](#_4.2_데이터베이스_관련)에서 다룬다.

표 17: CoordinateRepository 명세

|  |  |  |
| --- | --- | --- |
| **CoordinateRepository** | | |
| **Attribute** | -database: CoordinateDatabase | 데이터베이스 객체 |
| -dao: CoordinateDAO | Dao 객체 |
| -executor: ExectorService! | 데이터베이스 접근을 쓰레드로 동작하기 위한 객체 |
| **Operation** | +getCoordinate (Int): LiveData<List<Coordinate>> | 데이터베이스에서 시도코드를 통해 데이터를 가져오기 위한 메소드이다. |
| +insertCoordinate (coordinate) | 데이터베이스에 Coordinate를 삽입하기 위한 메소드이다. |
| +insertTestCase() | 데이터베이스에 실제 사고다발지역이 아닌 테스트 Coordinate를 삽입하기 위한 메소드이다. |

아래 **표 18**은 retrofit2를 이용해 http 통신을 하기 위한 LocationAPI이다. **표 18**과 **표 19**에서 명세한 클래스들을 이용해 retrofit을 사용하는 http 통신은 4.1 데이터 파싱 관련 코드에서 상세하게 다룬다.

표 18: LocationAPI 명세

|  |  |  |
| --- | --- | --- |
| <<Interface>>  **LocationAPI** | | |
| **Operation** | +getLocation(String, String): Call<Location> | 네트워크 통신을 위한 Retrofit2 Call 객체를 반환하는 메소드. 메소드는 @GET(‘주소’) 에노테이션을 통해 인터넷 연결을 위한 주소를 처리한다. 메소드의 인자 String 값들은 각각 @Query 에노테이션을 통해 @Get 주소 뒤에 추가된다. |

아래 **표 19**는 LocationInterceptor 클래스의 명세이다. 해당 클래스는 okhttp3의 Intercepter 객체를 상속받는다.

표 19: LocationInterceptor 명세

|  |  |  |
| --- | --- | --- |
| **LocationInterceptor** | | |
| **Operation** | +intercept (Interceptor.Chain): Response | Interceptor 객체에서 상속받은 메소드, 통신하는 url을 가로채서 중간에 쿼리를 추가하거나 변경하는 메소드. |

이외에 표를 작성하지는 않았지만, 클래스 다이어그램에서 나타낸 객체로 Location, Items, Item 이 있다. 해당 클래스들은 공공데이터 포털[1]에서 제공하는 JSON 형식의 데이터를 java 클래스로 변환한 클래스들이며, data class 로 이루어져 있다. Location 내부에 Items 배열이 있고, Items 배열은 Item 객체를 인자로 가지는 배열이다. 본 프로젝트에서는 Item 객체의 laCrd 변수와 loCrd 변수에 저장된 위도와 경도 정보만 사용한다.

# 4. 소스 코드

본 단락에서는 프로그램의 소스코드를 기능에 따라 구분하여 설명한다. 데이터 파싱 관련기능, 데이터베이스 관련기능, GPS 관련기능 그리고 백그라운드 알림 관련기능 순으로 진행한다.

## 4.1 데이터 파싱 관련 코드

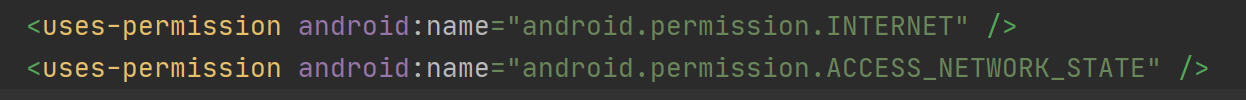
본 어플리케이션에서 활용하는 사고다발 지역의 데이터는 공공데이터 포털[1]에서 다운로드 할 수 있다. 해당 데이터를 다운로드 하여 활용하기 위해 어플리케이션의 manifests 파일에 아래 그림6과 같이 인터넷 권한을 허용하는 코드를 추가한다.

그림 6: 인터넷 권한 허용

본 프로젝트에서는 데이터를 다운받기 위해 http 통신을 사용하고, 이를 쉽게 사용하게 해 주는 retrofit2 API를 사용한다. 이를 사용하기 위해서 어플리케이션 레벨의 gradle 파일의 의존성에 아래 그림 7과 같이 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 7: 의존성 추가

retrofit은 HTTP API를 자바 인터페이스로 변경하기 때문에 아래 그림 8과 같이 LocationAPI 인터페이스를 추가한다. retfofit의 @GET 에노테이션을 통해 기본URL에 주소를 추가한다. @Query 에노테이션은 아래 소스코드에서 URL에 “siDo=’입력값’”, “guGun=’입력값’” 두 string 값을 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : LocationAPI

추가로 LocationInterceptor 클래스를 구현해야 한다. 해당 클래스는 Intercoptor 객체를 상속받으며, 이는 HTTP 통신을 진행하는 URL을 가로채서 주소를 변환시켜주는 역할을 하는 클래스이다. 자세한 내용은 MainActivity에서 마저 설명한다.

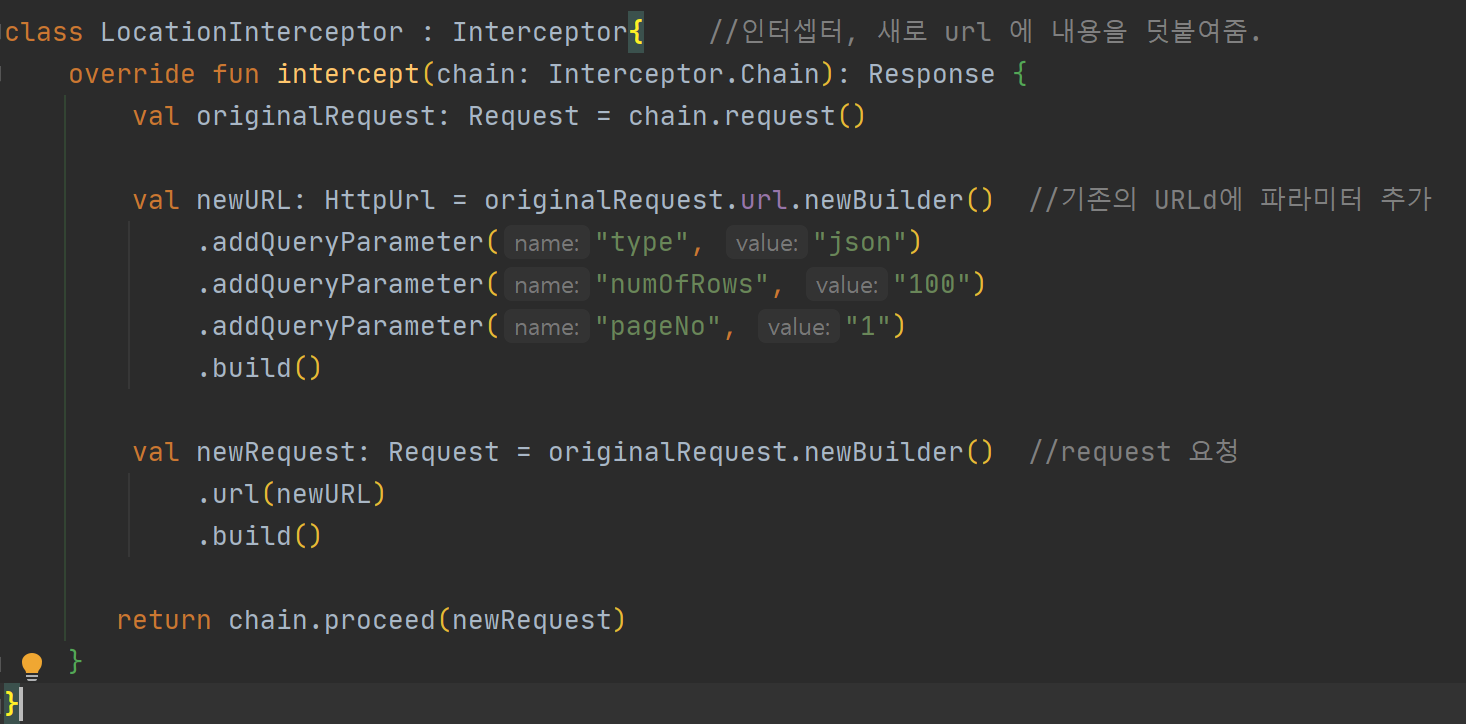


그림 : LocationInterceptor

MainActivity에서 OnCreate 바깥에, 즉 인스턴스 변수로 LocationAPI 변수를 선언한다. Interceptor 인스턴스와 Gson 컨버터 빌더 인스턴스도 생성한 뒤, Retrofit 객체를 생성한다. Gson 컨버터는 HTTP 통신의 결과로 받는 okhttp3.Response 객체를 String으로 변경해주는 역할을 한다. 이 과정에 대한 소스코드는 아래 그림 10에 나타난다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : Retrofit 객체 생성

이후 onCreate 메소드에서

locationAPI = retrofit.create(LocationAPI::class.java)

코드를 선언하여 loactionAPI 변수에 인스턴스를 할당하고, download 메소드를 실행한다. 해당 과정은 아래 그림 11과 같다. 여기서 사용하는 locCodeArr은 각 도시의 시도 코드와 구군 코드를 저장해 놓은 배열이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : LocationAPI 인스턴스 할당

download 메소드는 locationAPI를 사용해 데이터통신을 직접 실행하는 메소드이다. Download 메소드가 실행되면 입력받은 시도 코드와 구군 코드를 통해 그림 11에서 생성한 locationAPI 인스턴스의 getLocaiton(siDo, guGun)메소드를 통해 Call 객체를 생성한다. 이 Call은 retrofit2의 객체로, enqueue 메소드를 가진다. Call.enqueue를 실행하면 백그라운드 쓰레드에서 Call 요청을 처리하고, 결과값 Callback 객체를 반환한다. 이 Callback 객체는 Call 요청의 성공 여부에 따른 onFailure 메소드와 onResponse 메소드를 상속받고, 다운로드한 데이터를 처리하기 위해 onResponse에 코드를 작성하여 구현된다. 해당 메소드에서 데이터베이스에 사고다발지역 정보를 저장하는 과정을 실행한다. 데이터베이스에 저장하는 부분을 제외한 download 메소드는 아래 그림 12와 같다. 다운로드 결과는 Location 객체이며, 이는 [**3.2 Class Diagram 명세**](#_3.2_Class_Diagram)에서 설명한 바와 같이 공공데이터 포털[1]에서 제공하는 JSON의 형식을 객체로 생성한 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : download 메소드

위 그림 12의 loc 변수가 할당 받는 response.body()가 http 통신의 결과이다. 전체적인 데이터 파싱 과정을 정리한 순서는 아래와 같다.

• Retrofit.create로 생성된 API의 getLocation 메소드를 실행, 이 때 시도, 구군 코드를 매개변수로 가짐

• Retrofit 객체가 가지는 baseUrl에 API가 @GET으로 가지는 URL을 추가하고 매개변수 값을 @Query를 통해 URL에 추가

• API에서 만들어진 URL을 Interceptor 가 가로채서 newBuilder()를 통해 기존 URL에 URL을 덧붙임

• 해당 주소를 가진 Call<Location>이 enqueue 메소드를 통해 백그라운드에서 실행, 이 때 다운로드 하는 데이터는 okhttp3.Response 객체

• 해당 객체를 Converter가 String으로 변경시킴

• 일련의 과정의 결과로 다운로드한 데이터는 Location 객체, response.body()를 통해 접근함

## 4.2 데이터베이스 및 공유 프리퍼런스 관련 코드

본 어플리케이션은 사용자의 위치와 사고다발지역의 좌표를 비교하여 알림을 발생시켜준다. 매번 비교할 때 마다 인터넷에서 데이터를 다운로드 하는 것은 비효율적이기 때문에 한번 다운로드한 데이터를 기기에 저장하는 방식을 사용한다. 이 때 데이터를 영구적으로 저장하기 위해 Room을 이용한다.

Room을 이용하기 위해 어플리케이션의 gradle 파일에 room 관련 의존성을 추가해야 한다. 이 때, Kotlin의 에노테이션 동작을 위해 플러그인 또한 추가해야 한다. 아래 그림 13의 내용을 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : Room 의존성 추가

의존성을 추가한 뒤, Room을 사용하기 위해 데이터베이스의 테이블이 될 클래스를 생성해야 한다. 본 어플리케이션에서 저장해야 하는 데이터는 사고다발지역의 위도, 경도, 시도코드로 간단한 구조를 가지고 있다. 또한, Room을 사용하면 데이터베이스 테이블에 대한 높은 수준의 이해 없이 손쉽게 데이터베이스를 생성할 수 있다. 아래 그림 14와 같이 data class에 @Entity 에노테이션을 지정하고, @Primary Key를 지정하는 것 만으로 데이터베이스 테이블을 생성할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : 데이터베이스 테이블, Coordinate 클래스

이후 데이터베이스를 생성한다. 데이터베이스는 추상 클래스로 생성해야 하며, @Database 에노테이션으로 손쉽게 생성할 수 있다. 이후 추상 클래스를 구현했을 때 데이터베이스에 대한 접근을 위해 @Dao에노테이션을 가지는 데이터베이스 접근 객체(Database Access Object) CoordinateDAO인터페이스를 생성해야 한다. 본 DAO는 데이터베이스에 접근하는 쿼리를 메소드로 가진다. 그림 15, 16은 각각 데이터베이스와 DAO를 나타낸다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : CoordinateDatabase 추상 클래스

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : CoordinateDAO 인터페이스

이후 구글 개발자 문서에서 권장하는 리포지토리 패턴을 위해 Repository 클래스를 생성한다. 해당 리포지토리는 데이터베이스의 데이터에 접근하는 과정을 캡슐화 한다. 그림 17은 CoordinateRepository이다. Room.databaseBuilder의 build 메소드를 통해 데이터베이스를 생성한다. 이 때, Room은 Dao 인터페이스 내부의 함수에 대한 실행 코드를 자동으로 생성한다. Dao의 메소드를 통해 데이터베이스에 접근할 수 있도록 메소드를 작성한다. insertTestCase 메소드는 본 어플리케이션이 잘 작동하는지 확인하기 위해 데이터를 추가하는 메소드이다. 이는 사고 다발 지역이 아니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : Coordinate Repository

데이터베이스에 대한 모든 접근은 Repository 클래스가 진행하고, 다른 클래스들은 데이터베이스가 어떻게 작동하는지 관여하지 않도록 하기 위해 해당 클래스는 싱글톤으로 구성한다. 이를 위해 동반 객체를 생성한다. 동반 객체는 Java의 static과 유사하다. 동반 객체의 init 메소드를 통해 repository 인스턴스를 생성하고, 이후 모든 접근은 get 메소드를 통해 이루어지게 구현한다. 이를 위한 동반 객체는 아래 그림 18과 같이 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : 싱글톤을 위한 동반객체

최초에 어플리케이션이 실행될 때 repository의 init 메소드를 실행하기 위해 Application의 서브클래스를 생성한다. Application.onCreate 메소드는 어플리케이션이 최초로 메모리에 load 될 때 자동 실행된다. 따라서 초기화 작업을 수행하는데 용이하다. 아래 그림 19와 같이 서브클래스를 생성한 뒤, Manifest 파일에 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : Application을 상속받은 서브클래스

해당 과정들이 끝난 뒤에는 MainActivity에서 데이터베이스를 사용할 수 있다. 본 어플리케이션에서는 데이터를 파싱한 후, 해당 데이터를 저장한다. 인스턴스 변수로 CoordinateRepository.get()을 통해 repository 객체를 생성하고, onCreate에서 테스트케이스를 저장한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : MainActivity에서 데이터 삽입

## 4.3 GPS 관련기능

## 4.4 백그라운드 알림 관련기능

# 5. 부록

# 6. 색인

본 문서에서 참조한 문서, 사이트들의 목록이다.

**[1] 공공데이터 포털**: <https://www.data.go.kr/iim/api/selectAPIAcountView.do>