**-페이지 : 인트로 (introActivity)**

**기능 1 : 사용자 로그인 확인 (Firebase의 로그인 확인)**

발생 : <<Firebase 로그인 확인>>, <<로그인 이동>>, <<지도 홈 이동>>

액터 : USER, SYSTEM

목표 : Firebase에 사용자 정보가 있는 지 확인

시작조건 :

1. 앱을 실행 시

정상흐름:

1. 사용자 로그인 확인 (Firebase의 로그인 확인)

1.1. getCurrentUser()로 현재 앱에 로그인된 사용자의 Google계정

정보(currentUser)호출

1.2. Page 인트로(기능 2를 실행 완료하면, 지도 홈으로 이동)

대안흐름:

1. 사용자 로그인 확인 (Firebase의 로그인 확인)
   1. 사용자 정보가 없다면, 인트로(introactivity)으로 이동

**기능 2 : 유저 정보 생성 및 벨 조회**

발생 : <<데이터베이스 호출>>, <<객체 생성>>, <<싱글톤 생성 및 저장>>, <<지도 홈 이동>>

액터 : USER, SYSTEM

목표 : 데이터베이스를 호출하여 객체 생성과 Firebase에 리스트 저장

시작조건 :

1. 데이터베이스를 호출 시, 지도 홈(Street\_view)으로 이동

2. 데이터베이스를 호출 실패 시, 에러 로그를 기록

정상흐름:

1. Firebase 사용자 정보 생성

1.1. 사용자 정보가 null이 아니라면, Firebase Realtime Database의 루트 참조로 데이터베이스 액세스

1.1.1 child()로 "users"->유저의 uid() 파일-> "email" (데이터베이스의 경로)안에 사용자의 이메일을 저장

1.1.2 child()로 "users"->유저의 uid() 파일-> "name" (데이터베이스 키)안에 있는 사용자의 이름을 저장

1.1.3 child()로 "users"->유저의 uid() 파일-> "PhotoUrl" (데이터베이스 키) 안에 있는 사용자의 프로필 이미지를 저장

2. Firebase 안심벨 정보 호출

2.1 addOnCompleteListener()로 "safebell"(데이터베이스 키)의 데이터를 가져오는 작업이 완료될 때까지 동작을 정의

2.1.1 Firebase 인증 성공 시, Firebase성공을 디버그 로그로 출력

2.1.1.1. getResult()로 작업의 결과를 반환 받을 때까지 반복

2.1.1.2. snapshot과 getValue()로 "WGS84\_H"(데이터베이스 키)안에 있는 안심벨 경도를 문자열로 저장(WGS84\_H)

2.1.1.3. snapshot과 getValue()로 "WGS84\_W"(데이터베이스 키)안에 있는 안심벨

위도를 문자열로 저장(WGS84\_W)

2.1.1.3.1. 안심벨 경도와 위도값이 있다면, parseDouble()로 형변환 후 저장

(변수명 변경: WGS84\_H -> lo , WGS84\_W-> la)

2.1.1.3.2. List LatLng() 객체인 리스트(coordinateList)에 경도, 위도 값 추가

2.1.1.3.3. setFirebaseData()로 리스트를 FirebaseDataSingleton 인스턴스에 저장

3 Firebase 안심 cctv 정보 호출

안심 cctv 경도와 위도 값을 가져오는 과정은 2.1.1.2~2.1.1.3.3 동일 (cctv 리스트 이름: cctvList)\*

4. 마커 생성

4.1. bell 정보 저장

4.1.1 저장된 coordintaeList를 반복하여 maker를 찍을 위치에 setposition()으로 좌표 지정

4.1.2 마커의 크기를 지정하기 위해 setWidth, setHeight 80으로 지정

4.1.3 마커의 아이콘을 .setIcon으로 지정

4.1.4 bell의 마커 정보를 담고 있는 makers\_bell에 add()를 통해 값을 저장

4.2 cctv정보 저장

4.2.1 4.1 과 동일하게 수행

4.3 bell, cctv<marker>의 정보를 싱글톤에 저장

4.3.1 setFirebasemakerData(markers\_bell)을 저장

4.3.2 setFirebasemakerData(markers\_bell)을 저장

5 싱글톤 생성 및 마커, 좌표값을 싱글톤을 저장

5.1. 인스턴스가 null이라면, getInstance()로 객체 생성

5.1.1 setFirebasebellData() 통해 coordinateList 값을 싱글톤 내의 변수(coordinateList)에 저장

5.1.2 setFirebasecctvData() 통해 cctvList값을 싱글톤 내의 변수(cctvList)에 저장

5.1.3 setFirebasebellmarkerData() 통해 markers\_bell값을 싱글톤 내의 변수(markers\_bell)에 저장

5.1.4 setFirebasecctvmarkerData() 통해 markers\_cctv값을 싱글톤 내의 변수(markers\_cctv)에 저장

대안흐름:

1. Firebase 인증 실패 시, 알림창으로 정보를 못 불러왔다는 내용과 일시적 오류임을 출력

1.1 확인 버튼 클릭시, 알림창 종료

1.2 에러 로그를 기록

**- 페이지 : 로그인 (MainActivity)**

**기능: 로그인 기능**

발생 : <<Google 로그인 시도>>, <<Google 로그인 인증>>, <<Firebase 인증>>, <<지도 홈 이동>>

액터 : USER, SYSTEM

목표 : 로그인 버튼 클릭 시, 구글 계정으로 로그인을 시도

시작조건 :

1. 로그인이 되어 있다면(인트로에서 사용자 로그인 확인이 되었다면) , 로그인 기록 확인을 통해 메인

페이지로 이동

2. 로그인이 안 되어 있다면, 구글 계정으로 로그인 시도

정상흐름:

1. 사용자 인증 (Google 로그인 인증)

1.1. Google 로그인을 성공할 시, GoogleSignInAccount로 사용자의 Google계정 정보를 전달

getResult()로 실제 작업 결과를 추출

1.2 추출한 내용을 Firebase인증을 수행하기 위해 전달

2. 사용자 인증 (Firebase 인증)

2.1 GoogleAuthProvider 함수를 통해 구글의 ID 토큰에 대한 인증 자격 증명(credential)을 생성

2.2 signInWithCredential()를 호출하여 인증 자격 증명을 전달

2.2.1 Firebase에 대한 사용자 로그인을 시도

2.2.2 addOnCompleteListener()로 Firebase 인증 작업이 성공/실패 여부를 처리

2.2.2.1로그인에 성공했다면, getCurrentUser()로 사용자의 Google계정 정보를 FirebaseUs er객체로

반환

(사용자의 Google계정 정보: UserID(사용자 고유식별자), 이메일 주소, 표시 이름, 프로필 사진URL)

2.2.2.1.1 updateUI()에 사용자 정보를 전달하여 사용자가 로그인한 상태를 업데이트

2.2.2.1.2지도 홈(Street\_view)으로 이동

대안흐름 :

1. 사용자 인증 (Google 로그인 인증)

1.1 Google 로그인을 실패 시, 에러 로그를 기록

2. 사용자 인증 (Firebase 인증)

2.1 로그인에 실패했다면, Firebase 인증 실패에 대해 로그를 출력

2.1.1. 사용자에게 Firebase 인증 실패에 대해 토스트 메시지 알림

2.1.2. updateUI()에 null을 전달하여 로그인한 사용자가 없음을 업데이트

**- 페이지 : 지도 홈 (Street\_view, 메인)**

**기능 1: 네이버 지도 표시**

발생 : <<네이버 맵 api 호출>>, <<맵 불러오기>>

액터 : USER, SYSTEM

목표 : 사용자에게 네이버지도를 제공

시작조건 :

1. 지도 홈에 접속 시

정상흐름:

1. 네이버 맵 지도 객체 생성
   1. MapFragment가 null일 경우, FragmentManager의 beginTransaction()을 호출하여 Fragment 트랜잭션 시작
      1. MapFragment를 생성하여 "map\_fragment"라는 ViewGroup에 추가
   2. getMapAsync()를 호출하여 비동기적으로 지도 객체를 초기화하고, 완료 시점에 onMapReady()가 자동으로 호출됨
2. onMapReady 실행 (벨, CCTV 마커 생성)
   1. 초기화된 naverMap객체를 가져와 저장
   2. naverMap의 카메라 움직임에 대한 OnCameraChangeListener를 설정
      1. getCameraPosition().zoom을 호출하여 줌 정보를 zoom변수에 저장
      2. handleZoomChange(zoom)메서드를 호출하여 줌 상태에 따라 지도 요소를 조정
   3. CCTV 마커 목록인 markers\_cctv가 null인지 확인
      1. cutstomMarkerList를 JavaItem리스트로 생성
      2. markers\_cctv의 모든 항목을 cutstomMarkerList에 저장
   4. TedNaverClustering.with()를 호출하여 Context와 naverMap을 전달하고, 마커 클러스터링 설정을 시작
      1. costomMarker()메서드를 호출하여 확대 시 마커 모양을 지정
      2. setIcon()으로 CCTV 마커 아이콘을 설정하고 setWidth()와 setHeight()로 크기를 지정
      3. 설정된 마커를 반환하여 지도에 적용
   5. items(cutstomMarkerList)를 호출하여 cutstomMarkerList에 저장된 마커 정보를 TedNaverClustering에 전달
   6. make()를 호출하여 설정한 마커를 지도에 표시
3. handleZoomChange() 정의
   1. zoom값이 18이고 이전 줌 값인 prevZoom이 17일 경우
      1. marker\_bell가 null이 아닌지 확인
      2. for문을 통해 marker를 하나씩 반복하여 setMap(naverMap)을 호출하여 지도에 추가
   2. zoom이 17이고 prevZoom이 18일 경우
      1. for문으로 marker를 하나씩 반복하여 setMap(null)을 호출하여 지도에서 삭제
   3. 현재 줌 값인 zoom을 prevZoom에 저장하여 이후 줌 변화 감지 준비

대안흐름:

1. 지도 객체 생성 실패 시
   1. MapFragment초기화 실패나 네이버 맵 API 호출 불가 시, 사용자에게 "지도를 불러올 수 없습니다"라는 경고 메시지를 띄우고 화면을 종료함
2. 지도의 줌 수준 문제 발생 시
   1. zoom값이 -1이거나 잘못된 값일 경우, handleZoomChange()내에서 별도 작업 없이 메서드를 종료

**기능 2: 내 위치 표시 및 버튼**

발생 : <<위치 권한 확인>>, <<위치 마커 표시>>, <<카메라 이동>>, <<위치 업데이트>>, <<위치 버튼 누를 시>>, <<줌 확대>>

액터 : USER, SYSTEM

목표 :  위치 권한을 얻어 현재 위치에 대한 마커를 표시

시작조건 :

1. 내 위치 권한을 얻을 시

정상흐름:

1. checkSelfPermission()로 위치 권한을 가지고 있는 지 확인

1.1. 권한이 부여됐다면, getLocationManager()로 위치 서비스를 사용하기 위해 준비

1.1.1 getLatitude()로 현재 위치의 위도를 호출(latitude)

1.1.2 getLongitude()로 현재 위치의 경도를 호출(longitude)

1.1.3 가져온 경도 위도의 값이  서울의 동, 서, 남, 북 좌표 범위 안이라면

1.1.3.1 서울 범위(지정해둔 서울의 위도,경도) 내에 있다면,  moveCamera()로 현재 위치로 이동

요청(서울의 위도, 경도 좌표: (37.5665, 126.9780)

1.1.4 요청에 따라 카메라 이동 시, true로 값을 변경(isCameraMoved)

1.1.5 내 위치로 카메라 이동이 됐다면, getLocationOverlay()로 현재 위치 오버레이 객체를 호출

1.1.6 setVisible()로 위치 오버레이 표시

1.1.6.1 locationOverlay의 위치 설정을 위해 latitude. Longitude 값을setPosition()으로 위치 오버레이의 위치를 설정

1.1.7 checkSelfPermission()로 위치 권한을 가지고 있는 지 확인(업데이트)

1.1.8 권한이 부여됐다면, requestLocationUpdates()로 1초 또는 10미터 이동 시 위치 정보 업데이트 요청

2 내위치 버튼 기능

2.1 naverMap이 null이 아니라면,  getLocationOverlay()로 현재 위치를 나타내는 오버레이를 호출

(locationOverlay)

2.3 ‘locationOverlay’가 null이 아니라면, getPosition()로 ‘LocationOverlay’에서 현재 위치의 좌표를 호출(currentLocation)

2.3.1 ‘currentLocation’이 null이 아니라면

2.3.1.1 moveCamera()로 네이버 지도의 카메라를 현재 위치로 이동

2.3.1.2 zoomTo(30)로 현재 카메라의 줌 레벨을 30으로 설정

대안흐름:

1. 권한이 없다면, requestPermissions()로 사용자에게 위치 권한을 요청

(업데이트 과정도 동일하게 수행)

2. 카메라 이동이 안 됐다면, 서울 범위 내에 있는 지 확인

3. 지도가 띄워지지 않았다면, “지도가 아직 준비되지 않았습니다”라는 토스트 메시지를 표시

4. 위치 오버레이가 null이라면, “위치 정보를 사용할 수 없습니다”라고 토스트 메시지를 표시

5. 현재 위치가 null이라면, “현재 위치를 찾을 수 없습니다”라고 토스트 메시지를 표시

6. addOnFailureListener로 Task가 실패했을 때, 실패를 로그에 기록

6.1 Try블록으로 startResolutionForResult()로 사용자가 문제를 해결할 수 있도록 대화 상자를 표시

6.1.1 getLocation() 메서드인 1.1~1.6으로 이동

6.2. 예외가 발생 시, 로그 메시지를 기록

6.2.1 onActivityResult로 어떤 요청이 완료되었는지를 식별

**기능 3: 검색어 주소 변환 후 카메라 이동**

발생 : <<검색어 주소 변환>>, <<서버 요청>>, <<카메라 이동>>, <<마커 표시>>, <<줌 확대>>

액터 : USER, SYSTEM

목표 : 검색어 입력 시, 검색 단어에 대한 위치를 표시

시작조건 :

1. 검색어를 입력 시

정상흐름:

1. 검색어 주소 변환(검색어를 도로명으로 반환)

(

1.1 검색어 길이가 0이 아니면, 검색어 문자를 저장(address)

1.2 try블록 실행 시, 네이버 로컬 검색 API 호출(네이버에 지정된 코드)

1.3 getResponseCode()로 HTTP요청을 보낸 후 서버로부터 응답 코드를 저장(responseCode)

1.3.1 응답 코드가 200이라면, getInputStream()로 서버의 응답 데이터를 읽기 위한 입력

스트림을 호출(inputStream)

1.3.1.1 한 줄씩 읽어와서 append()로 내용을 저장(stringBuilder)

1.3.1.2.1 stringBuilder를 JSON형식으로 변환(responseJson)

1.3.1.2.2 저장된 “items”라는 키에 해당하는 값을 추출(items)

1.3.1.2.3 items에 결과가 있다면 첫번째 요소를 저장(firstItem)

1.3.1.2.3.1 getString()로 "roadAddress"라는 키에 해당 값을 저장(도로명 주소)

2. 좌표값으로 반환(도로명 주소를 가지고 좌표 값으로 반환)

2.1 1.4.2.~1.4.3.와 동일한 코드 (단, "addresses"라는 키에 해당 값을 저장(주소))

2.1.1 getDouble()로  "x" 키에 해당하는 값을 저장(latitude (위도))

2.1.2 getDouble()로  "y" 키에 해당하는 값을 저장(longitude (경도))

2.1.3 LatLng()로 latitude(위도), longitude (경도) 값을 객체 생성 후 반환

3. 검색어 좌표로 카메라 이동

3.1 검색어의 좌표 값이(latLng)가 null이 아니면,

3.1.1 moveCamera()로 검색어의 좌표로 카메라 이동

(해당 위치에 마커를 표시 (전의 마커 표시 코드와 동일)후 20레벨로 줌 확대)

대안흐름:

1. 검색어 주소 변환

1.1 응답 코드가 200이 아니면

1.1.1 Builder()로 "검색 결과가 없습니다.", "일시적인 오류입니다. 잠시 후에 다시 시도해주세요."

 라고 대화 상자 표시

2. 카메라 이동

2.1 Naver 지도 객체가 null이면, "NaverMap is null"로 로그를 출력

**기능 4: 친구위치 표시 기능**

발생 : <<위치 권한 확인>>, <<위치 마커 표시>>, <<위치 업데이트>>

액터 : USER, SYSTEM

목표 : 친구의 위치를 사용자의 화면에 표시

시작조건 :

1. 지도 홈에 접속 시

정상흐름:

1. Firestore 인스턴스 생성
   1. FirebaseFirestore db = FirebaseFirestore.getInstance()를 통해 Firestore 인스턴스를 생성하여 데이터베이스에 접근 준비
2. 현재 사용자 이메일 가져오기
   1. FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser()를 호출하여 현재 로그인한 사용자의 이메일를 가져옴
      1. currentUser != null조건을 통해 사용자의 정보(이메일, 이름)로 로그인된 상태인지 확인
      2. 사용자가 로그인되어 있으면 String userEmail = currentUser.getEmail()로 현재 사용자의 이메일을 저장
3. 친구 목록 조회 요청
   1. db.collection("users").document(userEmail).collection("친구목록").get()을 호출
      1. Firestore에서 현재 사용자의 친구 목록을 비동기로 요청
      2. Firestore에서 요청이 성공하면 task.getResult()를 통해 친구 목록 컬렉션의 각 친구 문서로 접근
4. 친구 목록 반복 처리
   1. for (QueryDocumentSnapshot friendDoc : task.getResult())구문을 사용하여 친구문서 접근 및 이메일 정보 저장을 반복 처리
      1. 각 friendDoc문서에서 friendDoc.getString("email")을 통해 친구의 이메일을 가져와 friendEmail변수에 저장
5. 친구 위치 정보 요청
   1. db.collection("users").document(friendEmail).collection("친구위치").document("location").get()을 호출하여 각 친구의 위치 정보(위도와 경도 값)를 Firestore에서 비동기로 요청
      1. Firestore에서 요청이 성공하면 locationTask.getResult()를 통해 친구의 위치 정보 문서로 접근
6. 위치 정보 확인 및 마커 생성
   1. if (locationDoc.exists())로 위치 정보 문서가 존재하는지 확인
      1. 존재할 경우 Double latitude = locationDoc.getDouble("latitude");와 Double longitude = locationDoc.getDouble("longitude")로 위도와 경도 값을 가져옴
      2. 위도와 경도가 모두 존재할 경우, LatLng friendLocation = new LatLng(latitude, longitude)로 친구의 위치를 생성
7. 지도에 마커 추가
   1. Marker friendMarker = new Marker()를 호출하여 친구 위치에 표시할 마커를 생성
      1. friendMarker.setPosition(friendLocation)로 마커 위치를 설정
         1. friendMarker.setIcon(OverlayImage.fromResource(R.drawable.marker))로 친구의 위치 마커 아이콘을 지정
            1. 마커의 크기는 friendMarker.setWidth(50)과 friendMarker.setHeight(50)으로 조정한 후, friendMarker.setMap(naverMap)으로 NaverMap에 마커를 추가

대안흐름:

1. Firestore에서 친구 목록 조회 요청 실패
   1. 요청이 실패할 경우, task.isSuccessful()이 false를 반환.
   2. Log.e("Firestore", "친구 목록 조회 실패", task.getException())를 사용하여 로그에 오류 메시지를 기록하고 사용자에게 "친구 목록을 불러오는 데 실패했습니다."라는 메시지를 표시.
2. Firestore에서 친구 위치 정보를 가져왔으나 위치 정보 문서가 존재하지 않는 경우
   1. if (!locationDoc.exists())로 위치 정보 문서가 존재하지 않는 경우를 확인.
      1. 위치 정보 문서가 존재하지 않을 경우, Log.d("Firestore", "친구 위치 정보 문서가 존재하지 않습니다.")로 로그에 기록.
      2. 사용자에게 "친구의 위치 정보가 존재하지 않습니다."라는 메시지를 표시하여 알림.

**- 페이지 : 햄버거 페이지 (Header)**

**기능 1: 나의 프로필**

발생 : <<사용자 정보 호출>>, <<데이터베이스 호출>>, <<UI요소 업데이트>>

액터 : USER, SYSTEM

목표 : 사용자 정보를 호출하여 UI요소에 업데이트를 수행

시작조건 :

1. 지도 홈을 실행 시

정상흐름:

1. 사용자 정보(currentUser)가 null이 아니라면
   1. getEmail()로 ‘사용자의 이메일’을 호출하여 setText()로 이메일 값을 넘겨줌
   2. ‘사용자의 이름’ 또한 마찬가지 1.1 과 동작 같음

대안흐름:

1. task로 비동기 작업이 실패할 시, 로그로 실패에 대한 오류 메시지를 기록

**기능 2: 친구검색**

발생 : <<친구 검색>>, <<친구 검색 버튼>>, <<프로필 리사이클러뷰 숨김>>, <<동일한 정보 목록 표시>>

액터 : USER, SYSTEM

목표 : 친구 검색창에 이름이나 이메일을 검색 시, 동일한 정보의 친구 목록을 표시

시작조건 :

1. 친구 검색창에 이름이나 이메일을 검색 시

정상흐름:

1. profile\_info\_adapter()로 profile\_info\_adapter 클래스의 생성자를 호출하여 새 객체를 생성
2. Firebase 데이터베이스 호출
   1. getReference()로 Firebase Realtime Database의 루트 참조
   2. child()와 get() “users”안에 있는 ‘사용자 정보 데이터’를 호출 (이메일, 이름, 프로필 사진URL)
   3. addOnSuccessListener()로 ‘사용자 정보 데이터’가 호출됐을 때 실행할 작업을 정의
      1. task로 비동기 작업이 성공 시, 데이터를 가져와 반복문으로 처리
         1. 데이터 스냅샷에서 ‘이메일 값’을 호출(email) 하여 ‘자신의 이메일 값’과 equals를 통해 비교
            1. 1.3.1.1 과 동일하게 수행, ‘이름’&’프로필 사진URL’을 호출(name, photoUrl)

addItem()로 ‘이름’, ‘이메일’, ‘프로필 사진URL’을 추가

notifyDataSetChanged()로 어댑터에 데이터가 변경되었음을 알려 UI요소를

업데이트

3,. setOnQueryTextListener()로 검색창의 텍스트 입력 감지하고 그에 따른 동작 정의

3.1 ‘onQueryTextChange’ 메서드는 텍스트가 변경될 때마다 호출

3.2 setVisibility()로 사용자 리스트를( 리사이클러뷰)화면에 표시

3.3 For문을 통해 ‘프로필 정보’를 ‘item’에 담아 반복

3.3.1 if문을 통해서 ‘item’의 ‘name’(이름)을 소문자로 변환 하여 contains()로 검색어(newText) 공백 제거, 소문자 변환 후 비교 일치하거나 || 동일한 방식으로 item’의 ‘email’(이메일)과 newText를 비교함 일치한다면

3.3.3.1 add()로 ‘item’을 추가(filteredList)

3.4 setItem()로 filteredList의 데이터를 어댑터에 설정

3.5 notifyDataSetChanged()로 어댑터의 데이터가 변경됐음을 알려줌

대안흐름:

**기능 3: 친구 요청 보내기 어댑터 (클래스 명: profile\_info\_adapter)**

발생 : <<리스트 호출>>, <<ViewHolder 생성 후 반환>>, <<데이터 결합>>, <<리스트 추가>>, <<데이터 호출>>, <<리스트 반환>>, <<리스트 설정>>, <<대화 상자>>, <<HashMap 생성>>

액터 : USER, SYSTEM

목표 : 친구 검색 시, 친구 검색과 요청할 수 있도록 돕는 어댑터

시작조건 :

1. 친구 검색 시
2. **기능 2**: 친구검색 기능에서 호출됨.

정상흐름:

1. 생성자 정의 (ViewHolder 클래스)
   1. ‘친구 요청’ 버튼을 누를 시, setOnClickListener()로 ‘showAlertDialog’ 메서드인 5.로 이동
2. 친구 요청 (showAlertDialog 메서드)
   1. Builder()로 AlertDialog를 만들기 위한 Builder 인스턴스를 생성 (builder)

(친구 요청할 때 알림이 띄워지는 부분)

* 1. setTitle()로 대화 상자의 제목을 “친구요청”으로 설정
     1. setMessage()로 “보내시겠습니까?”로 표시할 메시지 설정
     2. 친구요청 "확인" 버튼을 추가하고, 클릭 이벤트를 설정
        1. getAdapterPosition()로 RecyclerView 아이템 위치를 호출(position)
        2. 아이템의 위치(position) 가 유효하면
           1. collection()와 document()로 “user” -> “친구목록” -> position 해당 위치의 이메일을 반환하여 접근(requestCollectionRef)
           2. 3.2.2.1 이랑 동일하게 수행하여 친구 요청 목록에 접근함(washingtonRef)

requestCollectionRef로 지정된 문서에 해당 문서가 존재할 경우,

setMessage()로 친구 요청이 되었다는 메시지 알림 노출

해당 문서가 존재하지 않은 경우, Firestore에 set()을 통해서 새로운 문서를 추가하는 작업 수행

put()을 통해 ‘이메일주소’(request\_email) 키와 ‘사용자 이메일 값’을 HashMap에 저장

‘사용자의 이름’, ‘사용자의 프로필 사진’ 동일하게 코드 수행

( 'request\_name' 키와 사용자 이름 값, 'request\_photoUrl' 키와 프로필 사진 URL 값)

3.2.2.2.2.2.2.1 성공적으로 수행한 경우, 해당 위치의 사용자에게 친구 요청을

보냈습니다."으로 메시지 알림 노출

* + 1. setNegativeButton()로 알림 창의 "취소" 버튼을 추가하고, 클릭 이벤트를 설정

대안흐름:

1. 친구 요청을 보낼 지 확인 (showAlertDialog 메서드)
   1. addOnFailureListener로 작업이 실패 시, 아무 작업도 수행하지 않음
   2. 아이템의 위치(position) 가 유효하지 않으면, 아무 작업도 수행하지 않음

**기능 4**: 친구 요청 수락하기 어댑터 (클래스 명: request\_friend\_adapter)

발생 : <<리스트 호출>>, <<ViewHolder 생성 후 반환>>, <<데이터 결합>>, <<리스트 추가>>, <<데이터 호출>>, <<리스트 반환>>, <<리스트 설정>>, <<대화 상자>>, <<HashMap 생성>>, <<친구 수락 확인>>, <<친구 수락 취소>>

액터 : USER, SYSTEM

목표 : 친구 요청 수락 시, 수락에 대한 수행

시작조건 :

1. 친구 요청 버튼을 누를 시

2. 친구 요청 수락 시

정상흐름:

1. 생성자 정의 (ViewHolder 클래스)
   1. setOnClickListener()로 사용자가 ‘친구 수락 버튼’을 누를 시(check), ‘showAlertDialog’ 메서드인 5.로 이동
   2. OnClickListener()로 사용자가 ‘친구 수락 취소 버튼’을 누를 시( close)
      1. Builder()로 AlertDialog를 만들기 위한 Builder 인스턴스를 생성 (builder)
      2. “친구 요청” 제목의 친구를 “삭제하시겠습니까?” 알림을 노출
      3. 친구 수락 "확인" 버튼을 추가하고, 클릭 이벤트를 설정
         1. getAdapterPosition()로 RecyclerView의 아이템 위치를 호출(position)
            1. collection(), document()를 통해 ‘user’ -> ‘사용자의 이메일’ -> “친구 요청목록”

해당 요청의 ‘이메일’을 기반한 문서에 접근

1.2.3.1.2 delete()로 해당 문서를 삭제

* + - 1. show()로 AlertDialog를 화면에 표시

1. 친구 요청을 수락 확인 (showAlertDialog 메서드)
   1. Builder()로 AlertDialog를 만들기 위한 Builder 인스턴스를 생성 (builder)
   2. “친구요청” 제목의 “친구 수락하시겠습니까?”로 표시할 메시지 설정
   3. 친구 수락 "확인" 버튼을 추가하고, 클릭 이벤트를 설정
      1. getAdapterPosition()로 연결된 RecyclerView의 아이템 위치를 호출(position)
         1. 아이템의 위치(position)가 유효하면
            1. collection()와 document()로 Firestore의 ‘user’ -> ‘사용자의 이메일’ -> ‘친구목록’ -> 해당위치의 이메일 기반한 문서에 접근 (washingtonRef)
            2. ’(washingtonRef)에 set()을 통해서 새로운 문서를 추가하는 작업 수행

put()로 “요청 이메일”(email) 키와 “요청 이메일” 과 “요청 프로필 사진”을

HashMap에 저장

( 이름(name) 키와 사용자 이름 값, 프로필 사진 URL (photoUrl) 키와 프로필 사진 URL 값)

3.3.1.1.2.1.1 addOnSuccessListener로 성공적으로 수행한 경우

3.3.1.1.2.1.2 (대상 계정)님과 친구가 되었습니다." 로 메시지와 요청 수락 "확인" 버튼

이벤트 설정

3.3.1.1.2.1.3 collection()와 document()로 Firestore의 ‘users’ -> ‘사용자의 이메일’ -> “친구요청목록” -> 클릭된 아이템의 이메일을 기반한 문서에 접근

delete()로 해당 문서를 삭제

* + - * 1. collection()로 Firestore의 ‘사용자 정보’(users) -> ‘현재 사용자의 이메일’ -> “친구목록”에 -> 해당 아이템의 이메일을 기반한 문서에 접근(setfriend)
        2. (setfriend) 에 set()을 통해서 새로운 문서를 추가하는 작업 수행

put()로 이메일(email) 키와 사용자 이메일 값과 프로필 사진을

HashMap에 저장

( 이름(name) 키와 사용자 이름 값, 프로필 사진 URL (photoUrl) 키와 프로필 사진 URL 값)

* + - * 1. addOnSuccessListener로 성공적으로 수행했을 때 호출
  1. setNegativeButton()로 알림 창의 "취소" 버튼을 추가하고, 클릭 이벤트를 설정(onClick)
     1. show()로 AlertDialog를 화면에 표시

대안흐름:

1. addOnFailureListener로 삭제가 실패했을 때 호출
   1. 삭제 실패 시 수행할 작업을 정의(onFailure)
2. 친구 요청을 수락 확인 (showAlertDialog 메서드)
   1. addOnFailureListener로 삭제가 실패했을 때 호출
   2. addOnSuccessListener로 성공적으로 수행실패 시 호출

**기능 5**: 친구요청목록

발생 : <<친구요청목록 버튼>>, <<데이터베이스 호출>>, <<리스트 추가>>, <<리스트 삭제>>, <<어댑터 변경 알림>>, <<친구요청목록 화면에 표시>>

액터 : USER, SYSTEM

목표 :

1. 친구요청목록 버튼을 누를 시, 정보를 가져와 화면에 표시

2. 문서들이 추가, 제거, 수정에 따라 문서를 처리하기 위함

시작조건 :

1. 친구요청목록 버튼을 누를 시

정상흐름:

1. Firestore 데이터베이스 호출
   1. collection()와 document()로 Firestore 데이터베이스의 ‘user’ -> ‘사용자의 이메일 주소’ -> “친구요청목록” 을 참조(requestCollectionRef)
   2. addSnapshotListener()로 친구요청목록에 대한 실시간 리스너를 추가한다.
      1. 에러가 null이면, 문서들의 변경사항들을 반복한다.(dc)
         1. getType()로 문서의 변경 사항에 따라 조치
            1. 문서가 추가됐거나 수정됐을 시

getString()로 이메일(request\_email), 이름(request\_name), 프로필 사진 URL(request\_photoUrl)인 문자열 값을 호출

(사용자 정보: requestEmail, requestName, requestPhotoUrl)

‘ re\_addItem’ 메서드로 사용자 정보를 받아 저장(해당 아이템을 request\_list에 추가)

notifyDataSetChanged()로 친구요청목록의 어댑터가 변경됐음을 알림 (request\_friend\_adapter)

친구 요청 목록이 null이 아니면, setVisibility()로 화면에 “느낌표”(친구 목록)를 표시

* + - * 1. 문서가 제거됐을 시

getString()로 이메일 주소(request\_email)인 문자열 값을 호출

“친구요청목록” 리스트의 모든 항목을 반복(request\_list)

삭제될 이메일 주소’의 이메일이 일치하면

remove()로 친구요청목록 (request\_list)에서 목록 제거

1.2.1.1.2.2.1.2 notifyDataSetChanged()로 친구요청목록의 어댑터가 변경됐음을 알림

대안흐름:

1. 에러가 null이 아니면

1.2 로그로 "listen:error"라는 경고 메시지 출력 후 메서드를 종료

**기능 6**: 친구 목록 어댑터 (클래스 명: friend\_list\_adapter)

발생 : <<리스트,Context 호출>>, <<ViewHolder 생성 후 반환>>, <<데이터 결합>>, <<리스트 추가>>, <<데이터 호출>>, <<리스트 반환>>, <<리스트 설정>>, <<위치 반환>>, <<대화 상자>>, <<HashMap 생성>>

액터 : USER, SYSTEM

목표 : 친구 목록을 불러올 시, 친구 목록과 삭제를 돕는 어댑터

시작조건 :

1. 친구 목록 리스트를 얻어왔을 시
2. **기능 7**: 친구목록에 의해서 호출될 때

정상흐름:

)

1. 생성자 정의 (ViewHolder 클래스)
   1. 클릭된 아이템의 이메일(email) 필드가 ‘현재 사용자의 이메일’과 다를 경우, ‘showAlertDialog’ 메서드인 2.으로 이동
   2. setOnClickListener()로 사용자가 ‘친구 프로필 레이아웃’(linearLayout)을 누를 시
      1. getAdapterPosition()로 연결된 RecyclerView의 아이템 위치를 호출(position)
      2. ‘position’의 위치가 유효하면
         1. Intent()로 친구 채팅 방(friend\_list\_chat\_r)으로 이동할 intent 설정
         2. putExtra()로 Intent에 추가 데이터를 전달 (이메일, 프로필, 이름)
         3. stratactivity로 intent실행
2. 친구 목록에서 친구를 삭제 (showAlertDialog 메서드)
   1. Builder()로 AlertDialog를 만들기 위한 Builder 인스턴스를 생성 (builder)
      1. “친구 삭제”가 제목인 친구 삭제 하시겠습니까?” 메시지 노출
      2. 알림 창에 "확인" 버튼을 추가하고, 클릭 이벤트를 설정
         1. getAdapterPosition()로 연결된 RecyclerView의 아이템 위치를 호출(position)
         2. ‘position’의 위치가 유효하면
            1. collection()과 document()로 Firestore의 “users” -> “사용자의 이메일”

“친구목록” -> “친구이메일” 을 기반한 문서에 접근

get()로 해당 위치의 친구 이메일을 반환

delete()로 해당 문서를 Firestore에서 삭제

* + - 1. show()로 알림 창 실행

대안흐름:

1. addOnFailureListener로 삭제가 실패했을 때 호출
   1. 삭제 실패 시 수행할 작업을 정의(onFailure)

**기능 7**: 친구목록

발생 : <<친구목록 버튼>>, <<데이터베이스 호출>>, <<리스트 추가>>, <<리스트 삭제>>, <<어댑터 변경 알림>>, <<친구목록 화면에 표시>>

액터 : USER, SYSTEM

목표 : 친구목록 버튼을 누를 시, 정보를 가져와 화면에 표시

시작조건 :

1. 친구목록 버튼을 누를 시

정상흐름:

1. 기능 3의 친구요청목록 코드와 동일하게 수행

(사용자 정보: Email, Name, PhotoUrl,친구목록 어댑터: friend\_list\_adapter,친구목록 리스트: friend\_list)

대안흐름:

1. 기능 3의 “친구요청목록” 코드와 동일하게 수행

**기능 8:** 로그아웃

발생 : <<로그인 버튼 누를 시>>, <<로그인으로 이동>>

액터 : USER, SYSTEM

목표 : 로그아웃을 누를 시, 로그인(MainActivity)으로 이동

시작조건 :

1. 로그아웃 버튼을 누를 시

정상흐름:

1. 사용자 정보(currentUser)가 null이 아니라면, signOut()로 현재 사용자를 로그아웃
   1. 로그인(MainActivity)으로 이동

대안흐름:

**- 페이지 : 길찾기 (findStreet) -**

**기능 1: 네이버 지도 표시**

발생 : <<네이버 맵 api 호출>>, <<맵 불러오기>>

액터 : USER, SYSTEM

목표 : 맵을 사용자에게 제공

시작조건 :

1. 길찾기에 접속 시

정상흐름:

1. 네이버 맵 지도 객체 생성

1.1. MapFragment가 null이라면, FragmentManager로 Fragment 트랜잭션 시작

1.1.1 새로운 MapFragment를 "map\_fragment"라는 ViewGroup에 추가

1.2 getMapAsync()로 map\_fragment에 대해 비동기적으로 지도를 설정

(AndroidManifest.xml에 저장된 네이버 지도API 코드를 불러오는 과정)

2. onMapReady 실행

2.1 맵의 정보를 받아온다.

대안흐름:

1. 지도 홈의 기능 1과 동일하게 수행

**기능2 : 출발지, 목적지 마커, 경로 표시**

발생 : <<출발지 주소 변환>>, <<도착지 주소 변환>>,<<출발지 주소 입력>>,<<도착지 주소 입력>>,

<<주소 교환>>, <<마커 제거>>, <<마커 추가>>, <<경로 표시>>

액터 : USER, SYSTEM

목표 : 출발지, 도착지 입력 후 실행 시, 주소 변환 후 경로와 마커 표시

시작조건 :

1. 출발지, 도착지를 입력 시

정상흐름:

1. 사용자 입력에 따라 검색쿼리를 처리

1.1 출발지 입력 후 제출 버튼 누를 시(fir\_searchView) 이벤트 처리=

1.1.1 ‘currentIndex’를 0으로 초기화 (출발지와 목적지를 관리하기 위한 변수)

1.1.2 execute()로 입력된 주소(query)를 좌표로 변환하는 메서드로 매개변수로 전달

1.2 도착지 입력 후 제출 버튼 누를 시(sec\_searchView)

1.2.1 1.1.1~1.1.3. 과 동일한 코드를 수행(이때는 ‘currentIndex’를 1으로 초기화)

(해당 부분은 query를 좌표로 가져오는 것은 지도홈의 기능 3이랑 일치함 좌표를 성공적으로 가져왔을 때 실행되는 메서드가 다름 밑의 정의)

2 지도에서 마커 제거( clearMap 메서드)

2.1 출발지와 목적지를 관리하는 배열의 인덱스 값이 NULL이 아니라면

2.1.1 해당 인덱스의 마커 정보를 가져온다

(출발지 0, 도착지1의 형태로 저장되어 있는 배열)

2.1.2 마커의 정보가 NULL이 아니라면

2.1.2.1. setMap()로 마커를 지도에서 제거

2.1.1.2 remove()로 마커 리스트에서 해당 인덱스의 마커를 제거

2.1.1.3 검색 배열 해당 인덱스의 값을 초기화

2.2 경로 리스트의 모든 요소를 반복(pathOverlays)

2.2.1 setMap()로 ‘pathOverlays’를 지도에서 제거

2.3 clear()로 ‘pathOverlays’에 저장된 모든 요소를 제거

2.4 clear()로 ‘coords’에 저장된 모든 요소를 제거(리스트에 저장된 좌표: coords)

3 두 지점 간의 경로를 네이버 지도 API를 통해 조회 후, 결과를 지도에 표시하는 작업을 수행(check 메서드)

3.1 (‘searchResults’)출발지와 목적지를 관리하는 배열에 저장된 좌표 값이 null이 아니면

3.1.1 출발지 좌표(‘searchResults[0]’)로 첫 번째 요소를 호출(startlatLng)

3.1.2 도착지 좌표(‘searchResults[1]’)로 두 번째 요소를 호출(goallatLng)

3.1.3 출발지 좌표(‘startlatLng’), 도착지 좌표(‘goallatLng’) 객체의 값을 로그에 출력

3.2 네이버 지도 API를 사용하여 경로 탐색 요청을 위한 출발지, 목적지를 추가한 URL을

생성(urlStr)

3.3 백그라운드에서 HTTP GET 요청을 보내고 응답을 받는 역할(doInBackground 메서드)

3.3.1 try블록 실행 시, 네이버 로컬 검색 API 호출(네이버에 지정된 코드)

3.3.1.1 ‘지도 홈’에서 기능 3의 1.3 ~1.3.1.2와 동일하게 수행

3.4 네이버 지도 API로부터 받아온 경로 정보를 처리 (onPostExecute 메서드)

3.4.1 받아온 경로 정보가 null이 아닌 경우, try블록으로 지도에 표시할 수 있는 형태로 변환

3.4.1.1 JSONObject()로 받아온 경로 정보를 JSON형식으로 가져오기

"route" >"trafast" > "path" 키에 해당하는 JSONArray을 호출(pathArray)

3.4.1.2 ‘pathArray’에서 각 경로 점의 좌표를 반복하여 가져옴

3.4.1.2.1 getDouble()로 첫 번째, 두번째 값을 호출(latitude(위도), longitude(경도))

3.4.1.2.2 add()로 생성된 ‘LatLng’를 ‘coords’ (경로 좌표) 리스트를 생성

3.4.1.10 PathOverlay()로 경로를 나타내는 객체를 생성(pathOverlay)

3.4.1.11 coords 경로 좌표를 ‘PathOverlay’에 좌표를 설정

3.4.1.12 setColor(), setWidth()로 경로의 색상, 두께를 설정

3.4.1.13 setMap()로 해당 경로를 지도에 표시

3.4.1.14 add()로 ‘pathOverlays’리스트에 ‘pathOverlay’객체를 추가

3.4.1.15 ‘coords’ 리스트를 비워서 이전에 저장된 좌표들을 모두 제거

3.4.1.16 moveCamera()로 경로를 확인할 수 있게 이동시킴

5 지정된 좌표로 카메라를 이동시키고, 해당 위치에 마커를 추가 (moveCameraToCoordinates 메서드)

5.1 ‘naverMap’이 null이 아니라면

5.1.1 scrollTo()로 지정된 좌표(latitude, longitude)로 카메라 이동

5.1.2 moveCamera()로 카메라 위치를 업데이트

5.1.3 setPosition()로 마커 위치를 현재 좌표로 설정

5.1.4 ‘currentIndex’가 0이라, setIcon()로 아이콘을 ‘R.drawable.start’ 이미지로 설정

5.1.5 ‘currentIndex’가 1이라, setIcon()로 아이콘을 ‘R.drawable.goal’ 이미지로 설정

5.1.6 setWidth()와 setHeight()로 마커의 너비, 높이를 150으로 설정

5.1.7 setMap()로 마커를 지도에 추가(naverMap)

5.1.8 add()로 지정된 위치에 새로운 마커를 추가(markers 리스트)

대안흐름:

1 응답 코드가 200이 아니면, 응답 코드를 로그에 기록

1.1 에러가 발생할 시, "못 불러오는데?"라는 메시지와 해당 예외를 로그에 기록

1.2 스택 트레이스를 출력( 예외가 발생한 원인과 위치를 추적하는 데 사용)

1.2.1 예외 처리 후 null을 반환 (호출자에게 실패를 알림)

2 받아온 경로 정보가 null인 경우

2.1 "Failed to fetch directions"라는 메시지를 에러 로그에 출력

2.2 위치 권한이 없으면, requestPermissions()로 권한 요청

3 ‘naverMap’이 null이면, "NaverMap is null"이라고 오류 로그를 출력

**기능 3: 목적지, 출발지 변경**

발생 : <<안내 버튼>>, <<카메라 이동 설정>>, <<줌 레벨 설정>>, <<카메라 이동>>, <<줌 레벨 이동>>

액터 : USER, SYSTEM

목표 : 사용자가 주소(출발지, 목적지 주소)를 교환하는 버튼을 누를 시, 출발지와 도착지의 주소를 서로 바꿈

시작조건 :

1 사용자가 주소를 교환하는 버튼을 누를 시

정상흐름:

1 검색어 교환 버튼

1.2 setOnClickListener()로 사용자가 검색어 교환 버튼(‘imageView4’)을 클릭할 때, 수행

1.2.1 첫 번째 검색창 (fir\_searchView)의 현재 검색어를 문자열로 호출(tempfQuery)

1.2.2 두번째 검색창 (sec\_searchView)도 1.2.1과 동일하게 수행(tempsQuery)

1.2.3 ‘tempfQuery’와 ‘tempsQuery’가 null이 아닌 경우(즉, 두 검색 뷰 모두 비어있지 않은

경우)

1.2.3.1 첫 번째 검색창(fir\_searchView)의 검색어를 ‘tempsQuery’로 설정

1.2.3.2 두번째 검색창(sec\_searchView)의 검색어를 ‘tempfQuery’로 설정

1.2.3.3 toggleMarkerIcons()를호출(두 마커를 교환하는 작업)

1.2.3.3.1 ‘markers’ 리스트에 마커가 두 개 이상인지 확인

1.2.3.3.2 마커가 두 개 이상일 경우, 수행

1.2.3.3.2.1 ‘markers’ 리스트에서 첫 번째 마커를 호출( marker0)

1.2.3.3.2.2 ‘markers’ 리스트에서 두 번째 마커를 호출( marker1)

1.2.3.3.2.3 ‘marker0’와 ‘marker1’의 현재 아이콘을 호출(currentIcon0, currentIcon0)

1.2.3.3.2.4 ‘marker0’ 아이콘이 시작 아이콘인 경우(R.drawable.start)

1.2.3.3.2.4.1 ‘marker0’ 아이콘을 목표 아이콘으로 변경(R.drawable.goal)

1.2.3.3.2.4.2 ‘marker1’ 아이콘을 시작 아이콘으로 변경(R.drawable.start)

대안흐름 :

1 마커가 두 개 미만일 경우, 에러 로그를 출력

2 ‘marker0’ 아이콘이 시작 아이콘이 아닌 경우

2.1 ‘marker0’ 아이콘을 시작 아이콘으로 변경(R.drawable.start)

2.2 ‘marker1’ 아이콘을 목표 아이콘으로 변경(R.drawable.goal)

**기능 4: 안내버튼 (내 주변 위치)**

발생 : <<안내 버튼>>, <<카메라 이동 설정>>, <<줌 레벨 설정>>, <<카메라 이동>>, <<줌 레벨 이동>>

액터 : USER, SYSTEM

목표 : 안전 귀가 및 최단 경로 안내

시작조건 :

1. 시작지와 도착지를 입력 후 안내 버튼을 누를 시에 동작함.

정상흐름:

1 . 길안내 이미지 버튼을 ‘btn’에 저장

1.1 setOnclickListener()로 길안내 이미지 버튼(‘btn’)이 클릭 되었는지 확인

1.1.1 출발지와 도착지 좌표가 null이 아니면 (0:출발지,1:목적지)

1.1.1.1 ‘checklist’를 true로 설정

1.1.1.2 ‘getLocation()’메서드를 호출 기능 3의 4.4.1.17.1.1로 이동

2. 내 위치 정보 가져오기(getLocation()’)

2.1 page. 지도 홈 기능 3과 동일하게 수행

3 사용자의 현재 위치를 기반으로 마커를 업데이트(updateMarkers)

3.1 setMap(null)로 현재 지도에 표시된 마커들을 제거(‘markers\_bell’ 리스트)

3.2 clear()로 ‘markers\_bell’ 리스트를 비움

3.3 ‘coordinateList’에 있는 각 LatLng 객체를 반복 (안심벨 마커)

3.3.1 distanceTo()로 현재 위치(currentLocation)와 현재 좌표(coordinate) 간의 거리를 계산

3.3.2 현재 좌표가 업데이트 반경(UPDATE\_RADIUS) 내에 있다면

3.3.2.1 setPosition()로 마커 위치를 현재 좌표로 설정

3.3.2.2 setIcon()로 마커 아이콘을 벨 이미지로 설정

3.3.2.3 setMap()로 마커를 지도에 추가(naverMap)

3.3.2.4 setWidth()와 setHeight()로 마커의 너비, 높이를 80으로 설정

3.3.2.5 add()로 생성된 마커를 ‘markers\_bell’ 리스트에 추가

3.4 안심 CCTV마커도 5.4. ~5.4.2.6. 과 동일한 코드를 수행(‘markers\_cctv’에 리스트 추가)

3.5 현재 사용자의 위치를 ‘previousLocation’에 저장

(현재 위치를 이전 위치로 업데이트하여, 이후의 위치 변화 계산을 위해 저장하는 역할)

대안흐름:

1 출발지와 도착지 좌표가 null이면, “경로를 설정해주세요”메시지를 토스트로 표시

2 ‘naverMap’이 null이면, “지도가 아직 준비되지 않았습니다” 메시지를 토스트로 표시

3 ‘locationOverlay’가 null이라면, “위치 정보를 사용할 수 없습니다” 메시지를 토스트로 표시

4 ‘currentLocation’이 null이면, “현재 위치를 찾을 수 없습니다” 메시지를 토스트로 표시

5 위치 권한이 허용되지 않은 경우, 앱에게 위치 권한을 요청

5.1 예외가 발생시, 스택 추적을 출력

6 현재 위치와 이전 위치 간의 거리가 50미터 미만일 때

6.1 갱신하지 않고 메소드를 종료

7. page. 지도 홈 기능 3과 동일하게 수행

**- 페이지 : 메시지 (friend\_list\_chat\_r)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 메시지페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 메시지 페이지에 접속 시

정상흐름:

1. setSupportActionBar()로 툴바를 액션바로 설정
   1. setDisplayHomeAsUpEnabled(true)로 뒤로가기 버튼 활성화
   2. setTitle()로 ‘name’에 저장된 문자열로 제목 표시
2. 뒤로가기 버튼을 누를 시 (onOptionsItemSelected 메서드)
   1. getItemId()로 뒤로가기의 ID를 반환하여 switch문을 실행
      1. finish()로 현재 액티비티를 종료
      2. true를 반환하여 성공적으로 처리되었음을 알림
      3. onOptionsItemSelected()로 상위 클래스의 메서드를 호출

대안흐름:

-

**기능 2: 친구 메시지 어댑터 (클래스 명: friend\_list\_chat\_adapter)**

발생 : <<객체 생성>>, <<데이터 결합>>, <<리스트 추가>>, <<리스트 호출>>, <<리스트 반환>>, <<총 개수 반환>>, <<위치 반환>>, <<버전, 주요 변경 사항 표시>>

액터 : USER, SYSTEM

목표 : 리사이클뷰를 구성하기 위한 어댑터를 정의

시작조건 :

1. 사용자가 메시지를 보낼 시

2. 친구와의 채팅 리스트가 있을 시

정상흐름:

1. 클래스의 생성자 메서드를 정의(friend\_list\_chat\_adapter)
   1. ‘friend\_list\_chat’ 리스트를 어댑터의 멤버 변수 ‘f\_c\_data’에 할당
2. 메서드 정의 (setItem 메서드)
   1. 날짜와 시간을 가져와 날짜(f\_l\_i\_d)를 설정
   2. 이름을 가져와 이름(f\_l\_i\_n)를 설정
   3. 메시지 내용을 가져와 메시지 내용(text)를 설정
   4. getProfileurl()로 ‘friend\_list\_chat’에서 프로필 이미지 URL을 호출(imageUrl)
   5. get()로 Picasso 라이브러리의 인스턴스 호출
      1. load()로 지정된 URL에서 이미지를 로드
      2. transform()로 이미지를 원형으로 변환
      3. into()로 로드한 이미지를 프로필 이미지 (f\_l\_i\_p)에 표시

대안흐름:

**기능 2: 메시지 읽어오기**

발생 : <<Firestore에서 데이터 호출>>, <<어댑터 생성>>, <<Firestore에서 데이터 변화 감지>>

액터 : USER, SYSTEM

목표 : DB에서 메시지 데이터 불러오기 및 시간 설정

시작조건 :

1. 메시지 페이지에 접속하여 대화방에 접속 시

정상흐름:

1. Firestore 데이터베이스 호출
   1. collection()와 document()로 Firestore의 ‘users’- > ‘현재 사용자의 이메일 -> “친구목록” -> 특정 이메일” -> "chat" 기반으로한 문서를 선택
   2. orderBy()로 날짜("date")를 기준으로 결과를 정렬
   3. addSnapshotListener()는 Firestore 쿼리의 결과를 실시간으로 감시하고, 이를 즉시 수신할 수 있도록 해주는 메서드
      1. Firestore 쿼리 결과의 변경 사항을 처리 (onEvent 메서드)
         1. getDocumentChanges()로 변경된 문서 목록을 호출하여 반복 실행 (dc)
         2. dc의 변경사항을 감지한다.
            1. ADDED로 문서가 추가된 경우

name, date, photourl, text, email를 키 값으로 가져와 friend\_list\_chat객체로 해당 값들을 추가

( 채팅 메시지를 표시할 목록에 새로운 항목을 추가하는 메서드)

1.3.1.2.1.2notifyDataSetChanged()로 데이터가 변경되어 어댑터 알림(friend\_list\_chat\_adapter)

1.3.1.2.1.3 리사이클러뷰를 통해 데이터 노출(friend\_list\_chat)

대안흐름:

1. 에러가 발생한 경우, 에러 로그 출력 메서드의 실행을 즉시 종료

2. ‘snapshots’와 ‘e’를 매개변수로 받음 (snapshots: 쿼리 결과에 대한 스냅샷 e: Firestore 작업 중 발생한 예외)

2.1 에러가 발생한 경우, 에러 로그 출력 메서드의 실행을 즉시 종료

**기능 3: 메시지 보내기**

발생 :<<시간 형식 설정>>, <<Firestore에서 데이터 호출>>, <<어댑터 생성>>, <<스레드 생성>>, <<스레드 시작>>, <<메시지 보내기 버튼>>, <<시간 업데이트>>

액터 : USER, SYSTEM

목표 : DB에 시간과 메시지 입력하기 위함.

시작조건 :

1. 메시지 페이지에 접속하여 대화방에 접속 시

정상흐름:

1. 시간 형식 설정
   1. getTimeZone() 로 “Asia/Seoul”의 표준 시간을 설정 (timeZone)
   2. Date()로 현재 날짜와 시간 정보를 포함하는 객체 생성(currentDate)
   3. SimpleDateFormat()로 "yyyy-MM-dd HH:mm:ss" 형식으로 설정 (dateFormat)
   4. setTimeZone()로 'timeZone'을 'dateFormat' 형식으로 설정
   5. format()로 현재 시간을 'dateFormat' 형식으로 변환(formattedDate)
2. 메시지 보내기 버튼
   1. setOnClickListener()로 사용자가 메시지 보내기(imageButton)을 누를 시
   2. getText()로 사용자가(editText)입력한 텍스트를 호출(userInput)
   3. setText("")로 사용자가 입력한 텍스트(editText)를 지움
   4. collection()와 document()로 Firestore의 users -> 사용자의 “친구목록” -> 특정 사용자의 이메일

->"chat ->" 고유한 ID를 기반으로 한 문서에 접근

* + 1. set()로 Firestore에 문서를 설정
       1. HashMap을 생성
          1. put()로 현재 사용자의 이메일을 추가
          2. 2.4.6.1.1.과 동일하게 수행

( 현재 사용자의 표시 이름, 현재 사용자의 프로필 사진 URL, 현재 날짜와 시간, 사용자가 입력한 메시지 텍스트)

* + - * 1. addOnSuccessListener로 성공적으로 수행한 경우, 아무런 수행하지 않음
  1. 2.4. ~ 2.4.1.1.3. 와 동일하게 수행(여기선 대화 상대의 이메일을 기반으로 한 경로를 사용함)
  2. randomUUID()로 무작위로 생성된 고유한 식별자를 호출(chatDocumentId)

1. 스레드가 중단될 때까지 계속해서 시간을 갱신(run 메서드)
   1. while 루프로 현재 스레드가 중단될 때까지 실행
      1. try-catch 로 현재 시간을 가져온 후 한국 표준시간대로 설정
         1. Date()로 현재 날짜와 시간 정보를 포함하는 객체 생성(currentDate)
         2. SimpleDateFormat()로 "yyyy-MM-dd HH:mm:ss" 형식으로 설정 (dateFormat)
         3. setTimeZone()로 시간 형식을 설정
            1. “Asia/Seoul”로 한국 표준 시간대를 나타냄
         4. format()로 현재 시간을 'dateFormat' 형식으로 변환(formattedDate)
         5. sleep()로 현재 스레드를 1초간 일시 정지

대안흐름:

1. addOnFailureListener로 작업이 실패 시, 아무 작업도 수행하지 않음
2. 업데이트를 실패하면, "Error updating read status"로 실패 로그를 기록’
3. 예외가 발생 시 (스레드 인터럽트 시), interrupt()로 현재 실행 중인 스레드를 인터럽트 상태로 설정

**기능 4. 친구 목록 가져오기**

1. 햄버거 페이지 (Header) page. 친구 목록 기능

2. 햄버거 페이지 (Header) page.친구 목록 어댑터 기능

(흐름이 똑같음)

**- 페이지 : 설정 (set\_activity)**

**기능 1**: 툴바

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<지도 홈으로 이동>>

액터 : USER, SYSTEM

목표 : 사용자가 설정페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 설정페이지에 접속 시

정상흐름:

1. setSupportActionBar()로 툴바를 액션바로 설정
   1. setDisplayHomeAsUpEnabled(true)로 뒤로가기 버튼 활성화
   2. setHomeAsUpIndicator()로 뒤로가기 버튼의 이미지를 변경
   3. setTitle()로 “설정”이라고 제목 표시
2. 뒤로가기 버튼을 누를 시(‘onOptionsItemSelected’ 메서드)
   1. getItemId()로 뒤로가기의 ID를 반환하여 switch문을 실행
      1. 지도 홈 (메인페이지)으로 이동하는 Intent를 생성
      2. startActivity()로 Intent를 실행
      3. true를 반환하여 성공적으로 처리되었음을 알림

대안흐름:

1. switch문에서 처리되지 않은 항목이 있을 시, ‘onOptionsItemSelected’ 메서드를 호출하는 것으로 반환

**기능 2**: 페이지 이동

발생 : <<소리 및 알림 설정으로 이동>>, <<신고 설정으로 이동>>, <<언어 설정으로 이동>>, <<업데이트 정보로 이동>>, <<회원 탈퇴로 이동>>

액터 : USER, SYSTEM

목표 : 설정 페이지에 있는 페이지 이동 버튼을 누를 시, 해당 페이지로 이동

시작조건 :

1. 사용자가 설정 페이지에 있는 페이지 이동 버튼을 누를 시

정상흐름:

1. 마이페이지
   1. setOnClickListener()로 마이페이지로 이동하는 버튼을 누를 시
      1. 마이페이지로 이동하는 Intent를 생성
      2. startActivity()로 Intent를 실행
2. 소리 및 알림 설정, 신고 설정, 언어 설정, 업데이트 정보, 회원 탈퇴
   1. 1.1~1.1.2 와 동일한 코드

(소리 및 알림 설정, 신고 설정, 언어 설정, 업데이트 정보, 회원 탈퇴 페이지로 이동하는 버튼)

대안흐름:

-

**-페이지 : 마이페이지 (Mypage)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 개인정보 수정 페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 마이페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

(툴바 제목: “프로필”, 뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 페이지 이동

발생 : <<개인정보 수정으로 이동>>, <<친구 관리로 이동>>, <<FQA로 이동>>, <<개인정보 보호 정책으로 이동>>, <<공지사항으로 이동>>

액터 : USER, SYSTEM

목표 : 마이페이지에 있는 페이지 이동 버튼을 누를 시, 해당 페이지로 이동

시작조건 :

1. 사용자가 설정 페이지에 있는 페이지 이동 버튼을 누를 시

정상흐름:

1. 마이페이지
   1. setOnClickListener()로 개인정보 수정으로 이동하는 버튼을 누를 시
      1. 개인정보 수정으로 이동하는 Intent를 생성
      2. startActivity()로 Intent를 실행
2. 친구관리, FQA, 개인정보 보호정책, 공지사항
   1. 1.1~1.1.2 와 동일한 코드

(친구관리, FQA, 개인정보 보호정책, 공지사항 페이지로 이동하는 버튼)

**기능3** : 본인 위치 정보 업데이트

발생 : <<위치 권한 요청>>, <<위치 업데이트>>, <<위치 정보 표시>>

액터: USER, SYSTEM

목표 : 사용자의 위치좌표를 '현재위치기록창'에 표시하고 1분간격으로 업데이트

시작조건

1. 위치권한이 승인된 상태

정상흐름

1. 위치 권한 확인:
   1. ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCATION)메서드를 호출 하여 위치 권한이 부여되었는지 확인
   2. 권한이 승인된 경우 위치 정보 업데이트를 시작하는 메서드(startLocationUpdates())를 호출
2. 위치 업데이트 설정
   1. startLocationUpdates()메서드를 호출
      1. handler = new Handler(Looper.getMainLooper())로 UI스레드에서 위치업데이트 작업을 할 핸들러 객체를 생성
      2. locationUpdater의 run()메서드 내에서 updateLocation()함수를 호출하여 위치 데이터를 새로 갱신
         1. locationUpdater = new Runnable()로 주기적인 위치 업데이트를 위한 Runnable객체를 생성
         2. handler.postDelayed(locationUpdater, 60000)를 호출하여 1분(60,000 밀리초) 간격으로 locationUpdater의 run()메서드가 실행되도록 설정
            1. 1분 간격으로 Runnable을 실행하여 주기적으로 위치를 업데이트
3. 위치 정보 요청:
   1. updateLocation()메서드를 호출
      1. fusedLocationClient.getLastLocation().addOnSuccessListener(this, location -> { ... })호출하여 마지막으로 확인된 위도정보와 경도정보를 비동기적으로 가져오기 위한 준비를 함
   2. if (location != null)조건문을 사용하여 위치 정보가 유효한 경우(location이 null이 아닐 경우)
      1. String locationText = "위도: " + location.getLatitude() + ", 경도: " + location.getLongitude();
         1. 위치 정보를 문자열 형식으로 포맷
      2. currentLocationLogTextView.setText(locationText)호출
         1. UI에 위치 정보를 업데이트하여 표시
4. 위치 정보 주기적으로 갱신
   1. handler.post(locationUpdater)호출
      1. 설정한 Runnable을 호출하여 1분마다 위치 업데이트를 반복

예외 흐름

1. 위치 권한 거부 시
   1. 권한 요청 결과 처리 메서드 onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)에서:
      1. if (requestCode == LOCATION\_PERMISSION\_REQUEST\_CODE)조건문으로 요청 코드가 맞는지 확인합니다.
      2. 권한이 승인되지 않은 경우
         1. 메서드를 종료하고 사용자에게 위치 권한 요청 대화상자를 다시 표시
2. 위치 정보가 없는 경우
   1. getLastLocation()호출 시
      1. 위치 정보가 null일 경우 기존 위치 정보를 유지하며, UI에는 이전 위치 정보를 표시
         1. if (location == null) { ... }
         2. currentLocationLogTextView.setText("위치 정보를 불러올 수 없습니다.");와 같은 오류 메시지를 UI에 표시

**- 페이지 : 개인정보 수정 (ProfileActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 개인정보 수정 페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 마이페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

(툴바 제목: “프로필”, 뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 프로필 수정( 클래스 명: ProfileActivity)

발생 : <<데이터 호출>>, <<닉네임 저장>>, <<상태메시지 저장>>, <<이미지 선택>>

액터 : USER, SYSTEM

목표 : 사용자가 이미지, 닉네임, 상태메시지를 변경할 경우, 변경한 내용을 저장

시작조건 :

1. 마이페이지에서 개인정보 수정버튼을 눌러, ProfileActivity화면에 진입

정상흐름:

1. 화면초기화
   1. Firebase Authentication을 통해 현재 로그인된 사용자의 정보를 currentUser로 가져옴
      1. 닉네임: currentUser.getDisplayName()값을 editTextNickname에 설정.
      2. 이메일: currentUser.getEmail()값을 editTextEmail에 설정.
      3. 프로필 사진: currentUser.getPhotoUrl()을 Picasso라이브러리로 불러와 imageViewProfile에 로드
   2. Firestore에서 사용자 상태 메시지를 가져오기 위해 loadStatusMessage()가 호출
      1. Firestore에서 currentUser.getUid()로 해당 문서(document)를 조회하여 statusMessage값을 editTextStatus에 설정.
2. 프로필 사진 변경:
   1. 사용자가 "프로필 사진 변경" 버튼을 클릭하면 dispatchTakePictureIntent()메서드가 호출
   2. 호출된 dispatchTakePictureIntent()는 Intent를 생성
      1. Intent는 사용자가 이미지를 선택할 수 있도록 이미지갤러리를 표시
      2. 사진을 선택 후 onActivityResult()메서드를 통해 처리
         1. 선택한 이미지의 URI를 가져와 MediaStore.Images.Media.getBitmap()으로 Bitmap형식으로 변환
         2. 해당 Bitmap은 imageViewProfile에 표시되고 selectedBitmap변수에 저장
3. 닉네임, 상태메시지, 프로필의 변경 사항 저장
   1. 사용자가 "저장" 버튼을 클릭하면 saveProfileChanges()메서드를 호출하여 닉네임, 상태메시지, 프로필의 변경사항을 저장
      1. 닉네임 업데이트
         1. editTextNickname의 값이 변경되었을 경우, UserProfileChangeRequest.Builder객체가 생성
         2. setDisplayName(nickname)을 통해 닉네임을 설정하고, currentUser.updateProfile()로 Firebase Authentication에 업데이트
      2. 프로필 사진 업로드
         1. selectedBitmap이 null이 아닌 경우(사진선택을 한 상태), uploadImageToFirebaseStorage()메서드가 호출되어 Firebase Storage에 이미지를 업로드
         2. 이미지 업로드가 완료되면 다운로드 URL을 받아 profileUpdates.setPhotoUri(imageUri)를 호출하여 프로필에 새로운 사진 URL을 적용
         3. Firebase Authentication에 프로필 사진이 저장되며, 사용자의 프로필이 업데이트됨
      3. 상태 메시지 저장
         1. 사용자가 입력한 상태 메시지(editTextStatus)는 Firestore에 saveStatusMessage()메서드를 통해 저장
         2. firestore.collection("users").document(currentUser.getUid())로 참조한 문서에 상태 메시지가 병합(merge) 방식으로 저장

예외 흐름

1. 사용자가 갤러리에서 사진을 선택하지 않거나 잘못된 이미지 파일을 선택했을 경우
   1. onActivityResult()에서 URI 또는 Bitmap이 null일 경우 아무 동작도 수행되지 않고 기존 사진이 유지
2. Firestore에서 상태 메시지 또는 닉네임 업데이트 실패 시, 업데이트 실패 경고가 Toast로 표시

**- 페이지 : 소리 및 알림 설정 (SoundActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 소리 및 알림 설정페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 소리 및 알림 설정 페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행(툴바 제목: “소리 및 알림”, 뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능2**: 사운드 설정 관리

발생: <<소리 설정 화면 열림>>, <<볼륨 조절>>, <<알림 설정 변경>>

액터: USER, SYSTEM

목표: 사용자가 사운드 설정 화면에서 볼륨과 알림 설정을 관리

시작 조건:

1. 사용자가 사운드 설정 화면에 진입

정상 흐름:

1. 소리 설정 화면 표시
   1. 사용자가 소리 설정 화면(soundactivity\_layout.xml)에 진입 시, "SharedPreferences"을 불러옴
      1. 채팅 알람 볼륨: SharedPreferences에서 "ChatVolume" 값을 불러옴
      2. 비상벨 소리 볼륨: SharedPreferences에서 "EmergencyVolume" 값을 불러옴
      3. 채팅 알림 활성화 여부: SharedPreferences에서 "ChatNotificationEnabled" 값을 불러옴, 이 값이 없으면 기본값으로 true로 설정
2. UI 초기화
   1. AudioManager를 초기화하여 시스템 볼륨 조절을 준비
      1. AudioManager 인스턴스를 생성
      2. 해당 메서드(getStreamVolume(), setStreamVolume(), getStreamMaxVolume())를 사용하기위해AudioManager 를 초기화
   2. XML로부터 SeekBar와 SwitchUI 요소들을 로드하고 초기화
   3. initSeekBarAndSwitch()메서드를 호출하여 초기화된 설정 값을 UI에 반영
3. 채팅 알람 볼륨 조절
   1. 사용자가 채팅 알람 볼륨 SeekBar를 조절
      1. SeekBar의 값이 변경 시
         1. AudioManager를 통해 채팅 알림 볼륨이 실시간으로 변경
      2. 사용자가 SeekBar조작을 마치면, 변경된 볼륨 값이 SharedPreferences에 저장
4. 비상벨 소리 조절
   1. 사용자가 비상벨 소리 SeekBar를 조절
      1. SeekBar의 값이 변경 시
         1. AudioManager를 통해 비상 알림 볼륨이 실시간으로 변경
      2. 사용자가 SeekBar조작을 마치면, 변경된 볼륨 값이 SharedPreferences에 저장
5. 채팅 알림 설정 조정
   1. 사용자가 채팅 알림 설정 Switch를 조작
      1. Switch의 상태가 변경되면, 변경된 설정 값이 SharedPreferences에 저장

대안 흐름

1. SharedPreferences에서 설정 값을 불러오지 못할 경우, 기본값을 사용하여 UI를 초기화
   1. SharedPreferences에서 "ChatVolume" 값의 초기값은 50으로 설정
   2. SharedPreferences에서 "EmergencyVolume" 값의 초기값은 50으로 설정
   3. SharedPreferences에서 "ChatNotificationEnabled" 값이 null일 경우 ture로 설정

**- 페이지 : 신고 설정 (ReportActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 신고 설정페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 신고 설정페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행(툴바 제목: “신고 설정”, 뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 긴급 신고 시간 설정

발생: <<시간 선택 다이얼로그>>, <<긴급 신고 타이머 시작>>

액터: USER

목표: 사용자가 긴급 신고 시간을 선택, 신고버튼을 클릭 시 신고가 들어가기까지의 예의시간을 설정 할 수 있음

시작 조건:

1. 신고설정(ReportActivity.xml)이 화면에 표시된 상태.

정상 흐름:

1. 화면 진입 및 초기화
   1. 사용자가 비상 신고 설정 화면(reportactivity\_layout.xml)에 진입하면 SharedPreferences를 사용하여 이전에 저장된 신고 시간(selectedTimeInSeconds) 값을 불러옴.
   2. selectedTimeInSeconds 값을 화면에 표시하기 위해 updateDisplayedTime()메서드를 호출하여, timeDisplay텍스트 뷰에 현재 설정된 시간(SharedPreferences)이 표시.
2. 시간 감소 (왼쪽 화살표 클릭)
   1. 사용자가 왼쪽 화살표(leftArrow)를 클릭 시 , decreaseTime()메서드가 호출됨.
   2. 이 메서드는 selectedTimeInSeconds값이 7초보다 클 경우, 1초 감소시킴.
   3. 시간이 감소한 후:
      1. updateDisplayedTime()메서드를 호출해 '감소된 시간' 값을 화면에 표시.
      2. saveTimeToPreferences()메서드를 호출해 '감소된 시간'이 SharedPreferences에 저장됨.
3. 시간 증가 (오른쪽 화살표 클릭)
   1. 사용자가 오른쪽 화살표(rightArrow)를 클릭 시 , increaseTime()메서드가 호출됨.
   2. 이 메서드는 selectedTimeInSeconds값이 30초보다 작을 경우, 1초 증가시킴.
   3. 시간이 증가한 후
      1. 기능2의 정상흐름 2.3.1과 동일하게 수행
      2. 기능 2의 정상흐름 2.3.2과 동일하게 수행
4. 신고버튼 팝업창에 수정된 시간 적용
   1. SharedPreferences 에 저장된 값을 selectedTimeInSeconds에 저장
   2. setMessage를 통해 selectedTimeInSeconds값을 출력

대안 흐름:

1. 저장된 시간이 없을 경우
   1. SharedPreferences의 값이 null일 경우 기본값인 7초가 설정됨.
2. 최소 및 최대 범위 이탈
   1. 사용자가 설정을 7초 이하로 감소하거나 30초 이상으로 증가시키려 할 때, 조건에 맞지 않으므로 값이 변경되지 않음.

**기능 3**: 친구 선택

발생: <<친구 선택 화면 열기>>, <<친구 선택 결과 처리>>

액터: USER, SYSTEM

목표: 사용자가 긴급 신고 알림을 보낼 친구를 선택할 수 있음

시작 조건:

1. ReportActivity가 실행 중이며, 사용자가 친구를 선택할 준비가 되어 있음.

정상 흐름:

1. 친구 목록 데이터 로드
   1. FriendListActivity의 onCreate()메서드에서
      1. RecyclerView를 초기화하고 setLayoutManager(new LinearLayoutManager(this))를 호출하여 리스트 형태로 친구 목록을 표시
      2. FirebaseFirestore인스턴스를 db = FirebaseFirestore.getInstance()로 초기화
      3. loadFriendsFromFirestore()메서드를 호출하여 Firestore에서 친구 목록 데이터를 로드함
2. 친구 목록 표시
   1. loadFriendsFromFirestore()메서드 내에서
      1. db.collection("friends").get()을 호출하여 Firestore에서 "friends" 컬렉션의 데이터를 가져옴
      2. addOnCompleteListener(task -> {...})를 사용하여 데이터 로딩이 완료된 후
         1. for루프를 통해 각 친구 정보(selectedFriendName, selectedFriendEmail, selectedFriendStatus)를 Friend객체로 생성하여 friendList에 추가
         2. friendAdapter.notifyDataSetChanged()메서드를 호출하여 RecyclerView를 업데이트하고 사용자에게 친구 목록을 표시
3. 친구 선택
   1. 사용자가 RecyclerView의 특정 친구 아이템을 클릭
      1. 해당 친구의 정보(selectedFriendName, selectedFriendEmail, selectedFriendStatus)를 포함한 Intent를 생성
      2. setResult(RESULT\_OK, intent)를 호출하여 선택된 친구 정보를 ReportActivity로 반환
      3. finish()메서드를 호출하여 FriendListActivity를 종료
4. 선택된 친구 정보 수신
   1. ReportActivity의 onActivityResult(int requestCode, int resultCode, Intent data)메서드에서
      1. if (requestCode == 100 && resultCode == RESULT\_OK)조건을 확인
      2. data.getStringExtra("selectedFriendName"), data.getStringExtra("selectedFriendEmail"), data.getStringExtra("selectedFriendStatus")메서드를 사용하여 선택된 친구 정보를 수신하고 저장
      3. savePreferences()메서드를 호출하여 선택된 친구 정보를 SharedPreferences에 저장
      4. 이후 ReportActivity로 되돌아옴

대안 흐름:

1. 친구 선택 후 데이터가 null인 경우
   1. 사용자가 친구를 선택하지 않거나 데이터가 null일 경우
      1. Toast.makeText(getApplicationContext(), "친구를 선택해주세요", Toast.LENGTH\_SHORT).show()메서드를 호출하여 메시지를 표시

**기능 4: 신고버튼 기능**

발생 : <<긴급 신고 버튼 클릭>>, <<카운트다운 시작>>, <<112 전화 연결>>, <<앱 내 긴급 메시지 전송>>

액터 : USER, SYSTEM

목표 : 사용자가 버튼을 클릭 시 예의의 시간 초과 후, 112에 긴급신고

시작조건 :

1. 지도 홈(탐색)에서 긴급신고버튼을 클릭 시

정상흐름:

1. 비상 알람 실행
   1. AlarmManager가 예약된 시간이 되면 EmergencyBroadcastReceiver의 onReceive(Context context, Intent intent)메서드를 호출.
   2. intent에는 등록된 친구 목록(selectedFriends) 데이터가 포함되어 있음.
   3. 로그에 "onReceive called!" 메시지를 출력하여 알람 호출이 정상적으로 실행되었음을 기록.
2. 112 신고 호출
   1. sendEmergencyCall(Context context)메서드 실행
      1. CALL\_PHONE 권한 확인
         1. ActivityCompat.checkSelfPermission()을 통해 CALL\_PHONE권한이 있는지 확인.
         2. 권한이 있을 경우
            1. Intent.ACTION\_CALL로 112 번호로 전화를 걸도록 설정.
            2. 전화번호는 Uri.parse("tel:112")로 지정.
            3. context.startActivity(callIntent)를 호출하여 전화를 실행.
3. 친구 목록 가져오기
   1. Intent에서 selectedFriends데이터를 가져옴.
      1. 데이터가 없거나 비어 있을 경우
         1. getSelectedFriends(Context context)메서드를 호출하여 SharedPreferences에서 저장된 친구 목록을 로드.
      2. 로드된 친구 목록은 ArrayList<String>형태로 반환.
      3. 친구 목록이 여전히 비어 있을 경우:
         1. Log에 "친구 목록이 비어 있습니다." 메시지를 기록.
         2. 메시지 전송 작업 중단.
4. 비상 메시지 전송
   1. 친구 목록(selectedFriends)에 포함된 각 친구 이메일에 대해 sendEmergencyMessage(String friendEmail)메서드를 반복 실행
5. Firebase 초기화 및 사용자 정보 확인
   1. FirebaseAuth및 FirebaseFirestore인스턴스를 초기화.
   2. 현재 로그인된 사용자 정보를 가져오기 위해 FirebaseAuth.getInstance().getCurrentUser()호출.
      1. 현재 사용자 정보(FirebaseUser)가 null일 경우 메시지 전송 중단.
6. 비상 메시지 데이터 생성
   1. 비상 메시지에 포함될 데이터를 HashMap<String, Object>형태로 생성
      1. email: 현재 사용자의 이메일.
      2. name: 현재 사용자의 이름(getDisplayName()).
      3. photoUrl: 현재 사용자의 프로필 사진 URL(있을 경우).
      4. date: 현재 날짜와 시간(형식: "yyyy-MM-dd HH:mm:ss").
      5. text: 고정 메시지 "비상!!!".
7. 비상 메시지 저장
   1. Firebase Firestore의 두 경로에 비상 메시지를 저장
      1. 현재 사용자가 친구에게 전송하는 채팅 데이터
         1. 경로: users/{currentUserEmail}/친구목록/{friendEmail}/chat/{chatDocumentId}
      2. 친구가 전송받은 메시지 데이터
         1. 경로: users/{friendEmail}/친구목록/{currentUserEmail}/chat/{chatDocumentId}
   2. 저장 성공 시 Log에 "비상 메시지가 전송되었습니다" 메시지 출력.

대안흐름:

1. 1. 112 신고 실패
   1. CALL\_PHONE권한이 없는 경우.
      1. Log에 "CALL\_PHONE 권한이 없습니다." 메시지를 기록.
      2. 사용자가 필요한 권한을 허용하도록 안내하는 추가 UI 또는 로직 구현 가능.
2. CALL\_PHONE 권한이 없을 경우
   1. 권한이 없을 경우
      1. Log에 "CALL\_PHONE 권한이 없습니다." 메시지를 기록.
      2. 작업 중단.
   2. 권한 요청 로직 추가 가능.
3. 메시지 전송 실패 시
   1. 저장 실패 시 Log에 "비상 메시지 전송 실패" 메시지와 예외를 출력.

**- 페이지 : 언어 설정 (LanguageActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 언어 설정페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 언어 설정페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행(툴바 제목: “언어”,뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 언어 변경

발생 : <<언어 선택 스피너 설정>>, <<스피너 호출>>, <<언어 변경 적용>>, <<드롭다운의 리스트 표시>>, <<액티비티 재시작>>

액터 : USER, SYSTEM

목표 : 언어 변경이나 폰트 변경 시, 변경내용을 적용

시작조건 :

1. 언어 변경이나 폰트 변경 시

정상흐름:

1. 언어 선택 스피너 설정
   1. getSharedPreferences()로 "MyPrefs"라는 SharedPreferences 파일을 호출(preferences)
   2. set()로 "LANGUAGE\_PREF\_KEY" 키에 해당하는 값 호출
      1. 해당 키에 저장된 값이 없으면 기본값으로 "한국어"를 사용(savedLanguage)
   3. 사용할 언어를 선택할 수 있도록 스피너를 초기화 ,setupLanguageSpinner 메서드인 2.로 이동

(이때 ‘preferences’와 ‘savedLanguage’가 전달됨)

1. setupLanguageSpinner 메서드
   1. XML에서 정의한 spinner를 호출(spinner)
   2. 문자열 배열을 가져와서 언어 목록(arratAdapter)에 저장
   3. ArrayAdapter를 사용하여 spinner에 언어 목록 드롭다운 형식의 리스트 표시로 설정
   4. get()로 이전에 저장된 언어 설정을 호출 (savedLanguage)
   5. getPosition()로 spinner 어댑터에 있는 몇 번째 항목에 해당하는 확인(savedLanguagePosition)
   6. setSelection()로 spinner의 초기 선택을 이전에 저장된 언어로 설정
   7. setOnItemSelectedListener()로 spinner에 선택 이벤트 리스너를 설정
      1. getItemAtPosition()로 선택된 항목의 값을 나타냄(selectedLanguage)
      2. 이전에 선택된 언어(savedLanguage)와 현재 선택된 언어(selectedLanguage)가 다르면, 수행

(사용자가 이전에 선택한 언어와 동일한 언어를 다시 선택했을 때, 처리를 건너뛸 수 있음)

* + 1. edit()로 ‘preferences’에 데이터를 수정하기 위해 호출(editor)
    2. putString()로 "LANGUAGE\_PREF\_KEY" 키 값을 selectedLanguage로 설정(editor)
    3. apply()로 변경 사항을 적용
    4. true로 설정하여 언어가 변경되었음을 나타냄(isLanguageChanged )
    5. 앱의 언어를 선택된 언어(selectedLanguage)로 변경(setLocale 메서드)
       1. 선택된 언어에 따라 Locale 객체를 생성하여 ‘locale’에 할당
       2. switch-case문을 통해 선택된 언어 판단
          1. “영어”라면, 영어에 해당하는 Locale 객체를 생성(en)

break로 블록의 실행을 종료

* + - * 1. 2.9.8.2.1. ~ 2.9.8.2.1.1 과 동일하게 수행

(“한국어” =>“ko”, “중국어”=> “zh”, “스페인어”=>“es”)

* + - * 1. 위의 언어 중 어떤 것에도 해당하지 않는 경우, 기본 로케일(Locale.getDefault())로 생성

(일반적으로 시스템의 기본 언어를 따름)

break로 블록의 실행을 종료

* + - 1. setDefault()로 선택된 언어에 해당하는 locale을 설정
      2. getResources()로 현재 액티비티의 리소스를 호출(resources)
      3. getConfiguration()로 현재 구성을 가져옴
      4. setLocale()로 새 locale을 저장
      5. updateConfiguration()로 업데이트된 구성을 현재 리소스에 적용한다.
      6. true로 설정하여 언어가 변경되었음을 나타냄(isLanguageChanged )
    1. set()로 이전에 선택한 언어가 변경된 언어로 저장
    2. setSelection()로 spinner에서 선택된 항목을 지정된 위치(position)으로 설정
    3. recreate()로 현재 액티비티를 재시작

대안 흐름:

1. spinner에서 항목을 선택하지 않은 경우, 어떠한 수행하지 않음

**- 페이지 : 업데이트 정보 (UpdateActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 업데이트 정보페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 업데이트 정보페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행(툴바 제목: “업데이트 정보”,뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 업데이트 정보 어댑터 (클래스 명: update\_adapter)

발생 : <<객체 생성>>, <<데이터 결합>>, <<리스트 추가>>, <<리스트 호출>>, <<리스트 반환>>, <<총 개수 반환>>, <<위치 반환>>, <<버전, 주요 변경 사항 표시>>

액터 : USER, SYSTEM

목표 : 리사이클뷰를 구성하기 위한 ‘update\_adapter’를 정의

시작조건 :

1. 업데이트 정보가 있을 시

정상흐름:

1. 각 아이템을 표시하기 위한 ViewHolder 클래스를 정의(RecyclerView.ViewHolder 클래스를 확장하여 정의)

1.1 해당 뷰의 구성 요소를 나타내는 버전 (‘vs’)와 버전 설명(‘text’)를 선언 (ViewHolder 메서드)

2. ‘update\_list’ 데이터를 ViewHolder의 각 뷰에 설정 (setItem 메서드)

2.1 getVs()로 해당 객체의 업데이트 버전( ‘vs’) 값을 호출

2.2 setText()로 업데이트 버전(vs)값을 설정(버전을 설정)

2.3 getText()로 텍스트 값을 호출 (itemText)

3.4 replace("-", "\n")로 줄바꿈 문자로 대체 (modifiedText)

3.5 setText()로 modifiedText값을 설명글(TextView)에 설정(주요 변경 사항을 설정)

대안흐름:

-

**기능 3**: 업데이트 정보 표시

발생 : <<RecyclerView 설정>>, <<Firestore에서 데이터 호출>>, <<어댑터>>, <<업데이트 정보 표시>>

액터 : USER, SYSTEM

목표 : 업데이트의 정보와 버전을 표시

시작조건 :

1. 업데이트 정보 페이지에 접속 시

정상흐름:

1. RecyclerView 설정
   1. UpdateAdapter()로 업데이트 리스트(update\_list)에 대한 어댑터를 생성 (update\_adapter)
2. Firestore에서 데이터 호출
   1. getInstance()로 Firestore 데이터베이스의 인스턴스를 호출
   2. collection()로 "update"라는 이름의 컬렉션을 참조(updateCollection)
   3. get()로 ‘updateCollection’의 모든 문서를 호출하며 비동기적으로 작동
      1. isSuccessful()로 문서 호출에 성공 시, getResult()로 가져온 결과를 반복문으로 순회(이때 각 문서를 ‘QueryDocumentSnapshot’ 객체로 가져옴)
         1. getString()로 “버전”의 데이터를 호출 (version)
         2. getString()로 “주요 변경 사항”의 데이터를 호출 (content)
         3. 해당 값들을add()로 ‘item’을 ‘update\_list’에 추가
         4. notifyDataSetChanged()로 어댑터의 변경 사항을 알림

대안흐름:

-

**- 페이지 : 회원 탈퇴 (WithdrawalActivity)**

**기능 1**: 툴바 띄우기

발생 : <<툴바 생성>>, <<툴바 제목 표시>>, <<뒤로가기 버튼>>, <<설정으로 이동>>,

액터 : USER, SYSTEM

목표 : 사용자가 회원 탈퇴페이지에 접속 시, 어떤 페이지인지와 뒤로가기가 가능하도록 띄움

시작조건 :

1. 회원 탈퇴페이지에 접속 시

정상흐름:

1. 설정 페이지의 기능 1과 동일하게 수행(툴바 제목: “회원 탈퇴”, 뒤로가기: 설정페이지로 이동)

대안흐름:

1. 설정 페이지의 기능 1과 동일하게 수행

**기능 2**: 사용자 계정 탈퇴 및 데이터 삭제

발생: <<데이터베이스 호출>>, <<사용자 인증>>, <<팝업 창>>, <<앱 종료>>

액터: USER, SYSTEM

목표: 사용자가 계정 탈퇴를 요청할 경우, Firebase Firestore에서 사용자 데이터를 삭제하고 앱을 종료합니다.

시작 조건:

1. WithdrawalActivity가 실행되고 사용자 화면에 표시
   1. 사용자에게 계정 삭제를 위한 동의 (checkBoxAgree)를 요구

정상 흐름:

1. CheckBox 선택
   1. 사용자가 checkBoxAgree(이용약관 동의 확인용 체크박스)를 체크하면, setOnCheckedChangeListener()메서드가 호출됨
      1. setOnCheckedChangeListener()메서드는 isChecked값에 따라 buttonConfirm(확인 버튼)의 활성화 여부를 결정
2. 탈퇴 확인 다이얼로그
   1. 사용자가 활성화된 buttonConfirm을 클릭하면, 탈퇴를 재확인하는 showConfirmationDialog()메서드가 호출
      1. 이 메서드는 사용자에게 경고 메시지를 포함한 AlertDialog를 띄워 탈퇴 의사를 물음
      2. AlertDialog 구성
         1. '탈퇴하기' 버튼이 있으며, 이를 클릭하면 deleteUserData()메서드가 호출
         2. '취소' 버튼이 있으며, 취소를 누르면 아무런 액션도 취하지 않고 다이얼로그를 닫음
3. Firestore에서 사용자 데이터 삭제
   1. 사용자가 '탈퇴하기' 버튼을 클릭하면 deleteUserData()메서드가 실행되어, Firestore에 저장된 사용자 데이터를 삭제
      1. auth.getCurrentUser()를 통해 현재 로그인된 사용자의 FirebaseUser객체를 가져옴
      2. Firestore의 users컬렉션에서 해당 사용자의 document(user.getUid()로 가져온 사용자 ID를 키로 사용)를 참조하여 삭제
      3. Firestore 데이터가 삭제된 후, 계정을 삭제하기 위한 재인증을 진행하는 reauthenticateAndDeleteUser()메서드를 호출
4. Firebase Authentication 재인증 및 계정 삭제
   1. reauthenticateAndDeleteUser()메서드에서 Google 로그인을 사용하여 Firebase Authentication에 저장된 계정 정보를 삭제하기 위한 재인증을 수행
      1. GoogleSignIn.getLastSignedInAccount()를 호출하여 현재 로그인된 구글 계정 정보를 가져옴
      2. Firebase 자격증명
         1. GoogleSignInAccount객체에서 getIdToken()메서드를 호출하여 idToken(Google에 로그인 시 발급된 JWT(Json Web Token))을 추출
         2. idToken과 null값을 사용하여GoogleAuthProvider.getCredential()메서드를 통해 Firebase자격증명을 생성
            1. idToken은 사용자의 인증정보를 담고 있으며, 인증을 위한 추가정보가 불필요하기에 null값을 사용
         3. 생성된 자격 증명을 사용하여 user.reauthenticate(credential)메서드를 호출해 재인증을 시도
         4. 재인증이 성공하면 onCompleteListener가 호출되고, 이어서 user.delete()메서드를 통해 Firebase Authentication에서 사용자 계정이 삭제됨
   2. 계정 삭제 성공
      1. 계정 삭제가 성공하면 "계정이 성공적으로 삭제되었습니다"라는 토스트 메시지를 띄움
      2. 이후, MainActivity로 화면을 전환하여 로그인 화면으로 돌아감
         1. Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK플래그를 사용해 기존 액티비티 스택을 모두 지우고 새로운 로그인 화면으로 이동
      3. 마지막으로 finish()로 현재 탈퇴 화면을 종료

대안 흐름:

1. 계정 삭제 요청 취소
   1. 사용자가 "취소" 버튼 (buttonCancel)을 클릭하면, WithdrawalActivity가 종료되고 이전 화면으로 돌아감(finish()).
2. Firebase 사용자 데이터 삭제 실패 시3.2.1. Toast메시지로 "정보 초기화에 실패했습니다."를 표시
   1. 오류 로그를 기록
3. 재인증 실패: 만약 재인증에 실패할 경우
   1. "재인증에 실패했습니다"라는 토스트 메시지를 띄우고 계정 삭제 절차를 중단한다.

**-페이지 : 카메라 (introActivity)**

**기능 1 : 카메라 기능**

발생 : <<카메라 초기화>>, <<화면 미리보기>>

액터 : USER, SYSTEM

목표 : 사용자 요청에 따라 카메라의 기능을 활성화 시킴

시작조건 :

1. 사용자가 카메라 버튼을 눌러 화면이 이동
2. 카메라 권한이 활성화 상태인 경우

정상흐름 :

1. 권한 확인
   1. onCreate()호출 시 allPermissionsGranted()메서드 실행.
      1. 필수 권한: CAMERA, RECORD\_AUDIO.
      2. ContextCompat.checkSelfPermission()으로 카메라와 오디오 권한 상태를 확인.
   2. 사용자가 카메라와 오디오 권한을 허용하면 startCamera()호출.
2. 카메라 세팅
   1. startCamera()에서 ProcessCameraProvider.getInstance()를 통해 카메라 제공자를 비동기적으로 호출.
   2. cameraProviderFuture.addListener()를 통해 제공자 인스턴스 생성.
3. 미리보기 및 녹화 설정
   1. 미리보기
      1. Preview.Builder().build()로 카메라 미리보기를 하기 위한 객체 생성
   2. 촬영 화질:
      1. Recorder.Builder()에서 HD 화질로 초기화(QualitySelector.from(Quality.HD)).
      2. VideoCapture.withOutput(recorder)를 통해 녹화 기능 활성화.
4. 미리보기 화면 표시
   1. 레이아웃의 PreviewView에서 SurfaceProvider를 가져와 미리보기 데이터를 연결.
5. 로그 출력
   1. 초기화 성공 시 정상 로그 출력.
   2. 실패 시 예외 처리로 오류 로그 및 사용자 알림 표시.

대안흐름 :

1. 권한 거부
   1. 사용자가 권한을 거부하면 Toast로 권한부족 알림을 표시
   2. 권한이 없으므로 카메라 기능을 중단하고 화면 종료

**기능 2 : 녹화 기능**

발생 : <<녹화 기능>>

액터 : USER, SYSTEM

목표 : 사용자에게 녹화기능을 제공하고 녹화된 영상파일을 저장소로 저장

시작조건 :

1. 카메라 초기화가 이루어진 상태
2. 카메라 권한이 활성화 상태인 경우

정상흐름 :

1. 녹화 시작
   1. recording객체가 null인지 확인
      1. null이면 recording객체를 통하여 녹화 중이 아님을 확인
         1. startRecording()을 호출 후 녹화 시작
      2. null이 아닐경우 녹화 중으로 판단
         1. stopRecording()을 호출 후 녹화 중지
   2. startRecording()내부 작업
      1. 저장 디렉터리 설정
         1. Environment.getExternalStoragePublicDirectory()를 통해 경로 생성(Movies/MyAppVideos).
         2. 디렉터리가 없으면 mkdirs()호출로 생성.
      2. 파일 생성
         1. SimpleDateFormat을 사용해 파일명 생성(예: 2024-11-01-12-30-45.mp4).
         2. FileOutputOptions.Builder()로 파일 출력 옵션 설정.
      3. 녹화 시작
         1. videoCapture.getOutput().prepareRecording()호출로 녹화 준비.
         2. 오디오 활성화(withAudioEnabled()).
         3. .start()메서드로 녹화 시작.
         4. 이벤트 리스너 등록
            1. VideoRecordEvent.Start: "녹화 시작" 메시지를 Toast로 표시.
            2. VideoRecordEvent.Finalize:

녹화 완료 시 미디어 스캔을 요청해 갤러리에 추가.

1. 녹화 중지
   1. toggleRecording()에서 stopRecording()호출:
      1. recording.stop()호출로 녹화 종료.
      2. recording객체를 null로 초기화.
2. 미디어 스캔 요청
   1. 녹화 파일을 갤러리에 추가하기 위해 MediaScannerConnection.scanFile()호출.

대안 흐름 :

1. 권한 부족
   1. startRecording()에서 RECORD\_AUDIO권한이 없으면 ActivityCompat.requestPermissions()호출.
   2. 사용자 알림("녹화를 위해 오디오 권한이 필요합니다").
2. 녹화 실패
   1. videoCapture가 null인 경우 Toast로 오류 알림("녹화 기능이 초기화되지 않았습니다").

**기능 3 : 플래시 기능**

발생 : <<플래시 토글>>

액터 : USER, SYSTEM

목표 : 카메라의 플래시를 키거나 끌 수 있음

시작조건 :

1. 사용자가 플래시 버튼을 클릭 했을 경우
2. 카메라 초기화가 이루어진 경우

정상흐름 :

1. 플래시 상태 확인
   1. toggleFlash()호출 시 camera객체의 초기화 여부 확인.
   2. 현재 플래시 상태(isTorchOn)를 반전하여 설정.
   3. if (camera != null)조건을 통해 camera객체가 초기화되었는지 확인
2. 플래시 설정
   1. 현재 플래시 상태(isTorchOn)의 값을 반전시킨다.
      1. isTorchOn = !isTorchOn
   2. 플래시 제어
      1. camera.getCameraControl().enableTorch(isTorchOn)를 호출하여 플래시 상태를 업데이트.
         1. isTorchOn = true→ 플래시 켜짐.
         2. isTorchOn = false→ 플래시 꺼짐
         3. 상태 변경 후 isTorch0n값을 업데이트 하여 플래시상태를 유지
         4. 사용자에게 Toast메시지를 표시.

대안 흐름 :

1. 카메라 초기화 실패
   1. camera객체가 null일 경우 Toast로 오류 알림("카메라가 초기화되지 않았습니다").

**기능 4: 신고버튼 기능**

발생 : <<긴급 신고 버튼 클릭>>, <<카운트다운 시작>>, <<112 전화 연결>>, <<앱 내 긴급 메시지 전송>>

액터 : USER, SYSTEM

목표 : 사용자가 버튼을 클릭 시 예의의 시간 초과 후, 112에 긴급신고

시작조건 :

1. 카메라 화면에서 긴급신고버튼을 클릭 시

정상흐름:

신고설정 페이지의 기능 5과 동일하게 수행

대안흐름:

신고설정 페이지의 기능 5과 동일하게 수행

**기능 5: 네이버 지도 표시**

발생 : <<네이버 맵 api 호출>>, <<맵 불러오기>>

액터 : USER, SYSTEM

목표 : 사용자에게 네이버지도를 제공

시작조건 :

1. 지도 홈에 접속 시

정상흐름:

지도홈 페이지의 기능1과 동일하게 수행

대안흐름:

지도홈 페이지의 기능1과 동일하게 수행