Dandoor_AndroidApp

DandoorBTManager

`DandoorBTManager` 는 애플리케이션을 실행시킬 때 블루투스 권한을 확인하며, 차량을 제어하고 비콘 신호를 수신합니다.

다음 함수를 통해 블루투스 권한을 확인합니다. Activity를 인자로 받아서 블루투스 권한을 확인합니다. 권한이 있는 경우 true값을 callback함수를 이용하여 넘겨주고 권한이 없다면 해당 Activity에서 권한을 요청합니다.

```
fun checkBTPermission(activity: Activity, callback: (Boolean) -> Unit) {
    if (permissionManager.hasRequiredPermissions()) {
        callback(true)
    } else {
        permissionManager.requestBluetoothPermissions(activity)
    }
}
```

Activity에서 다음과 같이 활용 가능합니다. 현재의 Activity를 인자로 넣어서 함수를 호출하면 블루투스 권한이 있는 경우 granted에 callback으로 true값을 넘겨주어 이후 작업을 진행할 수 있으며, 그렇지 않은 경우 사용자에게 권한이 필요하다고 알리는 기능을 할 수 있습니다.

```
checkBTPermission(this) {granted ->
    if (granted) {
        // 블루투스 연결 이후 기능들
        // ex) 블루투스 차량 연결 이벤트를 처리할 callback 설정
    } else {
        // 사용자에게 권한 필요하다고 알림
    }
}
```

다음 코드를 통해 차량과 통신합니다. `setVehicleCallback` 함수는 차량 콜백 함수를 등록하여 이벤트가 발생하였을 때 메시지를 출력합니다. `connectToCar` 함수는 수동 페어링이 완료된 차량 블루투스 기기(HC-06)과 연결을 시도합니다. `sendCarCommand` 함수는 문자열을 받아서 차량 명령어를 통해 차량을 제어합니다. command: string { start, stop, toggle }

```
fun setVehicleCallback(callback: BTVehicle.VehicleConnectionCallback) {
    vehicleManager.vehicleConnectionCallback = callback
}
fun connectToCar() {
    vehicleManager.connectToCar()
}
fun sendCarCommand(command: String) {
    vehicleManager.sendCarCommand(command)
}
```

Activity에서 블루투스 권한을 확인하여 권한이 있는 경우 `setVecleCallback` 함수를 호출해서 차량 콜백 함수를 등록하며, `connectTocar` 함수 호출로 차량 블루투스 연결 시도, `sendCarCommand` 함수를 이용하여 상황에 맞는 명령으로 차량을 제어하는 기능으로 활용됩니다.

다음 코드를 통해 비콘 신호를 수신합니다. `startScan` 함수는 스캔을 시작하는 함수로 `callback` 을 등록하고 nRF필터를 통해 비콘의 신호를 수신합니다. `stopScan` 함수는 앞서 등록한 `callback` 객체를 통해 스캔을 종료합니다.

```
fun startScan() {
    beaconManager.startScan()
}
fun stopScan() {
    beaconManager.stopScan()
}
```

Activity에서는 다음과 같이 활용 가능합니다. 시간을 지정하지 않았다면 입력을 요청하는 메시지를 출력하고 정상적으로 입력이 되었다면 `startScan` 으로 스캔을 시작하며 타이머를 시작하고, 타이머가 종료되는 시점에 `stopScan` 을 호출하여 스캔을 종료합니다.

```
if (totalTimeInSeconds <= 0){
    // 시간 입력 요청 메시지
} else {
    // startScan() 으로 스캔 시작
    // 타이머 시작
}
// 타이머가 종료되면 stopScan()으로 스캔 종료
```

DataManager

`DataManager` 는 Room 패키지를 활용하여 Lab, Scan-data, Esti_data의 관리를 총괄하는 매니저 클래스입니다. 각각의 테이블의 CRUD 기능을 구현하고, 사용의 편리함을 위해 부수적인 함수를 구현합니다.

다음 코드를 통해 Room 패키지를 활용합니다. Room패키지는 SQLite 데이터베이스를 직접 쿼리를 사용하지 않고 Dao를 통해 보다 간단하게 활용할 수 있게 도와줍니다. Dao는 각 테이블에 접근하기 위한 API를 정의합니다.

```
private val db = Room.databaseBuilder(
    context.applicationContext,
    AppDatabase::class.java,
    "inner-db"
).fallbackToDestructiveMigration(true).build()

private val labDao = db.labDao()
private val scanDataDao = db.scanDataDao()
private val estiDataDao = db.estiDataDao()
```

이후 lab, scandata, estidata에 대한 Entity CRUD를 구현합니다. Room에서 정의한 Entity 클래스에 대해 데이터를 만들고, 읽고, 수정하고, 삭제하는 기능이 이에 해당됩니다.

ESTI_DATA

위치 추정 결과 데이터를 저장하는 테이블로 활용되며, 실험 번호 및 알고리즘에 따라 결과를 쿼리할 수 있게 설계되었습니다. PK(Primary Key)로 `id`를 활용하며 FK(Foreign Key)로 `lid`를 활용합니다. `esti_pos`는 추정된 위치, `real_pos`는 실제 위치를 의미하는 Position객체이며, `error`는 추정 오차로 정확도를 나타내는 지표로 활용됩니다. `method`는 위치 추정에 사용되는 알고리즘을 의미하며, lid는 labID로 실험 식별자를 의미합니다.

id	timestamp	esti_pos	real_pos	error	lid	method
1	1751950266553	Position(x=1.9,y=2.7,z=2.7)	Position(x=1.0,y=1.5,z=2.0)	1.655	3	TrilaterationEstimator
2	1751950268632	Position(x=1.1,y=1.2,z=2.8)	Position(x=2.0,y=2.5,z=2.0)	1.772	3	TrilaterationEstimator
3	1751950302098	Position(x=0.3,y=4.8,z=1.3)	Position(x=1.0,y=3.5,z=2.0)	1.633	3	TrilaterationEstimator
...	...	...	...	...	...	...

LAB

PK로 `labID`를 활용합니다. `beaconPositions`는 x,y,z 좌표로 구성되어 각 비콘의 위치를 의미합니다. 가변적으로 비콘의 개수를 입력받기는 하지만, 기본값으로는 4개의 비콘의 위치를 입력하게 고정합니다. `createdAt`는 실험이 시작된 시간을 의미하며, 이를 통해 real_pos를 추출할 수 있습니다. `alias`는 실험에 대한 별명으로 기본값으로 "Unknown"이 들어 있으며, 추후 lab 명칭을 변경할 수 있도록 개발 예정입니다.

labID	beaconPositions	createdAt	alias
1	[{"x":0.0, "y":0.0, "z":0.0},{"x":6.0, "y":0.0, "z":0.0},...]	1751969319531	Unknown
2	[{"x":0.0, "y":0.0, "z":0.0},{"x":6.0, "y":0.0, "z":0.0},...]	1751969406809	Unknown
3	[{"x":0.0, "y":0.0, "z":0.0},{"x":6.0, "y":0.0, "z":0.0},...]	1751969486576	Unknown
...	...	...	...

SCAN_DATA

스캔하여 획득한 beacon_data를 저장하며, 고유 식별자(id)와 실험 식별자(lid)를 부여합니다. lid는 labID로 Lab의 labID와 연결되어 있습니다. 이로 인해 실험 번호에 따라 실험 결과를 쿼리할 수 있습니다. PK로 `id`, FK로 `lid`를 활용하며 `beaconID`로 비콘을 식별하고 `rssi`는 해당 비콘으로부터 수신된 rssi값을 저장합니다.

id	timestamp	beaconID	rssi	lid(labID)
1	1751950266098	S1	-51	4
2	1751950267037	S1	-55	4
3	1751950268185	S1	-58	4
...	...	...	...	...

EstimationPluginManager

`EstimationPluginManager` 는 플러그인 형식을 지원하며, 수신되는 rssi값을 이용하여 삼변측량을 통해 예상되는 위치를 계산하여 평가 및 데이터를 저장합니다.

다음 코드를 통해 플러그인 형식을 지원합니다. 여러 가지의 위치 추정 알고리즘을 플러그인 형식으로 등록 해제할 수 있습니다. 위치 추정 플러그인들을 저장할 가변 리스트를 선언하여 `register` 함수를 통해 플러그인을 등록하며, `unregister` 함수를 통해 플러그 인을 해제합니다.
`getAvailablePlugins` 함수를 통해 등록된 플러그인들의 이름을 리스트로 반환할 수 있습니다.

```
private val plugins = mutableListOf<EstimationPlugin>()

fun register(plugin: EstimationPlugin) {
    plugins.add(plugin)
}

fun unregister(plugin: EstimationPlugin) {
    plugins.remove(plugin)
}

fun getAvailablePlugins(): List<String> = plugins.map { it.name }
```

다음 함수를 통해 단일 윈도우 데이터에 대해 평가를 진행합니다. 기본값으로는 삼변측량(Trilateration)이 설정되어 있으며 플러그인으로 다른 알고리즘을 가져온다면 해당 알고리즘을 활용합니다.

```
fun calcEstiPos(input: TimeWindowBeaconRssi, pluginName: String =
"Trilateration"): Position? {
    val plugin = plugins.find { it.name == pluginName }
    return plugin?.calcEstiPos(input)
}
```

다음 함수를 통해 다중 윈도우 데이터에 대해 일괄 평가를 진행합니다. 앞선 함수와 비슷한 방식으로 동작하며, EstimationSummary 리스트로 반환한다는 차이점이 있습니다. 일괄 평가를 진행하면서 예측 실패 시 문제를 바로 인식하기 위해 throw구문을 통해 즉각적인 예외처리를 통해 호출한 쪽에서 반드시 올바른 플러그인 이름을 넣도록 강제합니다.

```
fun calc(
    input: List<TimeWindowBeaconRssi>,
    config: config,
    pluginName: String = "Trilateration"
): List<EstimationSummary> {
    val plugin = plugins.find { it.name == pluginName }
        ?: throw IllegalArgumentException("Plugin $pluginName not
registered")
    return plugin.calc(input, config)
}
```

다음 코드를 통해 평가 및 저장 기능을 수행합니다. `calc` 함수를 호출하여 위치를 계산하고 리스트를 넘겨 받아 EstiData에 매핑시켜 데이터베이스에 저장합니다.

```
fun calcAndSave(
    labId: Long,
    input: List<TimeWindowBeaconRssi>,
    config: config = defualtConfig,
    pluginName: String = "Trilateration"
) {
    val plugin = plugins.find { it.name == pluginName } ?: throw
IllegalArgumentException("Plugin $pluginName not registered")
    val results = plugin.calc(input, config)
    val estiDataList = results.map { summary ->
        EstiData(
            timestamp = summary.timestamp,
            esti_pos = summary.estiPos,
            real_pos = summary.realPos,
            error = summary.error,
            lid = labId,
            method = plugin.name
        )
    }

    dtManager.saveAllEstiDataSync(estiDataList)
}
```

Activity에서는 저장된 EstiData를 통해 lab 기록 확인, 결과 분석, 시각화 등의 작업이 가능합니다.