CLIPS

Communication & Localization with Indoor Positioning Systems

Università di Padova

Manuale Sviluppatore 1.00

leaf.gruppo@gmail.com

Versione	1.00
Data Redazione	2016-05-09
Redazione	Eduard Bicego
	Marco Zanella
Verifica	Federico Tavella
	Oscar Elia Conti
Approvazione	Davide Castello
Uso	Esterno
Distribuzione	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	Miriade S.p.A.

Diario delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.00	2016-05-09	Davide Castello	Responsabile di Progetto	Approvazione del documento
0.13	2016-05-09	Oscar Elia Conti	Verificatore	Verifica delle correzioni
0.12	2016-05-08	Eduard Bicego	Amministratore	Correzioni varie
0.11	2016-05-08	Oscar Elia Conti	Verificatore	Verifica del documento
0.10	2016-05-08	Marco Zanella	Progettista	Aggiunta funzionalità Gestione preferenze
0.09	2016-05-08	Eduard Bicego	Amministratore	Aggiunta funzionalità Localizzazione utente & Navigazione
0.08	2016-05-07	Eduard Bicego	Amministratore	Aggiunta sezione Funzionalità & Mappatura edificio
0.07	2016-05-07	Marco Zanella	Progettista	Aggiunta sezione Strumenti di sviluppo e Componenti Esterni
0.06	2016-04-18	Federico Tavella	Verificatore	Verifica del documento
0.05	2016-04-17	Eduard Bicego	Amministratore	Stesura sezione Introduzione e Per iniziare
0.04	2016-04-17	Marco Zanella	Progettista	Stesura sezione Persistenza dei dati

Versione	Data	Autore	Ruolo	Descrizione
0.03	2016-04-16	Marco Zanella	Progettista	Aggiunti contenuti sezione Strumenti di sviluppo
0.02	2016-04-16	Eduard Bicego	Amministratore	Ristrutturato documento
0.01	2016-04-15	Eduard Bicego	Amministratore	Aggiunta struttura documento

Indice

Elenco delle figure

1 Introduzione

1.1 Panoramica generale

CLIPS è un'applicazione che offre funzionalità riguardanti la navigazione guidata all'interno di edifici, attraverso l'utilizzo di dispositivi mobile Android e dei Beacon.

Tale applicazione è stata sviluppata per avere la possibilità di accedere alle informazioni per raggiungere una specifica area di interesse di un edificio offrendo indicazione testuali, sonore e visuali.

Questo manuale ha lo scopo di illustrare le parti che compongono tale applicazione e far comprendere al lettore il funzionamento di tali parti con fine ultimo la manutenzione e l'estensione.

1.2 Struttura del manuale

Il manuale è strutturato in diverse sezioni:

1. **Introduzione**
2. **Per iniziare:** spiega come scaricare il progetto e configurare l'IDE Android Studio per aprirlo;
3. **Strumenti di sviluppo:** presenta tutti gli strumenti e kit di sviluppo utilizzati per supportare lo sviluppo dell'applicazione e la sua creazione;
4. **Componenti esterne:** raccoglie tutte le librerie esterne utilizzate nel progetto presentando il loro utilizzo all'interno dell'applicazione e la versione utilizzata;
5. **Mappatura edificio:** raccoglie la soluzione proposta per realizzare un sistema di navigazione interna e delle istruzioni per estendere gli edifici supportati dall'applicazione;
6. **Architettura applicazione:** presenta una panoramica astratta per comprendere come siano strutturate le componenti all'interno dell'applicazione;
7. **Funzionalità:** presenta le componenti dell'applicazione suddivise per funzionalità, ogni funzionalità è descritta attraverso diversi passaggi, prima raccolti in sequenza nella sottosezione *Panoramica* e successivamente descritti approfonditamente in una sottosezione per ognuno;

8. **Persistenza dei dati:** presenta e descrive come i dati necessari per il funzionamento dell'applicazione siano salvati in un database locale e remoto. Inoltre descrive la struttura degli oggetti (*row* delle *table* dei database SQL) trasmessi via rete Internet in formato Json.

1.3 Glossario e documentazione

Allo scopo di rendere più semplice e chiara la comprensione del manuale viene allegato il glossario nel quale sono raccolte le spiegazioni di terminologia tecnica o ambigua, abbreviazioni ed acronimi. I termini evidenziati in rosso sono link alla voce in esso. Per agevolare l'uso e facilitare l'approfondimento del contenuto descritto nel manuale, ogni componente, classe o interfaccia interna (appartenente a codice sviluppato dal gruppo *Leaf*) o esterna (codice sviluppato da terzi) è evidenziata in blu e corrisponde ad un collegamento ipertestuale alla documentazione di tale componente.

1.4 Riferimenti utili

- Documentazione AltBeacon:
  https://altbeacon.github.io/android-beacon-library/javadoc/;
- Documentazione Clips:
  http://leafswe.github.io/clips/;
- Documentazione Dagger:
  http://google.github.io/dagger/api/2.0/;
- Documentazione Gson:
  http://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/index.html;
- Documentazione Java JDK
  https://docs.oracle.com/javase/8/docs/api/overview-summary.html;
- Documentazione Picasso:
  https://square.github.io/picasso/2.x/picasso/;
- Documentazione Android SDK:
  http://developer.android.com/reference/packages.html
- Documentazione Gradle:
  http://gradle.org/documentation/

2 Per iniziare

Nel caso in cui si voglia utilizzare o estendere il codice di CLIPS si consiglia di seguire i passi di seguito descritti. Le procedure descritte valgono sia per computer con sistema operativo **Microsoft Windows 10** sia con **Linux Ubuntu 16.04**.

2.1 IDE

È consigliato aprire ed eventualmente modificare il progetto con l'IDE Android Studio, ossia l'IDE utilizzato ufficialmente nello sviluppo. La versione con cui è stato sviluppato il progetto è la 1.5.1. Questa sezione farà riferimento a tale versione.

Android Studio è disponibile gratuitamente al seguente link:

http://developer.android.com/sdk/index.html

Nota: Il progetto non è stato provato su Android Studio con versioni successive o precedenti, non dovrebbe portare elevate differenze per le versioni successive.

2.2 Download del progetto

Per poter accedere al codice è necessario accedere al link:

https://github.com/LeafSWE/clips

Successivamente selezionare il branch **master** e cliccare **DOWNLOAD ZIP** e **SALVA**:

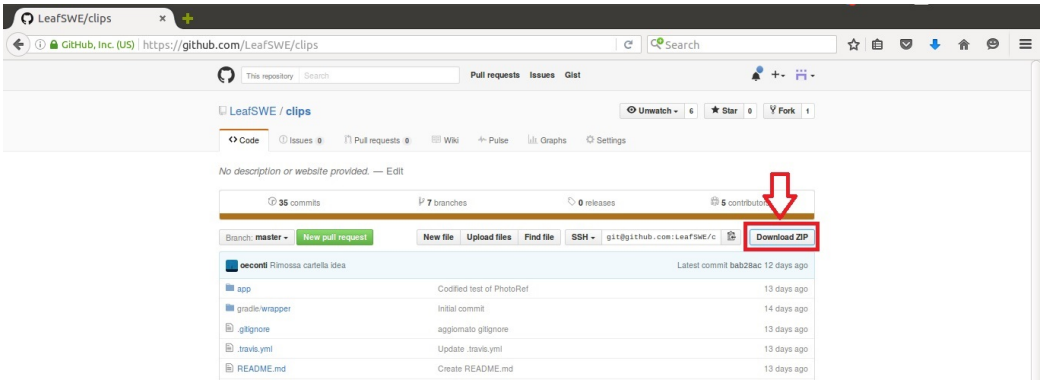


Figura 1: Download progetto da Github

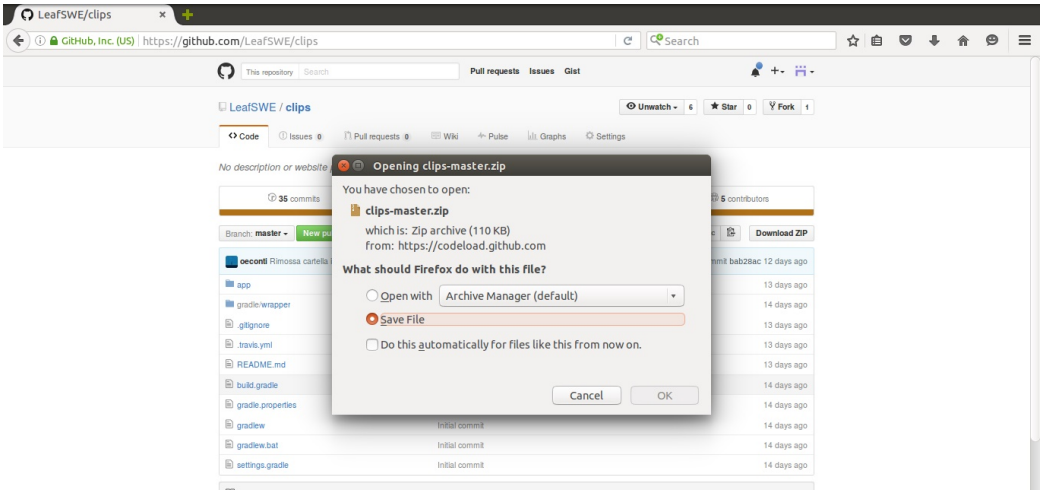


Figura 2: Download file progetto da Github

Scaricato il file `clips-master.zip` estrarlo con il tool per l'estrazione file preferito:

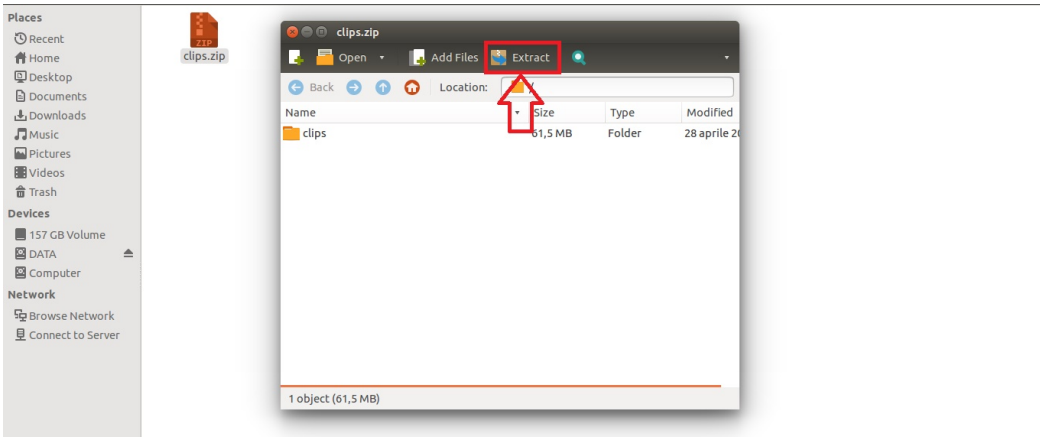


Figura 3: Estrazione file zip

2.3 Aprire il progetto con Android Studio

Aprire Android Studio e selezionare **Opening an existing Android Studio project**:

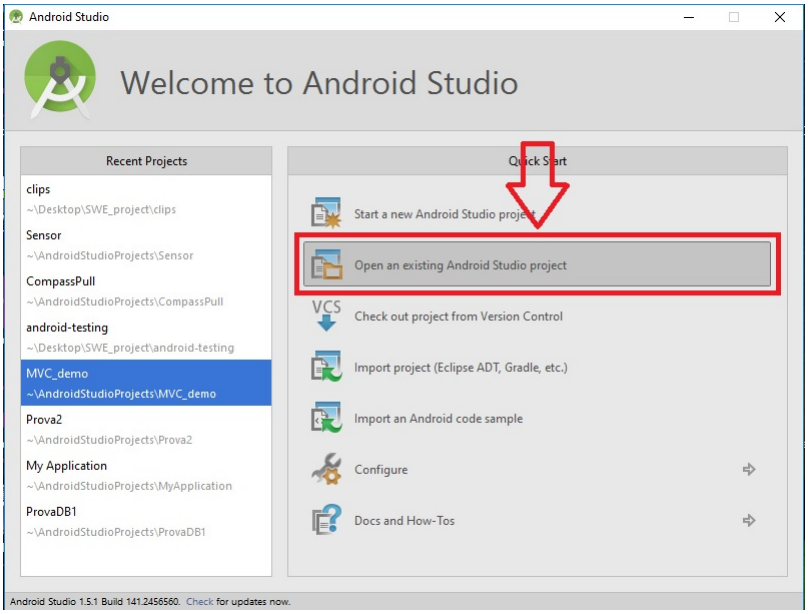


Figura 4: Aprire il progetto con Android Studio

Selezionare, seguendo il giusto path, la cartella del progetto `clips-master`. Attendere la **Build project info** di Gradle. Anche quando Android Studio è aperto attendere la conclusione del processo di Gradle:

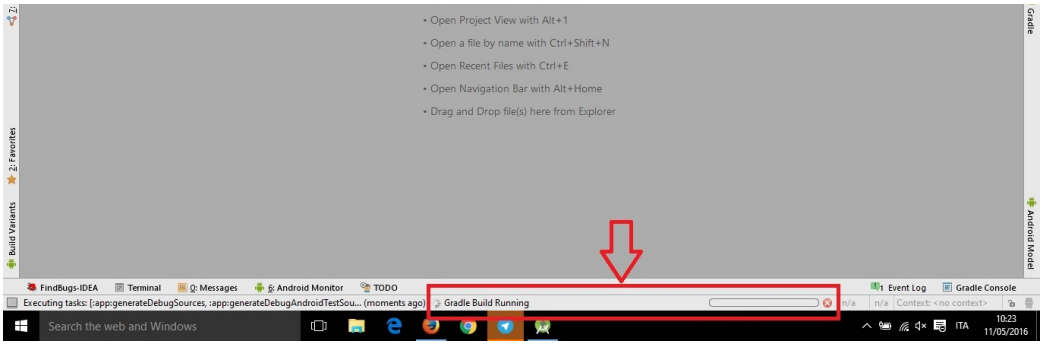


Figura 5: Build project info di Gradle

Nota: Nel caso in cui il processo di Gradle fallisca, come in figura 6, seguire le indicazioni nella sezione successiva 2.4

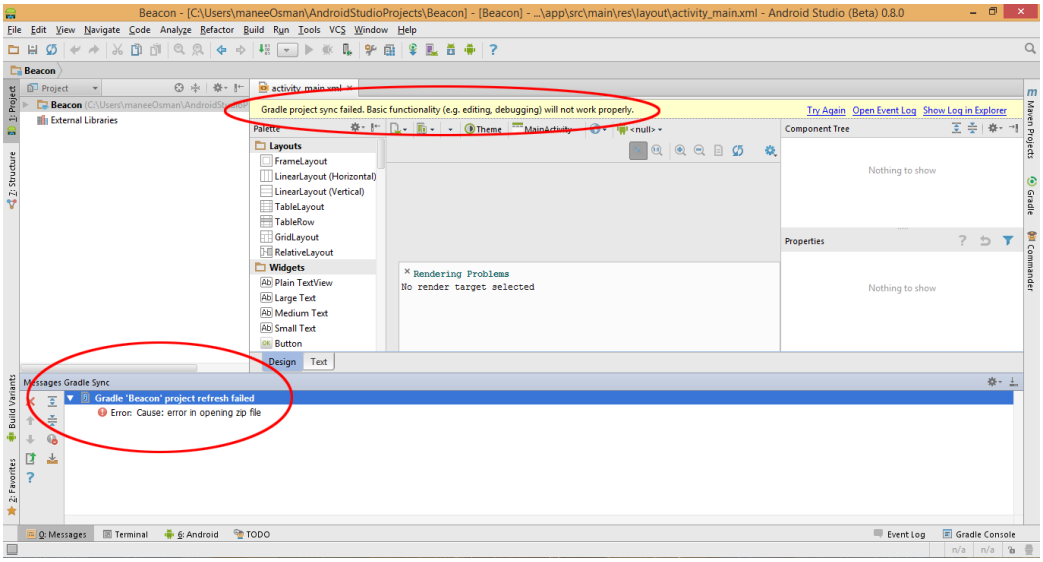


Figura 6: Errore build project info di Gradle

2.4 Configurazione Gradle in Android Studio

Per assicurarsi che la **build** di Gradle funzioni correttamente: in Android Studio cliccare **File** → **Settings** oppure premere **CTRL+ALT+S**:

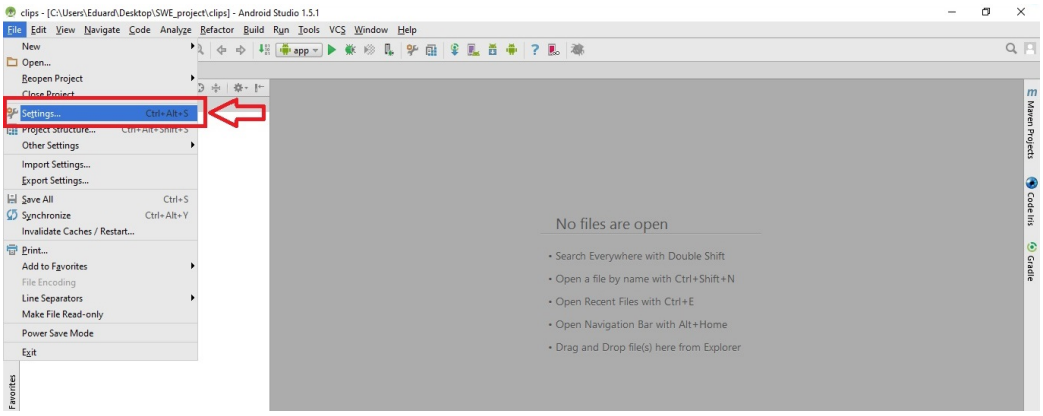


Figura 7: Aprire le impostazioni Android Studio

Nella nuova finestra aperta spostarsi su **Build, Execution, Deployment** → **Build Tools** → **Gradle**. Spuntare l'opzione **Use default gradle Wrapper (reccomended)** come in figura 8:

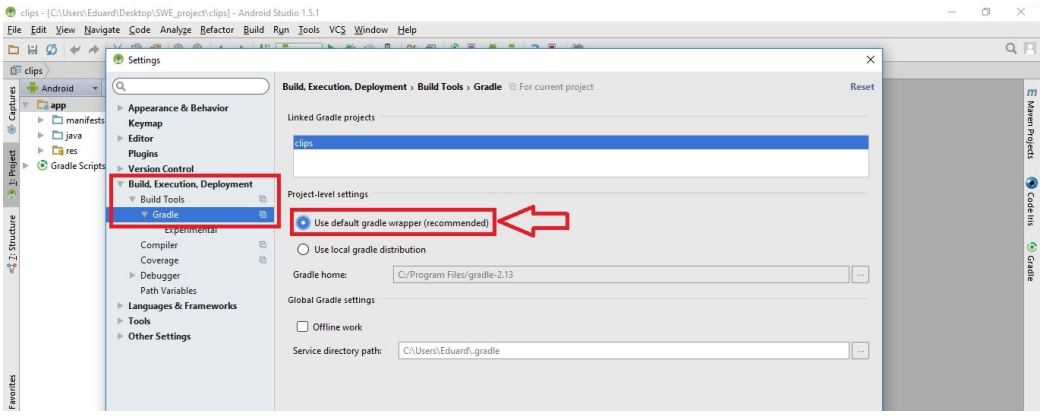


Figura 8: Impostare Gradle correttamente

3 Strumenti di sviluppo

Per la generazione del progetto sono necessari i kit di sviluppo Andrid SDK e Java JDK installati nel proprio computer, inoltre la generazione dell'applicativo con risoluzione delle dipendenze è automatica grazie all'uso di Gradle. Di seguito si indicano le versioni dei kit di sviluppo utilizzate nel progetto e la descrizione del file gradle e le istruzioni di come eseguirlo nell'ambiente di sviluppo Android Studio.

3.1 Android SDK

Framework di sviluppo per applicazioni Android.

- Versione SDK 24.4.1;
- Versione build tools 23.0.3;
- Versione target SDK 23;
- Versione minima SDK 19.

Reperibile a partire dal seguente link:

http://developer.android.com/sdk/older_releases.html

3.2 Java JDK

Insieme di strumenti di sviluppo per applicazioni Java.

- Versione oraclejdk8.

Reperibile al seguente link:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

3.3 Gradle

Sistema Open source per l'automazione dello sviluppo basato su Groovy.

- Vesione 2.1.0.

3.3.1 Contenuto file gradle

Nel file gradle vengono dichiarate tutte le dipendenze da risolvere per poter testare, compilare ed eseguire l'applicazione. Di seguito sono riportate le dipendenze dichiarate per l'applicativo.

- Dichiarazione dei plugin utilizzati

```
apply plugin: 'com.android.application'
apply plugin: 'com.neenbedankt.android-apt'
```

- Dichiarazione dipendenze per gli script di build

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.1.0'
    }
}
```

- Dichiarazione della configurazione di Android utilizzata

```
android {
    compileSdkVersion 23
    buildToolsVersion '23.0.3'

    defaultConfig {
        applicationId "com.leaf.clips"
        minSdkVersion 19
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
```

```
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile
              ('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    defaultConfig {
        testInstrumentationRunner "android.support.test.
          runner.AndroidJUnitRunner"
    }
}
```

- Dichiarazione delle dipendenze verso pacchetti esterni utilizzati
  - Dipendenze da componenti Android

```
compile 'com.android.support:appcompat-v7:23.3.0'
androidTestCompile 'com.android.support:
  appcompat-v7:23.3.0'
androidTestCompile 'com.android.support:
  support-annotations:23.3.0'

compile 'com.android.support:design:23.3.0'
androidTestCompile 'com.android.support:design:23.3.0'
compile 'com.android.support:support-v4:23.3.0'
androidTestCompile 'com.android.support:support-v4
    :23.3.0'
```

  - Dipendenze verso Dagger

```
compile 'com.google.dagger:dagger:2.0'
provided 'com.google.dagger:dagger-compiler:2.0'
provided 'org.glassfish:javax.annotation:10.0-b28'
```

  - Dipendenze verso JGraphT

```
compile 'org.jgrapht:jgrapht-core:0.9.1'
```

- Dipendenze verso Gson

```
compile 'com.google.code.gson:gson:2.6.2'
```

- Dipendenze verso Picasso

```
                compile 'com.squareup.picasso:picasso
    :2.5.2'
```

- Dipendenze verso JUnit(test)

```
testCompile 'junit:junit:4.12'
androidTestCompile 'com.android.support:
  support-annotations:23.1.1'
androidTestCompile 'com.android.support.test:rules:0.5'
androidTestCompile 'com.android.support.test:runner:0.5
    '
```

- Dipendenze verso Mockito(test)

```
testCompile 'org.mockito:mockito-core:1.10.19'
```

- Dipendenze verso Espresso(test)

```
androidTestCompile 'com.android.support.test.espresso:
  espresso-core:2.2.2'
androidTestCompile 'com.android.support.test.espresso:
  espresso-web:2.2.2'
androidTestCompile ('com.android.support.test.espresso:
  espresso-contrib:2.2.2') {
    exclude group: 'com.android.support',
      module: 'appcompat'
    exclude group: 'com.android.support',
      module: 'support-v4'
    exclude module: 'recyclerview-v7'
}
androidTestCompile 'com.android.support.test.espresso:
  espresso-idling-resource:2.2.2'
```

– Dipendenze verso FindBugs(analisi statica)

```
compile 'com.google.code.findbugs:jsr305:2.0.1'
```

4 Componenti esterne

4.1 SQLite

Libreria che implementa un DBMS SQL transazionale senza la necessità di un server. Viene utilizzata per salvare e gestire le mappe scaricate e installate nel dispositivo e il relativo contenuto.

- Versione utilizzata 3.9.2

4.2 AltBeacon

Libreria che permette ai sistemi operativi mobile di interfacciarsi ai Beacon, offrendo molteplici funzionalità. Viene utilizzata per permettere la comunicazione tra l'applicativo Android e i Beacon.

- Versione utilizzata 2.02

4.3 JGraphT

Libreria Java che fornisce funzionalità matematiche per modellare grafi. Viene utilizzata per la rappresentazione delle mappe e per il calcolo dei percorsi.

- Versione utilizzata 0.9.1

4.4 Gson

Libreria Java che fornisce funzionalità per la gestione di oggetti JSON. Tale libreria è utilizzata la gestione del download delle mappe da remoto.

- Versione utilizzata 2.6.2

4.5 Dagger

Libreria Android utilizzata per effettuare la dependency injection. Viene utilizzata per la creazione dei singleton.

- Versione utilizzata 2.0

4.6 Picasso

Libreria per la gestione delle immagini in remoto. Viene utilizzata per scaricare le immagini utilizzate durante la navigazione.

- Versione utilizzata 2.5.2

5 Mappatura edificio

Nella presente sezione si vuole fornire una descrizione della soluzione implementata per effettuare Navigazione indoor.

Per **mappatura edificio** si intende la costruzione teorica di un **grafo pesato** che rappresenta l'edificio in cui abilitare la funzionalità di navigazione offerta dall'applicazione.

Per un edificio quindi è necessario individuare i due elementi che compongono un grafo:

- Vertici;
- Archi.

5.1 Vertici: Region of Interest e Point of Interest

I **vertici** rappresentano le aree coperte dal segnale di un singolo Beacon, definite Region Of Interest (ROI). Ogni ROI, identificata da un unico Beacon, è definita come un insieme di possibili destinazioni da raggiungere all'interno dell'edificio da mappare. Tali destinazioni sono definite Point Of Interest (POI) e rappresentano nella realtà tutti i punti di interesse di un utente.

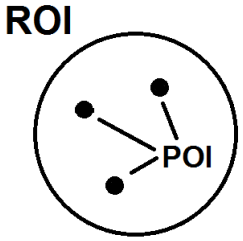


Figura 9: Visione teorica di ROI e POI contenuti in essa

Una ROI e quindi un Beacon, rappresenta più POI per soddisfare gli obiettivi di **massimizzare l'area da coprire** e **minimizzare il numero di beacon** da utilizzare.

Poiché è possibile che un POI abbia più punti d'accesso (si pensi ad una stanza con più porte d'accesso) è previsto che un singolo POI possa appartenere a più ROI. Il risultato è una relazione N a N tra ROI e POI. Nella figura 10 si illustra graficamente come un POI possa appartenere a più ROI, nel caso rappresentato il POI al centro appartiene sia alla ROI_1 sia alla ROI_2.

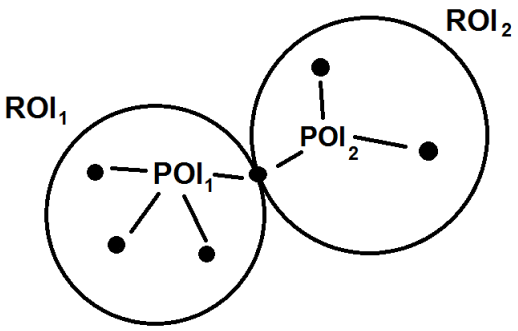


Figura 10: Visione teorica di un POI che appartiene a più ROI

5.2 Gli archi: Edge

Gli **archi** (Edge) invece rappresentano i collegamenti reali tra una ROI e un'altra ROI, per esempio un corridoio o delle scale. Essi quindi possono essere di varie tipologie e quindi pesati in modo diverso. Le tipologie fin'ora individuate sono:

- Default Edge;
- Stair Edge;
- Elevator Edge.

Ogni Edge contiene le informazioni necessarie per arrivare correttamente alla ROI di arrivo dalla ROI di partenza.

5.3 Percorso

Un POI è una possibile destinazione dell'utente ossia il punto d'arrivo di un possibile percorso mentre una ROI è l'area in cui l'utente si trova, grazie al rilevamento dei Beacon. Gli Edge e i ROI tra queste due entità rappresentano il percorso da percorrere.

Dato quindi un punto di partenza e uno di arrivo abbiamo la necessità di calcolare il percorso più corto per raggiungere il punto di arrivo. Poiché lavoriamo identificando gli spazi interni dell'edificio con elementi di un grafo, possiamo sfruttare tutta la conoscenza matematica su questi. Implementando l'algoritmo di Dijkstra otteniamo il percorso più corto da una ROI ad un'altra.

Poiché il punto di destinazione è un POI e questo può appartenere a diverse ROI nel calcolo del percorso bisogna identificare il percorso migliore tra la ROI di partenza e le ROI di cui il POI fa parte.

5.4 Mappare più piani

Se consideriamo ogni piano di un edificio come un grafo possiamo connettere tali grafi con nuovi Edge nei punti reali in cui questi piani si collegano, scale e ascensori. Otteniamo così da due grafi un grafo più grande.

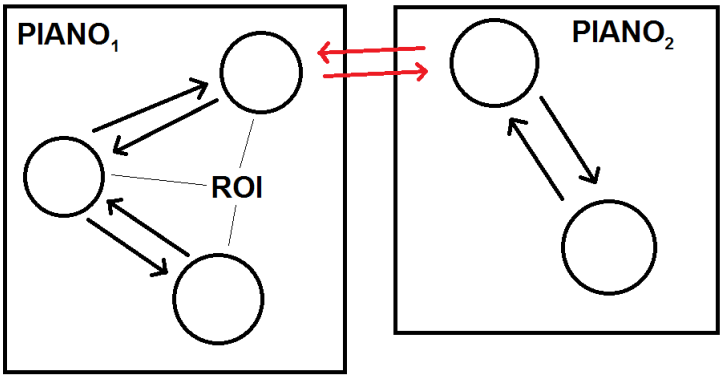


Figura 11: Visione teorica di un collegamento tra due grafi ossia piani di un edificio

5.5 Configurazione beacon

Per gestire questa soluzione teorica i beacon fanno da ROI del grafico ossia i vertici. La scelta della loro posizione è di fondamentale importanza per il corretto funzionamento della navigazione e della user experience.

I dati trasmessi dal Beacon sono il Major e il Minor, ovvero due stringhe composte da cinque caratteri numerici. Nella soluzione presentata sono così utilizzati:

Major: identificativo dell'edificio supportato dall'applicazione;

Minor: le prime due cifre identificano il piano mentre le ultime tre la ROI.

Di seguito si elencano quali sono le pratiche da seguire individuate dal team *Leaf* per ottenere un risultato ottimale:

- gli spazi interni dell'edificio devono essere il più possibile coperti dal segnale dei Beacon evitando il più possibile sovrapposizioni tra questi;
- ogni ROI e quindi ogni Beacon si consiglia di posizionarlo in tutti gli incroci di un edificio e nei luoghi di accesso di ogni stanza o punto di interesse (POI);

- il collegamento reale (Edge teoricamente) tra due ROI deve essere il più possibile una linea retta. L'applicazione infatti calcola alcune indicazioni da dare sulla base della coordinata magnetica/orientamento da una ROI ad un altro;

5.6 Procedura per la mappatura di un edificio

Nel caso si volesse che l'applicazione CLIPS supporti un altro edificio seguire questi passaggi:

1. procurarsi una planimetria dell'edificio;
2. identificare tutti i POI dell'edificio di ogni piano;
3. raggruppare i POI vicini in ROI garantendo che:
   - tutto l'edificio sia coperto dal segnale dei beacon;
   - non ci sia eccessiva distanza tra una ROI e un'altra;
4. collegare ogni ROI alle ROI raggiungibili da essa senza passare per altre ROI e identificando la tipologia di Edge (normale, scale, ascensori...). **NOTA:** se da un ROI posso raggiungere un'altra ROI e viceversa dovrò necessariamente documentare due Edge distinti.
5. per ogni Edge identificato rilevare le informazioni necessarie per descriverlo in toto:
   - distanza;
   - coordinata magnetica;
   - azione che l'utente deve seguire;
   - descrizione dettagliata;
   - ROI di partenza;
   - ROI di fine;

   Tali informazioni saranno fondamentali perché l'applicazione offra il maggior numero di informazioni all'utente;
6. assegnare ai Beacon da utilizzare un Major univoco che identifica l'edificio che si sta mappando;
7. assegnare ad ogni Beacon un Minor in cui le prime due cifre identificano il numero del piano e le ultime tre identificano la ROI in quel piano;
8. inserire nel server delle mappe tali informazioni. Per ulteriori informazioni su quest'ultimo punto si rimanda alla sezione 8.

5.7 Area sviluppatore

Nell'applicazione è stata inclusa un'**area sviluppatore** accessibile solo tramite password. Tale area ha lo scopo di facilitare la raccolta di dati e mostrarli tramite l'interfaccia grafica dell'applicazione. È stato fatto ciò per consentire di effettuare più facilmente e velocemente test sul campo durante la **mappatura di un edificio**. Di seguito si elencano i passaggi necessari per accedervi e le funzionalità offerte:

1. dal menu dell'applicazione selezionare **Area sviluppatore**;
2. inserire il codice sviluppatore sottostante:

   `miriade`

3. effettuare l'operazione desiderata:
   - visualizzazione dettagliata dei log salvati;
   - avvio di un nuovo log;
   - rimozione di un log salvato.

Per approfondire le possibilità offerte da tale funzionalità si rimanda alla consultazione del *Manuale Utente*.

6 Architettura applicazione

6.1 Pattern architetturale MVP

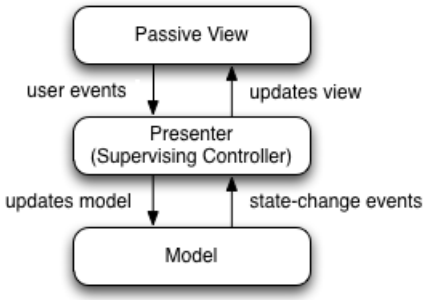


Figura 12: Struttura del pattern MVP

Model-View-Presenter (MVP) è un pattern architetturale derivato dal MVC (Model-View-Controller), utilizzato per dividere il codice in funzionalità distinte.

Il suo principale ambito di utilizzo è nelle applicazioni in cui sia necessario separare la logica dei componenti visivi della GUI dai componenti stessi consentendo così l'uso di diversi linguaggi per le due cose. Per esempio descrivere le componenti in XML e definirne la logica in Java.

6.1.1 Componenti

Il pattern è basato sul principio di disaccoppiamento di tre oggetti distinti, riducendo in questo modo le dipendenze reciproche; inoltre permette di fornire una maggiore modularità, manutenibilità e robustezza al Software.

Model Il Model rappresenta il cuore dell'applicazione: esso definisce il modello dei dati definendo gli oggetti secondo la logica di utilizzo dell'applicazione, ossia la sua business logic. Inoltre, indica le possibili operazioni che si possono effettuare sui dati.

View Nel pattern MVP, la View è un componente passivo che si occupa essenzialmente di notificare al Presenter eventuali interazioni con l'eventuale utente. È compito del Presenter raccogliere questi segnali ed elaborarli.

Presenter Il Presenter è l'intermediario tra il Model e la View. Si occupa di implementare l'insieme di operazioni eseguibili sul modello dei dati attraverso una particolare vista, ossia l'application logic. Solitamente ad ogni componente della View corrisponde un componente del Presenter.

6.1.2 Vantaggi

Il pattern MVP per l'architettura dell'applicazione CLIPS è stato scelto perché:

- Consente di separare completamente l'interfaccia grafica dalla logica e quindi di utilizzare il linguaggio XML per descrivere l'interfaccia dell'applicazione;
- La completa separazione di View e Presenter consente maggiore flessibilità nella manutenzione e nell'eventuale modifica dell'interfaccia grafica;
- È considerato dalla comunità Android il pattern di riferimento per un'applicazione Android;
- Mantiene tutti i vantaggi offerti dal pattern Model View Controller della separazione logica dei componenti.

6.2 Gestione dipendenze ed estensibilità

Durante la progettazione dell'applicativo oltre a seguire il pattern Model View Presenter si è cercato di mantenere divisi i contratti delle classi e l'implementazione concreta attraverso l'uso di **interfacce** e delle sue implementazioni. Inoltre si è utilizzata la **Dependency Injection** che permette un completo disaccoppiamento tra le componenti del Model e del Presenter e garantisce che alcune componenti del Model siano Singleton.

6.3 Dependency Injection

6.3.1 Dichiarazione delle dipendenze

Le dipendenze devono essere dichiarate annotando con `@Inject` i campi dati o il costruttore di cui Dagger deve costruire una istanza. In questo modo Dagger può assegnare, per esempio, ad ogni interfaccia l'implementazione corretta. Le classe in cui viene utilizzata tale annotazione sono:

- HomeActivity;
- DeveloperUnlockerActivity;

- LogInformationActivity;
- MainDeveloperActivity;
- MainDeveloperPresenter;
- MyApplication;
- NavigationActivity;
- NearbyPoiActivity;
- PoiCategoryActivity.

6.3.2 Module

I moduli vengono dichiarando annotando una classe con @Module. Tali classi sono necessarie per risolvere le dipendenze dichiarate. In queste classi devono essere dichiarati metodi annotati con @Provides. Questi servono per dichiarare a Dagger le azioni da compiere per risolvere una certa dipendenza. Un metodo può essere annotato con @Singleton. In questo caso verrà restituita sempre la stessa istanza per ogni dipendenza dichiarata verso quel metodo. La classe AppModule risolve:

- dipendenze verso Context, il metodo è annotato anche @Singleton;
- dipendenze verso Application restituendo una istanza di MyApplication, il metodo è annotato @Singleton.

La classe DatabaseModule risolve:

- dipendenze verso SQLiteDaoFactory, il metodo è annotato @Singleton;
- dipendenze verso RemoteDaoFactory, il metodo è annotato @Singleton;
- dipendenze verso DatabaseAccess restituendo un'istanza di BuildingAccess, il metodo è annotato @Singleton.

La classe InfoModule risolve:

- dipendenze verso InformationManager restituendo un'istanza di InformationManagerImp, il metodo è annotato come @Singleton.

La classe SettingModule risolve:

- dipendenze verso Setting restituendo un'istanza di SettingImp, il metodo è annotato come @Singleton.

6.3.3 Component

I component sono interfacce che Dagger autonomamente si occupa di implementare. Queste devono essere annotate con @Component e fanno da collegamento tra i moduli e le classi in cui devono essere iniettate le dipendenze. In tali interfacce devono essere dichiarate dei metodi con la seguente firma:

```
void inject(Type type);
```

Tali metodi devono richiedere come argomento un oggetto della classe che ha al suo interno annotazioni @Inject.

L'unica interfaccia annotata con @Component è InfoComponent. Tale interfaccia permette di risolvere le dipendenze di:

- HomeActivity;
- DeveloperUnlockerActivity;
- LogInformationActivity;
- MainDeveloperActivity;
- MainDeveloperPresenter;
- MyApplication;
- NavigationActivity;
- NearbyPoiActivity;
- PoiCategoryActivity.

6.3.4 Utilizzo dei metodi inject

Per poter effettivamente risolvere le dipendenze è necessario recuperare una istanza dell'implementazione dell'interfaccia annotata come @Component. Dagger a queste implementazioni dà come nome Dagger seguito dal nome dato al componente. Per recuperare tale istanza è necessario invocare il metodo statico builder() sulla classe creata da Dagger. Questo ritorna un Builder per la classe creata da Dagger. A questo Builder è necessario aggiungere i moduli in cui è dichiarato come risolvere le dipendenze delle classi richieste come argomenti ai metodi *inject()* dichiarati nell'interfaccia annotata come @Component. Per fare questo è possibile invocare i metodi che hanno nome

uguale alla classe annotata come @Module ma con nome che inizia con lettera minuscola. Quando sono stati aggiunti tutti i moduli è possibile invocare il metodo build() per ottenere l'istanza del componente.

Una volta creato un componente è possibile invocare il metodo inject() passando come argomento l'istanza in cui "iniettare" le dipendeze. In questo modo l'istanza di oggetto passata al metodo inject() avrà le dipendenze soddisfatte.

L'istanza di Dagger che implementa l'interfaccia InfoComponent e l'aggiunta dei moduli viene fatto in MyApplication, mentre i vari metodi inject() vengono invocati tutti nel metodo onCreate(), poiché le classi in cui è usata la dependency injection sono tutte Activity oppure Application nel caso di MyApplication.

7 Funzionalità

Nella presente sezione vengono spiegate nel dettaglio le componenti dell'applicazione e il loro scopo, presentate per funzionalità offerte dall'applicazione. Ogni funzionalità viene prima descritta in una sottosezione *"Panoramica"* e successivamente in sottosezioni che approfondiscono gli aspetti che compongono la funzionalità.

Ogni aspetto approfondito è accompagnato da una lista delle componenti interne di CLIPS e delle principali componenti esterne. Ogni nome di interfaccia e classe rappresenta un link ipertestuale alla documentazione ufficiale, comprese le componenti interne.

Le funzionalità offerte dall'applicazione e descritte in seguito sono:

- Localizzazione Utente 7.1;
- Gestione preferenze 7.2;
- Navigazione 7.3;
- Area sviluppatore 7.4.

7.1 Localizzazione utente

7.1.1 Panoramica

L'applicazione offre la funzionalità di localizzare l'utente all'interno di un edifico in cui risiedono i Beacon riconosciuti dall'applicazione e di mostrare semplici informazioni sull'edifico.

La localizzazione utente avviene seguendo le seguenti fasi:

1. l'utente avvia l'applicazione;
2. l'applicazione avvia il Monitoring per poter rilevare i Beacon circostanti;
3. l'applicazione avvia il Ranging e reperisce l'identificativo major dei Beacon;
4. l'applicazione si accerta che i Beacon rilevati siano pertinenti all'applicazione attraverso un confronto tra major rilevato e major dei Beacon nel database locale;
5. se il Beacon è riconosciuto ed esiste un match nel database locale:
   - vengono costruiti concretamente gli oggetti dai dati nel database locale e raccolti nell'oggetto BuildingMap;
   - vengono mostrate all'utente informazioni sull'edificio in cui si trova.

7.1.2 Interfaccia grafica

HomeViewImp implementa la vista principale che viene visualizzata subito dopo l'avvio dell'applicazione dopo la generazione della splash page da MainActivity la classe di avvio dell'applicazione. HomeView è legata alla componente del presenter HomeActivity e può essere di due tipologie:

- vuota, ossia priva di informazioni dell'edificio. Ciò significa che il device non ha ancora rilevato un Beacon riconosciuto dall'applicazione.
- mostra le informazioni dell'edificio in cui è l'utente. Ciò significa che l'applicazione ha rilevato almeno un Beacon riconosciuto dall'applicazione.

Componenti interne

- Package:

  com.leaf.clips.view;

- Interfacce e classi:

  HomeView, HomeViewImp;

Componenti esterne

- Interfacce e classi SDK:

  Activity, Toolbar, TextView, DrawerLayout, TextView, SearchView, FloatingActionButton, ListView, ActionBarDrawerToggle.

Nella figura 13 si indicano i seguenti widget offerti dal kit Android SDK:

1. SearchView;
2. TextView;
3. ListView;
4. FloatingActionButton.

Figura 13: HomeView - Screen con informazioni edificio

7.1.3 Presenter

Il compito del presenter per questa funzionalità è affidato all'oggetto HomeActivity che comunica con la vista HomeViewImp attraverso l'interfaccia HomeView. HomeActivity comunica con le componenti del model InformationManager attraverso la dependency injection.

Componenti interne

- Package:

  com.leaf.clips.view;

  com.leaf.clips.presenter;

- Interfacce e classi:

  HomeActivity, HomeView;

Componenti esterne

- Interfacce e classi SDK:

  Activity, InformationManager.

7.1.4 Rilevamento beacon

La classe BeaconManagerAdapter estende un bind Service (vedi appendice) ed ha il compito di effettuare il Ranging e il Monitoring dei Beacon circostanti. Il Monitoring è l'operazione svolta in background per riconoscere i Beacon circostanti senza eccessiva precisione mentre il Ranging è l'operazione che segue. Con il Ranging infatti i beacon vengono rilevati con più dettaglio e precisione con conseguente consumo di maggiore risorse, soprattutto energetiche. La comunicazione dei Beacon rilevati dal com.leaf.clips.model avviene attraverso l'uso degli oggetti MyBeacon inviati tramite Intent per cui serializzati. Gli Intent vengono recuperati tramite BroadcastReceiver implementato in altre classi.

Componenti interne

- Package:

  com.leaf.clips.model;

  com.leaf.clips.model.beacon;

- Interfacce e classi:

  BeaconManagerAdapter, MyBeacon, MyBeaconImp, MyDistanceCalculator, BeaconManagerAdapter.LocalBinder, BeaconRanger;

Componenti esterne

- Interfacce e classi AltBeacon:

  BeaconManager, BootstrapNotifier, BeaconConsumer, RangeNotifier, Region, BeaconParser, DistanceCalculator, Beacon;

- Interfacce e classi JDK:

  PriorityQueue;

- Interfacce e classi SDK:

  Intent, LocalBroadcastManager, Service, Binder, LocalBroadcastManager, IBinder.

7.1.5 Costruzione BuildingMap

L'oggetto InformationManagerImp comunica a DatabaseService la necessità di costruire l'oggetto BuildingMap. DatabaseService costruisce tale oggetto incaricando PointOfInterestService, RegionOfInterestService, EdgeService, PhotoService di costruire le componenti di BuildingMapossia:

- PointOfInterest;
- RegionOfInterest;
- EnrichedEdge;

Tali componenti sono prelevate e costruite facendo uso di semplici oggetti contenitori di dati grazie alle classi con suffisso *Dao*. Questi oggetti semplici sono descritti da un nome riferito all'elemento all'interno di una tabella del database locale e con il suffisso *Table*. Le operazioni eseguite per costruire BuildingMap si distinguono in due casi:

- I dati necessari per la costruzione di BuildingMap sono nel database locale. A questo punto se la versione dei dati in locale corrispondono alla versione in remoto l'InformationManagerImp incarica DatabaseService di costruire l'oggetto BuildingMap come spiegato in precedenza, altrimenti si effettuano le operazioni descritte nel caso successivo;
- I dati necessari per la costruzione di BuildingMap devono essere scaricati dal database remoto. A questo punto, dopo aver fatto richiesta all'utente di poter effettuare il download dei dati, DatabaseService reperisce le informazioni dal database remoto converte i dati dal formato Json trasmetto ad oggetti attraverso le classi con prefisso *Remote*, inserisce tali oggetti nel database locale attraverso le classi con suffisso *Dao* e infine costruisce l'oggetto BuildingMap desiderato.

Componenti interne

- Package:

  com.leaf.clips.model

  com.leaf.clips.model.dataaccess

  com.leaf.clips.model.dataaccess.service

  com.leaf.clips.model.dataaccess.dao

  com.leaf.clips.model.navigator.graph

com.leaf.clips.model.navigator.graph.edge

com.leaf.clips.model.navigator.graph.vertex

com.leaf.clips.model.navigator.graph.area

- Interfacce e classi:

  BuildingMap, BuildingMapImp, BuildingInformation, PointOfInterest, RegionOfInterest, EnrichedEdge, DatabaseService, BuildingService, EdgeService, PhotoService, PointOfInterestService, RegionOfInterestService, ServiceHelper, CursorConvert, DaoFactoryHelper, RemoteDaoFactory, SQLiteDaoFactory;

  con suffisso *Dao*: SQLDao, BuildingDao, CategoryDao, EdgeDao, EdgeTypeDao, PhotoDao, PointOfInterestDao, RegionOfInterestDao, RoiPoiDao, RemoteBuildingDao, RemoteCategoryDao, RemoteEdgeDao, RemoteEdgeTypeDao, RemotePhotoDao, RemotePointOfInterestDao, RemoteRegionOfInterest, RemoteRoiPoiDao;

  con prefisso *SQL* e suffisso *Dao*: SQLiteBuildingDao, SQLiteCategoryDao, SQLiteEdgeDao, SQLitePhotoDao, SQLiteEdgeTypeDao, SQLitePointOfInterestDao, SQLiteRegionOfInterestDao, SQLiteRoiPoiDao;

  con suffisso *Table* o *Contract*: BuildingTable, BuildingCommand, CategoryTable, CategoryContract, EdgeTable, EdgeContract, EdgeTypeTable, EdgeTypeContract, PhotoTable, PhotoContract, PointOfInterestTable, PointOFInterestContract, RegionOfInteresttable, RegionOfInterestContract, RoiPoiTable, RoiPoiContract;

Componenti esterne

- Interfacce e classi Gson:

  Gson, GsonBuilder, JsonObject, JsonArray, JsonParser;

- Interfacce e classi SDK:

  SQLiteOpenHelper, SQLiteDatabase, Cursor, ContentValue, BaseColumns.

7.2 Gestione delle preferenze

7.2.1 Panoramica

La funzionalità di gestione delle preferenze utente è resa disponibile dalle componenti dei Package:

- com.leaf.clips.model.usersetting.

Tali funzionalità permettono di impostare le preferenze riguardanti il percorso che viene presentato, la fruizione delle informazioni di navigazione e permette inoltre di sbloccare le funzionalità sviluppatore. Un utente può quindi:

1. impostare la preferenza verso scale o ascensori, utilizzata dall'applicazione nel calcolo del percorso da presentare;
2. impostare la preferenza verso indicazioni sonore, utilizzata dall'applicazione nella presentazione delle informazioni di navigazione;
3. inserire un codice sviluppatore per sbloccare le funzionalità sviluppatore.

7.2.2 Interfaccia grafica

Interfaccia grafica non ancora implementata.

7.2.3 Presenter

Presenter non ancora implementato.

7.2.4 Gestione delle preferenze di percorso

La classe SettingImp permette di impostare le preferenze di percorso utilizzando il metodo setPathPreference() e passando come argomento l'oggetto di tipo PathPreference che rappresenta la preferenza voluta. SettingImp provvederà a rendere tale preferenza persistente utilizzando le classi SharedPreferences e SharedPreference.Editor.

Componenti interne

- Package:

  com.leaf.clips.model.usersetting;

- Interfacce e classi:

  Setting, SettingImp, PathPreference;

Componenti esterne

- Interfacce e classi SDK:

  SharedPreferences, SharedPreference.Editor.

7.2.5 Gestione delle preferenze di fruizione delle informazioni

La classe SettingImp permette di impostare le preferenze di fruizione delle informazioni di navigazione utilizzando il metodo setInstructionPreference() e passando come argomento l'oggetto di tipo InstructionPreference che rappresenta la preferenza voluta. SettingImp provvederà a rendere tale preferenza persistente utilizzando le classi SharedPreferences e SharedPreference.Editor.

Componenti interne

- Package:

  com.leaf.clips.model.usersetting;

- Interfacce e classi:

  Setting, SettingImp, PathPreference;

Componenti esterne

- Interfacce e classi SDK:

  SharedPreferences, SharedPreference.Editor.

7.2.6 Sblocco funzionalità sviluppatore

La classe SettingImp permette di sbloccare le funzionalità sviluppatore utilizzando il metodo unlockDeveloper() e passando come argomento una stringa rappresentate il codice sviluppatore. SettingImp validerà tale codice utilizzando la classe DeveloperCodeManager utilizzando il metodo isValid() e salvando in modo persistente tale sblocco utilizzando le classi SharedPreferences e SharedPreference.Editor.

Componenti interne

- Package:

  com.leaf.clips.model.usersetting;

- Interfacce e classi:

  Setting, SettingImp, PathPreference;

Componenti esterne

- Interfacce e classi SDK:

  SharedPreferences, SharedPreference.Editor.

7.3 Navigazione

7.3.1 Panoramica

La funzionalità di navigazione è resa disponibile dalle componenti dei package:

- com.leaf.clips.model.navigator;
- com.leaf.clips.model.beacon;
- com.leaf.clips.model.compass;
- com.leaf.clips.model.dataaccess;
- com.leaf.clips.model.usersetting.

Esse permettono di guidare l'utente all'interno di un edificio. La navigazione è gestita attraverso queste fasi:

1. l'utente interagendo con l'interfaccia grafica avvia la navigazione;
2. la business logic dell'applicazione costruisce un grafo;
3. alle componenti di com.leaf.clips.model.navigator viene passato il grafo;
4. viene calcolato il percorso utilizzando la libreria esterna **JgraphT**;
5. vengono restituite le informazioni necessarie per guidare l'utente verso la destinazione da lui scelta;
6. l'interfaccia mostra all'utente le informazioni.

7.3.2 Interfaccia grafica

La navigazione comprende diverse viste. Tutto parte dalla vista della HomeView da cui è possibile selezionare la categoria della possibile destinazione, formalmente descritta come *point of interest* (POI). Una volta selezionata la categoria l'utente passa alla vista PoiCategoryView che mostra la lista di possibili destinazioni della categoria selezionata. Una volta selezionata la destinazione all'utente è mostrata una nuova vista che presenta tutti i passaggi necessari per raggiungerla. NavigationView e PoiCategoryView sono rispettivamente collegate alle classi che rappresentano il presenter NavigationActivity e PoiCategoryActivity.

Componenti interne

- Package:

  com.leaf.clips.view;

- Interfacce e classi:

  PoiCategoryView, PoiCategoryView, NavigationView, NavigationViewImp;

Componenti esterne

- Interfacce e classi SDK:

  AdapterView, ArrayAdapter, ListView;

Nella figura 14 si indicano i seguenti widget offerti dal kit Android SDK :

1. Toolbar;
2. ListView.

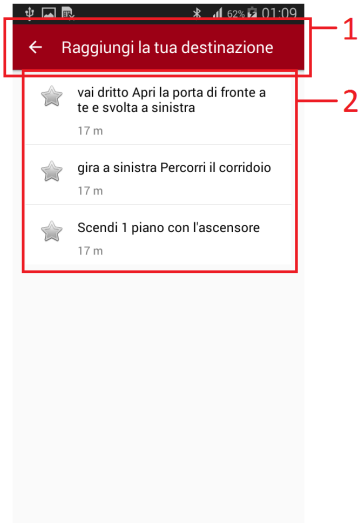


Figura 14: Navigazione - Lista istruzioni percorso

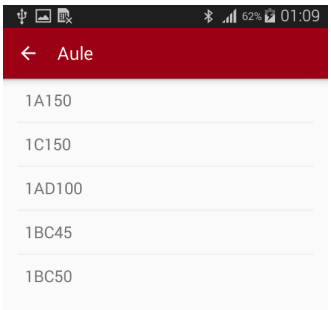


Figura 15: Navigazione - Scelta destinazione nella categoria *Aule*

7.3.3 Presenter

Il compito del presenter per questa funzionalità è affidato all'oggetto PoiCategoryActivity che comunica con la vista PoiCategoryView attraverso l'interfaccia HomeView e NavigationActivity che comunica con la vista NavigationViewImp attraverso l'interfaccia NavigationView. PoiCategoryActivity comunica con le componenti del model InformationManager attraverso la dependency injection mentre NavigationActivity comunica con le componenti del model InformationManager e NavigationManager sempre attraverso l'uso della dependency injection.

Componenti interne

- Package:

  com.leaf.clips.view;

  com.leaf.clips.presenter;

- Interfacce e classi:

  NavigationActivity, PoiCategoryActivity, NavigationView, PoiCategoryView, NavigationAdapter;

Componenti esterne

- Interfacce e classi SDK:

  Activity, AppCompatActivity;

7.3.4 Calcolo percorso

La classe NavigationManagerImp richiede il calcolo del percorso a Navigator implementata dalla classe NavigatorImp. Affinché NavigatorImp funzioni devono essere effettuate alcune operazioni nel giusto ordine qui riportato (**nota:** non vengono riportati i parametri, per essi si faccia riferimento alla javadoc):

1. eseguire la chiamata del metodo setGraph() settare il grafo dell'edificio rilevato;

2. eseguire la chiamata del metodo calculatePath() per far sì che NavigatorImp calcoli il percorso dal punto in cui si è (startROI) alla destinazione scelta (endPOI)

3. utilizzare uno o entrambi i metodi che ritornano informazioni incapsulate in ProcessedInformation:
    - getAllInstruction () per recuperare immediatamente tutte le informazioni per raggiungere la destinazione;
    - toNextRegion() per recuperare solo l'informazione del prossimo punto da raggiungere, guidando l'utente con un passaggio alla volta.

Componenti interne

- Package:

  com.leaf.clips.model.navigator;

  com.leaf.clips.model.naviagator.algorithm;

  com.leaf.clips.model.navigator.graph;

  com.leaf.clips.model.navigator.graph.area;

- Interfacce e classi:

  Navigator, NavigatorImp, ProcessedInformation, ProcessedInformationImp, PathFinder, DijkstraPathFinder, Compass, MapGraph, EnrichedEdge, PointOfInterest, RegionOfInterest;

Componenti esterne

- Interfacce e classi JGraphT:

  DijkstraShorterPath, SimpleDirectedWeightedGraph.

7.3.5 Bussola

La classe Compass permette all'applicazione di ricevere dati dai sensori hardware del device gestiti grazie alla classe Sensor. Compass rende disponibili i metodi per registrare i listener ai sensori e per disattivarli. Poiché i sensori comunicano attraverso eventi tramite interfaccia SensorEventListener i dati recuperati della bussola non corrispondono all'istante in cui sono recuperati.

Componenti interne

- Package:

  com.leaf.clips.model.compass;

- Interfacce e classi:

  Compass;

Componenti esterne

- Interfacce e classi SDK:

  SensorManager, Sensor, SensorEventListener.

7.3.6 Eccezioni e gestione

Componenti interne

- Package:

  com.leaf.clips.model;

  com.leaf.clips.model.navigator;

- Interfacce e classi:

  NavigationManagerImp, Navigator, NavigatorImp, NavigationExceptions, NoGraphSetException, PathException, NoNavigationInformationException;

Componenti esterne

- Interfacce e classi JDK:

  Exception.

Nel package com.leaf.clips.model.navigator vengono lanciate delle eccezioni per far sì che chiunque le utilizzi rispetti un particolare ordine. Tale ordine coinvolge le seguenti operazioni:

- Set del grafo in NavigatorImp;
- Calcolo del percorso attraverso NavigatorImp;
- Esecuzione della navigazione.

Il non rispetto di tale ordine può sollevare diversi tipi di eccezioni:

- NoGraphSetException se il grafo non è stato settato e si richiede il calcolo del percorso o l'esecuzione della navigazione;
- NoNavigationInformationException se si avvia la navigazione ma non si è calcolato il percorso precedentemente.

Mentre se il rilevamento dei Beacon non corrisponde con quanto previsto, a significare che l'utente sta sbagliando percorso, viene lanciata l'eccezione:

- PathException.

Nell'applicazione tali operazioni sono gestite da NavigationManagerImp.

7.4 Area sviluppatore

7.4.1 Panoramica

Selezionata dal menu dell'applicazione, l'opzione *Area sviluppatore* offre la possibilità di raccogliere e salvare un file Log che contiene tutti i dati dei Beacon rilevati finché il Log non è arrestato.

MainDeveloperPresenter è la classe che si occupa di controllare l'accesso a tale area:

- se l'utente ha già inserito correttamente almeno una volta la password gli viene mostrata la lista dei Log salvati;
- se l'utente non ha mai inserito correttamente gli viene chiesto il codice sviluppatore.

MainDeveloperActivity invece si occupa della gestione delle opzioni sviluppatore:

- visualizzazione lista log salvati localmente nel device;
- passare alla visualizzazione dettagliata log;
- avviare un nuovo log.

7.4.2 Interfaccia grafica

L'area sviluppatore è composta da diverse viste. Selezionando dal menu *Area sviluppatore* si vede la vista descritta in DeveloperUnlockerView la quale richiede l'inserimento di una password. Una volta effettuato correttamente l'accesso viene mostrata MainDeveloperView dove si presenta una lista dei log salvati e un pulsante per crearne uno nuovo. Alla creazione si passa alla vista LoggingView. Si può inoltre visualizzare un contenuto di un log per volta selezionandoli in MainDeveloperView e la visualizzazione interna viene gestita da LogInformationView.

Componenti interne

- Package:

  com.leaf.clips.view;

- Interfacce e classi:

  MainDeveloperViewImp, MainDeveloperView, DeveloperUnlockerViewImp, DeveloperUnlockerView, LoggingViewImp, LoggingView, LogInformationViewImp, LogInformationView;

Componenti esterne

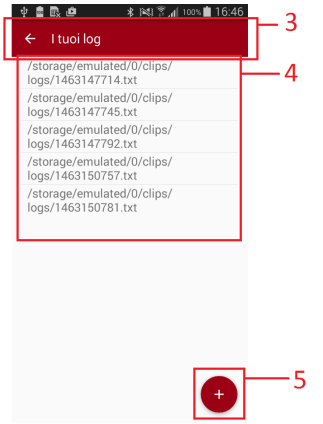
- Interfacce e classi SDK:

  EditText, Button, FloatingActionButton, ListView, TextView, Toolbar;

Nelle figure 16a e 16b si indicano i seguenti widget offerti dal kit Android SDK:

1. EditText;
2. Button;
3. Toolbar;
4. ListView;
5. FloatingActionButton;

(a) Area sviluppatore - Inserimento password



(b) Area sviluppatore - Visualizzazione lista log salvati

Figura 16: Area sviluppatore

7.4.3 Presenter

Il compito del presenter per questa funzionalità è affidato agli oggetti MainDeveloperPresenter, DeveloperUnlockerActivity, MainDeveloperActivity, LoggingActivitye LogInformationActivity. MainDeveloperPresenter comunica con Setting per accertarsi se l'utente sia sviluppatore, in tal caso la gestione passa a MainDeveloperActivity. Nel caso contrario la gestione passa a DeveloperUnlockerActivity la quale richiede la password per accedere all'area sviluppatore. In generale tutte le altre activity comunicano con la classe InformationManagerImp del model e la relazione avviene grazie all'uso della dependency injection.

Componenti interne

- Package:

  com.leaf.clips.view;

  com.leaf.clips.presenter;

- Interfacce e classi:

  MainDeveloperPresenter, MainDeveloperActivity, MainDeveloperView, DeveloperUnlockerActivity, DeveloperUnlockerView, LoggingActivity, LoggingView, LogInformationActivity, LogInformationView;

Componenti esterne

- Interfacce e classi SDK:

  AppCompatActivity;

7.4.4 Logging

La classe InformationManagerImp rende disponibile quattro metodi per eseguire l'operazione di logging:

- getLogInfo() per reperire i riferimenti ai file Log;
- startRecordingBeacon() per far sì che un nuovo file Log sia scritto;
- removeBeaconInformationFile() per rimuovere un file log dal dispositivo;
- saveRecordedBeaconInformation() per salvare il nuovo log inizializzato con startRecordingBeacon();

La classe Setting invece è utilizzata per ottenere il path della cartella in cui sono salvati i Log. Tale path è salvato nelle preferenze dell'applicazione rese disponibili da SharedPreferences.

Componenti interne

- Package:

  com.leaf.clips.model;

  com.leaf.clips.model.usersetting;

- Interfacce e classi:

  InformationManager, InformationManagerImp, Setting;

Componenti esterne

- Interfacce e classi SDK:

  SharedPreferences, Log;

8 Persistenza dei dati

8.1 Database locale e remoto

L'applicazione è costituita da un database locale in SQLite che in base alle necessità dell'utente si sincronizza con quello remoto in PostgreSQL attraverso un RESTful server. Il database contiene tutti i dati necessari a comporre i grafi degli edifici supportati. L'applicazione, ossia la parte client della comunicazione, effettua solo richieste GET dal server, nessun'altra operazione può essere effettuata (DELETE, POST e PUT) dall'applicazione.

8.2 Diagramma ER

Di seguito si descrive il Diagramma ER della base di dati rappresentato in figura 17.

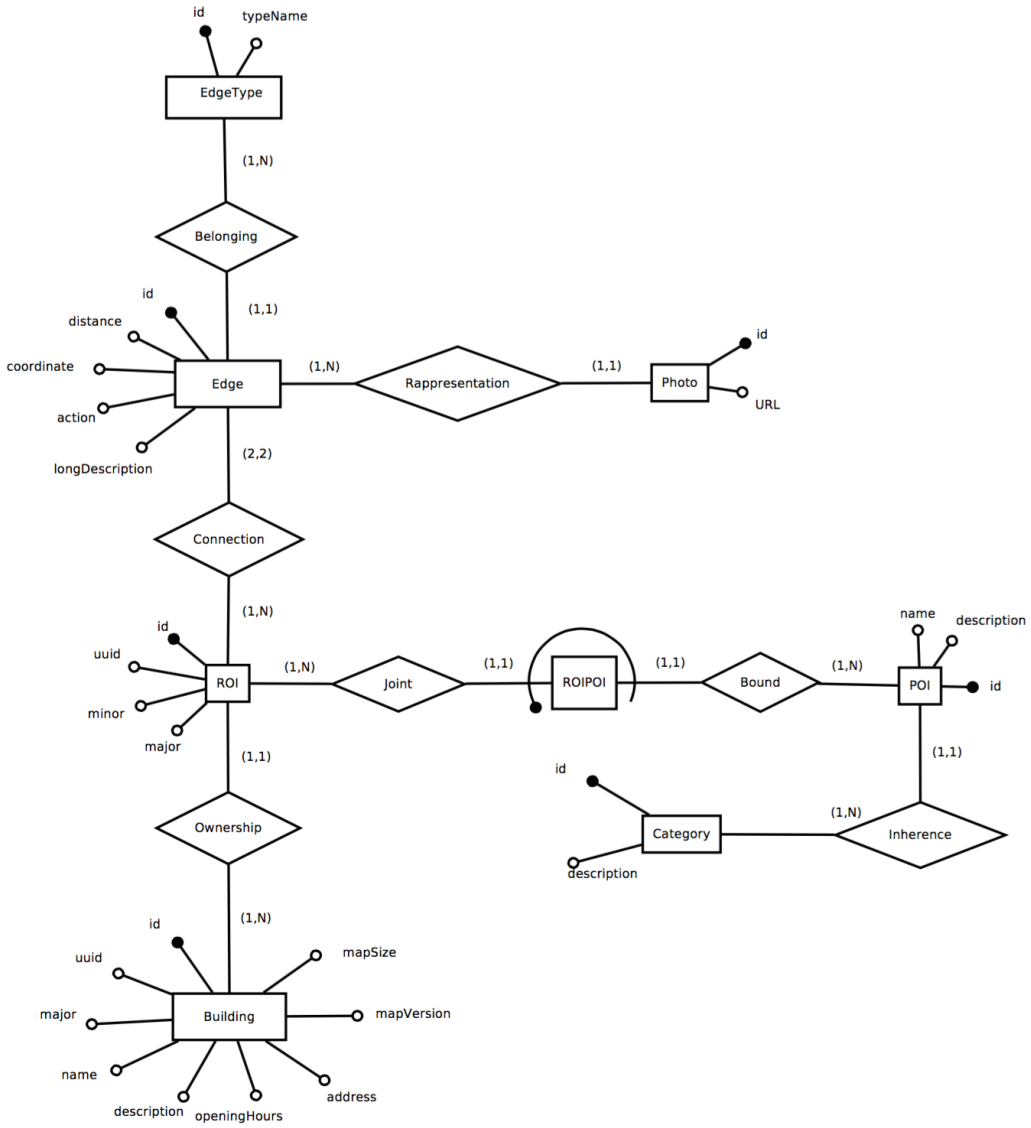


Figura 17: Diagramma ER - Base di dati dell'applicazione

8.3 Descrizione relazioni

8.3.1 Building

Relazione che contiene le informazioni degli edifici. Attributi:

- **id**: chiave primaria, intero;
- **uuid**: stringa, rappresenta l'identificativo UUID uguale per tutti i Beacon utilizzati dall'applicazione;
- **major**: intero, rappresenta l'identificativo Major uguali per tutti i Beacon di uno stesso edificio, chiave esterna verso la relazione Building;
- **name**: stringa, rappresenta il nome dell'edificio;
- **description**: stringa, rappresenta la descrizione dell'edificio e delle sue funzionalità;
- **openingHours**: stringa, rappresenta gli orari di apertura dell'edificio;
- **address**: stringa, rappresenta l'indirizzo dell'edificio;
- **mapVersion**: stringa, rappresenta la versione della mappa;
- **mapSize**: stringa, rappresenta la dimensione della mappa.

8.3.2 ROI

Relazione che contiene le informazioni delle Region of interest. Attributi:

- **id**: chiave primaria, intero;
- **uuid**: stringa, rappresenta l'identificativo UUID uguale per tutti i Beacon utilizzati dall'applicazione;
- **major**: intero, rappresenta l'identificativo Major uguali per tutti i Beacon di uno stesso edificio;
- **minor**: intero, rappresenta l'identificativo Minor che identifica univocamente un Beacon in un edificio.

8.3.3 ROIPOI

Relazione che contiene le associazioni tra Region of interest e Point of interest. Attributi:

- **idRoi**: intero, chiave esterna verso la relazione ROI;
- **idPoi**: intero, chiave esterna verso la relazione POI;
- chiave primaria (idRoi, idPoi).

8.3.4 POI

Relazione che contiene le informazioni dei Point of interest. Attributi:

- **id**: chiave primaria, intero;
- **name**: string, rappresenta il nome associato al Point of interest;
- **description**: stringa, rappresenta la descrizione del Point of interest.

8.3.5 Category

Relazione che contiene le informazioni delle categorie di Point of interest. Attributi:

- **id**: chiave primaria, intero;
- **description**: stringa, rappresenta la descrizione della categoria di Point of Interest.

8.3.6 Edge

Relazione che contiene le informazioni delle connessioni tra le Region of interest, che rappresentano gli archi del grafo che rappresenta un edificio. Attributi:

- **id**: chiave primaria, intero;
- **distance**: intero, rappresenta la lunghezza dell'arco;
- **coordinate**: string, rappresenta l'ampiezza dell'arco che ha per lati l'arco e il collegamento tra la Region Of Interest di partenza e il nord polare;
- **action**: string, rappresenta le azioni che bisogna compiere per superare l'arco;

- **longDescription**: string, rappresenta una descrizione dettagliata delle azioni che bisogna compiere per superare l'arco;
- **startROI**: intero, chiave esterna verso la relazione ROI, rappresenta la Region of interest di partenza dell'arco;
- **endROI**: intero, chiave esterna verso la relazione ROI, rappresenta la Region of interest di arrivo dell'arco.

8.3.7 EdgeType

Relazione che contiene le informazioni sui tipi degli archi. Attributi:

- **id**: chiave primaria, intero;
- **typeName**: stringa, rappresenta la descrizione del tipo di arco.

8.3.8 Photo

Relazione che contiene i link alle immagini associati ad un arco. Attributi:

- **id**: chiave primaria, intero;
- **URL**: stringa, rappresenta l'URL a cui recuperare l'immagine;
- **edgeId**: intero, chiave esterna verso la relazione Edge, rappresenta l'Edge a cui è associata l'immagine.

8.4 Descrizione associazioni

8.4.1 Ownership

Associazione che unisce ogni ROI all'edificio di appartenenza.
Molteplicità:(1,N) Ad ogni Building possono essere associati uno o più ROI, ogni ROI può essere associato un solo edificio.

8.4.2 Connection

Associazione che unisce ogni Edge ai ROI di partenza e arrivo.
Molteplicità:(2,N) Ad ogni Edge associa il ROI di inizio e fine, ogni ROI può essere associato ad uno o più Edge.

8.4.3 Joint

Associazione che unisce ogni ROI ai ROIPOI di appartenenza.
Molteplicità:(1,N) Ogni ROI può essere associato ad uno o più ROIPOI, ogni ROIPOI può essere associato ad un unico ROI.

8.4.4 Bound

Associazione che unisce ogni POI ai ROIPOI di appartenenza.
Molteplicità:(1,N) Ogni POI può essere associato ad uno o più ROIPOI, ogni ROIPOI può essere associato ad un unico POI.

8.4.5 Inherence

Associazione che unisce ogni POI alla categoria a cui appartiene.
Molteplicità:(1,N) Ogni POI può essere associato ad una unica Category, ogni Category può essere associata a più POI.

8.4.6 Rappresentation

Associazione che unisce ogni Photo all'Edge che rappresenta.
Molteplicità:(1,N) Ogni Photo può essere associata ad un unico Edge, ogni Edge può avere più Photo.

8.4.7 Belonging

Associazione che unisce ogni Edge al tipo a cui appartiene.
Molteplicità:(1,N) Ogni Edge può essere associato ad un unico EdgeType, ogni EdgeType può essere associato ad uno o più Edge.

8.5 Struttura oggetti Json

La struttura di seguito proposta ricalca la struttura data agli oggetti JSON scaricati dal database remoto per l'installazione di mappe in locale. Nel caso in cui si voglia cambiare tale struttura si consiglia di estendere le classi con prefisso **Remote** e suffisso **Dao** presenti nel package **dao**.

8.5.1 Esempio di oggetto Json rappresentante una mappa

```
{
  "building" : {
    "id" : 1,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
```

```
    "major" : 666,
    "name" : "Torre Archimede"
    "description" : "Edificio del Dipartimento di Matematica",
    "openingHours" : "08:00-19:00",
    "address" : "Via Trieste 63, 35121, Padova (PD)",
    "mapVersion" : "1.0",
    "mapVersion" : "5.2 KB"
  },
  "rois" : [ {
    "id" : 1,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
    "major" : 666,
    "minor" : 1001
  }, {
    "id" : 2,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
    "major" : 666,
    "minor" : 1002
  }, {
    "id" : 3,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
    "major" : 666,
    "minor" : 1003
  } ],
  "categories" : [ {
    "id" : 2,
    "description" : "Aule"
  }, {
    "id" : 1,
    "description" : "Bagni"
  } ],
  "pois" : [ {
    "id" : 1,
    "name" : "2BC60",
    "description" : "Aula 2BC60",
    "categoryId" : 2
  }, {
    "id" : 2,
    "name" : "Bagno femminile",
    "description" : "Bagno femminile",
    "categoryId" : 1
  } ],
  "roipois" : [ {
    "roiid" : 1,
    "poiid" : 1
  }, {
    "roiid" : 2,
    "poiid" : 2
  } ],
```

```
  "edgeTypes" : [ {
    "id" : 1,
    "description" : "Default"
  } ],
  "edges" : [ {
    "id" : 1,
    "startROI" : 1,
    "endROI" : 2,
    "distance" : 50,
    "coordinate" : "23",
    "typeId" : 1,
    "action" : "Alla fine del corridoio troverai
      il bagno femminile.",
    "longDescription" : "Esci da aula 2BC60,
      prosegui nel corridoio e in fondo a
      sinistra troverai il bagno femminile"
  } ],
  "photos" : [ {
    "id" : 1,
    "edgeId" : 1,
    "url" : "URL della prima foto"
  }, {
    "id" : 2,
    "edgeId" : 1,
    "url" : "URL della seconda foto"
  } ]
}
```

8.5.2 Descrizione oggetto building

```
...
  "building" : {
    "id" : 1,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
    "major" : 666,
    "description" : "Edificio del Dipartimento di Matematica",
    "openingHours" : "08:00-19:00",
    "address" : "Via Trieste 63, 35121, Padova (PD)",
    "mapVersion" : "1.0",
    "mapSize" : "5.2 KB"
  }
...
```

Tale oggetto è utilizzato per raccogliere le informazioni generali riguardanti un edificio e la sua mappa. In particolare:

- **id**: Rappresenta l'identificativo numerico univoco dell'oggetto;
- **uuid** Rappresenta l'identificativo UUID uguale per tutti i Beacon sfruttati dall'applicativo;
- **major** Rappresenta l'identificativo Major uguale per tutti i Beacon appartenenti ad uno stesso edificio;
- **name** Rappresenta il nome dell'edificio;
- **description** Rappresenta una descrizione dell'edificio. In questa parte si consiglia di spiegare la tipologia di edificio e per cosa tale edificio è utilizzato;
- **openingHours** Rappresenta l'orario di apertura e chiusura dell'edificio;
- **address** Rappresenta l'indirizzo dell'edificio;
- **mapVersion** Rappresenta la versione della mappa;
- **mapSize** Rappresenta la dimensione della mappa.

8.5.3 Descrizione oggetto rois

```
...
  "rois" : [ {
    "id" : 1,
    "uuid" : "f7826da6-4fa2-4e98-8024-bc5b71e0893e",
    "major" : 666,
    "minor" : 1001
  },
  ...
  ],
...
```

Tale oggetto è utilizzato per rappresentare tutti le Region Of Interest di un certo edificio. Ogni oggetto all'interno all'interno di tale array rappresenta una specifica Region Of Interest. In particolare:

- **id** Rappresenta l'identificativo numerico univoco dell'oggetto;

- **uuid** Rappresenta l'identificativo UUID uguale per tutti i Beacon sfruttati dall'applicativo;
- **major** Rappresenta l'identificativo Major uguale per tutti i Beacon appartenenti ad uno stesso edificio;
- **minor** Rappresenta l'identificativo univoco di un certo Beacon all'interno di un edificio.

8.5.4 Descrizione oggetto pois

```
...
  "pois" : [ {
  "id" : 1,
  "name" : "2BC60",
  "description" : "Aula 2BC60",
  "categoryId" : 2
    },
  ...
  ],
...
```

Tale oggetto è utilizzato per rappresentare tutti i Point Of Interest di un certo edificio. Ogni oggetto all'interno all'interno di tale array rappresenta uno specifico Point Of Interest. In particolare:

- **id** Rappresenta l'identificativo numerico univoco dell'oggetto;
- **name** Rappresenta il nome associato ad uno specifico Point Of Interest;
- **description** Rappresenta una descrizione associata ad un Point Of Interest. Si consiglia di mettere in tale descrizione la funzione di tale Point Of Interest;
- **categoryId** Rappresenta l'identificativo associato alla categoria di appartenenza del Point Of Interest.

8.5.5 Descrizione oggetto roipois

```
...

  "roipois" : [ {
  "roiid" : 1,
```

```
  "poiid" : 1
  },
  ...
  ],
...
```

Tale oggetto è utilizzato per rappresentare i collegamenti tra Region Of Interest e Point Of Interest in un certo edificio. Ogni oggetto all'interno all'interno di tale array rappresenta uno specifico collegamento. In particolare:

- **roiid** Rappresenta l'identificativo numerico univoco di una Region Of Interest;
- **poiid** Rappresenta l'identificativo numerico univoco di un Point Of Interest.

8.5.6 Descrizione oggetto edges

```
...
  "edges" : [ {
    "id" : 1,
    "startROI" : 1,
    "endROI" : 2,
    "distance" : 50,
    "coordinate" : "23",
    "typeId" : 1,
    "action" : "Alla fine del corridoio troverai
      il bagno femminile.",
    "longDescription" : "Esci da aula 2BC60,
      prosegui nel corridoio e in fondo a
      sinistra troverai il bagno femminile"
  },
  ...
  ]
...
```

Tale oggetto è utilizzato per rappresentare tutti gli archi che collegano Region Of Interest nel grafo che rappresenta un edificio. Ogni oggetto all'interno all'interno di tale array rappresenta uno specifico arco. In particolare:

- **id** Rappresenta l'identificativo numerico univoco dell'oggetto;

- **startROI** Rappresenta la Region Of Interest di partenza dell'arco;
- **endROI** Rappresenta la Region Of Interest di arrivo dell'arco;
- **distance** Rappresenta lunghezza dell'arco;
- **coordinate** Rappresenta l'ampiezza dell'arco che ha per lati l'arco e il collegamento tra la Region Of Interest di partenza e il nord polare;
- **typeId** Rappresenta l'identificativo associato al tipo di appartenenza dell'arco';
- **action** Rappresenta una descrizione basilare delle azioni da compiere per superare l'arco;
- **description** Rappresenta una descrizione dettagliata delle azioni da compiere per superare l'arco.

8.5.7 Descrizione oggetto edgeTypes

```
...
  "edgeTypes" : [ {
    "id" : 1,
    "description" : "Default"
  },
  ...
  ],
...
```

Tale oggetto è utilizzato per rappresentare tutti i tipi di arco che possono essere presenti all'interno di un edificio. Ogni oggetto all'interno all'interno di tale array rappresenta uno specifico tipo di arco. In particolare:

- **id** Rappresenta l'identificativo numerico univoco di un tipo;
- **description** Rappresenta una descrizione testuale del tipo di arco.

8.5.8 Descrizione oggetto photos

```
...
  "photos" : [ {
    "id" : 1,
    "edgeId" : 1,
```

```
    "url" : "www.imageurl.com"
  },
  ...
  ],
...
```

Tale oggetto è utilizzato per rappresentare i link alle immagini utili alla navigazione. Ogni oggetto all'interno all'interno di tale array rappresenta il collegamento ad una specifica immagine collegata ad uno specifico arco. In particolare:

- **id** Rappresenta l'identificativo numerico univoco dell'oggetto;
- **edgeId** Rappresenta l'identificativo numerico univoco della Region Of Interest a cui l'immagine è collegata;
- **url** Rappresenta l'URL da cui è possibile recuperare l'immagine.

A Fondamenti di Android

Nella presente sezione si riportano le principali conoscenze di Android utilizzate nell'applicazione e quindi indispensabili per una maggiore e più completa comprensione del manuale.

A.1 Activity

Un'Activity è una classe offerta dall'Android SDK che ha lo scopo di gestire una schermata della propria applicazione. Ogni view quindi deve essere supportata da una classe che estende Activity.

A.1.1 Ciclo di vita

Il potenziale di un'Activity risiede nella curata gestione del proprio ciclo di vita e quindi dell'interfaccia grafica associata. Infatti nei dispositivi mobile le applicazioni hanno un diverso ciclo di vita rispetto alle normali applicazioni nei comuni personal computer.

Un'applicazione mobile necessita di essere in diversi stati a seconda dell'interazione dell'utente, garantire una user experience sempre ottimale e sfruttare al meglio le risorse hardware limitate.

I diversi stati di una sottoclasse Activity sono i seguenti:

- onCreate() metodo invocato alla creazione dell'Activity;
- onStart() metodo invocato per ripartire l'Activity precedentemente abbandonata dall'utente e lasciata
- onResume() metodo invocato quando l'activity ha ripreso ad essere in primo piano;
- onPause() metodo invocato subito dopo che l'Activity perdi il posto in primo piano, per esempio per un pop-up; in background dal sistema operativo;
- onStop() metodo invocato dopo che l'activity lascia il posto ad un'altra;
- onDestroy() metodo invocato nel caso in cui il sistema operativo necessiti di risorse e l'activity non è utilizzata dall'utente.

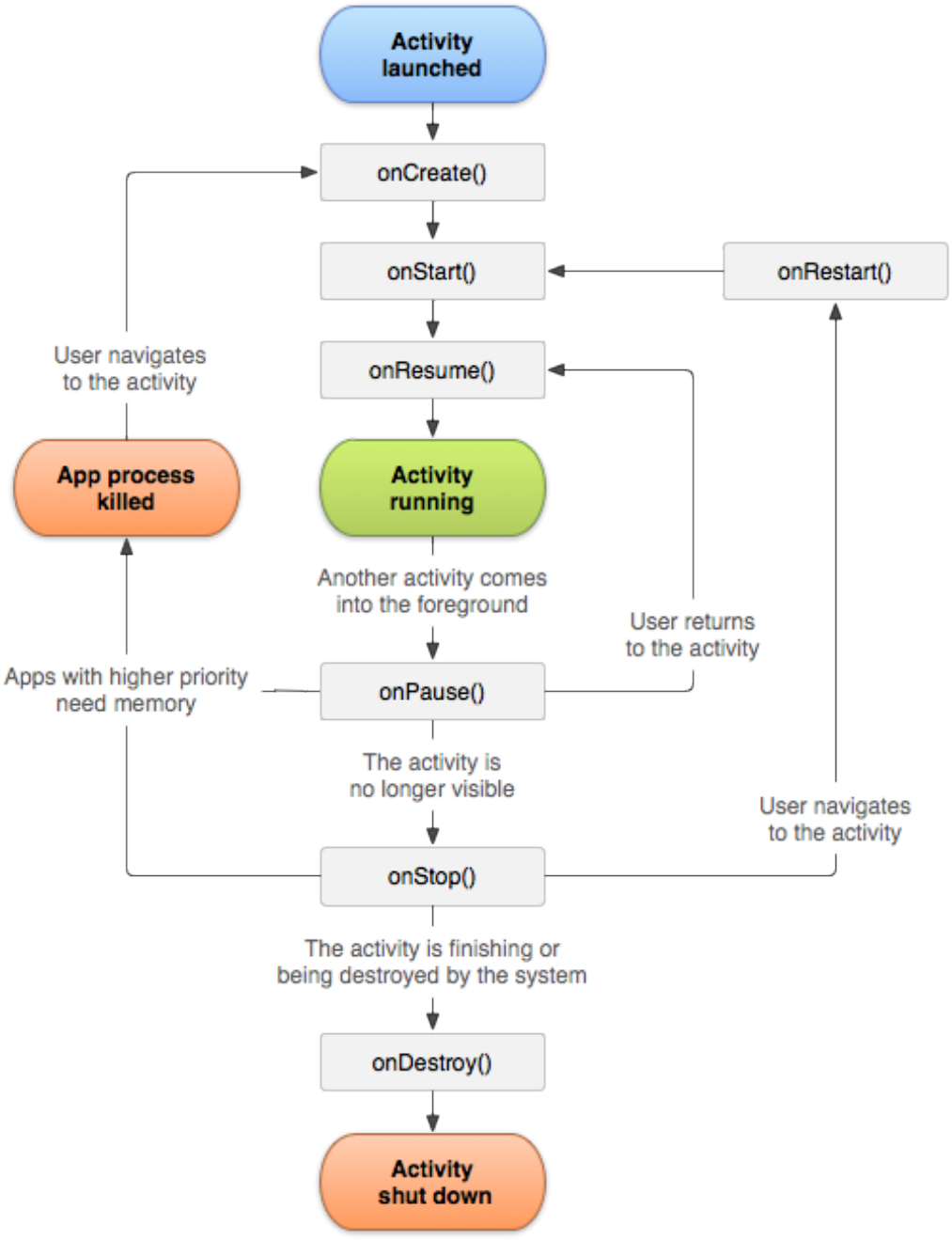


Figura 18: Ciclo di vita Activity

A.2 Service

Un Service è una classe offerta dall'Android SDK per poter gestire processi in background in una applicazione con non necessitano di un'interfaccia (come nel caso delle Activity).

Esistono due tipologie di Service:

unBind Service: è un Service che comunica con le altre componenti dell'applicazione solo attraverso messaggi Intent e BroadcastReceiver;

Bind Service: è un Service che rende disponibile la possibilità di ottenere un riferimento su di esso e quindi di invocare metodi implementati da esso stesso. Questa comunicazione avviene attraverso l'uso dell'IBinder.

A.2.1 Ciclo di vita

Come l'Activity anche il Service mette a disposizione tutti i metodi per gestire il proprio ciclo di vita in modo da istruire l'applicazione a rispondere alle richieste del sistema operativo, per esempio la richiesta di fermare un Service per liberare risorse. I metodi offerti oltre a quelli prima esposti nella sezione Activity sono:

- onStartCommand() è il metodo invocato quando un'altra componente avvia il Service con la chiamata startService(intent), spesso un'Activity.
- onBind() è il metodo messo a disposizione dal Bind Service per poter restituire un riferimento al Service stesso.

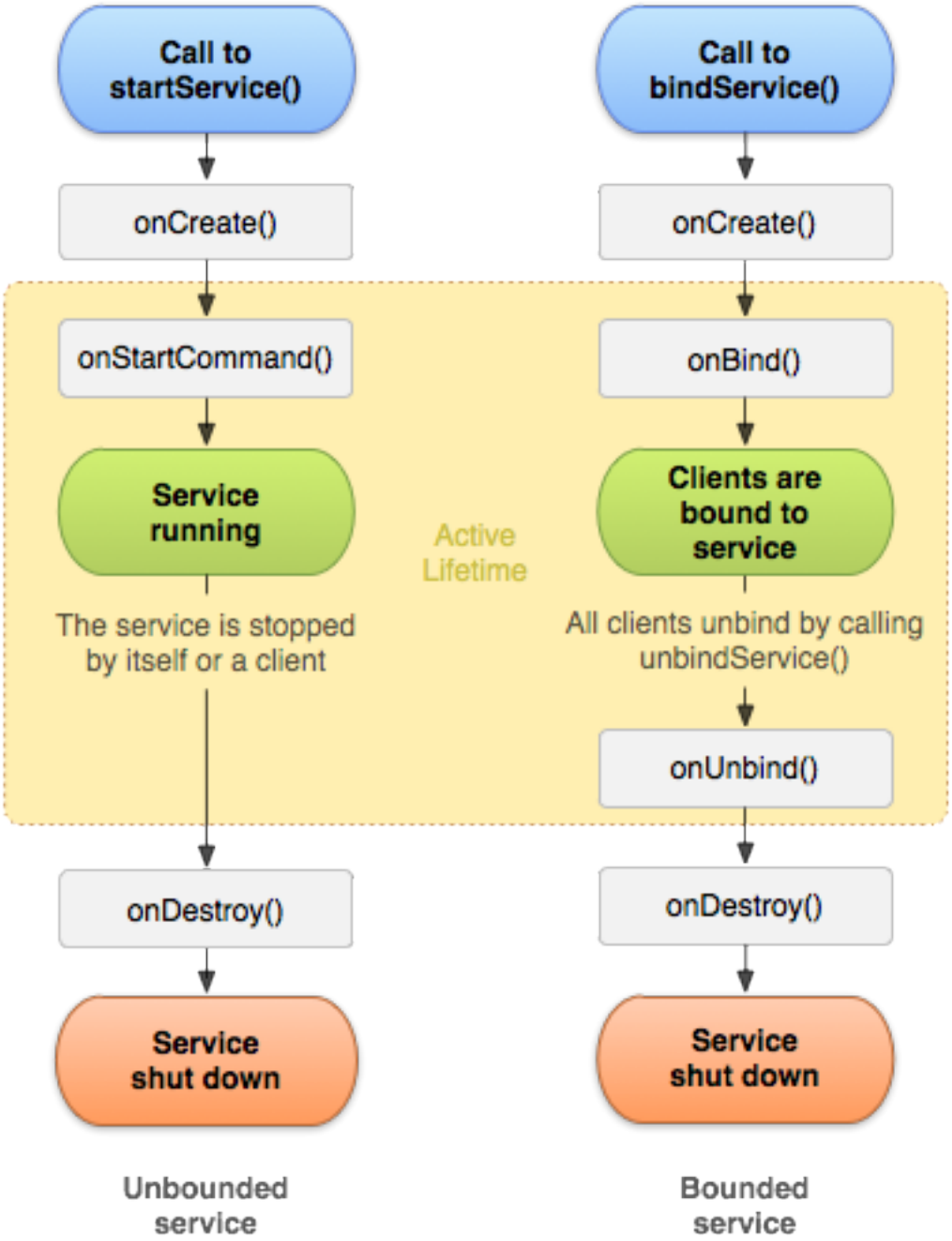


Figura 19: Ciclo di vita unBind Service e Bind Service

Glossario

AltBeacon Framework open source che offre potenti strumenti al fine di utilizzare il protocollo BLE utilizzato dai dispositivi beacon.. 2, 28

Android Sistema operativo open source per dispositivi mobili sviluppato da Google. 1, 2, 5, 7–10, 13, 20, 26, 35, 41

Android Studio IDE di riferimento per lo sviluppo di applicazioni per Android. 1, 3, 5, 7, 8

Beacon Trasmettitore Bluetooth. 1, 13–17, 25, 26, 28, 39, 40, 46, 52, 53

Framework Architettura logica di supporto su cui un software può essere progettato e realizzato facilitandone lo sviluppo da parte del programmatore.. 8

Java Linguaggio di programmazione orientato agli oggetti, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. 2, 8, 13, 19

Monitoring Azione scatenata dall'entrata o uscita dal raggio di una Region, permettendo di sapere se in quel determinato raggio sono stati rilevati dei beacon. Il monitoring può avvenire indipendentemente dal fatto che l'applicazione sia nello stato running, suspended o killed. 25, 28

Navigazione indoor Navigazione effettuata all'interno di un edificio o di una struttura. 14

Open source Software di cui gli autori (più precisamente i detentori dei diritti) rendono pubblico il codice sorgente, permettendo a programmatori indipendenti di apportarvi modifiche. Questa possibilità è regolata tramite l'applicazione di apposite licenze d'uso. 9

POI Acronimo di Point Of Interest, area di un edificio che può risultare interessante per l'utente. 14, 47, 49

Ranging Azione scatenata in prossimità di un beacon, permette di ricavarne determinati dati (es. potenza del segnale o distanza). Funziona solamente se l'applicazione è nello stato running. 25, 28

ROI Acronimo di Region of Interest, area coperta dal segnale di un beacon. 14, 47–49

Software In informatica, è l'informazione o le informazioni utilizzate da uno o più sistemi informatici e memorizzate su uno o più supporti informatici. Tali informazioni possono essere quindi rappresentate da uno o più programmi, oppure da uno o più dati, oppure da una combinazione delle due. 19

Wrapper Modulo software che ne avvolge un altro per evitare le dipendenze su quest'ultimo. 7