Comune di Bari

DOCUMENTAZIONE TECNICA

“Servizi di interoperabilità per i dati e di cooperazione applicativa”

Sistema Pubblico di Connettività - Lotto 3

Applicazione ConsoleIoT&BigData

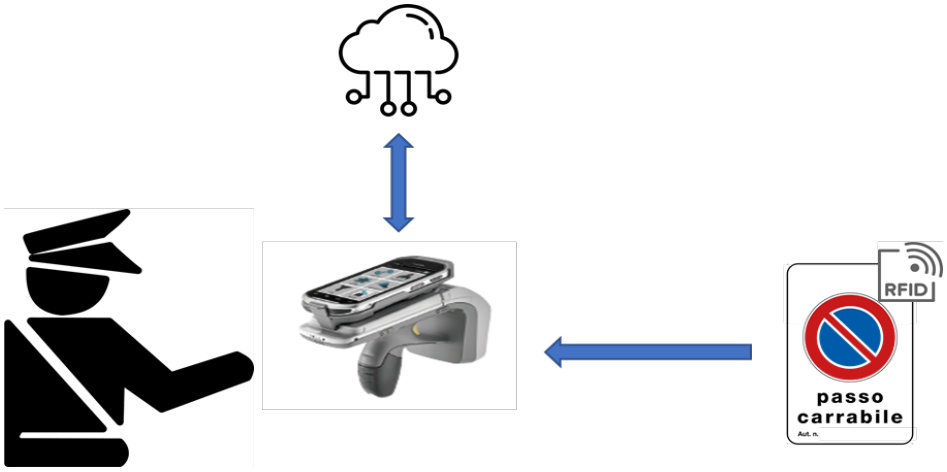
Ver.	Elabora	Verifica	Approva	Data emissione	Descrizione delle modifiche
1.0	Alessandro Viola	Flavia Rotondi			Prima emissione

SOMMARIO

1 INTRODUZIONE

1.1 PREMESSA

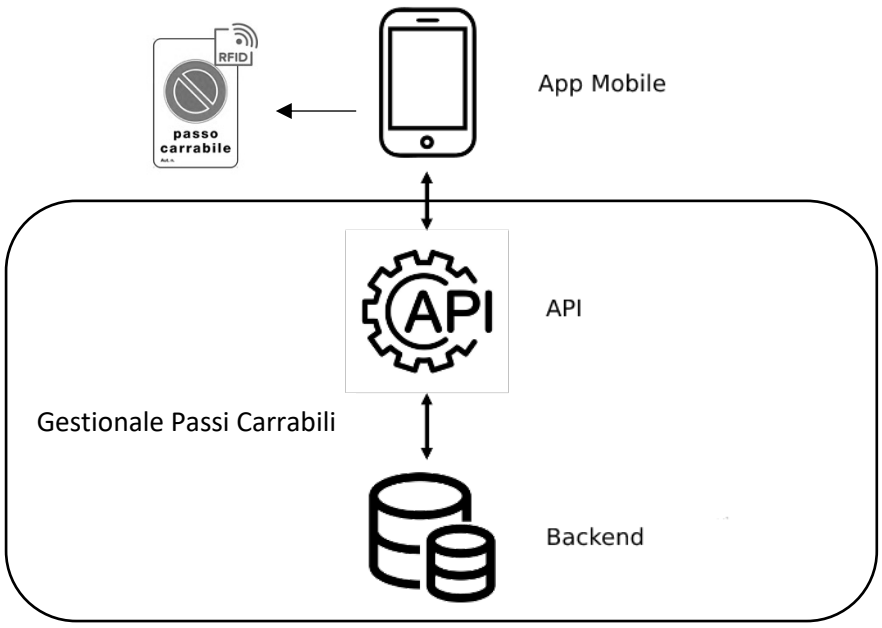
Scopo del documento è illustrare l'architettura e le specifiche tecniche dell'Applicazione per Tablet ConsoleIoT&BigData posta nell'ambito di un progetto più generale quale quello di Bari Città Connessa, esplicitando tecnologie scelte, frameworks e le scelte di user interface ed experience.



Attraverso l'applicazione in questione gli agenti di Polizia Locale (utenti dell'applicazione), dopo essersi loggati, avranno sulla mappa che gli verrà mostrata l'elenco dei passi carrabili presenti nel territorio del Comune di Bari. Potranno così visualizzare i dati relativi alle concessioni compresa la loro validità. Avranno strumenti che permetteranno loro di segnalare eventuali problemi od incongruenze e controllare la loro correttezza anche con strumenti esterni (quali lettori RFID e QRCode). Potranno infine visualizzare la propria anagrafica e l'elenco delle segnalazioni effettuate.

L'applicazione è nativamente integrata con la base dati del Gestionale Passi Carrabili (GPC), utilizzando i servizi forniti dallo stesso.

2 ARCHITETTURA



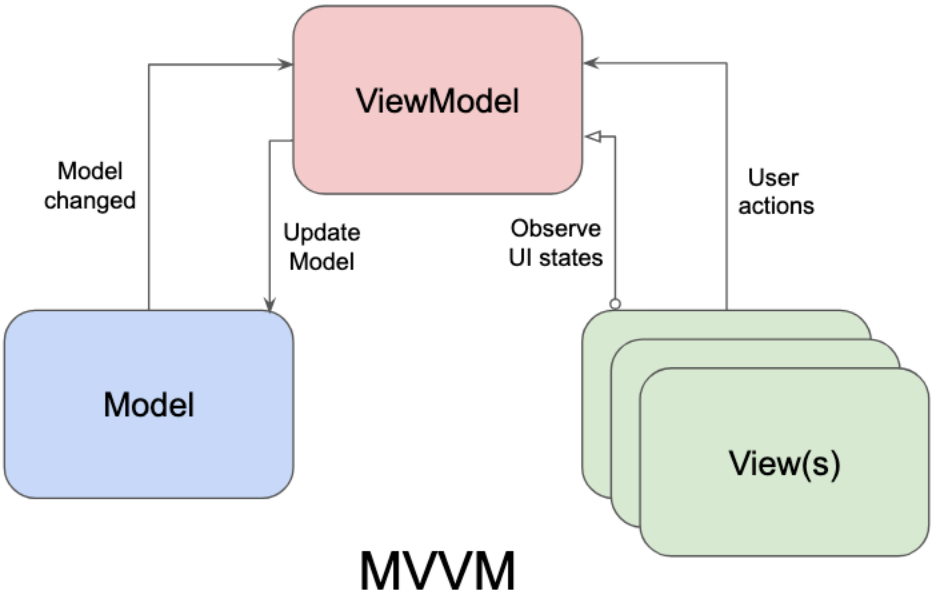
2.1 ARCHITETTURA HIGH LEVEL E PATTERN MVVM

L'architettura dell'applicazione Android, scritta in Kotlin, basata su una connessione a servizi con API REST, si basa sui principi architetturali del pattern MVVM (Model-View-ViewModel) e dell'architettura a strati.

La struttura principale dell'applicazione è composta da tre layer principali:

1. **Presentation Layer**: questo strato gestisce l'interfaccia utente dell'applicazione, compresa la presentazione dei dati e l'interazione con l'utente. In questo layer, viene implementato il pattern MVVM in cui le View rappresentano l'interfaccia utente dell'applicazione, il ViewModel gestisce lo stato dell'interfaccia utente e del modello dell'applicazione e il Model rappresenta il modello dei dati dell'applicazione.
2. **Business Layer**: questo strato rappresenta il core dell'applicazione e contiene la logica di business dell'applicazione. Il Business Layer interagisce con il DataAccess Layer per ottenere i dati dai servizi REST.
3. **DataAccess Layer**: questo strato gestisce l'accesso dei dati dal servizio REST. Utilizzando le librerie di connessione REST, il DataAccess Layer è responsabile di effettuare le richieste ai servizi REST dell'applicativo GPC e di trasformare i dati in un formato utilizzabile dalle altre componenti dell'applicazione.

L'architettura a strati offre numerosi vantaggi in termini di separazione dei compiti e isolamento delle componenti. Inoltre, il pattern MVVM sfrutta la separazione tra le componenti dell'interfaccia utente e la logica di business dell'applicazione, permettendo una maggiore testabilità del codice.



2.2 ARCHITETTURA NATIVA ANDROID

L'architettura nativa per applicazioni Android si basa su tre componenti fondamentali: Activity (e Fragment), ViewModel e Repository.

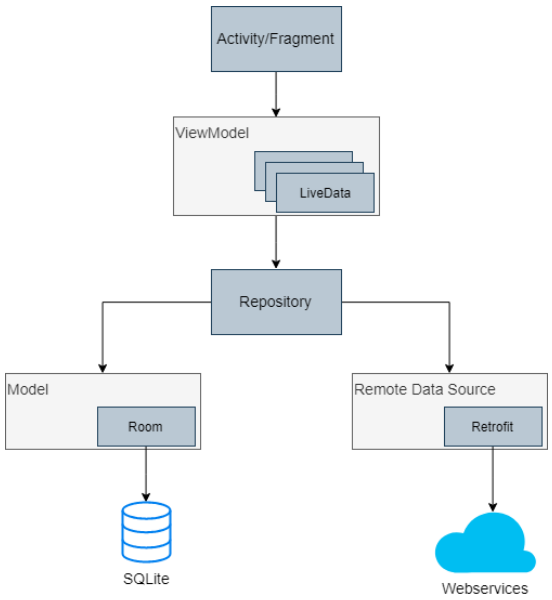
L'Activity rappresenta la finestra dell'applicazione in cui l'utente interagisce con l'applicazione. Le Activity possono comunicare tra di loro tramite l'invio di Intent. Ad ogni Activity è associato un layout XML nel quale vengono definiti tutti gli elementi visivi della schermata.

Il ciclo di vita di un'Activity è suddiviso in tre parti: onCreate(), onStart() e onResume(), che rappresentano rispettivamente il momento in cui l'Activity viene creata, il momento in cui diventa visibile all'utente e il momento in cui diventa attiva e riceve gli input dell'utente. In caso di distruzione dell'Activity da parte del sistema, il metodo onDestroy() viene richiamato e l'Activity viene eliminata.

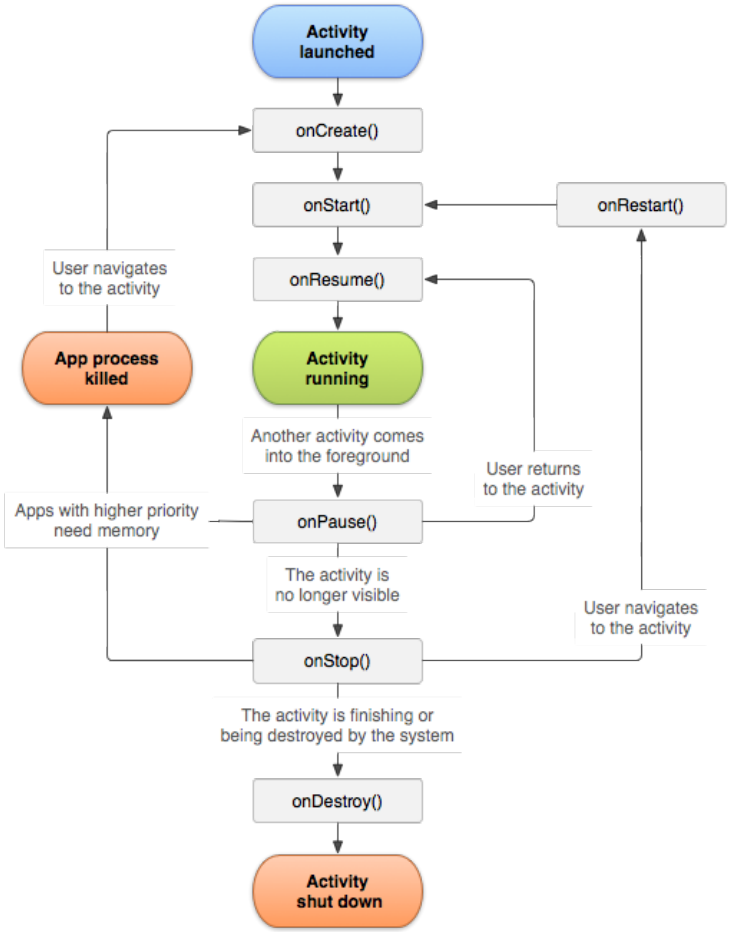
Il ViewModel rappresenta la parte di un'applicazione che si occupa di gestire i dati e la logica di business dell'applicazione. È uno strumento utile per garantire la separazione tra la logica dell'applicazione e l'interfaccia utente. Il ViewModel è indipendente dalla View e può essere utilizzato in modo indipendente dal ciclo di vita dell'Activity in cui è stato creato.

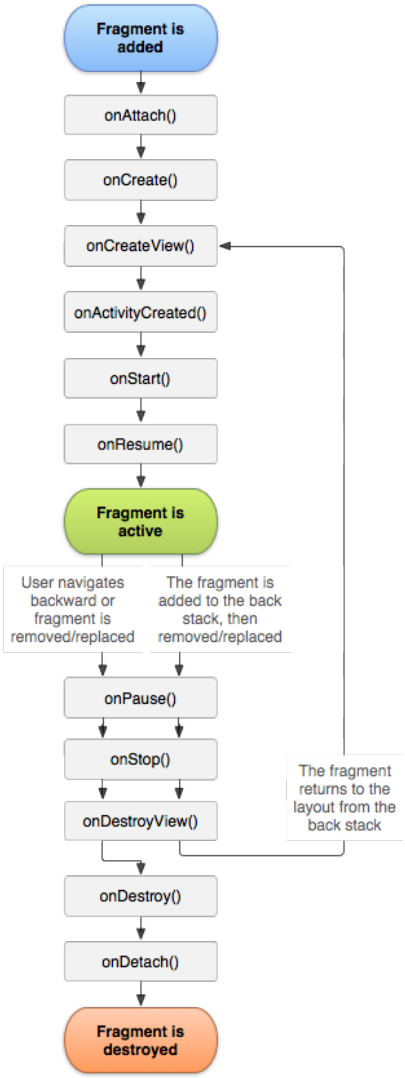
Il Repository rappresenta la parte dell'applicazione che si occupa di leggere e scrivere dati nell'applicazione, sia su un server remoto che su un database interno. In questo modo si garantisce una maggiore interoperabilità tra le diverse componenti dell'applicazione e si facilita la manutenzione del codice.

Il Repository è un'astrazione della sorgente dei dati che l'applicazione utilizza per ottenere i dati; esso utilizza chiamate ad API tramite Retrofit.



I Fragment sono le componenti visuali dell'applicazione e vengono utilizzati per creare interfacce utente frammentate. Il ciclo di vita di un Fragment è simile a quello di un'Activity poiché è suddiviso in tre parti: onCreate(), onCreateView() e onResume(). In caso di distruzione del Fragment, viene richiamato il metodo onDestroyView().





2.3 STRUTTURA CODICE

Il codice sorgente è disponibile al seguente indirizzo:

https://github.com/comunedibari/gestionale-passi-carrabili/tree/main/console-iotbigdata

Di seguito l'albero delle cartelle nel quale è suddiviso il codice del progetto Android:

1. **“java”**: questa cartella contiene tutto il codice sorgente del progetto, compresi i pacchetti e le classi.

   console-iotbigdata/app/src/main/java/com/zebra/rfidreader/demo/bari_citta_connessa

   I package principali in cui è suddiviso sono:

   1. **“adapter”** che contiene classi per la gestione delle liste
   2. **“data”** che contiene sia i modelli all’applicazione sia i template di mapping relativi agli oggetti che arrivano dai servizi
   3. **“network”** lo strato omonimo comprensivo di chiamate API ai servizi e creazione e gestione di Retrofit che permette di connettersi e utilizzarle
   4. **“ui”** con le activity, fragment e viewmodel descritti precedentemente
2. **“res”:** in questa cartella (console-iotbigdata/app/src/main/res) si trovano tutti i file di risorse dell'applicazione, come i layout XML, le immagini e i file audio.

   Più nel dettaglio:

   1. **“layout”** che contiene i file XML utilizzati per la definizione del layout grafico
   2. **“drawable”** con tutti i file di immagini utilizzati, dalle icone alle immagini di sfondo
   3. **“values”** file XML per definire le variabili globali quali le dimensioni dei testi ed i margini, un file per raccogliere le stringhe ed un altro per la catalogazione dei colori
   4. **“font”** per lo stile dei testi
3. **“manifest”:** nella cartella console-iotbigdata/app/src/main si trova il file AndroidManifest.xml che contiene le informazioni di configurazione dell'applicazione, come il pacchetto, le autorizzazioni e l'icona dell'applicazione.
4. **“Gradle”:** in questa cartella (console-iotbigdata/.gradle) sono presenti i file di build che vengono utilizzati da Gradle per la costruzione e la compilazione dell'applicazione.
5. **“Zebra”:** questa libreria presente nella cartella (console-iotbigdata/app/src/main/java/com/zebra/rfidreader/demo/zebra) esterna permette all’applicazione di utilizzare tutti i servizi messi a disposizione relativi alla connessione bluetooth tra il device e i lettori RFID per la scansione dei codici posti sui passi carrabili (descritti meglio in seguito)

2.4 TECNOLOGIE E LIBRERIE ANDROID

Di seguito alcune tecnologie native Android utilizzate nell'applicazione:

1. **Retrofit** è una libreria open-source che consente di gestire le chiamate alle API di un server, in modo semplice e intuitivo. Questa libreria combina il design pattern di Factory e Adapter per la creazione di oggetti e la gestione delle richieste HTTP. Retrofit permette di scaricare dati da un server e di utilizzarli all'interno dell'applicazione in modo efficiente e veloce. Consente inoltre di configurare facilmente il parser JSON per la gestione dei dati scaricati.

2. **Hilt** è un framework di Dependency Injection (DI) che semplifica l'iniezione delle dipendenze in un'applicazione Android e offre un modo standardizzato per gestirle. L'obiettivo principale di Hilt è semplificare lo sviluppo, facilitando l'iniezione delle dipendenze e riducendo il codice duplicato. Inoltre, Hilt è integrato con altre librerie di Google come Room, Retrofit e ViewModel.

3. Il pattern **Observable** in Android è utilizzato per la notifica degli eventi ai suoi osservatori. In questo pattern, il modello stesso gestisce la notifica agli osservatori in caso di eventuali cambiamenti nel modello. Nel sistema Android, il pattern Observable viene implementato tramite l'utilizzo della classe **LiveData**. Essa è un contenitore di dati che può essere osservato da componenti UI, come ad esempio Activity o Fragment. La classe LiveData si integra al meglio con l'architettura MVVM poiché consente di separare la logica della presentazione dai dati. LiveData garantisce la coerenza dati e nelle attività dei suoi osservatori, utilizzando un meccanismo di aggiornamento delle visualizzazioni solo quando necessario. In pratica, i dati vengono mantenuti all'interno degli oggetti di LiveData, che a loro volta hanno uno stato da verificare e attivare quando necessario per aggiornare i detentori di osservazione. Una volta che viene aggiornato lo stato dei dati in LiveData, verrà notificato a tutti i suoi osservatori, consentendo loro di aggiornare le visualizzazioni corrispondenti. Gli osservatori LiveData sono avvisati solo quando i dati sono effettivamente cambiati, in modo da evitare inutili aggiornamenti. Questo migliora le prestazioni dell'applicazione e ottimizza l'uso della memoria.

2.5 LIBRERIE ESTERNE

La libreria **Zebra** è una libreria di software sviluppata da Zebra Technologies; può essere utilizzata con dispositivi Android per consentire la comunicazione tra l'applicazione Android ed i lettori di tag RFID.

Per utilizzare la libreria Zebra con un dispositivo Android, essa è stata integrata direttamente all'interno dell'applicazione. Si è quindi potuto utilizzare la libreria Zebra per accedere alle funzioni di accoppiamento tra device e lettore RFID (tramite bluetooth) e conseguentemente dare la possibilità di leggere i valori TID e EPC presenti all'interno dei tag posizionati sui passi carrabili.

A livello di codice la libreria ha una propria classe "main" con all'interno tutte le funzioni relative all'accoppiamento e alla gestione del lettore (come, ad esempio, la visualizzazione dei livelli di batteria).

3 DETTAGLI PRELIMINARI

3.1 VERSIONI SISTEMA OPERATIVO E DEVICE TARGET

L'applicazione (disponibile solo per Android) è predisposta per supportare le seguenti versioni di sistema operativo:

- Versione minima SDK: 19 (corrispondente ad Android 4.4 "KitKat" rilasciato ad ottobre 2013)
- Version target SDK: 28 (corrispondente ad Android 9.0 "Pie" rilasciato ad agosto 2018).

L'applicazione è ottimizzata per funzionare con orientamento landscape (orizzontale) su dispositivi tablet.

I dispositivi target sono quelli dati in dotazione agli agenti della Polizia Locale del Comune di Bari:

- tablet Zebra con sistemi operativi Android 6.0 "Marshmallow" (dicembre 2015).

3.2 PERMESSI E AUTORIZZAZIONI

Il sistema operativo mostra all'utente le esplicite richieste di "permission" di utilizzo della posizione, connessione bluetooth, accesso fotocamera e tutte le altre che saranno strettamente necessarie per il corretto utilizzo dell'applicazione. In caso l'utente negasse uno o più di questi permessi alcune funzionalità dell'app potrebbero essere limitate o non disponibili.

3.3 LOCALIZZAZIONE

Al fine di permettere un adeguato flusso informativo e di aggregazione delle informazioni, l'app dovrà trasmettere ai sistemi di BE le coordinate geografiche del dispositivo recepite per mezzo del sensore GPS.

Sono riportate qui alcune casistiche che possono inibire il corretto funzionamento dell'app:

- L'utente non accetta la condivisione della sua posizione
- L'utente accetta la condivisione della sua posizione, ma successivamente rimuove tale permesso dalle impostazioni di sistema relative all'app
- Il segnale GPS non è disponibile.

Qualora dovesse verificarsi uno dei precedenti scenari, l'applicazione non potrà funzionare in quello che è il suo normale utilizzo previsto.

4 ELENCO API

Le seguenti API sono presenti all'interno nel pacchetto software "gpc-back-end" che si occupa di fornire i servizi di back end sia per l'applicazione web che per l'applicazione mobile. Si veda anche lo swagger disponibile in formato OpenAPI all'indirizzo:

https://github.com/comunedibari/gestionale-passi-carrabili/blob/main/Documentazione/openapi-gpc.yaml

API	endpoint	Method	Descrizione
Login	auth/login	POST	per l'autenticazione dell'agente tramite username e password
GetPoi	v1/getPoi	POST	per ottenere i dettagli di un asset (passo carrabile) dato il suo id
GetPoiAll	v1/getPoi	POST	per il dettaglio dei passi carrabili attorno ad un'area con un raggio prestabilito
GetCheckAsset	v1/checkAsset	POST	controlla la validità di un passo carrabile
GetReporting	v1/segnalazioni	GET	ritorna la lista delle segnalazioni effettuate
SendSignalation	v1/segnalazioni	PUT	per inviare una segnalazione relativa ad un passo carrabile
CheckAddress	v1/civico	POST	cerca e ritorna (se presente) l'indirizzo del civilario cercato
PraticheCittadino	passicarrabili/praticheCittadino	POST	tramite id, tag rfid o indirizzo ritorna l'elenco delle pratiche cittadino
SetDataVerificaCartello	v1/setDataVerificaCartello	POST	verifica (e torna) la data dell'ultima rilevazione del cartello. In caso la data di rilevazione sia successiva alla data registrata, quest'ultima viene sovrascritta