POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto e implementazione di un'architettura mobile a supporto della gestione logistica dei processi produttivi

Relatore

Prof. Giovanni MALNATI

Candidato

Domenico PIZZATA

Aprile 2021

Alla mia famiglia,
sempre al mio fianco,
nei momenti belli
e in quelli brutti.
Questo traguardo,
lo dedico a voi.

Sommario

Viviamo in una società all'ordine dei dati e tendenzialmente ad ogni ora del giorno possiamo riceverne e trasmetterne. Ogni secondo, una grande quantità di dati viaggia in tutto il mondo e ormai riusciamo a controllarla direttamente da una mano, grazie ad uno smartphone, una cosa impensabile fino al secolo scorso. Raccogliere ed analizzare i dati in modo preciso e puntuale sono, al giorno d'oggi, delle procedure rilevanti e fondamentali soprattutto per le aziende. La moltitudine di interpretazioni e possibili usi dei dati si trasforma però in uno scoglio nel processo di sviluppo di un meccanismo automatizzato, che porti le applicazioni ad essere rapide e funzionali in contesti dove il tempo gioca un ruolo chiave.

In questa tesi si affronta oltre ai problemi della raccolta e del trattamento dei dati, anche il problema dell'automazione in processi logistici, in particolare, con una soluzione al problema della configurazione delle applicazioni per dispositivi mobili, con sistema Android. Si riferisce ad un contesto industriale dove si opera in modo ordinario con la scansione e la trasmissione di tag RFID, codici a barre e QR code, richiedendo quindi una notevole facilità e velocità nella configurazione e nell'utilizzo delle applicazioni, per agire in modo corretto e tempestivo su una vasta scala di prodotti che forniscono i dati per il progetto.

Nella prima parte dell'elaborato, si introducono i problemi presenti in contesti lavorativi dove è necessario avere la massima funzionalità nel minor tempo possibile, discutendo di come le applicazioni possono usare i dati per arrivare a questo obbiettivo.

Nelle parti successive, si descrivono le scelte progettuali ed implementative con una possibile soluzione ai problemi sopra introdotti, sfruttando le caratteristiche di Android e le librerie messe a disposizione per programmare i dispositivi elettronici utilizzati. Si espongono nel dettaglio alcune delle principali applicazioni sviluppate per permettere le funzionalità necessarie.

Infine, sono proposti degli sviluppi futuri a questo progetto analizzando il lavoro svolto fino a questo momento.

Indice

Elenco delle figure VII

Abbreviazioni IX

1 Il problema dei dati nelle applicazioni 1
- 1.1 Operare con i dati . . . 1
- 1.2 I dati in un contesto logistico . . . 2
- 1.3 Lo Standard EPCIS . . . 3
- 1.4 Configurazione delle applicazioni . . . 5

2 Caratteristiche e specifiche di sistema 8
- 2.1 Android . . . 8
  - 2.1.1 Architettura Android . . . 9
  - 2.1.2 Applicazioni Android . . . 10
  - 2.1.3 Android Studio . . . 11
- 2.2 Dispositivi elettronici utilizzati . . . 12
  - 2.2.1 TC20 . . . 12
  - 2.2.2 RFD2000 . . . 13
- 2.3 Funzionalità delle applicazioni . . . 14
- 2.4 Backend del sistema . . . 15
  - 2.4.1 Il server . . . 16
  - 2.4.2 Il database . . . 17
  - 2.4.3 L'utilizzo di Docker . . . 18
  - 2.4.4 Swagger come strumento di supporto . . . 19
- 2.5 Analisi dello scenario di operazione . . . 19

3 Sviluppo e progettazione 22
- 3.1 Java e Kotlin . . . 22
- 3.2 Build di progetti Android complessi . . . 23
- 3.3 L'uso di Room . . . 24
- 3.4 Il pattern Model-View-ViewModel . . . 26

3.5 Autoconfigurazione dei dispositivi 27
3.5.1 Installazione dinamica delle applicazioni 28
3.6 Progettazione delle applicazioni 28
3.6.1 User Experience 29
3.6.2 Progettazione del launcher 30
3.6.3 Progettazione delle app per i processi logistici 31

4 Implementazione del software 34
4.1 Software per gestire i tag RFID 34
4.2 Software per gestire barcode e QR code 35
4.3 Implementazione delle applicazioni 36
4.3.1 Download e installazione di APK 36
4.3.2 Autenticazione client 37
4.3.3 TLS 38
4.3.4 OkHttp 40
4.3.5 Implementazione del launcher 41
4.3.6 Implementazione della parte comune delle app per la gestione dei processi logistici 44
4.3.7 Implementazione dell'app per la registrazione di un prodotto 47
4.3.8 Implementazione dell'app per il tracciamento di un prodotto 49
4.3.9 Implementazione dell'app per l'impacchettamento di prodotti 51
4.3.10 Implementazione dell'app per la verifica di un ordine 53

5 Conclusioni 57
5.1 Possibili sviluppi futuri 58

Bibliografia 60

Elenco delle figure

1.1 Tracciabilità delle merci . . . 3
1.2 Cause di tempi di inattività non pianificati . . . 5

2.1 Architettura Android . . . 9
2.2 Mobile computer TC20 . . . 13
2.3 Culla palmare RFID UHF RFD2000 . . . 14
2.4 Esempio di modello client-server . . . 16
2.5 Esempio di architettura delle API RESTful . . . 17
2.6 Virtual machines vs containeiners . . . 18

3.1 Architettura di Room . . . 25
3.2 Implementazione del MVVM . . . 26
3.3 Struttura del modello di Garrett . . . 30

4.1 Flusso di autenticazione con JWT . . . 37
4.2 TLS - fase di handshake 1 . . . 39
4.3 TLS - fase di handshake 2 . . . 39
4.4 Architettura RecyclerView . . . 42
4.5 Launcher . . . 43
4.6 Eventi salvati . . . 46
4.7 App per la registrazione di un prodotto 1 . . . 48
4.8 App per la registrazione di un prodotto 2 . . . 48
4.9 App per la registrazione di un prodotto 3 . . . 48
4.10 Scansione barcode, QR code e tag RFID . . . 49
4.11 App per il tracciamento di un prodotto 1 . . . 50
4.12 App per il tracciamento di un prodotto 2 . . . 50
4.13 App per l'impacchettamento di prodotti . . . 52
4.14 App per la verifica di un ordine 1 . . . 54
4.15 App per la verifica di un ordine 2 . . . 54

Abbreviazioni

API

Application Programming Interface

APK

Android PacKage

ARM

Advanced RISC Machines

COO

Chief Operating Officer

CPU

Central Processing Unit

DVM

Dalvik Virtual Machine

EPC

Electronic Product Code

EPCIS

Electronic Product Code Information Services

GPS

Global Positioning System

GS1

Global Standards

GLN

Global Location Number

GTIN

Global Trade Item Number

GUI

Graphical User Interface

GZIP

GNU ZIP

HTTP

HyperText Transfer Protocol

IDE

Integrated Development Environment

JSON

JavaScript Object Notation

JWT

Json Web Token

MVVM

Model View ViewModel

OAS

OpenAPI Specification

OOP

Object Oriented Programming

OS

Operating System

RAM

Random Access Memory

REST

REpresentational State Transfer

RFID

Radio Frequency IDentification

SDK

Software Development Kit

SGLN

S Global Location Number

SGTIN

S Global Trade Item Number

SSL

Secure Sockets Layer

TLS

Transport Layer Security

UHF

Ultra High Frequency

UI

User Interface

URL

Uniform Resource Locator

UX

User eXperience

VM

Virtual Machine

Capitolo 1

Il problema dei dati nelle applicazioni

Le operazioni di raccolta, trasmissione e analisi dei dati, che ricoprono una significativa parte di lavoro all'interno di un'azienda, sono dei processi potenzialmente cruciali che possono portare a complicazioni non trascurabili per lo sviluppo e la buona riuscita di un progetto.

Riguardo lo sviluppo di un'applicazione, che essa sia mobile o web, le precedenti operazioni sono ancora più importanti se si vuole sviluppare un software affidabile e robusto, essendo un'applicazione dipendente da tali dati in molte fasi come durante quella di configurazione.

1.1 Operare con i dati

Ormai ci troviamo ad avere a che fare con dati ovunque. Queste moli di informazioni richiedono spesso grandi capacità di progettazione per risolvere in modo efficace ed efficiente il problema per il quale sono stati raccolti ed analizzati.

Esistono molti modi per raccogliere i dati e anche molti dispositivi elettronici capaci di farlo. Alcuni dati si possono estrapolare dalle informazioni già raccolte e spesso essere anche trasmessi per essere analizzati in un secondo momento, dando origine anche al problema dell' interpretazione. Ci sono diversi standard che aiutano quando si opera con i dati e le aziende devono capire quale fa al caso loro, per i propri scopi ed obbiettivi.

Per le aziende è molto importante evitare di commettere errori in queste fasi perchè una cattiva attività di raccolta, organizzazione ed interpretazione dei dati può portare ad una perdita di guadagno, quindi è obbligatorio prendere le giuste contromisure prima di imbattersi in problemi di questa natura [1].

Inoltre, occorre anche essere precisi nella raccolta, per partire fin da subito con i dati corretti e non imbattersi in errori ancor prima di operare. Per questo motivo, molte aziende si occupano di produrre dispositivi elettronici capaci di acquisire le informazioni nel modo migliore possibile. Si descrive nei capitoli successivi la tecnologia utilizzata in questo lavoro di tesi e quindi anche dei dispositivi elettronici impiegati nella lettura dei dati necessari.

1.2 I dati in un contesto logistico

Quando si parla di logistica, non si può non pensare a grandi quantità di merci sottoposte ad una serie di processi che richiedono una buona conoscenza dei dati trattati. Tra le principali operazioni riconducibili a questo contesto troviamo:

- Organizzazione delle operazioni e dei sistemi di trasporto e stoccaggio, per ottimizzare il flusso delle merci.
- Organizzazione e gestione dei centri di distribuzione per il ricevimento e lo smistamento delle merci.
- Monitoraggio delle consegne degli ordini ai centri di distribuzione e ai punti vendita.
- Monitoraggio delle consegne effettuate e dell'integrità dei prodotti.
- Organizzazione del sistema informatizzato degli ordini.
- Controllo dei costi delle operazioni.
- Problem solving relativo al trasporto merci.

Si intuisce che nelle operazioni elencate, se i dati vengono trattati nella maniera più consona, i processi risultano guadagnare in facilità e rapidità e queste sono caratteristiche ormai necessarie per tenersi al passo con i tempi. Molto esplicito a questo riguardo è stato Claudio Caremi, COO di OPTIT, che riguardo il tema della ottimizzazione della logistica di fabbrica, sottolinea il fatto che per le aziende che si vogliono affermare, essere rapidi nel supportare le decisioni con i dati è diventato un "must have" [2]. Questo è facilmente intuibile, pensando alla grossa quantità di prodotti presenti nei più grandi magazzini. Risulta impensabile svolgere questi processi nel modo corretto e in poco tempo senza una buona raccolta e un buon trattamento dei dati, utilizzando il giusto hardware e software studiati appositamente per operare in questo contesto.

1.3 Lo Standard EPCIS

Soprattutto in un contesto logistico, avere ben chiaro il ciclo di vita di un prodotto è molto importante. EPCIS è uno standard GS1 per il monitoraggio in tempo reale e permette di tracciare l'intera vita del prodotto su tutta la filiera. Nella Figura 1.1 è possibile vedere un esempio di tracciamento del prodotto lungo la supply chain.

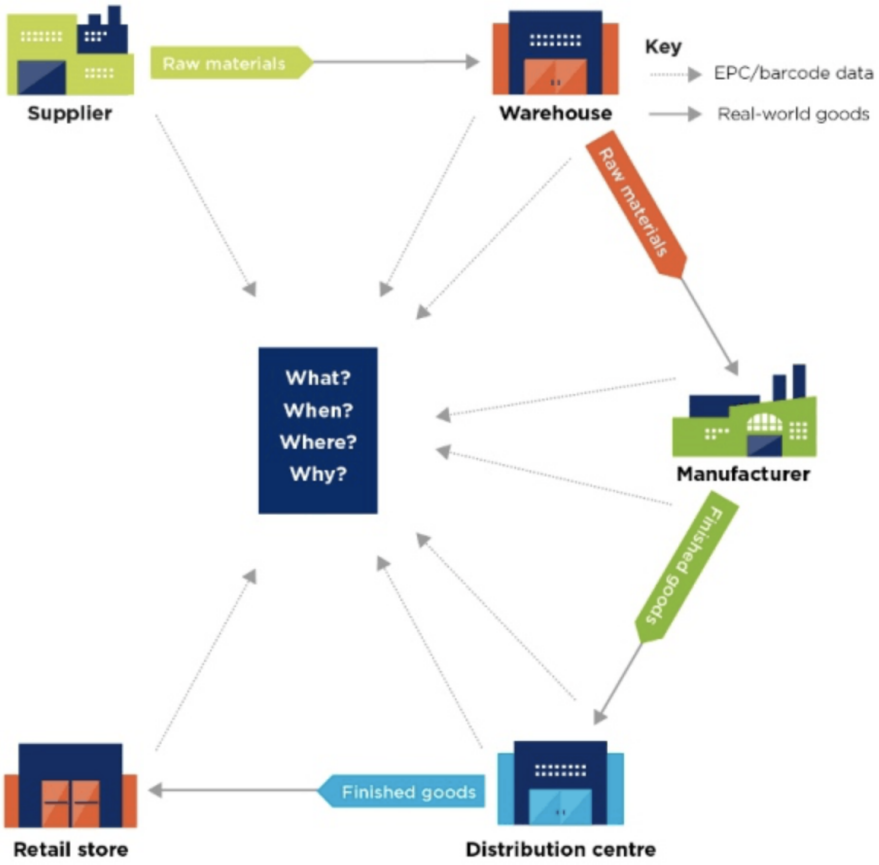

Figura 1.1: Tracciabilità delle merci

Per conoscere la storia di un particolare oggetto durante un determinato momento della sua vita, questo standard ricorre a quattro domande, rispondendo alle quali è possibile tracciare il prodotto. Queste domande sono:

- Cosa? (Gli identificatori dell'oggetto)
- Quando? (La data e l'ora in cui si è verificato l'evento)
- Dove? (L'identificativo del luogo in cui si è verificato l'evento)

- Perchè? (Informazioni sul contesto aziendale)

Per permettere di raccogliere ed associare i dati al prodotto considerato però è indispensabile che questo sia identificabile per mezzo di:

- GTIN + numero seriale.
- GTIN + lotto.

Il GTIN è la chiave GS1 usata in tutto il mondo per identificare i prodotti e tracciarli. Ne esistono di diversi tipi, in accordo con il tipo di prodotto da identificare.

- GTIN-8: codici di 8 cifre formati da codice prodotto e cifra di controllo, usati per identificare prodotti a peso fisso.
- GTIN-13: codici di 13 cifre formati da prefisso aziendale GS1, codice prodotto e cifra di controllo, usati per identificare prodotti a peso fisso, prodotti a peso variabile e unità imballo a peso fisso.
- GTIN-14: codici di 14 cifre formati da indicatore, GTIN-13 del prodotto contenuto senza cifra di controllo e cifra di controllo, usati per identificare unità imballo a peso fisso e unità imballo a peso variabile.
- SGTIN: codici generati a partire dal GTIN dell'unità commerciale e completato da un identificativo seriale che è il numero che l'azienda assegna univocamente al singolo pezzo, usati per identificare prodotti a peso fisso e unità imballo a peso fisso.

Il GLN è la chiave GS1 usata per identificare sedi legali e operative, reparti interni, magazzini, fabbriche ecc. Mentre l'SGLN serve per identificare univocamente un punto interno ad un luogo dell'azienda che è già identificato con un GLN.

In questa tesi, questo standard è stato usato ampiamente, dal momento che la maggior parte dei dati riguardanti i prodotti, sono trasmessi per essere analizzati attraverso degli eventi EPCIS. Le tipologie di eventi che lo standard mette a disposizione sono:

- Evento oggetto: gli oggetti specificati hanno partecipato a un evento.
- Evento di aggregazione: gli oggetti figlio specificati sono stati aggregati fisicamente a (o disaggregati da) l'oggetto padre specificato.
- Evento di trasformazione: gli oggetti di input specificati sono stati utilizzati e gli oggetti di output specificati sono stati prodotti.

- Evento di transazione: gli oggetti specificati sono stati definitivamente associati (o dissociati) con le transazioni commerciali specificate.

- Evento quantità (obsoleto): la quantità specificata della classe specificata ha partecipato a un evento. L'evento quantità è deprecato poiché è stato incluso nelle nuove funzionalità aggiunte all'evento oggetto.

Se implementato correttamente, questo standard può portare a molti benefici tra i quali migliorare e facilitare i processi logistici grazie ad informazioni accurate e gestione efficiente dell'inventario [3].

1.4 Configurazione delle applicazioni

Le applicazioni richiedono spesso un gran numero di dati per funzionare correttamente e svolgere le attività per le quali sono state sviluppate. Questi dati possono servire in punti diversi della vita di un'applicazione e possono essere recuperati da svariate fonti, come un database, uno o più file, un input dell'utente e così via.

Una delle prime fasi in cui servono sicuramente dei dati è la fase di configurazione, in cui vengono modificate le caratteristiche funzionali dell'applicazione dopo la sua installazione. Durante questa fase viene inizializzato il sistema ed è buona norma farla durare il minor tempo possibile, sia perchè potrebbe essere snervante aspettare per l'utente, sia perchè le funzionalità dell'applicazione potrebbero servire nell'immediato, come nel caso di un processo di un'azienda dove il tempo per svolgere un'attività potrebbe essere centellinato. Sorge quindi il problema di dover rendere questa fase il più breve possibile, ma d'altro canto si vuole avere un'applicazione ben configurata per garantire funzionalità e sicurezza.

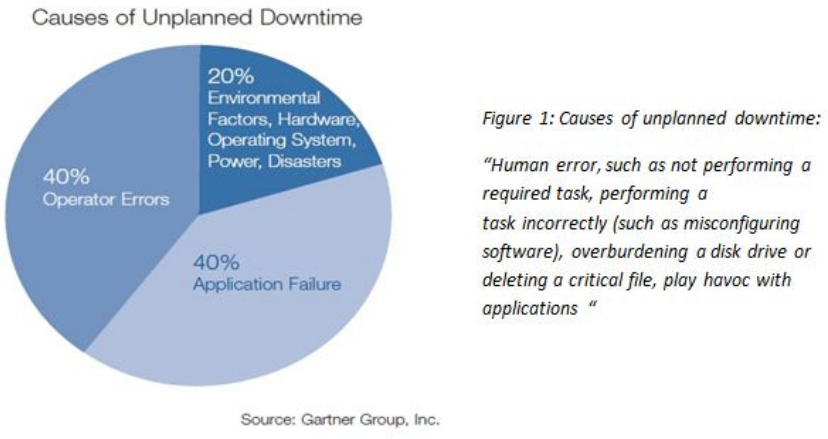

Figura 1.2: Cause di tempi di inattività non pianificati

Una delle operazioni che genera più errori e perdita di tempo in questa fase è far configurare il sistema direttamente a mano all'utente, inserendo per esempio input da tastiera, e se non sono dati che variano il sistema potrebbe configurarsi da se con maggiore rapidità e sicurezza. Nella Figura 1.2 si può vedere la conclusione tratta da studi che hanno valutato i tempi di fermo non pianificati quando si ha a che fare con un software [4], anche nella fase di configurazione. Molti di questi tempi sono dovuti ad errori da parte dell'uomo come è possibile notare, ricoprendo il 40% del grafico.

Un dato di fatto è che, al giorno d'oggi, ci si muove convintamente verso la direzione dell'automazione riducendo la necessità dell'intervento umano. Serve un primo sforzo per permettere alle macchine di svolgere da sole questo processo, ma se ben programmate i risultati sono migliori e soprattutto ci sono meno errori.

Questo è un punto chiave del quale si discute in questa tesi, fornendo una possibile soluzione al problema nel caso di applicazioni mobili in un contesto logistico.

Capitolo 2

Caratteristiche e specifiche di sistema

Prima di entrare in dettaglio nella spiegazione dell'implementazione svolta per questa tesi, è utile capire i principi e la tecnologia hardware e software che stanno dietro. Si è già detto nel primo capitolo, che l'idea base di questo lavoro è ottimizzare i processi produttivi per garantire la loro corretta esecuzione nel minor tempo possibile. Per fare questo, si è ricorso ai prodotti dell'azienda Zebra, produttrice di dispositivi elettronici di ottima qualità, utilizzati da molte altre compagnie in tutto il mondo. Il software sviluppato appositamente per sistemi Android, segue il principio di ricorrere solo alle funzionalità necessarie volta per volta, infatti in seguito viene trattata l'installazione dinamica e come essa aiuta anche a non appesantire i dispositivi senza motivo e a velocizzare i tempi, in un processo di autoconfigurazione dei dispositivi. Si fa anche una breve introduzione degli elementi del backend con cui comunicano le applicazioni mobili sviluppate e dello scenario di operazione.

2.1 Android

Dal momento che tutto il software sviluppato deve convivere in dispositivi con SO Android, è opportuno capire le caratteristiche base che permettono il funzionamento delle applicazioni al suo interno. Android è un sistema operativo per dispositivi mobili sviluppato da Google. Viene utilizzato principalmente in sistemi embedded come smartphone e tablet, ed è possibile trovarlo anche in TV, automobili, orologi, occhiali e altri. Un altro dispositivo con SO Android è il TC20, che viene introdotto in seguito. Le applicazioni Android sono scritte principalmente nei linguaggi di programmazione come Kotlin e Java e usano lo stesso SDK.

2.1.1 Architettura Android

Android ha un'architettura gerarchica con strutturata a layer come mostrato nella Figura 2.1.

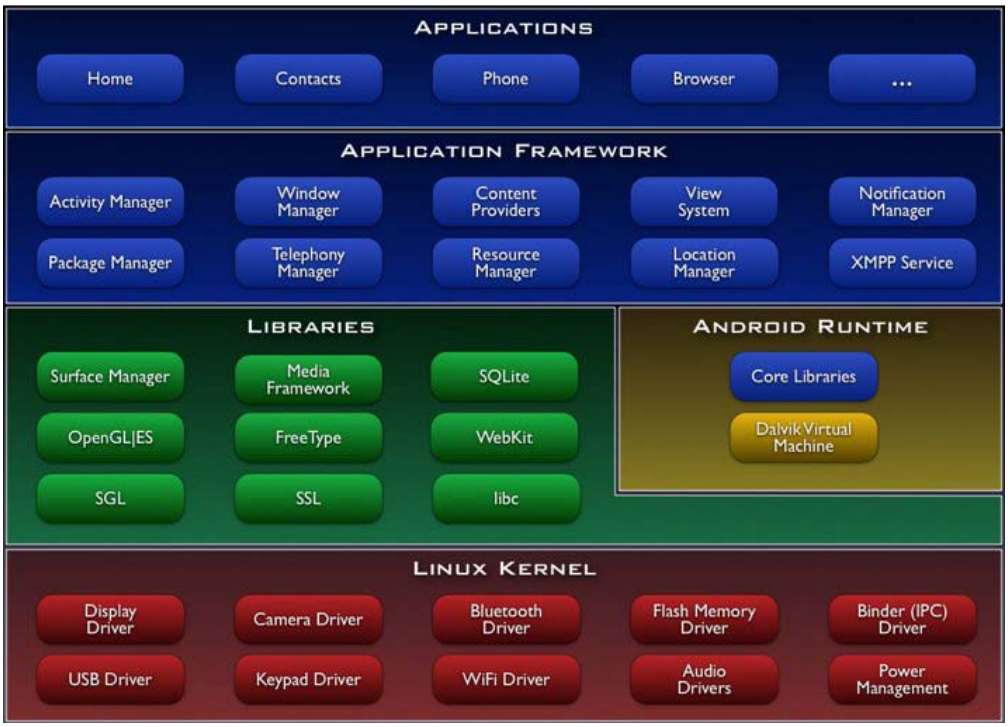

Figura 2.1: Architettura Android

I layer comprendono un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, una implementazione della VM e un insieme di librerie Java.

Il livello più basso è rappresentato dal kernel Linux che gestisce le periferiche multimediali, il display, le connessione Wi-Fi e Bluetooth, l'alimentazione, il GPS, la fotocamera ecc. La scelta di un kernel Linux si spiega attraverso la necessità di avere un'alta affidabilità per esempio, per garantire il servizio di telefonia. Gli utenti si aspettano l'affidabilità, ma allo stesso tempo vogliono un dispositivo che possa garantire servizi più evoluti e Linux permette di raggiungere entrambi gli scopi.

Salendo nella gerarchia troviamo un insieme di librerie native scritte C e C++ e l'Android Runtime. Quest'ultimo è costituito dalle librerie core Java e dalla DVM, una Virtual Machine ottimizzata per sfruttare la poca memoria dei dispositivi mobili e di cui ogni applicazione dispone di un'istanza. Le librerie invece fanno

parte di un insieme di progetti Open Source che aiutano nella gestione di tutto il sistema.

Al prossimo livello si trova l'Application Framework, costituito da un insieme di componenti che utilizzano le librerie native. Si tratta di un insieme di API e componenti per l'esecuzione di funzionalità di base del sistema Android.

Al livello più alto si trova il layer delle applicazioni native. Le funzionalità base del sistema, come per esempio il telefono, sono delle applicazioni scritte in Java e che girano ognuna nella sua VM. A questo livello verrano installate le app create dall'utente o scaricate dal Play Store.

2.1.2 Applicazioni Android

Un'applicazione Android è costituita da un insieme di dati e codice progettato per eseguire una o più operazioni. Queste informazioni consistono in uno o più componenti all'interno di un software "package" (APK), descritti nel Manifest file. Il Manifest file raccoglie informazioni basilari sull'app, informazioni necessarie al sistema per far girare qualsiasi porzione di codice della stessa e ogni applicazione ne deve avere uno. Ci sono quattro componenti essenziali che costituiscono un'applicazione Android:

- Activity: un componente dell'applicazione che fornisce una schermata con cui gli utenti possono interagire. Una normale applicazione consisterà in una sequenza di activity. Quindi, ci sono diverse schermate che si alternano sul display comunicando eventualmente tra di loro e scambiandosi delle informazioni.

- Service: un componente dell'applicazione atto ad eseguire operazioni a lungo termine in background, senza fornire alcuna interfaccia utente.

- Content Provider: un gestore dell'accesso ad un set di dati strutturati. Ogni applicazione può definire una o più tipologie di dati e rendere poi disponibili tali informazioni esponendo uno o più Content Provider. Invece, il sistema la metterà in contatto con il corrispondente Content Provider precedentemente installato se vorrà richiedere l'accesso ad un particolare tipo di dato.

- Broadcast Receiver: un componente che gestisce eventi di broadcast lanciati dal sistema. Il sistema genera molti messaggi di broadcast e tale funzionalità è svolta anche dalle applicazioni. I Broadcast Receiver (come i Service) non mostrano un'interfaccia utente, ma possono creare una notifica sulla status bar per avvertire l'utente che un determinato evento è avvenuto.

Alcuni di questi componenti possono comunicare tra loro attraverso degli Intent. L'Intent è un meccanismo di messaggi che può attivare tre dei componenti di base

di un'applicazione: activity, service e broadcast receiver. L'oggetto Intent è una struttura dati passiva che contiene una descrizione astratta dell'operazione da eseguire. Mandare un Intent è uno dei primi passi per far partire un'applicazione Android. Per avere un'idea migliore del processo di avvio un'app Android, di seguito sono riportati i passi principali:

- Con un tap dell'utente sull'icona dell'applicazione, le informazioni sull'activity scelta sono racchiuse in un Intent, che è poi consegnato ad un processo di sistema denominato Zygote, che si occupa di creare tutti gli altri processi tramite fork.
- Zygote inizializza una primissima istanza di macchina virtuale Dalvik, precaricata con tutte le classi comuni utilizzate dal framework dell'applicazione Android, il che rende il tempo di avvio più veloce, perché solo le classi specifiche dell'applicazione devono essere istanziate.
- Il processo creato da Zygote avvia un loop di messaggi e utilizzando l'Intent individua e carica l'apk corretto e il file manifest.
- Il thread principale del processo creato istanzia un oggetto Applicazione che crea un'istanza dell'activity principale. Il suo metodo OnCreate() è invocato e l'applicazione inizia a funzionare, popolando una GUI e mostrandola.

2.1.3 Android Studio

Android Studio è un IDE per lo sviluppo di applicazioni Android. Questo IDE, l'unico usato per lo sviluppo delle applicazioni di questa tesi, è organizzato in diversi pannelli che possono essere riorganizzati a piacere dall'utente, in modo tale da avere il necessario quando serve. Tra i pannelli che mette a disposizione si trovano:

- Finestra del progetto che fornisce una panoramica gerarchica del progetto considerato.
- Finestra dell'editor che visualizza un editor sensibile al contesto per il file selezionato dalla scheda.
- Barra di stato che visualizza il risultato della compilazione oppure i messaggi del registro di esecuzione.
- Barra degli strumenti che fornisce una selezione di scorciatoie per azioni eseguite di frequente.
- Barre della finestra degli strumenti che consentono di individuare rapidamente ulteriori finestre di ispezione del progetto.

Android Studio permette anche di scaricare diversi emulatori con diverse versioni di sistema operativo Android, per testare le app che si stanno sviluppando su diversi dispositivi virtuali se l'applicazione deve girare su una vasta gamma di prodotti. Dispone di un editor di codice intelligente che fornisce il completamento del codice per i linguaggi principali Android e permette anche di analizzare diversi APK, anche non prodotti con Android Studio, per avere un'idea delle dimensioni dell'app [5].

2.2 Dispositivi elettronici utilizzati

Come introdotto sopra, i dispositivi elettronici utilizzati provengono dall'azienda Zebra. Questa compagnia, fondata nel 1969, ha più di 4.400 brevetti. Tra i prodotti che offre è possibile trovare scanner, computer portatili, tablet e stampanti aziendali. Per il lavoro svolto, si è optato per due dispositivi particolari, di cui si approfondisce la trattazione, il TC20 e il RFD2000.

2.2.1 TC20

Il TC20 è un computer portatile costruito per resistere ad ambienti di lavoro impegnativi, come appunto i magazzini. Questo dispositivo è fornito di scansione di codici a barre integrata e anche se il design ricorda quello di uno smartphone, è un vero e proprio scanner. Nella Figura 2.2 è possibile vedere come si presenta esteticamente il TC20. Per quanto riguarda alcune caratteristiche tecniche [6], questo computer ha :

- CPU basata su processore ARM Cortex A53.
- memoria RAM 2GB – Flash 16GB.
- OS Android ver. 7.1.2 Nougat.
- Dimensioni 134 mm (lungh.) x 73,1 mm (largh.) x 16 mm (prof.).
- Schermo da 4,3 pollici.
- Lettore di codici a barre Imager SE4710 1D / 2D integrato.
- Fotocamera a colori con messa a fuoco automatica da 8 MP con flash.

Per l'obbiettivo della tesi, questo dispositivo è usato soprattutto per la raccolta e la trasmissione dei dati, ed infatti, è capace di raccogliere dati in maniera differente, anche grazie alla sua fotocamera. Nello specifico, i suoi componenti maggiormente usati per eseguire le scansioni in questo lavoro sono:

Figura 2.2: Mobile computer TC20

- Lettore di codice a barre integrato per leggere i codici a barre.
- RFD2000 (collegabile al TC20) per leggere i tag RFID.

2.2.2 RFD2000

RFD2000 è una culla palmare per il computer descritto in precedenza. Questo dispositivo è progettato per dotare il TC20 di funzionalità RFID UHF e fornisce le stesse prestazioni RFID di un dispositivo dedicato. Nella Figura 2.3 si nota graficamente il dispositivo RFD2000. L'ergonomia e il design compatto e leggero sono stati pensati per offrire comfort nelle attività di gestione dell'inventario, quindi in questo progetto facilita il compito degli operatori che lo dovranno utilizzare. Inoltre, collegarlo al TC20 è davvero semplice grazie ad un sistema ad incastro e un collegamento elettrico a 8 pin. Alcune delle sue caratteristiche tecniche [7] sono:

- Peso 310 grammi.
- Velocità di lettura fino a 700 tag / sec.
- Dimensioni 14,9 cm alt. x 7,9 cm largh. x 13,3 cm lungh.
- Motore RFID con tecnologia radio proprietaria di Zebra.

Figura 2.3: Culla palmare RFID UHF RFD2000

Esiste anche un'applicazione per il TC20 che fa da manager per questa culla. Grazie a questa applicazione, è possibile verificare la carica della batteria, lo stato di connesione e di attività del RFD2000.

2.3 Funzionalità delle applicazioni

Lo scopo di questo lavoro, è garantire agli operatori delle applicazioni performanti che migliorino i processi logistici che si attuano tutti i giorni. Per fare questo, le applicazioni devono essere pensate per risolvere i problemi introdotti nel primo capitolo. Il software sviluppato deve:

- Configurare il sistema automaticamente dove possibile, evitando l'intervento umano.
- Installare dinamicamente le applicazioni che servono in quel frangente, non sprecando spazio di archiviazione e tempo.
- Funzionare al meglio sull'hardware su cui risiede, essendo sviluppato appositamente per i dispositivi che saranno utilizzati.
- Essere ottimizzato, ma anche facile da utilizzare per gli utenti.
- Svolgere al meglio i processi logistici per i quali è stato pensato.

Queste sono alcune delle caratteristiche specificate all'inizio di questo progetto, e quindi quelle su cui si è lavorato per garantire la corretta implementazione. Come specificato nel capitolo precedente, l'autoconfigurazione del sistema è un punto chiave per questo lavoro.

Viene trattata nel dettaglio in seguito, ma come è possibile pensare, anche l'installazione dinamica fa parte di questa fase. Il software, infatti, deve capire di quali applicazioni i dispositivi hanno bisogno in quel determinato momento del loro utilizzo e provvedere ad installarle nel breve tempo possibile. Si evita di installare direttamente tutte le applicazioni perchè così facendo si aumenterebbero i tempi d'attesa e per le aziende questa non è una buona cosa, dal momento che quel dispositivo potrebbe essere dedicato per uno specifico processo ed avere all'interno tutto il software non avrebbe molto senso oltre ad essere anche uno spreco di spazio.

Le applicazioni con il compito di raccogliere i dati, quindi tag RFID, codici a barre e QR code, devono essere ottimizzate per i dispositivi TC20 e RFD2000. Infatti, per queste specifiche operazioni, sono state usate rigorosamente le librerie ed il software messi a disposizione da Zebra per programmare al meglio i propri dispositivi.

Si è tenuto presente anche lo scenario in cui si lavora, cercando di garantire il massimo rendimento, in modo facile e col minimo sforzo da parte degli operatori che dovranno utilizzare i dispositivi, quindi l'utilizzo delle applicazioni deve essere di facile comprensione.

Inoltre, la collaborazione con il backend è stata fondamentale in quanto i dispositivi mobili comunicano con le sue componenti, per svolgere le operazioni indispensabili ad attuare i processi logistici che i dispositivi non potrebbero portare a termine da soli.

2.4 Backend del sistema

Con il termine backend, si identifica l'insieme delle applicazioni e dei programmi essenziali al funzionamento del sistema, ma di cui l'utente non conosce le caratteristiche, non essendo queste influenti su l'uso corretto dell'applicazione considerata. Risulta essere comune, soprattutto nelle applicazioni che hanno a che fare con la rete, avere un sistema client-server, di cui si può vedere un esempio in Figura 2.4, in cui il client ha il compito di richiedere le informazioni e il server (con tutto il sistema di backend) quello di fornirgliele se lo ritiene opportuno. Anche se il backend di questo sistema non è lavoro di questa tesi, è opportuno fare una breve introduzione per capire con cosa interagiscono le applicazioni mobili sviluppate, partecipando al funzionamento di questo sistema.

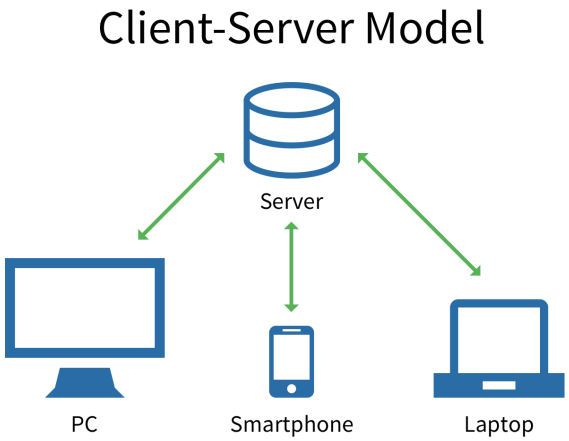

Figura 2.4: Esempio di modello client-server

2.4.1 Il server

Il server di questo progetto, si interfaccia col client attraverso delle API REST, ovvero un set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi conformi ai vincoli dell'architettura REST, note anche come API RESTful. Nella Figura 2.5 si può vedere un esempio di architettura che utilizza queste API. REST è uno stile architetturale usato nei sistemi distribuiti e rappresenta un sistema di trasmissione di dati basato su HTTP, che non supporta il concetto di sessione. Viene utilizzato con una struttura URL ben definita che identifica in modo univoco una risorsa o un gruppo di risorse e utilizza i metodi HTTP specifici per:

- Il recupero di informazioni (GET).
- La modifica (POST, PUT, PATCH, DELETE).
- Altri scopi (OPTIONS, ecc.).

Quando una richiesta viene inviata tramite un'API RESTful, questa trasferisce al richiedente uno stato rappresentativo della risorsa. L'informazione, o rappresentazione, viene consegnata in uno dei diversi formati tramite HTTP. Il formato JSON è uno dei più diffusi, perché indipendente dal linguaggio e facilmente leggibile da persone e macchine.

Il server, oltre a comunicare con il database per salvare, modificare e cancellare dati, ha il compito di elaborare gli eventi EPCIS che arrivano dai dispositivi mobili, per collaborare nell'esecuzione dei processi logistici.

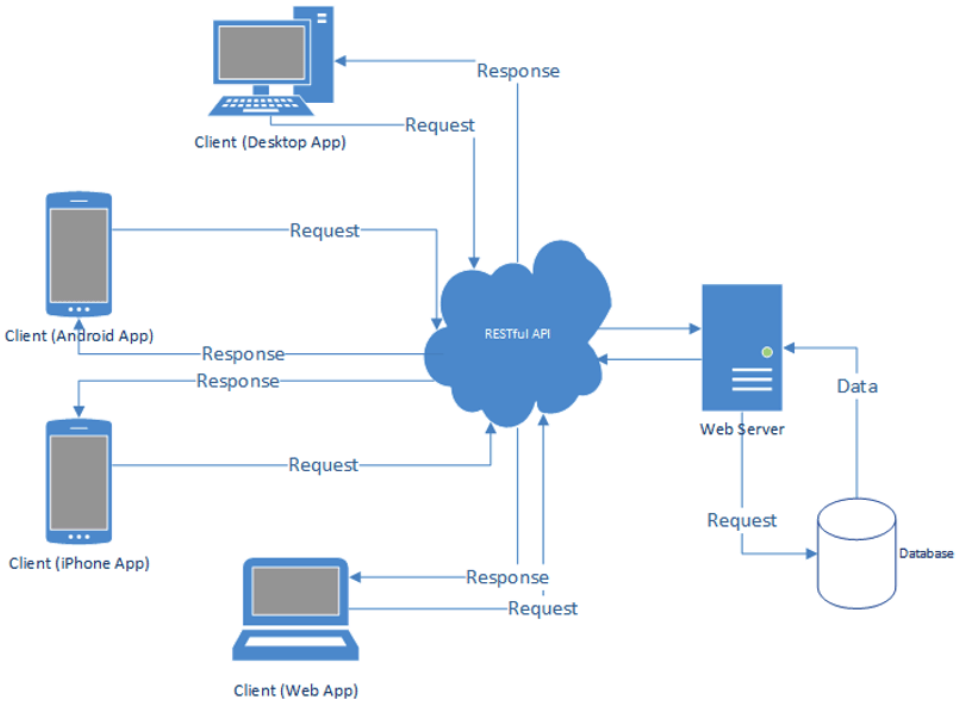

Figura 2.5: Esempio di architettura delle API RESTful

2.4.2 Il database

Il database utilizzato nel sistema di backend è MongoDB, un database distribuito che negli ultimi anni si è affermando soprattutto in progetti come quello trattato. MongoDB è un database document-based, cioè che archivia i dati sotto forma di documenti di tipo JSON [8]. Questa sua caratteristica è molto utile in questo contesto, perchè come si è detto, in questa tesi si lavora molto con i dati in formato JSON. Di seguito sono riportati alcuni vantaggi nell'utilizzare questo tipo di database:

- Latenza inferiore per query (Nessun join, transazioni). Quindi, può gestire un carico maggiore in termini di query al secondo.
- Molto scalabile grazie allo sharding, un metodo per distribuire i dati su più macchine, consentendo il ridimensionamento orizzontale.
- Maggiore velocità di scrittura perché non deve preoccuparsi di bloccare in caso di errori (No transazioni, rollback).
- Molto flessibile in quanto non ha uno schema.
- Esiste molta documentazione e una grande community.

Si ricorda quindi che si deve progettare bene il sistema per prevenire gli errori dovuti al fatto che MongoDB non supporta le transazioni.

2.4.3 L'utilizzo di Docker

Docker è una piattaforma software che consente di creare, testare e distribuire applicazioni alla massima velocità [9]. Docker permette di raccogliere il software in unità chiamate container, che forniscono tutto il necessario per una corretta esecuzione, incluse librerie, strumenti di sistema, codice ecc. In questo modo, si possono distribuire e ricalibrare le risorse delle applicazioni in qualsiasi ambiente controllando sempre il codice in esecuzione. Docker è installato sul server e fornisce semplici comandi con cui creare, avviare o interrompere i container. Nella Figura 2.6 si vede la differenza tra VMs e containers. Si nota come Docker permette di avere ambienti di esecuzione piccoli e leggeri che fanno uso condiviso del kernel del sistema operativo ma che si eseguono in modo isolato l'uno dall'altro.

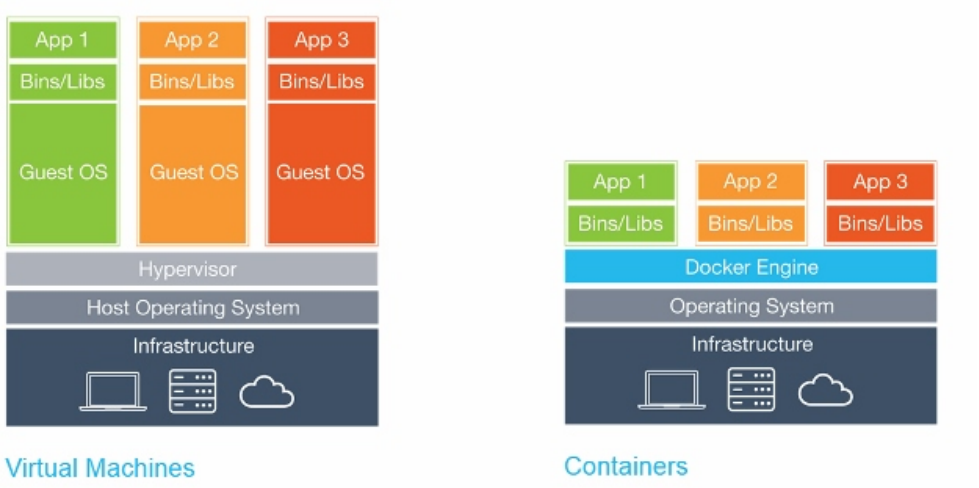

Figura 2.6: Virtual machines vs containeiners

In questo progetto, è usato per eseguire alcune applicazioni, come il database di cui si è parlato che all'avvio del server viene fatto partire in un container. Tra i benefici di usare i Docker container ci sono:

- Semplificare il deployment di tutte le applicazione.
- Garantirne il funzionamento in ogni ambiente.
- Ottima portabilità.
- Le applicazioni e le risorse sono isolate, perché ogni container ha risorse isolate da quelle degli altri.

- Maggiore sicurezza in quanto le app eseguite sui container sono completamente separate le une dalle altre.

2.4.4 Swagger come strumento di supporto

Swagger è un insieme di tool che seguono l'OAS (uno standard utilizzato per descrivere le API Rest) per creare e documentare le API [10]. Lo standard OAS è indipendente dai linguaggi di programmazione ed è concepito per essere compreso da esseri umani e computer. Lo scopo è rendere il contenuto fornito dal servizio più facile da capire senza dover accedere al codice sorgente o ad altri documenti. Quindi è possibile vedere cosa risponde il server ad una determinata richiesta senza implementare una funzione client apposita e realizzarla in seguito, quando si è capito come interagire con quella specifica API. Tutto questo è stato molto utile perchè ha permesso di comprendere il funzionamento del sistema prima di tuffarsi ad occhi chiusi nello sviluppo delle applicazioni mobili, avendo una chiara visione d'insieme e permettendo di realizzare le funzionalità richieste dalle diverse API.

2.5 Analisi dello scenario di operazione

Una caratteristica, la cui importanza non si deve sottovalutare, è lo scenario in cui le applicazioni sviluppate sono utilizzate. Un'applicazione mobile può essere ottimizzata, sicura, svolgere in modo impeccabile delle operazioni, ma tutto questo non serve a nulla se l'utente finale non capisce il suo funzionamento o non riesce ad utilizzarla per cause non riconducibili a lui personalmente. Nel contesto in cui il lavoro di questa tesi propone una soluzione, l'utente finale è un operaio che svolge dei processi logistici. Si è già discusso di come i dispositivi hardware siano pensati appositamente per scenari di questo tipo, come il lavoro in un magazzino, e anche il software non deve essere da meno. Si nota che l'utilizzatore delle applicazioni potrebbe non essere abituato ad utilizzare dei dispositivi mobili, e la presenza di tanti bottoni, menù, o in generale di componenti interattivi, potrebbe scoraggiarlo e rendere il suo lavoro più complicato. Un altro aspetto cruciale, è il fatto che l'utente in questione sta svolgendo un lavoro molto fisico, in posti polverosi e sotto pressione, quindi la sua interazione con l'app dovrebbe essere facilitata in qualche modo. Per risolvere questi problemi sono state fatte delle scelte che migliorano la UX. Tra le caratteristiche pensate sono presenti:

- Ridurre al minimo la complessità delle applicazioni con la giusta quantità di elementi interattivi, se un'operazione si può svolgere con la pressione di un bottone, non ha senso metterne di più.

- Le dimensioni dei vari componenti grafici devono facilitare il loro utilizzo, se l'operaio è sotto pressione ed affaticato non riuscirà mai a premere un bottone molto piccolo sullo schermo.
- I dispositivi elettronici dispongono già di alcuni componenti interattivi, non ha senso mettere altri nel software che hanno lo stesso scopo.

Queste sono solo alcune delle accortezze che sono state prese per facilitare l'utilizzo delle applicazioni, richiedendo poco impegno per svilupparle a livello di codice, ma essenziali per rendere le applicazioni usabili e di facile comprensione.

Capitolo 3

Sviluppo e progettazione

In questo capitolo sono riportate le principali scelte progettuali che hanno portato allo sviluppo del software. In particolare sono riportate le scelte riguardanti i linguaggi di programmazione, le decisioni a livello di struttura del progetto, le caratteristiche dell'autoconfigurazione dei dispositivi e come viene sviluppata e resa possibile.

3.1 Java e Kotlin

Quando si deve progettare un'applicazione Android, una delle decisioni che può migliorare incredibilmente lo sviluppo della stessa, è la scelta del linguaggio di programmazione da utilizzare. I due linguaggi principali sono Java, affermato già da molti anni, e Kotlin che negli ultimi anni sta scalando la classifica, portando i programmatori Android ad una difficile scelta tra i due quando devono iniziare a programmare un app. Per capire meglio i punti di forza e i difetti dei due linguaggi, di seguito sono riportate delle descrizioni di entrambi.

Java è utilizzato come linguaggio di programmazione dalle grandi imprese. Lo sviluppo Java domina il mondo delle applicazioni aziendali, grazie alle sue funzionalità come l'interoperabilità e la flessibilità. Tra i motivi principali del suo utilizzo ci sono il paradigma di architettura OOP e la robustezza del codice, rendendolo presente praticamente ovunque, come nei pc portatili, nelle console per videogiochi, negli smartphone, ecc. La programmazione in Java ha i seguenti vantaggi:

- È facile da imparare e da capire.
- È flessibile, cioè è eseguibile sia in rete, sia in un ambiente di esecuzione virtuale. Questo è utile quando si riutilizza il codice e si aggiorna il software.

- È parte integrante del kit di sviluppo Android, che contiene molte librerie Java per la creazione di app compatibili con il sistema operativo.
- Ha un grande ecosistema open source, aggiornato e ricco di funzionalità.
- Lo sviluppo Java consente di gestire progetti di grandi dimensioni e complessi, con l'aiuto di strumenti moderni che garantiscono la massima velocità di compilazione e di realizzazione.

Kotlin è un linguaggio di programmazione open-source, progettato dai programmatori di JetBrains nel 2011 e costantemente aggiornato dalla stessa community, compatibile con Java, pensato proprio per risolvere alcuni problemi dello sviluppo Java. Questo lo ha portato ad essere preferito da molte aziende, tra queste c'è anche Google. Tra i vantaggi di utilizzare Kotlin, ci sono:

- È un linguaggio di programmazione più conciso e rapido rispetto a Java, cioè è possibile scrivere meno codice che con Java.
- Ha la capacità di interagire e funzionare con lo stesso rivale Java, quindi è interoperabile
- Ha un'architettura sicura. Questo consente, a differenza di Java, di non incorrere in errori di codice, come il più comune che gli sviluppatori riscontrano in Java, il NullPointerException.
- È un linguaggio di programmazione tipizzato e questo aumenta la velocità di esecuzione, per esempio delle funzioni Lambda.

In questo progetto di tesi, sono stati usati entrambi i linguaggi, prendendo il meglio e sfruttando i vantaggi dei due. Per esempio, si è sfruttata la fama di Java con le librerie di Zebra scritte esclusivamente in Java, nonchè la sua flessibilità, mentre per quanto riguarda Kotlin si è fatto uso delle coroutine, soprattuto in fase di test e di monitoraggio delle applicazioni. Le coroutine sono blocchi di codice che vengono eseguiti in modo asincrono senza bloccare il thread dal quale vengono avviati.

3.2 Build di progetti Android complessi

Le applicazioni Android richieste e necessarie per questo progetto sono più di una, basta pensare alla varietà di processi che bisogna gestire in un contesto logistico. Un'idea non molto carina, almeno a livello di struttura, è quella di avere un progetto di Android Studio per ogni app sviluppata. Questa idea è stata subito abbandonata perchè ogni app ha delle caratteristiche in comune con le altre app,

come per esempio l'autenticazione o il recupero della propria configurazione, che a livello di codice sarebbe identico in ogni progetto. Riflettendo su quanto detto, si è optato per creare delle varianti della stessa app, in quanto la parte diversa consiste nell'esecuzione del processo che l'app deve svolgere, spesso tradotto in una o comunque poche activity diverse.

Per avere delle versioni diverse della stessa app nello stesso progetto, si è ricorso alla configurazione delle varianti di build. Le varianti di build sono un insieme specifico di regole per combinare impostazioni, codice e risorse configurate nei tipi di build e nelle versioni di prodotto (flavor). Non si configurano direttamente le varianti di build, si configurano i tipi di build e le versioni di prodotto che le formano. Una versione di prodotto specifica diverse caratteristiche e requisiti del dispositivo, come codice sorgente personalizzato, risorse e livelli API minimi, mentre il tipo di build applica diverse impostazioni di build e packaging, come opzioni di debug e firma chiavi. Si possono creare e configurare i tipi di build e i flavor nel build.gradle file all'interno del blocco android del progetto. Dopo aver effettuato la sincronizzazione, Gradle (un sistema open source per l'automazione dello sviluppo) crea automaticamente varianti di build in base ai tipi di build e ai flavor dei prodotti e le denomina in base a <flavor><tipo di build>. Si possono anche applicare dei filtri se si vuole che un determinato tipo di build non abbia un flavor, oppure viceversa, che un flavor non abbia un tipo di build.

Tutte queste funzionalità, hanno portato a creare un flavor denominato "main" in cui ci sono tutte le caratteristiche comuni a tutte le app e dei flavor diversi per ogni app sviluppata con la logica distinta per quella particolare app. In questo modo, quando una determinata variante di build è eseguita, viene eseguito il codice presente in "main", con le differenze che però sono presenti nel flavor di quella variante. Quindi è possibile testare e creare gli APK di tutte le app avendo solo un progetto che gestisce tutti processi logistici.

3.3 L'uso di Room

I dispositivi elettronici utilizzati devono essere collegati alla rete per poter comunicare con il server. La connessione però non è sempre garantita, soprattutto nello scenario considerato, e si vuole che i dispositivi continuino comunque a funzionare, avendo certamente delle limitazioni. Per questo motivo, si è deciso di utilizzare Room. Room è un database locale che fornisce un livello di astrazione su SQLite, usato soprattutto per memorizzare parti di dati rilevanti in modo che quando il dispositivo non è connesso alla rete, l'utente può comunque navigare in quel contenuto mentre è offline. Room è composto principalmente da tre componenti:

- La classe di database che contiene il database e rappresenta il punto di accesso principale per la connessione sottostante ai dati persistenti dell'app.

- Entità di dati che formano le tabelle nel database dell'app.
- Oggetti di accesso ai dati che forniscono metodi per eseguire le query nel database.

Nella Figura 3.1 si può notare graficamente l'architettura di Room.

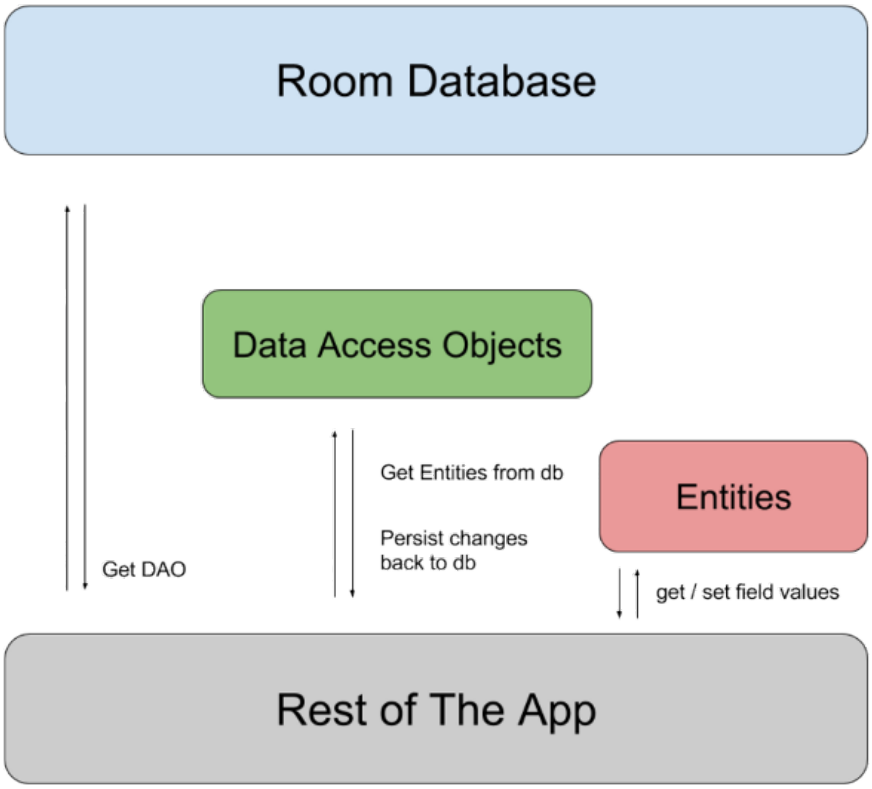

Figura 3.1: Architettura di Room

In questo progetto sono state create due entità:

- ItemCode che serve per salvare i codici dei prodotti che si devono registrare, per poi utilizzarli in un secondo momento quando servono senza doverli richiedere al server.
- ReadingEvent che viene utilizzata per salvare gli eventi EPCIS se la connessione alla rete viene a mancare, per mandarli al server al ritorno della connessione.

Le query eseguibili si trovano rispettivamente in ItemCodeDao e in ReadingEventDao, mentre l'istanza del database è la stessa.

3.4 Il pattern Model-View-ViewModel

I diversi componenti che formano un'applicazione Android possono essere avviati e distrutti in un qualsiasi momento, indipendentemente dagli altri. Quindi, è necessario che nessuno di essi memorizzi al proprio interno dei dati dell'applicazione e nessuno dovrebbe dipendere dall'esistenza di un altro componente. Il contenuto della UI deve riflettere un modello persistente in modo tale che gli utenti non perdano i dati se l'app viene distrutta o si ha a che fare con connessioni di rete problematiche. Tra i diversi modelli esistenti, si è scelto di utilizzare il MVVM. I principali attori nel pattern MVVM sono:

- Model, che contiene tutte le classi per ottenere e salvare i dati.
- View, che informa il ViewModel sulle azioni dell'utente e rappresenta lo stato corrente delle informazioni visibili.
- ViewModel, che espone flussi di dati rilevanti per la View. ViewModel e View sono collegati tramite Databinding e LiveData osservabili.

Nella Figura 3.2 si può vedere una delle possibili implementazioni del MVVM.

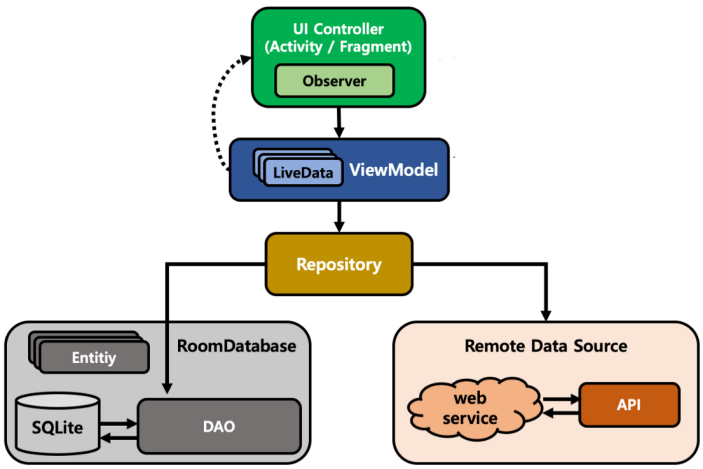

Figura 3.2: Implementazione del MVVM

Come è possibile notare, le View sono di solito rappresentate da Activity o Fragment (porzione di Activity) e hanno il compito di osservare i LiveData del ViewModel. LiveData è una classe di dati osservabile che avvisa il suo osservatore quando i dati sottostanti cambiano. Vengono aggiornati solo gli osservatori nei componenti che si trovano in uno stato del ciclo di vita attivo. I ViewModel recuperano i dati dalla repository che fornisce un accesso ai dati memorizzati in

un server remoto o in una memoria locale (come Room di cui si è già parlato). MVVM ha il vantaggio della separazione delle preoccupazioni e sfrutta i vantaggi delle associazioni di dati. Il risultato è un modello che guida il maggior numero di operazioni possibile, riducendo al minimo la logica nella vista. In questo progetto si sono utilizzati due repository e due ViewModel, nel dettaglio:

- ItemCodeRepository e ItemCodeViewModel, rispettivamente per avere accesso ai codici dei prodotti memorizzati e avere queste informazioni nel giusto ViewModel per osservarne il cambiamento nelle activity/fragment dell'app.

- ReadingEventRepository e ReadingEventViewModel, rispettivamente per avere accesso agli eventi EPCIS memorizzati e avere queste informazioni nel giusto ViewModel per osservarne il cambiamento nelle activity/fragment dell'app.

3.5 Autoconfigurazione dei dispositivi

Come già anticipato, questo lavoro verte molto sul concetto di autoconfigurazione. Questo processo deve facilitare e velocizzare il compito degli operatori finali che utilizzano i dispositivi, e garantire meno errori dovuti all'intervento umano. Tra la miriade di sorgenti da cui recuperare i dati per la configurazione, si è optato per un file JSON.

Un file JSON è un file che memorizza semplici strutture dati e oggetti in formato JSON, che è un formato di scambio dati standard. Viene utilizzato principalmente per la trasmissione di dati tra un'applicazione e un server, buona scelta dal momento che i dispositivi sono connessi ad una rete capace di comunicare con un server. I file JSON sono leggeri, basati su testo e facilmente leggibili dall'uomo. Il file di configurazione deve contenere:

- Informazioni per risalire al suddetto file, come id, nome e versione.

- Le credenziali per ottenere l'accesso alle applicazioni.

- L'indirizzo ip del server a cui fare le richieste.

- La lista delle applicazioni da scaricare ed installare.

Questo file, viene salvato nel dispositivo che ha richiesto di essere configurato e grazie ad un' apposita procedura, è possibile verificare la presenza di nuovi file di configurazione. Comunque, si approfondisce questo aspetto a livello di progettazione e codice implementato in seguito, quando si descrive l'applicazione launcher sviluppata appositamente.

3.5.1 Installazione dinamica delle applicazioni

Una delle operazioni principali, durante la fase di autoconfigurazione, è l'installazione dinamica delle applicazioni. Dietro questa operazione, c'è l'idea di avere solo le applicazioni necessarie sul dispositivo avendo benefici di spazio e di tempo. Dal file JSON di configurazione è possibile ottenere la lista degli APK, precedentemente caricati sul server, che devono essere installati sul dispositivo in questione. In particolare, ogni applicazione da scaricare ed installare presente nella lista è rappresentata come un oggetto JSON, ognuno dei quali è composto da:

- Id dell'applicazione.
- Nome dell'applicazione.
- Il template dell'evento EPCIS che l'applicazione deve poter generare per il processo che deve sostenere.

Per come è stato pensato il server con cui comunicano le applicazioni, per evitare di generare ogni volta un evento EPCIS completo, sono stati creati dei template specifici per ogni processo aziendale da supportare. Nel template che l'applicazione deve trattare, ci sono dei dati fissi, come il nome del template o il modo di calcolare la differenza di fuso orario che in una specifica sede è sempre uguale, e i dati che l'utente può scegliere quando avrà a che fare con quell'evento, come le liste di possibili valori per un determinato campo dell'evento. In questo modo le applicazioni vengono configurate automaticamente e all'utente finale viene lasciato il compito di decidere solo nei casi in cui è davvero richiesto il suo effettivo intervento. Con questa procedura, dell'installazione dinamica, i dispositivi evitano di appesantirsi e di perdere tempo ad installare applicazioni di cui magari non avranno mai bisogno, guadagnando in velocità durante la fase di scaricamento ed installazione e avendo comunque la possibilità di ottenere un'applicazione in futuro, quando sarà davvero necessaria la sua funzionalità.

3.6 Progettazione delle applicazioni

Ora si introducono le scelte progettuali delle diverse applicazioni sviluppate, spiegando le ragioni che hanno portato al loro funzionamento e le caratteristiche che possiedono, trattando anche le tecniche di progettazione della UX. In particolare si descrive il funzionamento di due tipi di applicazioni, il launcher e le applicazioni che gestiscono i processi logistici, portando in totale questo lavoro alla creazione di due progetti Android Studio.

3.6.1 User Experience

Una caratteristica fondamentale per lo sviluppo di applicazioni mobili è quella di progettare bene la UX altrimenti possono insorgere molti problemi legati al fatto che gli utenti non sono invogliati ad utilizzare l'app. La valutazione del prodotto dipende in larga misura dall'esperienza accumulata dall'utente nel processo di utilizzo, quindi la funzionalità del prodotto è essenziale. La UX deve fornire agli utenti la migliore esperienza utente in termini di facilità d'uso e sensazione che gli utenti provano dall'utilizzo dell'applicazione. Pertanto, ha un impatto sul miglioramento del valore aziendale perché aiuta a migliorare l'efficienza e l'attenzione dell'azienda ai clienti e operatori. L'esperienza utente deve essere comoda, intuitiva e chiara, progettata per trasformare gli utenti dell'app in utenti felici di utilizzarla. Un modello di progettazione della UX è dato da J.J. Garrett, un famoso User Experience Designer, secondo il quale ogni aspetto del design è il risultato delle volontà specifiche del progettista. Il modello di Garrett si basa sul fatto che il processo di progettazione della UX di un prodotto software è formato da cinque fasi che corrispondono a cinque piani diversi. Nella Figura 3.3 si può vedere la struttura del modello di Garrett. I cinque piani si sviluppano in verticale e dal basso verso l'alto le problematiche si fanno sempre più specifiche, portando quindi ogni piano ad essere dipendente dal piano precedente. Questa dipendenza significa che le scelte disponibili nel piano sono limitate dalle decisioni prese nel piano sottostante, quindi le scelte fatte al livello più basso (cioè il piano strategico) hanno un impatto su tutta la struttura. Garrett ha sottolineato l'importanza di iniziare le attività su un livello prima di completare le attività del livello successivo, per evitare di sovrapporre più di due livelli. Per quanto riguarda i cinque piani, partendo dal basso verso l'alto, quindi dal piano con problematiche più astratte a quello con problematiche più concrete, ci sono:

- Il piano della Strategia. In questo piano, il più astratto, viene definito l'utente del sistema, gli obiettivi che si è prefissato e quali sono le sue esigenze e gli obiettivi aziendali.

- Il piano dell'Obiettivo. In questo piano, vengono descritte le specifiche funzionali e il contenuto necessario del sistema per soddisfare le esigenze dell'utente.

- Il piano della Struttura. Questo piano è utilizzato per definire l'interazione tra l'utente e il sistema e l'organizzazione del contenuto in modo che l'utente possa accedervi rapidamente ed efficacemente.

- Il piano dello Scheletro. In questo piano, è indicato il design dell'interfaccia con cui gli utenti devono interagire, gli elementi che consentono agli utenti di

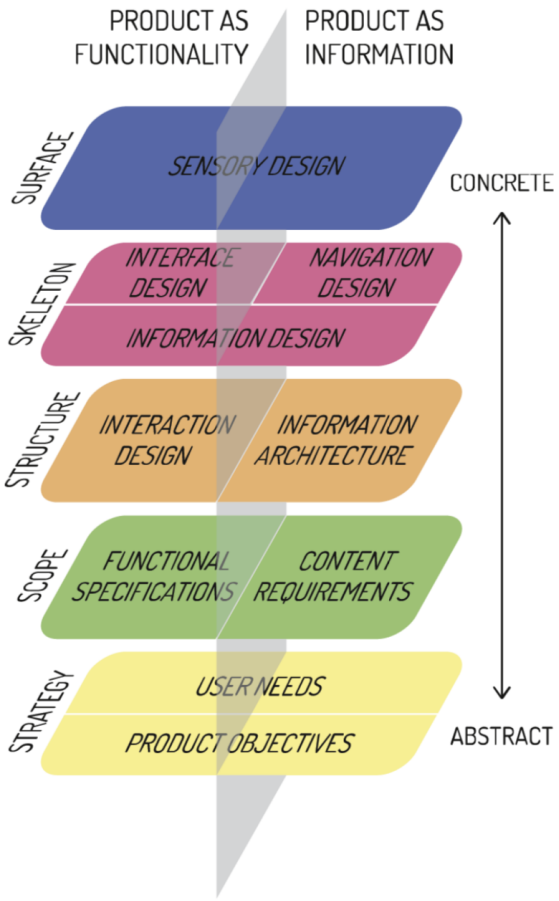

Figura 3.3: Struttura del modello di Garrett

navigare nei contenuti e il modo di presentazione delle informazioni che deve essere di facile comprensione.

- Il piano della Superficie. Questo è il piano più concreto, la parte visibile con cui interagisce l'utente in cui è trattata la parte grafica. Questo piano aiuta significativamente a decidere l'esperienza dell'utente che ha con il prodotto finale.

3.6.2 Progettazione del launcher

I launcher sono delle applicazioni che si sostituiscono alla schermata iniziale del dispositivo, in modo da fornire un accesso più comodo e possibilmente più rapido ad applicazioni e file. Come ogni dispositivo, anche il TC20 dispone di un launcher

di default, ma si è scelto di crearne uno ad hoc per avere il pieno controllo già dall'accensione del dispositivo. Il launcher creato ha due compiti principali:

- Gestire la fase di autoconfigurazione del dispositivo.
- Elencare le app necessarie, quindi quelle installate nella fase di autoconfigurazione, e l'app delle impostazioni, utile per gestire il dispositivo.

Le applicazioni sviluppate quindi si avviano solo attraverso la pressione della loro icona da questo launcher. Questa scelta nasce dal fatto che all'apertura dell'applicazione che serve, viene passata la configurazione per quell'app attraverso un Intent dal launcher. Inoltre esiste anche un apposito bottone, nel menu del launcher, che se premuto verifica la presenza di un nuovo file di configurazione ed è quindi possibile sia riconfigurare il dispositivo, sia eseguire l'aggiornamento o l'installazione di un'app. Il launcher è molto importante perchè ha il compito di inizializzare il dispositivo, anche per quanto riguarda l'autenticazione. L'autenticazione è trattata dettagliatamente nel capitolo successivo, in cui si affronta questo argomento descrivendo l'uso dei certificati X.509 e dei JWT. Quindi, il launcher è stato progettato per essere il punto di ingresso per tutte le altre app, indispensabile per avere il pieno controllo e i massimi risultati desiderati.

3.6.3 Progettazione delle app per i processi logistici

Questa tesi ha come compito quello di migliorare lo svolgimento dei processi logistici grazie all'utilizzo di applicazioni Android sviluppate appositamente. Tra i processi gestiti in questo progetto ci sono:

- Registrazione di un nuovo prodotto.
- Tracciamento di un prodotto.
- Impacchettamento di prodotti.
- Verifica di un ordine.

Ognuno di questi processi è gestito da una specifica app, che come già detto, è una variante dell'app main. Ogni app, quando è avviata dal launcher, ha il compito di salvare il file di configurazione passato attraverso un Intent, e quindi gestire il processo per il quale è stata creata prendendo la maggior quantità di dati dal file di configurazione senza chiederli all'utente. Tutte le app, gestiscono il loro processo grazie all'invio al server di un evento EPCIS, la cui creazione è facilitata dall'autoconfigurazione, ognuno dei quali è preso in carico e gestito internamente, lasciando ai dispositivi il compito di raccogliere e generare le informazioni necessarie,

non reperibili in nessun altro modo. La fase di login è comune per tutte le app, mentre cambia la gestione vera e propria del processo, spesso eseguita da una o più activity. Le applicazioni sviluppate seguono uno stile minimalista e questo è appositamente voluto per renderle facili e intuitive nel loro utilizzo. Il minimalismo si ha quando riducendo un insieme di componenti al minimo necessario si produce comunque l'effetto desiderato. Ciò può significare eliminare gli elementi non necessari e lasciare solo ciò che è necessario affinché l'interfaccia funzioni correttamente. Molte applicazioni mobili seguono il principio di rimuovere immagini o informazioni non necessarie, ottenendo così un design minimalista, che occuperà il contenuto principale. L'eliminazione di tutte le garanzie aggiuntive aiuterà gli utenti a concentrarsi sull'effettiva funzionalità dell'applicazione. Quando si hanno meno elementi di design, potrebbe essere necessario aumentare le dimensioni o il peso degli elementi rimanenti per avere la stessa attenzione. L'obiettivo è avere un impatto significativo anche con meno elementi di design considerando i dettagli perché hanno un peso maggiore nel design complessivo.

Capitolo 4

Implementazione del software

In questo capitolo si entra nel dettaglio della parte implementativa di questo progetto. Vengono trattate in maniera dettagliata le librerie utilizzate per sviluppare le applicazioni, soffermandosi sopratutto sulle librerie che permettono la scansione dei tag RFID, dei codici a barre e dei QR code, ma anche sulle librerie Android senza le quali sarebbe problematico e laborioso sviluppare le applicazioni per questa piattaforma. Quindi si descrivono le diverse applicazioni a livello di implementazione e di codice, ponendo particolare attenzione alle parti principali e quindi spiegando come il loro funzionamento è reso possibile.

4.1 Software per gestire i tag RFID

Unified Zebra RFID SDK per Android fornisce una serie di API per sfruttare al massimo le prestazioni, la funzionalità e la versatilità di RFD2000 e altri reader portatili Zebra. Comprende librerie di classi, applicazioni di esempio e codice sorgente, in modo che gli sviluppatori possano creare facilmente applicazioni che sfruttino le funzionalità dei dispositivi. Questo software è stato ampiamente utilizzato quando nelle applicazione si devono scansionare dei tag RFID, in particolare creando un'activity che gestisce le operazioni del lettore RFD2000 ed utilizzando la libreria RFID API3. Come prima cosa sono state inizializzate le variabili, necessarie per poter interagire con il lettore, come la lista di reader disponibili, il reader che si desidera utilizzare, il gestore degli eventi quando viene ricevuta una lettura ecc. Dopo aver creato l'istanza dei readers, i compiti principali di questa activity sono:

- Recuperare i lettori disponibili. Questa operazione viene eseguita per mezzo di un thread secondario. Questo thread viene usato per eseguire un'attività

asincrona, definita da un calcolo che viene eseguito in background e il cui risultato è condiviso e pubblicato sul thread UI. Quindi viene fornita la lista di readers disponibili attraverso GetAvailableRFIDReaderList e selezionato il primo della lista. In seguito, avviene la connessione del reader e si procede alla sua configurazione.

- Configurare il lettore utilizzato. Il lettore deve essere configurato per ottenere lo scopo desiderato. In particolare si possono scegliere i vari parametri del lettore, ad esempio il livello di potenza dell'antenna, il modo del trigger (in questo caso per leggere i tag RFID), aggiungere i gestori degli eventi ecc.
- Gestire gli eventi di operazione e di stato del lettore. La libreria permette di avere la massima gestione degli eventi ricevuti attraverso la classe EventHandler che implementa l'interfaccia RfidEventsListener. Si possono gestire sia gli eventi dell'avvenuta lettura dei tag RFID, usati per andare poi a popolare gli eventi EPCIS, sia gli eventi di pressione e rilascio del bottone del reader, per avere la massima personalizzazione.

Al momento dell'uscita, l'applicazione deve disconnettersi dal lettore e liberare l'istanza dell'SDK. Questi passi sono necessari perchè altrimenti il reader resterebbe connesso anche dopo la chiusura. Questa è la parte saliente dell'activity che gestisce la scansione dei tag RFID. Si va ad aggiungere alla logica per creare e popolare gli eventi EPCIS che l'applicazione deve fornire per eseguire il processo logistico trattato.

4.2 Software per gestire barcode e QR code

Per quanto riguarda la scansione dei codici a barre e dei QR code, si è optato per l'utilizzo di Enterprise Mobility Development Kit, o EMDK per Android che offre un set di funzionalità e di capacità per i dispositivi Zebra. Include una serie di API e codice di esempio, che consente di programmare i dispositivi di questa compagnia avendo il massimo delle prestazioni. Il lettore di codici a barre integrato del TC20 è sfruttato in questo contesto, perchè grazie a questo kit è possibile controllarlo e gestire le operazioni che richiedono la lettura. Anche in questo caso, è stata creata un'activity che gestisce il processo di scansione che viene poi riempita anche del codice per gestire i processi logistici considerati. Per prima cosa si deve modificare l'applicazione, in particolare il file Manifest.xml per utilizzare la libreria EMDK e per autorizzare l'EMDK alla scansione dei codici a barre e dei QR code ed inoltre bisogna aggiungere nell'activity i riferimenti alle librerie che si vogliono utilizzare. L'activity deve implementare:

- EMDKListener, con le funzioni onOpened e onClosed per gestire un riferimento all'EMDK in fase di apertura e di chiusura.

- StatusListener, per notificare gli eventi di scansione e lo stato dello scanner, con la funzione onStatus.
- DataListener, per sapere quando i dati scansionati sono disponibili e quindi operare con le letture in un thread secondario, con la funzione onData.

Una delle prime operazioni da fare è inizializzare l'EMDK con getEMDKManager e inizializzare ed abilitare lo scanner e i suoi listener utilizzando un oggetto BarcodeManager. Per effettuare le scansioni si utilizza il metodo read() dello scanner che si avvia dopo la pressione del tasto principale del RFD2000, mentre per sapere se lo scanner è già impegnato in una scansione si usa isReadPending(). Le scansioni ricevute si elaborano nella funzione onData mentre lo stato dello scanner nella funzione onStatus. Alla fine, quando si chiude l'applicazione è necessario rilasciare le risorse.

4.3 Implementazione delle applicazioni

4.3.1 Download e installazione di APK

L'app responsabile dello scaricamento e dell'installazione delle altre applicazioni è il launcher. Il launcher è trattato dettagliatamente in seguito, ma adesso si fa il punto sul meccanismo che permette lo svolgersi di questi compiti. Per fare questo, si è usato un download manager e le applicazioni si installano in modo programmatico. Per scaricare i diversi APK, che corrispondono alle applicazioni che gestiscono i processi logistici, c'è bisogno delle autorizzazioni di archiviazione e delle autorizzazioni Internet, mentre al termine dei download, le applicazioni si installano a livello di programmazione. Per questo motivo, serve anche l'autorizzazione per installare i pacchetti. Tutte le autorizzazioni necessarie sono state fornite nel manifest file. Per accedere ai contenuti scaricati si è utilizzato un file provider, mentre le operazioni di scaricamento ed installazione si trovano in un file chiamato DownloadController. Molto semplicemente si fa uso di:

- Il download manager di Android per scaricare gli APK, caricati precedentemente sul server. Il download manager ha bisogno di sapere la destinazione dove scaricare il file, si è scelto la classica cartella di download di Android, e l'URL da dove recuperare il contenuto, in questo caso l'URL della API che gestisce questo processo.
- L'installer di Android che provvede ad installare le applicazioni. L'installer ha bisogno di sapere il percorso del file che si deve installare e viene fatto avviare da un Broadcast Receiver che controlla quando il download è terminato per iniziare l'installazione.

Nell'activity principale del launcher, vengono controllati prima i permessi di archiviazione e in caso di concessione, viene chiamata la funzione di download che fa partire le operazioni presenti in DownloadController.

4.3.2 Autenticazione client

Le applicazioni sviluppate si basano sullo standard OAuth (Open Authorization), uno standard per l'autorizzazione, che prevede l'utilizzo di un access token. Quando un nuovo dispositivo viene registrato nel sistema, viene creato ed associato un certificato X.509. X.509 è un formato standard per i certificati a chiave pubblica. I certificati a chiave pubblica sono documenti digitali che legano in modo sicuro una coppia di chiavi crittografiche a identità quali siti Web, individui o organizzazioni. In questo caso, questi certificati vengono usati per l'autenticazione del client che in fase di inizializzazione viene marcato come unico utilizzatore di quel certificato e nessun altro dispositivo lo potrà più utilizzare evitando così una possibile clonazione. Dopo l'autenticazione del dispositivo, l'authentication server emette un access token e un refresh token. L'access token che permette alle applicazioni di accedere alle risorse del server, includendolo nell'header delle richieste, ha una validità pari ad un anno e poi si deciderà se aggiornarlo per mezzo del refresh token. La tipologia di token utilizzati prende il nome di JWT. JSON Web Token è uno standard per l'autenticazione attraverso token che trasmette informazioni sicure sotto forma di oggetto JSON. Nella Figura 4.1 si può vedere il flusso di autenticazione con JWT.

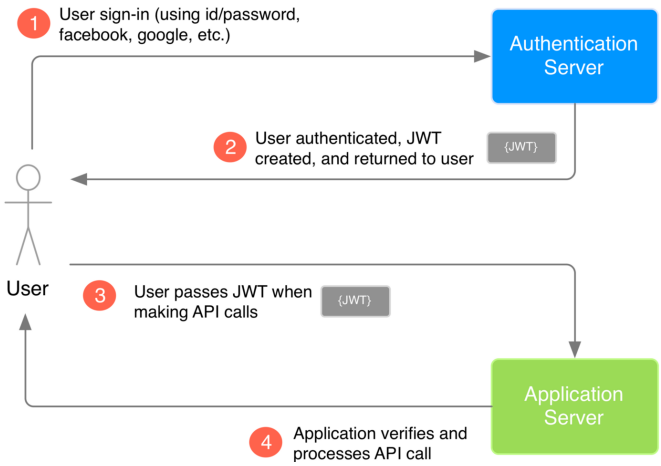

Figura 4.1: Flusso di autenticazione con JWT

I JWT sono formati da tre parti separati da punti (xxxxx.yyyyy.zzzzz). Le parti di un JWT sono:

- Header, formata da due parti: il tipo di token, che è JWT, e l'algoritmo di firma utilizzato, come HMAC, SHA256, RSA, ecc.
- Payload, un oggetto JSON che contiene una serie di informazioni sul token.
- Signature, la parte più importante che serve per verificare che il token non sia stato modificato lungo il percorso e quindi verificare la validità degli altri campi.

4.3.3 TLS

All'inizio dell'era tecnologica, la sicurezza ricopriva un ruolo nettamente inferiore rispetto a quello che ricopre oggi. Tutte le comunicazioni tra computer avvenivano in modo non crittografato e quindi chiunque aveva la possibilità di curiosare nel traffico altrui. Il protocollo TLS, chiamato anche SSL/TLS, visto che in principio aveva il nome di SSL, ha introdotto la crittografia dei contenuti trasmessi nella rete, aumentando la sicurezza a livello trasporto del Modello TCP/IP. Il nome TCP/IP è composto da due protocolli chiave utilizzati per la comunicazione Internet, Transmission Control Protocol e Internet Protocol. Però, il termine si riferisce anche ad altri protocolli come Internet Control Message Protocol e User Datagram Protocol che fanno parte di questo gruppo. Pertanto, il protocollo TCP/IP stesso non è una tecnologia specifica, ma un insieme di protocolli selezionati. Ciò che hanno in comune è che sono diventati lo standard per le comunicazioni di rete. Lo scopo del livello di trasporto è quello di fornire un canale logico di comunicazione end-to-end per pacchetti e TLS agisce aumentando la sicurezza in questo livello, fornendo un processo per crittografare il flusso di dati sulla rete, per permettere di leggerlo soltanto dai legittimi destinatari. Per permettere questo funzionamento c'è una prima fase in cui si utilizza un protocollo di handshake per:

- Concordare una coppia di chiavi che servono per confidenzialità ed integrità.
- Scambiare dei numeri casuali tra il client ed il server che servono come base per poi generare autonomamente le chiavi di crittografia.
- Stabilire una chiave simmetrica (da cui saranno poi derivate la chiave di cifratura e quella di autenticazione) tramite operazioni a chiave pubblica.
- Negoziare il session-id.
- Scambiare tutti i certificati a chiave pubblica necessari.

Nella Figura 4.2 e nella Figura 4.3 si hanno tutti i pacchetti che sono scambiati durante la fase di handshake.

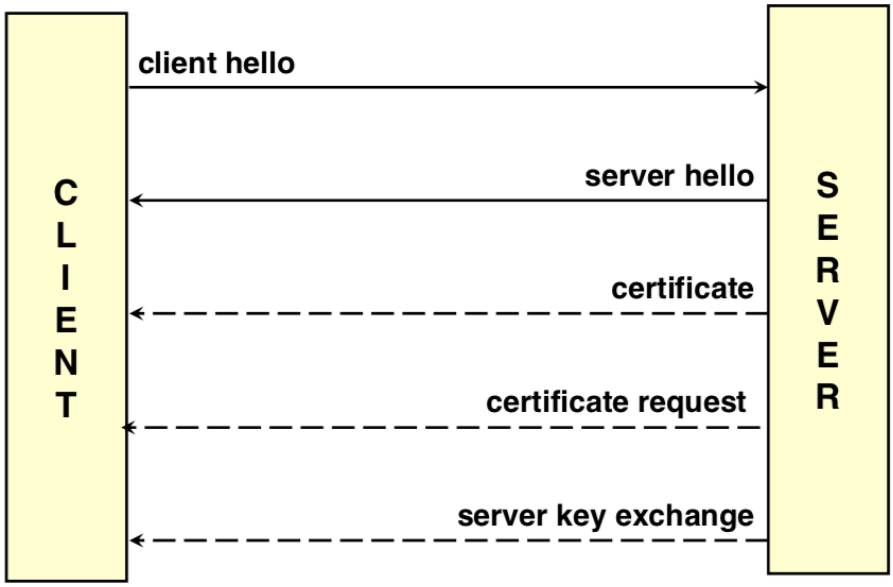

Figura 4.2: TLS - fase di handshake 1

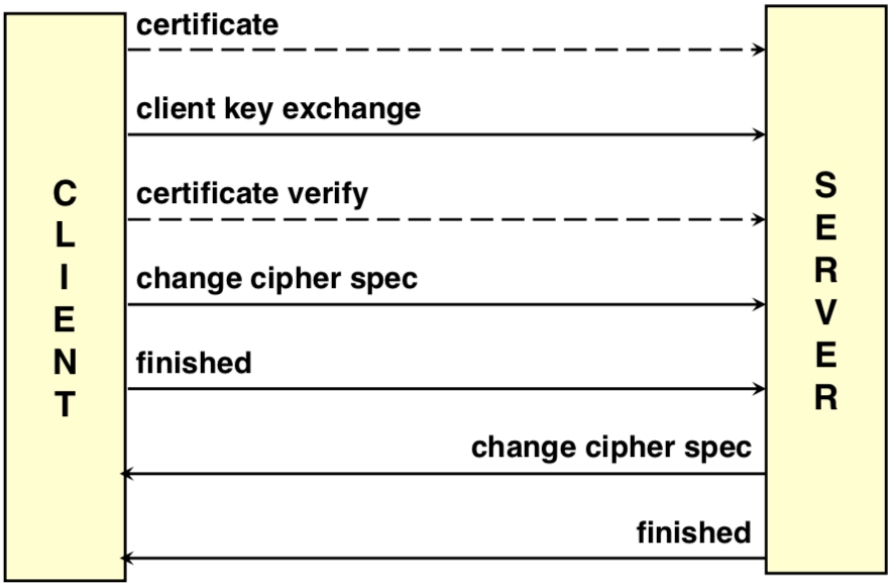

Figura 4.3: TLS - fase di handshake 2

La linea tratteggiata indica che in una qualche implementazione del protocollo i pacchetti corrispondenti sono opzionali. Il client inizia la comunicazione inviando un messaggio chiamato client hello, a cui il server risponde con un server hello. Dopodichè sono opzionali perchè per esempio si decide di usare il session-id, lo

scambio del certificato del server (certificate), la richiesta del certificato al client (certificate request), un messaggio per lo scambio delle chiavi del server (server key exchange), l'invio del certificato (certificate) nel caso in cui il client ne sia in possesso e il server ne abbia fatto richiesta. Invece è obbligatoria la client key exchange, che serve per generare le chiavi specifiche per questa connessione. Opzionale ancora una fase di verifica del certificato (certificate verify). Infine si ha l'invio del messaggio change cipher spec, che serve ad attivare i protocolli di protezione, e del messaggio finished, che serve a proteggere tutto lo scambio precedente. Questi ultimi due messaggi sono anche inviati dal server verso il client per comunicare al client che anche lui sta per attivare tutte le misure di sicurezza e per proteggere i messaggi da lui inviati.

4.3.4 OkHttp

Una delle prime cose da implementare per tutte le applicazioni sviluppate è stata la modalità di interazione con il server per accedere al servizio ed ai contenuti offerti. Si è progettato un meccanismo di connessione a Internet che semplifica la gestione per evitare di dover agire sui dettagli ed i meccanismi di basso livello, utilizzando una libreria che permette di ottimizzare la connettività. In questo progetto quindi, per quanto riguarda le richieste che le applicazioni mobili devono fare al server, è stato usato OkHttp [11], un client HTTP efficiente per applicazioni Android e anche Java. Tra le caratteristiche di OkHttp ci sono:

- La possibilità di condividere lo stesso socket tra richieste allo stesso host grazie al supporto HTTP/2.
- La possibilità di usare un pool di connessioni (se HTTP/2 non è disponibile).
- L'uso della compressione GZIP trasparente per riduce le dimensioni dei download.
- Memorizza nella cache le risposte per evitare completamente la rete per richieste ripetute.

Questa libreria consente di costruire una richiesta specificando alcuni parametri come:

- L'URL della risorsa a cui accedere.
- Il tipo di operazione da eseguire (GET, POST, PUT, DELETE).
- Un elenco di intestazioni, anche di autenticazione.
- Un corpo che può contenere delle informazioni adattate in un formato adatto allo scambio dei dati.

Prima di tutto bisogna creare un nuovo oggetto di tipo OkHttpClient che ha il compito di inviare le richieste e ricevere le risposte. OkHttpClient supporta due diverse modalità di esecuzione delle chiamate:

- La modalità sincrona attivata dal metodo execute() (che causa il blocco del thread chiamante in attesa di una risposta)
- La modalità asincrona attivata dal metodo enqueue() (accodando la richiesta e pianificando l'esecuzione futura). In questa modalità, verrà chiamata una Callback specifica quando sarà ricevuta la risposta.

Questa libreria, è stata scelta oltre al fatto di essere open-source, anche perchè offre la possibilità di implementare TLS, con la possibilità di specificare la versione e la cipher suite desiderata. Per far funzionare TLS sul dispositivo android si deve salvare e registrare il certificato della Certification Authority, un soggetto terzo di fiducia abilitato ad emettere un certificato digitale, su quest'ultimo. In questo modo è possibile verificare l'identità del server attraverso il metodo hostnameVerifier(), impostando le dovute configurazioni di rete nel manifest file in modo tale da sapere quali certificati sono attendibili. Così dopo la fase di handshake, descritta precedentemente, l'applicazione Android comunicherà con il server avendo tutte le misure di sicurezza di TLS per proteggere i messaggi scambiati.

4.3.5 Implementazione del launcher

Il launcher costituisce il punto d'ingresso e di inizializzazione dell'intero progetto, un'app che permette di controllare i dispositivi subito dopo l'accensione e la cui implementazione è indispensabile per le altre app. Questa applicazione, dopo la fase di autenticazione iniziale, con un login supportato da certificato X.509, deve elencare le altre app in modo tale da permettere all'utente di accedere a quella di cui ha bisogno. Popolare delle liste di elementi dinamicamente non è una buona idea perchè se non si conosce il numero di elementi a priori, questa potrebbe essere anche molto grande, e se il numero di voci da visualizzare aumenta in modo significativo, la gerarchia visiva aumenta e consuma molte risorse. Al giorno d'oggi esiste un approccio moderno, flessibile e performante per la visualizzazione degli elenchi che si basa su RecyclerView. Le RecyclerView sfruttano il fatto che di solito, è possibile visualizzare sullo schermo solo un insieme limitato di elementi, quindi si evita di creare tante visualizzazioni quanti sono gli elementi nella raccolta, ma si crea solo un piccolo numero. RecyclerView è una classe della libreria Jetpack che gestisce la presentazione di dataset potenzialmente grandi. Gli elementi renderizzati vengono elaborati dall'adattatore, che è responsabile di fornire la vista e popolarla con il contenuto pertinente nel set di dati. Ogni elemento nel set di dati viene presentato in una gerarchia visiva riutilizzabile gestita da ViewHolder. La rappresentazione

visiva dell'oggetto è delegata a LayoutManager, eventualmente coadiuvata da ItemAnimator e ItemDecorator e dato un modello di dati, di solito viene creato un layout corrispondente per presentarne il singolo contenuto della lista. Nella Figura 4.4 si vede graficamente l'architettura descritta.

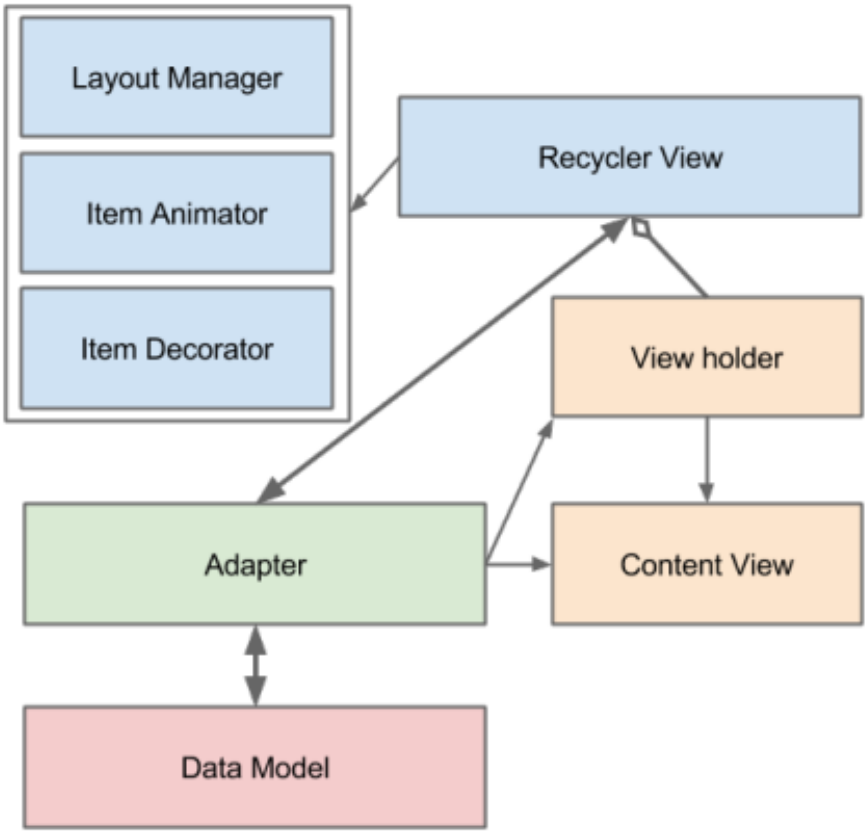

Figura 4.4: Architettura RecyclerView

In questo progetto, il singolo elemento della lista è formato da:

- label: il nome dell'app da visualizzare.
- packageName: il nome del package dell'app da visualizzare, utile per fare controlli per capire cosa visualizzare e non visualizzare (dal momento che si vuole avere solo le app necessarie nel launcher e non tutte le app presenti sul dispositivo).
- icon: l'icona dell'app da visualizzare.

Inoltre, si è scelto di utilizzare un GridLayoutManager come gestore di layout, con il quale si possono visualizzare i dati in una griglia come qualsiasi galleria fotografica,

una cosa molto comune per i launcher Android. Il launcher dispone anche di un context menu, un menu contestuale che viene visualizzato dopo un clic prolungato su una qualsiasi app. Questo menu serve per permettere di disinstallare un'app della quale non si ha più bisogno, per evitare di sprecare spazio dal momento che, come già detto, questo progetto verte molto sul risparmio di spazio e di tempo. Anche se in fase di bootstrap, viene scaricato il file di configurazione dal server, e viene configurato quindi il dispositivo, con annessa installazione delle app necessarie, esiste anche un bottone nell'Appbar del launcher che permette di verificare la presenza di un nuovo file di configurazione e quindi di scaricarlo. Questa funzione del launcher serve per verificare la necessità di riconfigurare il dispositivo oppure verificare e quindi procedere con l'aggiornamento o l'installazione di un'app. Nella Figura 4.5 si vede graficamente il launcher implementato.

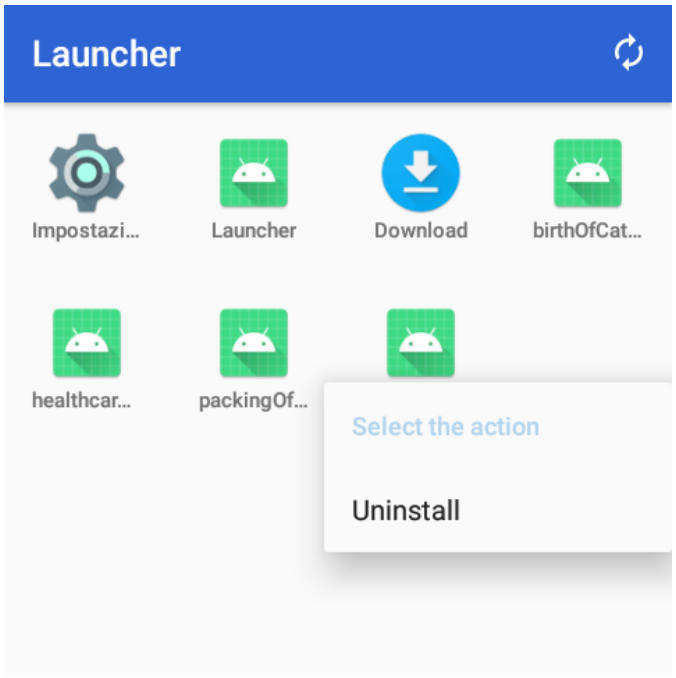

Figura 4.5: Launcher

Dopo un click sull'icona di un'app, viene avviata l'app in questione con context.startActivity(intent), in cui intent racchiude il file di configurazione da passare all'app per permetterle di avere tutti i dati per configurarsi e molto importate, per avere l'event template (il template dell'evento EPCIS che dovrà generare per mandarlo al server, per svolgere un determinato processo logistico). L'event template possiede i dati (fissi, non fissi, da calcolare ecc) per evitare che l'app debba costruire l'intero evento da zero, ma avendo già un punto di partenza. Gli event template di ogni app sono trattati quando si descrive quell'app in seguito, in cui si

capisce anche come costruire progressivamente l'evento. Un altro compito che ha il launcher è quello di fornire un sistema di monitoraggio. Viene mandato un ping al server al verificarsi di una di queste tre condizioni:

- Accensione del dispositivo.
- Apertura di un'app.
- Ogni ora, per implementare un sistema di heartbeat.

Per tutte queste condizioni, ma soprattutto per l'ultima, si è implementata una coroutine, che nell'ultimo caso ogni ora manda un ping al server. Inoltre è stato sviluppato anche un launcher di debug nel progetto delle app che gestiscono i processi logistici, usato per far partire un launcher minimal che lista solo le app e passa ad esse un file di configurazione standard, utile per testare le diverse app in fase di sviluppo avendo davanti un solo progetto Android Studio.

4.3.6 Implementazione della parte comune delle app per la gestione dei processi logistici

Dal momento che tutte le app per la gestione dei processi logistici sono una versione differente della stessa app, tutte quante hanno delle parti di implementazione in comune. In particolare le parti in comune sono la parte di configurazione, di autenticazione, di struttura e di invio degli eventi EPCIS al server. Una delle prime cose fatte appena un utente apre un'app è recuperare il file di configurazione dall'intent mandato dal launcher, questo spiega il motivo della chiusura dell'app se viene avviata da un launcher diverso da quello sviluppato appositamente perchè non sarebbe mandato un intent con il file di configurazione. Grazie al contenuto di questo intent l'app si può autoconfigurare con le credenziali di accesso (un token JWT di accesso e un token JWT di refresh), l'indirizzo ip del server con cui comunicare e molto importante l'event template del processo che deve gestire che viene utilizzato per creare l'evento EPCIS associato e salvarlo nel database locale, fin quando non viene mandato al server in condizioni online. Il modello generico di un template ha il seguente formato JSON:

- "name": MODEL_NAME
- "type": EVENT_TYPE
- "what": [FieldDef, ...]
- "when": [FieldDef, ...]
- "why": [FieldDef, ...]

- "where": [FieldDef, ...]
- "ilmd": [AttrDef, ...]

MODEL_NAME è una stringa che rappresenta il nome univoco di quel modello di evento. EVENT_TYPE è una stringa che rappresenta il tipo di evento EPCIS. Può assumere uno dei seguenti valori: ObjectEvent, AggregationEvent, TransactionEvent, TransformationEvent. Dalla scelta del tipo dipendono gli altri campi del modello con cui saranno valorizzati i FieldDef. I restanti campi sono liste di elementi JSON con un certo formato. Nello specifico: what, why, where, when sono liste di FieldDef. Ognuna di queste liste ha sempre almeno un elemento di tipo FieldDef al suo interno. Un FieldDef è un oggetto JSON che racchiude le proprietà di un determinato campo di una delle dimensioni W dell'evento EPCIS. Questo oggetto è formato da alcuni campi obbligatori come name, type e input, e da alcuni campi opzionali come alias, value, fun, whitelist, strict, optional e children. Attraverso un FieldDef e alla composizione dei suoi campi è possibile sapere quali campi sono già disponibili, quali invece deve inserire a mano l'utente e quali magari calcola in maniera autonoma il sistema. Per esempio se input é external, la whitelist ha dei valori e il flag strict è true, l'utente può selezionare come valore solo gli elementi presenti nella lista, mentre se input è fixed, il valore non lo deve scegliere l'utente ma è presente nel campo value, permettendo all'evento di configurarsi da solo con quel valore senza chiedere nulla all'utente. Una diversa configurazione dei campi del FieldDef porta a diversi modi di inserire i dati necessari, così facendo l'utente deve intervenire solo quando è strettamente necessario. La chiave "ilmd" contiene una lista, anche vuota, di AttrDef. Un campo AttrDef è un oggetto JSON che rappresenta un attributo specifico dell'event template considerato. Se è presente, si compone di campi obbligatori come ns e name, e di alcuni campi opzionali come input, value, default e optional. Per gli ilmd è plausibile che il campo input sia sempre external perchè rappresenta una lista di attributi che generalmente non sono conosciuti a priori, quindi vengono spesso riempiti attraverso degli Spinner o degli EditText direttamente dall'utente, che opera sul campo e sa cosa ha davanti. Anche in questo caso, come per i FieldDef, una diversa configurazione dei campi porta a diversi modi di inserire i dati necessari.

Dopo aver salvato tutte le informazioni, ogni app esegue l'accesso per mezzo di un token JWT che come quello per il launcher scade dopo un anno e può essere aggiornato per mezzo del refresh token. Come ulteriore parte in comune tra le app c'è la parte di struttura. Le app sono progettate per essere in continua evoluzione quindi si è scelto di basarle sul Navigation drawer. Il Navigation drawer permette di navigare con facilità all'interno dell'app grazie ad un menu a tendina che di solito si apre dalla parte sinistra alla parte destra dello schermo. Consiste nella combinazione di un DrawerLayout che racchiude un gestore di layout e un NavigationView. Il primo avvolge qualsiasi contenuto deve essere mostrato sulla pagina. Il secondo

crea una finestra di dialogo utilizzata per visualizzare i collegamenti di navigazione nell'app(formato da un headerLayout, che in genere trasmette alcuni grafici, e un menu, in cui ogni elemento rappresenta una vista di destinazione da mostrare nel gestore layout. Quando c'è un cambiamento di connessione alla rete, viene attivato un Broadcast Receiver che controlla gli eventi EPCIS salvati in locale, e se si è online, vengono effettuati dei controlli di correttezza e vengono mandati tutti gli eventi per quell'app al server, cancellandoli dal database. Esiste anche un Fragment, accessibile da tutte le app attraverso il Navigation Drawer, in cui sono visualizzati tutti gli eventi EPCIS salvati e si ha la possibilità, premendo il tasto "PUSH", di mandarli ugualmente al server anche in presenza di errori e quindi di cancellarli. Nella Figura 4.6 si vede il Fragment che gestisce quanto detto.

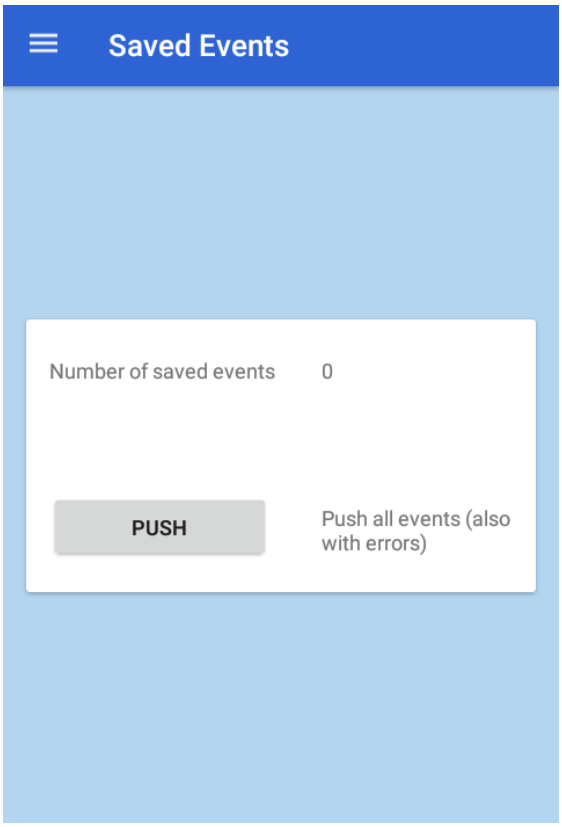

Figura 4.6: Eventi salvati

4.3.7 Implementazione dell'app per la registrazione di un prodotto

Questa app ha come scopo quello di registrare un nuovo prodotto e per svolgerlo correttamente deve scrivere un tag RFID e mandare l'evento EPCIS appropriato. La parte saliente di quest'app è l'activity FirstGeneration. Questa activity deve innanzitutto generare l'id del tag RFID da scrivere dopo averlo scansionato. Per fare questo l'app recupera dall'event template la lista di codici dei prodotti supportati per questa operazione e ha il compito di chiedere al server un batch di codici EPC per ognuno, salvandoli sul database locale per poterli utilizzare al bisogno. A questo punto l'app verifica dall'event template i campi che deve riempire l'utente a mano, quindi solo quelli che hanno bisogno di un input esterno. In questa categoria rientrano gli attrDef e le location (che possono essere anche selezionati da uno spinner). Dopo aver selezionato anche il codice del prodotto che si intende utilizzare, si provvede alla scansione del tag RFID che si desidera scrivere e premendo sul bottone responsabile della scrittura e dell'invio dei dati al server, viene sia scritto il tag RFID, sia mandato l'evento EPCIS corrispondente. L'applicazione chiama il metodo reader.Actions.TagAccess.writeWait per scrivere i dati in un banco di memoria specifico, quindi in questa fase si risale dal codice EPC all'id del tag che poi viene scritto. L'evento EPCIS generato viene comunque salvato sul database locale e se sono presenti le condizioni di rete necessarie, viene estratto dal database e mandato al server. Queste operazioni di salvataggio sul database vengono eseguite per poter lavorare anche in condizioni offline e un Broadcast Receiver verifica le condizioni di rete per sapere quando si torna online per poter mandare tutti gli eventi salvati precedentemente. Come caso d'uso è stato usato quello della nascita del bestiame, considerando un'azienda che svolge come processi logistici quelli legati a questi animali. Nelle Figure 4.7, 4.8, 4.9 si vede l'activity che gestisce questa app.

In questo specifico caso l'utente deve selezionare a mano obbligatoriamente i valori di alcuni AttrDef che posseggono il marchio "required", in particolare gli attributi ear tag, birth country, breed e mother ear tag, senza i quali non si potrebbe procedere, mentre dall'event template non risultano obbligatori gli attributi sex e father ear tag, quindi l'utente può scegliere se valorizzare o no tali campi. Per quanto riguarda il product code da usare, l'utente può sia selezionarlo da uno spinner, i cui valori sono recuperabili dalla whitelist del FieldDef chiamato epcList, oppure selezionarlo da tastiera in quanto il flag strict è uguale a false. Analogamente, la location è recuperabile da uno spinner, popolato dai valori della whitelist del FieldDef chiamato bizLocation, solo che in questo caso si deve richiedere al server i nomi effettivi dal momento che la whitelist contiene dei codici che identificano le location e altrimenti l'utente avrebbe a che fare con dei codici di poca comprensione. Anche in questo caso il flag strict è false quindi l'utente può

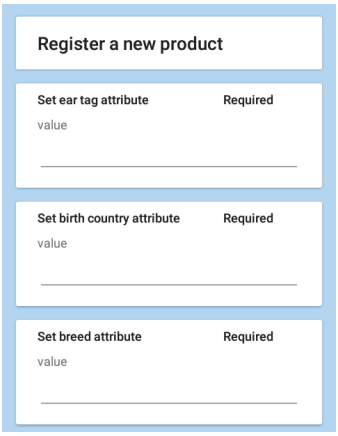

Figura 4.7: App per la registrazione di un prodotto 1

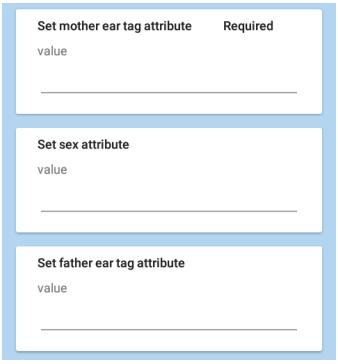

Figura 4.8: App per la registrazione di un prodotto 2

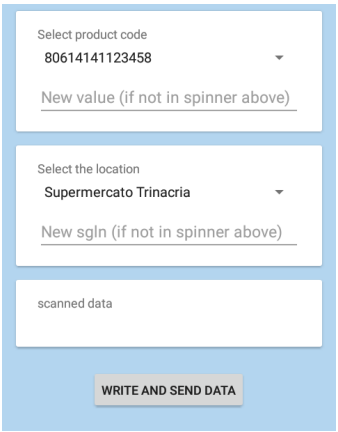

Figura 4.9: App per la registrazione di un prodotto 3

selezionare un codice da tastiera se proprio lo ritiene opportuno. La location è usata per valorizzare il FieldDef bizLocation. Al momento della scansione di un tag RFID, la TextView relativa a scanned data viene riempita col valore del tag scansionato e alla pressione del bottone "WRITE AND SEND DATA", viene scritto con il nuovo tag generato da un codice EPC (formato dal product code selezionato più un numero seriale che decide il server), usato per valorizzare il FieldDef epcList.Viene quindi creato anche l'evento EPCIS in questo caso usando il template dal nome "birth_of_cattle" riempito dei FieldDef e AttrDef descritti, con l'aggiunta dei FieldDef eventTimeZoneOffset, recordTime, eventTimeZoneOffset calcolati senza chiedere nulla all'utente e il FieldDef bizStep di tipo "commissioning".

4.3.8 Implementazione dell'app per il tracciamento di un prodotto

Questa app serve per osservare e quindi tracciare un determinato prodotto che si trova in un determinato momento del suo ciclo di vita. Si compone di due activity principali visibili in Figura 4.10, la prima serve per scansionare un prodotto identificato da un codice a barre o da un QR code e la seconda svolge le stesse operazioni con la differenza che il prodotto è recuperabile per mezzo di un tag RFID.

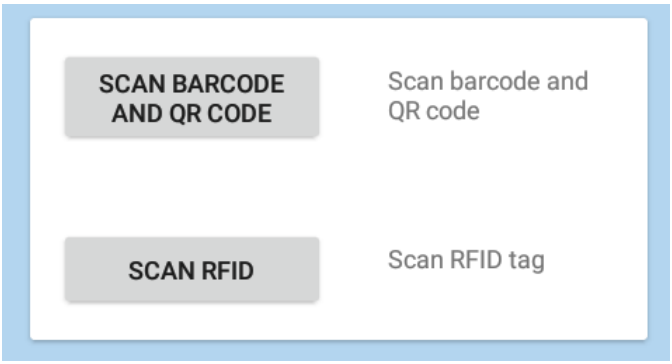

Figura 4.10: Scansione barcode, QR code e tag RFID

Si è scelto di separare queste due activity perchè la logica di scansione è differente e si usano librerie diverse, che porterebbe ad una difficile gestione del dispositivo se fossero unificate. A differenza della prima app descritta, questa non ha bisogno di avere a che fare con la generazione di un codice ma si limita a trasformare i codici scansionati in codici EPC, sempre però controllando dall'event template che i codici scansionati siano validi, per non accettare qualsiasi codice. Anche quest'app deve permettere all'utente di riempire i campi della lista ilmd con i valori non recuperabili da nessuna altra parte se non dall'utente, con l'eccezione della

location che può essere selezionata da uno spinner. Dopo la scansione e la verifica del codice è possibile anche scattare una foto del prodotto per avere maggiore chiarezza su quello che si sta tracciando. A questo punto l'evento è salvato sul database locale con conseguente invio e cancellazione dal database se le condizioni di rete lo permettono. Il caso d'uso in questione riguarda il trattamento di salute fatto al bestiame che è da tracciare nella vita degli animali. Nelle Figure 4.11 e 4.12 si vede l'activity relativa alla scansione dei tag RFID che non si differenzia molto a livello visivo da quella per la scansione di codici a barre e QR code, ma solo a livello di codice implementato.

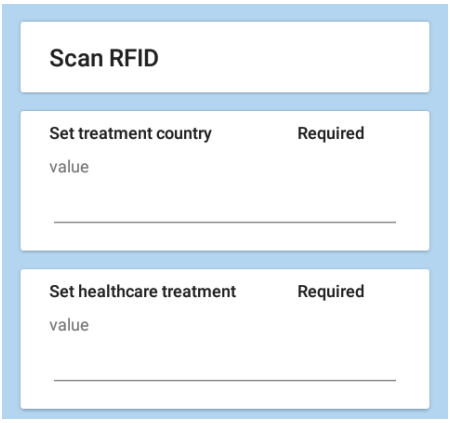

Figura 4.11: App per il tracciamento di un prodotto 1

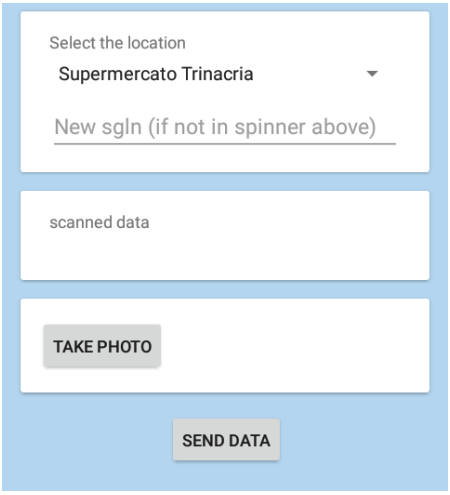

Figura 4.12: App per il tracciamento di un prodotto 2

In questo particolare caso dall'event template si sono recuperati gli AttrDef che sono tutti obbligatori da riempire con dei valori, come è possibile vedere dal marchio

"required", consistono negli attributi treatment country e healthcare treatment. La parte di selezione della location, dove questa scansione ha avuto luogo, è identica a quella descritta nella prima app in quanto i diversi campi del FieldDef bizLocation nell'event template hanno gli stessi valori. La location è usata per valorizzare il FieldDef bizLocation. Dopo la scansione del tag RFID (o nell'altra activity, del codice a barre o QR code), che viene poi trasformato in un codice EPC in fase di creazione dell'evento EPCIS da mandare per valorizzare il FieldDef epcList, è possibile scattare una foto per avere una panoramica migliore dal momento che questa app serve proprio per osservare un prodotto. Premendo il bottone "TAKE PHOTO" si apre l'app fotocamera del dispositivo TC20 e dopo la cattura della foto, quest'ultima viene mandata al server che restituisce l'id usato per popolare un AttrDef. Questa operazione può essere fatta solo in condizioni online in quanto non sarebbe possibile mandare la foto al server ed avere il suo id altrimenti. Premendo il bottone "SEND DATA" viene creato l'evento EPCIS utilizzando il template dal nome "healthcare_treatment" formato dai FieldDef e AttrDef descritti, con l'aggiunta dei FieldDef eventTimeZoneOffset, recordTime, eventTimeZoneOffset calcolati dal sistema e il FieldDef bizStep di tipo "other".

4.3.9 Implementazione dell'app per l'impacchettamento di prodotti

Un altro processo logistico da gestire in questo progetto è quello dell'impacchettamento dei prodotti. L'app che svolge questo compito ha alcune differenze rispetto a quella che osserva/traccia i prodotti. Come quest'ultima, è composta da due activity principali, la prima per la scansione di codici a barre e QR code e l'altra che lavora con i tag RFID. Il distaccamento di queste due activity è dovuta sempre ai motivi già introdotti, quindi la difficoltà di gestione del dispositivo che dovrebbe gestire contemporaneamente più tipologie di scansione. La differenza sostanziale, almeno a livello grafico è che questa app rispetto a quella di tracciamento deve scansionare obbligatoriamente più codici. Infatti come prima cosa bisogna scansionare il parent, che può essere visto come il contenitore degli atri prodotti, quindi dopo la verifica di una corretta scansione del parent si procede alla scansione degli altri prodotti da inserire all'interno. Una prima verifica da fare è quella che l'identificativo del parent non sia tra quelli dei prodotti interni, per evitare il paradosso di avere un prodotto all'interno di se stesso, poi si prosegue con la verifica dei prodotti, avendo cura di non avere un contenitore vuoto per questo processo logistico. La gestione e la creazione dell'evento EPCIS è uguale a tutte le altre app con l'eccezione dell'event template che si differenzia in base al processo. Il caso d'uso considerato verte sul posizionamento del bestiame all'interno di un recinto oppure di un mezzo per il suo trasporto. Nella Figura 4.13 si vede visivamente l'activity che gestisce la scansione

dei codici a barre e dei QR code (l'activity che gestisce i tag RFID a livello visivo e di interazione con l'utente è molto simile).

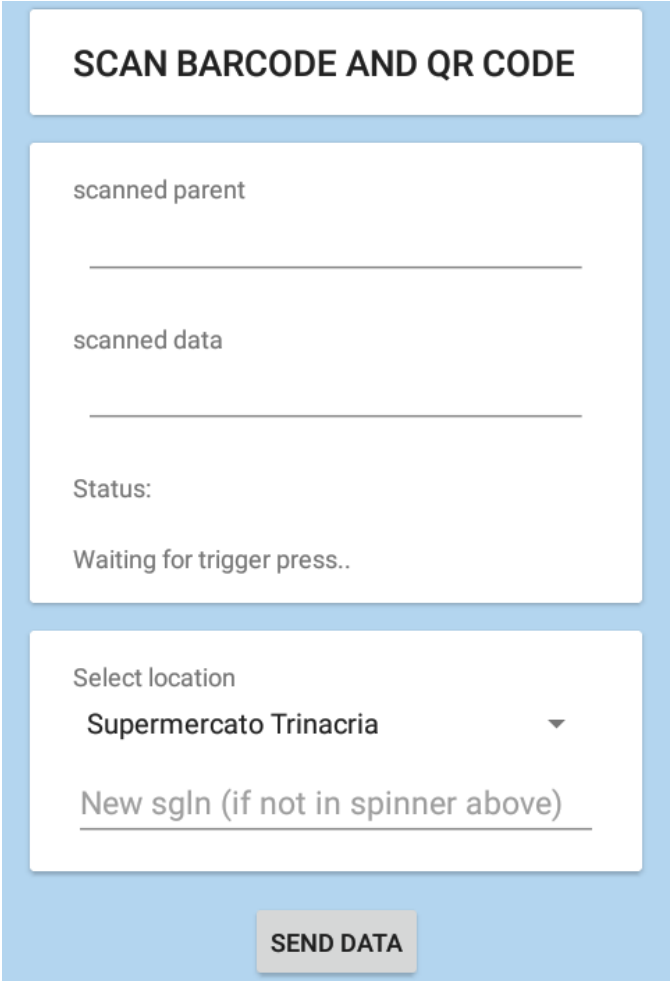

Figura 4.13: App per l'impacchettamento di prodotti

Come è possibile notare, in questo caso l'event template non contiene la lista ilmd, infatti non ci sono AttrDef da valorizzare da parte dell'utente. Si devono scansionare almeno due codici, il primo scansionato rappresenta il codice del parent e quindi quello del "contenitore" dove va introdotto il bestiame, mentre le scansioni successive sono quelle relative ai singoli bovini da inserire all'interno. Questi codici nel caso di QR code rappresentano già i codici EPC con i quali vengono valorizzati i FieldDef parentID e childEPCs, mentre nel caso di codici a barre, devono prima essere trasformati in codici EPC validi. Per quanto riguarda la location, il funzionamento è analogo a quanto descritto nelle precedenti app, in cui al FieldDef bizLocation viene fornito il codice della location che l'utente seleziona

dallo spinner o da tastiera. Alla pressione del bottone "SEND DATA" viene creato l'evento EPCIS dal template chiamato "packing_of_items" composto dai FieldDef descritti, con l'aggiunta dei FieldDef eventTimeZoneOffset, recordTime, eventTimeZoneOffset calcolati senza chiedere nulla all'utente e il FieldDef bizStep di tipo "packing".

4.3.10 Implementazione dell'app per la verifica di un ordine

L'ultima app sviluppata è quella per la verifica di un ordine, per sapere se tutti i prodotti in questione sono presenti. Ancora una volta, sono presenti due activity per la diversa tipologia di scansione. Per prima cosa l'app deve scansionare l'identificativo dell'ordine e di conseguenza andare a recuperare gli identificativi dei prodotti che lo formano. Questa procedura deve per forza essere svolta online perchè non si può conoscere in anticipo il codice dell'ordine che si scansionerà. Dopo aver salvato in una lista tutti i prodotti dell'ordine, si procede con la scansione dei prodotti che si hanno effettivamente di fronte, avendo la cura di salvare anche i loro identificativi. Al momento della verifica dell'ordine viene fatto un controllo e notificato l'utente dei seguenti possibili casi:

- Tutti i prodotti dell'ordine sono presenti dopo la scansione.
- Tra i prodotti dell'ordine risultano alcuni prodotti che non sono stati scansionati, quindi non sono presenti tutti i prodotti.
- Tra i prodotti scansionati risultano alcuni non presenti tra i prodotti dell'ordine, quindi ci sono più prodotti del dovuto.

Come caso d'uso per quest'ultima app è stato preso in considerazione la ricezione del bestiame acquistato. Nelle Figure 4.14 e 4.15 si vede l'activity relativa alla scansione di tag RFID, ricordando anche in questo caso che quella relativa a codici a barre e QR code è analoga.

Anche in questo caso, come nell'app per l'impacchettamento di prodotti, la lista ilmd dell'event template è vuota, quindi non sono presenti degli AttrDef da richiedere all'utente. Si nota subito la differenza con le altre app per quanto riguarda la location. In questo caso sono presenti tre tipologie, la prima rappresenta dove la scansione sta avendo luogo, la seconda dove l'ordine è partito e la terza dove l'ordine è arrivato o deve arrivare. La gestione delle diverse location è simile alle altre app con la differenza che vengono valorizzati rispettivamente i FieldDef bizLocation, sourceList e destinationList. Per prima cosa si deve scansionare il tag dell'ordine per recuperare dunque, trasformando il tag RFID scansionato, il codice EPC usato per valorizzare il FieldDef bizTransactionList e in seguito vengono scansionati

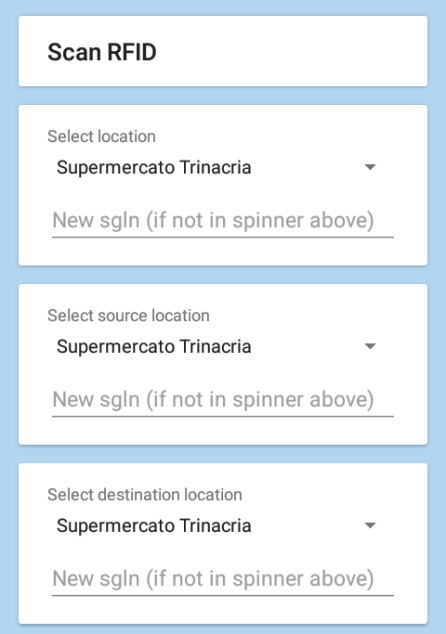

Figura 4.14: App per la verifica di un ordine 1

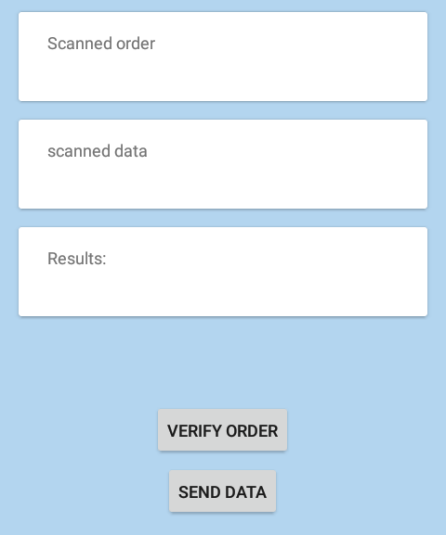

Figura 4.15: App per la verifica di un ordine 2

tutti i prodotti, in questo caso i bovini presenti davanti all'operatore, in modo tale da riempire il FieldDef epcList con i relativi codici EPC. Se si preme il bottone "VERIFY ORDER" la CardView dei risultati viene valorizzata dal risultato della verifica dell'ordine mentre una volta premuto il tasto "SEND DATA" viene creato

l'evento EPCIS relativo al template dal nome "receiving_of_purchased_cattle" costituito dai FieldDef descritti, con l'aggiunta dei FieldDef eventTimeZoneOffset, recordTime, eventTimeZoneOffset calcolati dal sistema e il FieldDef bizStep di tipo "receiving".

Capitolo 5

Conclusioni

Per la realizzazione di questa tesi sono stati affrontati tanti argomenti e sono stati messi alla luce diversi aspetti della programmazione mobile, nello specifico della programmazione Android. Si può sottolineare come queste applicazioni siano un buon punto di partenza per la realizzazione di un prodotto in continua evoluzione rispetto alle funzionalità implementate.

Le caratteristiche di facilità di utilizzo e di gestione del tempo e dello spazio che sono state prefissate dall'inizio del lavoro, in fase di progettazione, sono state portate a termine sviluppando applicazioni intuitive e semplici nei contenuti, alla portata di tutti. Molto utile è stato il software messo a disposizione da Zebra per programmare al meglio i dispositivi utilizzati, riadattando il codice fornito insieme alle librerie in base alle caratteristiche da ottenere.

L'obiettivo della tesi è stato quello di automatizzare i processi in un contesto logistico, spesso soggetti ad errori dovuti all'intervento umano, trattando i dati nel modo corretto e risparmiando tempo di operazione del processo e spazio di archiviazione nei dispositivi che portano al loro interno solo le applicazioni e i dati indispensabili al lavoro che devono eseguire.

Per il raggiungimento di questo obbiettivo è stato fondamentale capire le modalità di lavoro attuali per trovare delle soluzioni appropriate. Sono stati analizzati i dati e le varie fasi della loro vita (raccolta, trattamento, interpretazione, invio e ricezione) per seguire al meglio lo standard EPCIS che tratta ampiamente il contesto di questa tesi. Poi è stato studiato e adattato il software che permette l'interazione con questi dati, sfruttando il massimo dai dispositivi e del sistema Android con i suoi linguaggi di programmazione utilizzati nel momento opportuno. Sono state garantite le buone norme di programmazione Android che includono i modelli architetturali e quelli di progettazione moderni. Anche la sicurezza è stata gestita implementando delle applicazioni che fanno uso di TLS e dei token JWT per accedere alle risorse del backend.

Sono state sviluppate delle applicazioni base, prendendo in considerazione dei casi d'uso specifici che hanno un contesto che prevede come prodotto da trattare il bestiame e tutte le sue fasi di vita all'interno dell'azienda. Anche se le soluzioni ai processi trattati soddisfano pienamente gli obbiettivi prefissati, fornendo delle soluzioni facilmente adattabili con poco sforzo a molti altri processi, di seguito sono riportati alcuni elementi che potranno essere presi in considerazione per migliorare il prodotto e perfezionare il servizio offerto nella sua complessità.

5.1 Possibili sviluppi futuri

Dopo l'analisi del software sviluppato, è possibile evidenziare alcuni possibili sviluppi futuri, mirando ad estendere le funzionalità esistenti e inserire anche nuove funzioni per migliorare la qualità del prodotto proposto. Una cosa da prendere in considerazione sicuramente è l'implementazione di altre app per gestire i processi lasciati fuori da questa tesi, prendendo spunto dalle app sviluppate, dal momento che i dispositivi avranno sempre a che fare con la scansione di tag RFID, codici a barre e QR code e le app prodotte posseggono già tutti i meccanismi per operare con questi oggetti.

Come è stato descritto, il design minimalistico delle app è stato voluto per semplificare il loro utilizzo e prevedendo solo le informazioni necessarie, però si potrebbe avere un design minimalistico esteticamente migliorato sfruttando anche la personalizzazione da parte dell'utente. Inoltre le app che permettono la navigazione con il Navigation Drawer potrebbero suggerire nuove schermate come ad esempio qualcuna che prevede delle statistiche o dei grafici sul processo logistico in questione.

Sicuramente il lavoro è stato svolto per permettere l'evoluzione continua delle app garantendo le funzionalità necessarie, ma aprendo la strada anche ad implementazioni future.

Bibliografia

[1] URL: https://www.produzioneagile.it/i-5-letali-errori-sulla-raccolta-dati (cit. a p. 1).

[2] URL: https://www.optit.net/2020/06/05/lottimizzazione-della-logistica-nel-contesto-dellemergenza-sanitaria (cit. a p. 2).

[3] URL: https://gs1it.org/assistenza/standard-specifiche/epcis-tracciabilita-prodotti-tempo-reale (cit. a p. 5).

[4] URL: https://blogs.sap.com/2013/12/16/standalone-task-manager-for-lifecycle-management-automation (cit. a p. 6).

[5] URL: https://developer.android.com/studio (cit. a p. 12).

[6] URL: https://www.zebra.com/gb/en/products/spec-sheets/mobile-computers/handheld/tc20.html (cit. a p. 12).

[7] URL: https://www.zebra.com/gb/en/products/spec-sheets/rfid/rfid-handhelds/rfd2000.html (cit. a p. 13).

[8] URL: https://www.mongodb.com (cit. a p. 17).

[9] URL: https://www.docker.com (cit. a p. 18).

[10] URL: https://swagger.io (cit. a p. 19).

[11] URL: https://square.github.io/okhttp (cit. a p. 40).