



UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea in Informatica

**Progettazione e sviluppo di una APP
android per il supporto alla
coltivazione in terreni salini.**

Relatori:
Prof. Paolo Milazzo
Prof. Antonella Castagna

Presentata da:
Federico Bernacca

**Sessione autunnale
Anno Accademico 2019/2020**

Indice

1	Introduzione	4
2	Android	6
2.1	Background	6
2.1.1	Linux	7
2.1.2	Embedded Linux	7
2.2	Storia	8
2.3	Funzionalità	8
2.3.1	Interfaccia	8
2.3.2	Applicazioni	9
2.3.3	Gestione della memoria	9
2.4	Sviluppo	10
2.4.1	Kernel Linux	10
2.5	Android App	11
2.5.1	Struttura di un App	12
2.5.2	Componenti di un App	12
2.5.2.1	Activity	12
2.5.2.2	Ciclo di vita di un'Activity	15
2.5.2.3	Fragment	16
2.5.2.4	Intent	16
2.5.2.5	Dialog	17
3	Strumenti utilizzati	18
3.1	Linguaggi di programmazione	18
3.1.1	Java	18

3.1.2	Java Swing	19
3.1.3	XML	19
3.1.4	JavaScript Object Notation	20
3.2	Integrated development environment	20
3.2.1	IntelliJ IDEA	20
3.2.2	Android Studio	20
3.2.2.1	Features	21
3.3	Firebase	22
3.4	Google Maps Platform	22
4	Implementazione	24
4.1	Cloud Firestore	24
4.2	App	26
4.2.1	Point.java	28
4.2.2	Field.java	31
4.2.3	FieldWithDate.java	32
4.2.4	MainActivity.java	34
4.2.4.1	Connessione a Cloud Firestore	36
4.2.5	MyDialogFragment.java	40
4.2.6	MapsActivity.java	42
4.2.6.1	Perimetro del campo	44
4.2.6.2	Punti del campo	46
4.2.6.3	Analisi dei punti	53
4.2.6.4	Heat map	59
4.2.6.5	QR code	62
4.2.6.6	Salvatggio su Cloud Firestore	63
4.2.7	Programma desktop	64
5	Test	67
6	Conclusioni	69

Ringraziamenti

Per questa speciale occasione che mi ha permesso il raggiungimento di un grande traguardo, desidero porgere i miei più sinceri ringraziamenti ai relatori di questo tirocinio: Paolo Milazzo e Antonella Castagna, che si sono dimostrati disponibili e pronti a darmi qualunque tipo di consiglio.

Vorrei inoltre ringraziare i più fedeli ed instancabili compagni di studio, nonché amici, Fabio e Matteo.

Impossibile non menzionare Andrea, che in quella sera di estate mi ha dato la giusta spinta.

Un grazie agli amici di una vita.

Un grazie a chi mi ha accompagnato lungo questo Percorso, anche se per un breve tragitto.

Per ultimi, ma non ultimi, vorrei ringraziare la mia famiglia, nulla di tutto questo sarebbe accaduto senza il loro continuo sostegno; grazie.

Financial support has been provided by PRIMA, a programme supported by European Union Member States, Horizon 2020 Associated Countries and Mediterranean Partner Countries, in the context of the project "Optimization of Halophyte-based Farming systems in salt-affected Mediterranean Soils (HaloFarMs)".

Capitolo 1

Introduzione

Il tirocinio descritto in questa tesi è stato svolto presso il Dipartimento di Informatica dell'Università di Pisa in collaborazione con quello di Agraria. Il lavoro è stato condotto nell'ambito del progetto PRIMA Section 2 call multi-topics 2019 Halofarms (Development and Optimization of Halophyte-based Farming systems in salt-affected Mediterranean Soils).

L'obiettivo è stato quello di creare un'applicazione Android che assista il coltivatore durante la coltivazione; le specifiche sono state le seguenti.

Nell'ambito di un progetto in corso di svolgimento presso il Dipartimento di Agraria si sta studiando come utilizzare piante alofite¹ per consentire le coltivazioni orticole anche in terreni salini. L'idea è di studiare possibili abbinamenti di alofite e altre piante che rendano la coltivazione possibile. Per fare questo è necessario partire da una misurazione del livello di salinità del terreno che consenta di valutare quale strategia di utilizzo delle piante alofite possa essere più efficace.

¹Piante capaci di assorbire elevate quantità di sale dal terreno, abbassandone così la concentrazione.

Per supportare l'agricoltore in queste misurazioni e durante la coltivazione, l'idea è stata quella di realizzare una app Android che consenta di:

1. Identificare il perimetro dell'area da coltivare, utilizzando il GPS.
2. Determinare i punti in cui effettuare campionamenti di terreno per testarne la salinità (assistendo anche il coltivatore nell'etichettatura dei campioni).
3. Sulla base dei risultati delle analisi di salinità, mostrare una heat map del terreno per suggerire la metodologia di coltivazione più adatta per ogni area.
4. Mantenere uno storico dei risultati delle analisi di salinità in un database online.

Inoltre, è stato scritto un programma desktop che permette la gestione del database.

Il seguito di questa relazione è organizzato come segue.

- Nel Capitolo 2 viene intrapreso un approccio bottom-up: inizialmente vengono introdotti concetti utili per spiegare cos'è Android e com'è nato; viene poi introdotto il concetto di applicazione mobile fino ad arrivare ai concetti chiave per lo sviluppo di esse.
- Nel Capitolo 3 vengono illustrati i linguaggi di programmazioni e gli strumenti software utilizzati rispettivamente per lo sviluppo dell'applicazione e del programma di gestione del database. Verranno inoltre illustrate le librerie esterne utilizzate.
- Nel Capitolo 4 viene descritta in modo dettagliato l'implementazione del database, dell'app e del software gestionale, illustrando le principali funzionalità di cui sono composte.
- Nel Capitolo 5 vengono illustrati i test effettuati durante lo sviluppo dell'app.
- Nel Capitolo 6 si hanno le conclusioni.

Capitolo 2

Android

2.1 Background

In questo capitolo verranno illustrati i concetti alla base del tirocinio svolto; si parte da una breve panoramica su quello che c'è sotto ad Android, alle App e al loro sviluppo.

Riportiamo innanzitutto i concetti chiave alla base di Android: Linux e le sue derivazioni, seguendo un approccio bottom-up descritto dal seguente grafo.

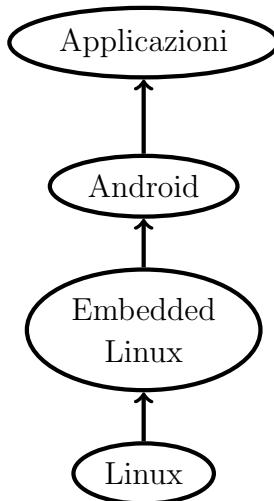


Figura 2.1: Approccio Bottom-Up: da Linux alle Applicazioni.

2.1.1 Linux

Linux è un sistema operativo, ovvero quell'insieme di programmi essenziali per far funzionare il computer e farci qualcosa di utile o divertente; è una alternativa a Windows e a MacOS, e può essere installato al loro posto (o insieme, sullo stesso computer).

Più in generale Linux è il primo rappresentante del software cosiddetto “libero”, ovvero quel software che viene distribuito con una licenza che ne permette non solo l'utilizzo da parte di chiunque ed in qualsiasi circostanza ma anche la modifica, la copia e l'analisi.

Linux è tipicamente usato come termine generico per indicare un sistema operativo con determinate qualità, nel concreto esistono le distribuzioni. Queste sono raccolte di software selezionato e predisposto per essere installato ed utilizzato nel modo più semplice possibile da parte degli utenti, fornendo una serie di strumenti essenziali per iniziare fin dall'inizio a usare il proprio PC nel pieno del potenziale.

Le distribuzioni Linux sono innumerevoli, ed ognuna si differenzia dall'altra per le scelte sul software installato di default, per le configurazioni iniziali, per essere maggiormente orientate alla facilità di utilizzo o all'ottimizzazione delle risorse del computer[2].

2.1.2 Embedded Linux

Un sistema embedded identifica genericamente tutti quei sistemi elettronici di elaborazione a microprocessore progettati appositamente per un determinato utilizzo (special purpose), ovvero non riprogrammabili dall'utente per altri scopi, spesso con una piattaforma hardware ad hoc, integrati nel sistema che controllano e in grado di gestirne tutte o parte delle funzionalità richieste.

Con il termine Embedded Linux ci si riferisce a quell'insieme di distribuzioni linux concepite per essere utilizzate su sistemi embedded, cioè integrati. Le caratteristiche principali di tali sistemi impongono dei vincoli molto severi al sistema operativo in termini di memoria flash occupata, memoria centrale necessaria, tempi di avvio brevi.

2.2 Storia

Nell’ottobre 2003 Andy Rubin, Rich Miner, Nick Sears e Chris White, fondarono una società, l’Android Inc. per lo sviluppo di software per dispositivi mobili.

Il 17 agosto 2005 Google LLC acquisì l’azienda[16], in vista del fatto che la società di Mountain View desiderava entrare nel mercato della telefonia mobile. È in questi anni che il team di Rubin comincia a sviluppare un sistema operativo per dispositivi mobili basato sul kernel Linux. La presentazione ufficiale del “robottino verde” avvenne il 5 novembre 2007 dalla neonata OHA (Open Handset Alliance), un consorzio di aziende del settore Hi Tech.

2.3 Funzionalità

Android è un sistema operativo per dispositivi mobili sviluppato da Google e basato sul kernel Linux, da considerarsi propriamente una distribuzione embedded Linux e non un sistema unix-like né una distribuzione GNU/Linux (dato che la quasi totalità delle utilità GNU è sostituita da software in Java)[4] [19] [22], progettato principalmente per sistemi embedded quali smartphone e tablet.

Ad aprile 2017 è il sistema operativo per dispositivi mobili più diffuso al mondo, con una fetta di mercato attestata a quota 62,94% sul totale, seguito da iOS con il 33,9%[8].

2.3.1 Interfaccia

L’interfaccia utente di Android è basata sul concetto di manipolazione diretta per cui si utilizzano gli ingressi mono e multi-touch come strisciare, tocchi e pizzichi sullo schermo per manipolare gli oggetti visibili sullo stesso[10]. La risposta all’input dell’utente è stata progettata per essere immediata e tentare di fornire un’interfaccia fluida.

La schermata principale, identificata con un’icona che rappresenta una casa, è quella in cui ci si trova appena il dispositivo è stato avviato, oppure

premendo il tasto Home. Questa è in genere occupata dalle icone delle applicazioni. La schermata principale vera e propria (cioè la schermata iniziale) può essere integrata da altre pagine tra cui l'utente può scorrere avanti e indietro.

Componente classico del mondo Android è il Launcher ovvero l'applicazione di sistema che sovrintende e gestisce essenzialmente la schermata principale e, secondariamente, le scorciatoie (shortcut), il cassetto delle applicazioni (app drawer), la barra inferiore e la barra di stato, il menù notifiche e le impostazioni rapide. Oltre a quello predefinito esistono numerosi launcher di terze parti che offrono una vasta gamma di personalizzazioni.

2.3.2 Applicazioni

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android.

Verranno analizzate in dettaglio in una delle prossime sezioni.

2.3.3 Gestione della memoria

Poiché i dispositivi Android sono generalmente alimentati a batteria, Android è progettato per gestire i processi e per ridurre al minimo il consumo di energia. Quando un'applicazione non è in uso, il sistema ne limita l'uso di risorse in modo tale che, sebbene disponibile per l'uso immediato, non utilizza quantitativi significativi di batteria o risorse della CPU[14][20]. Android gestisce automaticamente le applicazioni archiviate in memoria: quando la memoria è insufficiente, il sistema inizia, in modo invisibile, a chiudere automaticamente i processi inattivi, a partire da quelli che sono rimasti inattivi per più a lungo[13] [21]. Nel 2011 Lifehacker riferì che le applicazioni task killer di terze parti fanno più danni che benefici[12].

2.4 Sviluppo

2.4.1 Kernel Linux

Come già accennato, Android è costituito da un kernel Linux, con Librerie e API scritte in C (o C++) e software in esecuzione su un framework di applicazioni che include librerie Java. Android fino alla release 4.4 KitKat ha usato la Dalvik virtual machine con un compilatore just-in-time¹ per l'esecuzione di Dalvik dex-code (Dalvik Executable), che di solito viene tradotto da codice bytecode Java[15]. Questa virtual machine, ormai obsoleta, viene rimpiazzata con ART (Android RunTime) integrato in Android 5.0 Lollipop. Questa nuova virtual machine, sviluppata da Google, utilizza un compilatore Ahead-of-time (atto di compilazione di un programma da un linguaggio ad alto livello (come il C o il C++) o da un linguaggio intermedio (come Java bytecode) in un binario dipendente dal sistema).

La compilazione JIT può rallentare le prestazioni dell'esecuzione dato che il programma prima di essere eseguito deve venir compilato dal sistema. La compilazione ahead-of-time sposta la fase compilazione prima dell'esecuzione, tipicamente durante l'installazione del programma. Quindi durante l'installazione del programma il codice in linguaggio intermedio viene compilato in codice binario nativo della piattaforma. Questo permette di eseguire il programma usando codice binario nativo evitando la fase di compilazione durante l'esecuzione del programma e in generale migliorando le prestazioni e la reattività dei programmi.

¹La compilazione viene effettuata durante l'esecuzione del programma piuttosto che precedentemente.

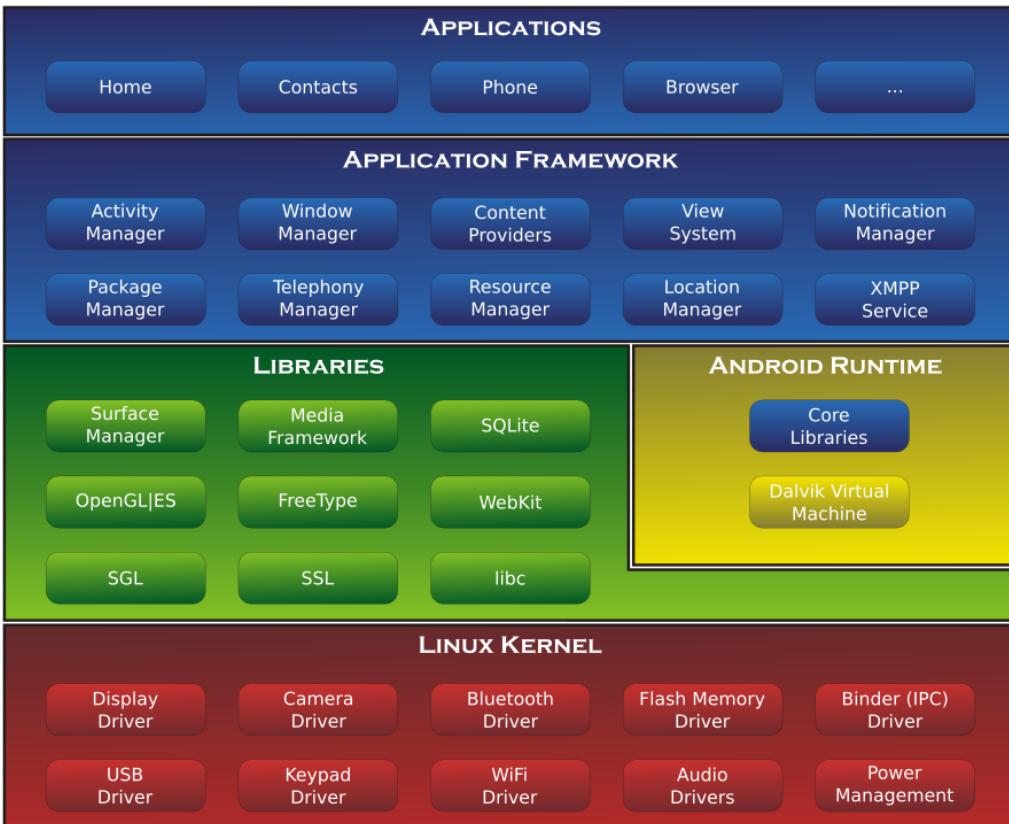


Figura 2.2: Schema di Architettura

2.5 Android App

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android. Esse possono essere scaricate sia dal catalogo ufficiale Google Play, sia da altri cataloghi. Le applicazioni Android possono anche essere installate direttamente a partire da un file APK fornito dal distributore del software.

Le applicazioni Android sono Java-based; in effetti le applicazioni scritte in codice nativo in C/C++ devono essere richiamate dal codice java, tutte le chiamate a sistema fatte in C (o C++) devono chiamare codice virtual machine Java di Android: infatti le API multimediali di SDL (libreria libe-

ra multimediale multi piattaforma, scritta in C) sotto Android richiamano metodi in Java; questo significa che il codice dell'applicazione C/C++ deve essere inserito all'interno di un progetto Java, il quale produce alla fine un pacchetto Android (APK).

Per comprendere al meglio il capitolo sull'implementazione è necessario definire qualche concetto chiave riguardante lo sviluppo di un App Android.

2.5.1 Struttura di un App

Un applicazione può assumere diverse forme nel corso della sua vita:

- **In sviluppo:** il layout dell'app è sul disco (progetto).
- **In deployment:** l'App è trasformata nel file .apk.
- **In esecuzione:** la struttura si trova in memoria.

Le varie forme sono legate da tre processi:

- **Build:** da sorgente a .apk
- **Deploy:** scaricamento dell'.apk, ad esempio da un market, sul device.
- **Run:** l'App è in run, il processo è in memoria[17].

2.5.2 Componenti di un App

Le applicazioni Android non sono un blocco monolitico, ma un insieme di componenti cooperanti; vengono riportate quelle utilizzate all'interno dell'App sviluppata.

2.5.2.1 Activity

La classe Activity è un componente cruciale di un'app Android e il modo in cui vengono avviate e messe insieme è una parte fondamentale dell'applicazione. A differenza dei paradigmi di programmazione in cui le app vengono avviate con un metodo main(), il sistema Android avvia il codice in un'istanza Activity richiamando metodi di callback specifici che corrispondono a fasi specifiche del suo ciclo di vita.

L'esperienza dell'app per dispositivi mobili è diversa dalla sua controparte desktop in quanto l'interazione di un utente con l'app non inizia sempre nello stesso posto ma spesso in modo non deterministico. Ad esempio, aprendo un'app di posta elettronica dalla home, viene visualizzato un elenco di e-mail; invece, utilizzando un'app di social media che avvia l'app di posta elettronica, si potrebbe andare direttamente alla schermata dell'app per comporre un'e-mail.

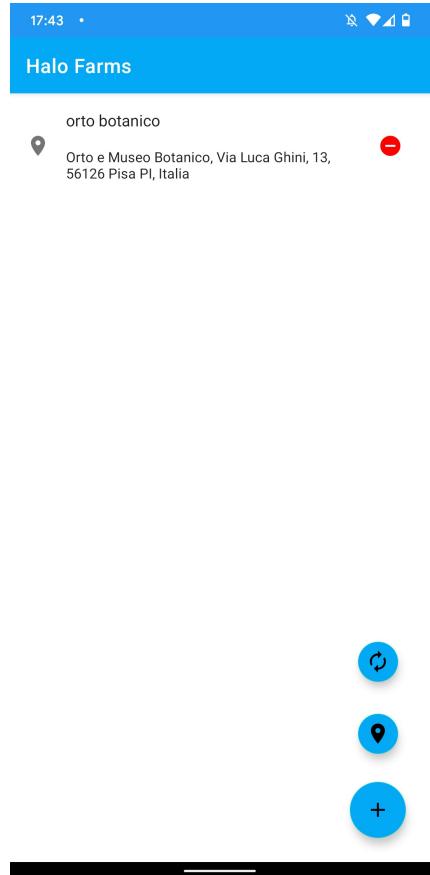
La classe Activity è progettata per facilitare questo paradigma. Quando un'app ne richiama un'altra, l'app chiamante richiama un Activity nell'altra app, piuttosto che l'app nel suo insieme atomico. In questo modo, l'Activity funge da punto di ingresso per l'interazione di un'app con l'utente. Si implementa un'Activity come sottoclassificazione della classe Activity.

Un'Activity fornisce la finestra in cui l'app disegna la sua interfaccia utente. Questa finestra in genere riempie lo schermo, ma potrebbe essere più piccola dello schermo e rimanere mobile sopra le altre finestre.

La maggior parte delle app contiene più schermate, il che significa che comprendono più Activity. In genere, un'Activity in un'app viene specificata come principale, ovvero la prima schermata che viene visualizzata quando l'utente avvia l'app. Ognuna può quindi avviare altre per eseguire azioni diverse.

Sebbene le Activity lavorino insieme per formare un'esperienza utente coerente in un'app, ognuna di esse è legata solo vagamente alle altre; di solito ci sono dipendenze minime tra esse. Queste infatti, spesso, ne avviano di appartenenti ad altre app.

Riassumendo, un'Activity può essere vista come un'attività atomica dell'utente concretizzata da una schermata[6][18].



(a) Main Activity



(b) Maps Activity

Figura 2.3: Esempi di Activity tratte dall'App sviluppata; cliccando sull'item “orto botanico” si passa dall'Activity *a* all'Activity *b*.

2.5.2.2 Ciclo di vita di un'Activity

Quando un utente naviga, esce e torna in un'app, le istanze di Activity passano attraverso diversi stati nel loro ciclo di vita. Questa classe fornisce una serie di callback che consentono all'Activity di sapere che uno stato è cambiato; ovvero che il sistema sta creando, arrestando, riprendendo un'Activity o distruggendo il processo in cui essa risiede.

All'interno dei metodi di callback del suo ciclo di vita, si può dichiarare come si comporta quando l'utente esce o rientra. In altre parole, ogni richiamata consente di eseguire un lavoro specifico appropriato per un determinato cambiamento di stato. Fare il lavoro giusto al momento giusto e gestire correttamente le transizioni rende l'app più robusta e performante[11].

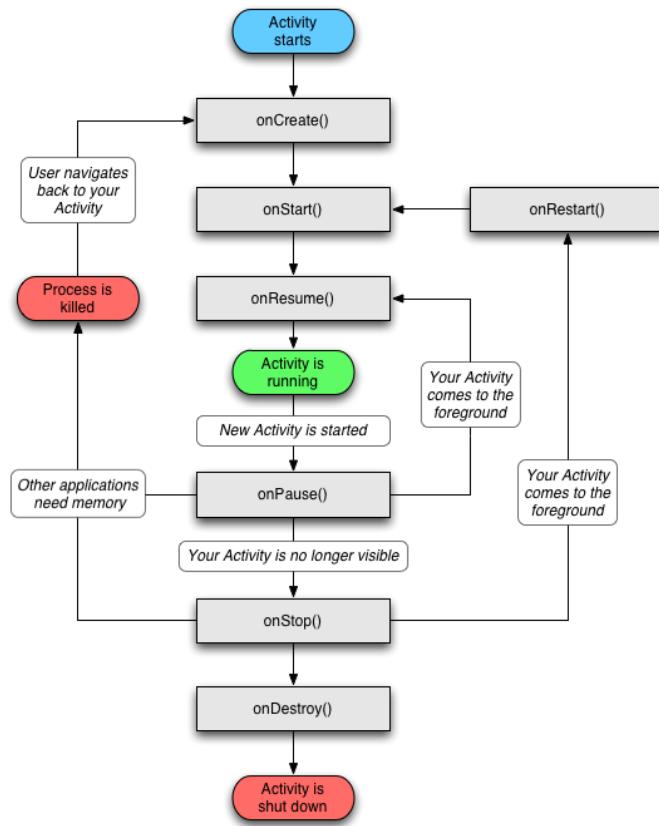


Figura 2.4: Ciclo di vita di un'Activity

2.5.2.3 Fragment

Un Fragment rappresenta una parte dell’interfaccia utente. Si possono combinare più Fragment in una singola Activity per creare una UI con più riquadri e riutilizzare un Fragment in più Activity. Può essere visto come una sezione modulare di un’Activity, che ha un proprio ciclo di vita, riceve i propri eventi di input e che si può aggiungere o rimuovere mentre l’Activity è in esecuzione.

Un Fragment deve essere sempre ospitato in un’Activity e il ciclo di vita del frammento è direttamente influenzato da quello dell’host. Ad esempio, quando l’Activity è in pausa, o quando viene distrutta, lo sono anche tutti i Fragments. Tuttavia, mentre un’Activity è in esecuzione, è possibile manipolare ogni Fragment in modo indipendente, ad esempio aggiungendolo o rimuovendolo[5].

2.5.2.4 Intent

I componenti di un’applicazione dialogano attraverso un sistema di messaggistica che è alla base di Android.

Un Intent è un “messaggio” che esprime un’intenzione dell’utente o di una applicazione affinché qualcosa avvenga.

Si hanno numerosissimi Intent di sistema, ed ogni app può definirne altri. Una parte della struttura del messaggio è fissata, ma possono includere dati “extra” a piacere, ad esempio per “trasportare” dati da un Activity all’altra. Gli Intent dunque possono essere indirizzati a uno specifico componente, oppure emessi in broadcast.

Una tipica applicazione Android si esegue seguendo questi passaggi:

1. Il Launcher (che è esso stesso un Activity) avvia la prima Activity dell’App, inviandole un Intent che indica l’intenzione di lanciarla;
2. L’Activity chiama `setLayout()` per impostare la sua UI;
3. Il sistema chiama certi metodi dell’Activity (callback) in risposta alle azioni dell’utente. A seconda dei casi, questi metodi potranno:

- lanciare altre Activity (inviando loro opportuni Intent), sia dell'applicazione sia di altre applicazioni;
- inviare Intent ad Activity già in esecuzione o, in broadcast, a tutti gli interessati;
- terminare l'Activity (tornando alla precedente)[18].

2.5.2.5 Dialog

Una oggetto **Dialog** è una piccola finestra che richiede all'utente di prendere una decisione o di inserire informazioni aggiuntive; non riempie lo schermo e viene normalmente utilizzato per eventi modali che richiedono agli utenti di eseguire un'azione prima di poter procedere.

Capitolo 3

Strumenti utilizzati

Vengono di seguito riportati gli strumenti software utilizzati per lo sviluppo dell'App e del software gestionale del database.

3.1 Linguaggi di programmazione

Il programma di gestione del database è stato scritto interamente in Java, idem per quanto riguarda lo sviluppo dell'App.

Il layout grafico di quest'ultima è stato realizzato utilizzando il meta-linguaggio XML.

3.1.1 Java

Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione (tramite compilazione in bytecode prima e interpretazione poi da parte di una JVM).

Uno dei principi fondamentali del linguaggio è espresso dal motto WORA (write once, run anywhere, ossia “scrivi una volta, esegui ovunque”): il codice compilato che viene eseguito su una piattaforma non deve essere ricompilato per essere eseguito su una piattaforma diversa; infatti il prodotto della compilazione è in un formato chiamato bytecode che può essere eseguito da

una qualunque implementazione di un processore virtuale detto Java Virtual Machine; al 2014 Java risulta essere uno dei linguaggi di programmazione più usati al mondo, specialmente per applicazioni client-server, con un numero di sviluppatori stimato intorno ai 9 milioni[7][9].

Java venne creato per soddisfare cinque obiettivi primari[13]:

1. essere semplice, orientato agli oggetti e familiare;
2. essere robusto e sicuro;
3. essere indipendente dalla piattaforma;
4. contenere strumenti e librerie per il networking;
5. essere progettato per eseguire codice da sorgenti remote in modo sicuro.

3.1.2 Java Swing

Swing è un framework per Java, orientato allo sviluppo di interfacce grafiche. Parte delle classi del framework Swing sono implementazioni di widget (oggetti grafici) come caselle di testo, pulsanti, pannelli e tavole.

La libreria Swing viene utilizzata come libreria ufficiale per la realizzazione di interfacce grafiche in Java. È un'estensione del precedente Abstract Window Toolkit. La differenza principale tra i due è che i componenti Swing sono scritti completamente in codice Java.

Java Swing è stato utilizzato nel programma desktop per permettere la gestione del database dall'amministratore tramite una semplice interfaccia grafica.

3.1.3 XML

XML (eXtensible Markup Language) è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Il nome indica che si tratta di un linguaggio marcatore (markup language) estensibile (eXtensible), in quanto permette di creare tag personalizzati.

3.1.4 JavaScript Object Notation

JavaScript Object Notation (JSON) è un formato di file standard aperto e un formato di interscambio di dati, che utilizza testo leggibile dall'uomo per memorizzare e trasmettere oggetti costituiti da coppie attributo-valore ed array.

JSON è un formato di dati indipendente dal linguaggio. È stato derivato da JavaScript, ma molti linguaggi di programmazione moderni includono codice per generare e analizzare dati in formato JSON.

3.2 Integrated development environment

Un ambiente di sviluppo integrato (IDE) è un software che, in fase di programmazione, supporta i programmatore nello sviluppo e debugging del codice sorgente di un programma[3]: spesso l'IDE aiuta lo sviluppatore segnalando errori di sintassi del codice direttamente in fase di scrittura, oltre a tutta una serie di strumenti e funzionalità di supporto alla fase stessa di sviluppo e debugging.

Per lo sviluppo sono stati utilizzati i seguenti IDE: Android Studio per lo sviluppo dell'App; IntelliJ IDEA per l'applicazione desktop gestionale.

3.2.1 IntelliJ IDEA

IntelliJ IDEA è un ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java. Sviluppato da JetBrains, è disponibile sia in licenza Apache che in edizione proprietaria commerciale. Ogni aspetto di IntelliJ IDEA è stato progettato per massimizzare la produttività degli sviluppatori; l'assistenza intelligente alla codifica e il design ergonomico rendono lo sviluppo non solo produttivo ma anche piacevole.

3.2.2 Android Studio

Android Studio è un ambiente di sviluppo integrato (IDE) per lo sviluppo per la piattaforma Android. È stato annunciato il 16 maggio 2013 in occasione della conferenza Google I/O.

Basato sul software di JetBrains IntelliJ IDEA, Android Studio è stato progettato specificamente per lo sviluppo di applicazioni Android[1]. È l'IDE primario di Google per lo sviluppo nativo di applicazioni Android.

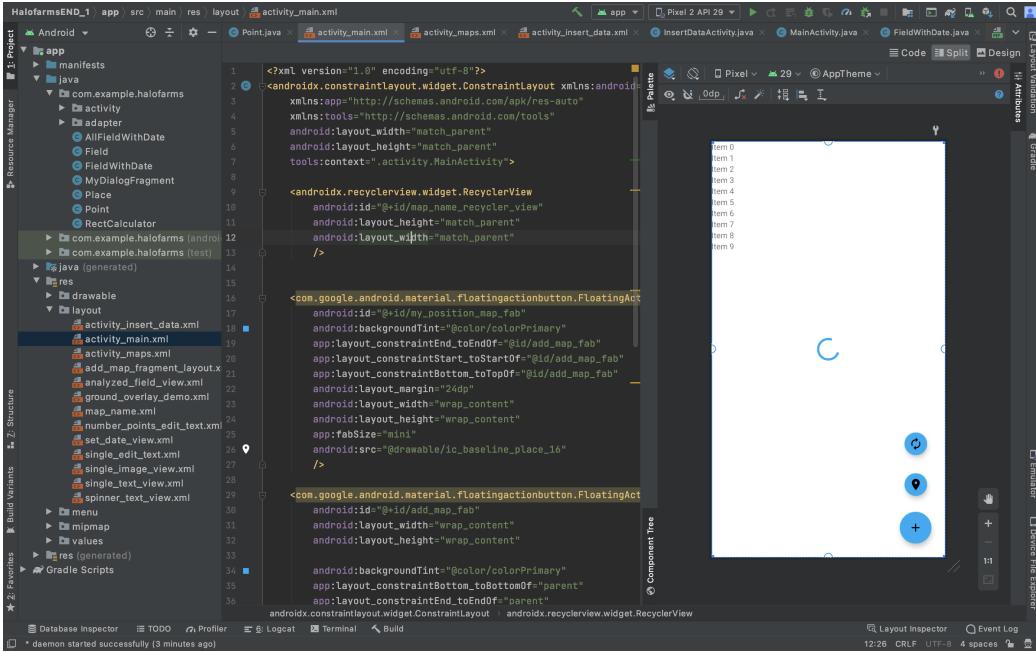


Figura 3.1: Schermata di esempio tratta dall'App sviluppata.

3.2.2.1 Features

Riportiamo le principali features di questo ide:

- Visual layout editor:** consente di creare rapidamente layout trascinando gli elementi dell'interfaccia utente in un editor di progettazione visiva invece di scrivere manualmente l'XML. L'editor di progettazione può visualizzare in anteprima il layout su diversi dispositivi e diverse versioni Android, può ridimensionare dinamicamente il layout per assicurare che funzioni bene su schermi di dimensioni diverse.
- Emulatore Android:** simula i dispositivi Android sul computer in modo che si possa testare l'applicazione su una varietà di dispositivi e livelli di API Android senza bisogno di avere ogni dispositivo fisico;

l'emulatore fornisce quasi tutte le funzionalità di un vero dispositivo Android.

3. **Code editor intelligente:** permette la stesura di codice migliore, lavorare più velocemente e essere più produttivi con un editor che fornisce il completamento del codice per i linguaggi utilizzati come Kotlin, Java e C/C++.

3.3 Firebase

Firebase è una piattaforma per la creazione di applicazioni per dispositivi mobili e web sviluppata da Google. La piattaforma Firebase ha 18 prodotti divisi in tre gruppi: Sviluppo, Qualità, e Crescita.

Nello sviluppo dell'App si sono utilizzati i seguenti servizi di Firebase:

- **Firebase Authentication:** permette agli utenti di eseguire l'accesso con le proprie credenziali, ad esempio tramite account Google.
- **Cloud Firestore:** Archivia e sincronizza i dati tra utenti e dispositivi, su scala globale, utilizzando un database NoSQL¹ ospitato nel cloud. Cloud Firestore offre la sincronizzazione in tempo reale e il supporto offline insieme a query di dati efficienti.

In particolare, gli utenti vengono registrati e loggati nell'App tramite il loro account Google e i loro dati vengono salvati/scaricati all'interno del database.

Nel capitolo successivo verrà spiegato in dettaglio l'utilizzo di questa piattaforma.

3.4 Google Maps Platform

L'API di Google Maps, ora chiamata Google Maps Platform, ospita circa 17 API diverse, suddivise in temi nelle seguenti categorie; Mappe, luoghi e percorsi.

All'interno dell'App sono state utilizzate le seguenti:

¹Movimento che promuove sistemi software dove la persistenza dei dati è in generale caratterizzata dal fatto di non utilizzare il modello relazionale.

- **Maps SDK for Android:** permette la visualizzazione e la modifica di mappe all'interno di un'Activity: aggiungere alle mappe indicatori, linee, colori, immagini e poligoni personalizzati. Offre agli utenti la possibilità di creare e condividere mappe personalizzate, oltre che di utilizzare lo zoom, pizzicare, ruotare e inclinare le mappe per esplorarle più in dettaglio.
- **Places API:** è un servizio che restituisce informazioni sui luoghi utilizzando richieste HTTP. I luoghi sono definiti all'interno di questa API come stabilimenti, località geografiche o punti di interesse importanti.

Nel capitolo successivo verrà spiegato in dettaglio l'utilizzo di queste API per lo sviluppo dell'App.

Capitolo 4

Implementazione

Nel seguente capitolo vengono spiegati in dettaglio l'implementazione del database, dell'App e del programma desktop di gestione del database.

4.1 Cloud Firestore

Come database online è stato scelto Cloud Firestore della suite Google Firebase.

È stato creato un account google apposito che funge da amministratore del database; è utilizzato per accedere a tutti i dati dell'applicazione e gestirli: account google degli utenti, campi di ognuno di essi, attributi del campo.

Il seguente screen mostra com'è strutturato.

The screenshot shows the Cloud Firestore interface. At the top, there's a navigation bar with icons for home, collections, and documents, followed by the path: fedebnacca@... > carrara. Below this, there's a list of documents in the 'carrara' collection. One document is highlighted: 'fedebernacca@gmail.com'. This document has three sub-fields: 'address' (value: "54033 Carrara MS, Italia"), 'fields' (value: [{"points": [{"analyze": false}]}]), and 'name' (value: "carrara"). There are also buttons for 'Aggiungi documento' (Add document) and 'Avvia raccolta' (Start collection).

Ogni raccolta ha come id l'account google dell'utente, per ogni suo campo viene creato un documento con identificativo il nome del campo, all'interno del documento ci sono i suoi attributi.

Per aggiungere questo database all'app e al programma desktop sono state generate due API key che ne permettono l'utilizzo.

Inoltre il programma desktop permette una gestione semplice del database tramite una GUI costruita con Java Swing.

Table		
USERS	FIELDS	POINTS TO ANALYZE
alessiobibbiani97@gmail.com	casa	4
concetta.federico@gmail.com	orto botanico	23
fedebernacca@gmail.com		

Figura 4.1: GUI costruita con Java Swing per il programma desktop.

4.2 App

Nella seguente sezione viene spiegata in dettaglio l'implementazione dell'App, scrivendo un paragrafo per ogni Activity/Classe Java importante, riportando frammenti di codice e allegando screenshot tratti direttamente dall'App per rendere i concetti più chiari.

Innanzitutto, riportiamo uno schema di come il progetto è organizzato su Android Studio:

- **com.example.halofarms**
 - **activity**
 - * **InsertDataActivity**
 - * **MainActivity**
 - * **MapsActivity**

Gli **Adapter** sono oggetti che permettono una rappresentazione efficiente di liste nella UI:

- **adapter**
 - * **AnalyzedFieldAdapter**
 - * **DateAdapter**
 - * **MainActivityAdapter**
 - * **StringAdapter**

Classi di utilità:

- **Field**
- **FieldWithDate**
- **MyDialogFragment**
- **Point**

A seguire un grafo che schematizza un tipico flusso di utilizzo dell'App.

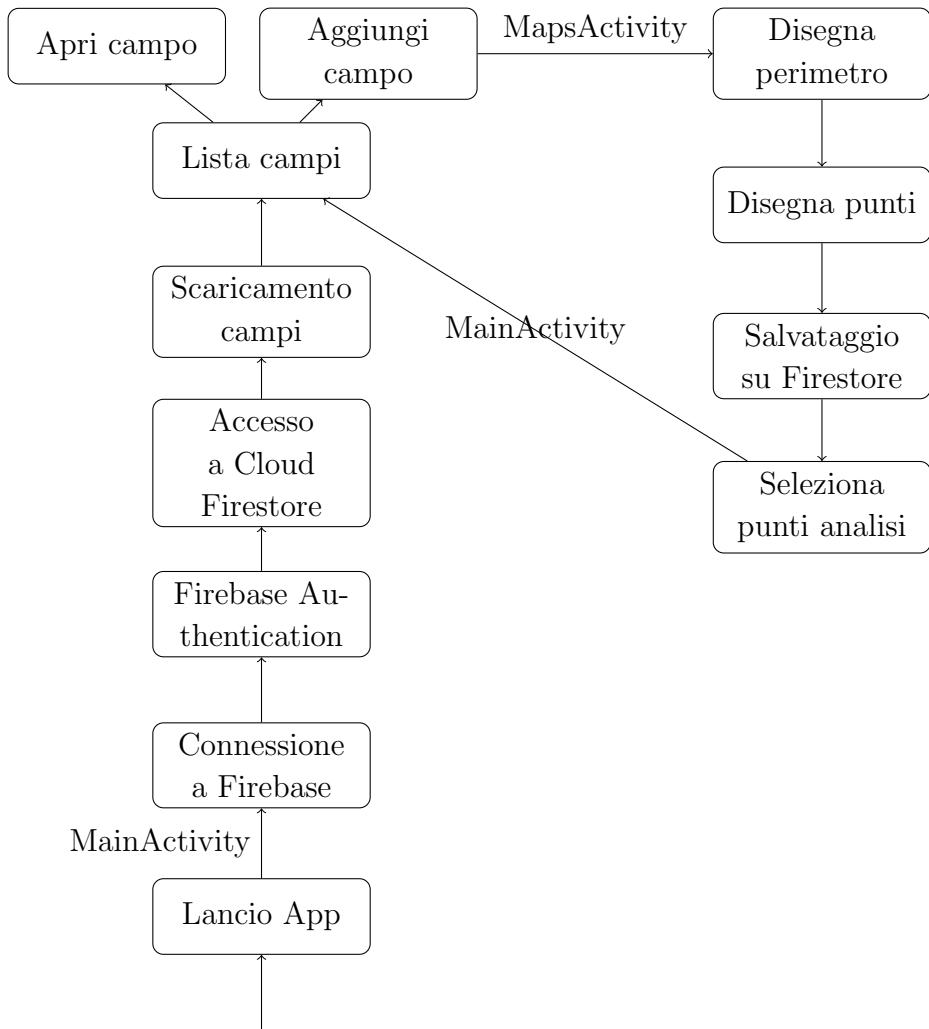


Figura 4.2: Diagramma di flusso dell'App.

4.2.1 Point.java

La seguente classe rappresenta un punto di campionamento sul campo con tutti i suoi attributi.

```
public class Point {
    // global unique id for point in json format
    private String jsonPoint;
    // flag that indicates if the point is to analyze
    private boolean analyze;
    // attributes of point
    private float ec, sar, ph, cec;
    // local id of point: 0, 1, 2, 3..
    private int zoneId;
    // corners of square containing point
    private String square;
    // coordinates of point (center of above square)
    private String suggestedPoint;
    // for every attribute there is an heat map with 4 colors
    private ArrayList<Integer> ecColors, sarColors, phColors, cecColors;
    public Point() {}
    public Point(String square, int zoneId, String suggestedPoint) {
        this.analyze = false;
        this.square = square;
        this.zoneId = zoneId;
        this.suggestedPoint = suggestedPoint;
        this.ecColors = new ArrayList<>();
        this.sarColors = new ArrayList<>();
        this.phColors = new ArrayList<>();
        this.cecColors = new ArrayList<>();
    }
    ...
}
```

Di seguito sono spiegate le variabili definite:

- **jsonPoint**: è in formato JSON, identifica univocamente un punto all'interno dell'intera App. Si costruisce concatenando nome del campo in cui si trova, id locale del punto **zoneId** e la sua data di campionamento:

```
p.setJsonPoint(new Gson().toJson(field.getName()
+ " " + p.getZoneId() + " " + field.getDate()));
```

- **analyze**: quando settato a true sta ad indicare che il punto del campo ha bisogno di essere analizzato; il risultato dell'analisi può essere inserito direttamente dall'App o attraverso il programma desktop;
- **ec, sar, ph, cec** sono i parametri di campionamento del punto;
- **zoneId** è l'identificativo univoco locale del punto all'interno del campo;
- **square**: i punti di campionamento si trovano al centro di un quadrato, questa variabile è la concatenazione delle coordinate GPS dei quattro vertici di questo quadrato. Il discorso viene approfondito nella sezione 4.2.6.2;
- **suggestedPoint**: coordinate GPS del punto di campionamento;
- **ecColors, sarColors, phColors, cecColors**: liste contenenti 4 interi che rappresentano il colore delle heat map disegnate secondo i valori dei parametri di campionamento.



Figura 4.3: Ogni punto rosso rappresenta un'istanza della classe **Point.java**

4.2.2 Field.java

La seguente classe rappresenta un campo con tutti i suoi punti.

```
1  public class Field {
2      // name and address of field
3      private String name, address;
4      // date of analysis
5      private String date;
6      // list of points inside field
7      private List<Point> points;
8      public Field() {}
9      public Field(String name, String address, List<Point> points) {
10         this.name = name;
11         this.address = address;
12         this.points = points;
13         this.date = "Not yet analyzed";
14     }
15     ...
16 }
```

La Figura 4.3 mostra un campo con la rispettiva lista **points**.

4.2.3 FieldWithDate.java

La seguente classe rappresenta differenti analisi eseguite in date diverse dello stesso campo.

```
1  public class FieldWithDate {
2      // name and address of field
3      private String name, address;
4      // list containing different analysis of the same field
5      private List<Field> fields = new ArrayList<>();
6      public FieldWithDate() {}
7      public FieldWithDate(String name, String address,
8              List<Field> fields) {
9          this.name = name;
10         this.address = address;
11         this.fields = fields;
12     }
13     ...
14 }
```

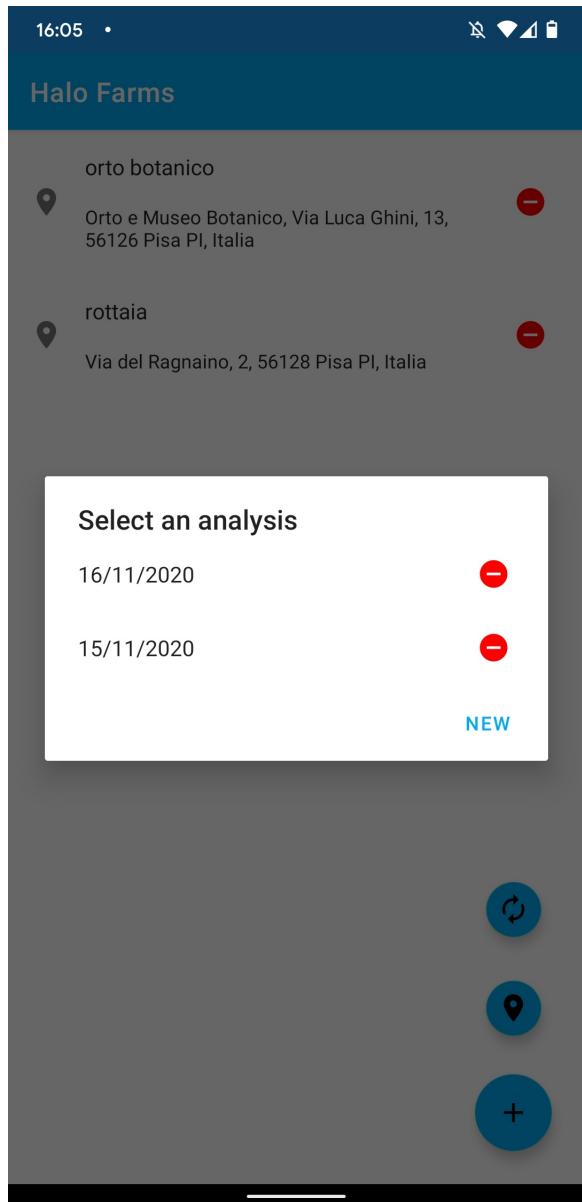


Figura 4.4: Un'istanza della classe **FieldWithDate.java**: le due date rappresentano due analisi differenti del medesimo campo.

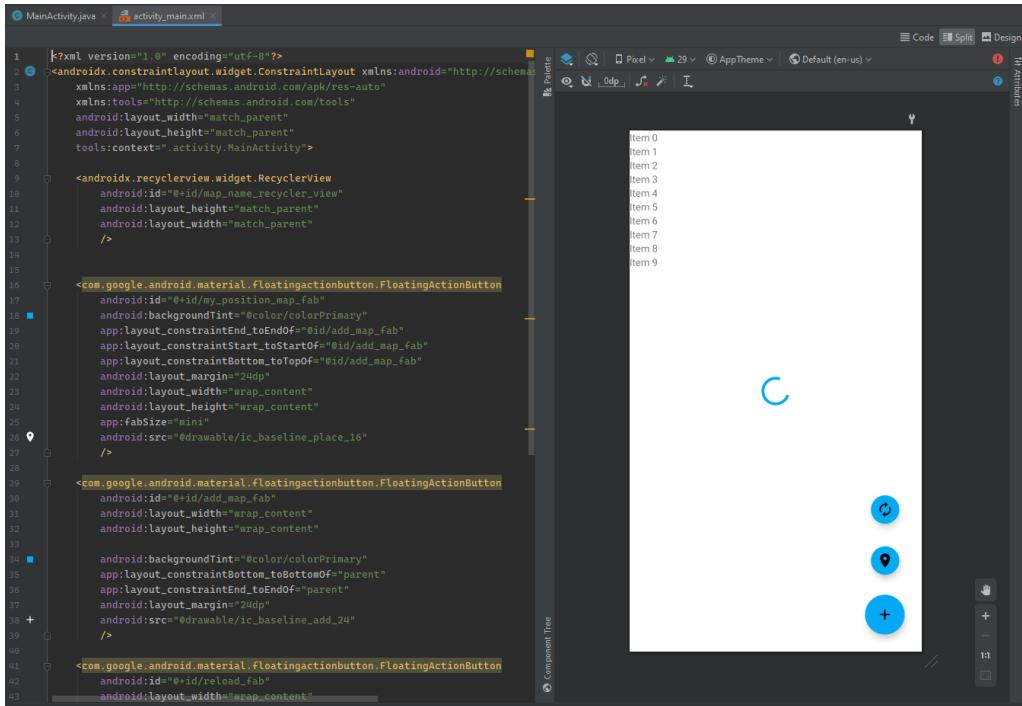
4.2.4 MainActivity.java

All'apertura dell'applicazione la prima Activity che viene lanciata è **MainActivity.java**; il primo metodo che viene eseguito è il seguente:

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      // set layout containing UI items
5      setContentView(R.layout.activity_main);
6      // references to UI items
7      addMapFab = findViewById(R.id.add_map_fab);
8      progressBar = findViewById(R.id.progress_bar);
9      reloadFab = findViewById(R.id.reload_fab);
10     myPositionMapFab = findViewById(R.id.my_position_map_fab);
11     RecyclerView recyclerView =
12         findViewById(R.id.map_name_recycler_view);
13     // build the recyclerView (list)
14     recyclerView.setHasFixedSize(true);
15     RecyclerView.LayoutManager layoutManager =
16         new LinearLayoutManager(this);
17     recyclerView.setLayoutManager(layoutManager);
18     mAdapter = new MainActivityAdapter(fields);
19     recyclerView.setAdapter(mAdapter);
20     // set listeners
21     addMapFab.setOnClickListener(this);
22     myPositionMapFab.setOnClickListener(this);
23     reloadFab.setOnClickListener(this);
24     mAdapter.setOnItemClickListener(onItemClickListener);
25     // user's login
26     auth();
27 }
```

Listing 1: Primo metodo eseguito all'avvio dell'App.

Quando l'Activity viene creata, come prima cosa si costruisce la UI: le componenti grafiche definite nel file **activity_main.xml**



The screenshot shows the Android Studio interface with the XML code for `activity_main.xml` on the left and the corresponding UI preview on the right.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".activity.MainActivity">
8
9      <androidx.recyclerview.widget.RecyclerView
10         android:id="@+id/map_name_recycler_view"
11         android:layout_height="match_parent"
12         android:layout_width="match_parent"
13     />
14
15
16      <com.google.android.material.floatingactionbutton.FloatingActionButton
17          android:id="@+id/my_position_map_fab"
18          android:backgroundTint="#color/colorPrimary"
19          app:layout_constraintEnd_toEndOf="@+id/add_map_fab"
20          app:layout_constraintStart_toStartOf="@+id/add_map_fab"
21          app:layout_constraintBottom_toTopOf="@+id/add_map_fab"
22          android:layout_margin="24dp"
23          android:layout_width="wrap_content"
24          android:layout_height="wrap_content"
25          app:fabSize="mini"
26          android:src="@drawable/ic_baseline_place_16"
27      />
28
29
30      <com.google.android.material.floatingactionbutton.FloatingActionButton
31          android:id="@+id/add_map_fab"
32          android:layout_width="wrap_content"
33          android:layout_height="wrap_content"
34
35          android:backgroundTint="#color/colorPrimary"
36          app:layout_constraintBottom_toBottomOf="parent"
37          app:layout_constraintEnd_toEndOf="parent"
38          android:layout_margin="24dp"
39          android:src="@drawable/ic_baseline_add_24"
40      />
41
42      <com.google.android.material.floatingactionbutton.FloatingActionButton
43          android:id="@+id/reload_fab"
44          android:layout_width="wrap_content"

```

The UI preview shows a RecyclerView at the top containing 10 items labeled "Item 0" through "Item 9". Below the RecyclerView are three floating action buttons (FABs): a blue FAB with a location pin icon, a blue FAB with a circular arrow icon, and a blue FAB with a plus sign icon.

Figura 4.5: file **activity_main.xml** che definisce il layout di **MainActivitiy.java**.

vengono “collegate” al codice tramite il metodo **findViewById**, subito dopo gli vengono associati dei listeners: dei metodi che eseguono qualcosa al tap dell’utente su una di queste componenti. Infine viene chiamato il metodo che autentica l’utente col proprio username e lo connette al database (26); riportato nella pagina successiva.

4.2.4.1 Connessione a Cloud Firestore

```
1  private void auth() {
2      // Choose authentication providers
3      List<AuthUI.IdpConfig> providers = Collections.singletonList(
4          new AuthUI.IdpConfig.GoogleBuilder().build());
5      // Create and launch sign-in intent
6      startActivityForResult(
7          AuthUI.getInstance()
8              .createSignInIntentBuilder()
9              .setAvailableProviders(providers)
10             .build(),0);
11 }
```

Listing 2: Metodo che permette l'autenticazione dell'utente.

Come authentication provider è stato scelto di utilizzare il proprio account google (3 - 4). Viene dunque creato e lanciato l'intent che permette il login (6 - 10); poiché questo si avvia chiamando il metodo **startActivityForResult()**, l'Activity chiamante si aspetta di ricevere un risultato al termine dell'operazione.

Per identificare l'operazione richiesta viene utilizzato un intero, in questo caso lo 0 nella riga 10.

Il risultato dell'operazione avviata viene restituito nel metodo **onActivityResult()**. A pagina seguente è listato e descritto questo metodo.

```
1  @Override
2  protected void onActivityResult(int requestCode, int resultCode,
3          Intent data) {
4      super.onActivityResult(requestCode, resultCode, data);
5      // get the result of google login
6      if (requestCode == 0) {
7          if (resultCode == RESULT_OK) {
8              // Successfully signed in
9              FirebaseAuth user = FirebaseAuth
10                  .getInstance()
11                  .getCurrentUser();
12              // reference to preferences,
13              // here will be stored the username in local
14              SharedPreferences sharedPref = getApplicationContext()
15                  .getSharedPreferences(
16                      getString(R.string.preference_file_key),
17                      Context.MODE_PRIVATE);
18              // save the email that will be used
19              // as unique key for accessing data in db
20              if (user != null) {
21                  // save username in app
22                  sharedPref.edit().putString("USERNAME",
23                      username = user.getEmail()).apply();
24                  // user is authenticated, get his maps from db
25                  readMapsFromFirestore();
26              }
27          } else {
28              // authentication fails, show an error message
29              Toast.makeText(this, "Something wrong",
30                  Toast.LENGTH_SHORT).show();
31          }
32      }
33      ...

```

Listing 3: Ricezione esito operazione di autenticazione.

Come spiegato sopra, grazie al controllo sul **requestCode** si è a conoscenza di qual è stata l'operazione richiesta in precedenza, e dunque si riconosce il ramo if corretto in cui entrare.

A riga 6 si controlla se l'operazione richiesta era quella di login. Se l'operazione è andata a buon fine si ottiene un'istanza dell'utente appena loggato. Si estrae il suo username e lo si salva in un file locale all'App al quale si ha accesso in tutte le Activity.

A questo punto si richiama il metodo a riga 25 che permette la connessione al database; in particolare si ha accesso alla entry che ha come id l'username dell'utente.

```
1  private void readMapsFromFirestore() { // reference to user's entry
2      FirebaseFirestore.getInstance().collection(username).get()
3          .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
4              @Override
5              public void onComplete(@NonNull Task<QuerySnapshot> task) {
6                  if (task.isSuccessful()) {
7                      // iterate over user's documents
8                      for (QueryDocumentSnapshot doc : task.getResult()) {
9                          if (doc.exists()) {
10                              // get all fields with the date of analysis
11                              // of this document treated as a field
12                              FieldWithDate fieldWithDate = doc
13                                  .toObject(FieldWithDate.class);
14                              // add field to the list shown in UI
15                              fields.add(fieldWithDate.getFields().get(0));
16                              // add all field to the list shown in dialog
17                              // when tap on an element of above list
18                              analyzedFields.add(fieldWithDate);
19                              // update the UI
20                              mAdapter.notifyDataSetChanged();
21
22                      }
23                  }
24              }
25          }
26      ...
27  }
```

Listing 4: Scaricamento campi dal database.

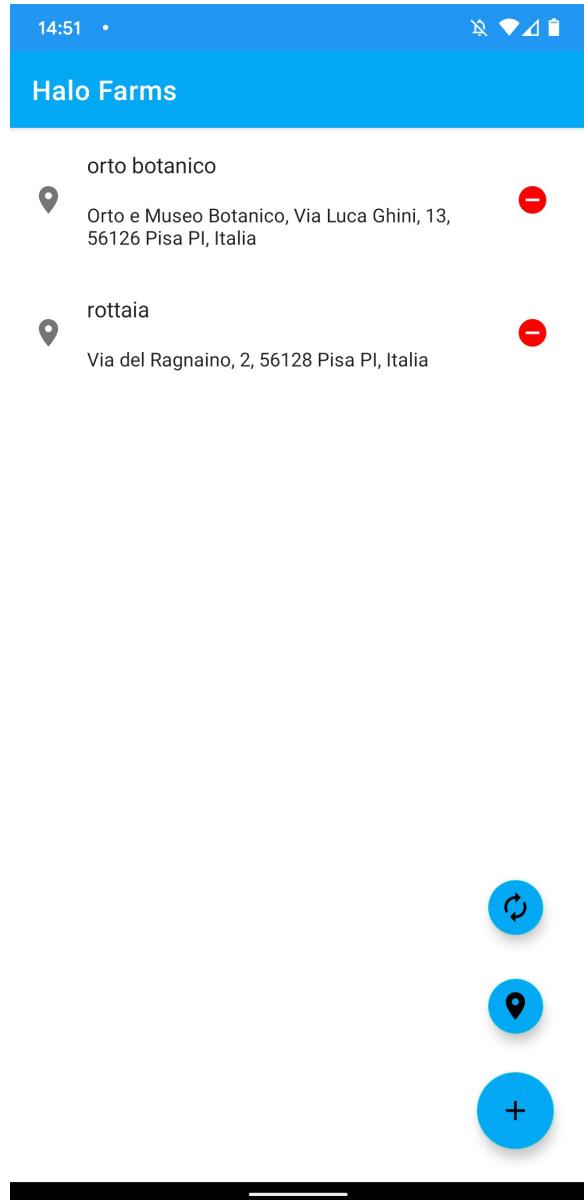
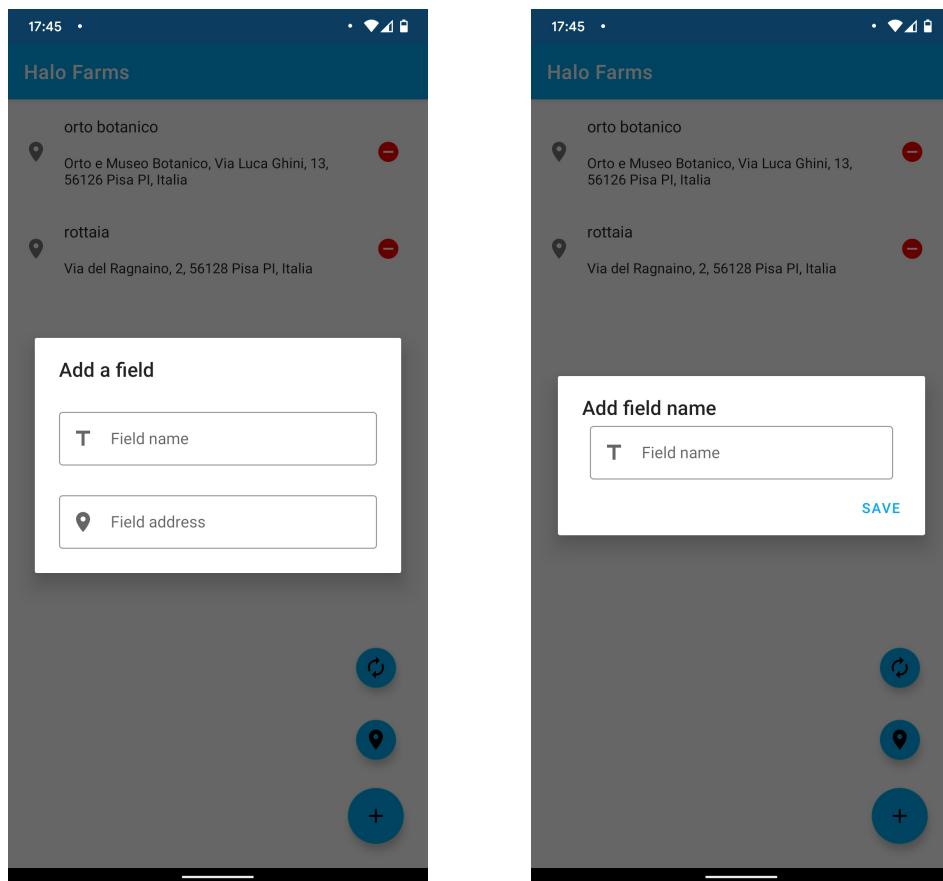


Figura 4.6: UI dopo aver eseguito correttamente il login ed aver scaricato i campi dal database.

4.2.5 MyDialogFragment.java

Nella UI mostrata dalla figura 4.6 si notano tre **FloatingActionButton** (bottoni circolari) con le seguenti icone: facendo un tap su uno di essi vengono eseguite diverse azioni.

Toccando viene nuovamente interrogato il database; toccando le altre due vengono aperti i seguenti Dialog.



(a) Tap su : viene salvato il nome del campo e viene aperta la mappa

(b) Tap su : viene salvato il nome del campo e viene aperta la mappa sull'indirizzo scelto sulla propria posizione

Figura 4.7: Una volta compilati i form si passa ad un'altra Activity: **MapsActivity.java**

Consideriamo 4.7b: il codice che genera quel Dialog fa parte della classe **MyDialogFragment.java** ed è il seguente:

```
1  @Override
2  public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
3      ...
4      case 1: {
5          ...
6          // build custom dialog
7          return new AlertDialog.Builder(
8              ...
9              .OnClickListener() {
10             @Override
11             public void onClick(DialogInterface dialog, int which) {
12                 // get string from edit text
13                 Field field = new Field(fieldNameEditText
14                     .getText().toString(), null, null);
15                 // build intent
16                 Intent intent = new Intent(getContext(),
17                     MapsActivity.class);
18                 // extra data
19                 intent.putExtra("IS_MAP_DRAWN", false);
20                 intent.putExtra("POSITION", true);
21                 intent.putExtra("FIELD", new Gson().toJson(field));
22                 // start MapsActivity and wait for result
23                 getActivity().startActivityForResult(intent,1);
24             }
25         }).create();
26     ...
27 }
```

Si crea un Intent che avvierà **MapsActivity.java**, si inseriscono poi alcuni messaggi dentro questo intent; il primo sta ad indicare che il campo non è ancora disegnato, il secondo indica di utilizzare la propria posizione, il terzo scrive il campo con una parte dei suoi attributi in formato JSON (17 - 22).

Infine si avvia l'Activity e si rimane in attesa di un risultato, specificato dall'intero 1 (24).

4.2.6 MapsActivity.java

Questa è l'Activity più importante dell'App, quella che mostra la mappa del campo coi rispettivi punti di campionamento eventualmente analizzati. Si avvia questa Activity in tre modi differenti: due di questi sono rappresentati in figura 4.7. Il terzo è facendo un tap su un campo presente nella lista visualizzata nella UI di **MainActivity**.

Continuiamo l'esempio della Figura 4.7b; una volta compilato il form nel Dialog viene avviata questa Activity; di seguito un frammento di codice della sua creazione.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_maps);
5      // Obtain the SupportMapFragment and
6      // get notified when the map is ready to be used.
7      SupportMapFragment mapFragment = (SupportMapFragment)
8          getSupportFragmentManager()
9          .findFragmentById(R.id.fragment_map_fragment);
10     if (mapFragment != null) {
11         mapFragment.getMapAsync(this);
12     }
13     // get intent from main activity
14     Intent intent = getIntent();
15     ...
16     // if the file is already drawn
17     isMapDrawn = intent.getBooleanExtra("IS_MAP_DRAWN", false);
18     // field (maybe only name and address)
19     field = gson.fromJson(intent.getStringExtra("FIELD"),
20                           Field.class);
21     // if position of user is needed
22     positionNeeded = intent.getBooleanExtra("POSITION", false);
23 }
```

Si richiede di costruire e visualizzare la mappa nella UI (7 - 12); questo task viene eseguito in maniera asincrona, da un altro thread diverso da quello

principale (UI thread). Una volta che la mappa è pronta viene mostrata nella UI.

Nelle righe successive si estraggono i messaggi inviati dall'Activity chiamante, questi vengono salvati e poi testati una volta che la mappa è pronta: se il booleano **positionNeeded** è settato a true, la UI è disegnata come segue.



Figura 4.8: UI quando la mappa è pronta e si è richiesta la propria posizione.

4.2.6.1 Perimetro del campo

Si passa adesso alla spiegazione di come viene disegnato il perimetro del campo.

L'idea iniziale è stata quella di disegnare il perimetro a mano libera: l'utente avrebbe dovuto toccare lo schermo e trascinare il dito per disegnarlo, ma questo avrebbe portato ad alcuni problemi:

- se la dimensione del campo fosse stata di molti ettari, la durata del trascinamento sullo schermo sarebbe durata troppo, rovinando la User Experience;
- il Disegno del perimetro sarebbe potuto risultare poco preciso.

Si è optato allora per il seguente approccio: l'utente esegue N tap sulla mappa, ad ogni tap viene disegnato un oggetto di tipo **Marker** (punto).

```
@Override
public void onMapClick(LatLng latLng) {
    // add the tapped point in a list
    // which coordinates will be used for drawing perimeter
    points.add(latLng);
    // create marker for drawing the perimeter
    MarkerOptions marker;
    // if perimeter isn't already drawn
    if (area == 0) {
        marker = new MarkerOptions().position(latLng)
            .icon(bitmapDescriptorFromVector(
                R.drawable.black_round_shape));
    }
    ...
    // add marker in map
    markers.add(googleMap.addMarker(marker));
}
```

Listing 5: Metodo che disegna un Marker (punto) ad ogni tap dell'utente

Una volta terminati i tap, i punti vengono collegati tra di loro con delle linee creando un poligono **Polygon**. Ogni punto rappresenta un vertice e questo poligono rappresenta il perimetro del campo.

La seguente istruzione mostra il disegno del perimetro: collegamento tra i punti tappati sulla mappa dall'utente.

```
perimeterPoly = googleMap  
    .addPolygon(new PolygonOptions()  
        .addAll(points));
```

Nella lista **points** sono contenuti i punti tappati precedentemente dall'utente; sono trattati come coordinate GPS, oggetti di tipo **LatLng** (coppie <latitudine, longitude>) ed utilizzati come vertici del poligono.



(a) Vertici del perimetro.



(b) Perimetro.

4.2.6.2 Punti del campo

Ogni punto del campo è un oggetto di tipo **Point** descritto nella sezione 4.2.1.

Per disegnare i punti all'interno del perimetro sono stati implementati tre diversi approcci a discrezione dell'utente:

1. **marcando la propria posizione;**
2. **a mano libera;**
3. **seguendo schemi di campionamento.**

1: L'utente può, dopo aver disegnato il perimetro del campo, spostarsi all'interno di esso e marcare le proprie posizioni come punti di campionamento.

2: L'approccio a mano libera segue indicativamente il modello per disegnare il perimetro; una volta disegnato, l'utente tocca punti al suo interno che verranno salvati come punti di campionamento.

3: Esistono alcuni schemi di campionamento consigliati che prevedono una griglia a quadrati, con campionamento al centro o in uno dei vertici, e schemi di campionamento a W o X.

È stato scelto lo schema che prevede una griglia a quadrati, con campionamento al centro di ognuno di essi.

L'idea implementativa è stata la seguente; dato un poligono di qualsiasi forma:

1. è stato calcolato il minimo rettangolo che lo contiene tutto;
2. è stata creata la griglia di quadrati per quel rettangolo;
3. per ogni quadrato è stato calcolato il centro;
4. ogni centro che sta dentro la figura è stato usato come punto di campionamento.

Di seguito una parte di codice dell'algoritmo che esegue questi calcoli.

```
1 private void drawGrid(List<LatLng> coordinates) {
2     // include all point of perimeter
3     // for drawing the minimum rectangle containing perimeter
4     LatLngBounds.Builder builder = LatLngBounds.builder();
5     for (LatLng latLng : coordinates) {
6         builder.include(latLng);
7     }
8     // extract the top-right vertex
9     LatLng northeast = builder.build().northeast;
10    // extract the bottom-left vertex
11    LatLng southwest = builder.build().southwest;
12    // calculate top-left vertex
13    LatLng northwest = new LatLng(northeast.latitude, southwest.longitude);
14    // calculate bottom-right vertex
15    LatLng southeast = new LatLng(southwest.latitude, northeast.longitude);
16    // minimum rectangle containing perimeterPoly
17    Polygon rectanglePoly = googleMap.addPolygon(new PolygonOptions()
18        .add(northwest)
19        .add(northeast)
20        .add(southeast)
21        .add(southwest));
```

Figura 4.10: Calcolo del minimo rettangolo contenente il perimetro.

Implementazione punto 1: nella lista **coordinates** sono presenti i vertici del perimetro; viene istanziato l'oggetto **builder** di tipo **LatLngBounds.Builder** (4), tra i suoi metodi ne esiste uno che, dati N punti di coordinate **LatLng** calcola il punto a nordest e a sudovest.

Vengono passate a **builder** gli elementi (coordinate) di **coordinates** (5 - 6); nelle successive righe vengono estratti i punti a nordest e sudovest (9 - 11) e calcolati i due vertici rimanenti del minimo rettangolo: nordovest e sudest (13 - 15). Infine viene creato il poligono che rappresenta questo rettangolo (17 - 21).

Altro frammento dell'algoritmo importante per la costruzione della griglia è il seguente:

```
1 ...
2 // start to build the internal squares:
3 // use an image of square for every cell and drawn on top of it a polygon
4 // first square-image: start from bottom-right corner
5 GroundOverlayOptions options = new GroundOverlayOptions()
6     .visible(false)
7     .image(bitmapDescriptorFromVector(R.drawable.square))
8     .anchor(0, 0)
9     .position(southeast, meters);
10 GroundOverlay prev, curr = googleMap.addGroundOverlay(options);
11 // move in horizontal
12 for (int i = 0; i < iterations; i++) {
13     prev = curr;
14     // move in vertical
15     for (int j = 0; j < iterations; j++) {
16         // build new square-image
17         curr = googleMap.addGroundOverlay(new GroundOverlayOptions()
18             .visible(false)
19             .image(bitmapDescriptorFromVector(R.drawable.square))
20             .anchor(1, 1)
21             .position(curr.getBounds().northeast, meters));
22         // extract the top-right vertex
23         northeast = curr.getBounds().northeast;
24         // extract the bottom left vertex
25         southwest = curr.getBounds().southwest;
26         // calculate top-left vertex
27         northwest = new LatLng(northeast.latitude, southwest.longitude);
28         // calculate bottom-right vertex
29         southeast = new LatLng(southwest.latitude, northeast.longitude);
30         // check if at least one vertex is inside minimum rectangle
31         List<LatLng> rectVertex = rectanglePoly.getPoints();
32         if (PolyUtil.containsLocation(northeast, rectVertex, false)
33             || PolyUtil.containsLocation(northwest, rectVertex, false)
34             || PolyUtil.containsLocation(southeast, rectVertex, false))
```

```

35     || PolyUtil.containsLocation(southwest, rectVertex, false)) {
36         // build the poly-square on top of square-image
37         Polygon polygon = googleMap.addPolygon(new PolygonOptions()
38             .visible(false)
39             .add(northwest)
40             .add(northeast)
41             .add(southeast)
42             .add(southwest));
43         // add polygon (square) to list that will be used
44         // for drawing suggested points
45         polygons.add(polygon);
46         // remove actual square-image
47         curr.remove();
48     }
49 }
50 // build the new square-image on top of the previous one
51 curr = googleMap.addGroundOverlay(new GroundOverlayOptions()
52     .visible(false)
53     .image(bitmapDescriptorFromVector(R.drawable.square))
54     .anchor(1, 1)
55     .position(prev.getBounds().southwest, meters));
56 // no more needed: remove
57 prev.remove();
58 }

```

Implementazione punto 2: per creare le regioni interne al minimo rettangolo, si è utilizzata la figura di un quadrato definita in formato XML.

```

<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <stroke
        android:width="0.1px"
        android:color="#000000" />
    <size
        android:height="32dp"
        android:width="32dp" />
</shape>

```

Questa è inserita ripetutamente all'interno del rettangolo finché non lo ha riempito tutto.

Il primo quadrato viene creato col suo vertice top-left in corrispondenza del vertice **southeast** del minimo rettangolo (5 - 8).

Inizia dunque il ciclo che itera finché non si è riempito tutto il rettangolo. Nel ciclo for più esterno (12) si sposta la figura in orizzontale; in quello interno (15) si sposta in verticale.

Si “incolla” un nuovo quadrato sopra quello precedente: il lato in basso del nuovo quadrato viene messo sopra quello in alto del precedente (spostamento verticale) (17 - 21); nelle righe successive si estraggono i quattro vertici del quadrato (23 - 29), utili per controllare se almeno uno di questi risiede all'interno del minimo rettangolo (32 - 35). In caso affermativo, viene creato **polygon** con queste coordinate ed inserito nella lista **polygons** che verrà, al termine dell'algoritmo, utilizzata per disegnare il punto di campionamento suggerito.

Usciti dal ciclo for interno, si crea un nuovo quadrato alla sinistra del primo disegnato nell'iterazione attuale (51 - 55). Si ripetono questi passaggi finché tutto il rettangolo non è riempito.

Inoltre, in base alla dimensione del campo, questi quadrati differiranno in numero e in dimensione: più è grande il terreno, più zone verranno create. La dimensione delle zone non è fissa, ma cambia anch'essa in base alla dimensione del campo, ad esempio un campo che ha un'area piccola avrà i vari punti di campionamento più vicini di quanto non abbia un campo con area grande.

Infine, il frammento di codice che implementa i punti 3 e 4.

```
1 ...
2 // iterate over all cells inside minimum rectangle
3 for (int i = 0; i < polygons.size(); i++) {
4     Polygon poly = polygons.get(i);
5     // save the square of suggested points (center of polygon)
6     List<LatLng> coordinates = poly.getPoints();
7     // extract center of poly
8     LatLng center = LatLngBounds.builder()
9         .include(coordinates.get(0)).include(coordinates.get(1))
10        .include(coordinates.get(2)).include(coordinates.get(3))
11        .build().getCenter();
12     // build point: square containing the point, zoneId,
13     // coordinates of point
14     Point p = new Point(fromPolygonToString(poly), zone++,
15             center.latitude + " " + center.longitude);
16     // set global unique id
17     p.setJsonPoint(new Gson()
18         .toJson(field.getName() + " " + p.getZoneId() + " "
19             + field.getDate()));
20     pointss.add(p);
21     // draw on map the point suggested if it is inside perimeter
22     if (PolyUtil.containsLocation(center, perimeterPoly.getPoints())) {
23         markers.add(googleMap.addMarker(new MarkerOptions()
24             .position(fromStringToLatLng(p.getSuggestedPoint()).get(0))
25             .title(p.getZoneId() + "").snippet(makeSnippet(p))
26             .icon(bitmapDescriptorFromVector(R.drawable.red_round_shape)));
27     }
28 }
```

Implementazione punti 3 e 4: una volta costruito il minimo rettangolo con al suo interno le regioni, si passa a disegnare i punti di campionamento.

Inizia il ciclo che itera sulle regioni (3); si calcola il centro di ognuna di esse (8 - 13); si crea il punto (14 - 19) e lo si disegna sulla mappa se è all'interno del campo (22 - 26).



Figura 4.11: Costruzione del campo: perimetro, **minimo rettangolo**, **regioni**, **punti**.

4.2.6.3 Analisi dei punti

Quando il perimetro e i punti sono stati disegnati, e quando il coltivatore ha effettuato un campionamento su uno o più punti, si può passare all'analisi (o richiesta di analisi) di questi.

L'utente esegue un tap sui punti campionati, e questi vengono marcati come da analizzare:

```
1  @Override
2  public boolean onMarkerClick(final Marker marker) {
3      // permits to show qr code
4      qrCodeButton.setVisibility(View.VISIBLE);
5      // find point to analyze
6      int id = Integer.parseInt(marker.getTitle());
7      for (Point p : field.getPoints()) {
8          if (p.getZoneId() == id) {
9              qrCodePoint = p;
10             // if field isn't yet analyzed
11             if (isEditable) {
12                 // for each tap change state
13                 p.setAnalyze(!p.isAnalyze());
14                 // red if point is marked as !to be analyzed
15                 if (!p.isAnalyze()) {
16                     toAnalyze.remove(p);
17                     marker.setIcon(fromVector(R.drawable.red_round));
18                 } else {
19                     // yellow if point is marked as to be analyzed
20                     toAnalyze.add(p);
21                     marker.setIcon(fromVector(R.drawable.yellow_round));
22                 }
23             ...
24     }
```

Quando viene eseguito un tap sulla mappa, in realtà non si clicca un oggetto **Point** ma uno **Marker**; questo metodo ricerca all'interno della lista contenente i punti qual è quello associato al Marker che lo rappresenta sulla mappa (7 - 8).

Se il campo non è ancora stato analizzato (11), si setta il punto cliccato come da analizzare/non più da analizzare (13) rispettivamente di colori giallo o rosso.



Figura 4.12: In giallo i punti da analizzare.

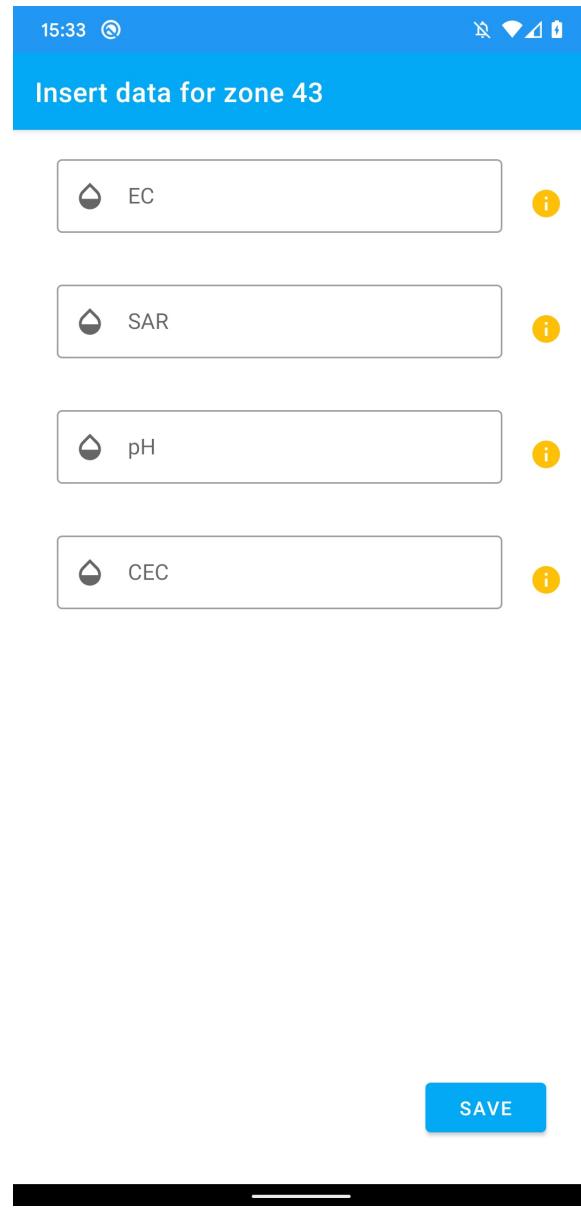
Una volta analizzati i campioni, ci sono due possibilità per inserire i risultati all'interno dell'App; uno è tramite il programma desktop che verrà spiegato in una successiva sezione, l'altro è direttamente dall'App.

Cliccando sulla finestra espansa dal tap su un punto, quella nella figura 4.12, viene avviata una nuova Activity: **InsertDataActivity** che permette l'inserimento dei risultati delle analisi; il codice che la avvia:

```
1  private void analyzePoints() {  
2      ...  
3      Intent intent = new Intent(this, InsertDataActivity.class);  
4      intent.putExtra("ID_ZONE", toAnalyze.get(0).getZoneId());  
5      intent.putExtra("FIELD", new Gson().toJson(field));  
6      toAnalyze.remove(0);  
7      startActivityForResult(intent, 3);  
8  }
```

Questo metodo viene richiamato finché non si sono inseriti tutti i risultati delle analisi; si porta alla nuova activity l'identificativo del punto da analizzare e il campo in cui è stato eseguito il campionamento (4 - 5); si rimane in attesa del risultato (7) che permetterà di disegnare diverse heat map a seconda dei valori.

Durante l'esecuzione di **InsertDataActivity** la UI è la seguente:



Una volta compilati i campi e premuto su salva, **MapsActivity** riprende la sua esecuzione ed è pronta per marcare come analizzati i precedenti punti gialli e a disegnare le heat map che si desiderano:

```

1  @Override
2  protected void onActivityResult(int requestCode, int resultCode,
3      @Nullable Intent data) {
4      super.onActivityResult(requestCode, resultCode, data);
5      // come back from InsertDataActivity
6      if (requestCode == 3 && resultCode == RESULT_OK) {
7          if (data != null) {
8              // get result
9              ArrayList<String> extras = data.getStringArrayListExtra("ID");
10             // update field with new analysis
11             field = new Gson().fromJson(data
12                 .getStringExtra("FIELD"), Field.class);
13             // update date of analysis
14             field.setDate(dateOfAnalysis);
15             if (extras != null) {
16                 // extract id of point from intent
17                 int id = Integer.parseInt(new StringTokenizer(extras
18                     .get(0)).nextToken());
19                 // write in the marker all results of analysis
20                 Marker marker = markers.get(id);
21                 marker.setSnippet(extras.get(1));
22                 marker.setIcon(fromVector(R.drawable.green_shape));
23                 marker.showInfoWindow();
24                 // build heat map for the just analyzed point
25                 for (Point p : field.getPoints()) {
26                     if (p.getZoneId() == id) {
27                         buildHeatMap(p);
28                     }
29                 }
30             ...

```

Questo frammento di codice aggiorna il campo **field** con i valori dei risultati delle analisi (11 - 14).

Aggiorna poi la UI: cambia l'informazione visualizzata dai vari **marker** scrivendo i valori dei risultati, inoltre colora i punti analizzati di verde (20 - 23). Infine costruisce la heat map per i punti analizzati (25 - 27).

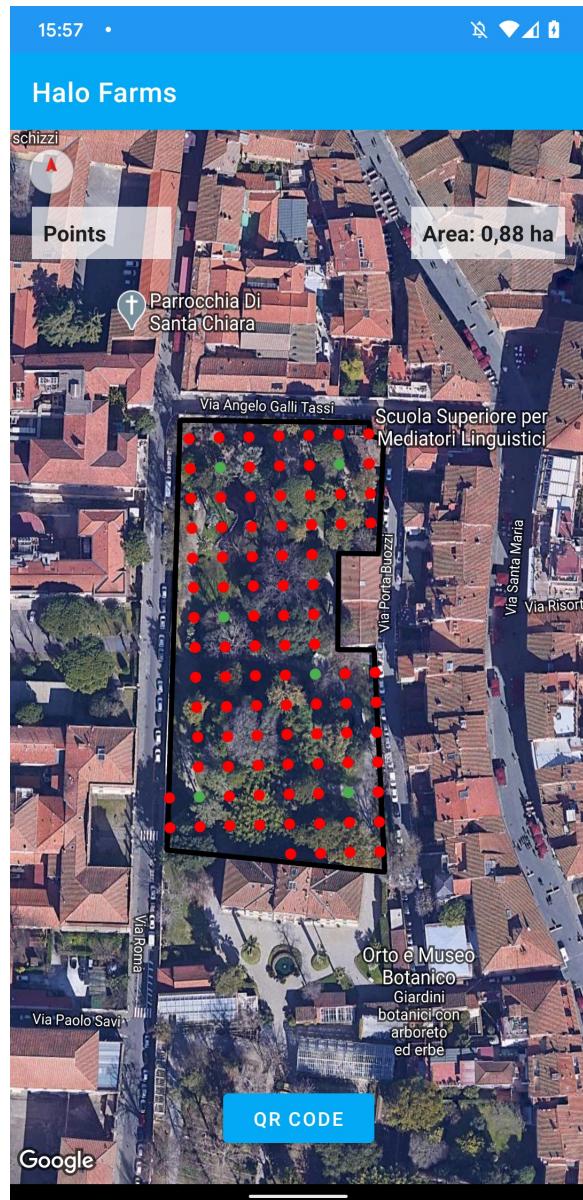


Figura 4.13: In verde i punti analizzati.

4.2.6.4 Heat map

Un altro punto cruciale del tirocinio, era quello di costruire una heat map del terreno, per far capire all'agricoltore in quale punto era consigliata una coltivazione piuttosto che un'altra.

Dunque sono state implementate 4 diverse heat map; una per ogni attributo da analizzare del punto (ec, cec, pH, SAR).

Per iniziare, una volta inseriti i valori dei quattro attributi, si associa una lista di colori al punto per ognuno di essi.

```
1  private void colorPoints(Point p) {
2      ArrayList<Integer> colorsEc = new ArrayList<>();
3      ...
4      if (p.getEc() <= 2) {
5          colorsEc.add(ContextCompat.getColor(getApplicationContext(),
6              R.color.light_green_A100));
7          colorsEc.add(ContextCompat.getColor(getApplicationContext(),
8              R.color.light_green_A200));
9          colorsEc.add(ContextCompat.getColor(getApplicationContext(),
10             R.color.light_green_A400));
11         colorsEc.add(ContextCompat.getColor(getApplicationContext(),
12             R.color.light_green_A700));
13     } else if (p.getEc() <= 4) {
14         ...
15     }
16     // update list of color of p
17     p.setEcColors(colorsEc);
18     ...
```

Figura 4.14: Associazione di un colore per l'attributo **ec** del punto **p**

I colori associati al punto dipendono dai valori degli attributi e possono variare in numero.

Ad esempio, il valore della conducibilità elettrica (EC) può far rientrare il terreno in 5 categorie diverse:

1. Non salino (0 - 2);

2. Leggermente salino (2 - 4);
3. Abbastanza salino (4 - 8);
4. Tanto salino (8 - 16);
5. Tantissimo salino (> 16);

Quindi, la lista **colorsEc** del punto **p** può assumere 5 sfumature diverse di colori. Queste vanno dal verde al rosso: meno è salino il terreno, più tendono al verde, viceversa, più è salino più tendono al rosso.

Ragionamenti analoghi per gli altri 3 attributi.

A seguire si associano i colori dei punti ad un oggetto che costruisce la heat map per ogni valore da mostrare nella UI:

```

1  public void buildHeatMap(Point p) {
2      // starting point for each color,
3      // given as a percentage of the maximum intensity
4      float[] startPoints = { 0.1f, 0.3f, 0.6f, 1f };
5      // get ec colors of p
6      List<Integer> colors = p.getEcColors();
7      // convert to array
8      int[] ecColors = {
9          colors.get(0), colors.get(1), colors.get(2), colors.get(3)};
10     // create the tile Provider for building heat map
11     HeatmapTileProvider ecProvider = new HeatmapTileProvider.Builder()
12         // coordinates of p
13         .data(fromStringToLatLng(p.getSuggestedPoint()))
14         // color to use
15         .gradient(new Gradient(ecColors, startPoints))
16         .build();
17     ecProvider.setRadius(100);
18     ...
19     // show on map
20     EcTileOverlays.add(googleMap.addTileOverlay(new TileOverlayOptions()
21             .tileProvider(ecProvider)
22             // heat map is visible if spinner is set to Heat Map
23             .visible(toShow.equals("ec heat map"))));

```

Si continua a considerare come esempio il parametro EC.

In **startPoints** sono inseriti dei float che indicano il punto di partenza per ognuna delle 4 sfumature di colore (4); si prende il riferimento alla lista **ecColors** del punto **p** e la si converte in array perché questo serve come parametro per costruire le sfumature (6 - 9).

Si crea ora la heat map passando come parametri le coordinate del punto e i colori da utilizzare, si setta il raggio e poi si disegna sulla mappa (11 - 23).

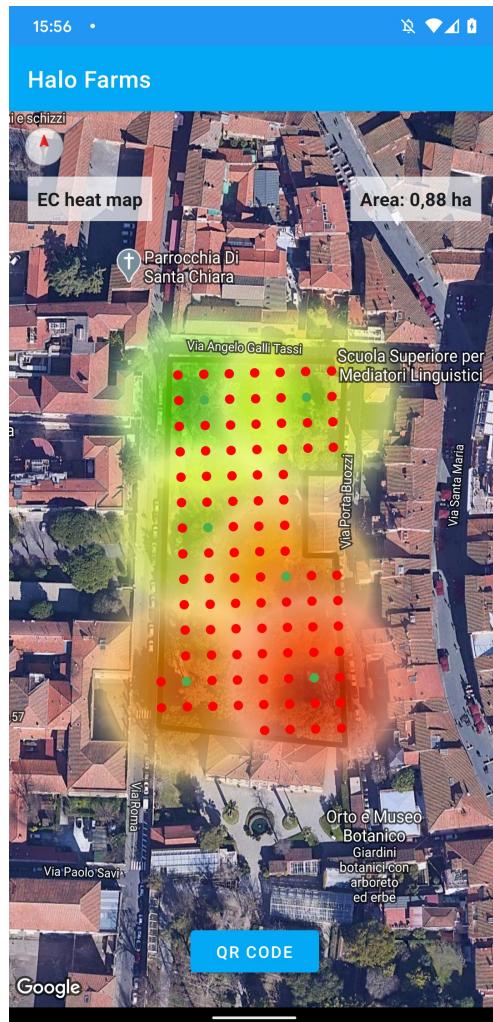


Figura 4.15: Heat map per EC.

4.2.6.5 QR code

Come scritto nella sezione 4.2.1, ogni punto **Point** ha un identificativo in formato JSON univoco globale all'interno dell'app; questo è usato per generare un QR code che in futuri sviluppi dell'app potrà, ad esempio, essere stampato con una stampante di etichette bluetooth e usato nell'applicazione desktop per l'inserimento dei dati di laboratorio.

L'id viene codificato in una matrice di bit che è utilizzata per costruire il qr code.



Figura 4.16: QR code di un punto.

4.2.6.6 Salvataggio su Cloud Firestore

Il salvataggio sul database online avviene ad ogni modifica significativa del campo:

- disegno del campo coi rispettivi punti di campionamento;
- marcamento di un punto da analizzare;
- caricamento dei risultati delle analisi.

```
1  private void saveToFirestore() {
2      // update name and address of object containing
3      // ALL analysis of this field
4      fieldWithDate.setName(field.getName());
5      fieldWithDate.setAddress(field.getAddress());
6      ...
7      // add the updated field to the list containing other analysis
8      fieldWithDate.getFields().add(field);
9      // save to firestore
10     FirebaseFirestore.getInstance().collection(username)
11         .document(field.getName())
12         .set(fieldWithDate);
13 }
```

Viene aggiornato il nome e l'indirizzo dell'oggetto che mantiene al suo interno la lista con tutte le analisi di questo campo (4 - 5). Viene dunque aggiunta l'istanza attuale del campo a questa lista (8). Infine viene salvato l'oggetto sul database nella entry dell'utente (12):

collection: **username** → document: **field.getName()** → attributes: **fieldWithDate**.

4.2.7 Programma desktop

Per gestire il database, o, più precisamente per inserire i risultati dei campionamenti del terreno, è stato realizzato un programma gestionale in Java che presenta una UI costruita con Java Swing.

L'idea è la seguente; quando l'agricoltore ha effettuato un campionamento e speditolo in laboratorio, marca sull'App i punti campionati, questi verranno salvati sul database e quindi visibili nel programma desktop, poiché all'avvio, questo, si connette al database e scarica i dati.

L'analista del laboratorio, una volta analizzati i punti, scrive i risultati nel programma. Una volta inseriti saranno dunque resi disponibili nell'applicazione.

L'applicazione desktop è prototipale; è stata implementata a scopo di testing e dimostrativo. L'idea è quella che in una prossima versione più avanzata, si possano utilizzare i QR code dei punti per semplificare l'inserimento delle analisi.

Viene riportato il codice che permette l'inserzione dei risultati delle analisi di uno o più punti.

```
1 pointsToAnalyzeJList.addMouseListener(new MouseAdapter() {
2     @Override
3     public void mouseClicked(MouseEvent e) {
4         // if list is empty, ignore mouse click
5         if (points.size() == 0) return;
6         // reference to element just clicked
7         int i = pointsToAnalyzeJList.getSelectedIndex();
8         // get the point at position
9         Point p = points.get(i);
10        // create the box containing date
11        JTextField date = new JTextField();
12        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
13        String d = sdf.format(new Date());
14        // set the date to the field in analyzing
15        field.setDate(d);
16        date.setText(d);
17        // build boxes for insert results of analysis
```

```

18     JTextField ec = new JTextField(), cec = new JTextField();
19     JTextField sar = new JTextField(), ph = new JTextField();
20     final JComponent[] inputs = new JComponent[] {
21         new JLabel("Date"), date, new JLabel("Ec"), ec,
22         new JLabel("Cec"), cec, new JLabel("Sar"), sar,
23         new JLabel("pH"), ph };
24     // show the dialog to fill
25     int result = JOptionPane.showConfirmDialog(null, inputs,
26         "Analysis of point " + p.getZoneId(), JOptionPane.DEFAULT_OPTION);
27     if (result == JOptionPane.OK_OPTION) {
28         // update point
29         p.setEc(Float.parseFloat(ec.getText()));
30         p.setCec(Float.parseFloat(cec.getText()));
31         p.setSar(Float.parseFloat(sar.getText()));
32         p.setPh(Float.parseFloat(ph.getText()));
33         // save to database
34         document.set(fieldWithDate);
35         // remove from lists the analyzed points
36         pointsToAnalyzeModel.remove(i);
37         points.remove(i);
38     ...

```

Questo metodo è un **MouseListener**, ovvero un metodo che risponde ad una azione eseguita dal mouse, in questo particolare caso ad un click su un oggetto visibile nella UI.

Se la lista cliccata è vuota, il click viene ignorato (5), altrimenti si considera il punto cliccato (7 - 9): si crea un **JTextField**: un oggetto al cui interno è atteso un input da tastiera dell'utente, in questo caso aspetta la data dell'analisi (11 - 13). Si setta poi la data appena immessa come data di campionamento del campo (15).

Si costruisce un dialog nel quale si immettono altri **JTextField**, uno per ogni parametro dell'analisi (18 - 28). Una volta compilati, si aggiorna il punto analizzato (31 - 36).

Infine si aggiorna il database e si rimuovono dalle liste i punti appena analizzati (37 - 39).

Illustrato un tipico ciclo di utilizzo App - programma Desktop.



(a) Pt gialli: da analizzare.

Analysis of point 18

Date	19/11/2020
Ec	1
Cec	2
Sar	3
pH	4

OK

(b) Inserimento risultati da PC.



(c) Pt verdi: analizzati.

Capitolo 5

Test

L'App e il programma desktop sono stati sviluppati e testati su due differenti macchine, con le seguenti componenti:

- **Pc desktop:**

- **CPU:** Intel core i7 7700K @4.5GHz;
- **RAM:** 16gb DDR4 @3000MHz;
- **GPU:** NVIDIA GeForce GTX 1070, 8gb;

- **Macbook Pro 2020**

- **CPU:** Intel Core i5 8257u @1.4Ghz - @3.9Ghz
- **RAM:** 16GB LPDDR3 @2133MHz

L'App è stata testata utilizzando l'emulatore integrato in Android Studio ed anche con diversi smartphone con diverse versioni di SO e di architettura, in particolare:

- **Xiaomi Mi 8:**

- **Android OS:** Android 10, Q;
- **CPU:** Qualcomm Snapdragon 845;

- **Samsung Galaxy A10:**

- **Android OS:** Android 10, Q;
- **CPU:** 2x 1.6 GHz Cortex-A73 + 6x 1.35 GHz Cortex-A53;

- **Samsung Galaxy s10e:**

- **Android OS:** Android 10, Q;
- **CPU:** Samsung Exynos 9 Octa (9820);

- **Samsung Galaxy s8:**

- **Android OS:** Android 9, Pie;
- **CPU:** Samsung Exynos 8 Octa (8895);

Ognuno di questi dispositivi presenta account google diversi; una volta effettuato il login correttamente, si è aggiunto e disegnato un numero arbitrario di campi e si è effettuato un numero arbitrario di analisi del terreno senza riscontrare bug o crash.

Inoltre, è stato testato l'accesso concorrente con lo stesso account google su più dispositivi; ad esempio, l'app è stata aperta su due dispositivi nello stesso momento, in uno di questi si è creato un campo, marcati i punti da campionare e si è tornati alla home. Nell'altro dispositivo si è ricevuto il nuovo campo, lo si è aperto, si sono inseriti i risultati delle analisi dei punti richiesti di campionare e si è chiusa l'app; il primo dispositivo ha ricevuto correttamente l'inserimento dei risultati. La sincronizzazione sembra funzionare, nessun problema è stato riscontrato.

Capitolo 6

Conclusioni

La partecipazione a questo tirocinio è stata un'esperienza molto istruttiva e formativa in quanto mi ha permesso di approfondire più in generale la conoscenza del linguaggio Java, fino ad arrivare più nello specifico alla programmazione applicata allo sviluppo di App, ambito che mi ha sempre appassionato.

Le nozioni assimilate durante i corsi di Programmazione I, Programmazione II, Sistemi Operativi, Reti Di Calcolatori e Sviluppo di Applicazioni Mobili hanno allargato e sistematizzato le mie conoscenze in materia, consolidando quella competenza che mi ha consentito di giungere a quest'ultima fase del mio percorso, il tirocinio, soddisfatto del mio bagaglio culturale. Questo bagaglio, accumulato nel percorso formativo, e la competenza acquisita mi hanno permesso di approcciare lo sviluppo dell'App con la giusta mentalità; i problemi riscontrati durante l'implementazione sono stati risolti grazie alla dimestichezza algoritmica acquisita.

Ho cercato di rendere il codice il più chiaro possibile commentandolo e documentandolo, sperando che un eventuale futuro programmatore non abbia particolari problemi nel leggerlo, comprenderlo e modificarlo.

Inoltre, grazie al corso di Programmazione Di Interfacce, ho avuto le giuste competenze per costruire una UI semplice ed intuitiva, fruibile a tutti, che offre una buona User Experience dato che gli utenti che l'hanno testata hanno comunicato, per il momento, feedback positivi.

L'App attualmente non è ancora in uso, ma dovrebbe essere pronta per

entrare in beta testing.

Il progetto nel suo complesso ha avuto un esito positivo sia per quanto riguarda la parte teorica necessaria per iniziare lo sviluppo dell'app -Android OS, Android Studio, componenti di un'app-, sia per quanto concerne la progettazione e l'implementazione dell'applicazione. Con buona approssimazione è stato rispettato il termine messo a disposizione per il conseguimento degli obiettivi posti a capo di questo progetto; tutte le funzionalità richieste sono state implementate correttamente.

Bibliografia

- [1] Android studio: an ide built for android. <https://web.archive.org/web/20151104132727/http://android-developers.blogspot.in/2013/05/android-studio-ide-built-for-android.html>.
- [2] Cos'è linux. <https://www.linux.it/linux>.
- [3] Cos'è un ambiente di sviluppo integrato (ide)? <https://www.redhat.com/it/topics/middleware/what-is-ide>.
- [4] Domande ricorrenti su gnu/linux. <https://www.gnu.org/gnu/gnu-linux-faq.html#linuxsyswithoutgnu>.
- [5] Fragments. https://developer.android.com/guide/components/fragments?gclid=Cj0KCQiAwMP9BRCzARIIsAPWTJ_HCcKI1_1P80gAjNiFQbkKApehUX-piuZN087WVG0plh5CrWl-MQ8oaAsMgEALw_wcB&gclsrc=aw.ds.
- [6] Introduction to activities. <https://developer.android.com/guide/components/activities/intro-activities>.
- [7] The java language specification. <http://java.sun.com/docs/books/jls/>.
- [8] Operating system market share. <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>.
- [9] Programming language popularity. <https://www.webcitation.org/67yahbjPg?url=http://www.langpop.com/>.

- [10] Touch devices - android open source. <https://web.archive.org/web/20120906025617/http://source.android.com/tech/input/touch-devices.html>.
- [11] Understand the activity lifecycle. <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [12] Update. <https://web.archive.org/web/20121105184651/http://lifehacker.com/5862994/real-world-test-show-that-android-task-killers-are-still-useless>.
- [13] Android psa: Stop using task killer apps. <https://web.archive.org/web/20130217024640/http://phandroid.com/2011/06/16/android-psa-stop-using-task-killer-apps-now/>, 2011.
- [14] The truth about android task killers and why you don't need them. <https://web.archive.org/web/20121023073622/http://www.phonedog.com/2011/06/26/the-truth-about-android-task-killers-and-why-you-don-t-need-them/>, 2011.
- [15] Tim Bray. Che cosa è android. <https://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is>, 2010.
- [16] Ben Elgin. Google buys android for its mobile arsenal. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm, 2005.
- [17] Vincenzo Gervasi. Architettura di un sistema android. <http://circe.di.unipi.it/~gervasi/SAM19/Lezione%2002-03.pdf>.
- [18] Vincenzo Gervasi. Componenti di un'applicazione. <http://circe.di.unipi.it/~gervasi/SAM19/Lezione%2004-05.pdf>.
- [19] Chris Hoffman. Android is based on linux, but what does that mean? <https://www.howtogeek.com/189036/android-is-based-on-linux-but-what-does-that-mean/>.

- [20] Victor Matos. Lesson 3: Android application's life cycle. <https://web.archive.org/web/20140222153131/http://grail.cba.csuohio.edu/~matos/notes/cis-493/lecture-notes/Android-Chapter03-Life-Cycle.pdf>, 2013.
- [21] Reto Meier. Professional android 4 application development. <https://books.google.com/books?id=g3hAdK1IBkYC&pg=PT53/>, 2012.
- [22] Richard Stallman. Is android really free software? <https://www.theguardian.com/technology/2011/sep/19/android-free-software-stallman>.