POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Cross platform mobile framework: Analisi dello stato dell'arte e utilizzo di Flutter nell'app Wher

Relatore:

Prof. Giovanni MALNATI

Candidato:

Ylenia PLACANICA

Marzo 2020

A Nonno Nuccio,
occhi di ghiaccio e cuore incandescente,
e Nonna Mariuccia,
amore e tenerezza infinite.

Sommario

Android vs iOS: la sfida eterna della tecnologia che ogni sviluppatore mobile ha affrontato almeno una volta nella vita. La presenza di più linguaggi di programmazione diversi rappresenta un ostacolo allo sviluppo rapido di applicazioni compatibili con entrambi i sistemi operativi, ma con il paradigma cross-platform lo sviluppo combinato per entrambe le piattaforme si semplifica. Questa tesi si concentra su una delle soluzioni più giovani ed interessanti: Flutter, framework open-source di casa Google. Attraverso il linguaggio Dart e l'utilizzo dei widget è possibile creare interfacce adattive per Android e iOS integrando perfettamente le funzioni native e hardware dei vari smartphones, ottenendo cosi delle applicazioni fluide ed ottimizzate in tempi molto brevi. L'uso e le funzionalità di Flutter vengono presentate all'interno del caso di studio di W-her, un'app che contiene mappe e navigatore utili alle donne, in quanto non suggerisce la strada più veloce ma quella più sicura, basata sulle valutazioni di una vasta community di donne in giro per l'Italia e l'Europa.

Indice

Elenco delle figure

Abbreviazioni

A-GPS

Assisted GPS

AoA

Angle of Arrival

AOT

Ahead Of Time

API

Application Programming Interfaces

APK

Android PacKage

ARM

Advanced RISC Machine

BLE

Bluetooth Low Energy

BSSID

Basic Service Set IDentifier

CPU

Central Processing Unit

GPS

Global Positioning Services

GPU

Graphic Processing Unit

GMS

Global System for Mobile communications

IDE

Integrated Development Environment

I/O

Input Output

IP

Internet Protocol

IT

Information Technology

JIT

Just In Time

MAC

Media Access Control

MSA

Mobile Station Assisted

MSB

Mobile Station Based

OEM

Original Equipment Manufacturer

OS

Operating System

PDF

Portable Document Format

RISC

Reduced Instruction Set Computer

RSSI

Received Signal Strength Indicator

SDK

Software Development Kit

ToF

Time of Flight

UI

User Interface

VM

Virtual Machine

WLAN

Wireless Local Area Network

Capitolo 1

Sviluppo di applicazioni mobile

Le applicazioni mobile sono software creati appositamente per dispositivi portatili, come smartphone e tablet. La presenza di più sistemi operativi per questi dispositivi ha portato alla necessità di creare diverse modalità di sviluppo per semplificare la creazione di applicazioni per le varie piattaforme.

1.1 Applicazioni native

Lo sviluppo nativo è quello che include la maggioranza delle applicazioni mobile esistenti ad oggi, ovvero quelle sviluppate per un sistema operativo specifico. Ogni sistema operativo prevede l'uso di un linguaggio di programmazione dedicato, come Objective-C e Swift per iOS e Java per Android, e fornisce librerie dedicate per utilizzare lo stile standard e tutti i componenti, hardware e non, del dispositivo (GPS, fotocamera, bluetooth, connessione dati etc.) con cui ci si interfaccia direttamente, come mostrato in figura 1.1.

Questo tipo di programmazione fornisce diversi vantaggi:

- vasta gamma di funzionalità, come l'accesso al file system, bluetooth e sensori, disponibili grazie alla perfetta integrazione con l'hardware del dispositivo.
- software veloce e reattivo, poiché il codice viene eseguito interamente all'interno del dispositivo, sfruttandone al massimo le potenzialità.

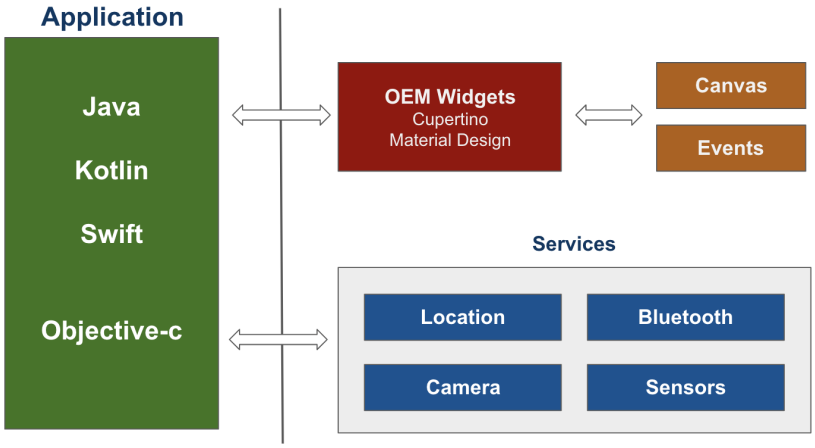


Figura 1.1: Architettura di applicazioni native

- interfaccia utente perfettamente coerente al sistema operativo, che fornisce una miglior esperienza utente, in quanto gli utenti tendono ad abituarsi più facilmente agli schemi utilizzati con minori tempi di apprendimento e migliori sensazioni [1].

- grande varietà di strumenti per il test ed il debug che rendono più veloce ed agevole lo sviluppo.

- maggiore diffusione in quanto è facile reperire l'applicazione tramite gli store ufficiali.

Di contro, sono presenti alcuni punti di debolezza:

- costi di manutenzione e gestione più alti in quanto le problematiche su dispositivi con sistemi operativi diversi possono non essere simili, dunque è richiesto più sforzo per individuarle e correggerle.

- tempo di sviluppo maggiore poiché serve utilizzare linguaggi di programmazione diversi per i vari sistemi operativi e non c'è possibilità di riutilizzare codice già scritto per una piattaforma diversa.

- limitate ai dispositivi per i quali sono sviluppate.

1.2 Applicazioni ibride

Le applicazioni ibride combinano elementi di applicazioni native ed applicazioni web. Possono essere distribuite sugli store come le applicazioni native ed allo stesso tempo possono essere utilizzate anche sui sistemi operativi desktop e sui browser. Si interfacciano con i componenti hardware e non del dispositivo tramite un *bridge* che permette l'accesso a questi servizi, come mostrato in figura 1.2.

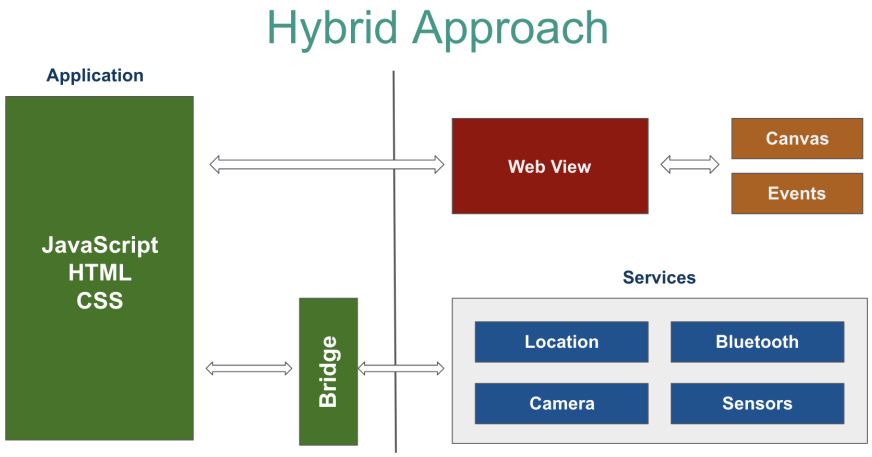


Figura 1.2: Architettura di applicazioni ibride

In questo caso i vantaggi sono:

- basso costo in quanto un'applicazione unica è adattabile a più piattaforme.
- grande scalabilità in quanto, essendo basate su una struttura HTML, è facile replicarle e renderle cross-platform.

Questo tipo di programmazione presenta anche alcune debolezze:

- performance non ottimali in quanto l'accesso alle funzionalità del dispositivo non è completo e diretto.
- interfaccia non nativa, poiché viene sviluppata in maniera unica per tutte le piattaforme e per il web.
- assenza di strumenti di test e debug specifici al di fuori del browser e della console.

1.3 Applicazioni cross platform

Con sviluppo cross platform si intende la creazione di codice sorgente in un unico linguaggio che viene gestito da un cross compiler [2], ovvero un compilatore che si occupa di tradurre il codice sorgente in file binari nei vari linguaggi nativi facendo quindi da ponte con l'hardware dei dispositivi, come mostrato figura 1.3. Questa metodologia permette di utilizzare le interfacce grafiche caratteristiche dei vari sistemi operativi e tutte le componenti hardware del dispositivo, ma necessita di un compilatore efficiente e molto affidabile.

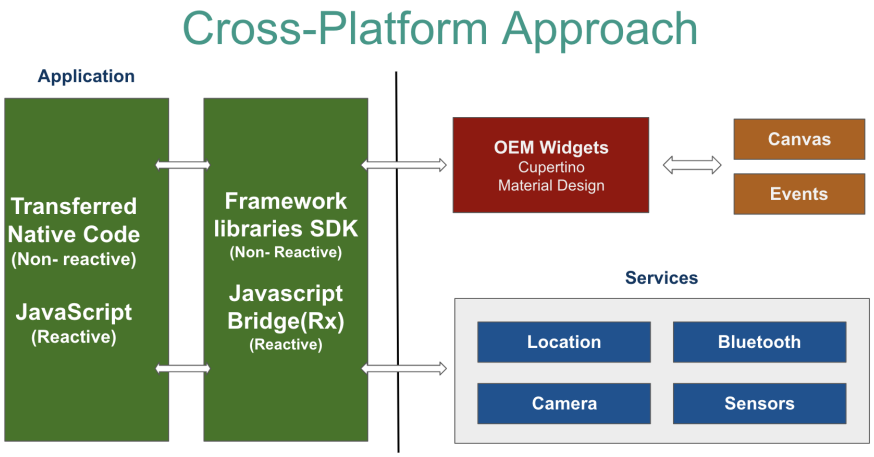


Figura 1.3: Architettura di applicazioni cross platform

I benefici derivanti da questo tipo di programmazione sono:

- sviluppo rapido in quanto, utilizzando la stessa codebase per più sistemi operativi, è possibile riutilizzare diverse parti di codice.
- costi minori poiché si utilizzano linguaggi universali.
- coerenza tra le versioni dei vari sistemi operativi, che vengono programmate in maniera univoca.

Allo stesso tempo, gli svantaggi derivanti sono:

- minori prestazioni nonostante il codice venga tradotto in codice nativo da compilatore.

- design più complesso poiché è necessario gestire le eccezioni di tutte le piattaforme.
- tempi più lunghi per ottenere le nuove funzionalità dei sistemi operativi.
- limitazioni all'accesso alle funzionalità specifiche del dispositivo e della piattaforma.

1.4 Flutter

Flutter è una delle idee più importanti dell'ultimo decennio [3]. È un framework open-source di casa Google nato nel 2017 che permette la creazione di applicazioni per Android e iOS e rientra nei sistemi di sviluppo cross platform ma introduce alcune novità: non utilizza le WebView o i widget OEM, ma sfrutta il suo motore di rendering per creare i widget migliorando notevolmente l'esperienza dell'utente.

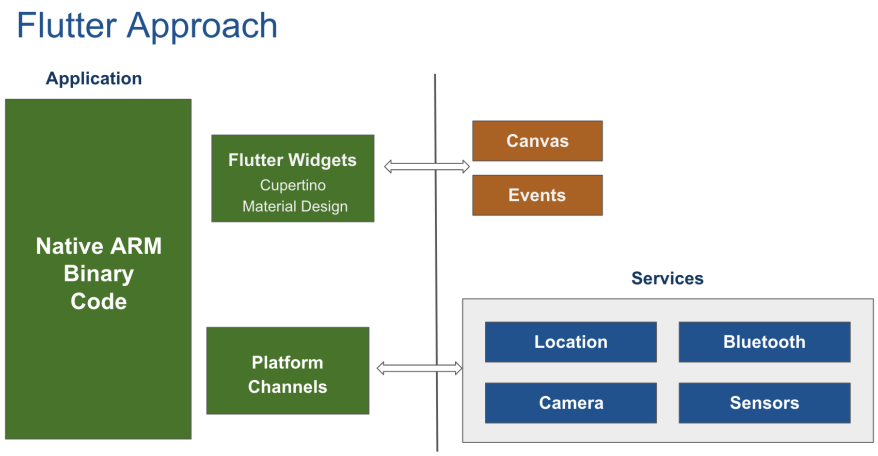


Figura 1.4: Architettura di applicazioni in Flutter

Flutter è ricco di vantaggi tra cui:

- sviluppo rapido dato dagli innovativi strumenti forniti e dalla facilità di costruzione di applicazioni complesse a partire dalla combinazione di widget basilari.
- velocità e performance pari a quelle di un'applicazione nativa.

- design ed esperienza utente al pari delle applicazioni nativa grazie a librerie con widget dedicati.
- grande supporto da parte della casa madre e della grande community di sviluppatori legata a Google.
- linguaggio di programmazione con una curva di apprendimento veloce, in quanto orientato agli oggetti e con molte similitudini a Java e C#.

Presenta però anche alcuni svantaggi:

- dimensione dell'APK relativamente grande rispetto agli altri tipi di applicazione.
- nuovo tool con alcune instabilità date dalla novità del framework.
- difficoltà nell'integrazione con strumenti di terze parti che ancora non supportano questo framework.
- linguaggio di programmazione poco diffuso e conosciuto dagli sviluppatori.

Capitolo 2

Flutter

Flutter è uno strumento che permette di creare applicazioni native cross platform per Android e iOS, ovvero usa una sola codebase ed un solo linguaggio di programmazione per creare due applicazioni perfettamente adattate allo stile di ognuno dei due sistemi operativi.
A partire dalla versione 1.12, in beta è stata rilasciata un'altra funzionalità interessante, ovvero la possibilità di ottenere un'applicazione web dallo stesso codice utilizzato per creare l'applicazione mobile.
Flutter fornisce un SDK per compilare facilmente il codice sorgente in codice adatto ad Android o iOS ed una libreria ricca di widget, funzioni e package per personalizzare l'interfaccia grafica [4].
Come mostrato in figura 2.1, le principali componenti di Flutter sono:

- il framework basato su Dart, linguaggio finalizzato allo sviluppo di interfacce utente frontend che quindi può essere utilizzato, oltre che per le applicazioni mobile, anche per sviluppare applicazioni web. È un linguaggio ad oggetti fortemente tipizzato, con una sintassi simile a Javascript, Java e C#.

- l'engine scritto in C++ ed utilizzato per il rendering di basso livello attraverso l'uso della libreria grafica Skia Graphics di Google. È il punto di contatto con gli SDK delle varie piattaforme. Una delle caratteristiche principali è l'*hot reload*: tramite l'iniezione del codice modificato nella Dart VM attiva, Flutter aggiorna le classi e le funzioni, ricostruisce l'albero di widget e permette di visualizzare più velocemente le modifiche apportate.

- l'embedder, basato sulla piattaforma, è il punto di contatto con il dispositivo in quanto interagisce con il codice nativo.

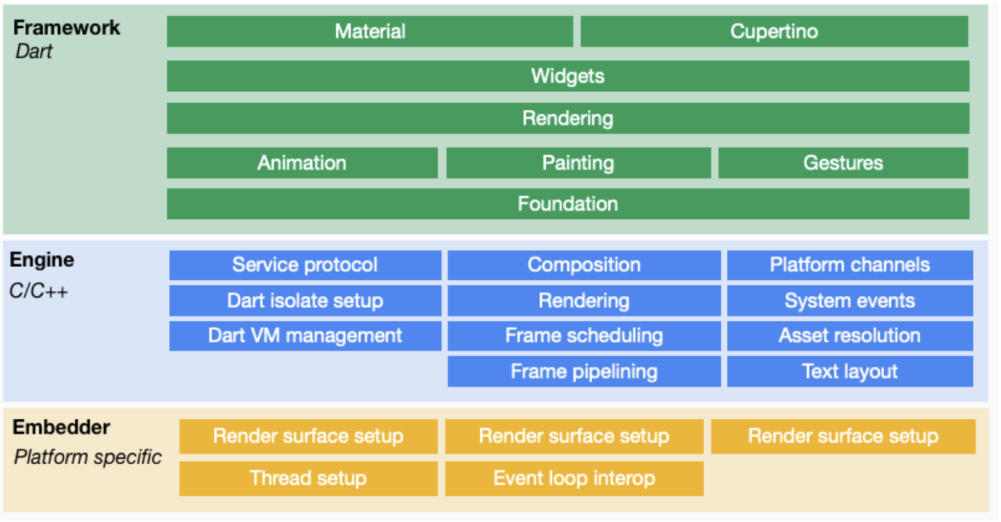


Figura 2.1: Architettura di Flutter

2.1 Il framework

Flutter si basa su un uso massiccio della composizione, in quanto tutte le interfacce che vengono create sono composte da un alto numero di widget. Il carico di lavoro viene quindi gestito tramite algoritmi sub lineari per costruire i widget e strutture che rendano efficiente la creazione dell'albero in modo da avere un'ottimizzazione costante [5].

2.1.1 Widget

In Flutter tutto è un *widget*, ovvero un elemento grafico che può adattarsi dinamicamente in base alle combinazioni ed alle logiche scelte dal programmatore. Per questo motivo, le applicazioni create in Flutter vengono chiamate *alberi di widget*.

I widget possono essere di due tipi:

- Stateless widget, non contengono informazioni sul loro stato.
- Stateful widget, contengono informazioni sul loro stato.

Stateless widget

I widget di tipo Stateless creano una parte dell'interfaccia grafica e contengono al loro interno altri widget a cascata, che descrivono più nei dettagli la grafica. Vengono utilizzati principalmente per quelle porzioni di interfaccia che dipendono esclusivamente dalle informazioni di configurazione fornite all'oggetto stesso.
Il metodo alla base dei widget è il *build()*, che permette di definire il contenuto del widget stesso e la sua logica. Viene richiamato ogni volta che una dipendenza del widget cambia, e quindi il suo aspetto deve essere modificato [6].

Stateful widget

Gli Stateful widget posseggono uno stato mutabile. L'informazione legata allo stato può essere letta in maniera sincrona quando il widget viene creato, tramite la build come per gli stateless widget, ma può anche cambiare durante il ciclo di vita del widget stesso. Per assicurarsi che questa modifica venga visualizzata anche a livello grafico, quest'ultima deve essere segnalata tramite il metodo *setState()*, fondamentale per i widget di questo tipo. Il metodo contiene la logica e forza Flutter a ricostruire l'albero dei widget secondo le modifiche attuate [7].

Librerie Cupertino e Material

A partire dal 2014, Google ha adottato un nuovo design chiamato *Material*, che segue delle regole di progettazione sull'uso di layout a griglia insieme ad animazioni, transizioni ed effetti di profondità basati su luce ed ombre. Questo design è supportato nativamente solo a partire da Android 5.0 ma, attraverso l'utilizzo di Flutter, è possibile ottenerlo anche su dispositivi con versioni precedenti, in quanto i widget implementano la libreria Material che segue lo stile di casa Google. Per mantenere uno stile familiare anche per le applicazioni destinate ad iOS, Flutter include anche la libreria Cupertino, che contiene gli stessi widget della libreria Material ma modificati secondo il design di Apple.

2.1.2 Composizione di widget

La *componibilità aggressiva* è uno degli aspetti caratteristici di Flutter [8]. Tutti i widget infatti sono composti da tanti widget single-purpose più piccoli, che vengono combinati tra di loro per ottenere il miglior effetto possibile. Uno degli esempi più comuni è il *Container*, che permette di posizionare nello spazio, dare forma, colore e stile ad un altro widget. Tutto questo avviene tramite *LimitedBox*, *Padding*, *DecoratedBox* ed altri widget che lo compongono, i quali valori possono essere modificati per ottenere qualsiasi combinazione. A loro volta questi widget sono composti da widget ancora più semplici, a volte ridotti a singole proprietà, che ne descrivono le caratteristiche più specifiche. Un esempio di composizione è mostrato in figura 2.2.

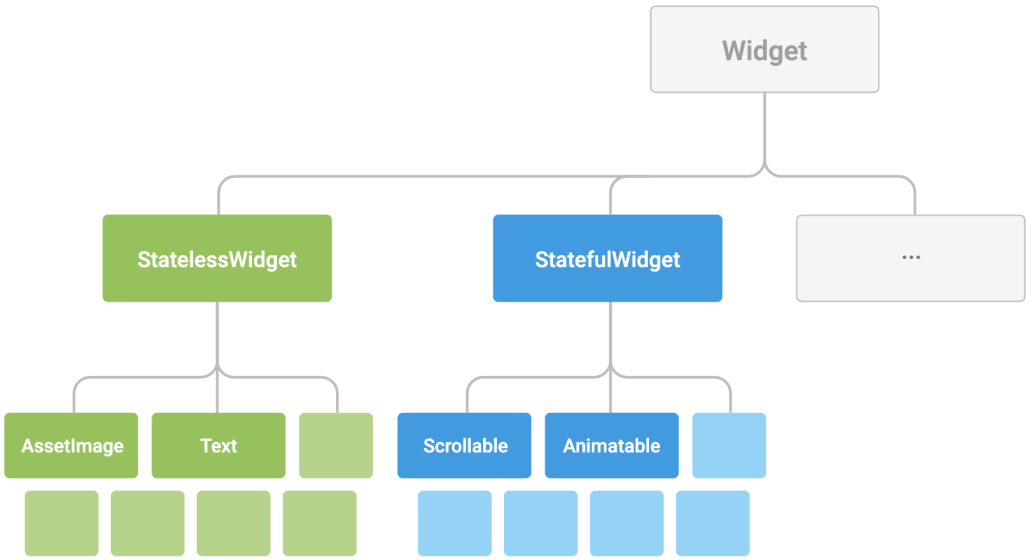


Figura 2.2: Composizione di widget

2.1.3 Layout dei widget

La composizione basata sui widget è un vantaggio a livello di programmazione, ma deve essere ben gestita al momento del rendering per non rallentare eccessivamente il processo di sviluppo. Di fondamentale importanza è l'algoritmo di *layout*, ovvero quell'algoritmo che determina la geometria degli oggetti da renderizzare [9]: contrariamente ad altri toolkit che usano algoritmi con complessità quadratica, Flutter punta a raggiungere una complessità lineare

nel caso della prima compilazione e sub lineare per l'aggiornamento di un layout già creato.
Flutter esegue un layout per frame e l'algoritmo lavora in un singolo passaggio. All'interno dell'albero di widget, gli elementi padre chiamano il metodo di layout su tutti i loro figli in maniera ricorsiva, fino ad arrivare alle foglie: queste ultime eseguono il metodo e restituiscono la geometria risultante ai nodi sovrastanti, fino a tornare alla radice dell'albero. I nodi che ritornano non verranno visitati nuovamente per il layout fino al frame successivo.
Durante l'operazione di layout, gli unici dati che vengono trasmessi nell'albero sono quelli inerenti la geometria degli elementi; esistono quindi vari casi in cui il lavoro dell'algoritmo può essere ridotto:

- se un elemento non ha modificato la propria geometria e quindi può ritornare immediatamente.
- se un elemento padre non utilizza la geometria dei figli e quindi può non attendere il loro ritorno.
- in presenza di vincoli stringenti, che possono essere soddisfatti da una sola geometria, imposti dall'elemento padre che quindi può non attendere il layout dei figli in caso di modifiche.
- se un elemento utilizza i vincoli forniti dall'elemento padre solo per creare la sua geometria, anche se non si tratta di vincoli stringenti, l'elemento padre può non attendere il suo nuovo layout, in quanto questo può cambiare solo in caso di modifica ai vincoli forniti dall'elemento padre stesso.

Questi casi particolari permettono dunque all'algoritmo di layout di non visitare l'intero albero migliorando le prestazioni.

2.1.4 Build dei widget

Un altro algoritmo con complessità sub lineare è quello di costruzione dei widget [10]. Subito dopo la costruzione, i widget sono inseriti all'interno dell'*albero degli elementi*, che sostiene la logica di tutta l'interfaccia utente. La presenza di questa struttura è fondamentale in quanto i widget non mantengono le relazioni con gli elementi padri e figli. L'albero, inoltre, contiene le informazioni sullo stato associate ai widget di tipo stateful.
Gli elementi modificati, per esempio per un cambio di stato in seguito ad

una *setState()*, vengono contrassegnati come “dirty” e viene creata una lista contenente questi ultimi: durante la fase di build saranno gli unici ad essere visitati per essere modificati. In questa fase gli elementi vengono visitati solo una volta in quanto una volta aggiornati non possono tornare "sporchi" e, per induzione, anche tutti i nodi sovrastanti saranno aggiornati.
Per evitare di dover visitare tutti i nodi durante il processo di costruzione, Flutter utilizza gli *InheritedWidget*: si crea una hash table per ogni elemento che contiene tutti gli elementi ad esso collegati. Solitamente la stessa hash table è condivisa da più elementi e viene modificata solo all'aggiunta di un nuovo *InheritedWidget*.

Una caratteristica interessante della build dei widget è la cosiddetta build-on-demand [11] : questo sistema permette a Flutter di iniziare la build degli elementi solo quando effettivamente devono essere visualizzati a schermo (per esempio durante lo scorrimento di una lista). Questo avviene interlacciando le fasi di build e layout della pipeline di compilazione: in qualsiasi momento della fase di layout Flutter può iniziare a costruire nuovi widget su richiesta, a patto che tali widget siano discendenti dell'oggetto di rendering che sta eseguendo il layout. Questo interlacciamento è reso possibile dai controlli svolti dagli algoritmi di build e layout sulla propagazione delle informazioni. Durante la fase di compilazione infatti, le informazioni possono propagarsi solo in maniera discendente all'interno dell'albero: quando un oggetto di rendering sta eseguendo il layout, l'algoritmo di layout non avrà ancora visitato il sotto albero dell'oggetto, quindi i risultati della build di quel sotto albero non possono invalidare le informazioni necessarie al calcolo del layout. Allo stesso modo, una volta che il layout è terminato in un oggetto di rendering, quell'oggetto di rendering non sarà mai più visitato durante questo layout, il che significa che qualsiasi scrittura generata dai calcoli di layout successivi non può invalidare le informazioni utilizzate per costruire il sotto albero dell'oggetto di rendering.

2.1.5 Reconciliation

Nonostante sia ricorrente il concetto di albero di widget, Flutter non utilizza un algoritmo di *tree-diffing*: il framework infatti utilizza un algoritmo lineare per determinare quali elementi figli possono essere riutilizzati per ogni elemento [12]. Alcuni casi permettono all'algoritmo di *reconciliation* di ottimizzare il proprio lavoro:

- la lista degli elementi figli è vuota, dunque non è necessario creare elementi.

- due liste di elementi figli sono esattamente identiche, quindi tutti gli elementi possono essere riutilizzati.

- c'è l'aggiunta o la cancellazione di uno o più elementi in una sola posizione della lista.

- se ogni lista contiene un widget con la stessa chiave, allora quest'ultima produce un match.

In generale, si controllano l'inizio e la fine delle due liste da confrontare cercando dei match tra le chiavi dei widget, con la possibilità di trovare un intervallo non vuoto contenente tutti gli elementi che non producono un match. Questi elementi vengono inseriti in una lista che viene visitata e confrontata con l'hash table delle chiavi per un confronto: in caso di match i widget vengono costruiti a partire dall'elemento trovato, altrimenti la costruzione inizia da zero.

2.1.6 Spostamento di widget

Il riutilizzo di elementi è fondamentale per le performance in quanto all'interno degli elementi sono contenute informazioni vitali per l'interfaccia, come lo stato e il rendering dei livelli più bassi. Riutilizzare un elemento porta a preservare lo stato e le informazioni di layout, evitando cosi di attraversare nuovamente l'albero di widget per ricostruirle. Per favorire il riutilizzo degli elementi Flutter supporta le modifiche non locali all'albero per preservare lo stato e le informazioni sul layout [13].
Le modifiche non in locale vengono eseguite associando una *GlobalKey* ai widget: ogni chiave è univoca all'interno dell'intera applicazione ed è contenuta in una hash table specifica per un singolo thread. Spostando dei widget con una Globalkey in una qualsiasi posizione dell'albero degli elementi questi non vengono ricostruiti da zero, ma viene fatta una ricerca per una corrispondenza nella hash table per spostare l'elemento nella nuova posizione preservando tutto il sotto-albero che lo contiene.
Gli elementi spostati nel nuovo albero mantengono le informazioni sul loro layout e gli elementi sovrastanti vengono contrassegnati come "sporchi", in

quanto la lista dei loro figli è cambiata. Nel caso in cui il nuovo elemento padre possieda gli stessi vincoli del precedente, il figlio può ritornare immediatamente il suo layout, ottimizzando il processo.

2.1.7 Ottimizzazioni a fattore costante

Per raggiungere la componibilità aggressiva alla quale Flutter punta non sono sufficienti gli algoritmi altamente ottimizzati finora descritti, ma sono necessarie anche delle ulteriori ottimizzazioni a fattore costante [14].

- *Assenza di child-model*: contrariamente agli altri framework, Flutter non utilizza un child-model specifico. Le classi infatti non possiedono metodi specifici per la visita dei singoli nodi figli, ma dei metodi astratti che visitano indistintamente tutti i figli. Alcune sottoclassi invece, non supportano una lista di figli ma un singolo nodo figlio gestito all'interno della classe come una variabile.

- *Alberi logici e virtuali*: l'albero di rendering utilizzato da Flutter lavora su un sistema di coordinate visive indipendente del dispositivo, che posiziona i valori più piccoli verso sinistra, indipendentemente dal verso di lettura. L'albero dei widget invece utilizza le coordinate logiche, la cui interpretazione è vincolata al verso di lettura. La trasformazione da coordinate logiche a coordinate visive avviene nel passaggio dall'albero dei widget all'albero di rendering. Questo approccio risulta essere il più efficiente in quanto le modifiche visive attuate dall'albero di rendering avvengono più spesso del passaggio tra i due alberi, quindi è possibile evitare la continua conversione delle coordinate.

- *Gestione del testo*: per gestire al meglio la complessità del testo si utilizza un oggetto di rendering apposito, il *RenderParagraph*, che rappresenta una foglia dell'albero di rendering. Questo elemento consente di evitare la ricomputazione del layout del testo fin quando i nodi sovrastanti non modificano i propri vincoli di layout.

- *Observable*: Flutter utilizza entrambi i paradigmi *model-observation* e *reactive*. Il primo è utilizzato per alcuni strutture dati contenute nelle foglie, ad esempio le animazioni, mentre il secondo è quello dominante all'interno del framework.

2.1.8 Separazione tra alberi

Gli alberi di rendering e quello dei widget sono isomorfi, in quanto il primo è un subset del secondo. Una semplificazione che sembrerebbe ovvia sarebbe l'unione dei due alberi in uno singolo, ma ci sono diversi vantaggi che questa separazione fornisce [15]:

- *Prestazioni*: in caso di modifiche al layout vengono percorse solo le parti più rilevanti dell'albero e, grazie alla composizione, è possibile evitare la visita di diversi altri nodi.
- *Chiarezza*: la separazione netta tra i due alberi permette ai due protocolli di essere molto specifici per le diverse necessità, oltre a diminuire la possibilità di bug e la difficoltà nel test.
- *Type-safety*: l'albero di rendering fornisce una maggiore sicurezza nella gestione dei tipi in quanto garantisce che a runtime i nodi figli avranno il tipo appropriato al loro utilizzo. L'albero dei widget invece non è a conoscenza del sistema di coordinate utilizzato durante il layout e questo richiederebbe un'ulteriore visita dell'albero per verificare la tipologia degli oggetti da renderizzare.

2.2 L'engine

L'Engine di Flutter è un runtime portatile per applicazioni mobili di alta qualità che implementa le librerie di base tra cui animazione e grafica, file e I/O di rete, supporto all'accessibilità, architettura a plugin, un runtime Dart e una toolchain per lo sviluppo, la compilazione e l'esecuzione di applicazioni Flutter.
Al suo interno troviamo una shell che ospita una libreria di rendering grafico in 2D, *Skia*, e *Dart*, una VM per il linguaggio di programmazione orientato agli oggetti.

2.2.1 Skia

Skia è una libreria grafica 2D open-source scritta in C++ creata da Google [16]. Fornisce API funzionanti su una grande varietà di piattaforme hardware e software, come Google Chrome e Chrome OS, Android, Mozilla Firefox e Firefox OS. Tra i vari backend che Skia possiede possiamo trovarne uno per

la rasterizzazione del software basato su CPU, uno per l'output PDF e uno per OpenGL, altra libreria grafica scritta in C++, accelerato da GPU.

2.2.2 Dart VM

Per la programmazione nativa, la VM di Dart include due modalità operative, JIT e AOT.

JIT

La modalità JIT [17] supporta l'interpretazione pura e l'ottimizzazione a runtime. In questa modalità la VM riesce a caricare dinamicamente il codice sorgente in Dart, eseguire il parsing e compilarlo in codice macchina nativo *on the fly* per eseguirlo. Questa modalità viene utilizzata durante lo sviluppo dell'applicazione e fornisce funzionalità come il debugging e l'hot reload.

AOT

La modalità AOT [18] viene utilizzata nel momento in cui l'applicazione è pronta per essere sviluppata in produzione, quindi è necessario compilare l'applicazione in codice macchina ARM o x64. Questa modalità non supporta caricamento, parsing e compilazione dinamiche, ma soltanto il caricamento e l'esecuzione di codice macchina precompilato. Anche questo codice macchina, però, necessita della VM per essere eseguito, in quanto la VM fornisce un sistema runtime che fornisce un *garbage collector*, diversi metodi nativi necessari alle librerie di Dart per funzionare e informazioni sui tipi a runtime. Il codice macchina precompilato che viene fornito alla modalità AOT è generato da una particolare modalità della VM di Flutter quando si esegue il build applicazione in modalità rilascio.

2.3 L'embedder

L'engine non crea né gestisce i suoi thread: questo processo è compito dell'embedder, che crea e gestisce i processi per l'engine e gli fornisce i *task runner* [19]. In aggiunta ai thread gestiti dall'embedder per l'engine, la VM possiede una sua thread pool, alla quale né l'engine né l'embedder hanno accesso.

2.3.1 Configurazione dei task runner

I riferimenti ai task runner richiesti dall'engine all'embedder sono 4. L'embedder non controlla se i riferimenti puntano allo stesso task runner o se più task runner sono in esecuzione sullo stesso thread, ma per gestire al meglio le prestazioni l'embedder dovrebbe creare un thread dedicato per ogni task runner. L'engine si aspetta che la configurazione dei thread rimanga stabile fino alla fine del suo ciclo di vita quindi, una volta che l'embedder assegna una thread ad un dato task runner, quest'ultimo verrà sempre eseguito su quel thread, finché l'engine non verrà fermato.
I task runner principali sono descritti in seguito.

Platform Task Runner

Questo task runner viene assegnato al thread che l'embedder considera il principale. Ogni interazione con l'engine di Flutter deve avvenire solo tramite il thread associato al platform task runner, in quanto tutte le altre interazioni non sono sicure nelle build di rilascio.
Questo task runner esegue tutti i messaggi della piattaforma in attesa: questo avviene perché l'accesso alla maggior parte delle API della piattaforma è sicuro solo sul thread principale della piattaforma. Anche se il thread non andrà in blocco in caso di sovraccarico, le piattaforme impongono delle restrizioni sull'utilizzo del thread per operazioni computazionalmente costose: queste ultime possono essere eseguite su un thread separato (non incluso nei 4 thread forniti all'engine) per mettere poi in coda le risposte sul thread principale.

UI Task Runner

Questo task runner è quello in cui l'engine esegue tutto il codice Dart per il *root isolate*, ovvero una porzione di codice che possiede i binding necessari a Flutter per funzionare. Il root isolate esegue il codice principale dell'applicazione, ed i binding in esso contenuti sono impostati dall'engine in modo tale da schedulare e inviare i frame.
Per ogni frame che Flutter deve renderizzare:

- la root isolate specifica all'engine quale frame renderizzare.
- l'engine chiede alla piattaforma di essere notifica sul prossimo vsync.

- la piattaforma attende il vsync successivo.
- al vsync l'engine richiama il codice Dart e:
  - aggiorna gli interpolatori delle animazioni.
  - ricostruisce i widget.
  - posiziona i widget in un albero a strati che non sarà ancora rasterizzato, ma che contiene solo le informazioni riguardo ciò che deve essere mostrato.
  - crea o aggiorna un albero contenente le informazioni semantiche dei widget da mostrare.

Poiché questo thread gestisce tutto ciò che è visualizzato a schermo, lunghe operazioni asincrone possono provocare ritardi nell'applicazione. Queste lunghe operazioni possono solo essere causate dal codice Dart, in quanto l'engine non inserirà nessun task in codice nativo su questo task runner.

GPU Task Runner

Questo task runner è addetto all'esecuzione di compiti che necessitano dell'uso della GPU del dispositivo. Riceve quindi l'albero di livelli creato dallo UI task runner e applica i comandi GPU più appropriati per renderizzarlo. I componenti del GPU task runner sono anche responsabili della configurazione di tutte le risorse GPU per ogni singolo frame. Questo include il contatto con la piattaforma per impostare il framebuffer, la gestione del ciclo di vita della schermata e la garanzia che le texture e i buffer per un particolare frame siano completamente pronti.
A seconda del tempo necessario all'elaborazione dell'albero di livelli e alla GPU per finire di visualizzare il frame, i vari componenti di questo task runner possono ritardare la programmazione di ulteriori frame sul thread dell'interfaccia utente: questo può dunque portare a ritardi nella programmazione dei frame dell'applicazione, per questo è consigliato avere un thread dedicato al GPU task runner per ogni istanza dell'engine.

I/O Task Runner

La funzione principale del I/O task runner è la lettura di immagini compresse dagli asset e la verifica della disponibilità di queste ultime per il rendering sul GPU task runner. Per assicurarsi che una texture sia pronta per il

rendering, deve prima essere letta come un blob di dati compressi dall'asset, decompressi in un formato compatibile con la GPU e caricati su di essa. Queste operazioni sono costose e causano un blocco se eseguite sul GPU task runner. Poiché solo quest'ultimo può accedere alla GPU, i componenti del I/O task runner impostano un contesto speciale che si trova nello stesso sharegroup del contesto principale del GPU task runner. L'unico modo per ottenere una risorsa è tramite una chiamata async: questo permette al framework di parlare con il runner di I/O in modo che possa eseguire in modo asincrono tutte le operazioni di texture menzionate. L'immagine può quindi essere immediatamente utilizzata in un frame senza che il thread della GPU debba fare un lavoro costoso.

Capitolo 3

Geolocalizzazione e tracking

3.1 Geolocalizzazione

La geolocalizzazione è la tecnologia che utilizza i dati acquisiti da un computer o dispositivo mobile personale per identificare e descrivere la posizione fisica attuale dell'utente [20].
Con l'avvento dei dispositivi mobili e delle nuove tecnologie, la posizione degli utenti o dei beni è diventata un dato prezioso che può essere sfruttato per diversi fini quali: la pubblicità mirata, la navigazione, il tracciamento delle spedizioni e dei trasporti, la realtà aumentata, la guida autonoma, la sicurezza, la sanità e tanti altri. Questo ha spinto a creare tecnologie sempre più precise e a risparmio energetico, in modo da poterle integrare su chip sempre più piccoli. Tra queste tecnologie le più usate sono:

- geolocalizzazione BLE, basata sul più recente standard del Bluetooth.
- geolocalizzazione WiFi-based, che sfrutta tutte le reti WiFi nelle vicinanze.
- geolocalizzazione network-based, che recupera la posizione utilizzando i segnali delle celle telefoniche.
- GPS, tecnologia che utilizza i satelliti che orbitano attorno alla Terra.
- A-GPS, versione migliorata e più efficiente del GPS.

3.1.1 Geolocalizzazione BLE

Il Bluetooth è uno standard di comunicazione wireless a corto raggio, infatti viene utilizzato per far comunicare dispositivi non più lontani di 10 metri. La nascita del Bluetooth risale a quasi due decenni fa ma negli ultimi anni la sua versione a bassa energia sta facendo grandi passi nell'utilizzo in campo di geolocalizzazione e posizionamento.
L'aspetto vantaggioso di questa tecnologia è il non richiedere interazioni con l'infrastruttura IT per le comunicazioni e che la sua installazione è molto semplice, in quanto avviene tramite dispositivi a batteria molto economici; I trasmettitori bluetooth usano il BLE per trasmettere un identificatore univoco universale, che viene utilizzato per individuare la posizione del ricevitore stesso, ma non per tracciarne i movimenti. La precisione della posizione individuata cresce all'aumentare dei trasmettitori bluetooth presenti nelle vicinanze.
Il BLE funziona con molta precisione all'interno ed in zone con dimensioni ridotte mentre all'esterno la precisione diminuisce notevolmente, per questo motivo viene utilizzato principalmente per creare dei piccoli tracker da attaccare ai dispositivi di uso quotidiano, come chiavi o portafogli, che spesso vengono dimenticati nei punti più remoti delle nostre abitazioni.
In generale questa tecnologia può essere integrata facilmente nelle infrastrutture già presenti e fornisce una durata della batteria molto alta, grazie al consumo energetico molto basso che richiede.

3.1.2 Geolocalizzazione WiFi-based

La geolocalizzazione WiFi-based rientra nelle reti WLAN, ovvero quelle reti che si connettono a delle frequenze radio specifiche, come 2.4GHz e 5.0GHz. I dati che viaggiano su questo tipo di rete hanno un raggio di trasmissione di massimo 100 metri, per cui questo tipo di localizzazione può funzionare sia all'interno che all'esterno.
La caratteristica principale di questa modalità è che sfrutta anche le reti WiFi alle quali non possiamo accedere: per geolocalizzare un elemento infatti si utilizzano le informazioni pubbliche delle reti, come l'IP, il BSSID ed il MAC, per determinarne la posizione. Esistono diversi tecniche per sfruttare questo tipo di localizzazione, ed è possibile raggrupparle in 4 tecnologie:

- *RSSI*: questa tecnica si basa sulla misura dell'intensità del segnale inviato dal dispositivo a diversi access point. La combinazione delle informazioni

con un modello di propagazione permette di determinare la distanza tra il dispositivo e l'access point; queste distanze possono quindi essere utilizzate per calcolare la posizione del dispositivo, relativamente agli access point.

- *fingerprinting*: stesso funzionamento del RSSI ma con l'aggiunta di un database che conserva le posizioni del dispositivo durante le fasi offline, le quali vengono calcolate in modo probabilistico [21]. Nel momento in cui è necessario determinare la posizione si compara il vettore di dati ottenuto tramite RSSI con la posizione più recente salvata all'interno del database per avere un calcolo più veloce.

- *ToF*: questo metodo utilizza il timestamp fornito dall'interfaccia wireless per calcolare il ToF dei vari segnali inviati, utilizzando questa informazione per stimare la distanza tra il dispositivo e l'access point. Anche in questo caso le informazioni vengono combinate per calcolare la posizione del dispositivo, relativamente agli access point. Per utilizzare questa tecnica è necessario prestare attenzione al problema della sincronizzazione degli orologi interni ai dispositivi, che però può essere eliminato tramite degli approcci matematici [22].

- *AoA*: per sfruttare questa tecnologia sono necessarie delle antenne direzionali, che possano quindi ricevere il segnale in maniera accurata solo da una direzione. Ogni antenna nelle vicinanze del dispositivo la localizzare calcola l'angolo che si forma tra il loro punto di ricezione ed il segnale inviato dal dispositivo; tramite l'intersezione degli angoli calcolati da due o più antenne è possibile quindi stabilire la posizione del dispositivo, come mostrato in figura 3.1.

I vantaggi della localizzazione WiFi sono, oltre al non aver bisogno di nessuna infrastruttura, il basso consumo energetico e la buona precisione fino a 10 metri.

3.1.3 Geolocalizzazione network-based

La posizione di un oggetto può essere determinata tramite l'infrastruttura di rete di un fornitore di servizi: la precisione di questa modalità varia in base al numero di stazioni presenti nelle vicinanze dell'oggetto, non ha bisogno di un'infrastruttura dedicata ed ha un consumo di energia molto ridotto.
La tecnica utilizzata per ottenere la posizione è chiamata *triangolazione*, ed

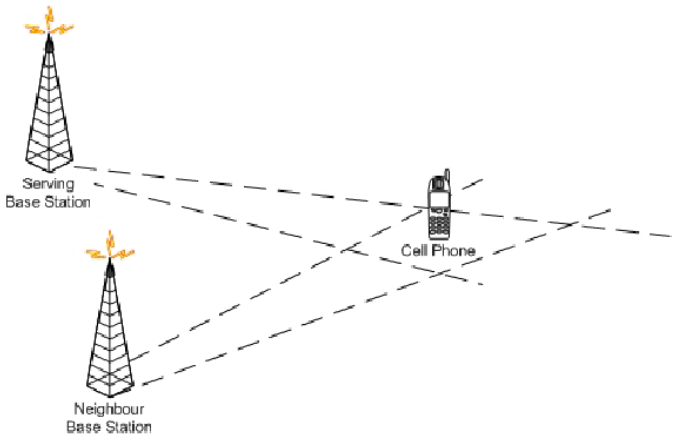


Figura 3.1: Acquisizione della posizione tramite AoA

è mostrata in figura 3.2: grazie alla connessione dei dispositivi a più celle e con l'utilizzo di algoritmi matematici si è in grado di determinare (con un errore di qualche km) la posizione del dispositivo. Esistono diverse celle, che forniscono diverse informazioni e precisione per la localizzazione:

- *cella omnidirezionale*: é creata da antenne che irradiano in tutte le direzioni indistintamente, e permette di conoscere la posizione con precisione massima pari al raggio della cella stessa. Per aumentare la precisione di localizzazione si può ricorrere al *timing advance*: La rete, stimolando il dispositivo ad una connessione, stima la distanza, calcolando i tempi di ritardo nella trasmissione. La zona di localizzazione si riduce da un cerchio ad una corona circolare.
- *cella settore*: se il sito irradia più settori, é possibile localizzare il dispositivo con la precisione massima del settore circolare coperto dalla cella. Anche in questo caso si può ricorrere al timing advance: la zona di localizzazione si riduce da un settore circolare ad un settore di corona circolare.

3.1.4 GPS

Il GPS nasce con lo scopo di determinare con precisione e continuità la posizione e la velocità di un oggetto per fornire supporto alla navigazione. È un sistema basato sull'utilizzo di 30 satelliti orbitanti attorno alla Terra, configurati in modo da fornire agli utenti indicazioni su latitudine, longitudine

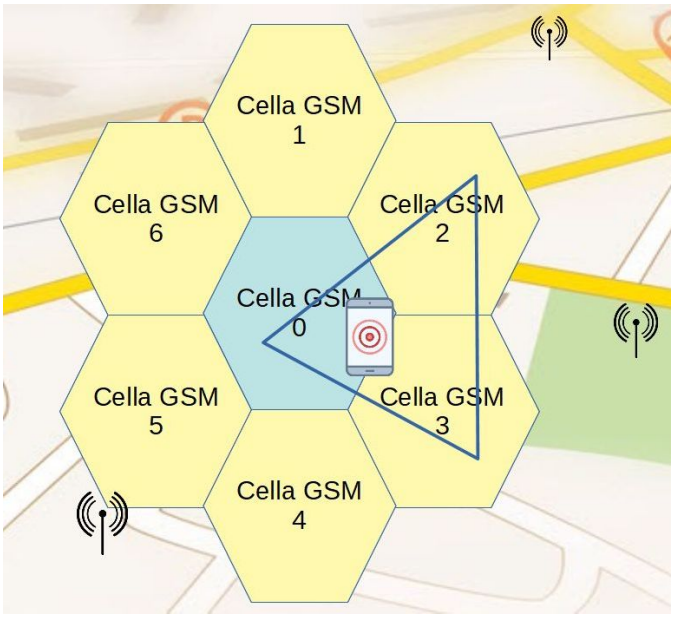


Figura 3.2: Triangolazione di rete

ed elevazione [23]. La posizione viene individuata tramite triangolazione, come mostrato in figura 3.3; per ottenere la posizione iniziale con l'indicazione di latitudine e longitudine sono necessari almeno 3 satelliti, 4 nel caso in cui si voglia conoscere anche l'elevazione, mentre ne bastano solo 3 per continuare a mantenere la posizione per il tracciamento. La fase di ricerca iniziale della posizione è chiamata *allineamento*, e può essere molto lunga a causa del tempo necessario alla ricerca dei satelliti. La fase successiva, il *posizionamento*, permette di stabilire la posizione precisa su una mappa, tramite l'utilizzo combinato di software di cartografia. L'ultima fase è quella di *navigazione* tramite la quale è possibile, con l'ausilio di software specifici, calcolare un percorso ottimale per raggiungere un altro punto sulla mappa.

Esistono alcuni problemi tecnici che devono essere gestiti per l'utilizzo del GPS:

- *Sincronizzazione degli orologi dei singoli satelliti e dei dispositivi*: tutti i satelliti sono equipaggiati di un orologio atomico che permette di mantenere l'orario aggiornato e preciso. Per fare in modo che anche il dispositivo ricevente abbia un orario accurato si utilizza il terzo satellite (o quarto nel caso in cui si calcoli anche l'elevazione), che comunica costantemente il suo orario in modo che il ricevente possa conoscere con

esattezza la differenza di tempo che intercorre tra i due orologi.

- *Posizione dei satelliti aggiornata in tempo reale*: questo problema viene risolto tramite l'utilizzo di un alto numero di stazioni di controllo posizionate sulla Terra che hanno il compito di misurare e trasmettere la posizione di ogni singolo satellite ai dispositivi che provano a collegarvisi.

Essendo basato su satelliti orbitanti attorno al pianeta, il GPS è praticamente sempre disponibile all'esterno, è molto accurato in quanto la precisione delle posizioni include un errore di soli 5 metri, e non richiede infrastrutture aggiuntive per il funzionamento. Questa metodologia presenta però anche alcuni svantaggi, come l'elevato consumo di batteria, dovuto alla ricerca ed alla comunicazione con i satelliti, la scarsa precisione all'interno di edifici e la possibilità di essere disturbata da eventi meteorologici avversi.

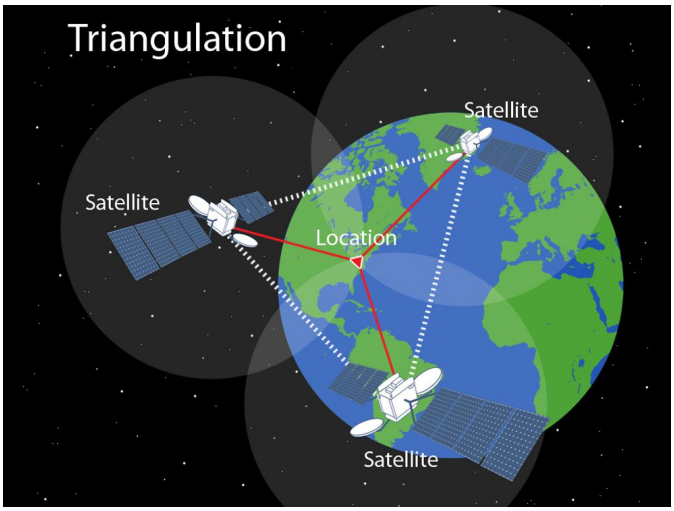


Figura 3.3: Triangolazione con satelliti GPS

3.1.5 A-GPS

L'A-GPS è un sistema che migliora e potenzia la ricezione del GPS in caso di segnale scarso da parte dei satelliti, appoggiandosi a rete dati e rete telefonica, in modo da ottenere informazioni aggiuntive sulla posizione, da aggiungere a quelle che derivate dai satelliti nello spazio, a scopo di ottenere un allineamento più rapido e mantenere un segnale più intenso durante il percorso. L'allineamento del GPS infatti, può richiedere dai 3 ai 30 minuti;

con l'utilizzo della modalità assistita invece, i tempi si riducono notevolmente passando ad un intervallo tra i 5 e i 20 secondi, in quanto l'indicazione iniziale della rete aiuta ad abbattere notevolmente i tempi di prima localizzazione da parte dei satelliti GPS. La riduzione dei tempi di allineamento permette all'A-GPS di essere particolarmente preciso ma meno dispendioso a livello di energia rispetto al GPS, per questo motivo è la tecnologia attualmente in uso su smartphone e dispositivi mobili.
Esistono due modalità di A-GPS:

- *MSA*: in questa modalità il dispositivo riceve assistenza nell'acquisizione, tempo di riferimento ed altre informazioni di assistenza opzionali da un service provider mobile. Il provider tiene un log aggiornato dei satelliti GPS e con l'aiuto di questi dati riesce a calcolare la posizione del dispositivo e ad inviargliela.

- *MSB*: in questa modalità il dispositivo riceve la posizione di riferimento, il tempo di riferimento ed altre informazioni di assistenza opzionali dal server A-GPS; tramite queste informazioni il dispositivo può individuare più facilmente i satelliti visibili e calcolare la propria posizione.

3.2 Tracking in background

L'attivazione della localizzazione tramite A-GPS è una funzione molto dispendiosa per gli smartphone, in quanto necessità di invio continuo di dati. Volendo utilizzare questa tecnologia per il rilevamento automatico delle strade percorse è servito trovare un compromesso per utilizzare la localizzazione in background in maniera ottimizzata.
Analizzando lo stato dell'arte, le soluzioni proposte sono state:

- la *user-triggered version*, ovvero l'attivazione e disattivazione manuale della localizzazione da parte dell'utente. Questa soluzione è sicuramente efficace ma non ottimizzata a livello di esperienza utente.

- l'utilizzo dei sensori integrati nel dispositivo, come accelerometro e giroscopio, per determinare le situazioni in cui attivare la localizzazione per acquisire dati [24].

- l'utilizzo di software di terze parti, che potessero essere integrati in Flutter all'interno dell'applicazione.

3.3 Strumenti di terze parti

3.3.1 Lotadata

Lotadata è un servizio gratuito di gestione e analisi di posizioni, formato da tre componenti che interagiscono l'uno con l'altro per fornire la migliore esperienza utente possibile [25].
Il primo, GeoSDK, è un componente software leggero ed efficiente, creato per raccogliere dati accurati dalle applicazioni mobile con la richiesta esplicita del consenso degli utenti; troviamo poi GeoDash, una dashboard basata su una mappa che fornisce un'interfaccia semplice ed intuitiva per le analisi delle posizioni, che riceve dati da GeoSDK e permette agli sviluppatori di creare geofences ed eventi trigger, oltre che attivare le notifiche tramite GeoPush, servizio di notifiche push attivato dai dati raccolti da GeoSDK.

Le posizioni raccolte tramite l'applicazione demo di Lotadata, raggruppabili per singolo dispositivo tramite un codice univoco che viene assegnato dal sistema, contengono anche [26]:

- un'indicazione sul tipo di movimento (fermo, oscillazione, camminata, bicicletta, veicoli) basato sull'algoritmo di riconoscimento interno al sistema. Questo permette di impostare automaticamente la frequenza con cui richiedere nuove posizioni.

- l'accelerazione su ogni asse e la velocità con cui il movimento è stato rilevato, utile per poter scremare ulteriormente i dati.

- la luminosità percepita dal sensore frontale dello smartphone (dove presente), che può essere utilizzata, nel caso di Wher, per analizzare la luminosità delle strade mentre il telefono viene utilizzato in modalità navigatore.

3.3.2 Transistorsoft

Transistorsoft è un software a pagamento per il tracking in background con intelligenza integrata per il motion-detection per Android ed iOS. Include anche una versione per Flutter.
Il cuore della filosofia della geolocalizzazione [27] in background è, come anticipato in 4.4, di tenere traccia della posizione del dispositivo nel modo meno

dispendioso possibile a livello energetico: proprio per questo entra in gioco il sistema di motion-detection, ovvero il determinare quando il dispositivo è in movimento o meno. Questa analisi avviene tramite l'utilizzo di accelerometro, giroscopio e magnetometro. Quando il dispositivo risulta essere in movimento il plugin attiva automaticamente i servizi di localizzazione ed inizia a registrare le posizioni sulla base del filtro di distanza impostato; contrariamente quando viene rilevata una situazione stazionaria, i servizi di localizzazione vengono disattivati per risparmiare energia.

3.4 Utilizzo della geolocalizzazione in Wher

Per un'applicazione basata su mappe e navigatore, la possibilità di avere un sistema di geolocalizzazione efficace ed efficiente è, senza dubbio, un aspetto cruciale per il funzionamento della stessa. Wher al momento utilizza un suo SDK personalizzato basato su Mapbox e sfrutta la geolocalizzazione quasi esclusivamente per la navigazione: questo comporta il non avere la necessità di una precisione estrema, in quanto l'utente riesce facilmente ad individuare la propria posizione sulla mappa avendo un'indicazione sulle strade che la circondano e visualizzando a schermo una posizione, anche se non estremamente accurata.
Nel caso della funzionalità che vuole essere implementata sfruttando il tracking in background invece, l'accuratezza è fondamentale in quanto è necessario avere un'indicazione più precisa possibile delle strade percorse.
Lo studio e l'analisi delle varie tecniche di localizzazione e tracking in background quindi sono stati basilari per poter implementare questa nuova funzionalità sull'applicazione basata sul sistema GPS; in particolare, l'obiettivo è l'utilizzo del tracking in background, ovvero la possibilità di ottenere un resoconto delle strade visitate durante la giornata per poterle valutare in maniera più pratica e veloce.
Gli strumenti e le tecniche utilizzate per implementare questa funzione sono descritti nel capitolo successivo.

Capitolo 4

Caso di studio

4.1 Wher

Wher è una startup tutta italiana nata da una grande idea e da una grande necessità: aiutare le donne a muoversi liberamente e senza paura per le vie delle città, conosciute e non [28]. Alla base di questo progetto c'è una grande community tutta al femminile, che a suon di passi e valutazioni colora le strade della mappa in base a illuminazione, affollamento e percezione soggettiva, come mostrato in figura 4.1. Sulla base della media delle valutazioni il navigatore dell'applicazione propone la strada più sicura invece che quella più veloce, anche nelle diverse fasce orarie.

Wher è adatta a tutte le donne, studentesse, lavoratrici o viaggiatrici che siano, l'importante è non volersi fermare e andare avanti con la consapevolezza dell'ambiente che le circonda.

Alla base di Wher ci sono tre concetti fondamentali: *insieme*, perché la forza di tutta l'applicazione sono le Wherrior, guerriere urbane che fanno propria la città vivendola, valutandone le strade ed aiutando altre donne a sentirsi più libere e sicure; *fiducia in se stesse*, in quanto il sentirsi a proprio agio per le strade dà una marcia in più alla nostra vita, aiutandoci ad abbattere il modello che impone che le donne non possano camminare da sole per strada; *impatto*, in quanto con un piccolo contributo e in poco tempo possiamo portare ad un grande cambiamento, aiutando migliaia di donne vicine a noi.

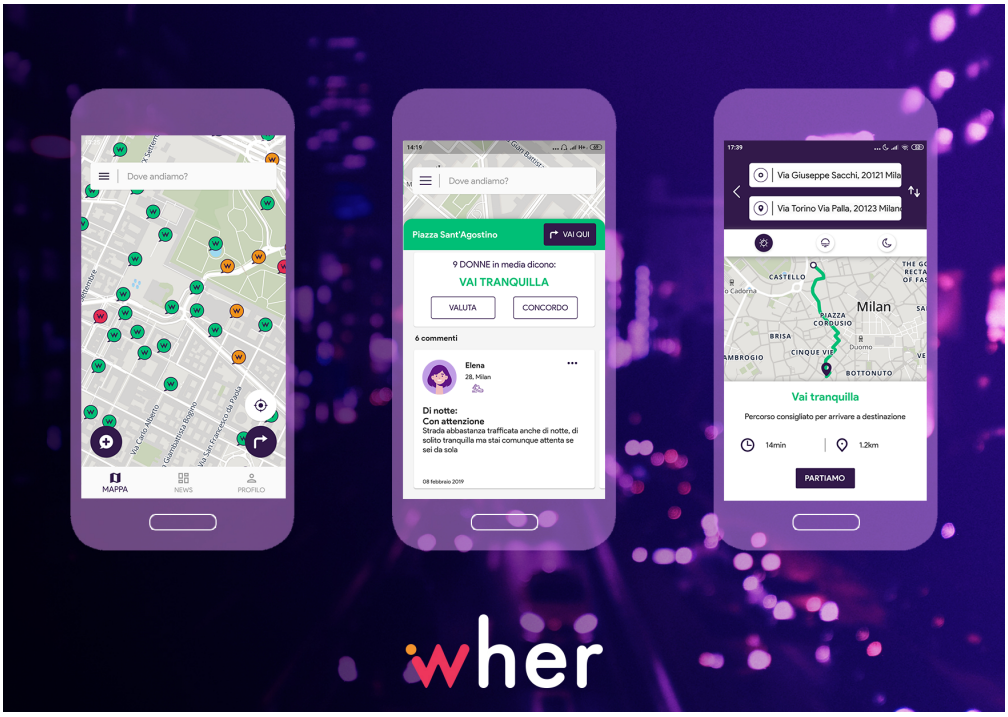


Figura 4.1: Schermate dall'applicazione Wher: mappe, valutazioni delle strade e navigatore.

4.2 Wher e l'innovazione tecnologica

In quanto startup Wher ha sempre avuto un numero ristretto di membri all'interno del proprio team, in particolare il settore IT era inizialmente formato da un unico sviluppatore full stack. La scelta iniziale della modalità da utilizzare per creare le prime versioni dell'applicazione è ricaduta sul nativo in quanto, oltre a fornire prestazioni e strumenti adeguati allo scopo, era una competenza già posseduta dallo sviluppatore presente e quindi risultava essere un buon compromesso in quanto non necessitava di studio e formazione aggiuntiva; di contro vi era la consapevolezza che tale scelta avrebbe portato ad un carico di lavoro maggiore e ad un maggiore investimento a livello finanziario.

Nonostante questa scelta iniziale, il team è sempre stato attento alle nuove proposte in ambito programmazione mobile ed poco dopo è venuto a conoscenza della nascita di Flutter; il primo rilascio di una versione stabile

del framework è stato per Wher un trampolino di lancio per sperimentare qualcosa di nuovo. Il passaggio da programmazione nativa a Flutter è stato un grande cambiamento che ha permesso l'ampliamento del settore IT del team, in quanto non era più necessario avere due sviluppatori specializzati per la programmazione Android ed iOS. Questo cambiamento ha permesso non solo di avere un notevole risparmio a livello economico e di tempo, ma anche di poter proporre dei progetti di tesi, come questo, che potessero lavorare in parallelo al progetto principale di sviluppo per lavorare a nuove funzionalità che potessero migliorare l'esperienza delle utenti.

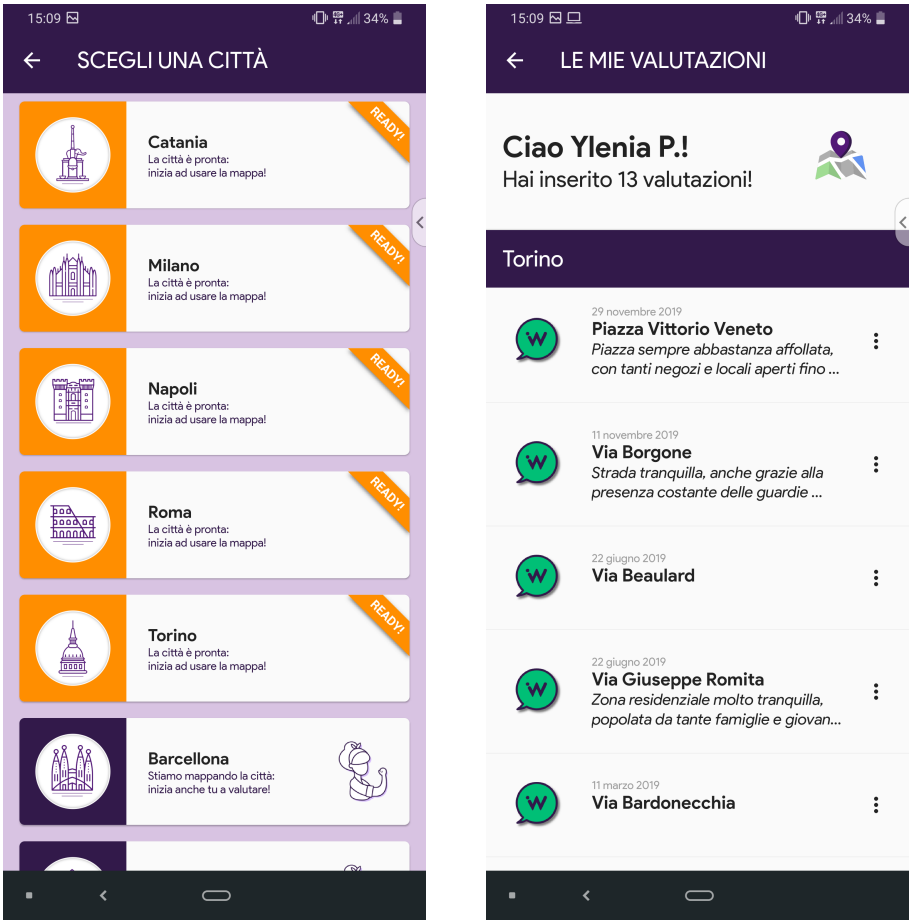
4.2.1 La mappa integrata

Wher ha creduto in Flutter fin dalla prima versione stabile, quando ancora le librerie e gli SDK a disposizione di questo nuovo framework erano scarsi; la scelta di voler usare Mapbox, piattaforma open source per la gestione delle mappe, ha messo il team davanti ad una grande sfida: quella di dover creare da zero un nuovo SDK per poter sfruttare tutte le funzioni che prima erano facilmente reperibile per la programmazione nativa. Il settore IT ha quindi creato un proprio SDK per poter integrare tutte le funzioni necessarie, come mappa, navigatore e segmenti di strade, all'interno dell'applicazione in maniera totalmente personalizzata. Il lavoro di tesi svolto e descritto nei paragrafi successivi andrà ad inserirsi all'interno di questo nuovo SDK. Poiché questo pacchetto è stato mantenuto riservato per essere utilizzato in esclusiva da Wher si è comunque deciso, vista la grande esperienza maturata nel campo delle mappe e della navigazione, di contribuire attivamente alla crescita ed allo sviluppo degli SDK pubblici di Mapbox presenti su Github, per condividere con più persone possibili le competenze acquisite durante questo lavoro.

4.3 Flutter in Wher

Wher inizialmente proponeva due app native per Android e iOS, ma con l'avvento di Flutter ha subito deciso di essere sull'onda del cambiamento, migrando l'intera piattaforma e riscrivendo tutto in Dart. La versione attuale (*v 3.4.6*) è totalmente sviluppata con l'utilizzo di Flutter, quindi è facile individuare al suo interno tutti gli elementi chiave di Dart.

La parte fondamentale di un'applicazione scritta in Dart è sicuramente lo *Scaffold* [29], cioè quel widget che fornisce l'implementazione della struttura per le schermate della nostra applicazione. Al suo interno include varie funzionalità che costruiscono visivamente la nostra schermata, per esempio l'*AppBar*, ovvero la barra superiore della pagina che riporta il titolo della schermata come quella mostrata in figura 4.2 (a), che contiene a sua volta proprietà come titolo, elevazione, luminosità etc, ed il *Drawer*, il menù laterale al quale è possibile accedere con uno swipe dal bordo sinistro dello schermo, come mostrato in figura 4.3 (b), o tramite il bottone detto "hamburger" in alto a sinistra.

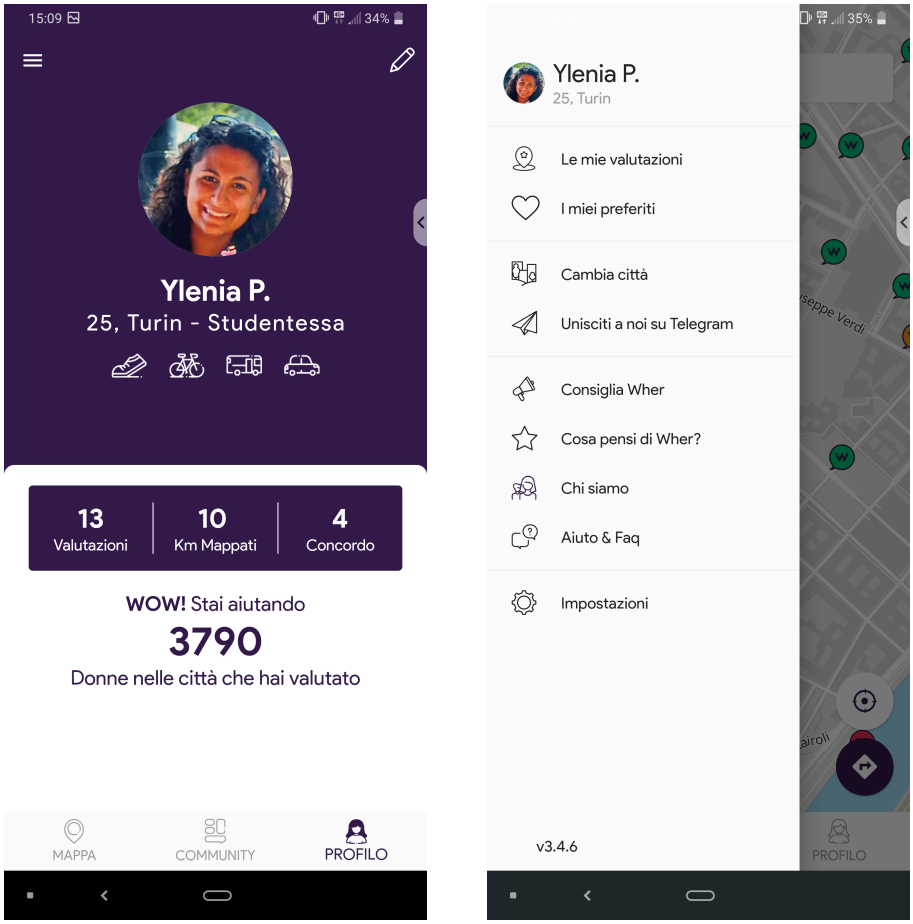


(a) Menù di scelta della città

(b) Lista valutazioni della singola utente

Figura 4.2: Utilizzo di Container e Column

Un altro elemento molto utilizzato è la *Column* [30], che permette di disporre più widget figli in verticale. Insieme alla *Row* [31], che permette la disposizione orizzontale dei widget, è uno degli elementi più utilizzati per comporre le interfacce grafiche in Flutter; un esempio di questi elementi è visibile in figura 4.3 (b), dove la lista delle impostazioni è appunto creata tramite una Column, mentre ogni singolo elemento è una Row che concatena l'immagine ed il testo.

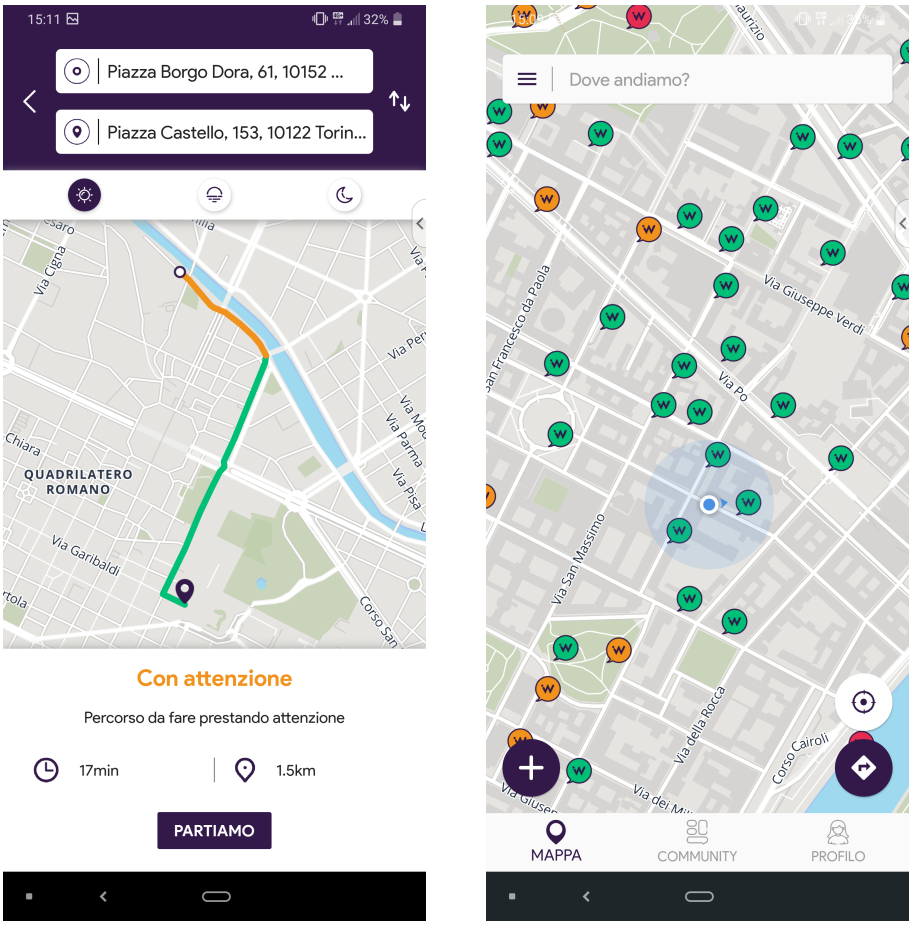


(a) Profilo utente (b) Menù laterale

Figura 4.3: Utilizzo di CustomScrollView e Scaffold

La Column però è un elemento statico, non adatto a situazioni come quella di un menù a scorrimento, come mostrato in figura 4.2 (a) e (b), quindi viene utilizzata una *ListView* [32], che permette lo scorrimento per visualizzare il contenuto non visibile del widget.

In figura 4.4 si vedono due schermate in cui viene utilizzato lo *Stack* [33], componente molto versatile che permette di sovrapporre più widget a livello visivo; nel primo caso la mappa è sovrapposta allo Scaffold sottostante, nel secondo caso invece sono i bottoni, la barra di ricerca e il menù ad essere sovrapposti alla mappa. Un ultimo componente molto utilizzato nelle applicazioni è il *CustomScrollView* [34], visibile in figura 4.3 (a), che permette di avere una parte di pagina che scorre sopra quella sottostante che rimane ferma, continuando a mostrarne solo la parte superiore.



(a) Navigatore integrato in Wher

(b) Mappa della città

Figura 4.4: Utilizzo dello Stack

4.4 Obiettivo

In Wher le valutazioni delle strade vengono inserite manualmente, selezionando ogni singolo segmento di strada da valutare. Il mio lavoro è consistito nell'ottimizzare e velocizzare questa procedura, pensando ad un sistema di tracciamento in background che acquisisca le posizioni delle utenti per poi unirle in un report che, a fine giornata, permetta di valutare velocemente, senza doverle selezionare, tutte le strade percorse durante la giornata. Questa ottimizzazione richiede inoltre la creazione di alcune schermate in Flutter per rendere fruibili i dati alle utenti, e per aggiungere la nuova modalità di acquisizione di strade da mappare.

Questa soluzione è stata realizzata attraverso strumenti di terze parti che permettono di recuperare tutte le posizioni ottenute dallo smartphone dividendole in base al movimento, per esempio camminata, bicicletta, veicoli o movimenti del solo dispositivo. La scelta è ricaduta su gli strumenti analizzati in 3.3, in quanto sono risultati i meno dispendiosi a proporre tutte le funzionalità necessarie all'acquisizione delle posizioni delle utenti. Sono stati anche analizzati la disponibilità del tool per l'integrazione in Flutter ed il supporto presente, oltre alla possibilità di poter provare il prodotto con tutte le sue funzionalità in maniera gratuita.

In base all'analisi dei fattori appena citati, si è deciso di utilizzare Transistorsoft, in quanto fornisce un SDK compatibile con Flutter da utilizzare senza bisogno di alcuna modifica, una dashboard che può essere sviluppata sul proprio server senza quindi dover cedere dati a terzi, ed una versione di prova completa installabile su un dispositivo per eseguire tutti i test necessari alla valutazione di precisione e consumo energetico. Transistorsoft inoltre è presente su Github, dove gli sviluppatori possono darsi supporto in caso di problemi, oltre che avere un team di sviluppo che è risultato essere molto efficiente nelle risposte in caso di problemi.

4.5 Strumenti utilizzati

Il servizio offerto da Transistorsoft è stato testato dal punto di vista della precisione e del consumo energetico durante il tracking in background. L'obiettivo è stato quello di trovare un buon compromesso tra energia ed

affidabilità delle misure.

Per l'analisi sono stati utilizzati:

- la dashboard fornita da Transistorsoft sviluppata su *Heroku*, un servizio web che fornisce gratuitamente l'infrastruttura di un web server. La dashboard fornisce diverse funzionalità per visualizzare i dati ed analizzarli, come la possibilità di visualizzare le posizioni di un solo dispositivo, la scelta di un lasso di tempo specifico, la visualizzazione dei marker delle posizioni raccolte e delle linee create unendo i punti individuati dai marker stessi; viene inoltre fornito un database che assembra tutte le posizioni raccolte, permettendo di visualizzarne i dettagli ma non di eseguire query per analisi più approfondite.
- *Turf*, una libreria per *Node.js* utilizzata per creare i segmenti delle strade percorse. È stato scelto questo strumento in quanto è lo stesso utilizzato a livello server per creare i segmenti delle strade su Wher.
- l'applicazione demo, scritta in Flutter, installata su un *LG G7 ThinQ* con SO Android 9.

4.6 Preparazione dei dati

Per poter analizzare la correttezza e precisione dei dati che si andranno, è stato creato un programma Javascript che, data una lista di posizioni, tenesse in considerazione solo quelle percorse a piedi (al fine della valutazione delle strade). I segmenti utilizzati sono strutture cosi composte:

```
Line = {
      line: [],
      accuracy: [],
      date: [],
      activity: []
    };
```

dove *line* contiene latitudine e longitudine della posizione rilevata, *accuracy* la precisione con la quale è avvenuta la rilevazione, *date* la data e ora della rilevazione ed *activity* il tipo di movimento rilevato.

Le posizioni vengono utilizzate per creare dei segmenti del percorso che verranno poi utilizzati per segnalare i tratti di strada percorsi e confrontarli con quelli già mappati sull'applicazione; i punti di separazione tra i diversi segmenti sono dati da due posizioni distanti più di 50 metri l'una dall'altro e vengono cosi creati:

```
var l = 0;
Line Lines[l] = {
      line: [],
      accuracy: [],
      date: [],
      activity: []
    };

for(i=0; i<array.length-1; i++) {
    var from = turf.point([array[i].lon, array[i].lat]);
    var to = turf.point([array[i+1].lon, array[i+1].lat]);

    if( array[i].activity.equals("walking")
        && array[i+1].activity.equals("walking")){

        Lines[l].line.push([array[i].lon, array[i].lat]);
        Lines[l].accuracy.push(array[i].accuracy);
        Lines[l].date.push(toTimestamp(array[i].date));
        Lines[l].activity.push(array[i].activity);

       if(turf.distance(from, to) > 0.05) {
            l++;
            Lines[l] = {
                line: [],
                accuracy: [],
                date: [],
                activity: [],
                type: []
            };
        }
    }
}
```

La variabile *array* contiene la lista di tutte le posizioni acquisite dall'applicazione demo. Con l'aiuto di due variabili di appoggio di tipo *turf.point*, si verifica che l'attività collegata alla posizione sia di tipo *walking* e che la distanza tra i punti non sia superiore ai 50 metri; se la distanza supera la soglia imposta, allora si inserisce il punto come ultimo del segmento e se ne crea uno nuovo. I segmenti creati vengono utilizzati per visualizzare graficamente i percorsi e valutarne la precisione.

4.7 Analisi delle prestazioni

Una volta creato il codice descritto precedentemente, si è analizzata la precisione della geolocalizzazione percorrendo lo stesso tratto di strada, in figura 4.5, al variare della precisione scelta per l'acquisizione dei dati.

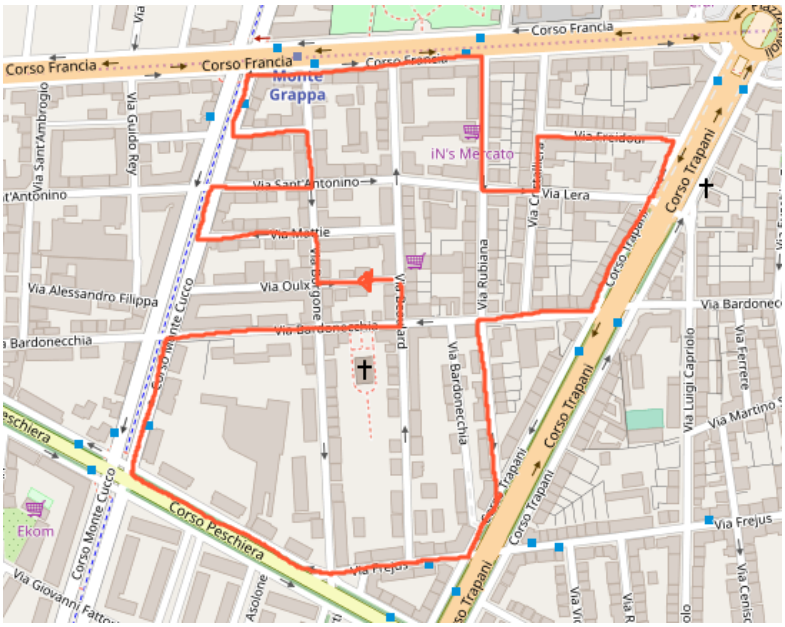


Figura 4.5: Strade percorse per il test

Le prove di acquisizione dati sono state svolte settando, oltre al livello di precisione, anche altri parametri presenti all'interno dell'applicazione demo [35]:

- *no elasticity*: l'elasticità controlla il variare del numero di posizioni rilevate all'aumentare della velocità. Si è scelto quindi di mantenere

fisso il numero di posizioni per poter valutare al meglio l'efficienza delle diverse precisioni senza influenze da parte della velocità del soggetto

- *location update interval 5000*: intervallo, in millisecondi, per l'aggiornamento della posizione. Questa opzione è disponibile solo per Android
- *activity recnognition interval 10000*: intervallo di tempo per il riconoscimento di attività in millisecondi. Si è mantenuto il valore di default nonostante con valori più alti si abbia un maggiore risparmio di batteria ma, di contro, si ha una minore precisione nel riconoscimento del tipo di attività

Durante tutte le prove il telefono era riposto in una borsa con lo schermo disattivato, senza nessun'altra applicazione aperta in background, in modo da poter valutare con più precisione il consumo di batteria del plugin.

Per ogni prova si è valutato, sulla base degli istanti di acquisizione della prima e dell'ultima posizione e sulle percentuali iniziale e finale di batteria, quanti millisecondi fossero necessari per consumare l'1% di batteria per la precisione scelta.

4.7.1 Precisione bassa

La prima prova è stata effettuata con l'impostazione della precisione al minimo. Le posizioni registrate si interrompono in alcuni angoli, vedi figura 4.6, ma i segmenti creati da Turf collegando i punti sono precisi e coprono in maniera corretta le strade percorse, ad eccezione di un punto sulla destra, in cui il segmento creato risulta essere sul viale invece che sul controviale, in figura 4.7. Questa imprecisione è trascurabile, in quanto in Wher viale e controviale sono inseriti nello stesso segmento.
Di seguito sono riportati i risultati ottenuti durante il test.

Istante prima posizione	16:04:29:456
Istante ultima posizione	16:39:46:089
Millisecondi trascorsi	2116633
Percentuale batteria iniziale	68%
Percentuale batteria finale	65%

$$\frac{2116633ms}{3\%} = 705544{,}33ms/\%$$

4.7.2 Precisione media

La seconda prova è stata effettuata con l'impostazione della precisione a livello medio. Le posizioni registrate sono molto imprecise in alcuni punti, vedi figura 4.8, infatti Turf fatica di più a creare i segmenti giusti per unire queste posizioni, in figura 4.9. Questo porta alla creazione di alcuni piccoli segmenti anche in vie non percorse, ma tali tratti sono molto corti e quindi irrilevanti, in quanto l'algoritmo inserito in Wher considera come segmento solo l'intero tratto compreso tra l'incrocio di due strade.
Di seguito sono riportati i risultati ottenuti durante il test.

Istante prima posizione	11:39:56:779
Istante ultima posizione	12:16:03:937
Millisecondi trascorsi	2167158
Percentuale batteria iniziale	39%
Percentuale batteria finale	35%

$$\frac{2167158ms}{4\%} = 541789{,}5ms/\%$$

4.7.3 Precisione alta

La terza ed ultima prova è stata effettuata con l'impostazione della precisione a livello alto. Anche in questo caso alcune posizioni risultano decentrate rispetto al percorso effettivo, vedi figura 4.10. Turf è comunque riuscito a creare i segmenti, anche se è presente un movimento errato in alto a sinistra e l'errore tra viale e controviale in due punti (in alto e a destra), in figura 4.11.
Di seguito sono riportati i risultati ottenuti durante il test.

Istante prima posizione	22:36:44:572
Istante ultima posizione	23:13:30:420
Millisecondi trascorsi	2205848
Percentuale batteria iniziale	30%
Percentuale batteria finale	26%

$$\frac{2205848ms}{4\%} = 551462ms/\%$$

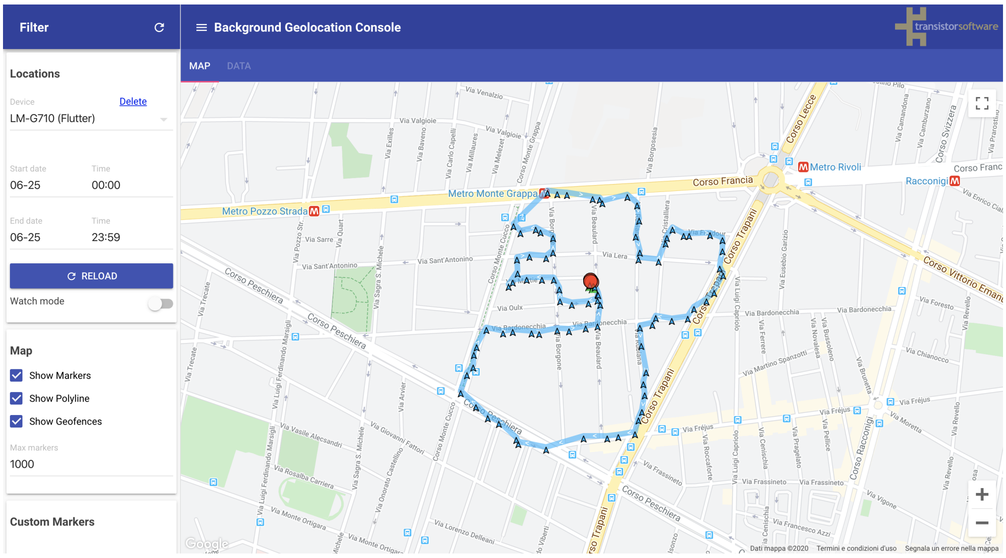


Figura 4.6: Posizioni acquisite a bassa precisione

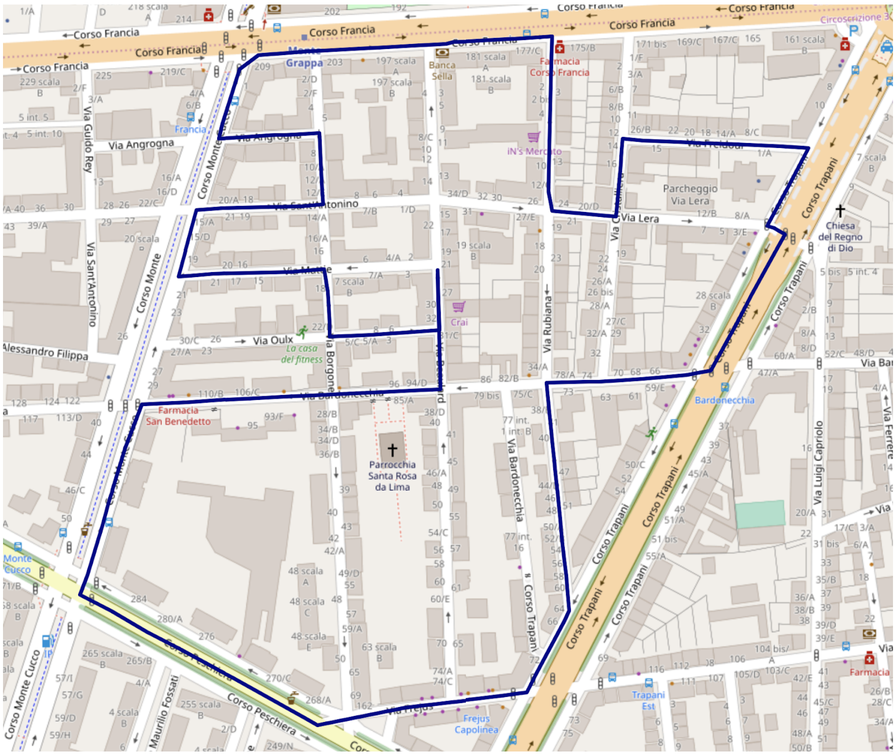


Figura 4.7: Segmenti interpolati a bassa precisione

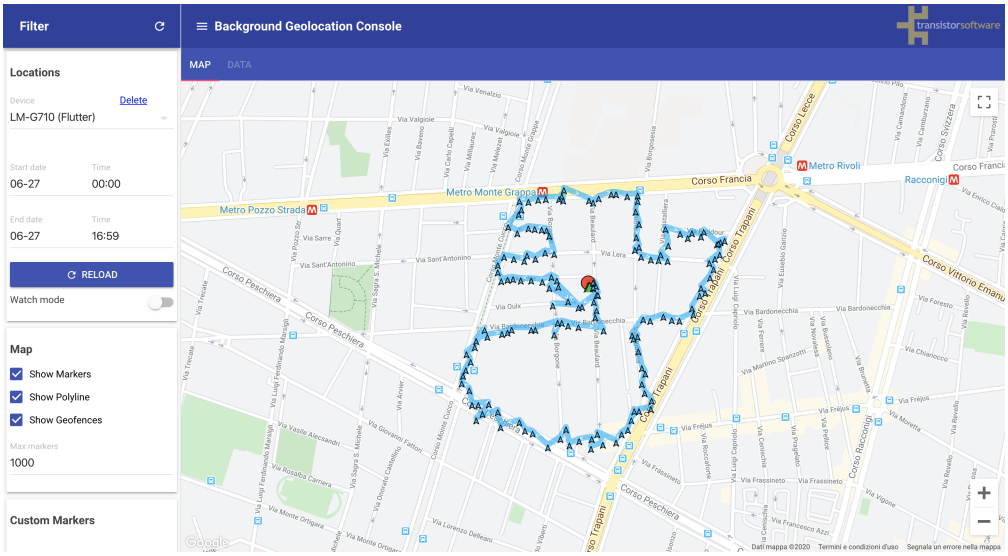


Figura 4.8: Posizioni acquisite a media precisione

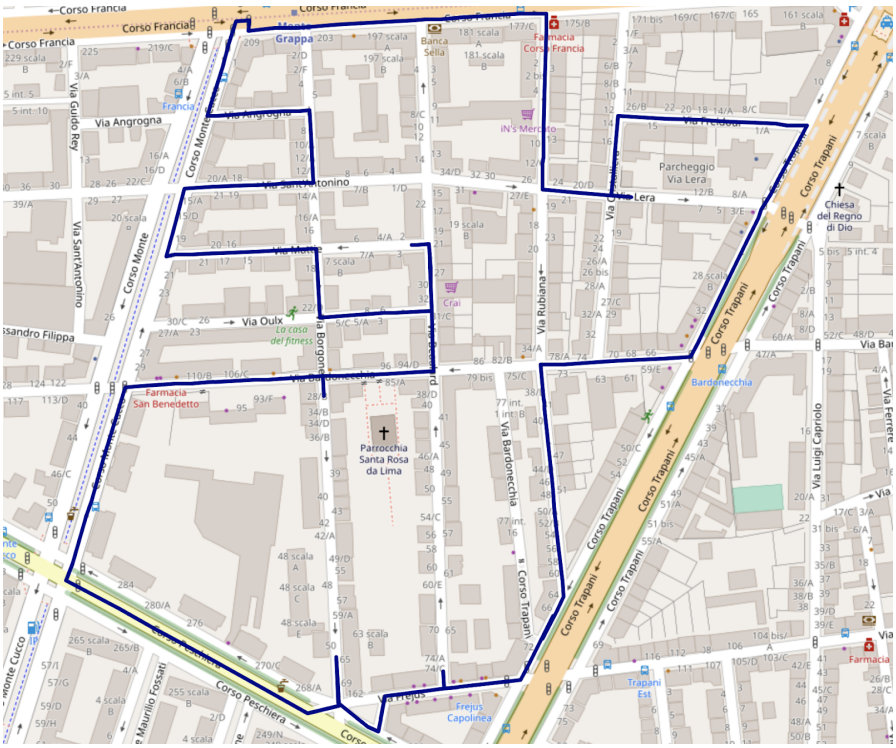


Figura 4.9: Segmenti interpolati a media precisione

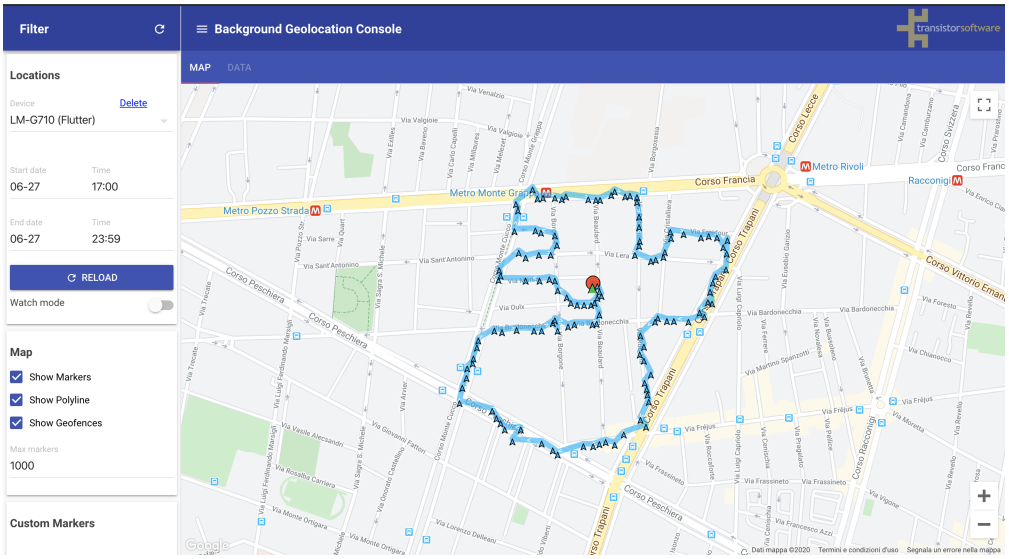


Figura 4.10: Posizioni acquisite ad alta precisione

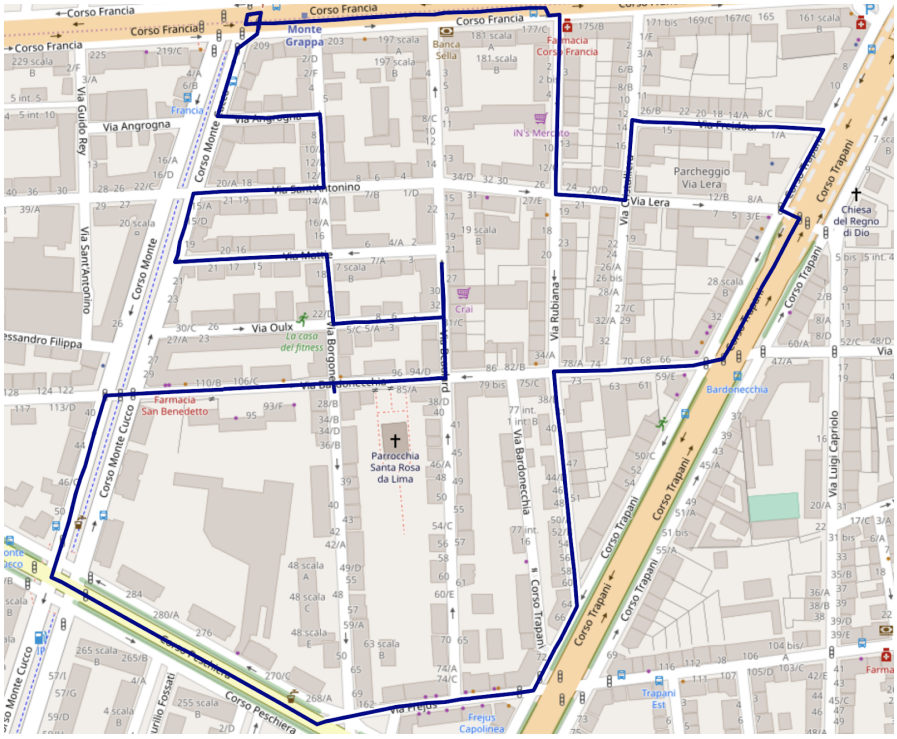


Figura 4.11: Segmenti interpolati ad alta precisione

4.8 Risultati ottenuti

Tutti i risultati ottenuti dalle prove effettuate con Transistorsoft sono stati soddisfacenti, ma è stato necessario individuare una modalità da usare in maniera definitiva all'interno dell'applicazione. In seguito a questa scelta, è stato possibile implementare in Flutter le interfacce che avrebbero ospitato i dati raccolti dal plugin di Transistorsoft.

4.8.1 Scelta del livello di precisione

Analizzando i dati ottenuti dalle prove eseguite si può notare come a livello di rapporto tempo di utilizzo su consumo di batteria la modalità più efficiente risulta essere quella a media precisione, ma allo stesso tempo questa modalità risulta la più imprecisa sui risultati ottenuti nel tracking. I risultati più precisi invece sono stati ottenuti con la modalità a bassa precisione che però, contrariamente alle aspettative, è risultata quella più dispendiosa a livello energetico. Il perfetto compromesso si trova con la modalità ad alta precisione, che si posiziona a metà sia a livello energetico che a livello di precisione.

4.8.2 Nuove schermate per l'applicazione

Il plugin è quindi stato integrato all'interno dell'applicazione con l'impostazione di alta precisione, e per gestirne i dati è stata creata una nuova schermata che permettesse di visualizzare i tratti di strada percorsi e filtrati con Transistorsoft e valutarli, oppure concordare con le valutazioni già presenti.
Come si può notare in figura 4.12, ogni utente può accedere ad una pagina che include la lista delle posizioni percorse; le posizioni sono raggruppate, oltre che per giorni, anche per intervallo di tempo in quanto, al trascorrimento di 10 minuti senza l'acquisizione di nuove posizioni in modalità camminata, viene creata una nuova mappa contenente le posizioni acquisite. Ogni scheda contiene quindi un insieme più ristretto di strade che possono essere già valutate, e quindi individuate con i colori derivanti dalle valutazioni delle altre utenti, oppure senza nessuna valutazione: nel primo caso l'utente potrà decidere se concordare con le valutazioni già presenti o inserire una nuova valutazione e commento, nel secondo caso invece l'unica opzione possibile sarà

quella dell'inserimento di una nuova valutazione; come schermata di valutazione, in entrambi i casi, viene utilizzata quella già presente sull'applicazione, con le strade da valutare già evidenziate.

Figura 4.12: Nuova schermata progettata per integrare le funzionalità del tracking in background

Le posizioni da inserire all'interno di questa schermata vengono raccolte durante le 24 ore di ogni giornata, e per ricordare all'utente di inserire le valutazioni è stata programmata una notifica giornaliera alle ore 9:00, come

mostrato in figura 4.13. Nel caso in cui l'utente non inserisca o convalidi nessuna valutazione inerente alle strade percorse per 7 giorni consecutivi, la frequenza della notifica viene modificata in settimanale.

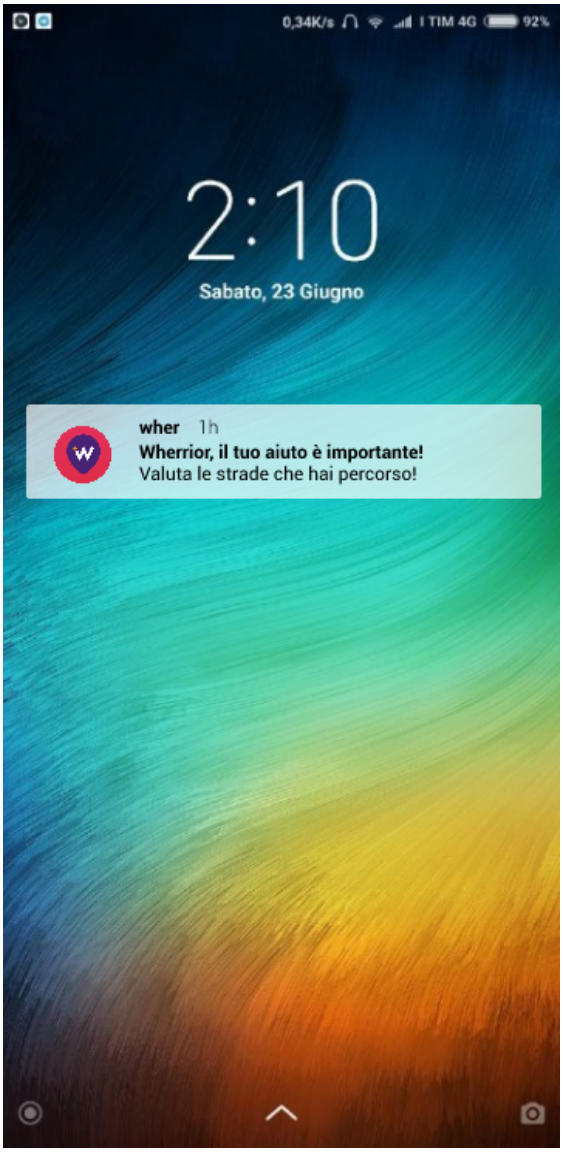


Figura 4.13: Notifica giornaliera per la valutazione delle strade percorse

4.8.3 Aggiornamento delle policy di privacy

Al momento Wher ha una privacy policy basata sulle direttive europee per la privacy, scritta e valutata alla presenza di un legale. L'aggiunta di questa nuova funzionalità porterà ovviamente delle nuove modifiche alla policy: in generale le posizioni acquisite all'interno di Wher non vengono cedute ad aziende terze e vengono utilizzate, al momento, solo per la localizzazione istantanea e la navigazione dell'utente. Con la nuova funzionalità le posizioni verranno conservate all'interno di web server proprietari, ma sempre mantenendole riservate e ad uso esclusivo di Wher.

Capitolo 5

Conclusioni

Questa tesi ha toccato diversi argomenti, come i vari paradigmi di programmazione per applicazioni mobile, la geolocalizzazione e le tecnologie che la rendono possibile. Filo rosso è stato Flutter, un framework cross-platform che permette una programmazione veloce e reattiva, grazie anche al linguaggio di programmazione Dart, caratterizzato da una curva di apprendimento molto veloce.

Flutter segue il paradigma cross-platform per lo sviluppo di applicazioni mobile e, contrariamente agli altri framework che seguono lo stesso paradigma, non utilizza i widget OEM (Original Equipment Manufacturer) o le WebView, ma utilizza un proprio motore di rendering per creare i widget, migliorando notevolmente l'esperienza utente. Tramite la composizione dei widget è possibile creare interfacce adattive per Android e iOS integrando perfettamente le funzioni native e hardware dei vari smartphone, ottenendo cosi delle applicazioni fluide ed ottimizzate in tempi molto brevi.

È stata svolta un'analisi approfondita sui vari componenti di Flutter: il framework ed i suoi widget, cuore pulsante di tutto Flutter, che permettono tramite diverse ottimizzazioni a livello interno, di velocizzare il processo di sviluppo e test; l'engine, che fornisce l'ambiente grafico appropriato alla creazione delle applicazioni; l'embedder, che fornisce l'intelligenza necessaria allo svolgimento delle azioni che rendono funzionale l'applicazione.

L'uso e le funzionalità di Flutter sono state presentate all'interno del caso di studio di Wher, un'applicazione che contiene mappe e navigatore utili alle

donne, in quanto non suggerisce la strada più veloce ma quella più sicura, basata sulle valutazioni di una vasta community di donne in giro per l'Italia e l'Europa.

L'obiettivo della tesi è stato quello di automatizzare la selezione delle strade da valutare in Wher, includendo tutte le strade percorse a piedi, tramite l'integrazione di un sistema di tracking in background all'interno dell'applicazione.

Per raggiungere questo obiettivo è stato necessario analizzare lo stato dell'arte della geolocalizzazione, focalizzandosi sui metodi utilizzati nei dispositivi mobili di ultima generazione. È stato poi analizzato lo stato dell'arte del tracking in background, che ha evidenziato la possibilità di utilizzare software ad hoc per tracciare ed analizzare i movimenti tramite l'uso combinato del GPS degli smartphone e di sensori come accelerometro e giroscopio. Questo ha portato all'analisi di strumenti di terze parti che potessero essere integrati direttamente all'interno del codice Dart, aggiungendo cosi la possibilità di monitorare tutti gli spostamenti dell'utente e di eseguire un'analisi basata sul tipo di movimento svolto.

Dopo aver individuato il software, Transistorsoft, che più risultava adeguato allo scopo, sono state svolte diverse prove di utilizzo al variare del livello di precisione, per valutare l'impatto energetico di questo plugin sull'utilizzo quotidiano dello smartphone. Infine è stato necessario creare una schermata aggiuntiva per l'applicazione, in modo da permettere all'utente di visualizzare i dati raccolti ed utilizzarli per valutare le strade percorse ogni giorno.

Tutte queste tecnologie sono converse nella creazione di una nuova funzionalità per Wher, che avrà un strumento in più per aiutare le donne a cambiare il mondo camminando senza paure per le strade delle città.

5.1 Sviluppi futuri

L'implementazione del tracking in background può fornire a Wher un grande margine di crescita sul fornire uno strumento che dia sicurezza alle donne. Il software utilizzato fornisce altre funzionalità interessanti, come la possibilità di rilevare la luminosità di un luogo tramite il sensore frontale dello smartphone e quella di creare dei confini sulle mappe che, una volta superati, inviino delle notifiche alle utenti: sarà quindi possibile creare un sistema di alert, per avvisare quando si percorre una strada segnalata poco sicura o poco illuminata, fornendo la possibilità non di precludersi il passaggio in alcune zone della città, ma lo strumento per farlo con maggiore consapevolezza ed attenzione.
Un'altra funzionalità che è stata richiesta dalle utenti, e che adesso è possibile implementare, è quella del condividere in real time la propria posizione e, incrociando i dati con quelli del navigatore, trovare vicino altre donne che stanno percorrendo le nostre stesse strade, in modo da poter camminare in compagnia e con più tranquillità.

Bibliografia

[1] URL: https://www.dieffe.tech/perche-le-app-native-sono-le-migliori/.

[2] URL: https://www.nextre.it/app-native-ibride-come-orientarsi/.

[3] URL: https://www.fastcompany.com/90442092/the-14-most-important-design-ideas-of-the-decade-according-to-the-experts.

[4] URL: https://flutter.dev/docs/resources/technical-overview.

[5] URL: https://flutter.dev/docs/resources/inside-flutter.

[6] URL: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html.

[7] URL: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html.

[8] URL: https://flutter.dev/docs/resources/inside-flutter#aggressive-composability.

[9] URL: https://flutter.dev/docs/resources/inside-flutter#sublinear-layout.

[10] URL: https://flutter.dev/docs/resources/inside-flutter#sublinear-widget-building.

[11] URL: https://flutter.dev/docs/resources/inside-flutter#building-widgets-on-demand.

[12] URL: https://flutter.dev/docs/resources/inside-flutter#linear-reconciliation.

[13] URL: https://flutter.dev/docs/resources/inside-flutter#tree-surgery.

[14] URL: https://flutter.dev/docs/resources/inside-flutter#constant-factor-optimizations.

[15] URL: https://flutter.dev/docs/resources/inside-flutter#separation-of-the-element-and-renderobject-trees.

[16] URL: https://skia.org/.

[17] URL: https://dart.dev/platforms#lightning-fast-developer-workflow-dart-vm-jit.

[18] URL: https://dart.dev/platforms#optimized-production-code-dart-aot.

[19] URL: https://github.com/flutter/flutter/wiki/The-Engine-architecture.

[20] URL: https://www.isaca.org/-/media/files/isacadp/project/isaca/articles/journal/2016/volume-5/geolocation-the-risk-and-benefits-of-a-trending-technology_joa_eng_0916.pdf.

[21] Moustafa Youssef, Ashok Agrawala e A. Shankar. «WLAN location determination via clustering and probability distributions». In: apr. 2003, pp. 143–150. ISBN: 0-7695-1893-1. DOI: 10.1109/PERCOM.2003.1192736.

[22] Moustafa Youssef, Adel Youssef, Chuck Rieger, A. Shankar e Ashok Agrawala. «PinPoint: An asynchronous time-based location determination system». In: *Copyright* 1 (lug. 2), pp. 59593–195. DOI: 10.1145/1134680.1134698.

[23] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger e James Collins. *Global Positioning System. Theory and practice*. Feb. 2001. ISBN: 3211835342. DOI: 10.1007/978-3-7091-6199-9.

[24] Mikkel Kjærgaard, Jakob Jensen, Torben Godsk e Thomas Toftkjær. «EnTracked: Energy-efficient robust position tracking for mobile devices». In: gen. 2009, pp. 221–234. DOI: 10.1145/1555816.1555839.

[25] URL: https://docs.lotadata.com/.

[26] URL: https://docs.lotadata.com/geosdk/geo-sdk/overview/payload-data.

[27] URL: https://github.com/transistorsoft/flutter_background_geolocation/wiki/Philosophy-of-Operation.

[28] URL: https://w-her.com/.

[29] URL: https://api.flutter.dev/flutter/material/Scaffold-class.html.

[30] URL: https://api.flutter.dev/flutter/widgets/Column-class.html.

[31] URL: https://api.flutter.dev/flutter/widgets/Row-class.html.

[32] URL: https://api.flutter.dev/flutter/widgets/ListView-class.html.

[33] URL: https://api.flutter.dev/flutter/widgets/Stack-class.html.

[34] URL: https://api.flutter.dev/flutter/widgets/CustomScrollView-class.html.

[35] URL: https://pub.dev/documentation/flutter_background_geolocation/latest/index.html.