

# Design and Implementation of a Context-Aware Indoor Navigation Application for Android

**Relatore:** Daniela Micucci

**Co-relatore:** Davide Ginelli

**Relazione della prova finale di:**

Lorenzo Monti - Matricola 869960

**Anno Accademico 2022-2023**



# Indice

<b>Introduzione</b>	<b>4</b>
<b>1 Funzionalità</b>	<b>5</b>
1.1 Sondaggi Sulla Navigazione Indoor . . . . .	5
1.2 Navigazione Indoor . . . . .	7
1.3 Navigazione OutDoor . . . . .	7
<b>2 Tecnologie analizzate</b>	<b>10</b>
2.1 MapsPeople . . . . .	10
2.2 MapBox . . . . .	12
2.3 MazeMaps . . . . .	13
2.4 Graphooper . . . . .	14
2.5 Navigine . . . . .	15
2.6 OpenStreetMap . . . . .	16
2.6.1 Più nel dettaglio JOSM . . . . .	19
2.6.2 Più nel dettaglio OsmDroid e MapsForge . . . . .	19
2.6.3 Più nel dettaglio OpenLevelUp . . . . .	20
2.7 Routing no tool . . . . .	21
2.8 Position no Tool . . . . .	21
2.8.1 Principali tecnologie di tracciamento . . . . .	22
2.8.2 Affluenza . . . . .	23
2.8.3 L'influenza degli errori . . . . .	23
<b>3 Proposte</b>	<b>25</b>
3.1 Navigazione esterna . . . . .	25
3.1.1 Implementazione della navigazione esterna e dei piani . . . . .	25
3.1.2 Implementazione delle mappe . . . . .	26
3.2 Routing no tool per la navigazione indoor . . . . .	30
3.2.1 Struttura generale della soluzione . . . . .	30
3.2.2 Gestione della mappa e sua visualizzazione . . . . .	30
3.2.3 Il grafo e il calcolo del percorso . . . . .	30
3.2.4 L'algoritmo di Dijkstra e sue modifiche . . . . .	30

3.2.5	Selezione dei punti e tracciamento del percorso . . . . .	33
3.2.6	Gestione delle coordinate dei punti cambiando risoluzione e gran- dezza dell'immaginee di partenza . . . . .	33
3.2.7	Gestione dei cammini personalizzati . . . . .	34
3.3	Soluzione Tridimensionale Per L'Ateneo Bicocca Basato Su Visione . .	34
3.3.1	A* . . . . .	34
3.3.2	Funzionamento Del Navigation System Di Unity . . . . .	35
3.3.3	Esempio . . . . .	36
3.4	Accedere All'Immagine Catturata Dalla Camera . . . . .	37
<b>4</b>	<b>Proposta adottata</b>	<b>39</b>
4.1	Architettura della soluzione . . . . .	39
4.2	Implementazione . . . . .	42
4.2.1	Database . . . . .	43
4.3	Progettazione interfaccia utente . . . . .	44
4.3.1	Relazioni tra i diversi componenti . . . . .	49
<b>5</b>	<b>Conclusioni e Sviluppi Futuri</b>	<b>52</b>

## Introduzione

Durante il periodo di stage, è stato dedicato tempo allo studio delle tecnologie per la creazione di un'applicazione Android che affronti il tema della navigazione indoor, risolvendo così il senso di smarrimento che spesso accompagna la ricerca delle aule. Inoltre, è stato riconosciuto il potenziale di utilità di un'applicazione del genere per coloro che presentano disabilità o problemi psicologici, offrendo loro un percorso personalizzato e adatto alle loro esigenze. L'idea alla base del lavoro svolto era quindi quella di creare un'applicazione che semplificasse e accelerasse il processo di individuazione delle aule.

Per implementare la mappa, è stato fatto uso di OpenStreetMap, un servizio di mappatura mondiale, combinato con Osmndroid, che ha permesso di visualizzare la mappa su dispositivi mobili Android. Tuttavia, l'utilizzo di queste tecnologie per la navigazione indoor è ancora in fase di sviluppo e quindi limitato. Per superare questa problematica, è stata creata una mappatura personalizzata, sviluppando anche una logica per il calcolo del percorso minimo all'interno dell'edificio.

I recenti progressi nelle tecnologie di tracciamento degli utenti, come i sensori di prossimità, le reti Wi-Fi e i Beacon Bluetooth, hanno aperto nuove opportunità per affrontare il problema della navigazione indoor. Tuttavia, nonostante gli sforzi compiuti, la navigazione indoor è ancora un'area di ricerca attiva poiché la precisione e l'affidabilità di tali sistemi devono essere migliorate per garantire un'esperienza utente soddisfacente.

Nel primo capitolo, "Funzionalità", verranno definiti i requisiti funzionali dell'applicazione e i relativi comportamenti in risposta alle azioni dell'utente, delineando così gli obiettivi prefissati.

Nel secondo capitolo, "Tecnologie Cercate", verranno esaminate le possibili opzioni che avrebbero potuto essere utilizzate, ma che, per vari motivi, non si sono rivelate idonee allo scopo, non soddisfacendo tutti i requisiti richiesti.

Nel terzo capitolo, "Proposte", verrà presentata la soluzione possibile scelta tra le alternative esposte nel secondo capitolo, mostrando delle possibili scelte per la proposta indicata.

Nel quarto capitolo, "Proposta adottata", verrà descritta l'architettura dell'applicazione, compresi i componenti interni ed esterni utilizzati, insieme alle relative tecnologie e strumenti impiegati.

Infine, nel quinto capitolo, verranno esplorate le possibili future implementazioni dell'applicazione e saranno condivise le conclusioni sulla ricerca e lo sviluppo di questa soluzione di navigazione indoor.

# 1 Funzionalità

## 1.1 Sondaggi Sulla Navigazione Indoor

Tramite un'indagine, attraverso l'utilizzo di Google Form, sono stati raccolti dati che aiutano a capire l'importanza di studiare metodi di Navigazione Indoor. Le domande sono mirate a capire quanto si sentano a proprio agio gli studenti e il personale degli atenei nell'orientarsi all'interno degli edifici della propria università. In questa sezione vengono mostrate le risposte ottenute dai sondaggi.

Quanto pensi sia facile per persone con disabilità orientarsi all'interno dell'ateneo?

84 risposte

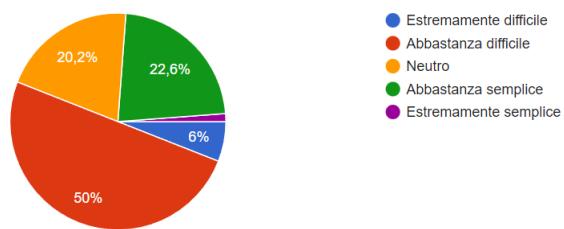


Figura 1: Persone Affette Da Disabilità

In Figura 1 è possibile vedere che il 56% dei partecipanti ritiene difficile, per le persone affette da disabilità, orientarsi all'interno dell'ateneo.

E quanto pensi sia semplice, invece, per persone affette da fobie come ad esempio la claustrofobia?

81 risposte

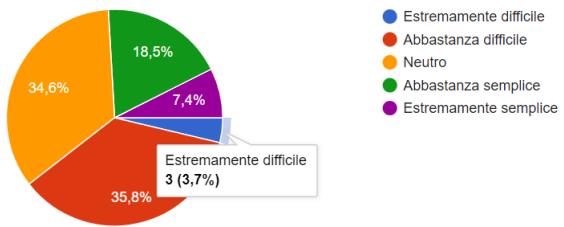


Figura 2: Persone Affette Da Fobie

In Figura 2 si può notare come quasi il 40% dei partecipanti ritenga difficile, per le persone affette da fobie (e.g. Claustrofobia), orientarsi all'interno dell'ateneo.

Credi che un'app di navigazione indoor sarebbe utile per muoversi all'interno dell'ateneo?

84 risposte

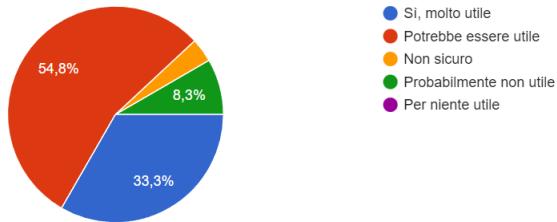


Figura 3: Utilità di Un'App di Navigazione Indoor

Dalla domanda visualizzabile in Figura 3, riguardante l'utilità di un'applicazione di Navigazione Indoor, il 54,8% delle persone ritiene che uno strumento di questo tipo potrebbe essere utile e il 33,3% delle persone lo ritiene molto utile.

Quanto sei preoccupato per la privacy e la sicurezza dei tuoi dati se dovessi utilizzare un'app di navigazione indoor? □

84 risposte

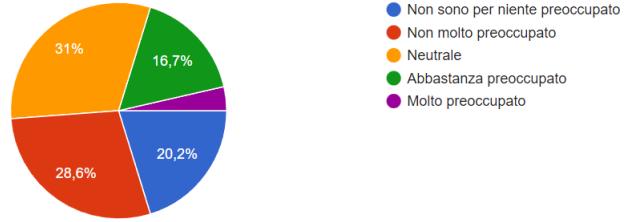


Figura 4: Favore Alla Condivisione della Posizione

L'ultimo grafico a torta visualizzabile è quello in Figura 4, è stato richiesto di rispondere con un certo livello di preoccupazione relativo alla condivisione della posizione o dei propri dati. In totale, solo il 20,3% dei partecipanti è restio nel condividere i propri dati.

Sono state poste anche domande a risposta aperta relative ai disagi percepiti mentre si è all'interno dell'ateneo e quali funzionalità sarebbero utili in un'applicazione di Navigazione Indoor. I disagi principali sono: gli ascensori non funzionanti, l'affollamento di persone e la numerazione o nomenclatura complessa di alcune aule e uffici. Le funzionalità più richieste sono invece: ottenimento di indicazioni sui diversi percorsi disponibili, poter scegliere il percorso più adatto alle proprie esigenze, calcolo della distanza di percorso per arrivare alla destinazione scelta, informazioni sull'affollamento, segnalazione di ascensori o scale mobili non funzionanti e infine una modalità di visione con realtà aumentata che indichi il giusto percorso.

## 1.2 Navigazione Indoor

Obiettivo: L'obiettivo dell'applicazione è fornire un sistema di navigazione interna, utile alla ricerca e al tracciamento di un percorso verso aule, sale studio e qualsiasi altro luogo interno agli edifici.

1. Navigazione interna: Il sistema di navigazione interna consentirà agli utenti di inserire una posizione di partenza e una di destinazione andando successivamente a tracciare il percorso più breve. Il sistema sarà in grado di tracciare percorsi personalizzati, ovvero, in grado di evitare aree non percorribili da un utente.

- input: Due stringhe di testo rappresentanti la partenza e l'arrivo.
- Output: Percorso stampato sulla mappa
- Scenari Alternativi:
  - Destinazione o partenza non esistenti: Il percorso non viene stampato
  - L'utente seleziona un'impostazione per cui evita certi nodi (scale). In questo caso andrà a mostrarti il path senza i nodi da evitare.
  - Il percorso attraversa più piani: L'utente potrà visualizzare i differenti piani dell'edificio con il continuo del percorso selezionato

2. Navigazione a step: L'utente una volta selezionati i punti di partenza e destinazione, può scegliere di procedere con una navigazione a step, che mostrerà il percorso passo a passo.

- input: Due stringhe di testo rappresentanti la partenza e l'arrivo. Preferenze dell'utente dei percorsi da evitare (ad esempio senza scale), e click sull'impostazione show step.
- Output: Percorso stampato con la possibilità di mostrare il percorso step-by-step.

3. Cambio piano: L'utente può visualizzare a sua scelta i differenti piani dell'edificio

- input: Il piano da visualizzare.
- Output: "Stampa" del piano desiderato.
- Scenari Alternativi:
  - L'utente seleziona un piano non esistente: il piano non viene mostrato restando sull'attuale visualizzazione.

## 1.3 Navigazione OutDoor

Obiettivo: L'obiettivo principale dell'applicazione è fornire una mappa dell'università che possa essere utilizzata per ottenere informazioni utili principalmente per la navigazione all'aperto tra gli edifici, ma anche, in modo parziale, all'interno degli edifici stessi, includendo le aule. L'applicazione mostrerà i diversi piani dell'edificio con le relative aule ricercate dall'utente, al fine di facilitarne la

navigazione. Inoltre, l'applicazione offrirà la possibilità di passare alla modalità di navigazione indoor nel caso in cui l'utente desideri un aiuto aggiuntivo per spostarsi all'interno degli edifici.

1. Descrizione della Navigazione esterna: Il nostro sistema di navigazione esterna consentirà agli utenti di inserire la loro posizione di partenza e la destinazione desiderata, che successivamente, mostrerà diverse opzioni selezionabili e calcolerà il percorso più breve, visualizzandolo sulla mappa. Il calcolo del percorso sarà basato sulla modalità di trasporto selezionata dall'utente, tra cui "a piedi", "in macchina" o "in bicicletta". Inoltre, il sistema sarà in grado di calcolare un nuovo percorso nel caso in cui l'utente si allontani dalla strada corrente, fornendo così indicazioni aggiornate per raggiungere la destinazione correttamente.
  - Input: Caselle di testo sottoforma per la partenza, la destinazione ed inoltre il tag ("piedi", "macchina", "bici").
  - Output: Path stampato sulla mappa e puntatore sulla destinazione
  - Scenari Alternativi:
    - Se l'unico valore presente è nella casella destinazione, l'applicazione mostrerà solo la posizione del luogo selezionato tramite un puntatore.
    - Se il valore nella casella di destinazione è un aula, l'applicazione andrà a mostrare la posizione e il piano ad essa associati.
2. Descrizione dei piani e delle classi, il sistema sarà in grado di visualizzare bottoni pertinenti al cambio di piano in base allo zoom e alla posizione dell'utente. Questi bottoni mostreranno il numero e le immagini dei vari piani disponibili. Se l'utente ha selezionato una specifica aula durante la scelta della navigazione esterna, il sistema terrà conto sia del piano che dell'aula sulla mappa, fornendo così indicazioni precise riguardanti la navigazione outdoor tra edifici, e un puntatore che evidenza l'aula selezionata.
  - Input: Un edificio dell'ateneo, livello di zoom e piano dell'edificio selezionato
  - Output: Piantina riguardante il piano cliccato
  - Scenari Alternativi: L'utente avrà la possibilità di passare alla navigazione indoor, qualora fosse necessaria, tramite un apposito bottone.
3. Descrizione GPS: il sistema dovrà mostrare la posizione dell'utente in base ai permessi da esso abilitati, e in base all'attivazione del gps tramite l'icona apposita e il bottone dell'applicazione apposito
  - Input: Abilitazione dei permessi e attivazione gps.
  - Output: Posizione indicativa sulla mappa
  - Scenari Alternativi:

- Se i permessi sono abilitati ma il gps è inattivo allora il sistema non mostrerà la posizione dell’utente
  - Nel caso in cui permessi necessari non siano stati accettati dell’utente, il sistema richiederà di abilitarli ogni qualvolta si cerca di attivare il gps.
4. Mappa aggiornata: il sistema dovrà mostrare una mappa aggiornata in base ai permessi da esso abilitati e in base all’attivazione del wi fi, così da mostrare eventuali lavori all’interno dell’atneo e modificare eventuali percorsi.
- Input: Abilitazione dei permessi e attivazione del wi-fi/rete mobile
  - Output: mappa aggiornata
  - Scenari Alternativi:
    - Se i permessi sono abilitati ma il wi-fi è inattivo allora il sistema mostrerà una mappa non aggiornata relativa solo all’atneo, senza la possibilità di routing.
    - Nel caso in cui permessi necessari non siano stati accettati dell’utente, il sistema si comporterà nella stessa maniera del primo scenario alternativo.

## 2 Tecnologie analizzate

Con l'obiettivo di esplorare diverse soluzioni e approcci per raggiungere gli obiettivi sia nell'ambito della navigazione indoor che di quella outdoor, è stato dedicato impegno nella ricerca di varie tecnologie che potessero supportare l'applicazione. È stato dedicato tempo alla lettura di articoli, studi e approfondimenti al fine di identificare le opzioni più promettenti.

Verrà presentata una panoramica delle diverse tecnologie esaminate, valutando i loro vantaggi e svantaggi. Saranno approfondite le ragioni che hanno portato a selezionare alcune tecnologie rispetto ad altre, spiegando le motivazioni dietro le scelte effettuate.

Attraverso questa ricerca, si mira a fornire una visione ampia delle varie tecnologie considerate e degli approcci esplorati per affrontare le sfide della navigazione indoor e outdoor. Ciò permetterà di comprendere meglio le alternative che sono state valutate nel contesto del progetto e degli obiettivi.

### 2.1 MapsPeople

MapsPeople è un provider di servizi di navigazione indoor, il quale offre soluzioni di mappatura e navigazione per ambienti interni ma non solo, poiché appoggiandosi a googleMaps è in grado di fornire anche servizi di routing esterno. Inoltre, MapsPeople, consente agli utenti di accedere a mappe interattive, individuare posizioni specifiche e trovare il percorso più breve utilizzando la funzionalità aggiuntiva delle indicazioni Passo-Passo. Con l'aggiunta di tecnologie di posizionamento, come i beacon e sensori di prossimità, può offrire un'esperienza più immersiva e semplice. MapsPeople non solo permette di creare applicazioni per android, ma anche per ios e web app. Per un eventuale comprensione più approfondita del suo utilizzo consiglio "MapsIndoor" sul playstore e ios.[6]

- Funzionamento: per prima cosa bisogna creare la propria mappa, e questo si può fare tramite il loro CMS/SDK, Figura 5, il quale ci permette di creare tramite dei tag: piani, path, muri, stanze, tavoli...

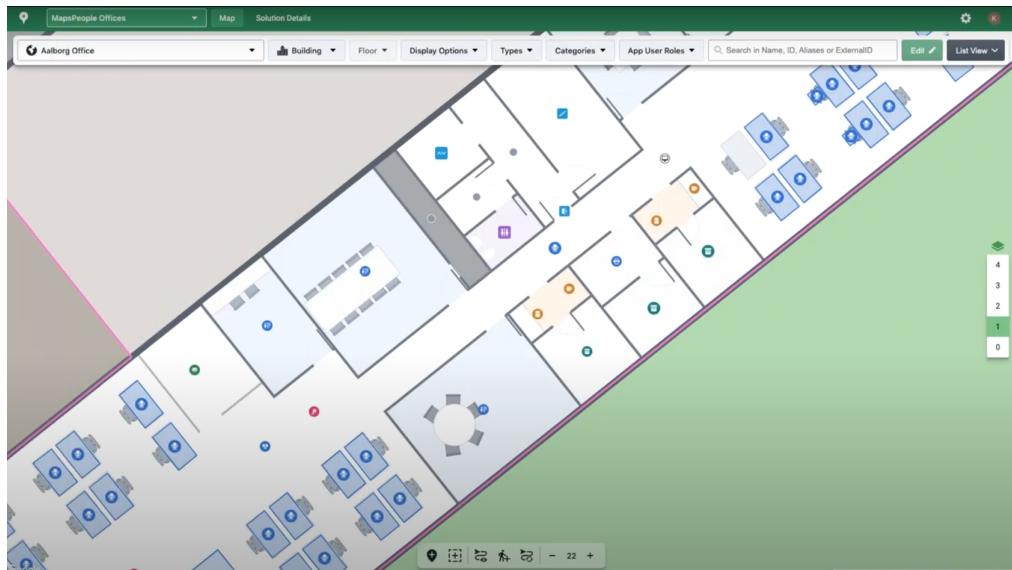


Figura 5: CMS MapsPeople

Fatto questo sarà possibile creare la propria applicazione, tramite le linee guida fornite da MapsPeople, come in Figura 6 e 7.

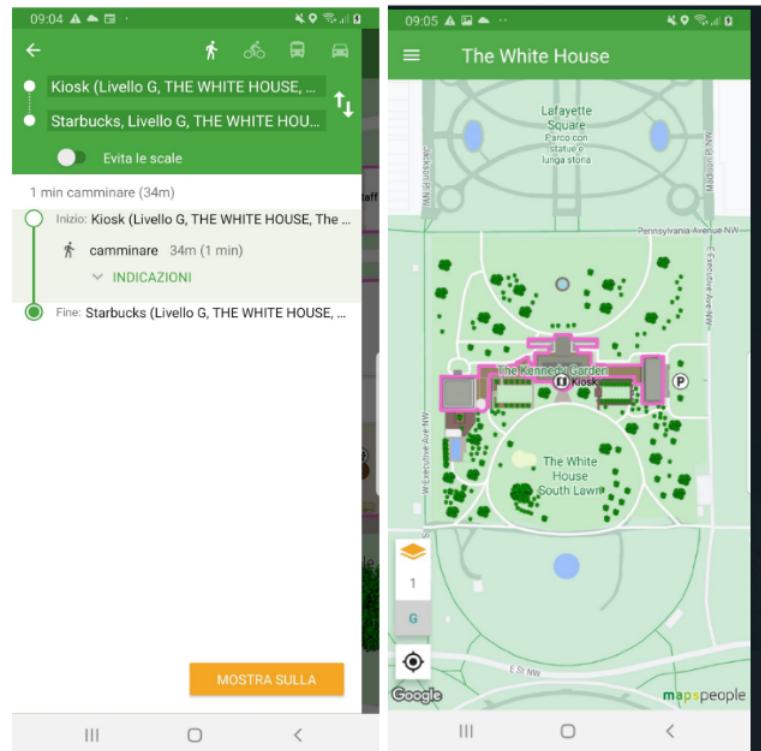


Figura 6: Demo MapsPeople percorso

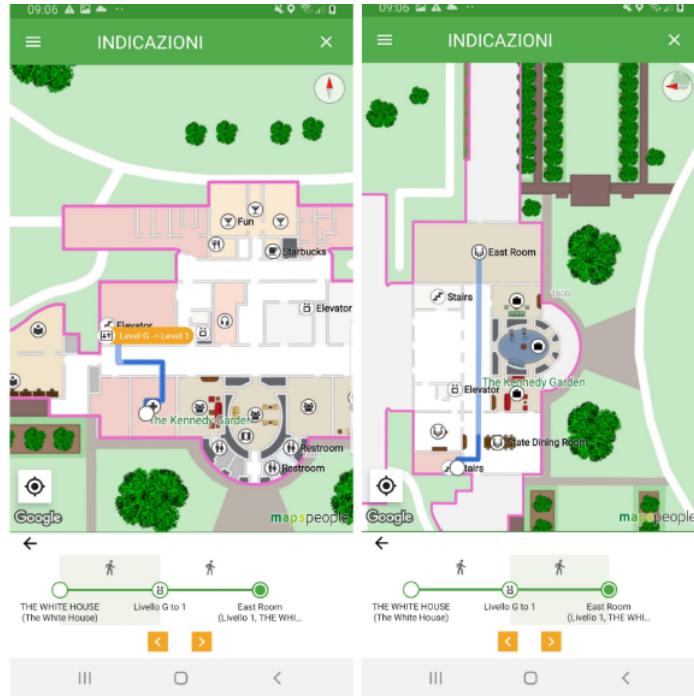


Figura 7: Demo MapsPeople routing

Per aggiungere i piani degli edifici bisognerà avere un API KEY, riferita al piano stesso, il quale consentirà di mostrare le mappe.

- Vantaggi: pratica, facilmente utilizzabile, docs completa e supporto tecnico molto disponibile per eventuali dubbi.
- Svantaggi: non è stato possibile testarla a causa dell'assenza di una demo che consenta di verificare tutti gli obiettivi richiesti.

Tag riguardanti percorsi personalizzati, potrebbero essere non presenti.

## 2.2 MapBox

MapBox è un'azienda che fornisce soluzioni riguardanti la mappatura, geolocalizzazione e visualizzazione dei dati geospatiali. Offre una piattaforma che garantisce funzioni avanzate e personalizzazione per la creazione di applicazioni riguardanti diversi settori: trasporto, immobiliare, turismo, logistica e altro.

- Mapbox Studio: un editor di mappe online che consente di personalizzare l'aspetto delle mappe, modificare lo stile dei dati geospatiali e aggiungere elementi visivi come etichette, icone e colori.

- Mapbox Maps SDKs: SDK (Software Development Kit) disponibili per diverse piattaforme, tra cui iOS, Android e Web, che consentono agli sviluppatori di integrare mappe interattive e funzionalità di navigazione nelle proprie applicazioni.
- Mapbox Geocoding API: un servizio di geocodifica che consente di convertire indirizzi o luoghi in coordinate geografiche (latitudine e longitudine) e viceversa. Questo servizio facilita la ricerca e la visualizzazione di luoghi specifici sulla mappa.
- Mapbox Directions API: un servizio di calcolo del percorso che consente di ottenere indicazioni stradali tra due o più punti sulla mappa. Questo è utile per implementare funzionalità di navigazione e tracciamento dei percorsi nelle applicazioni.
- Mapbox Data Services: una serie di servizi per l'acquisizione, l'elaborazione e la gestione dei dati geospaziali. Questi servizi includono la creazione di mappe personalizzate, l'analisi dei dati, la visualizzazione dei dati geospaziali in tempo reale e altro ancora.

Per un riferimento alle funzionalità, lascio il link per la demo di [Mapbox](#)[5]

- Vantaggio: è la soluzione con il maggior numero di funzionalità.
- Svantaggio: cercando inizialmente una soluzione free l'abbiamo scartata, poiché, alla fase di registrazione, andrà a richiedere il metodo di pagamento con cui si andrà a pagare, qualora si eccederà la domanda effettiva.

### 2.3 MazeMaps

MazeMap è una piattaforma di mappatura e navigazione indoor, la quale fornisce una navigazione interna complessa tramite le sue varie funzionalità, dove queste funzionalità possono essere: beacon, sensori di prossimità e sensori di calore. Consente agli utenti di avere mappe interattive (posizione in tempo reale), individuare posizioni specifiche, quindi trovare il path minimo con la funzionalità aggiunta per le indicazioni Passo-Passo e la pianificazione di percorsi. Una funzionalità molto interessante è la versatilità del progetto stesso, infatti quest'ultimo può essere integrato con altre app e sistemi, semplificando e ottimizzando così l'esperienza all'interno di grandi strutture. Un'altra funzionalità che mette a disposizione MazeMap è la navigazione con una mappa in 3D, rendendo la piantina ancora più chiara. Per un eventuale comprensione più approfondita del suo funzionamento consiglio di guardare il video riguardante la mappatura del [Salone del mobile](#)

- Funzionamento: per prima cosa bisogna creare la propria mappa, e questo si può fare tramite il loro CMS/SDK il quale permette di creare tramite dei tag: piani, path, muri, stanze, tavoli... Dopo questo si manderà la piantina finita a MazeMap, dove i loro addetti informatica creeranno l'applicazione in base all'esigenze del cliente.

- Vantaggio: molte funzionalità (utili per i nostri vari problemi che dobbiamo tenere in considerazione), ben fatta, trasportabile su diverse piattaforme, versatile.
- Svantaggio: come per MapsPeople non esiste una prova gratuita e quindi una demo, non potendo così andarla a testare. Un secondo problema riguarda la creazione che avviene, solo, tramite l'azienda stessa.

Considerazioni: MazeMap è un prodotto di grande valore in sé, offrendo numerose funzioni utilizzabili con diversi tipi di tag specificamente creati. Ad esempio, potrebbe essere utile per la misurazione del calore, che potrebbe contribuire a risolvere molti problemi, come quelli di natura sociale. Tuttavia, il problema principale è che l'applicazione è a pagamento anziché gratuita, come ci aspettavamo. Inoltre, l'applicazione verrà sviluppata direttamente dall'azienda, perdendo così di personalizzazione.

## 2.4 Graphooper

GraphHopper è un framework open-source per il calcolo dei percorsi e la navigazione che offre funzionalità di routing avanzate per una vasta gamma di scenari, inclusi percorsi stradali, percorsi per pedoni e percorsi indoor.

- GraphHopper Directions API: con l'aiuto delle sue API si può integrare un routing plain da A to B, navigazione passo per passo, con calcoli isocrone (determinare le aree raggiungibili entro un determinato intervallo di tempo o di distanza da un punto di origine). Connnettendo la tua applicazione con i suoi algoritmi
- Routing and Navigation: fornisce i tempi di percorrenza, le distanze, le istruzioni di svolta e la geometria del percorso in base al profilo di viaggio selezionato, che potrai visualizzare su qualsiasi mappa desideri.
- Route Optimization: Assegna percorsi ai veicoli in modo che i costi totali di trasporto siano minimizzati e può prendere in considerazione un numero arbitrario di vincoli specifici del business come finestre di tempo, competenze dei conducenti, capacità dei veicoli e altro ancora. (Problema Commesso Viaggiatore)
- Geocoding: Ottieni un indirizzo per la tua posizione geografica attuale o ottieni le coordinate per un indirizzo, chiamato "geocodifica inversa". Supportiamo anche servizi di geocodifica esterni che completano la qualità della soluzione predefinita.
- Time-dependent Route Optimization: Considera le variazioni dei tempi di percorrenza durante le ore di punta e fuori dalle ore di punta. Considera anche informazioni dettagliate sul traffico storico da TomTom® all'interno dell'ottimizzazione. Per ogni strada e fascia oraria, utilizziamo dati di tempo di percorrenza aggregati da milioni di punti dati raccolti in modo

anonimo. Ciò consente di prevedere i tempi di percorrenza in modo molto più preciso, ottenendo piani di percorso dei veicoli più realistici ed economicamente vantaggiosi.

- Advanced routing tools like the Matrix API: Calcolo dei cluster - suddivide le posizioni inviate in gruppi definiti, Corrispondenza della mappa - converte le tue tracce GPS in percorsi allineati alle strade, Calcolo delle matrici - fornisce matrici di tempo e distanza molto veloci, Calcolo delle isocrone - fornisce l'area che può essere raggiunta da una posizione specifica in un determinato intervallo di tempo, basandosi sul metodo di viaggio (ad esempio, quanto lontano puoi viaggiare in bicicletta in 30 minuti).[1] [15]

## 2.5 Navigine

Navigine è una società che si occupa nello sviluppo di soluzioni di navigazione e localizzazione indoor. In questo caso offre una piattaforma completa e gratuita per l'implementazione di piantine e la costruzione di elementi con i relativi tag, andando tramite i loro algoritmi (Dijkstra e A\*) a calcolare il percorso minimo tra due elementi. Può andare ad utilizzare la tecnologia del tracciamento della posizione in tempo reale all'interno dell'edificio tramite: beacon, tecnologia RFID, sensori e dispositivi mobili. Offre anche la possibilità di integrarsi con altre tecnologie e sistemi di terze parti, consentendo un implementazione flessibile e su misura per le esigenze di ogni cliente.

- Funzionamento: per prima cosa andremo a creare una mappa e salvarla in un formato compatibile, figura 8



Figura 8: Mappa jpg usata

Dopo aver creato la mappa, dovremo andare sul sito di [Navigine](#) dove ci chiederà di cercare l'edificio, all'interno di una mappa simil Google Maps, al quale vogliamo aggiungere il piano effettivo.

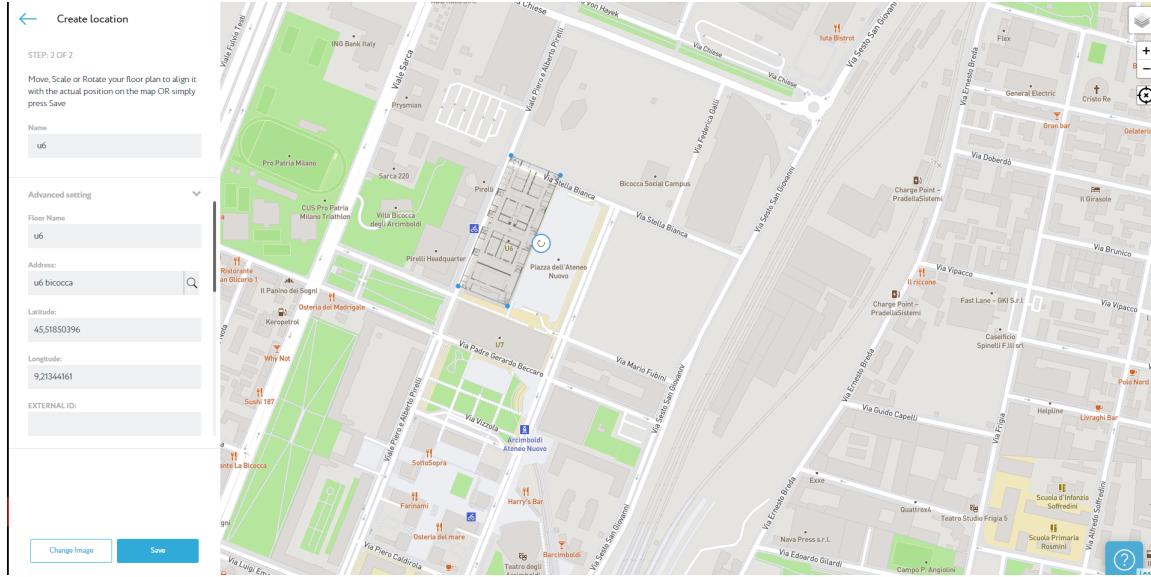


Figura 9: Posizionamento dell’immagine

Dopo aver aggiunto il piano, come mostrato nella Figura 9, possiamo andarcì a lavorare sopra la piantina e aggiungerci tutti i rispettivi tag utili per la creazione del path e dell’edificio.

Questa mappa verrà salvata sul server di Navigine, così da poter essere utilizzata e aggiornata su qualsiasi dispositivo.

Ogni utente avrà un codice hash che servirà per andare a ricevere le mappe ad esso associato.

Navigine fornisce già una demo all’interno del playstore e sul loro github, permettendo così di vedere come la mappa verrà renderizzata.

- **Vantaggi:** ben fatta, docs completa, funzionalità presenti, app già utilizzabile.
- **Svantaggi:**

Considerazioni: Navigine è una scelta molto valida se si vuole andare a creare un applicazione indoor. Nel nostro caso va a racchiudere tutti gli obiettivi prefissati del capitolo 1.

Grazie alla sua docs si poteva andare ad implementare l’applicazione oppure utilizzare direttamente la loro demo offerta che va a implementare tutte le funzionalità per il corretto utilizzo.

## 2.6 OpenStreetMap

OpenStreetMap è un progetto il cui scopo è creare una mappa del mondo libera ed editabile da chiunque. Infatti, tramite una piattaforma (JOSM) o openStreetMap stesso è possibile modificare strade, marciapiedi, edifici ed altro tramite dei tag. Un’altra funzione integrata in questa tecnologia, è il calcolo del percorso minimo e la presenza di punti di interesse che possono essere chiamati tramite le API.

Qui di seguito viene spiegato come poter utilizzare OpenStreetMap, ad esempio per l'aggiunta di Tag per la gestione dei differenti piani di un edificio. [3] ”OSM indoor.”

- Funzionamento: Accedendo alla pagina di OpenStreetMap ”Mappa OSM”, è possibile scaricare il file riguardante il tileset del mondo nel formato ”\*.osm”.

Presi la mappa si procede a scaricare JOSM, un tool esterno, che permetterà di modificarne il contenuto più in dettaglio rispetto al semplice editor online di OSM stesso, con ad esempio la possibilità di aggiungere il tag ”level=\*” per ogni elemento indoor: corridoio, stanza, porta, finestra, ect.. come mostrato nella Figura 10.

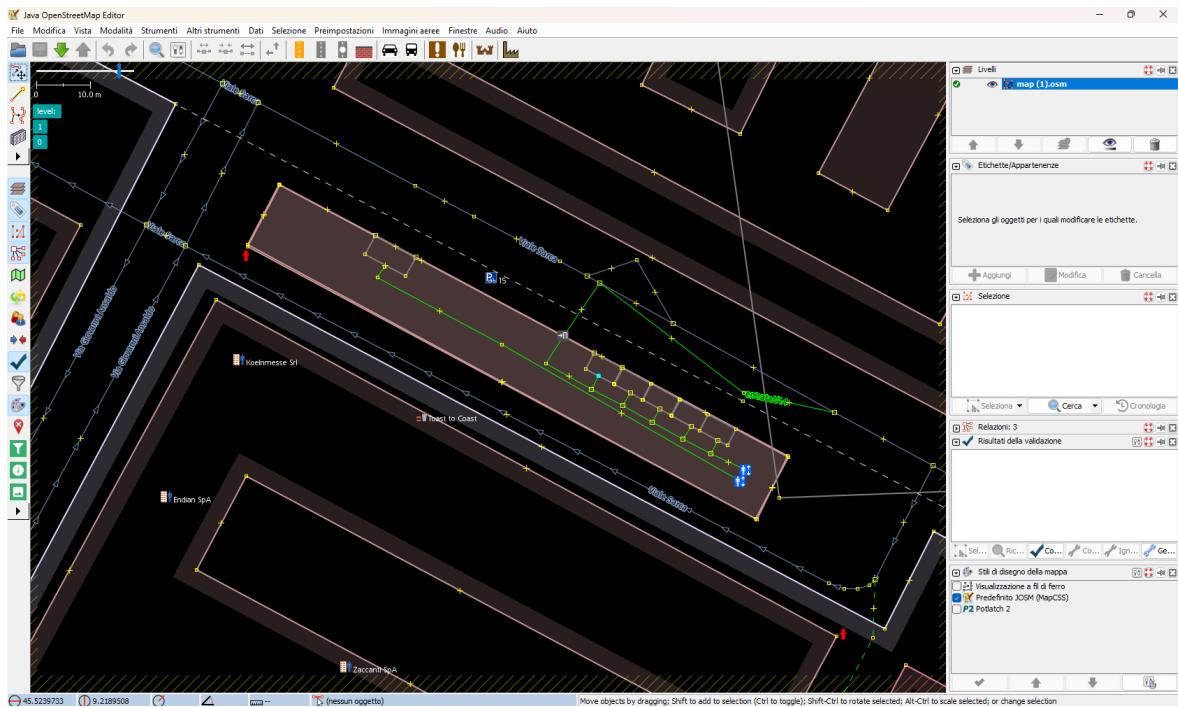


Figura 10: [Editor jsm](#)

Dopo aver fatto tutte le modifiche necessarie, tramite un tool esterno, [OpenLevelUp](#), sarà possibile vedere se tutti gli elementi modificati contengono i tag precedentemente definiti, osservando così se l'edificio è stato modificato in maniera corretta, come in Figura 11 e 12.

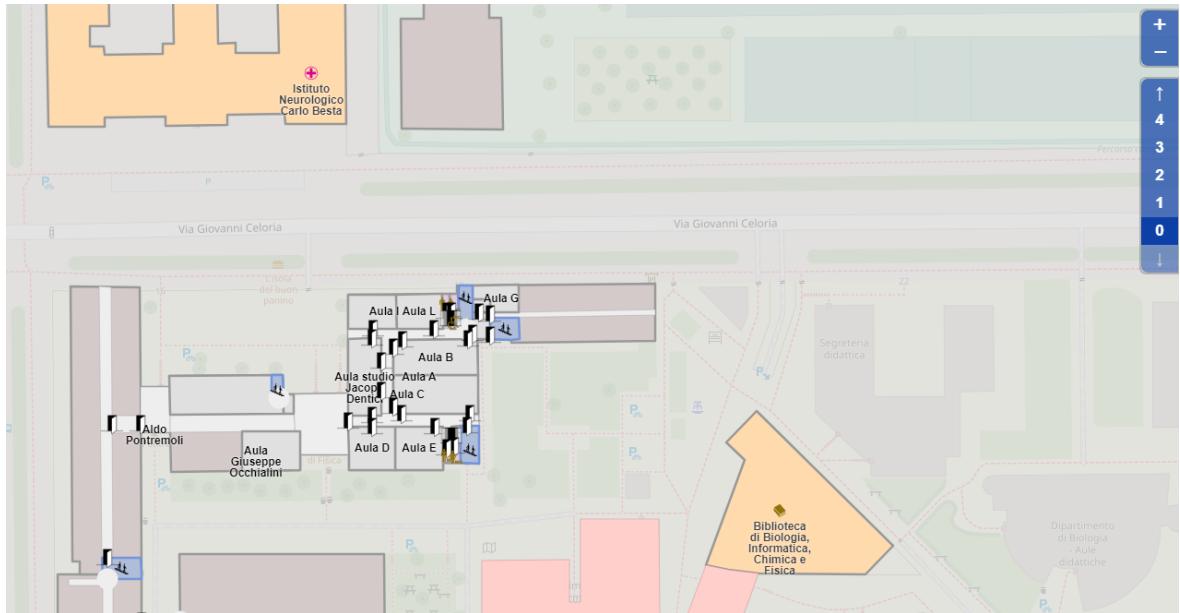


Figura 11: OpenLevelUp piano 0

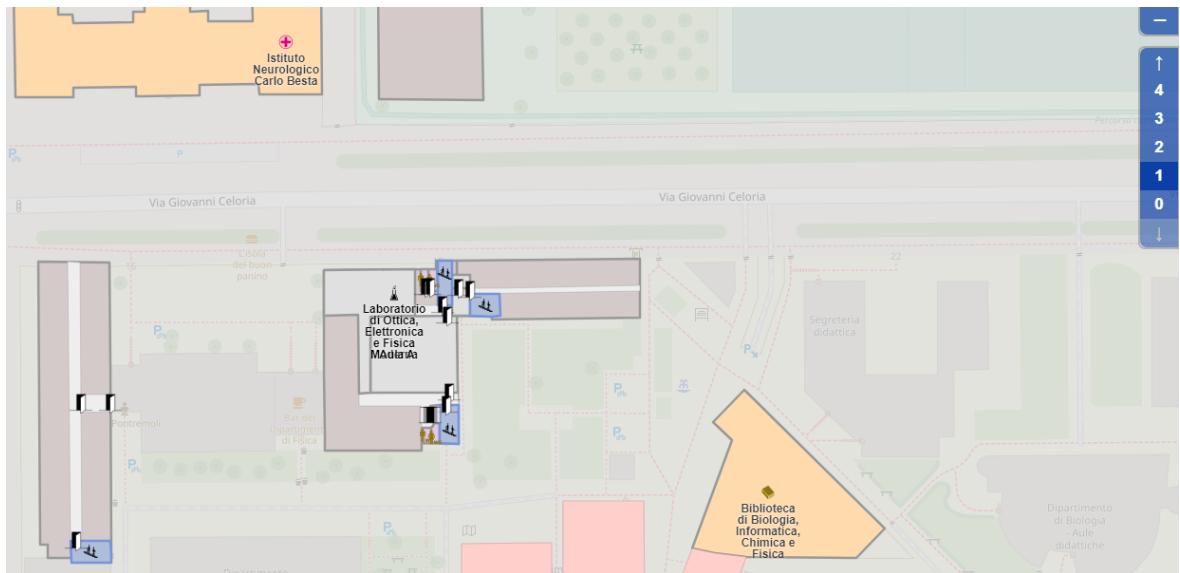


Figura 12: OpenLevelUp piano 1

Dopo aver constatato che tutto risulta conforme alle modifiche apportate, si potrà utilizzare la libreria "OsmDroid", la quale permetterà di implementare i dati provenienti da OSM ad esempio da una applicazione Android. Questa libreria fornisce tutte le funzionalità necessarie per integrare correttamente i dati OSM all'interno di un'applicazione.

- Vantaggio: Piattaforma solida con docs completa e ampie funzionalità, il tutto appoggiato da una community molto presente.
- Svantaggio: Rendering indoor non effettivo.

Uno dei principali problemi riguardanti OpenStreetMap riguarda il rendering, in quanto anche se esistono dei tag specifici per l'indoor mapping, essi non vengono renderizzati correttamente e si sovrappongono l'uno all'altro. Per il rendering, spesso si fanno affidamento a programmi esterni come OpenLevelUp.

L'indoor mapping in OpenStreetMap rappresenta un campo aperto in cui non esiste ancora una singola soluzione efficiente per la sua esecuzione. Nel testare le funzionalità, è stato fatto utilizzo della libreria OsmDroid come una delle opzioni disponibili per implementare l'indoor mapping.

### 2.6.1 Più nel dettaglio JOSM

[JOSM](#) è un editor desktop per la modifica e pubblicazione di mappe riguardanti OpenStreetMap. JOSM offre un interfaccia utente più intuitiva e ricca di funzionalità per la modifica della mappa.

- Visualizzazione dei dati di OpenStreetMap: JOSM consente agli utenti di visualizzare i dati di OSM su una mappa interattiva. È possibile zoomare, spostare e selezionare aree specifiche per la modifica.
- Modifica dei dati: Gli utenti possono creare nuovi elementi geografici, come strade, edifici, punti di interesse, e modificare quelli esistenti. JOSM fornisce strumenti per disegnare, modificare, spostare e cancellare elementi sulla mappa.
- Gestione dei tag: JOSM permette di aggiungere e modificare i tag associati agli elementi di OpenStreetMap. I tag forniscono informazioni aggiuntive sugli elementi geografici, come il tipo di strada, il nome di un edificio, ecc.
- Supporto per importare dati esterni: JOSM consente di importare dati geografici da diverse fonti esterne, ad esempio file GPX, immagini satellitari, dati GIS, etc..
- Controllo di qualità dei dati: JOSM fornisce strumenti per verificare e migliorare la qualità dei dati di OpenStreetMap. Gli utenti possono eseguire controlli di validità, correggere errori comuni e migliorare la precisione dei dati.
- Caricamento dei dati: Dopo aver effettuato le modifiche, gli utenti possono caricare i dati sulla piattaforma di OpenStreetMap per renderli disponibili agli altri utenti. [4]

### 2.6.2 Più nel dettaglio OsmDroid e MapsForge

MapsForge e OsmDroid sono due librerie OpenSource per la visualizzazione di mappe su dispositivi mobile Android.

- **MapsForge**: MapsForge è una libreria per il rendering di mappe offline, supporta il formato dati di mappa MapsForge, che è un formato compresso e ottimizzato. Offre anche funzionalità per il rendering della mappa in tutte le sfaccettature che possono andare dall'etichettatura al routing e la gestione degli eventi.

Problema: questa libreria ha bisogno di mappe Offline (quindi non aggiornabili automaticamente), con un formato ottenibile solo tramite un loro plug-in, il quale è andato in disuso circa nel 2020.

- **OsmDroid**: D'altra parte osmDroid è una libreria che permette di scaricare e utilizzare anche mappe Offline, ma in modo da risultare meno efficienti rispetto a quelle di MapsForge. Il principale utilizzo è quello di fornire un wrapper per l'utilizzo di mappe OSM su dispositivi Android, in questo modo si avranno mappe aggiornate in tempo reale, implementando automaticamente anche il routing esterno automatico il quale avrà un tempo di aggiornamento medio di 3 giorni . Anche esso offre funzionalità per il rendering della mappa in tutte le sue sfaccettature che possono andare dall'etichettatura fino al routing.

In entrambi casi, offrono possibilità di tracciamento GPS e molte altre interazioni basate sulla mappa.

### 2.6.3 Più nel dettaglio OpenLevelUp

OpenLevelUp è una applicazione web che andrà ad utilizzare i dati provenienti da OSM per renderizzare i vari livelli degli edifici tramite i tag definiti in precedenza tramite JOSM. Offre la funzionalità di guardare gli interni degli edifici, livello per livello interattivamente. Per chiarimenti riguardanti il suo funzionamento si consiglia di visitare il seguente link:

<https://openlevelup.net/14/45.4848/9.2118>

- Utilizzo: durante la navigazione sulla mappa, è necessario un caricamento per ottenere i dati in tempo reale da OpenStreetMap (OSM). Quando ci si trova sopra un edificio mappato con il tag "level" in OpenStreetMap e si ha un determinato livello di zoom, OpenLevelUp mostrerà una lista che permetterà di selezionare il piano desiderato. A un livello di zoom più elevato, verrà visualizzata l'intera mappa con i relativi dati indoor, come illustrato nelle Figure 11 e 12 precedentemente mostrate. Per navigare tra i diversi livelli, sarà possibile utilizzare pulsanti presenti nel "level selector", e ogni elemento sarà cliccabile per accedere ai dettagli degli oggetti correlati, insieme alla loro descrizione. Nel caso in cui la mappa non sia sufficientemente ingrandita, verranno mostrate le aree relative ad OpenStreetMap, con un colore che gradualmente tende al rosso in base alla quantità di dettagli indoor presenti in quella zona di mappa. Come in Figura 13. [7]

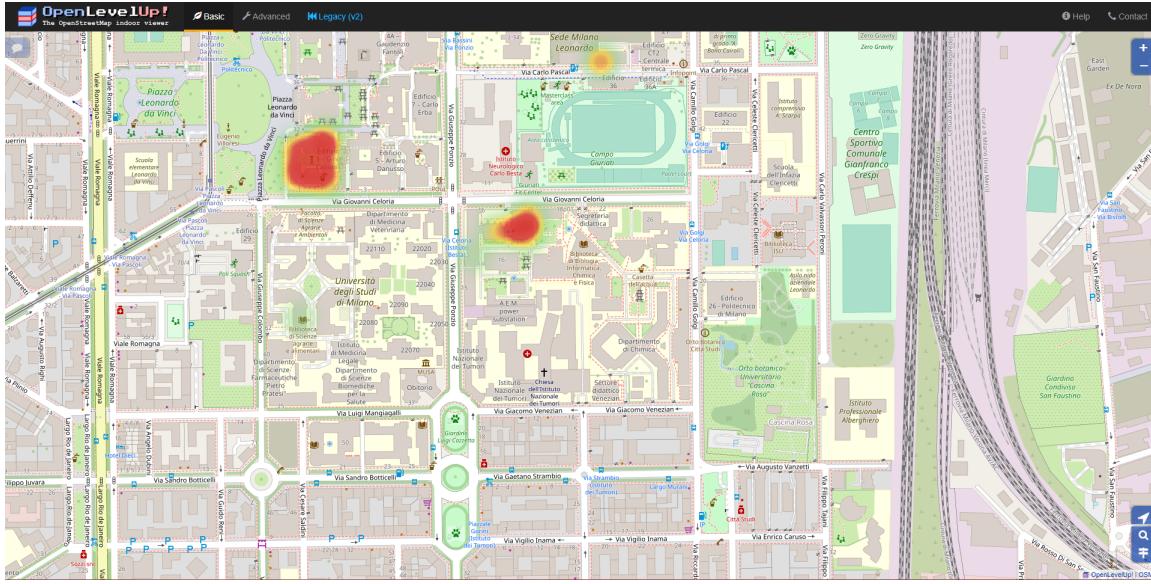


Figura 13: OpenLevelUp con uno zoom minore

Problema: attualmente, non è disponibile una documentazione specifica per le chiamate API relative alla mappa. L'integrazione della mappa su un sito web è l'unica forma di utilizzo documentata. Tuttavia, è importante notare che sono presenti diverse difficoltà che rendono l'utilizzo della mappa complicato. Queste difficoltà includono limitazioni nell'abilità di personalizzare molti degli elementi della mappa stessa. Inoltre, la mancanza di recenti aggiornamenti indica che il progetto potrebbe essere in disuso. Gli ultimi aggiornamenti e le ultime conversazioni documentate risalgono a circa tre anni fa, suggerendo una possibile mancanza di attività di sviluppo e supporto attuali.

## 2.7 Routing no tool

L'idea del "Routing no tool" è quella di mettere insieme algoritmi e tecnologie per lo sviluppo di un sistema per la navigazione indoor, senza l'ausilio di servizi di terze parti. Nel paragrafo 3.2 viene descritta nel dettaglio una proposta di questo tipo, composta da tre parti:

- Un sistema per la rappresentazione grafica delle mappe
- Un sistema per mappare la mappa al fine di calcolare i percorsi
- Un sistema per rappresentare graficamente questi percorsi sulle mappe

## 2.8 Position no Tool

Le attuali tecnologie di tracciamento differiscono nelle tipologie di errori che forniscono e nella precisione del calcolo della posizione dell'utente/dispositivo.

### 2.8.1 Principali tecnologie di tracciamento

- UWB: Gli hardware utilizzati in questi sistemi sono molto costosi e permettono di trasmettere impulsi ultra-brevi su multiple frequenze; basano il calcolo della posizione su tecniche come “Time Of Arrival” del segnale a partire da un “Tag”, posizionato su una persona, verso gli “Anchor” per calcolare la Multi-Trilateration. L’accuratezza di questa tecnologia è la migliore tra tutti i sistemi di posizionamento e sotto circostanze ideali può raggiungere precisioni al centimetro. Il costo elevato rende proibitiva la reale applicazione, fortunatamente la logica di implementazione è utilizzata dai principali sistemi di tracciamento basati sui segnali ma non raggiungono lo stesso livello di precisione.
- Segnali Wi-Fi : Ogni smartphone è in grado di rilevare segnali Wi-Fi e molti edifici hanno più di un Access Point che invia questi segnali. Questo è uno dei principali motivi per cui il posizionamento attraverso Wi-Fi è uno dei più popolari per la localizzazione indoor. Per calcolare la posizione dell’utente possono essere impiegate diverse tecniche: Time Of Arrival (ToA) (Come Round Trip Time), Angle Of Arrival (AoA), Received Signal Strength (RSS) oppure Fingeprinting. RSS è la tecnica più semplice perché non richiede un costo computazionale elevato ed è facile da implementare,
- Magnetismo: Il magnetometro non è influenzato solo dal campo magnetico terrestre, ma anche dai cambiamenti nel campo geomagnetico all’interno di un edificio [15]. Ciò significa che utilizzarlo come bussola potrebbe non rilevare sempre con precisione il polo nord magnetico al chiuso. Shu et al. hanno raggiunto un’accuratezza fino a 0.9 metri registrando l’impronta geomagnetica di un edificio e utilizzando quel pattern per localizzare l’utente nel tempo. Questo approccio è possibile solo perché i campi magnetici specifici della posizione sono generalmente stabili nel tempo e, quindi, possono essere mappati. Tuttavia, questo approccio ha i suoi limiti. Gli oggetti in movimento o persino le persone all’interno dell’edificio o i cambiamenti di temperatura possono influenzare il magnetometro e ridurne l’accuratezza in modo non uniforme. Può essere utilizzato come approssimazione dell’orientamento di un utente all’interno di un edificio.
- Bluetooth: Distribuendo i beacon Bluetooth in un edificio, i dispositivi al suo interno sono in grado di ricevere diversi di questi segnali e utilizzare algoritmi di trilaterazione per determinare la loro posizione relativa a tali beacon. L’approccio per calcolare la posizione dell’utente è molto simile alle soluzioni Wi-Fi, in quanto utilizzano RSS, Angle Of Arrival o tecniche simili. Lo svantaggio principale dei sistemi di posizionamento Bluetooth è che si basano su hardware speciale rispetto ai punti di accesso Wi-Fi già installati.
- Basato su Visione: E’ possibile mappare una scena tridimensionale in un’immagine bidimensionale, ad esempio creando un ambiente 3D in Unity, il quale fornisce anche package

per la realtà aumentata. Le informazioni sul mondo circostante possono quindi essere utilizzate per calcolare la posizione della telecamera all'interno di quell'ambiente. Questa localizzazione può essere effettuata raccogliendo punti di riferimento unici all'interno dell'immagine e associandoli a un dataset che contiene la posizione delle immagini di riferimento e la posizione correlata della telecamera all'interno dell'edificio. È anche molto comune che i sistemi di posizionamento basati sulla visione incorporino altri sistemi di posizionamento che si basano su sensori diversi. Ad esempio, gli smartphone dispongono di una serie di sensori a loro disposizione, come accelerometri, giroscopi e magnetometri. Questi possono essere tutti utilizzati insieme per compensare le relative limitazioni.

I vantaggi di questo sistema includono la sua capacità di passare in modo trasparente dalla navigazione outdoor a quella indoor e il fatto che non si basa su infrastrutture aggiuntive, rendendolo scalabile e più economico rispetto ad altri approcci. Uno svantaggio è che la qualità dell'immagine è influenzata dalle condizioni di illuminazione e in particolare dall'occlusione più di qualsiasi altra tecnica. Anche le superfici riflettenti possono ridurre drasticamente l'accuratezza del tracciamento. L'accuratezza dei sistemi di posizionamento basati sulla visione può variare da 1 mm a 1,5 m. Questa elevata precisione è possibile solo in un ambiente controllato e di piccole dimensioni. Utilizzare un sistema del genere in uno spazio più ampio riduce tale precisione.

### **2.8.2 Affluenza**

Le tecnologie descritte nel capitolo 2.8.1 come: la UWB, segnali WI-FI, sensori di prossimità, infrarossi e Bluethoot (beacon), possono essere impiegate per determinare l'affluenza in una specifica area, al fine di fornire un routing ottimizzato che tenga conto delle problematiche legate la presenza del sovraffollamento.

La capacità di raccogliere e analizzare dati in tempo reale tramite sensori e dispositivi mobili consente di monitorare l'affluenza di persone in diverse aree cruciali. Utilizzando determinati algoritmi è possibile analizzare questi dati e fornire all'applicazione queste aree, così da andare a disattivare determinati nodi nel caso l'utente richieda un percorso più libero e quindi suggerire percorsi meno affollati.

L'utilizzo di queste tecnologie rappresenta quindi un importante passo avanti nel garantire un'esperienza di navigazione personalizzata e inclusiva, che tenga in considerazione le esigenze specifiche di diverse categorie di persone, tra cui coloro che sono più sensibili all'afflusso e all'ansia sociale.

### **2.8.3 L'influenza degli errori**

Ogni sistema di localizzazione è soggetto a qualche grado di errore. La natura fisica dei sensori impiegati in questo processo introduce variabili che influenzano le misurazioni. Anche se esistesse un

sensore perfetto che riesca ad avere una performance perfettamente lineare nello spazio di misurazione, questo potrebbe essere influenzato da sorgenti esterne. Nella Figura 14 andrò a mostrare l'accuratezza per ogni dispositivo.

Tracking System	Accuracy
UWB	1 cm – 0.3 m
Wi-Fi Based Positioning System	1 – 10 m
Magnetic Positioning System	1 – 8 m
GPS	1 – 10 m
Bluetooth	0.5 – 10 m
Vision-Based	1 mm – 1m
PDR	n/a

Figura 14: Precisione di misurazione

Ad esempio un magnetometro, nonostante sia capace di misurare perfettamente il campo magnetico terrestre, potrebbe essere influenzato dalla presenza non prevista di un magnete molto potente.

Altre sorgenti che inducono errori sono la vibrazione e la temperatura. Se si utilizzano sensori per PDR, ad esempio il giroscopio o l'accelerometro di uno smartphone, questi potrebbero essere soggetti alle vibrazioni del corpo dell'utilizzatore che porteranno a errori di calcolo. I valori inaccurati delle misurazioni possono essere attenuati attraverso algoritmi di filtraggio ma hanno un costo computazionale abbastanza alto da introdurre latenza nel calcolo del segnale filtrato. In questo lavoro si mostreranno le principali tecnologie e algoritmi impiegati nella Navigazione Indoor e il corrente stato dell'arte.

## 3 Proposte

### 3.1 Navigazione esterna

Una possibile soluzione alla navigazione esterna è l'utilizzo dell'SDK di Navigine, che come menzionato precedentemente, permette di modellare spazi interni ed esterni e collegarli attraverso un routing automatico. Navigine si focalizza sugli spazi interni, perciò modellare tutti gli spazi esterni risulta particolarmente macchinoso. La soluzione adottata per la navigazione esterna è stata quindi l'utilizzo di [OsmDroid](#), poiché consente di sfruttare tutte le funzionalità di OpenStreetMap e di avere un'interfaccia simile a Google Maps, con un alto livello di informazioni sui principali punti di interesse di ciascuna città. La navigazione esterna di OpenStreetMap è attuata con routing automatico tra gli edifici e segnali GPS per stabilire la posizione dell'Utente.

#### 3.1.1 Implementazione della navigazione esterna e dei piani

- Servizio Esterno: la prima proposta dopo aver studiato il funzionamento di OpenStreetMap con JOSM, è stata quella di cercare un tool che permetesse di eseguire il rendering dei diversi piani in automatico. Questo perchè OpenStreetMap ti permette di creare dei piani, ma non li renderizza da solo. Durante la ricerca effettuata, sono stati trovati alcuni tool, elencati al link [https://wiki.openstreetmap.org/wiki/Indoor\\_Mapping](https://wiki.openstreetmap.org/wiki/Indoor_Mapping). Tra tutti, quello che più destava maggior interesse è OpenLevelUp, il quale consente di dividere gli edifici in piani in base al tag "level=\*".

Problema: non esistono più delle chiamate Api che fanno riferimento a questo sito e quindi ai suoi servizi.

- OverPassApi: La seconda proposta è stata quella di utilizzare un linguaggio per l'interrogazione (query) e l'ottenimento di dati geografici da OpenStreetMap. L'obiettivo era fornire un metodo strutturato e personalizzato per accedere a tali dati, consentendo di ottenere informazioni riguardanti mappe, strade, punti di interesse, edifici e altro ancora. Era possibile filtrare i dati in base a criteri specifici e scaricarli in vari formati come XML, JSON o CSV.

Tuttavia, durante la ricerca delle diverse funzioni offerte da Overpass API e tramite richieste sui forum di OSM, non è stato possibile trovare le istruzioni necessarie per cambiare il tileset acquisito da OpenStreetMap utilizzando OsmDroid. Questo ha reso impossibile modificare la mappa e nascondere determinati livelli durante un evento specifico.

Questo problema è principalmente dovuto alla complessità nella costruzione di tecnologie indoor, ambito ancora in fase di esplorazione e sviluppo. Le combinazioni possibili tra Overpass API e OSMDROID si limitano principalmente alla ricerca di luoghi, senza possibilità effettive di modificare la mappa stessa, ma piuttosto di aggiungere elementi sopra la mappa.

Di conseguenza, abbiamo optato per un’alternativa più semplice per raggiungere i nostri obiettivi.

- OsmDroid groundOverlay: la terza proposta, dopo aver esplorato i vari metodi messi a disposizione da OsmDroid, è stata quella di riprodurre un comportamento molto simile a quello di GoogleMaps, la quale implementazione è solo per gli edifici dell’università. Il comportamento proposto è stato quello di salvare le immagini dei piani riguardanti gli edifici all’interno dell’applicazione (essendo nel nostro caso non tanti), per poi prenderli e posizionarli nelle coordinate (Longitudine,Latitudine) della mappa fornita da OpenStreetMap, in questo caso non si andranno a cambiare gli elementi della mappa stessa, ma bensì a posizionarli sopra un Overalay.

Più in dettaglio: quando si esegue uno zoom sopra un edificio dell’università, il sistema riconoscerà l’edificio che si sta guardando e quindi andrà a prendere quella mappa corrispondente all’edificio stesso.

Perchè usiamo OSM? La navigazione interna è stato deciso di dividerla dalla navigazione esterna eseguita da OSMDROID, infatti andrà solo a mostrare i vari piani con un puntatore sull’aula nel caso si volesse cercare un oggetto più in specifico, e cambiare i piani in base ai listener. Infatti, è stato deciso di usare OSM proprio perchè poteva essere utile tener aggiornata non solo la mappa indoor, ma anche quella esterna, andando a modificare le strade tramite il tool web di OpenStreetMap nel caso di un lavoro o impedimento. Infatti dopo aver apportato modifiche, il sistema ci impiega un lasso di tempo di circa due giorni per l’aggiornamento del routing esterno. Ovviamente si poteva andare ad implementare anche il routing interno su OSM, ma risultava scomodo: per il massimo zoom possibile, per il design che andava a complicarsi (user experience) e soprattutto per evitare funzionalità diverse messe insieme andando a rallentarsi a vicenda.

### 3.1.2 Implementazione delle mappe

La gestione delle mappe online verrà eseguita da OsmDroid stesso, fornendo diversi metodi implementativi per richiamare i diversi formati proposti da OSM stesso o da altre fonti. Per quanto riguarda l’implementazione delle mappe offline, in modo che se un utente non abbia connessione può comunque orientarsi, abbiamo usato dei tool esterni che potessero fornirci mappe aggiornate.

- MobileAtlas: Mobile Atlas Creator (MOBAC) è un software open-source che consente di creare mappe personalizzate per dispositivi mobili. È progettato per estrarre dati cartografici da varie fonti, inclusi servizi web di mappe online, come OpenStreetMap, Google Maps, Bing Maps, e convertirli in formati compatibili con dispositivi mobili.

Con MOBAC, è possibile selezionare l’area geografica di interesse, definire il livello di zoom desiderato e scegliere il formato di output compatibile con il dispositivo mobile specifico, come

Atlas SQLite, RMaps SQLite, OruxMaps SQLite, Garmin Custom Map, ecc. Il software scarica quindi le immagini delle mappe e le memorizza nel formato desiderato.

MOBAC offre anche opzioni per personalizzare il rendering delle mappe, inclusa la scelta dei livelli di zoom, la selezione dei tipi di dati da includere (strade, punti di interesse, ecc.), l'aggiunta di annotazioni e altro ancora.

Le mappe create con MOBAC possono essere utilizzate offline sui dispositivi mobili, consentendo agli utenti di visualizzare e navigare nelle mappe anche senza connessione internet. MOBAC è compatibile con diversi sistemi operativi, tra cui Windows, macOS e Linux. Il problema è che stato bloccato il traffico da OSM stesso per uso eccessivo. Il link per la guida a MOBAC è il seguente <https://mobac.sourceforge.io/quickstart/using2.htm>

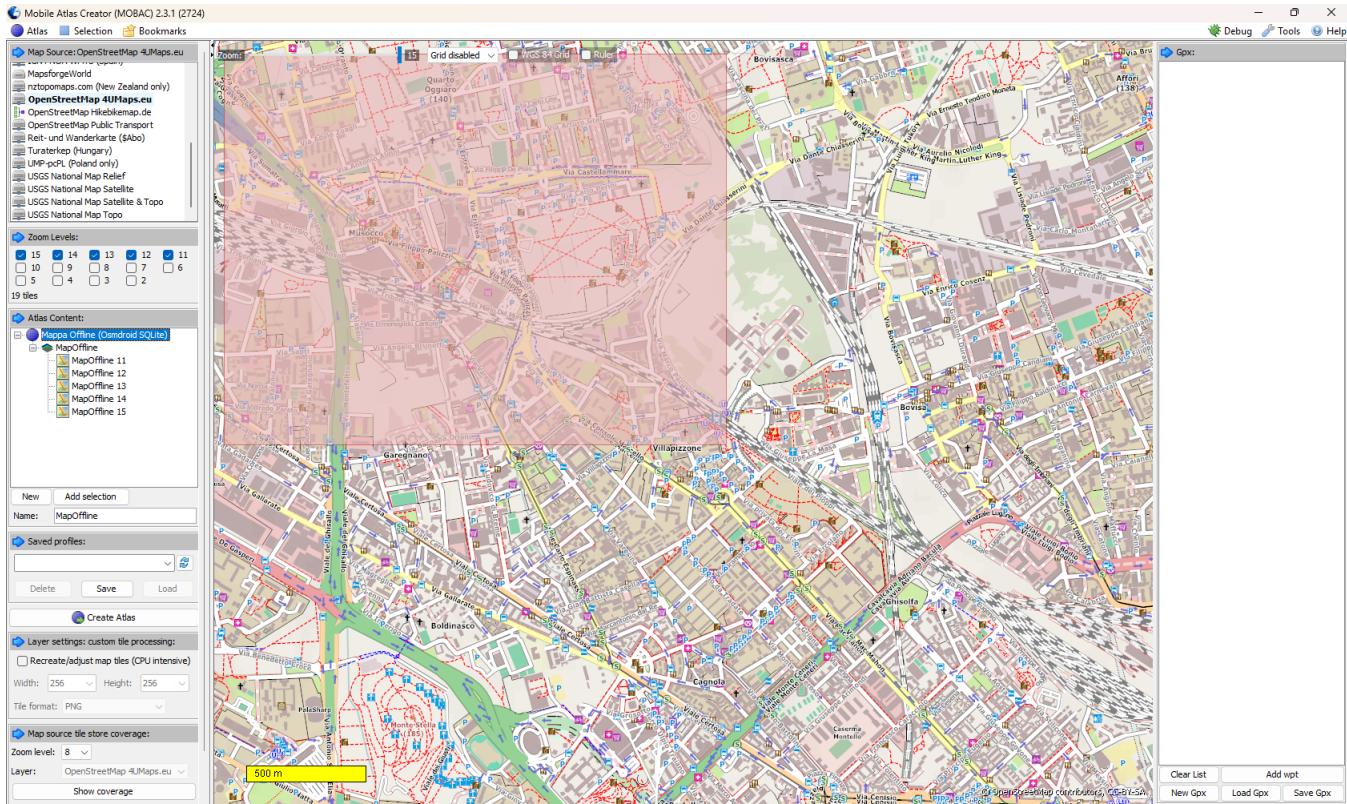


Figura 15: Interfaccia MOBAC

L'interfaccia di MOBAC, Figura 15, si presenta nel seguente modo:

- Map Source (alto a sinistra): nel menù in alto a sinistra MOBAC andrà a mostrarcì tutte le fonti dove poter andare a prendere il tileset. Molti di questi però sono bloccati e uno di questi è proprio OpenStreetMap.

- Zoom level: andrà a scaricare le immagini in base allo zoom minimo e massimo. A zoom level 15, che in questo caso è il massimo, andrà a scaricare tutte le immagini che riguardano quella zona più in dettaglio, quindi rispetto allo zoom level 11, che sarà la mappa rimpicciolita, la sua cartella conterrà più immagini per completare la mappa scelta.
  - Atlas Content: andrà a mostrarti gli elementi scelti fino ad ora, dallo zoom level alla source. Si potrà aggiungere più di una MapSource.
  - Saved Profiles: serve per andare a definire un nome per il file e per creare definitivamente il nostro tileset.
  - Layer settings & Map Source tile coverage & selection Coordinates: sono opzioni per le quali si andranno a ridefinire piccoli dettagli riguardanti le aree scelte e la coverege riguardante ogni zoom level per la mappa scelta.
  - Mappa (centrale) : nel seguente riquadro sarà possibile visualizzare la mappa che si desidera scaricare:  
il riquadro rosso indica la mappa selezionata insieme alle relative dimensioni presenti nel riquadro giallo. In alto invece potremo andare a modificare lo zoom level per capire quanto dettagliatamente vogliamo lo zoom riguardante la mappa.
  - Riquadro a destra: viene usato per creare e gestire una serie di punti (waypoints) e tracciati usando il formato GPS eXchange format.
- Maperitive: Maperitive è un software open-source utilizzato per generare mappe personalizzate e interattive a partire dai dati di OpenStreetMap. È progettato per essere utilizzato principalmente come strumento di rendering cartografico e offre una vasta gamma di funzionalità per visualizzare, analizzare e creare mappe basate sui dati di OpenStreetMap. Maperitive permette agli utenti di selezionare specifici elementi della mappa, applicare stili di rendering personalizzati, aggiungere etichette, simboli e altre annotazioni, e generare immagini o file di mappa in vari formati. Può essere utilizzato per creare mappe per scopi diversi, come escursionismo, ciclismo, pianificazione di viaggi, analisi geografiche e altro ancora. Maperitive è un software indipendente che può essere eseguito su computer desktop Windows, macOS e Linux. Il link per la guida a Maperitive è il seguente <http://maperitive.net/>

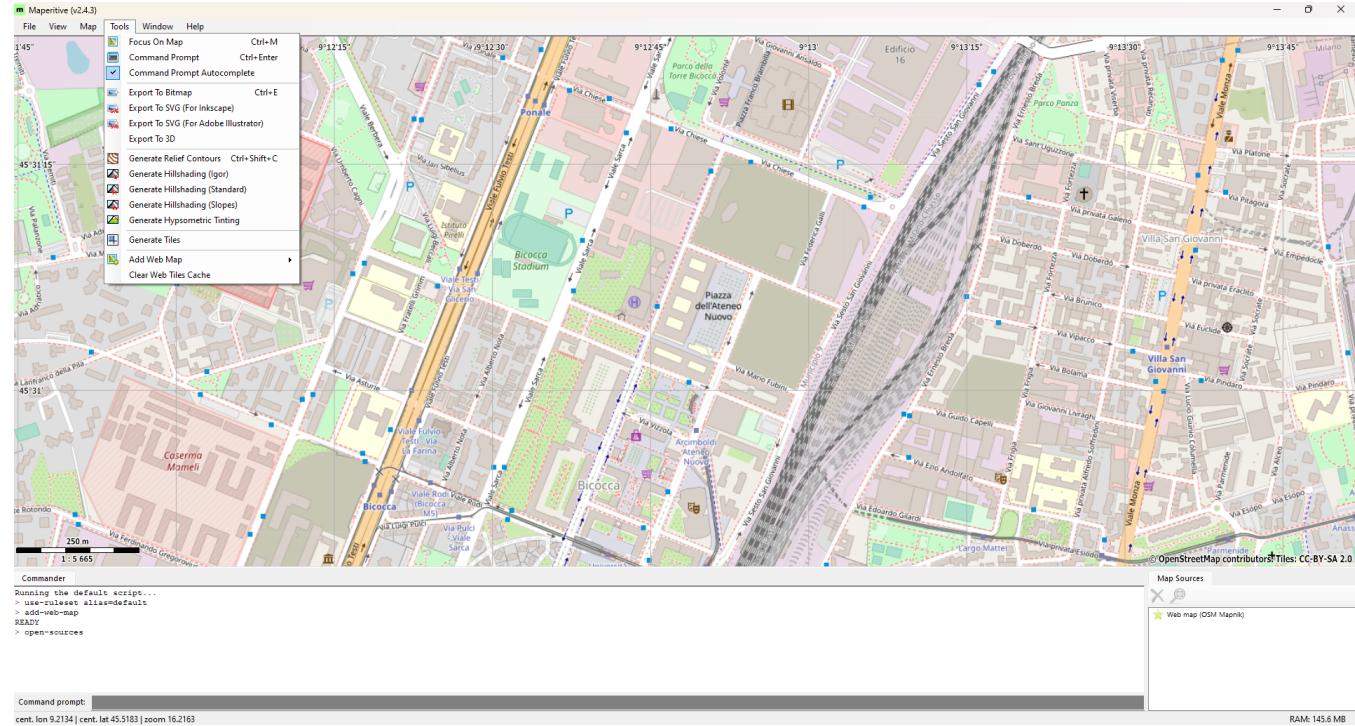


Figura 16: Interfaccia Maperative

L’interfaccia di Maperative, Figura 16, si presenta nel seguente modo:

- File: si potrà cambiare la source della mappa, nel nostro caso si è scelto di utilizzare OSM. Oppure andare ad eseguire uno script direttamente da un file presente sul computer.
- View: cambio di zoom senza l’utilizzo di mouse, e si potrà andare a definire dei point come la Home.
- Map: sono presenti tutte le informazioni che riguardano la mappa stessa. Si potranno andare a definire dei contorni, togliere la gridView e la possibilità di scaricare dati grazie alle api di OSM (Overpass Api) potendo modificare le regole della mappa stessa.
- Tools: andrà a mostrati tutte le informazioni per l’esportazione della mappa nel formato voluto per poi generare il tile, oppure aggiungere altre mappe sopra quella già presente.

La nostra scelta per generare le mappe Offline ricade quindi su Maperative, non potendo utilizzare MOBAC.

## 3.2 Routing no tool per la navigazione indoor

### 3.2.1 Struttura generale della soluzione

La soluzione routing no tool è basata sull'idea di sviluppare una tecnologia per la navigazione indoor senza dipendere da servizi di terze parti. Di seguito viene fornita una descrizione di questa tecnologia:

La mappa dell'edificio viene importata come un'immagine png. Successivamente vengono selezionati i punti di interesse all'interno della mappa, come stanze e corridoi, e vengono rappresentati inizialmente come coordinate all'interno dell'immagine. Questi punti diventeranno i nodi del grafo, insieme alle relazioni tra di essi (gli archi), che verranno utilizzati per la navigazione. Il sistema di tracciamento delle linee sulle quali verrà creato il percorso effettivo sfrutta le funzionalità di Bitmap, Canvas e Paint.

### 3.2.2 Gestione della mappa e sua visualizzazione

Come accennato precedentemente, la mappa è stata inserita come un PNG e successivamente "racchiusa" in un PhotoView. Un PhotoView ha dei comportamenti estremamente simili a quelli di un ImageView, al contrario di quest'ultimo però, fornisce una visualizzazione di immagini avanzata con la possibilità di zoomare e trascinare l'immagine. Uno dei problemi di questa soluzione è il cambio di risoluzione dell'immagine dato da differenti dispositivi, più avanti mostreremo come causerà incertezza sulle coordinate dei punti.

### 3.2.3 Il grafo e il calcolo del percorso

Una delle classi più importanti che costituisce questa soluzione è appunto la classe rappresentante il grafo. Questa classe fa un uso diretto delle classi utili a rappresentare i nodi e gli archi. Il suo funzionamento è dunque quello di un classico grafo pesato non orientato; ci sono dei nodi e delle relazioni tra essi, ovvero gli archi. Ogni nodo contiene un id per la sua identificazione e delle coordinate x e y del punto sulla planimetria, inoltre contiene una lista di archi in cui il nodo è "coinvolto". Ogni arco invece contiene un nodo di destinazione e un peso. A questo punto è stato selezionato, come algoritmo per il calcolo del percorso con peso minore, l'algoritmo di Dijkstra. Esiste dunque un metodo che ricostruisce il percorso con peso minore individuato da Dijkstra e successivamente restituisce una lista dei nodi ordinati dal punto di partenza al punto di arrivo. A questo punto sarà compito di altre parti della soluzione gestire il tracciamento effettivo.

### 3.2.4 L'algoritmo di Dijkstra e sue modifiche

L'algoritmo di Dijkstra, determina la distanza minima, in termini di peso, tra un nodo sorgente e tutti i nodi del grafo preso in questione. Qui di seguito vi è lo pseudocodice di Dijkstra. Sia:

- G: Un grafo orientato e pesato, avente pesi non negativi

- $w$ : La funzione peso che associa ad ogni arco in  $G$  un valore reale positivo
- $start$ : Il nodo sorgente

```

function Dijkstra(G, w, start):
    init_source(G, start)
    S = {};                      // la soluzione è l'insieme vuoto
    H = V;                      // nella struttura dati inseriamo tutti i vertici V
    while(!H.isEmpty){
        u = extract_min(H)
        S = S U {u}           // aggiungiamo u alla soluzione S
        for(all v in Adj(u)}   // per ogni vertice nella lista di adiacenza di u
            relax(u, v, w)
    }

function init_source(G, s){
    for(v in V){
        d(v) = inf.
        pred(v) = null
    }
    d(s) = 0
}

function relax(u, v, w){
    if(d(v) > d(u) + w(u, v))
        d(v) = d(u) + w(u, v)
        pred(v) = u
}

```

L'algoritmo si suddivide essenzialmente in due parti precedute da una fase di inizializzazione:

- Inizializzazione: qui si assegna ad ogni nodo una distanza "infinita" tranne al nodo sorgente a cui si assegna una distanza pari a zero. Inoltre si settano tutti i predecessori a null.
- Fase 1: Si sceglie tra tutti i nodi inesplorati quello con distanza minore, chiameremo questo nodo "A". Se non sono presenti nodi inesplorati l'algoritmo termina.
- Fase 2: Scelto il nodo A, si aggiornano le distanza dei nodi collegate direttamente con un arco ad A. L'aggiornamento prevede la sostituzione del predecessore e la sostituzione del valore

della distanza del nodo in esame con la distanza di A sommata al peso della arco tra A e il nodo. Questo solo se la disitanza è minore a quella assegnata precedentemente al nodo in esame. a questo punto si torna alla fase uno.

Nel nostro caso non è interessante sapere le distanze minime tra il nodo sorgente e tutti gli altri, ma piuttosto la distanza minima tra il nodo sorgente e il nodo di arrivo. Possiamo dunque considerare di modificare le condizioni di "fermo" dell'algoritmo. In particolare nella fase uno, una volta selezionato il nodo "A", con distanza minima tra quelli inesplorati, se questo è anche il nodo di arrivo, allora l'algoritmo giunge al termine. Infatti, siamo sicuri che una volta selezionato A, la sua distanza dalla sorgente non può migliorare ulteriormente. Questo perchè, se esistesse un'altra strada con peso minore, allora A, in questa fase, non verrebbe selezionato tra i nodi inesplorati, in quanto ci sarà un altro nodo B, avente un valore della distanza "migliore" rispetto ad A. Di seguito lo pseudocodice dell'algoritmo modificato[11]:

```

function Dijkstra(G, w, start, end){      // aggiunta nel nodo di destinazione "end"
    init_source(G, start)
    S = {};
    H = V
    while(!H.isEmpty){
        u = extract_min(H)
        if(u == end){           // una volta trovato il nodo si interrompe il ciclo
            break;
        }
        S = S U {u}
        for(all v in Adj(u))
            relax(u, v, w)
    }
    return reconstructPath(G, start, end)
}

```

Una volta arrivati a questo punto, dobbiamo ricostruire il percorso in modo da poter restituire una lista di nodi ordinati che vanno dal nodo sorgente al nodo di destinazione. Per farlo ci avvaliamo del seguente algoritmo:

```

function reconstructPath(predecessors, start, end){
    path = create empty list

```

```

currentNode = end

while (currentNode != null AND currentNode != start){

    add currentNode at the beginning of path
    currentNode = predecessors[currentNode]

}

if (currentNode != null AND currentNode == start){

    add start at the beginning of path
} else{

    clear path
}

return path
}

```

### 3.2.5 Selezione dei punti e tracciamento del percorso

Una volta selezionati i punti di interesse (rappresentanti le stanze, i corridoi, ecc.), ne sono stati scelti dei nuovi che all'interno della presente relazione d'ora in avanti verranno chiamati "punti di appoggio". Questi punti di appoggio rappresentano dei nodi utili al tracciamento del percorso.

Tramite questo sistema riusciamo a gestire il disegno del percorso sulla mappa, che seguendo i punti di interesse e i punti di appoggio, non dovrà "preoccuparsi" di evitare ostacoli come ad esempio i muri, in quanto basterà posizionare tutti i punti in modo strategico.

### 3.2.6 Gestione delle coordinate dei punti cambiando risoluzione e grandezza dell'immaginee di partenza

Tutti i punti selezionati devono essere rappresentati tramite coordinate all'interno dell'immagine. Il problema di cui accennato al punto 1 nella sezione "struttura generale della soluzione", riguarda il fatto che queste coordinate sono state inizialmente prelevate dall'immagine (a una certa "grandezza" e risoluzione) e successivamente devono poter rappresentare gli stessi punti nella stessa immagine ma con grandezza e risoluzione differente (in quanto dispositivi differenti avranno risoluzioni degli schermi e layout differenti). Per tale ragione invece di salvare semplicemente le coordinate x e y è stato deciso di salvare il rapporto tra la coordinata (x e y) effettiva del punto e la grandezza (x e y) totale dell'immagine. Ad esempio: la x dell'immagine totale ha lunghezza 1200 pixel e il punto interessato si trova a 600 pixel sulla x, allora verrà salvata come coordinata x: 600/1200, ovvero 1/2. Il medesimo procedimento verrà effettuato con la coordinata y. Ora, a prescindere della grandezza e

dalla risoluzione dell'immagine, basterà moltiplicare il rapporto precedentemente calcolato con la dimensione totale della coordinata x (o y) della PhotoView in quel "momento", ottenendo così una trasposizione tra le coordinate iniziali e quelle effettive sull'immagine presa attualmente in considerazione.

### 3.2.7 Gestione dei cammini personalizzati

Tra le specifiche del progetto vi è la gestione del percorso per persone con disabilità e/o con problematiche che rendono impossibile o difficoltoso il percorrimento di una certa zona dell'ateneo. Tra gli esempi che ci sono stati forniti vi erano:

- persone con disabilità per cui non è possibile percorrere le scale
- persone con fobie per cui è preferibile o è meglio evitare totalmente gli ascensori
- persone con fobie tali che mettono a disagio il soggetto in ambienti molto affollati

Per la gestione di questi percorsi "personalizzati" i nodi hanno un attributo che ne identifica il tipo di stanza che rappresenta. Ad esempio un nodo che rappresenta delle scale potrebbe avere un attributo che specifica la presenza di quest'ultime. Nell'applicazione quindi vi è implementato un sistema che evita di percorrere certi nodi in base all'esigenza dell'utente. In particolare, una volta che l'algoritmo di Dijkstra "modificato", mostrato precedentemente, incrocia un nodo, nella arco della "Fase 1" che non può essere percorso, allora lo salta, non estraendolo mai dall'insieme dei nodi inesplorati. Con tale sistema, si evita che così altri nodi abbiano come predecessore quest'ultimo, evitando che nella fase di ricostruzione si generi un percorso con nodi non validi.

## 3.3 Soluzione Tridimensionale Per L'Ateneo Bicocca Basato Su Visione

Mentre le soluzioni di navigazione 2D tradizionali offrono indicazioni chiare su una mappa bidimensionale, l'evoluzione delle tecnologie di realtà aumentata (AR) ha aperto nuove possibilità per una navigazione più immersiva e intuitiva. Unity, un motore di sviluppo di giochi e simulazioni ampiamente utilizzato, consente di combinare elementi virtuali con il mondo reale, offrendo una visione sovrapposta di informazioni digitali direttamente sulla vista del mondo reale attraverso un dispositivo come uno smartphone o un visore AR. Questo approccio consente agli utenti di percepire in modo più accurato la loro posizione e orientamento all'interno di un edificio, fornendo una guida visiva diretta e riducendo la possibilità di errori di navigazione. In questo capitolo, esploreremo le potenzialità offerte da questa tecnologia, analizzando l'algoritmo di navigazione di Unity, i modi di visualizzazione e metodi per correggere la posizione.

### 3.3.1 A\*

L'algoritmo A\* è un algoritmo di ricerca che viene utilizzato per trovare il percorso ottimo tra due punti in un grafo o in una griglia. L'algoritmo combina il miglioramento dei costi di Dijkstra con

una stima euristica dei costi rimanenti per raggiungere la destinazione. Utilizza una funzione di valutazione  $f(n)$  per ogni nodo  $n$ , che rappresenta il costo totale stimato per raggiungere il nodo  $n$  dalla posizione di partenza, includendo il costo finora per raggiungere  $n$  (costo  $g(n)$ ) e una stima del costo rimanente per raggiungere la destinazione da  $n$  (costo  $h(n)$ ). L'algoritmo mantiene due set di nodi: il set aperto, che contiene i nodi che devono ancora essere esplorati, e il set chiuso, che contiene i nodi che sono già stati esplorati. L'algoritmo inizia inserendo il nodo di partenza nel set aperto. Successivamente, l'algoritmo seleziona il nodo con il valore di  $f(n)$  più basso dal set aperto e lo espande. Vengono esaminati i vicini del nodo corrente e per ciascun vicino vengono calcolati i valori di  $g(v)$  (il costo finora per raggiungere  $v$ ) e  $h(v)$  (la stima del costo rimanente per raggiungere la destinazione da  $v$ ). Se il vicino  $v$  non è presente nel set aperto o se il nuovo valore di  $f(v)$  è inferiore al valore precedente,  $v$  viene aggiunto o aggiornato nel set aperto. Il processo continua fino a quando il nodo di destinazione viene raggiunto o fino a quando il set aperto è vuoto, indicando che non esiste un percorso possibile. Durante l'esecuzione, l'algoritmo A\* mantiene un puntatore da ciascun nodo al nodo precedente che ha il costo minimo per raggiungerlo. Questo consente di ricostruire il percorso ottimo una volta raggiunta la destinazione. [8][9]

### 3.3.2 Funzionamento Del Navigation System Di Unity

Quando si desidera spostare intelligentemente gli agenti nella propria simulazione, è necessario risolvere due problemi: come ragionare sull'ambiente per trovare la destinazione e come spostarsi effettivamente verso quella destinazione. Il primo è un problema più globale e statico, poiché si basa sull'intera scena. Il raggiungimento della destinazione è un problema più locale e dinamico, poiché bisogna calcolare la direzione dello spostamento migliore in modo da evitare gli ostacoli e trovare il percorso al minor costo. Le aree percorribili definiscono i luoghi nella scena in cui l'agente può muoversi in base alle sue dimensioni (possibilmente umane). In Unity, gli agenti vengono descritti come cilindri. L'area percorribile viene creata automaticamente dalla geometria della scena mediante il test delle posizioni raggiungibili dall'agente. Successivamente, le posizioni vengono connesse a una superficie che si posa sulla geometria della scena. Questa superficie è chiamata Navigation Mesh (NavMesh). La NavMesh memorizza questa superficie come poligoni convessi. I poligoni convessi ci permettono di sapere che non ci sono ostacoli tra due punti all'interno di un poligono. Oltre ai contorni dei poligoni, memorizziamo informazioni su quali poligoni sono vicini l'uno all'altro. Questo ci consente di ragionare sull'intera area percorribile, Figura 17.

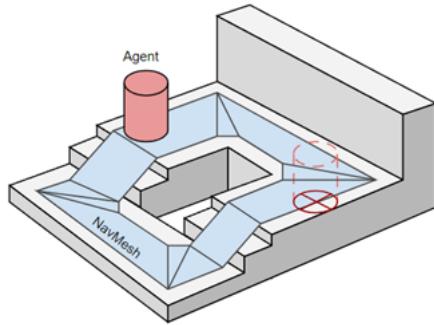


Figura 17: Esempio area percorribile

Per trovare un percorso da percorrere per raggiungere un punto, prima dobbiamo mappare le posizioni di partenza e di destinazione ai poligoni più vicini. Successivamente, iniziamo la ricerca dalla posizione di partenza, visitando tutti i poligoni vicini fino a raggiungere il poligono di destinazione. Seguendo i poligoni visitati, possiamo trovare la sequenza di poligoni che condurranno dalla partenza alla destinazione. Un algoritmo comune per trovare il percorso è l'A\* (pronunciato "A star"), che è l'algoritmo utilizzato da Unity. La sequenza di poligoni che descrive il percorso dal punto di partenza al poligono di destinazione viene chiamata corridoio. L'agente raggiungerà la destinazione dirigendosi sempre verso l'angolo successivo visibile del corridoio. Nel caso della navigazioni indoor, è sufficiente individuare tutti gli angoli del corridoio in una sola volta, modellare una linea attraverso LineRenderer e permettere al personaggio (Utente) di muoversi lungo i segmenti di linea che collegano gli angoli. Figura 18. [13]

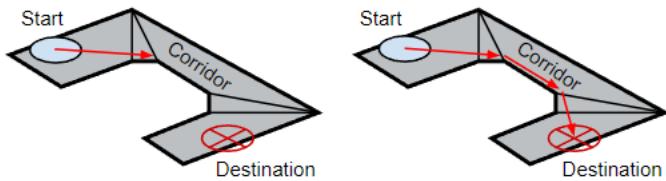


Figura 18: Un agente che segue il percorso

### 3.3.3 Esempio

Se si vuole visualizzare la linea relativa al percorso, allora dobbiamo utilizzare LineRender. LineRenderer è un Component, i Component contengono proprietà modificabili per definire il comportamento di un GameObject, e in particolare permette di renderizzare sull'ambiente una linea

formata da un parallelepipedo. Aggiungendo LineRenderer, Figura 20 come Component ad un oggetto che segue la camera del telefono, la posizione della telecamera viene presa come origine della linea e viene quindi continuamente aggiornata, Figura 19. "NavMesh.CalculatePath" ci permette di calcolare il path su tutta l'area di una NavMesh in corrispondenza di un punto iniziale e un punto finale (target), salvando le informazioni sulla linea questa può essere "renderizzata" infine all'interno dell'ambiente.

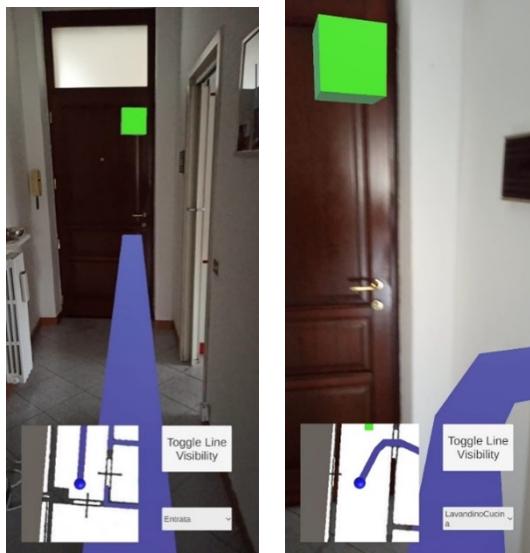


Figura 19: Realtà aumentata

Positions			
Index	X	Y	Z
0	0	-0.07166667	0
1	-2.993334	-0.07166667	-2.866667
2	-7.133334	-0.07166667	-2.866667
3	-7.933334	-0.07166667	-2.466667
4	-8.38	0.1065307	-2.23

Figura 20: Immagini LineRenderer

Nella prima immagine si può vedere una semplice linea retta verso un target, nella seconda si può vedere una curva che percorre la NavMesh verso un altro poligono, nella terza immagine invece si possono vedere l'insieme di punti che formano una curva, il punto "0" è la partenza, il punto "4" la destinazione e i punti intermedi indicano dove la linea deve curvare.

### 3.4 Accedere All'Immagine Catturata Dalla Camera

Unity ci permette di accedere e processare le immagini ottenute dalla camera attraverso l'utilizzo della CPU[12]. Questa modalità per ottenere le immagini visualizzate dalla camera è particolarmente utile se si vuole usare un algoritmo di Computer Vision da applicare all'immagine

reperita. Unity fornisce direttamente un codice utilizzabile, nel caso della navigazione indoor dobbiamo processare direttamente l'immagine e quindi ci serve un metodo sincrono.

Quando viene ricevuto un frame durante la gestione degli eventi dell'ARCamera, si prende l'immagine dalla camera e si stabiliscono dei parametri di conversione, conversionParams è una struttura che contiene quattro diversi dati: il primo è l'immagine in input, il secondo è un vettore di due interi contenente le dimensioni dell'immagine con downsample di 2 (dimezza le dimensioni), il terzo contiene il formato in dell'immagine in output e l'ultimo contiene una trasformazione (MirrorY) da applicare sull'immagine per ottenerle una specchiata lungo l'asse verticale.

Si stabilisce la dimensione utilizzata per salvare l'immagine finale ed è calcolata sull'immagine del frame catturato con applicati i conversionParams, si alloca quindi un buffer per salvare l'immagine, l'immagine viene convertita nel formato specificato e salvata nel buffer. Il frame può essere quindi rimosso dalla Camera che può acquisire un nuovo frame. Viene definita una Texture con i conversionParams (dimensioni e formato) che dovrà contenere l'immagine inserita nel buffer, una volta applicata l'immagine alla texture, anche il buffer può essere svuotato del frame.

E' possibile applicare il metodo Decode() della libreria ZXing per ottenere un testo da un QR Code (se presente all'interno del frame) questo metodo richiede come parametri i pixel della texture (possono essere scannerizzati riga per riga attraverso il metodo GetPixels32() partendo dall'angolo in basso a sinistra [14]), la larghezza e l'altezza della texture. ZXing permette quindi il riconoscimento di QR Code, l'utilizzo di questi codici è molto utile perchè permette in modo semplice, ad un utente, di ristabilire correttamente la sua posizione senza l'utilizzo di segnali esterni. Ciascun QR Code scannerizzabile all'interno di un edificio è collegato, semplicemente tramite il testo che contiene, ad un oggetto inserito nell'ambiente 3D di Unity; una volta ottenuto il testo, basta cercare la posizione dell'oggetto, il cui nome è lo stesso ottenuto dalla funzione Decode(), e utilizzare quest'ultima come nuova posizione origine del telefono, che coincide con la posizione della Camera AR.

## 4 Proposta adottata

La proposta scelta per la creazione della nostra applicazione android è suddivisa in 2 parti:

- Proposta per la navigazione outdoor: 3.1.1, terza scelta riguardante il groundOverlay.
- Proposta per la navigazione indoor: rispetta la proposta descritta nel paragrafo 3.2.

### 4.1 Architettura della soluzione

L'applicazione implementa il pattern software architetturale MVVM (Model-view-viewModel).

Astrae lo stato di view (visualizzazione) e il suo comportamento, mentre il modello di presentazione astrae una vista (crea un view model) che non dipende da una specifica piattaforma. In questo modo si favorisce la separazione delle responsabilità, con conseguente riuso del codice e test dell'applicazione più agevole.

E' composta da 3 componenti principali, come possiamo vedere in Figura 21:

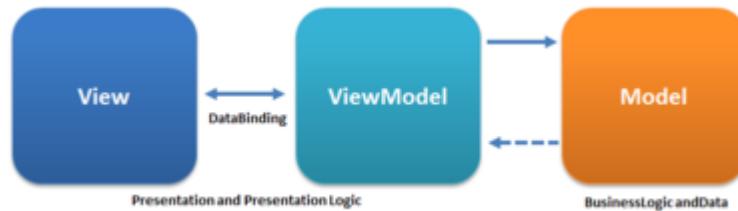


Figura 21: MVVM

- Model: rappresenta i dati e quindi tutte le operazioni ad essi associati, dal suo accesso, alle regole di validazione. Il Model non conosce nulla dell'interfaccia utente e della logica con cui verranno mostrati i dati, ma saprà solo come andarli a prendere e come verranno definiti.
- ViewModel: funge da intermediario tra il model e la view. E' responsabile della presentazione dei dati dal model alla view in un formato adatto per la visualizzazione e dell'elaborazione degli input dell'utente della view per l'aggiornamento di questi ultimi. Andrà anche ad implementare la logica di presentazione, come la gestione degli stati, la formattazione e l'interazione col model. Avrà dei comandi che la view potrà chiamare per eseguire azioni nel ViewModel
- View: corrisponde all'interfaccia dell'utente della nostra applicazione. Questa verrà utilizzata dall'utente per interagire con la nostra applicazione. Quello che farà sarà andare a mostrare i dati del viewModel e reagire agli eventi dell'utente, come dei click.

La comunicazione tra i componenti MVVM avviene tipicamente attraverso il data binding, il quale permette di collegare in modo dichiarativo gli elementi dell'interfaccia utente alle proprietà del View-Model, così da renderli sincronizzati tra loro durante la modifica dei dati.[2][10]

Nella nostra applicazione questa logica viene implementata come in Figura 22:

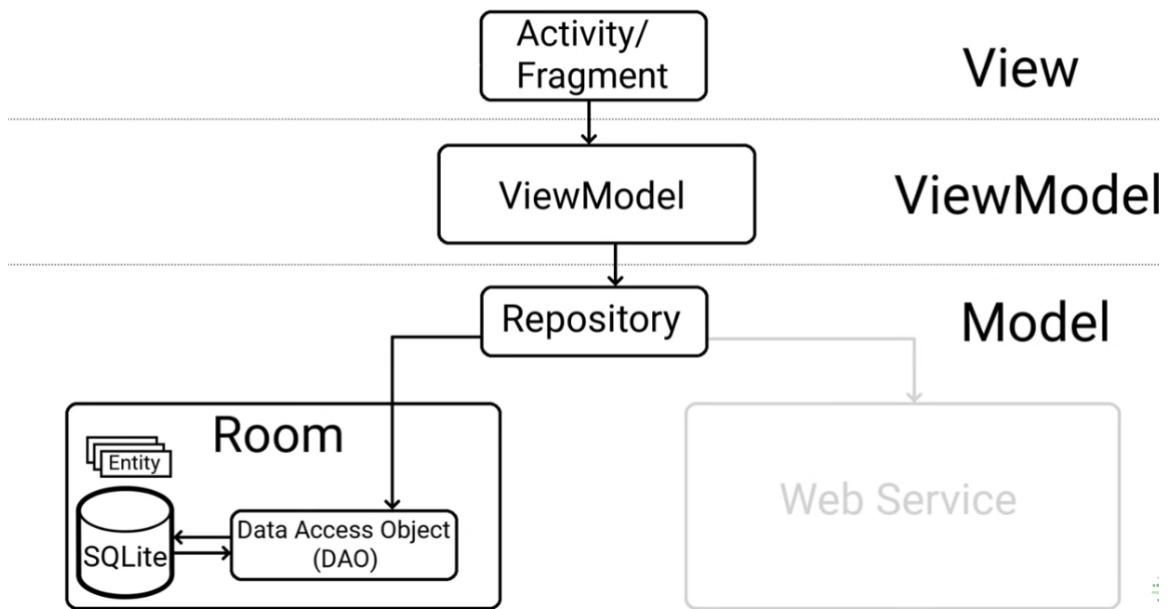


Figura 22: MVVM implementazione

- Room: wrapper attorno a SQLite che si occupa di creare le diverse tabelle e le diverse hyper class e fare le operazioni su queste ultime, così da evitare di scrivere il codice SQL il quale molto spesso portava a crash del sistema. Un' altra funzionalità riguarda gli errori i quali segnano degli errori invece di far partire l'applicazione e farla crashare, rendendo così la ricerca dell'errore più facilitata. Offre un livello di astrazione superiore rispetto alla gestione diretta del database SQLite, supportando anche funzionalità avanzate per il controllo dei tipi di dati a livello di compilazione e le relazioni tra le tabelle e le query complesse.
  - Entity: rappresenta una tabella nel database, ogni istanza di una classe entity rappresenterà una riga nella tabella del database e i campi alle colonne della tabella.
  - DAO (Data Access Object): il dao sarà un interfaccia la quale potrà essere chiamata per accedere ai dati del database ed eseguire tutte le operazioni che ne conseguono (lettura, scrittura, aggiornamento, eliminazione) sui dati del database. L'implementazione dei metodi dell'interfaccia per i DAO verrà eseguita da Room in fase di compilazione.

- Database: Rappresenta il database in sé e funge da punto di accesso principale per interagire con il database. È annotato con l'annotazione `@Database` e definisce la lista delle classi Entity e delle interfacce DAO associate. Room crea l'implementazione concreta della classe Database.
- Repository: è un design pattern che viene utilizzato per gestire l'accesso ai dati, astraendo la logica di accesso ai dati sottostanti, fornendo un'interfaccia semplificata e coerente per l'applicazione. È uno strato intermedio tra i diversi servizi esterni e il `viewModel`.  
Andrà a nascondere i dettagli implementativi riguardanti le funzionalità del database e dei servizi esterni e offrire così metodi ben definiti per eseguire operazioni CRUD (creazione, lettura, aggiornamento, eliminazione) sui dati.
  - Astrazione: nasconde i dettagli specifici dell'implementazione del database o del servizio web sottostante. Così da non renderla dipendente dalla tecnologia utilizzata nel caso in un futuro si trova qualcosa di migliore.
  - Organizzazione e gestione operazioni: va a definire metodi, come detto prima, che implementano operazioni CRUD. Occupandosi di costruire e invocare le query appropriate per le operazioni di accesso ai dati, occupandosi anche delle transazioni.

In questo modo si potrà avere una gestione e divisione delle responsabilità migliore, e così anche la manutenibilità.

- `ViewModel`: andrà a fornire i dati e comportamenti specifici alla View in base alla logica definita, riuscendo anche a far sopravvivere il fragment/activity al cambio di configurazione il quale potrebbe avvenire per esempio con il giramento di schermo.  
Andrà lui stesso a parlare con il Repository per recuperare e salvare i dati da mostrare nella View, trasformandoli in dei formati più adatti al caso.  
Il `viewModel` viene spesso utilizzato con il data Binding per consentire una comunicazione biderazionale tra view e `viewModel`, permette di collegare gli elementi della UI direttamente alle proprietà del `ViewModel` e viceversa.
- View: come abbiamo detto prima rappresenta la UI e quindi tutto quello che la riguarda. Può andare dalla presentazione della sua interfaccia, alla navigazione tra le schermate, alle animazioni, alla gestione degli errori e degli eventi.
- `liveData`: è un wrapper che può contenere qualsiasi tipo di dato, e che può essere osservato. Permette inoltre di facilitare la comunicazione asincrona tra i componenti diversi dell'applicazione
  - Osservabilità: `LiveData` implementa il pattern Observer, consentendo ad altre componenti di registrarsi come osservatori e ricevere notifiche quando i dati cambiano. Le componenti

osservanti, come le View, possono sottoscriversi a un oggetto LiveData utilizzando il metodo observe() o observeForever() per essere notificate dei cambiamenti dei dati.

- Gestione del ciclo di vita: LiveData è consapevole del ciclo di vita delle componenti Android, come le Activity o i Fragment, e gestisce automaticamente la registrazione e la rimozione degli osservatori in base allo stato del ciclo di vita. Ciò significa che le componenti osservanti riceveranno solo gli aggiornamenti dei dati quando sono attive e pronte a visualizzarli.
- Gestione automatica degli aggiornamenti dell'UI: Quando un oggetto LiveData viene aggiornato, tutti gli osservatori registrati vengono notificati e ricevono i nuovi dati. La View può reagire a questi aggiornamenti aggiornando automaticamente l'interfaccia utente in modo coerente con lo stato dei dati.

## 4.2 Implementazione

- Fragment indoor: andrà a mostrare la logica riguardante il routing interno:
  - La struttura del grafo è realizzata tramite la classe Graph, che a sua volta contiene un insieme di nodi, oggetti della classe Node. Il grafo è stato implementato tramite liste di adiacenza, questo vuol dire che ogni nodo contiene una lista dei nodi con cui è direttamente collegato. Questo tipo di implementazione del grafo è comoda al fine di utilizzare l'algoritmo di Dijkstra, che, come spiegato in modo più dettagliato nel paragrafo 3.2, sfrutta la ricerca dei nodi direttamente collegati tramite un arco al nodo selezionato.
  - Il sistema per il disegno dei percorsi sulla mappa è reso possibile tramite la classe MapDrawer.
  - Il FragmentIndoor si occupa di chiamare la classe grafo per il calcolo del percorso e successivamente passa i dati restituiti alla classe MapDrawer che si occuperà di disegnare il percorso sulla mappa. I dati sull'edificio e i rispettivi piani e mappe, verranno prese tramite il ViewModel.
- Fragment Map: andrà a mostrare tutta la logica riguardante la mappa, utilizzando la libreria OsmDroid e le relative classi.
  - Tramite le chiamate riguardanti osmDroid andremo a chiamare la mappa di OpenStreetMap, andandola a caricare man mano che l'utente si muove sullo schermo, così da rendere la navigazione più fluente.
  - Calcolo del routing avviene da un servizio esterno sempre implementato nella libreria, in questo caso ci sarà la scelta del mezzo che si utilizzerà per il percorso esterno.
  - Posizionamento GPS, il fragmentOSM andrà a chiamare la classe GpsManager, la quale si occuperà di mantenere lo stato del gps e quindi disabilitare/abilitare l'icona sulla mappa.

- Wi-fi, tramite un broadCasterReciver definito nelle classi NetworkChangeReciver e NetworkViewModel, andrà a osservare lo stato del wi-fi, così il fragmentOsm capirà quale mappa andare a implementare (Offline/Online aggiornata) e se abilitare il servizio di routing o meno.
- Mostrare le immagini e il numero dei piani dell’edificio corrente (descriveremo il comportamento nel capitolo seguente.) Il numero di elementi da mostrare verranno implementati tramite un CustomAdapter il quale in base all’edificio scelto andrà ad implementarne il numero corretto. Tutte queste informazioni, riguardanti gli edifici e le aule, verranno prese dal database attraverso le classi ViewModel e Repository.
- Tramite un AutoCompleteTextView e il suo adapter potremo andare a mostrare dei suggerimenti riguardanti l’edificio/l’aula che si sta cercando man mano che l’utente digita le lettere.
- Fragment Novità: Andrà a salvare tutte le informazioni riguardanti le problematiche che portano a una modifica dei grafi, e a una modifica del routing esterno tramite i relativi pulsanti.
  - Calcola anche la distanza tra l’edificio più vicino e la tua posizione, se è attivo il gps , tramite la formula di Haversin.
  - Tramite dei broadCasterReciver capirà se i vari servizi implementati : bluethoot, wi-fi, gps sono attivi.

#### 4.2.1 Database

Il sistema implementa un room database per l’applicazione mobile, sfruttando diversi componenti fondamentali. In primo luogo, è stato sviluppato un repository che agisce come intermediario tra il database e il sistema stesso. Tale repository ospita le operazioni di accesso e manipolazione dei dati. Sono stati definiti due DAO (Data Access Object): uno per l’entità ”Edificio” e uno per l’entità ”Aula”. Il DAO relativo all’entità ”Edificio”, denominato ”EdificioDAO”, offre una serie di metodi per l’accesso ai dati relativi agli edifici, tra cui l’inserimento, l’aggiornamento e l’eliminazione. Analogamente, il DAO dell’entità ”Aula”, denominato ”AulaDAO”, fornisce funzionalità simili per l’accesso ai dati relativi alle aule.

Al fine di rappresentare le informazioni sugli edifici, è stata creata la classe ”Edificio entity”. Questa entità comprende i campi necessari per descrivere un edificio, quali il nome, il numero di piani e ulteriori dettagli. Ogni istanza della classe ”Edificio entity” corrisponde a una riga all’interno della tabella del database.

Analogamente, è stata definita la classe ”Aula entity” per rappresentare le informazioni relative alle aule. Tale entità include attributi come il numero dell’aula, la sua posizione e altre caratteristiche pertinenti. Ogni istanza della classe ”Aula entity” rappresenta una riga all’interno della tabella del database.

Attraverso questa struttura, è stato realizzato il room database per il sistema, mediante l'utilizzo del repository, dei DAO e delle entità "Edificio" e "Aula". Questa architettura permette di gestire agevolmente l'accesso e la manipolazione dei dati relativi agli edifici e alle aule all'interno dell'applicazione.

### 4.3 Progettazione interfaccia utente

L'interfaccia della nostra applicazione è stata progettata con l'utilizzo di una BottomNavigationView per consentire una navigazione intuitiva tra i diversi frammenti dell'applicazione: il FragmentIndoor, il FragmentOSM e il FragmentNovità, che verranno contenuti in un Activity. Questa potrà essere vista nelle successive immagini come il riquadro inferiore.

#### Fragment Map

All'interno di questo fragment l'utente potrà trovare:

- Due barre di ricerca in cui inserire rispettivamente, l'edificio di partenza e di destinazione, Figura 23. Se si inserisce solo la destinazione verrà visualizzato sulla mappa un marker che identifica la posizione dell'edificio.

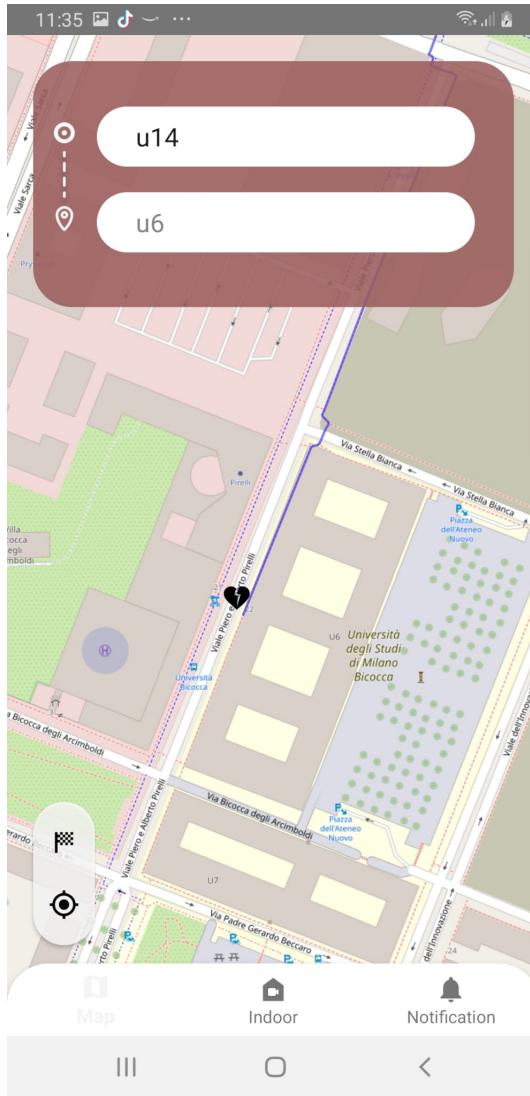


Figura 23: Routing

Attraverso le barre di ricerca, gli utenti possono inserire l'edificio di partenza e di destinazione, visualizzando un marcitore per la posizione dell'edificio di destinazione.

- Una "barra degli strumenti" laterale che contiene diversi componeti:
  - Un'icona che se premuta, mostra la tua posizione nella mappa
  - Un'icona che una volta premuta mostra la posizione dell'edificio selezionato, utile quando l'utente spostando la visuale nella mappa perde di vista la destinazione.
  - Una volta effettuato uno zoom sull'edificio selezionato, sarà possibile, tramite dei pulsanti numerati, che faranno riferimento al numero di piani, visualizzare la mappa del piano desiderato, come in Figura 24. Inoltre, se si volessero maggiori informazioni sull'edificio o se si volesse passare alla modalità per la navigazione indoor, vi è un icona apposita che una volta premuta porterà l'utente alla visualizzazione del FragmentIndoor

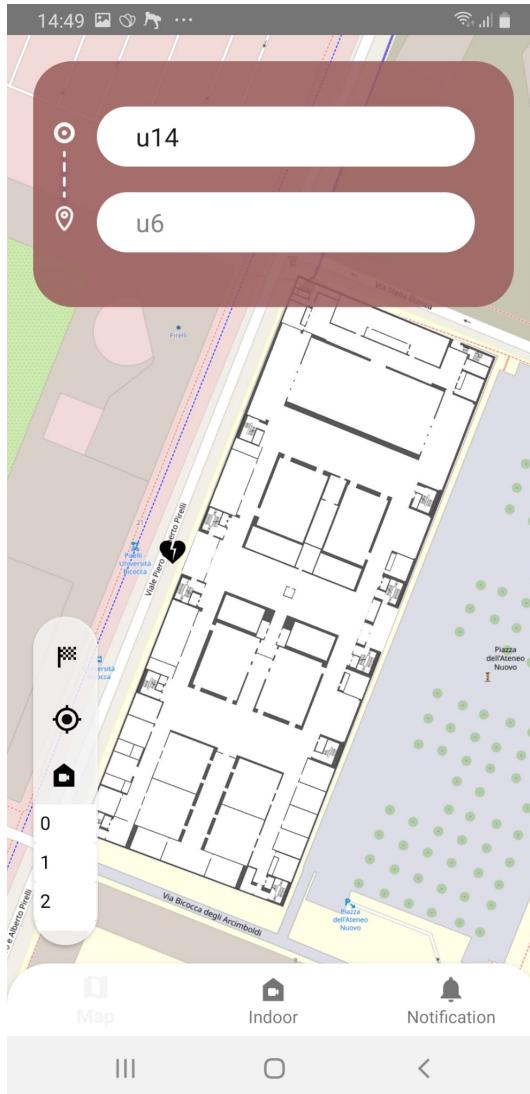


Figura 24: Change Floor

## Fragment Indoor

Il FragmenIndoor, Figura 25, è un UI Element responsabile della visualizzazione dei dati relativi alla mappa interna degli edifici e alla visualizzazione dei percorsi selezionati dagli utenti. Inoltre, è responsabile della raccolta dei dati, relativi alla posizione iniziale e di arrivo da e verso cui gli utenti vogliono "navigare".

All'interno di questo fragment l'utente potrà trovare:

- due caselle di testo, una per l'inserimento del punto di partenza, mentre l'altro per l'inserimento del punto di destinazione.

- un bottone "Show path" che una volta selezionati punti di arrivo e destinazione validi, sarà in grado, una volta premuto, di mostrare il percorso desiderato.
- un bottone "Show step" che mostrerà in step più brevi il percorso che l'utente dovrà percorrere.
- 2 bottoni situati nella parte bassa del fragment utili a cambiare la visualizzazione del piano dell'edificio attualmente mostrato.

Il FragmentIndoor è un UI controller che delega la logica relativa al calcolo del percorso alla classe Graph e il disegno del percorso su mappa alla classe MapDrawer.

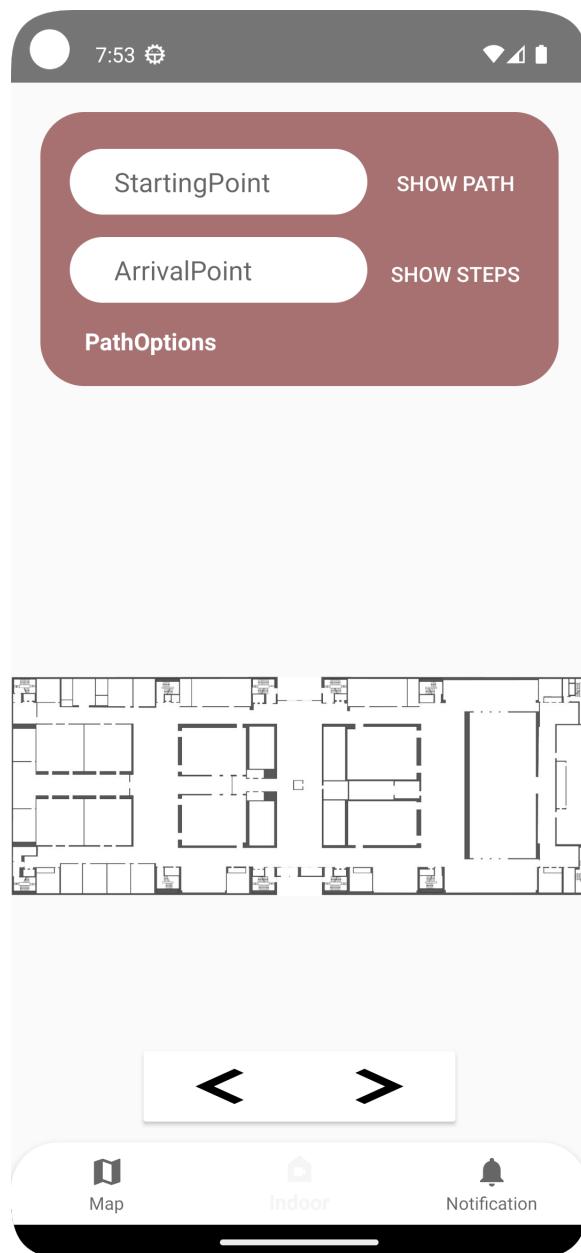


Figura 25: FragmentIndoor

## Fragment Notification

All'interno di questo fragment, Figura 26, l'utente potrà trovare:

- Informazioni generali (Location to App Version): queste informazioni andranno a mostrare all'utente quali servizi sono stati abilitati e quali no, così da rendere partecipe l'utente di eventuali servizi che verrano preclusi proprio per la non disponibilità di quest'ultimi. Location invece andrà a mostare l'edificio in cui sarà l'utente, se abbastanza vicino ad esso.
- Impostazioni per percorsi personalizzati: sono dei buttoni che riguardano i vari tipi di percorsi personalizzati, e servirà per capire, come andare a precludere determinati percorsi riguardanti il grafo per il calcolo del routing interno.
- Change routing esterno: sono dei buttoni che riguardano il cambio di routing esterno in base al mezzo scelto.

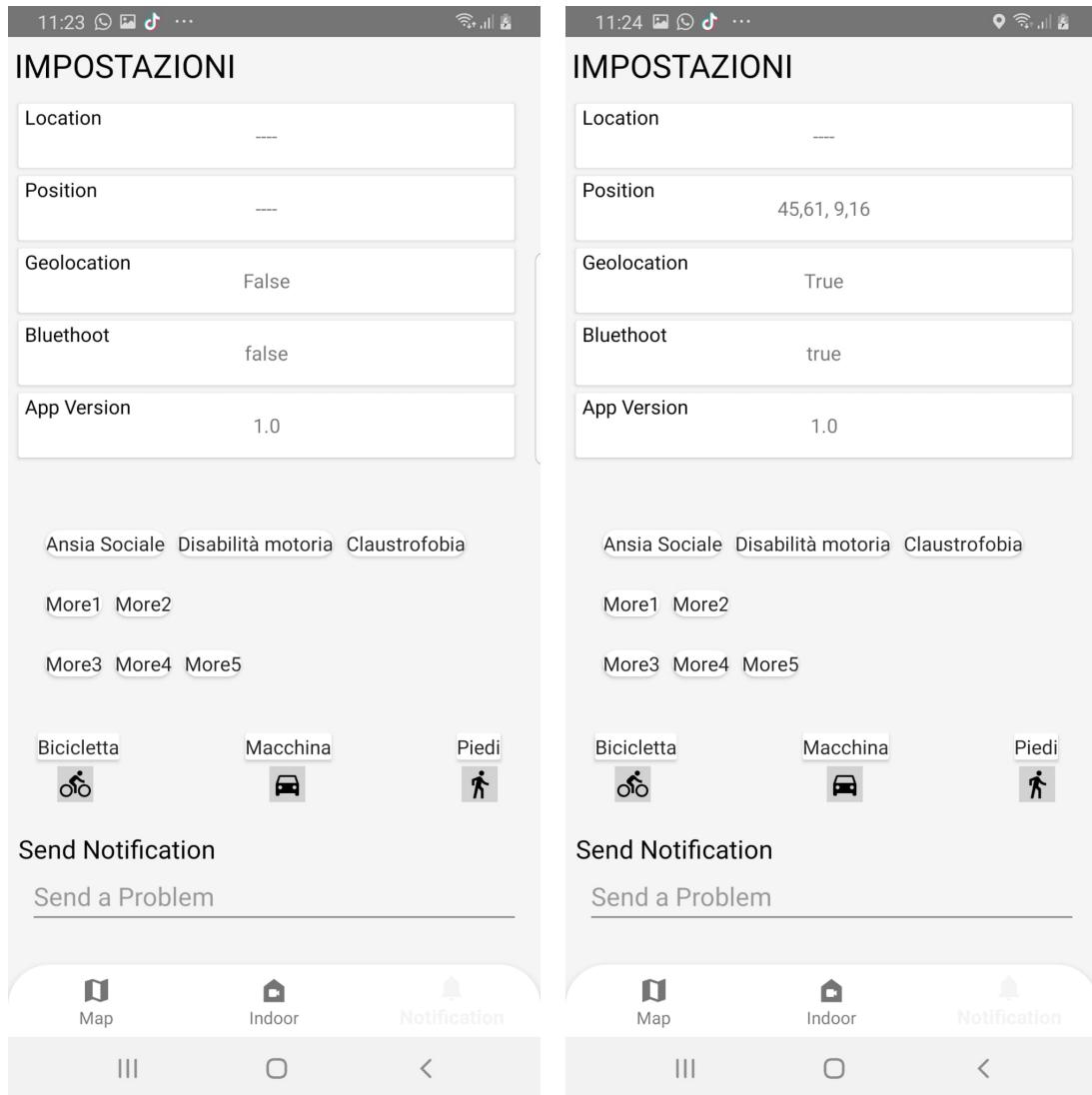


Figura 26: prima e dopo l'accesso ai vari servizi

#### 4.3.1 Relazioni tra i diversi componenti

In questo capitolo andremo a mostrare come andranno a collegarsi e che tipo di informazioni verranno passate tra i diversi fragment e classi.

La navigazione tra i fragment ,descritti nel capitolo precedente, avverrà grazie al :"Navigation Component", una risorsa XML che conterrà tutte le informazioni relative alla navigazione quindi le destinazioni e le azioni (possibili percorsi). Fornisce animazioni e transizioni gestendo in autonomia i diversi cicli di vita dei fragment. In questo caso sarà la bottomNavigationView a prendersene carico.

## Fragment Notification to Fragment Map

Tramite i bottoni che riguardano il routing esterno del Fragment Notification, possiamo andare a modificare il calcolo del path riguardante il fragment map, come mostrato in Figura 27.

L'informazione verrà salvata tramite le "SharedPreferences", così da mantenerla salvata nel tempo come coppia di chiave e valore.

In questo modo anche se uscissi dall'applicazione questa informazione verrebbe salvata e verrebbe utilizzata per il calcolo del routing, per i successivi accessi.

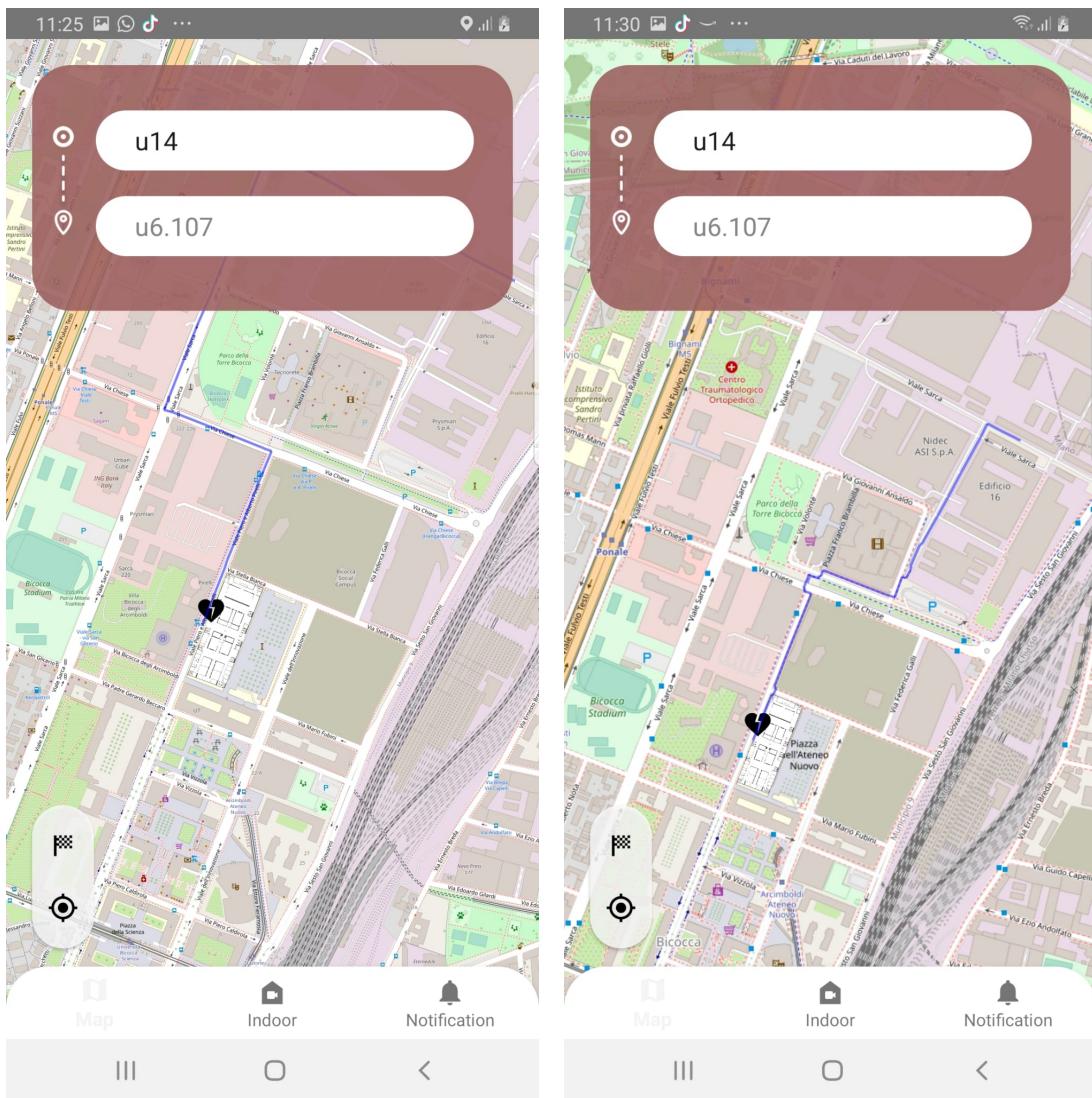


Figura 27: Percorso in macchina e poi a piedi

## Fragment Map to Fragment Indoor

Il fragment map andrà a passare le informazioni riguardanti l'edificio di destinazione oppure l'edificio di partenza in base allo zoom relativo all'edificio stesso, come mostrato in Figura 28.

Permette al fragment indoor di creare un routing in base alle informazioni date dal fragment map, senza dover ricercare i punti selezionati.

Le informazioni verranno passate tramite il Navigation Component e non più dalle shared preferences così da evitare di salvare le informazioni al riavvio dell'applicazione.

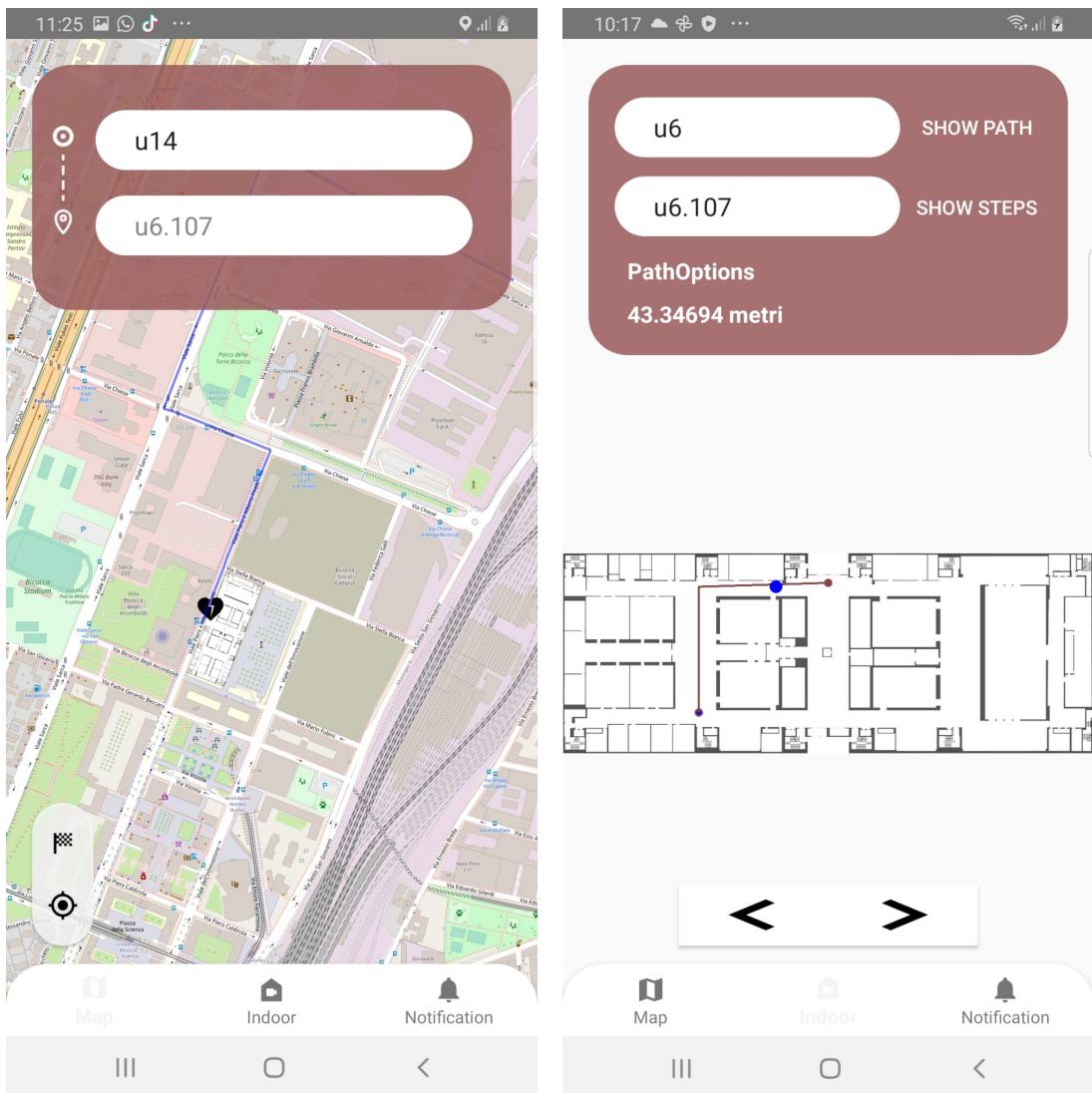


Figura 28: Routing interno automatico

### Fragment Notification to Fragment Indoor

Il fragment notification andrà a passare le informazioni riguardanti i vari disturbi dell'utente.

Permette al fragment indoor di creare un routing in base alle informazioni date dal fragment notification.

Le informazioni verranno salvata tramite le "SharedPreferences" così da mantenerla salvata nel tempo, come coppia di chiave e valore.

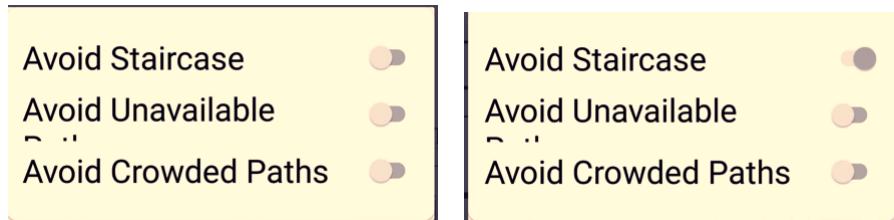


Figura 29: Prima e dopo aver cliccato disabilità motoria

In questo caso, dopo aver cliccato il tag: "disabilità motoria", andremo a precludere i path che avranno delle scale nell'intermezzo, come in Figura 29.

## 5 Conclusioni e Sviluppi Futuri

La navigazione indoor propone molte sfide complesse e differenti rispetto alla navigazione outdoor.

La creazione di mappe dettagliate e precise richiede un notevole sforzo e risorse, dato che gli ambienti interni possono essere complessi e in continua evoluzione. La presenza di ostacoli come muri, corridoi intricati, scale, ascensori e cambiamenti strutturali richiede soluzioni innovative per garantire una navigazione accurata e affidabile.

A differenza della navigazione outdoor, dove Google Maps ha consolidato la sua posizione dominante come principale provider di servizi di navigazione, la navigazione indoor non ha ancora un competitore principale che abbia raggiunto lo stesso livello di riconoscimento e copertura globale. Questo scenario apre le porte a molteplici aziende che si contendono il mercato, cercando di sviluppare soluzioni competitive per la navigazione interna.

Alcune delle tecnologie più promettenti, che abbiamo esplorato all'interno della relazione, e che consentono una maggior controllo sul risultato finale sono:

- Navigine: dove grazie al suo SDK e alle sue librerie gratuite, è possibile andare a definire un applicazione per la navigazione indoor.

- OpenStreetMap: dove grazie alla sua mappa mondiale aggiornabile e richiamabile tramite la libreria OpenSource OsmDroid è possibile andare a mostrarla nelle applicazioni android, con le relative funzionalità.

OSM è un progetto molto all'avanguardia e soprattutto molto popolare tra gli utenti, con una community molto partecipe, sia per quanto riguarda OSM stesso che per quanto riguarda il suo sviluppo su applicazioni: android, web etc...

Infatti queste problematiche sono state riscontrate da molti utenti, portando il problema in primo piano (per quanto riguarda la renderizzazione).

Esistono comunque diverse soluzioni implementative, nel nostro caso si è scelto di implementare i GroundOverlay col nostro personale algoritmo di routing per quanto riguarda l'indoor navigation.

Inoltre, è interessante notare che con il continuo sviluppo delle tecnologie cartografiche, come OpenStreetMap (OSM), è possibile che nel futuro emergano nuove soluzioni per il rendering automatico dei dati cartografici. Queste tecnologie potenzialmente potrebbero semplificare il processo di visualizzazione delle mappe e migliorare la qualità delle informazioni geografiche fornite. Pertanto, sarà interessante esplorare ulteriormente come le future evoluzioni delle tecnologie OSM potrebbero influire sul nostro lavoro e fornire nuove opportunità di analisi e visualizzazione dei dati. La prospettiva per il futuro della nostra applicazione, invece, sarebbe l'integrazione di sensori per quanto riguarda il posizionamento e l'affluenza.

Nel nostro caso pensavamo di collegare alla nostra applicazione dei Beacon, così da mostrare il posizionamento dell'utente.

E utilizzare dei sensori di prossimità o di pressione per capire l'affluenza riguardante dei punti chiave del percorso.

In conclusione, la navigazione indoor rappresenta una sfida affascinante, che richiede sforzi significativi per mappare accuratamente gli edifici e fornire indicazioni precise agli utenti.

Nonostante l'assenza di un leader indiscusso nel settore, l'ampiezza della competizione indica un crescente interesse e impegno nel fornire soluzioni innovative. Il futuro della navigazione indoor promette sviluppi sempre più avanzati e affascinanti, offrendo agli utenti un'esperienza di navigazione fluida e senza intoppi all'interno degli edifici.

## Riferimenti bibliografici

- [1] GraphHopper. The graphhopper directions api route planning for your application.  
<https://www.graphhopper.com/developers/>.
- [2] Coding in flow. Room + viewmodel + livedata + recyclerview (mvvm).  
<https://www.youtube.com/watch?v=ARpn-1FPNE4>.
- [3] Comunità italiana di OpenStreetMap. Introduzione a openstreetmap.  
[https://wiki.openstreetmap.org/w/images/1/19/Libretto\\_introduzione\\_a\\_OpenStreetMap\\_2017\\_09.pdf](https://wiki.openstreetmap.org/w/images/1/19/Libretto_introduzione_a_OpenStreetMap_2017_09.pdf).
- [4] learnosm. Primi passi con josm. <https://learnosm.org/it/josm/start-josm/>.
- [5] MapBox. Unlock geospatial analysis in snowflake with mapbox.  
<https://www.mapbox.com/snowflake-app>.
- [6] MapsPeople. Innovative indoor mapping that's tailored to your business.  
<https://www.mapspeople.com/>.
- [7] OpenStreetMap. Openlevelup. <https://wiki.openstreetmap.org/wiki/OpenLevelUp>.
- [8] B. Raphael P. E. Hart, N. J. Nilsson. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE, 1968.
- [9] Ravikiran A S. A\* algorithm concepts and implementation.  
<https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm>.
- [10] Android Studio. Guide to app architecture. <https://developer.android.com/topic/architecture>.
- [11] Ronald L. Rivest Clifford Stein Livio Colussi Thomas H. Cormen, Charles E. Leiserson. *Introduzione agli algoritmi e strutture dati*. McGraw-Hill Education, 2010.
- [12] Unity. Accessing the camera image on the cpu.  
<https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/cpu-camera-image.html>.
- [13] Unity. Inner workings of the navigation system.  
<https://docs.unity3d.com/Manual/nav-InnerWorkings.html>.
- [14] Unity. Texture2d.getpixels32.  
<https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/cpu-camera-image.html>.
- [15] From Wikipedia. Graphhopper. <https://en.wikipedia.org/wiki/GraphHopper>.