POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Composizione di guide multimediali e context-aware per visite culturali personalizzate

Relatore
prof. Daniele Jahier Pagliari

Tutor Aziendali
Paola Dal Zovo, Francesco Cuscito

Candidato
Lorenzo Cannizzaro

Anno Accademico 2020/2021

A Mamma e Papà.

Ricorda di guardare in alto alle stelle, e non ai tuoi piedi.
Cerca di dare un senso a quello che vedi e chiediti quello che fa vivere l'universo.
Sii curioso.
Stephen Hawking

Ringraziamenti

Grazie agli amici di famiglia per l'unione indissolubile, nel ricordo della nostra Annamaria.
Grazie alle mie sorelle e a mio cognato per la pazienza.
Grazie alle mie nonne per i perenni sorrisi.
Grazie a tutti i miei amici per la spensieratezza e la fratellanza.
Grazie ai miei genitori per l'esempio e il supporto.
Grazie alla mia ragazza per l'amore e la felicità.

Sommario

Negli ultimi anni, con l'avanzamento della trasformazione digitale del paese in vari settori, abbiamo assistito alla nascita di progetti di varia natura e con diversi obiettivi. In particolare, una forma di digitalizzazione che è diventata sempre più popolare negli ultimi anni è quella relativa ai contenuti culturali, molto spesso presenti soltanto in maniera "analogica", oppure già digitalizzati ma con uno scarso supporto nell'ottica della fruizione verso l'utente finale.
Scopo di questa tesi è illustrare il mio lavoro all'interno dello sviluppo di un'applicazione mobile cross-platform per VASARI, un progetto avviato verso la fine del 2018 cofinanziato dal MIUR con il sostegno dell'Unione Europea. Il progetto VASARI (VAlorizzazione Smart del patrimonio ARtistico delle città Italiane) ha come obiettivo quello di creare una piattaforma digitale in grado di valorizzare il patrimonio culturale ed artistico del panorama italiano (il più grande a livello mondiale) e avvicinarne le persone, offrendo un'esperienza innovativa e del tutto immersiva. In particolare, all'interno di questa tesi, verrà descritto il modulo di Smart Visit, un macroservizio di VASARI che dovrà permettere al visitatore di ottenere i classici servizi che si potrebbero ottenere da una guida turistica tradizionale. Questi includono l'accompagnamento e la navigazione lungo il sito culturale (indoor e outdoor), e la fruizione culturale stessa tramite modelli orientati allo storytelling e all'attivazione di contenuti multimediali e suggerimenti che possono migliorare l'esperienza del visitatore.
Nel corso dello sviluppo dell'applicazione mobile, sono state effettuate delle analisi riguardo la User Interface e la User Experience (UI/UX). Inoltre, sono stati confrontati diversi approcci riguardo la facilità di sviluppo, le funzionalità e l'usabilità finale dell'applicazione. Ad esempio, alcune delle scelte di design discusse nella tesi riguardano l'utilizzo di un framework cross-platform piuttosto che uno nativo, e la scelta della metodologia più appropriata nell'ambito del posizionamento utente.

Indice

1 Il progetto VASARI e il macroservizio Smart Visit 1
1.1 VASARI . . . 1
1.1.1 Obiettivi e funzionalità . . . 1
1.1.2 Infrastruttura globale . . . 1
1.2 Smart Visit . . . 2
1.2.1 Obiettivi e funzionalità . . . 2
1.2.2 Stato dell'arte di applicazioni simili . . . 3
1.2.3 Differenze proposte in base al contesto applicativo . . . 5
1.3 Sviluppo dell'applicazione mobile . . . 7
1.3.1 Analisi della User Interface (UI) . . . 7
1.3.2 Analisi della User Experience (UX) . . . 8
1.3.3 Confronto tra sviluppo nativo e cross-platform . . . 9

2 Strumenti utilizzati 13
2.0.1 Angular . . . 13
2.0.2 Ionic . . . 18
2.0.3 Plugin nativi Cordova/Capacitor . . . 22

3 Ricerca di punti di interesse e gestione dei percorsi 25
3.1 Analisi UI/UX . . . 25
3.1.1 Ricerca di punti di interesse . . . 25
3.1.2 Creazione e gestione di bozze . . . 26
3.1.3 Generazione e gestione dei percorsi . . . 27

4 Guida del turista lungo il percorso fisico 35
4.1 Descrizione del formato e del contenuto dei dati . . . 35
4.1.1 JSON per informazioni sui punti di interesse . . . 35
4.1.2 GeoJSON per la modellazione del sito culturale . . . 37
4.2 Librerie utilizzate . . . 38
4.2.1 Leaflet . . . 38
4.2.2 Turf . . . 39
4.2.3 TensorFlow . . . 40
4.3 Analisi UI/UX . . . 40
4.3.1 La mappa . . . 40
4.3.2 Stato della visita . . . 45
4.4 Posizionamento . . . 47
4.4.1 Outdoor . . . 47
4.4.2 Indoor . . . 49
4.5 Navigazione . . . 64
4.5.1 Algoritmo A* . . . 65
4.5.2 Definizione della griglia e degli ostacoli . . . 66
4.5.3 Indicazioni tramite frecce . . . 67

4.5.4 Curve di Bèzier 68

5 Fruizione di contenuti culturali 71
5.1 Analisi UI/UX 71
5.1.1 Contenuti culturali 71
5.1.2 Galleria multimediale 78
5.1.3 Interazione con l'utente 80

6 Conclusioni 87

Elenco delle figure 89

Elenco delle tabelle 93

Riferimenti bibliografici 94

Capitolo 1

Il progetto VASARI e il macroservizio Smart Visit

1.1 VASARI

1.1.1 Obiettivi e funzionalità

L'obiettivo di VASARI è quello di fornire un insieme di servizi in grado di valorizzare e gestire il patrimonio artistico su una base sovra-museale, con una fruizione dei contenuti che può essere classica (contenuti testuali e/o utilizzo di audioguide), o moderna (contenuti multimediali e in realtà aumentata). Permette inoltre la personalizzazione della propria fruizione culturale (anche in ottica multisito), e grazie a ciò, al suggerimento di percorsi in linea con i gusti e le aspettive dell'utente.

1.1.2 Infrastruttura globale

Da un punto di vista tecnico e architetturale, per lo sviluppo di questa piattaforma si è pensato di procedere con un approccio "scalable by design", attraverso i più moderni paradigmi di Edge e Fog Computing, oltre che Internet of Things (IoT). Partendo da una definizione di spazio culturale che concerne la fisicità di un luogo in cui sono presenti dei punti di interesse, si è cercato di interconnettere con una logica multi-sito questi luoghi e questo insieme di beni culturali, in cui al centro di tutto vi è il visitatore. Per cui, vi è una dicotomia tra "spazio virtuale" e "spazio fisico": nel primo vengono forniti dei macroservizi, prevalentemente digitali, riguardanti gli oggetti culturali, i loro contenuti, e i loro luoghi, mentre nel secondo si ha la classica fruizione culturale (che si appoggia però su questa infrastruttura).
L'idea guida di VASARI è quella di unificare l'intero patrimonio artistico e culturale italiano in una struttura digitale, per poi, con una logica sovra-museale e complesse relazioni logiche tra le singole opere, personalizzare la fruizione per ogni singolo utente. Esistono perciò dei macroservizi in grado di fornire un supporto per una visita multi-sito (ad esempio generando percorsi all'interno della stessa città, sempre in relazione al profilo utente) o spettacolarizzare la visita (tramite contenuti in realtà aumentata e virtuale). Oltre che macroservizi per i visitatori, sono presenti anche dei macroservizi per gli operatori. Quest'ultimi, grazie all'ausilio di microservizi posizionati in-loco, forniscono dettagli globali come il crowd management (l'intensità della folla attorno un particolare punto di interesse o sito culturale), o analytics di utenti e opere. Di particolare rilevanza è il visitatore: oltre ad essere il punto cardine su cui si fonda lo scopo di VASARI, è pure uno degli attori in gioco più importanti.
Grazie al flusso della propria visita, all'interazione con l'ambiente circostante e ai propri feedback, permette all'infrastruttura globale di avere un livello di personalizzazione per ogni singolo utente. Il tutto, è accompagnato da principi di sicurezza dei dati e di privacy (per quanto riguarda l'utilizzo di dati dei sensori, di informazioni personali e sensibili).

In Figura 1.1, tratta da [33], è riportata l'architettura globale del sistema VASARI, in cui è presente anche il macroservizio che funge da caso di studio per questa tesi, ossia Smart Visit.

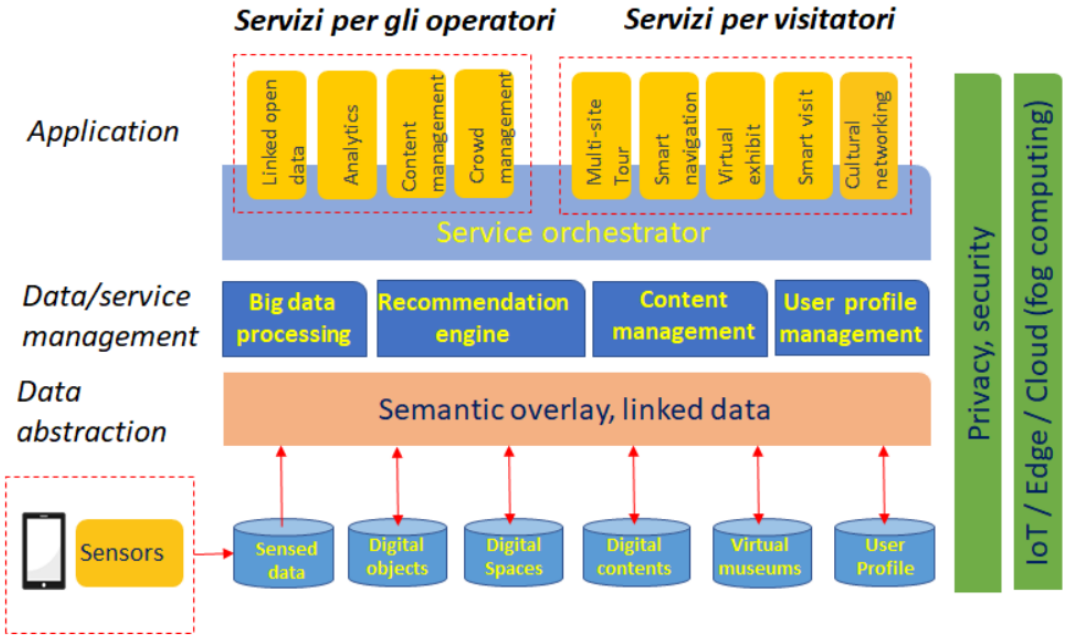

Figura 1.1: Architettura del progetto VASARI [33]

1.2 Smart Visit

1.2.1 Obiettivi e funzionalità

Le funzionalità di Smart Visit si articolano in 3 differenti task all'interno dell'applicazione VASARI: definire il percorso di cui si vuole usufruire all'interno di un particolare sito culturale, accompagnare l'utente lungo questo percorso e fornigli i vari contenuti informativi, testuali e multimediali, e collegamenti tematici. Per quanto riguarda la definizione del percorso, verrà data possibilità all'utente di cercare opere, beni culturali e siti/musei, sfruttando l'enorme mole di contenuti messa a disposizione dalla piattaforma DatabencArt. Da qui, si potranno generare automaticamente i percorsi di cui successivamente si usufruirà per la visita on-site. Verranno anche proposti una serie di percorsi predefiniti e all'occorrenza personalizzati. Quando il visitatore si troverà all'interno del sito culturale, potrà avviare la propria esperienza di visita: l'applicazione lo guiderà alla scoperta dei vari punti di interesse, dando un feedback sulla propria posizione all'interno di una comoda pagina contenente una mappa con la raffigurazione del sito culturale e fo rnendo la locazione fisica dei vari punti di interesse (secondo l'ordine del percorso scelto in precedenza). L'utente, tramite il proprio senso dell'orientamento, o tramite un comodo sistema di guida implementato nella suddetta pagina, dovrà completare il proprio percorso per terminare la visita di quello specifico sito, per poi, in caso di tour multisito, procedere verso il prossimo.
Questa funzionalità di Smart Visit, nel contesto globale di VASARI, potrà essere reiterata all'interno di un percorso multisito, il cui studio e implementazione non è oggetto di questa tesi. Nel corso del flusso dell'applicazione, l'utente riceverà, oltre che informazioni relative ai

punti di interesse visitati, anche delle informazioni sul percorso, lo stato della visita e informazioni relative al sito culturale che si sta visitando. Infine, tramite l'utilizzo di uno scanner integrato nell'applicazione, l'utente potrà inquadrare dei codici QR per aiutarsi nella navigazione o usufruire dei contenuti di punti di interesse al di fuori del proprio percorso. Il flusso logico del macroservizio Smart Visit è mostrato in Figura 1.2.

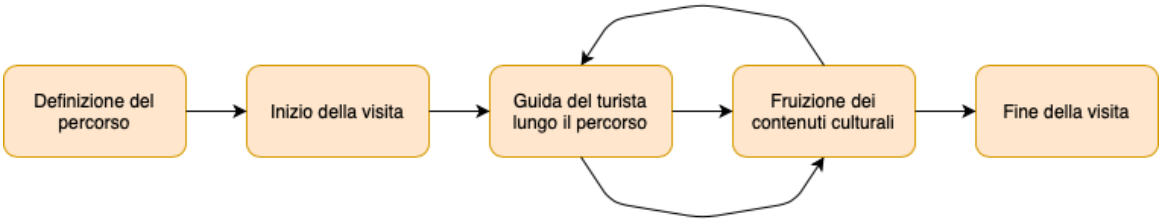

Figura 1.2: Flusso logico del modulo di Smart Visit

1.2.2 Stato dell'arte di applicazioni simili

Nel corso degli ultimi anni, vi è stata una significativa quantità di ricerche relative ad applicazioni, in ambito culturale e turistico, di varie tecnologie e metodi di trasmissione di patrimonio culturale tramite esse. Tra i principali strumenti, troviamo soprattuto i palmari (noti anche come PDA, Personal Digital Assistant) e, in tempi più recenti, gli smartphone. In [14] vengono esaminati una serie di casi reali che portano, in conclusione, ad alcuni requisiti che bisogna soddisfare per avere successo (in termini di soddisfazione dell'utente, oltre che dal punto di vista economico) nello sviluppo di applicazioni culturali su PDA e smartphone. Il primo punto cardine è senza dubbio la "dinamicità" della presentazione e della fruizione culturale: le soluzioni di accompagnamento e guida dell'utente lungo i siti culturali avveniva prevalentemente mediante info-points, sistemi di proiezione (ad esempio degli schermi lungo il percorso) o audio-guide fornite all'ingresso. Il tutto era però limitato dallo spazio fisico del museo, o dal ristretto utilizzo di una sola fonte di informazioni, ossia tramite audio. In breve tempo è apparso palese come il turista abbia bisogno di arricchire la propria esperienza, rimanendo però focalizzato sul proprio percorso. Perciò, nel 1997, lo Smithonian Istitute cominciò a fornire dei computer portatili con cui ottenere informazioni testuali e narrative su circa 90 oggetti culturali, per poi estendere il tutto anche a suoni, immagini, e tecnologie location-aware. [14]
Per quanto l'accompagnamento fisico dell'utente, si cominciò a sperimentare su metodi di posizionamento indoor basati su beacon a infrarossi (il progetto Hyper Interaction with Physical Space, HIPS [14]). Un altro passo in avanti per quanto riguarda le applicazioni di ambito culturale, fu il progetto PEACH [14]: si trattava di un'iniziativa che integrasse l'infrastruttura esistente di un museo con dei dispositivi mobile. L'articolo di ricerca parla di "musei intelligenti", ossia di interazione tra dispositivi (fissi o mobile) con la varietà di servizi offerti in un contesto incentrato sull'utente, come quello di un sito culturale. L'utilizzo di questi sistemi in un contesto outdoor permette di fornire contenuti che di per se non ci sarebbero (testuali e multimediali), mentre in un contesto indoor di semplificarne l'utilizzo, senza ricorrere a guide cartacee, chioschi di informazioni, e tutta una serie di "distrazioni" dalla vera e propria fruizione culturale.
Per il corretto design di questo tipo di applicazioni, in [14] sono elencati tutta una serie di fattori non banali, che possono anche far trapelare nuove problematiche. Senza dubbio, la prima è riguardante il contesto applicativo, ossia quello relativo ad un dispositivo mobile: l'applicazione deve essere progettata per favorire la mobilità, con tutti i fattori che ne derivano di multimodalità (poter eseguire la stessa operazione in più modi e facilmente), di limitata potenza di calcolo e di ristrette metodologie di input e output. L'applicazione deve

essere progettata per garantire la fruibilità da parte di varie tipologie di utenti, ognuna con un proprio livello di familiarità con lo strumento, e deve garantire una serie di informazioni basate sulla posizione dell'utente. Per la trasmissione delle informazioni, bisogna infine considerare il limitato spazio a disposizione dello schermo del device. Infatti, a differenza dei PDA (con grandezze più o meno simili), lo sviluppo dell'applicazione deve tenere conto che ci potrebbero essere 3 categorie di dispositivi: di piccole, medie e grandi dimensioni. Uno dei più grandi problemi rimane però il fatto che una non trascurabile percentuale di utenti utilizzi dispositivi obsoleti, che potrebbero compromettere l'utilizzo dell'applicazione nel momento in cui essa richieda troppa potenza computazionale, quantità di memoria o supporti grafici. Inoltre, alcuni contenuti multimediali potrebbero non essere abilitati del tutto. In generale, gli smartphone possiedono meno capacità di calcolo dei PDA, perciò le applicazioni che vengono eseguite su uno smartphone devono essere altamente ottimizzate e raggiungere velocemente l'obiettivo rispettando i requisiti di usabilità e impatto visivo verso i vari utenti.
Per quanto riguarda le funzionalità di più alto livello, l'applicazione deve fornire dei feedback grafici al visitatore, riguardanti le proprie azioni e lo sviluppo della propria visita, e le informazioni fornite devono essere corte e compatte (soprattutto quando si parla di informazioni testuali). Infine, eventuali informazioni uditive, non devono estraniare troppo il visitatore, ma devono essere invece di accompagnamento e arricchire il percorso culturale. Il design di una UI piacevole da vedere e da usare è veramente importante, ma non deve andare a discapito dell'usabilità e dell'intuitività nell'utilizzo: in [14] viene consigliato perciò un design minimale ma piacevola alla vista, tramite il quale è possibile accedere in maniera semplice alle varie informazioni. Infine, dal punto di vista funzionale, l'applicazione deve supportare più lingue e deve adattarsi al contesto di utilizzo (ad esempio, potrebbero esserci situazioni di mancanza di connettività per cui alcune funzionalità dovrebbero essere sempre presenti).
In [5], in cui viene preso in considerazione un caso di studio reale di un'applicazione mobile per la promozione del turismo a Malta, viene discusso il possibile utilizzo di un mobile website piuttosto che di un'applicazione (che deve essere sviluppata nativamente o multipiattaforma). Sebbene l'ottimizzazione di un sito web sia tendenzialmente più semplice (e con strumenti più semplici e mainstream) e sia di per se indipendente dalla piattaforma (e da politiche di pubblicazione, in quanto non bisogna pubblicare l'app in uno Store), l'usabilità per l'utente finale risulta sempre migliore tramite un'applicazione mobile. Jakob Nielsen, un ricercatore danese che ha conseguito il dottorato in design dell'interfaccia utente, sostiene infatti che "Un'applicazione può raggiungere le specifiche limitazioni e abilità di ogni singolo dispositivo meglio di un sito web che opera all'interno di un browser" [5]. Infatti, un'applicazione mobile può fare un utilizzo migliore di funzionalità derivanti di GPS, bussola e fotocamera. In questo articolo, si consiglia anche di procedere con un approccio cross-platform, in quanto la totalità degli utilizzatori di smartphone si divide fondamentalmente tra due sistemi operativi: Android e iOS.
Nello sviluppo dell'applicazione di Malta, sono descritti diversi tipi di contenuti con cui si ha a che fare: testo, immagini, audio, video e mappe. Per quanto riguarda il testo, viene detto che la regola d'oro è il mantererlo più corto possibile (come d'altronde espresso anche in precedenza). Solitamente infatti le persone, quando utilizzano un'app, piuttosto che leggere preferiscono scansionare [5]: tendono cioè a scrollare il testo finchè non arrivano ad un punto che può catturare l'attenzione, da cui comincia la lettura vera e propria. Testi lunghi sono consigliati solo per dispositivi con grandi schermi come i tablet (che sono utilizzati più in contesti domestici che in mobilità). Le immagini soffrono dello stesso problema del testo, ossia della dimensione dello schermo. Un'immagine troppo dettagliata, oltre a pesare per la sua risoluzione (sul device o durante la trasmissione in rete), può anche essere fondamentalmente inutile, perché i tanti dettagli non sarebbero così vistosi in piccoli dispositivi. Perciò si consiglia la possibilità di allargare all'occorrenza un'immagine in una pagina apposita o semplicemente mostrarla a schermo intero. Immagini con un elevato contrasto sono preferibili per focalizzare l'utente su determinati elementi. Parlando di audio, esso è un mezzo eccellente per comunicare informazioni e soddisfare l'esperienza dell'utente, per cui non dovrebbero mai

mancare in applicazioni di ambito culturale, artistico e turistico. Bisogna tenere in mente però che l'audio stesso dovrebbe essere di buona qualità (nitido, piacevole e senza sottofondi musicali che possano coprire la voce). Inoltre, situazioni ambientali sfavorevoli (un museo particolarmente affollato) potrebbero comprometterne l'utilizzo. I contenuti video soffrono degli stessi problemi delle immagini, ma hanno gli stessi benefici degli audio. Video corti e concisi sono solitamente ben apprezzati, mentre video lunghi sono da favorire per un utilizzo off-site. Infine, uno degli elementi principali è senza dubbio l'utilizzo di una mappa. Oltre ad essere una delle caratteristiche di più successo all'interno di vari tipi di applicazioni, l'ambito culturale si sposa a perfezione con questo strumento. Grazie alla mappa, possono essere rappresentate delle informazioni di grande utilità, come la posizione dei punti di interesse o la propria e avere un feedback sul proprio movimento e quindi sul proprio orientamento all'interno del percorso. Una significativa sezione di questa tesi verterà sull'analisi UI/UX (User Interface / User Experience) nell'utilizzo della mappa, essendo un elemento fondamentale all'interno del modulo Smart Visit dell'applicazione VASARI. Nell'articolo si parla anche di caratteristiche "social": l'utente potrebbe trarre vantaggi o semplicemente piacere nel condividere alcuni contenuti sui vari Social Network o in privato con altre persone.
Infine, vengono consigliate alcune buone norme di progettazione dell'app, dal punto di vista della navigazione e dei menù. Una regola generale è rendere intuitivo il primo aspetto, mediante per esempio l'utilizzo di pulsanti "back", e per quanto riguarda il secondo, mantenersi sull'approccio moderno del menù laterale (che in generale permette di rendere più modulari le varie sezioni dell'applicazione mobile). Le interfacce utente devono essere semplici e intuitive, ed è importante sottolineare come le dita dell'utilizzatore possano essere imprecise nel tocco su schermo. Perciò, bisogna rendere "clickable" tutta l'area circostante ad esempio un'immagine (se si vuole mostrarla a tutto schermo), e fare in modo che i vari pulsanti siano raggiungibili facilmente e senza intromissione da parte di altri elementi dell'interfaccia. Inoltre, aspetto fondamentale, è considerare che l'applicazione potrebbe essere utilizzata anche da persone anziane o affette da qualche tipo di disabilità, per cui eventuali scritte e indicazioni in-loco devono avere un font e una grandezza adeguata, oltre che con una colorazione non fastidiosa e che garantisca una corretta visibilità anche in condizioni di scarsa illuminazione. Nelle figure 1.3, 1.4, 1.5 e 1.6 vengono mostrate le principali sezioni dell'applicazione culturale descritta in [5].

1.2.3 Differenze proposte in base al contesto applicativo

Paragonando l'esempio fornito da [5] con i requisiti del modulo Smart Visit all'interno dell'applicazione per il progetto VASARI, vi sono numerose analogie. Anche nella sezione di app sviluppata in questo lavoro di tesi, vi è una fase di accompagnamento del turista che si è deciso di implementare tramite l'utilizzo di una mappa, che mostra informazioni globali come la posizione dei punti di interesse o dell'utente stesso, ma anche informazioni dipendenti dallo stato della visita (come il prossimo punto di interessa di visionare, o informazioni geospaziali relative). Una differenza sostanziale su questa parte è però parte del contenuto stesso della mappa: mentre nell'esempio su Malta si stava parlando di una intera città (e perciò la topografia classica della città bastava a rappresentarla), in Smart Visit si può anche avere a che fare con dei siti al chiuso, per cui è necessario rappresentare la pianta o la mappa del suddetto luogo. Da ciò, deriverà anche il metodo di posizionamento dell'utente, in quanto la tecnologia GPS è apprezzabile solamente nel momento in cui ci si trovi in uno spazio aperto, mentre in uno spazio chiuso essa risulta essere abbastanza imprecisa e inadatta. Si parlerà successivamente di tutta una serie di metodologie di "positioning", in relazione al caso d'uso (ossia, al singolo tipo di percorso all'interno del singolo tipo di sito culturale). Parlando dei contenuti, la *Card* mostrata nell'esempio citato è rappresentativa: contiene gli elementi necessari alla fruizione del contenuto (titolo, immagine e descrizione). Ispirandosi ad essa, ne verrà creata una del punto di interesse, con in più collegamenti tematici, contenuti multimediali e opzioni di condivisione social, di gradimento tramite feedback, ecc... Verrà aggiunta inoltre

Figura 1.3: Dettagli di un punto di interesse dell'applicazione su Malta [5]

Figura 1.4: Fruizione di un contenuto multimediale dell'applicazione su Malta [5]

Figura 1.5: Guida dell'utente tramite mappa e informazioni geospaziali dell'applicazione su Malta [5]

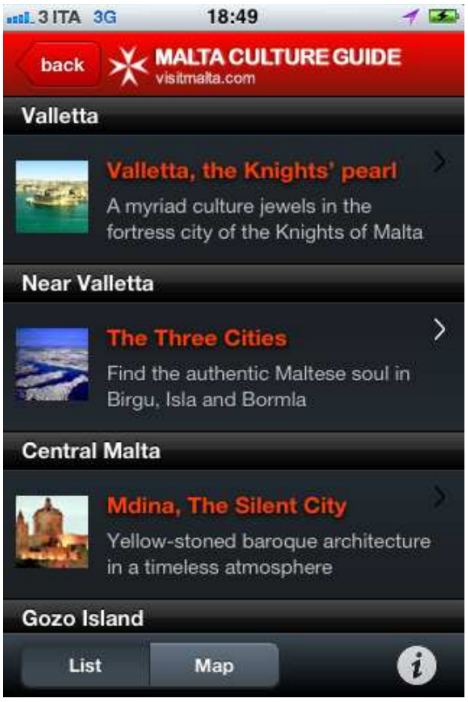

Figura 1.6: Listato di punti di interesse dell'applicazione su Malta [5]

una certa dinamicità al percorso, in maniera tale da poterlo modificare on-the-fly in relazione al desiderio dell'utente.

1.3 Sviluppo dell'applicazione mobile

1.3.1 Analisi della User Interface (UI)

Nella progettazione e nello sviluppo di un'applicazione mobile, è bene tenere presente l'importanza di due fattori che sono strettamente collegati, ma che saranno trattati separatamente. Il primo è la User Interface (UI), ossia l'interazione reciproca, attraverso comandi, input o contenuti, che si ha tra un sistema e l'utente che lo utilizza. Lo sviluppo di questo aspetto, che dalla sua definizione può apparire banale, è tuttavia un lungo processo iterativo [12]: il suo design, a seconda del contesto applicativo, deve essere in continua evoluzione (in quanto lo è il software stesso). In altre parole, a mano a mano che un'applicazione mobile diviene più complessa (con più funzionalità), l'interfaccia grafica stessa segue lo stesso trend.
Molti esperti del settore hanno realizzato come un sistema complesso con una UI piena di elementi funzionali all'applicazione non può essere realizzato in un unico passo. La domanda che ci si fa solitamente, all'interno del design, è "Può essere migliorato facendo ulteriori cambiamenti?" [12].
Nel corso dello sviluppo del modulo di Smart Visit, più volte si è entrati in una fase in cui, con l'aumentare delle funzionalità richieste, si presentava la necessità di arricchire e migliorare l'interfaccia grafica delle varie sezioni dell'app. In Figura 1.7, tratta da [12], è descritto il Flow Chart su cui ci si è basati per lo sviluppo della User Interface delle varie sezioni dell'app. Tra i

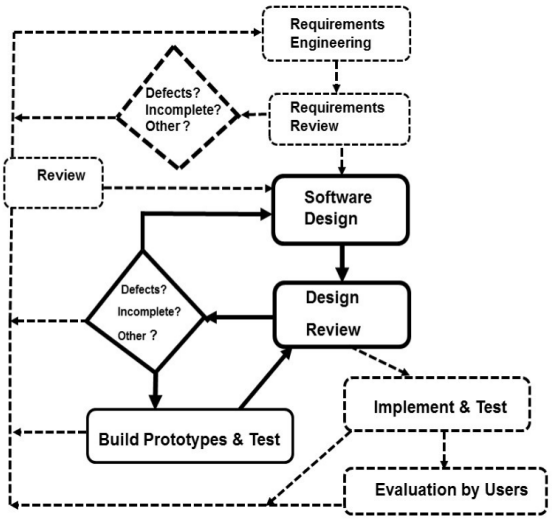

Figura 1.7: Flow Chart dello sviluppo della User Interface [12]

diversi tipi, vi sono le GUI (Graphical User Interface), contenenti icone, pulsanti, menu, e così via, che all'interno di un'app rivestono un ruolo chiave (e gestiscono azioni dirette da parte dell'utente), e le NUI (Natural User Interface) che, mediante input particolari come voce, movimento, gesti e segnali biologici, riescono a capire l'intenzione dell'uomo in maniera più indiretta [26]. L'utilizzo tra questi due tipi di interfacce, globalmente inseriti all'interno del più generale termine UI, sarà visibile quanto, nei capitoli successivi, si parlerà della mappa e del posizionamento dell'utente (NUI), piuttosto che la visione dei contenuti culturali (GUI).
Come viene sostenuto in [26], Dal 2017 ad oggi si è cercato di seguire delle best-practices riguardo il design delle User Interfaces. L'approccio di avere un layout minimalista si è rivelato vincente, in quanto viene focalizzata immediatamente l'attenzione dell'utente verso quelli che sono gli elementi principali di una pagina. Ad esempio, bisogna ridurre il numero di pulsanti al minimo e utilizzare dei *Floating Action Button* che facciano capire quale sia l'azione primaria all'interno di una pagina, o non riempire la view di troppe informazioni che

facciano disorientare l'utente (piuttosto, permettere l'allargamento di testi lunghi e renderli scrollabili). Altri elementi chiave sono l'incremento di "micro interazioni", che unite a immagini in movimento e animazioni, permettono di aumentare l'apparenza di utilizzo di tutti i sensi dell'uomo. I colori devono avere dei toni ricchi, con palette nitide, ma mantenendo un certo livello di contrasto con i caratteri. [26]
L'aspetto che deve essere più chiaro all'interno dello sviluppo è che i designer e gli sviluppatori dell'app, non sono gli usufruitori della stessa: se una cosa ha pienamente senso per uno sviluppatore, non si può dire lo stesso per l'utente medio. Bisogna perciò utilizzare le tecniche più comuni di navigazione, di gestures, di design dei pulsanti e dei metodi di input e le icone più significative (piuttosto che inventarne di nuove). Anche il flusso logico deve essere semplice e coerente con se stesso, in quanto è l'utente che deve approcciarsi a ciò, e non lo sviluppatore che ha alle sue spalle lunghi periodi di tempo di progettazione e integrazione delle varie funzionalità.

1.3.2 Analisi della User Experience (UX)

La User Experience si riferisce, a differenza della User Interface, all'esperienza nel complesso dell'utilizzatore di un sistema, e alla sua percezione, reazione e gradimento nell'utilizzo. Il concetto di UX è strettamente legato a quello di User Interface, perciò molto spesso si parla di entrambi con la sigla UI/UX.
La User Experience di un elemento software ha 4 pilastri fondamentali: il bisogno, l'aspettativa, gli attributi e le capacità. Il bisogno si riferisce alla necessità di andare incontro all'azione da eseguire, l'aspettativa a quello che l'utente effettivamente vorrebbe che accadesse, gli attributi a quelle informazioni di cui si usufruisce, e infine le capacità ai limitati servizi, tempo o immersione nel sistema che l'utente può mettere a disposizione. Vale un principio simile per quanto detto parlando di User Interface, ossia che anche il design dell'esperienza utente è un processo in continua evoluzione, perchè con l'aumentare delle funzionalità implementate, aumenta la necessità di collegare le varie sezione in maniera semplice, consistente, e piacevole per quello che sarà l'utilizzatore finale.
Nel grafico a torta mostrato in Figura 1.8, vengono elencati i fattori da prendere in considerazione per il buon design della UX, tra cui l'interazione, la funzionalità, l'usabilità, la tipografia, la strategia dei contenuti, e così via: La User Experience è legata, tra le tante

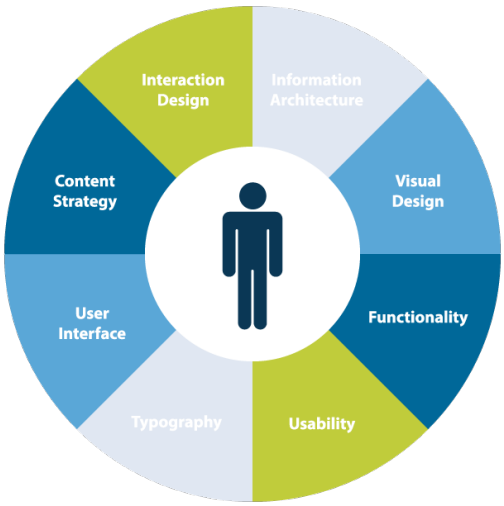

Figura 1.8: Fattori da prendere in considerazione per il design della UX [15]

cose, proprio alla User Interface. Il design dell'interfaccia grafica è fondamentale per la buona riuscita di un prodotto software, in quanto la semplicità del suo utilizzo, unita alle emozioni e feedback che può generare usandola, permettono ad un'applicazione mobile di ottenere il massimo dal punto di vista dell'esperienza utente. All'interno dello sviluppo di Smart Visit, è stato dato grosso peso alla progettazione anche dal punto di vista UI/UX, oltre che dal punto di vista funzionale e logico. In Figura 1.9, tratta da [10], viene mostrata la complementarità tra la User Interface e la User Experience, con i rispettivi elementi caratterizzanti. Lo sviluppo della User Interface incapsula gli aspetti di:

- **Layout**: la disposizione dei vari elementi all'interno della pagina.
- **Branding**: la personalizzazione della pagina secondo lo stile del marchio, per differenziare il prodotto da altri con funzionalità e obiettivi simili.
- **Design**: l'ideazione e la progettazione della pagina, sia dal punto visivo che interattivo.

Mentre, per quanto riguarda la User Experience, essa si concentra su:

- **Storytelling**: nell'ambito di un'applicazione mobile, è la capacità di comunicare e descrivere le funzionalità della stessa in maniera visiva, multimediale o interattiva.
- **Engagement**: la capacità di attrarre l'utente e massimizzare il suo periodo di utilizzo del prodotto.
- **Obiettivi**: rendere chiaro lo scopo di una singola sezione e le funzionalità tramite le quali l'utente può raggiungerlo in maniera semplice e intuitiva.
- **Usabilità**: rendere l'esperienza di utilizzo performante, veloce e immersiva da parte dell'utente.

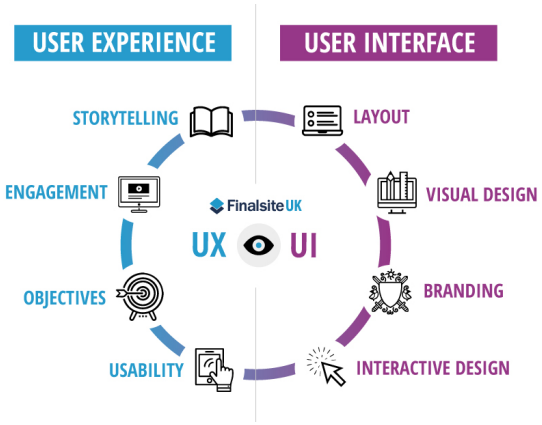

Figura 1.9: Complementarità tra User Interface e User Experience (UI/UX) [10]

1.3.3 Confronto tra sviluppo nativo e cross-platform

Dal punto di vista tecnico, lo sviluppo di un'applicazione mobile può avvenire in due modalità: sviluppo nativo (su un determinato sistema operativo) o sviluppo cross-platform. Un'app sviluppata in maniera nativa è esclusiva per quella specifica piattaforma: viene scritta con un linguaggio di programmazione adeguato (ad esempio, Swift per iOS o Kotlin per Android) e grazie a ciò permette di sfruttare meglio le caratteristiche della piattaforma. Lo sviluppo cross-platform invece permette di creare applicazioni compatibilli con più sistemi operativi, ma con funzionalità più limitate. La Figura 1.10 riporta l'attuale distribuzione a livello mon-

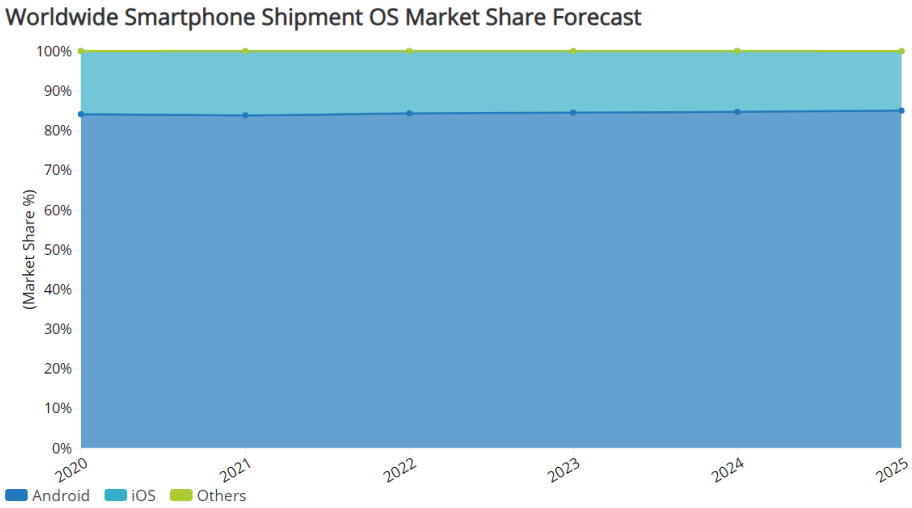

Figura 1.10: Situazione attuale della distribuzione sul mercato dei sistemi operativi per smartphone, secondo IDC (International Data Corporation) [38]

diale dei sistemi operativi, in cui si può notare la completa dominanza da parte di Android e iOS. Nonostante il grafico suggerisca che lo sviluppo nativo su Android sia preferibile in quanto l'80% dei possessori di uno smartphone abbia a bordo quel sistema operativo, vediamo ora di confrontare questi due approcci, facendo emergere una serie di differenze alla base delle quali si può prendere una decisione a seconda del contesto applicativo. L'analisi condotta nell'articolo [11] suggerisce come lo sviluppo nativo sia superiore sia in termini di performance che di User Experience, e quindi da preferire (quando possibile). Infatti, un'applicazione nativa fa un uso migliore delle risorse allocate dalla piattaforma d'utilizzo, ed è globalmente più veloce, responsive e meno soggetta a crash. Inoltre, l'integrazione con il sistema operativo è ottimale, a tal punto da poter utilizzare le funzionalità derivanti dai sensori senza limitazioni. D'altra parte, il tempo di sviluppo aumenta sensibilmente (quasi due volte il tempo che occorre nello sviluppare la stessa app in maniera cross-platform), così come il costo per certi ambiti applicativi: se si vorrà estendere il contesto ad un altro sistema operativo, bisognerà sviluppare una nuova app da zero. Questo si ripercuote anche nella gestione di eventuali bug (ogni piattaforma potrà avere i propri) e anche in progettazione di UI/UX diverse per ogni sistema operativo (basandosi sulle best-practices di ognuno). Per lo sviluppo di VASARI, si è deciso di utilizzare l'appoccio multipiattaforma, sia per raggiungere la quasi totalità degli utenti, sia per un discorso di costo (che diminuisce in quanto le skills necessarie allo sviluppatore sono minori nel caso del cross-platform, utilizzando prevalentemente tecnologie web mainstream). Il costo di manutenzione è inoltre molto minore rispetto allo sviluppo nativo. Bisogna però stare attenti al fatto che compromettere le performance e l'esperienza utente solamente per risparmiare tempo e soldi può essere controproducente. Nel corso dello sviluppo di Smart Visit, ci si è focalizzati anche su quest'ultimo aspetto: la facilità nello sviluppo cross-platform, da cui ne può conseguire un inserimento frequente di nuove funzionalità, può destabilizzare progressivamente le performance e la UX, elementi fondamentali per il successo di un'applicazione mobile. In Figura 1.11 sono sintetizzate le caratteristiche dei due approcci in ognuno dei principali aspetti di sviluppo, quali il costo ed il tempo di progettazione e manutenzione, il range di utenza e il livello di competenze necessarie.

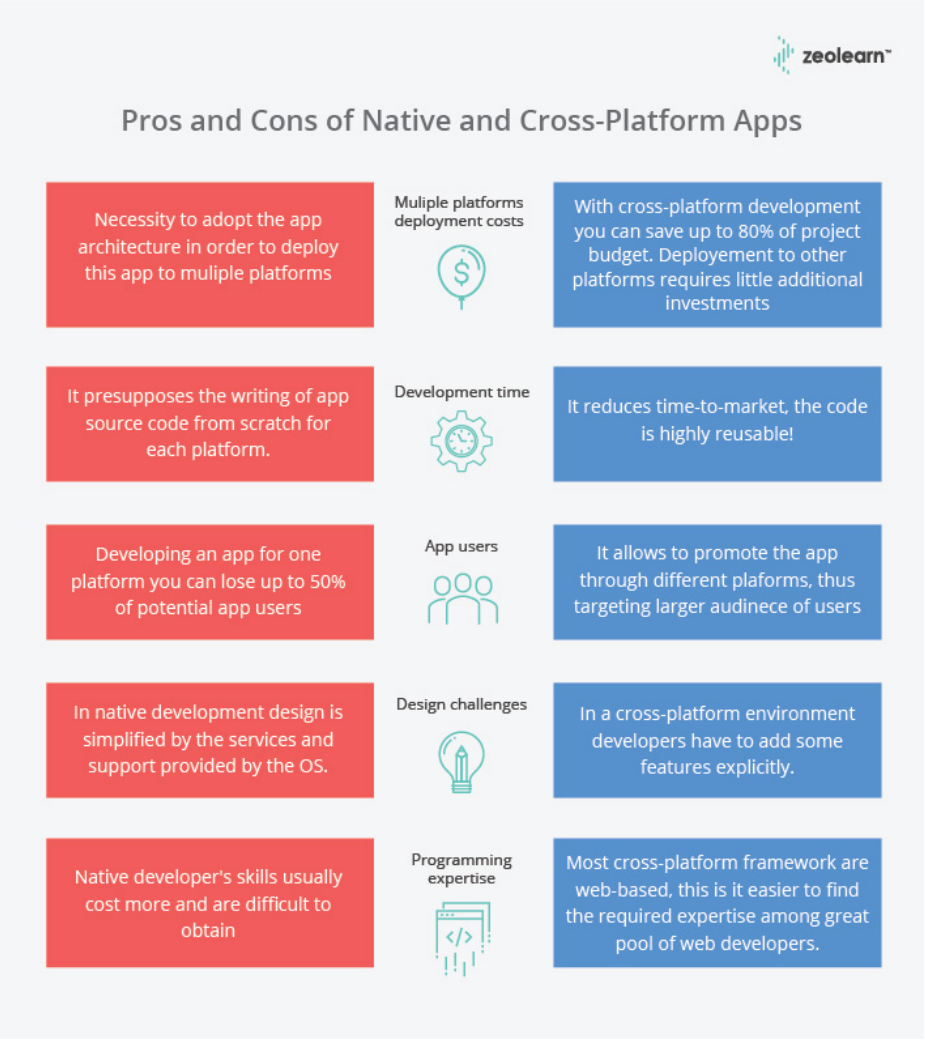

Figura 1.11: Confronto tra sviluppo nativo e cross-platform [11]

Capitolo 2

Strumenti utilizzati

Come discusso alla fine del capitolo precedente, per lo sviluppo dell'applicazione mobile di VASARI si è deciso di procedere con un approccio cross-platform, piuttosto che con sviluppo nativo. Tra i tanti possibili framework nel panorama dello sviluppo mobile cross-platform, la scelta è ricaduta su Ionic, in combinazione con la piattaforma di sviluppo Angular. Essendo inoltre necessari l'utilizzo di sensori e di funzionalità specifiche della piattaforma per le finalità dell'applicazione, si sono utilizzati dei plugin nativi Cordova/Capacitor. Si propone, all'interno di questo capitolo, un piccolo e minimale esempio progressivo di applicazione mobile cross-platform (un semplice gestore di una To Do List), in cui si dimostra l'estrema cooperabilità tra i vari strumenti, unita alla semplicità globale nello sviluppo.

2.0.1 Angular

Angular [2], rilasciato nel 2016, è un framework open source sviluppato da Google. Derivante da AngularJS, differisce da quest'ultimo dal linguaggio di programmazione utilizzato (il moderno TypeScript [25], derivante anch'esso dal JavaScript), e dalla finalità, ossia lo sviluppo di "progressive web app" (PWA), con la caratteristica di essere responsive, ossia in grado di adattarsi alla grandezza del device (mentre AngularJS non fornisce nessun supporto allo sviluppo mobile). Uno dei principali punti di forza di Angular è la possibilità di implementare il "two-way data binding", ossia un meccanismo con cui condividere dati tra componenti e rimanere in attesa di eventi, per poi aggiornare simultaneamente il valore di una variabile tra componente padre e figlio [2]. Tramite l'utilizzo della programmazione ad oggetti e delle classi Component e Service di Angular, si è implementato, nel corso dello sviluppo del modulo di Smart Visit, il pattern architetturale "MVC" (Model-View-Controller), con cui si è in grado di separare la logica di presentazione dei dati, dalla logica di manipolazione e utilizzo degli stessi. In questo pattern, il modello descrive direttamente i dati utilizzati a basso livello (ed eventualmente una logica di base indipendente dal contesto, ad esempio tramite dei metodi appositi), mentre la vista è la rappresentazione delle informazioni che si basano sui quei dati. Questi due sono collegati tramite il controller, che si pone come intermediario tra le due parti e si fa carico della logica di conversione e gestione dei dati. Tramite l'utilizzo del TypeScript, nell'esempio che segue e in tutto il lavoro di tesi, i dati sono stati incapsulati in delle classi e le varie istanze sono state gestite da particolari oggetti della classe Service (definendo così il modello). I componenti di Angular fungono invece da controller e si pongono tra i Service e le View, che sono infine delle pagine HTML/CSS (che si collegano al controller tramite l'utilizzo di direttive e implementazione del two-way data binding). Infine, durante il processo di sviluppo, l'utilizzo del TypeScript rispetto al JavaScript grezzo permette di definire meglio il comportamento di un oggetto, aggiungendo alla definizione dei suoi campi un tipo statico, che consente di validare più velocemente il codice scritto e generare tramite il compilatore del codice JavaScript pulito ed efficiente. In Figura 2.1, tratta da [7], è riportato uno schema dei vari ruoli all'interno del pattern Model-View-Controller. Nell'esempio di seguito, viene

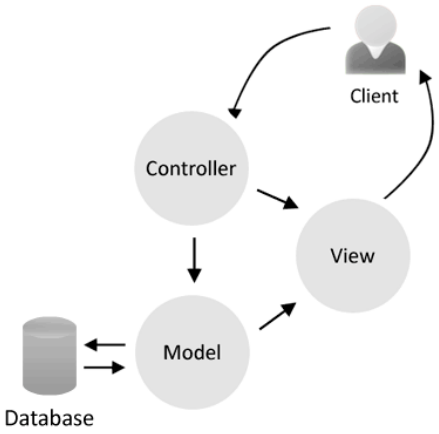

Figura 2.1: Pattern MVC [7]

sviluppato un gestore di una semplice To Do List, con un modus operandi e una struttura progettuale che verrà utilizzata anche per lo sviluppo della sezione Smart Visit di VASARI. Per prima cosa, definiamo le varie classi che modellano i dati. La classe ThingToDo, contenente un titolo (identificativo) e un livello di priorità, caratterizza un singolo elemento all'interno di una lista, incapsulata nella classe ToDoList. Nella creazione delle classi, vengono seguiti alcuni paradigmi e best-practices della Programmazione ad Oggetti quali l'incapsulamento, l'utilizzo di metodi getter e setter (per non accedere direttamente alle proprietà dell'oggetto), limitare la logica della classe alla gestione di semplici problemi non dipendenti dal contesto applicativo, e così via.

```
1  enum Priority {
2    low,
3    middle,
4    high
5  }
6
7  class ThingToDo {
8
9    private _title: string;
10   private _priority: Priority;
11
12   constructor(title: string, priority: Priority) {
13     this.title = title;
14     this.priority = priority;
15   }
16
17   get title(): string {
18     return this._title;
19   }
20
21   set title(value: string) {
22       this._title = value;
23   }
24
25   get priority(): Priority {
26       return this._priority;
27   }
28
29   set priority(value: Priority) {
```

```
30        this._priority = value;
31    }
32
33    public isLowPriority(): boolean {
34        return this.priority === Priority.low;
35    }
36
37    public isMiddlePriority(): boolean {
38        return this.priority === Priority.middle;
39     }
40
41     public isHighPriority(): boolean {
42        return this.priority === Priority.high;
43    }
44
45 }
46
47 class ToDoList {
48
49    private _things: ThingToDo[];
50
51    constructor() {
52        this._things = [];
53    }
54
55    get things(): ThingToDo[] {
56        return this._things;
57    }
58
59    set things(value: ThingToDo[]) {
60        this._things = value;
61    }
62
63    public addThingToDo(thingToDo: ThingToDo) {
64        if (!this.things.find(el => el.title === thingToDo.title)) {
65          this.things.push(thingToDo);
66        }
67    }
68
69    public removeThingToDo(thingToDo: ThingToDo) {
70        this.things = this.things.filter(el => el !== thingToDo);
71    }
72
73 }
```

Si noti come l'accesso ai campi privati della classe avviene tramite metodi getter e setter, facendo in modo sia di rendere questa operazione, di per sé sensibile, più sicura, e inoltre potendo inserire della logica aggiuntiva ogniqualvolta si debba modificare o restituire il valore di una proprietà (ad esempio, se una classe contiene più proprietà tra esse collegate, è possibile modificare automaticamente il valore di altre proprietà, alla chiamata di un singolo metodo setter). Per quanto riguarda la logica all'interno delle classi, si presuppone che essa sia distinta da quella relativa al contesto applicativo e che riguardi solamente istanze di quella classe. Nell'esempio mostrato, la classe *ToDoList* contiene il metodo *AddThingToDo(thingToDo: ThingToDo)*, il quale, prima di inserire un elemento all'interno della lista, fa un controllo per evitare il caso di inserire elementi con un titolo duplicato. Questo è un esempio di logica che è distinta dal contesto applicativo, ossia solamente relativa alla struttura

della classe e di come essa di è deciso di modellare in fase di progettazione. Eliminare questa logica dalle classi e inserirla invece all'interno del contesto applicativo, porterebbe di base al problema di ricordare, in fase di programmazione, la logica dietro ogni classe (gestione di identificativi, di controlli di sicurezza, ecc...), complicando lo sviluppo è aumentando sensibilmente la probabilità di inserire bug nell'applicazione.
Per completare la prima fase del pattern MVC, viene creato un Service apposito che possa gestire la logica dietro l'acquisizione o il salvataggio dei dati (nell'esempio, la nostra lista di cose da fare), e che possa semplificare alcune operazioni di base come il passaggio parametri tra più pagine. Inoltre, si segue in questo modo il principio di avere un unico punto di accesso per oggetti globali all'interno dell'applicazione (facendo in maniera tale da evitare l'instanziamento multiplo di più istanze di una classe, anche nel caso in cui ci si riferisca logicamente allo stesso oggetto).

```
1  class ToDoListService {
2
3      private _toDoList: ToDoList;
4      private _selectedThingToDo: ThingToDo;
5
6      constructor() {
7          this.toDoList = new ToDoList();
8      }
9
10     get toDoList(): ToDoList {
11         return this._toDoList;
12     }
13
14     set toDoList(value: ToDoList) {
15         this._toDoList = value;
16     }
17
18     get selectedThingToDo(): ThingToDo {
19         return this._selectedThingToDo;
20     }
21
22     set selectedThingToDo(value: ThingToDo) {
23         this._selectedThingToDo = value;
24     }
25
26 }
```

All'interno del costruttore del Service, può essere inserita la logica di acquisizione dei dati (dallo storage locale, da un server, ecc...), mentre in altre sezioni può essere inserita la logica di salvataggio dati (ad esempio, quando si modifica la lista, oppure con dei metodi appositi). Di particolare interesse è l'instanza della classe ThingToDo, che ha come compito quello di semplificare il passaggio parametri tra più pagine. Nella progettazione del gestore, sarà presente una pagina di "dettagli" per la singola azione da compiere all'interno della To Do List, per cui, prima di navigare in quella pagina, viene settato l'elemento da considerare per la visualizzazione. In questo modo, viene semplificato il passaggio parametri (non dovendo fare altro che settare una variabile in maniera opportuna, invece di inserire dati del routing dell'applicazione, con il rischio di rendere più pesante questa operazione in caso di grosse quantità di dati), e viene anche costruito indirettamente un pattern che prevede che, per ogni classe che necessita di pagine particolari (di visualizzazione dettagli, piuttosto che per altro), venga definito un Service che ne modelli le instanze, e ogni pagina che si appoggia su questo Service utilizzi una particolare istanza della classe (nota prevalentemente come *_selected<className>*, all'interno anche della sezione dell'app VASARI relativa a Smart Visit).

Con la definizione delle classi e dei Service, si completa il primo tassello (il Model) per l'implementazione del pattern MVC. L'applicazione di esempio, molto minimale, contiene solamente due pagine: la prima che si occupa di gestire la sezione riguardante la lista nel complesso (con operazioni quali l'inserimento di nuovi elementi o la cancellazione di altri già presenti), mentre l'altra si riferisce solamente ad un singolo elemento, fungendo da pagina di "dettagli" (ossia, in cui si espande la visione del singolo elemento e si modifica in-loco). Le pagine stesse, su cui in seguito si appoggeranno le View del pattern MVC (che saranno pagine HTML/CSS arricchite con il framework Ionic per lo sviluppo cross-platform), agiscono da Controller (ossia, intermediano tra le View e i Model, gestiti dall'accoppiata Classe/Service). Di particolare interesse, all'interno di questa sezione, è l'utilizzo di una serie di componenti Angular (costruiti sopra Ionic) che permettono l'utilizzo di widget e funzionalità di uso comune nello sviluppo mobile. Ad esempio, si può creare un Alert (*AlertController*) che faccia confermare all'utente la volontà di eliminare un elemento dalla lista, oppure un Popover (*PopoverController*) su cui si inseriscano le informazioni per aggiungere un nuovo elemento alla lista. Altri esempi di componenti simili sono il *NavController*, molto utile nel gestire la navigazione tra le varie pagine seguendo l'ordine logico dello Stack con le operazioni di push e pop di una pagina, il *ToastController* (per mostrare piccoli messaggi di conferma di un'operazione), il *LoadingController* (per mostrare una fase di caricamento), e così via. Un esempio di utilizzo di questi componenti si ha di seguito, in cui si vuole far mostrare l'Alert di conferma per cancellare un elemento dalla lista, il Popover per crearne un'altro e il meccanismo di navigazione nativo all'interno dell'app:

```
1  class HomePage {
2
3      public componentVisible: boolean;
4
5      constructor(public toDoListService: ToDoListService,
6                  private popoverController: PopoverController,
7                  private alertController: AlertController,
8                  private navController: NavController) {}
9
10     public async addNewThingToDo() {
11         this.componentVisible = true;
12         const popover = await this.popoverController.create(
13             {
14                 component: AddThingToDoPopoverComponent
15             }
16         );
17         await popover.present();
18         await popover.onDidDismiss();
19         this.componentVisible = false;
20     }
21
22     public async removeThingToDo(thingToDo: ThingToDo) {
23         const alert = await this.alertController.create(
24             {
25                 header: 'Sei sicuro?'
26                 buttons: [
27                     {
28                         text: 'Annulla',
29                         role: 'cancel'
30                     },
31                     {
32                         text: 'Elimina',
33                         handler: () => {
34                             this
35                                 .toDoListService
```

```
                                    .toDoList
                                    .removeThingToDo(thingToDo) }
                    }
                ]
            }
        );
        await alert.present();
    }

    public navigateToDetails(thingToDo: ThingToDo) {
        this.toDoListService.selectedThingToDo = thingToDo;
        this.navController
            .navigateForward(['thing-to-do-details']);
    }

}
```

Si noti come alcuni componenti necessitano la creazione di pagine apposite (che vedremo nella sezione successiva), mentre altri hanno un meccanismo più semplice. Per quanto riguarda il *NavController*, la struttura della funzione che permette di navigare sarà di largo uso anche all'interno di Smart Visit: essa per prima cosa setta la variabile di selezione all'interno del Service apposito (in questo caso della To Do List), e in seguito naviga verso la pagina di riferimento. Anche questa parte verrà esplorata nella sezione che segue. Introduciamo ora il framework Ionic per discutere in seguito l'integrazione con Angular per il pattern MVC e il "two-way data binding".

2.0.2 Ionic

Ionic [23], rilasciato per 2013 e giunto alla versione stabile 5, è un SDK open-source di sviluppo mobile cross-platform. Costruito all'inizio sopra AngularJS e Apache Cordova, è divenuto nel corso del tempo un framework di largo utilizzo anche mediante strumenti terzi come Vue.js o React. Il vantaggio principale di questo framework, paragonandolo ad altri presenti nel mercato open-source, è quello di fornire tutta una serie di componenti di User Interface che, a seconda della piattaforma di esecuzione, vengono renderizzati in maniera nativa. Possedendo inoltre un'elevata integrazione con Angular, permette di sviluppare graficamente delle applicazioni in maniera molto semplice, robusta e scalabile. Un'applicazione Ionic è composta da una serie di blocchi ad alto livello che permettono di costruire velocemente delle UI moderne e reattive, con un approccio molto simile al classico HTML/CSS. Ionic fornisce una lunga serie di componenti grafici caratterizzati dal prefisso *ion* e che, come in HTML5, è possibile incapsulare tra di essi per formare strutture grafiche più o meno complesse. Degli esempi basilari possono essere i tag *ion-button*, *ion-card*, parlando di componenti più di alto livello, ma anche i tag *ion-back-button* o *ion-menu-button* per elementi più inerenti al sistema di navigazione. Un grosso vantaggio di Ionic è anche l'elevata interoperabilità con Angular, tramite una serie di direttive che permettono di usare la funzionalità del two-way data binding, ossia di poter aggiornare il valore di una variabile a partire da un evento avvenuto sull'interfaccia grafica, e viceversa. Riprendendo l'esempio del gestore di una To Do List, è qui riportato il codice HTML relativo al *Popover* con cui si chiede all'utente di inserire un nuovo elemento nella lista:

```
<ion-item>
  <ion-label position="floating">Nuovo elemento</ion-label>
  <ion-input [(ngModel)]="title"></ion-input>
</ion-item>
<ion-radio-group [(ngModel)]="priority">
  <ion-item>
```

```
7       <ion-label>Bassa</ion-label>
8       <ion-radio value="low"></ion-radio>
9     </ion-item>
10  <ion-item>
11      <ion-label>Media</ion-label>
12      <ion-radio value="middle"></ion-radio>
13    </ion-item>
14  <ion-item>
15      <ion-label>Alta</ion-label>
16      <ion-radio value="high"></ion-radio>
17    </ion-item>
18  </ion-radio-group>
19  <ion-button expand="block" fill="outline"
        (click)="addNewThingToDo()">Aggiungi</ion-button>
```

In cui le variabili *title* e *priority* sono proprie del file TypeScript relativo al componente, e che vengono aggiornate in seguito agli eventi correspondenti della UI (l'inserimento di testo nella casella dell'input e la selezione di un radio button particolare). Essendo il binding bilaterale, un eventuale aggiornamento delle variabili a livello di flusso (e non a livello di UI) avrà comunque come conseguenza quella di aggiornare anche i componenti a cui sono associate. Nel rendering della pagina HTML, a seconda del sistema operativo, si avrà un risultato grafico diverso: Nel rendering mostrato nelle Figure 2.2 e 2.3, vediamo dei cambiamenti sia a livello di

Figura 2.2: Componente renderizzato nel sistema operativo Android

Figura 2.3: Componente renderizzato nel sistema operativo iOS

forma del componente (la versione iOS appare più snella, con un pulsante meno spigoloso e più tendente al rotondo rispetto ad Android), sia a livello di colorazione (la versione Android, oltre ad avere un sfondo diverso, renderizza dei colori nell'input di testo, che mancano nella versione iOS), e infine a livello concettuale (i radio buttons della versione iOS vengono selezionati con delle spunte, mentre quelli Android sono più vicini alla loro concezione classica con cerchi vuoti/pieni). Cambiamenti grafici più o meno vistosi possono essere notati in vari aspetti di un'applicazione sviluppata in Ionic, dal gran numero di icone di Ionicons, agli elementi di navigazione come i Tabs o un classico menù laterale, ecc.
Questa variabilità di rendering dell'interfaccia grafica, unita alla semplicità di sviluppo, rende Ionic un framework veramente potente e che permette di creare applicazioni mobile con i più moderni design di sviluppo mobile (Il Material Design di Google per quanto riguarda Android e l'iOS Design di Apple), inserendo anche, dove possibile, l'implementazione di gesture comuni ed efficaci. L'unione di Angular e Ionic permette, come già detto, di aumentare sensibilmente la scalabilità del codice e la sua potenza, mantenendo tuttavia una semplicità di sviluppo

elevata. Il two-way data binding permette di far comunicare in maniera efficace tutte le componenti dell'applicazione, e l'utilizzo di altre direttive come *ngIf* e *ngFor* permette di gestire cambiamenti di stato, condizioni booleane per il rendering di determinati componenti e generazione automatica e ciclica di codice HTML. Si riporta di seguito la pagina principale di questa demo, ossia quella che mostra nel complesso la To Do List, e il risultato grafico nei due sistemi operativi Android e iOS nelle Figure 2.4 e 2.5.

```
<ion-header>
  <ion-toolbar color="primary">
    <ion-title>
      <h1 style="color: white">Cose da fare</h1>
    </ion-title>
  </ion-toolbar>
</ion-header>
<ion-content>
  <ion-backdrop *ngIf="componentVisible" style="opacity: 0.1">
  </ion-backdrop>
  <ion-list *ngIf="toDoListService.toDoList">
    <ion-item-sliding
            *ngFor="let thing of toDoListService.toDoList.things">
      <ion-item-options side="end">
        <ion-item-option color="none" (click)="removeThingToDo(thing)">
            <ion-icon name="trash-outline"
                      color="danger"
                      size="large">
            </ion-icon>
        </ion-item-option>
      </ion-item-options>
      <ion-item lines="none" (click)="navigateToDetails(thing)"
                             class="ion-activatable ripple-parent">
        <ion-ripple-effect type="unbounded"></ion-ripple-effect>
        <h3 *ngIf="thing.isLowPriority()" style="color: #2dd36f">
            {{thing.title}}
        </h3>
        <h3 *ngIf="thing.isMiddlePriority()" style="color: #e0ac08">
            {{thing.title}}
        </h3>
        <h3 *ngIf="thing.isHighPriority()" style="color: #eb445a">
            {{thing.title}}
        </h3>
      </ion-item>
    </ion-item-sliding>
  </ion-list>
  <ion-fab vertical="bottom" horizontal="end" slot="fixed">
    <ion-fab-button (click)="addNewThingToDo()">
      <ion-icon name="add" style="color: white"></ion-icon>
    </ion-fab-button>
  </ion-fab>
</ion-content>
```

In conclusione, è possibile attivare una serie di eventi che avvengono sulla UI. Nell'esempio, al click di uno degli elementi della lista, viene attivata la funzione *navigateToDetails()* mostrata precedentemente, la quale mostra i dettagli del singolo elemento e ne permette la modifica, oppure, con una semplice gesture, è possibile fare swipe e mostrare un pulsante rappresentante un cestino, che ha lo scopo di eliminare, dopo aver mostrato un *Alert*, l'elemento selezionato. All'interno sia di questa applicazione di esempio, che di VASARI, l'utilizzo di pagine di dettaglio di vari aspetti è di largo utilizzo, per cui è stato deciso e utilizzato il pattern discusso in precedenza, ossia l'utilizzo di una classe che modella i dati, un Service che tiene traccia di varia istanze (tra cui una di selezione), e una pagina che si riferisce esclusivamente all'istanza di selezione. Nell'esempio, dopo aver cliccato su un singolo elemento, l'istanza di

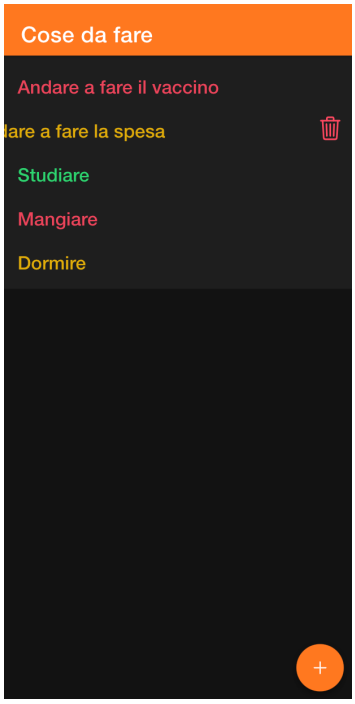

Figura 2.4: Pagina principale renderizzata nel sistema operativo Android

Figura 2.5: Pagina principale renderizzata nel sistema operativo iOS

selezione della classe ThingToDo viene settata con l'elemento, e tramite il *NavController* viene aggiunta una pagina allo Stack che, in ogni tag Ionic, si riferisce esclusivamente a quell'istanza. Nelle Figure 2.6 e 2.7 sono contenuti i rendering grafici della pagina di dettagli, per i sistemi operativi Android e iOS:

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button></ion-back-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
<ion-content>
  <div style="display: flex; justify-content: center">
    <h1 *ngIf="toDoListService.selectedThingToDo.isLowPriority()">
      {{toDoListService.selectedThingToDo.title}}
    </h1>
    <h1 *ngIf="toDoListService.selectedThingToDo.isMiddlePriority()">
      {{toDoListService.selectedThingToDo.title}}
    </h1>
    <h1 *ngIf="toDoListService.selectedThingToDo.isHighPriority()">
      {{toDoListService.selectedThingToDo.title}}
    </h1>
  </div>
  <br><br><br>
  <div style="display: flex; justify-content: center">
    <h3 style="color: #2dd36f" *ngIf="priority === 0">
        Priorita: bassa
    </h3>
    <h3 style="color: #e0ac08" *ngIf="priority === 1">
        Priorita: media
```

```
27      </h3>
28      <h3 style="color: #eb445a" *ngIf="priority === 2">
29          Priorita: alta
30      </h3>
31    </div>
32    <ion-range min="0" max="2" step="1" snaps="true" color="primary"
33          [(ngModel)]="priority"
34          (ionChange)="updatePriority()">
35    </ion-range>
36  </ion-content>
```

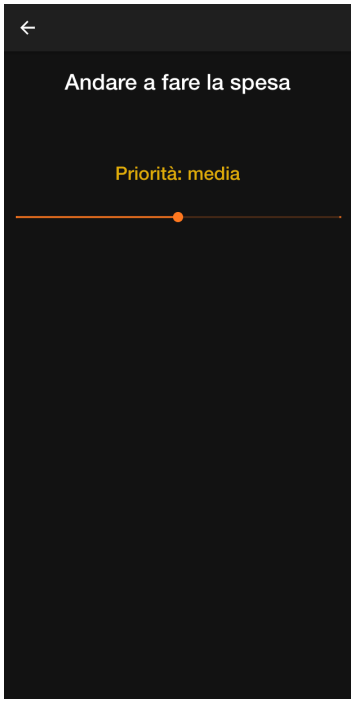

Figura 2.6: Pagina di dettagli renderizzata nel sistema operativo Android

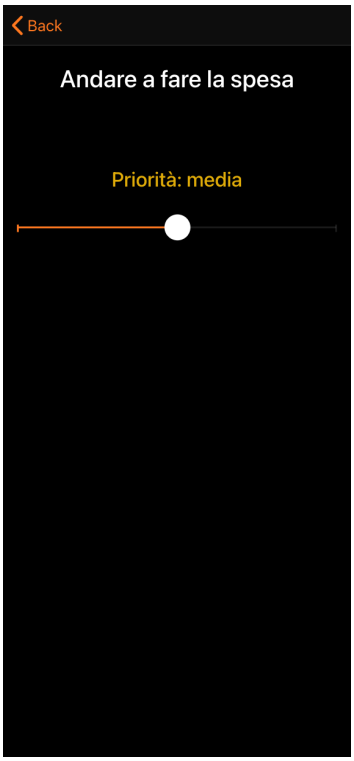

Figura 2.7: Pagina di dettagli renderizzata nel sistema operativo iOS

2.0.3 Plugin nativi Cordova/Capacitor

Ultimo aspetto di cui parlare di questo capitolo è l'integrazione dell'app con le funzionalità native del dispositivo. In particolare, all'interno dell'app VASARI e nello specifico nel modulo di Smart Visit, verrà fatto largo uso delle funzionalità derivanti da sensori e altri elementi nativi del device fisico. È possibile, all'interno di un'app Ionic, attivare tutta una serie di funzionalità native grazie ai plugin Cordova o Capacitor. Apache Cordova [19] è un framework open-source che permette di fare accesso a funzionalità di un dispositivo, incapsulando il tutto con moderne tecnologie di sviluppo web come HTML5/CSS3 e JavaScript/TypeScript. Capacitor [9] è uno strumento più recente che fornisce gli stessi benefici, ma con un approccio più moderno e orientato alla semplicità di sviluppo, permettendo di utilizzare un unico set di API (a prescindere dalla piattaforma di esecuzione), invece di un set di API per ogni sistema operativo. Come si discuterà in seguito, è stato fatto largo uso di plugin Capacitor (e all'occorrenza Cordova), per utilizzare funzionalità quali GPS, la bussola, player multimediali

nativi, vibrazione, e tante altre. Ad esempio, il codice seguente mostra come sia possibile, tramite Capacitor, utilizzare i dati derivanti dal sensore (nell'esempio vengono stampati sulla console di debug) ed essere notificati (tramite la sottoscrizione all'evento) ogni volta che essi cambiano.

```
1 constructor(private geolocation: Geolocation) {}
2 .
3 .
4 .
5 this.geolocation.watchPosition().subscribe((data) => {
6   console.log(data.coords.latitude);
7   console.log(data.coords.longitude);
8 });
```

A livello di sviluppo, dopo aver opportunamente impostato il proprio progetto all'utilizzo dello specifico plugin, l'utilizzo dello stesso diventa minimale, da qui una grande semplicità unita all'efficienza che rende Ionic/Capacitor uno dei principali strumenti dello sviluppo mobile odierno.

Capitolo 3

Ricerca di punti di interesse e gestione dei percorsi

Dopo una prima fase iniziale dell'app, il cui design e progettazione non è oggetto di studio di questa tesi, l'utente si ritroverà a dover decidere un percorso culturale, personalizzato o predefinito. In questo capitolo descriviamo questa sezione di Smart Visit, alla quale seguiranno gli step di guida lungo il percorso scelto e di fruizione di contenuti culturali.

3.1 Analisi UI/UX

Lo scopo principale di questa sezione è quella di creare delle bozze di percorso, ossia delle liste contenenti punti di interesse o interi siti culturali. A partire da esse, in seguito ad una comunicazione con uno specifico server, verranno generate delle liste formate da una serie di percorsi multi-site e da una serie di percorsi on-site. In una pagina dedicata alla visualizzazione dei dettagli di un percorso, potranno essere effettuate delle modifiche, ed infine avviarlo. Verrà predisposta una pagina dedicata anche alla ricerca di punti di interesse per nome, con alcuni filtraggi, per poter arricchire una bozza prima di convertirla in uno o più percorsi.
Questa è la sezione di Smart Visit che prevede l'utilizzo di più pagine, e quindi con il flusso di navigazione più complesso. Occorre che le varie pagine siano ben collegate tra di loro e, soprattutto, che l'accesso a pagine comuni possa avvenire da punti diversi. Bisogna, ad esempio, permettere di visionare i dettagli di un punto di interesse (tramite una pagina apposita che verrà descritta nell'ultimo capitolo) sia nella pagina di ricerca, che in quella di dettagli di un percorso. Questa sezione è ricca anche di piccoli componenti, come *Popover* o *Alert*: questo rende il tutto più intuitivo e coerente con i più moderni concetti di User Experience nello sviluppo mobile.
L'inizio di questa fase di Smart Visit avviene nella pagina iniziale dell'app, mostrata nelle Figure 3.1 e 3.2, che appare in seguito all'autenticazione dell'utente. É presente un menù laterale, contenente varie sezioni dell'app non oggetto di studio di questa tesi (come il profilo utente, o le impostazioni dell'applicazione), ma soprattuto quella relativa alla gestione dei percorsi. Tale menù è mostrato nelle Figure 3.3 e 3.4.

3.1.1 Ricerca di punti di interesse

Cliccando sulla lente di ingrandimento, in alto a destra all'interno del menù laterale, si viene reindirizzati verso un'apposita pagina di ricerca, mostrata nelle Figure 3.5 e 3.6. Qui l'utente può digitare un testo, filtrando la ricerca con una serie di opzioni mutuamente esclusive, visualizzate tramite dei *Radio Button*: opere per titolo, opere per autore e siti culturali. In seguito, cliccando sul pulsante "Cerca", vengono mostrati i risultati della ricerca (nelle Figure 3.7 e 3.8). Ognuno di essi è formato da un titolo e un'immagine rappresentativa. Cliccando

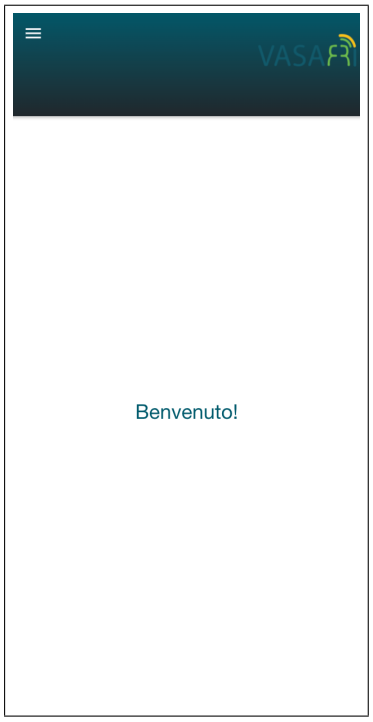

Figura 3.1: Pagina iniziale renderizzata nel sistema operativo Android

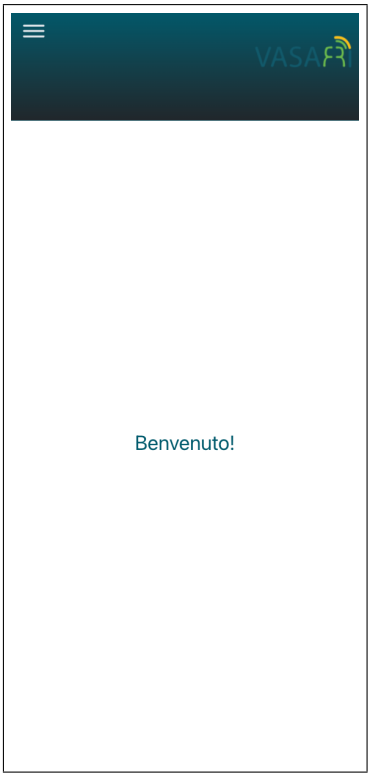

Figura 3.2: Pagina iniziale renderizzata nel sistema operativo iOS

su uno di essi, viene aperto un *Pullup* che espande l'elemento (nelle Figure 3.9 e 3.10). Sono presenti due pulsanti, che servono ad aggiungere il punto di interesse ad una bozza o ad un percorso. Nel primo caso, è possibile selezionare una bozza esistente, oppure crearne una sul momento. Nel secondo caso, vengono visualizzati tutti i percorsi già esistenti (predefiniti o generati precedentemente dall'utente) il cui sito culturale di riferimento coincide con il sito culturale del punto di interesse selezionato. I componenti di selezione sono mostrati nelle Figure 3.11, 3.12, 3.13 e 3.14.
In questo frangente, il corpo della pagina che contiene i vari input viene sostituito con un particolare componente Ionic, chiamato *Backdrop*: esso permette di rendere una parte della schermata (o tutta) non cliccabile, in maniera tale da rendere obbligatoria la chiusura del *Pullup* prima di poter fare altre ricerca. In questo modo, oltre ad evitare contrasti logici tra le operazioni di ricerca e di aggiunta a bozze/percorsi, permette di focalizzare l'attenzione sull'elemento ingrandito, riducendo l'opacità del contenuto sottostante.

3.1.2 Creazione e gestione di bozze

Come detto, nel momento in cui si espande un elemento derivante da una ricerca, è possibile aggiungerlo ad una bozza. Essa rappresenta un elenco di punti di interesse o di siti culturali che, in qualsiasi momento, può essere validato lato server e convertito in un insieme di percorsi (tra più siti o nello stesso sito). La differenza tra bozza e percorso sta quindi nel tipo di elementi contenuti: la bozza può contenere sia punti di interesse che siti culturali, mentre il percorso contiene esclusivamente dei punti di interesse (associati allo stesso sito culturale). In fase di design è stato deciso questo approccio sia per evitare che l'utente crei dei percorsi inconsistenti, e sia per generare dei percorsi ottimali in relazione ai contenuti. L'utente perciò agirà esclusivamente sulle bozze, riempiendole, o su percorsi già esistenti, aggiungendo punti di interesse coerentemente filtrati per appartenere allo stesso sito culturale, o eliminandone.

Figura 3.3: Pagina con menù renderizzata nel sistema operativo Android

Figura 3.4: Pagina con menù renderizzata nel sistema operativo iOS

Nel menù laterale è possibile accedere, tramite il pulsante "I miei percorsi", ad una pagina che contiene l'insieme delle bozze momentanee e dei percorsi definitivi. Sono presenti due sezioni distinte, navigabili tramite un *Tabs Menù* posto in basso. L'utente può, attivando l'opportuna sezione mostrata nell Figure 3.15 e 3.16, visionare l'elenco delle proprie bozze: per ognuna di esse è presente il titolo e il numero di elementi contenuti. All'interno della *Toolbar* è presente un pulsante che permette di creare nuove bozze, tramite un *Popup* apposito (nelle Figure 3.17 e 3.18). Per eliminare invece una bozza, basta farla scorrere verso sinistra e selezionare l'opzione corrispondente, facendo apparire un *Alert* per confermare l'operazione. Cliccando su un bozza, si naviga verso una pagina di dettagli, mostrata nelle Figure 3.19 e 3.20. Qui è possibile vedere l'elenco completo degli elementi che la compongono, e il *FAB (Floating Action Button)* "Genera percorsi" tramite il quale è possibile convertire la bozza in uno o più percorsi definitivi. É possibile aggiungere o eliminare elementi con la stessa logica descritta in precedenza.

3.1.3 Generazione e gestione dei percorsi

Nella stessa pagina delle bozze, sono presenti anche i percorsi. Essi sono una sequenza (ordinata o no) di punti di interesse (all'interno dello stesso sito culturale) tramite la quale viene attivata la vera e propria visita culturale smart. Possono essere autogenerati (ossia predefiniti o specifici in base al profilo utente), oppure creati manualmente. Per creare un percorso, occorre prima generare una bozza e riempirla, per poi selezionare l'opzione corrispondente (il *FAB* "Genera percorsi"). La bozza scomparirà dall'elenco, e uno o più percorsi verrano inseriti nella sezione apposita. Analogamente per la lista delle bozze, la lista dei percorsi prevede, per ogni percorso, il proprio titolo e il numero di punti di interesse contenuti. Per eliminare un percorso, si utilizza la stessa gesture vista in precedenza (fare scorrere l'elemento verso sinistra e selezionare l'opzione corrispondente). La sezione relativa ai percorsi è mostrata nelle

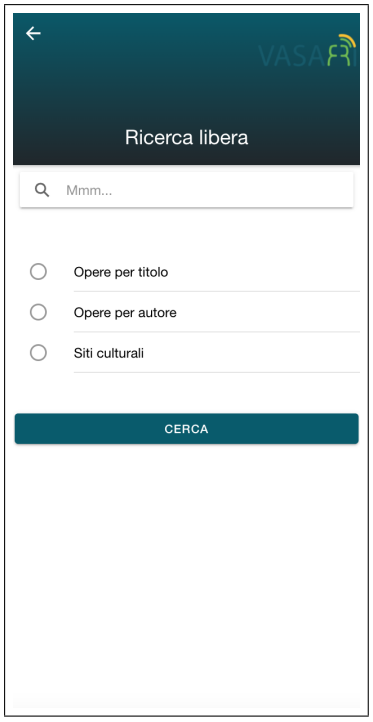

Figura 3.5: Pagina di ricerca renderizzata nel sistema operativo Android

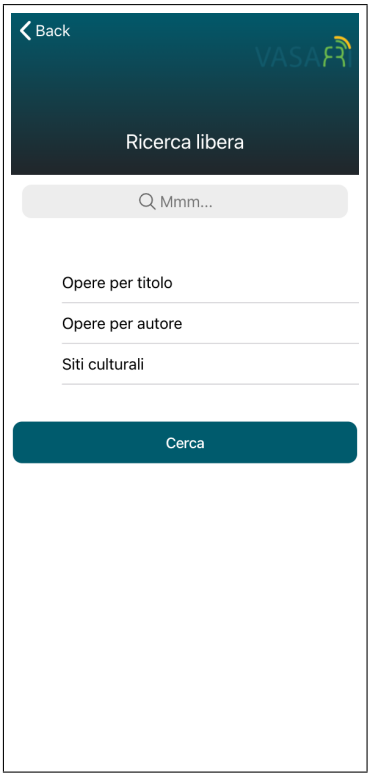

Figura 3.6: Pagina di ricerca renderizzata nel sistema operativo iOS

Figure 3.21 e 3.22. Cliccando su uno dei percorsi, appare la propria pagina dei dettagli (nelle Figure 3.23 e 3.24). A differenza delle bozze, si è deciso di visualizzare i punti di interesse in una maniera diversa, aumentando la grandezza delle immagini e inserendo a sinistra una linea il cui colore indica se il punto di interesse è stato visualizzato o meno (in relazione a quanto verrà discusso nei capitoli successivi). Questo perchè, durante la visita smart, sarà possibile accedere a questa sezione in qualsiasi momento, per modificare dinamicamente il proprio percorso. É possibile eliminare, riordinare o aggiungere dei punti di interesse seguendo la stessa logica delle bozze: si è deciso di mantenere omogenee queste operazioni per evitare di dover eseguire azioni diverse per raggiungere obiettivi simili. Il *FAB* "Avvia il percorso", infine, permette di accedere alla sezione successiva di Smart Visit, ossia quella di guida del turista lungo il percorso fisico. Nel momento in cui un percorso viene avviato ma l'utente torna alle sezioni precedenti dell'app, la pagina iniziale viene aggiornata e viene inserito un collegamento rapido per riprendere il percorso (come mostrato nelle Figure 3.25 e 3.26).

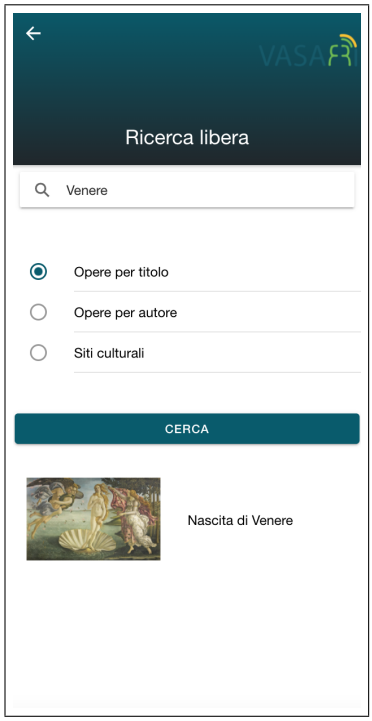

Figura 3.7: Pagina con risultati della ricerca renderizzata nel sistema operativo Android

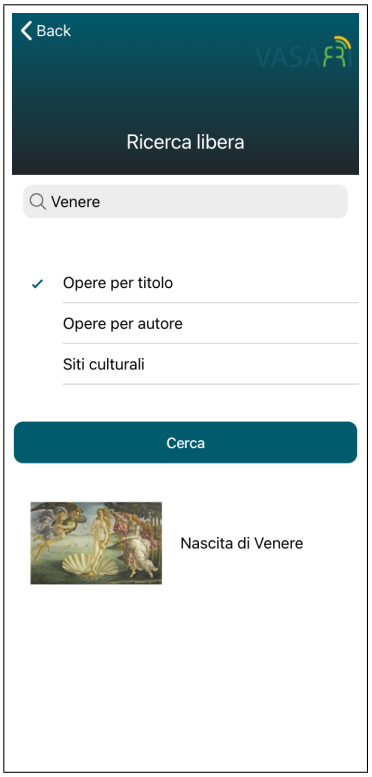

Figura 3.8: Pagina con risultati della ricerca renderizzata nel sistema operativo iOS

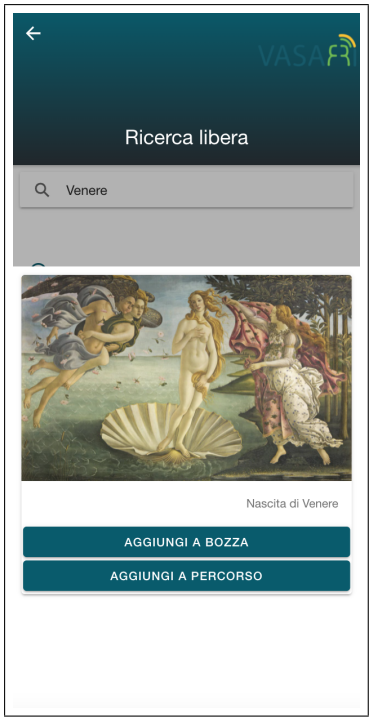

Figura 3.9: Pagina con dettagli di un risultato della ricerca renderizzata nel sistema operativo Android

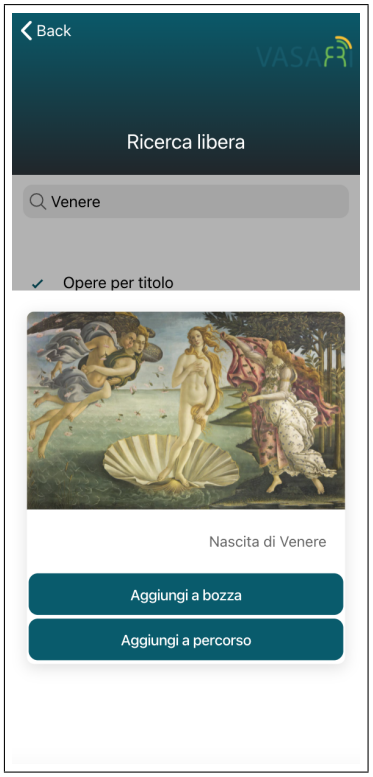

Figura 3.10: Pagina con dettagli di un risultato della ricerca renderizzata nel sistema operativo iOS

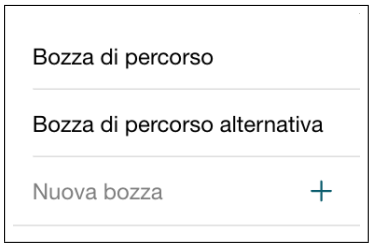

Figura 3.11: Componente di selezione bozza renderizzato nel sistema operativo Android

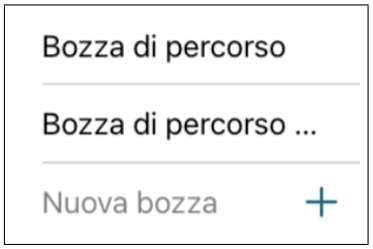

Figura 3.12: Componente di selezione bozza renderizzato nel sistema operativo iOS

Figura 3.13: Componente di selezione percorso renderizzato nel sistema operativo Android

Figura 3.14: Componente di selezione percorso renderizzato nel sistema operativo iOS

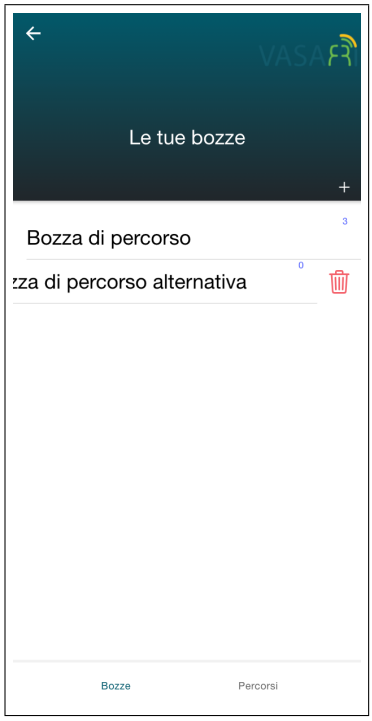

Figura 3.15: Pagina delle bozze renderizzata nel sistema operativo Android

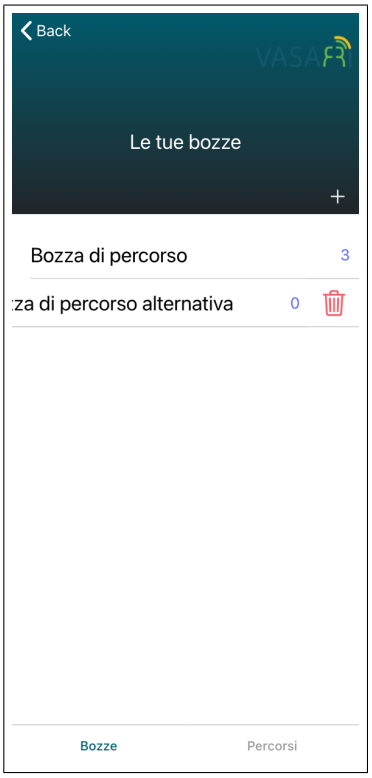

Figura 3.16: Pagina delle bozze renderizzata nel sistema operativo iOS

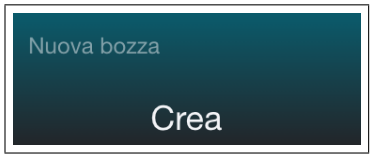

Figura 3.17: Componente di creazione bozza renderizzato nel sistema operativo Android

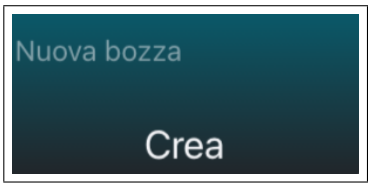

Figura 3.18: Componente di creazione bozza renderizzato nel sistema operativo iOS

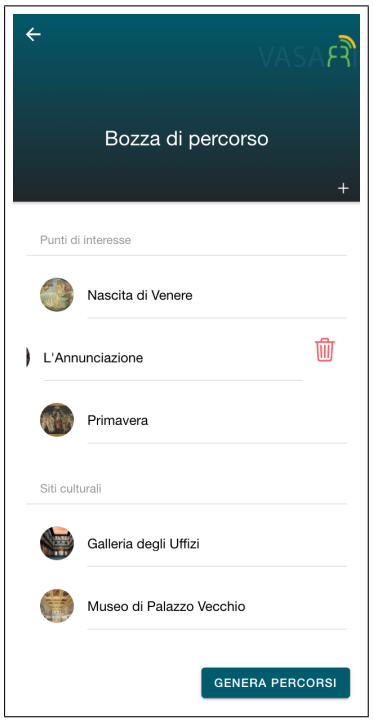

Figura 3.19: Pagina di dettagli di una bozza renderizzata nel sistema operativo Android

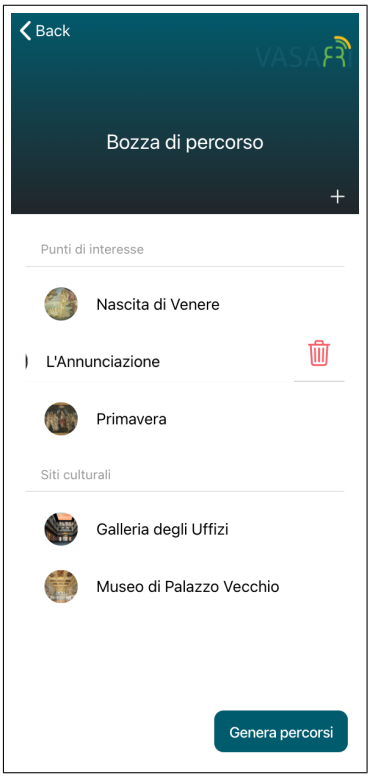

Figura 3.20: Pagina di dettagli di una bozza renderizzata nel sistema operativo iOS

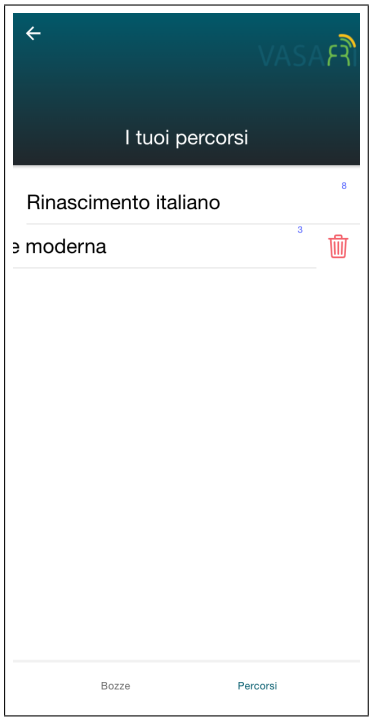

Figura 3.21: Pagina dei percorsi renderizzata nel sistema operativo Android

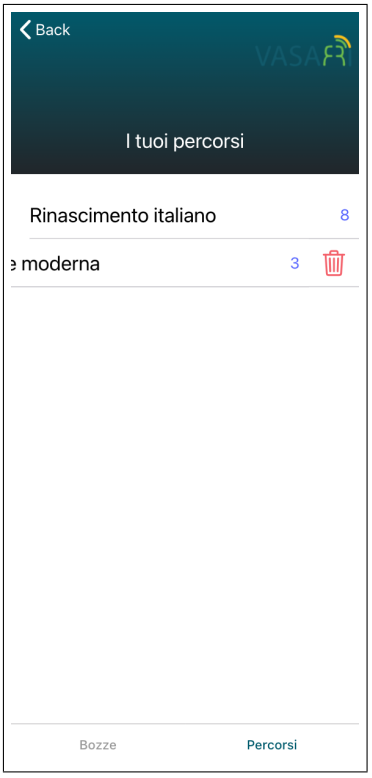

Figura 3.22: Pagina dei percorsi renderizzata nel sistema operativo iOS

Figura 3.23: Pagina di dettagli di un percorso renderizzata nel sistema operativo Android

Figura 3.24: Pagina di dettagli di un percorso renderizzata nel sistema operativo iOS

Figura 3.25: Pagina iniziale con collegamento rapido al percorso renderizzata nel sistema operativo Android

Figura 3.26: Pagina iniziale con collegamento rapido al percorso renderizzata nel sistema operativo iOS

Capitolo 4

Guida del turista lungo il percorso fisico

Dopo aver selezionato un percorso, relativo ad un sito culturale, si vuole guidare l'utente all'interno di esso, dando un opportuno feedback riguardo la sua posizione, quella dei punti di interesse da visitare, e fornendo un meccanismo di navigazione che possa semplificare al meglio questa fase, che di per sè non deve alienare l'utente e distrarlo dal contesto della visita culturale.
Verrà analizzato in questo capitolo il formato dati che si è deciso di utilizzare per le varie informazioni geospaziali (tra cui la mappa fisica del sito culturale), per poi fare un'analisi UI/UX relativa alle varie problematiche in fase di design. Infine, verranno analizzate varie tecniche di posizionamento, con uno studio dello stato dell'arte, e si parlerà del sistema di navigazione che si è implementato all'interno di Smart Visit.

4.1 Descrizione del formato e del contenuto dei dati

Per il "proof of concept" della sezione Smart Visit, sono state ipotizzate delle possibili rappresentazioni dei dati utilizzati, a partire da quella per modellare i punti di interesse fino a quella per mappare l'interno di un sito culturale.

4.1.1 JSON per informazioni sui punti di interesse

Il formato JSON (JavaScript Object Notation) [22] è uno dei più comuni formati per lo scambio di informazioni in sistemi client/server, grazie anche alla veloce diffusione del JavaScript per la programmazione web. Permette di rappresentare dati alfanumerici e booleani, oltre che array (anche associativi).
Per la modellazione di un punto di interesse generico, si è creata in un primo momento una struttura ipotetica, per poi migrare a quella che sarebbe stata l'effettiva rappresentazione JSON del punto di interesse (essendo il formato deciso a priori per l'intero sistema VASARI). Senza dubbio, la rappresentazione finale è molto più complessa di quella ipotizzata solo nello sviluppo di Smart Visit: tuttavia, l'elevata scalabilità del formato (basandosi fondamentalmente su coppie chiave-valore associate) ha permesso nelle fasi iniziali di semplificare queste rappresentazioni, oltre che garantire in ogni momento un possibile cambio di rappresentazione (una scheda VASARI completa o semi-completa). Le informazioni più rivelanti per quello che è il fine di Smart Visit sono l'ID del punto di interesse (e del sito culturale), il suo nome (e del sito culturale), le sue coordinate (espresse in termini di latitudine e longitudine), il piano a cui appartiene, e i contenuti culturali (testuali, o multimediali intesi come URL remoti). A seconda della necessità di avere tutta la rappresentazione di un'opera, o solamente alcuni dettagli, in ogni singola pagina dell'app, la rappresentazione stessa potrà essere parziale o

completa (con alcuni elementi fissi, come il proprio ID o quello del sito di riferimento) per ottimizzare tempi di trasmissione e dati salvati in RAM in un determinato istante di tempo (che potrebbero essere abbastanza significativi). Ad esempio, per la visualizzazione dell'anteprima di un'opera in un contesto off-site, basterebbero poche informazioni testuali, mentre per la visualizzazione dei contenuti multimediali, la quantità di dati necessaria aumenterebbe significatamente. Un esempio di rappresentazione JSON di un punto di interesse è la seguente:

```
{
    "id": "47124",
    "siteID": "98731",
    "name": "Nascita di Venere",
    "description": "...",
    "author": "Sandro Botticelli",
    "date": "1485",
    "floor": 1,
    "imageURL": "...",
    "coordinates": [43.76801027601708, 11.255718469619751],
    "contents:" {
        "images": [
            {
                "title": "...",
                "url": "..."
            },
            .
            .
            .
        ],
        "audios": [
            {
                "title": "...",
                "url": "..."
            },
            .
            .
            .
        ],
        "videos": [
            {
                "title": "...",
                "url": "..."
            },
            .
            .
            .
        ],
        "extended": [
            {
                "title": "...",
                "url": "..."
            },
            .
            .
            .
        ]
    }
}
```

Listato 4.1: Esempio di JSON contenente informazioni su un punto di interesse

4.1.2 GeoJSON per la modellazione del sito culturale

Un'elemento fondamentale per la guida del turista è senza dubbio l'utilizzo di una mappa che possa dare un feedback della propria posizione, soprattutto in relazione al sito fisico (in quale stanza o piano si trovi). Occorre, cioè, delineare quello che è il perimetro del sito, ed i suoi ostacoli, per fare in modo di dare all'utente la possibilità di orientarsi autonomamente o farsi guidare dall'app (tramite un sistema di navigazione). Nella progettazione di questa sezione dell'app, si sono valutate diversi approcci: da un lato l'utilizzare una vera mappa del sito, disegnata o generata con appositi strumenti, e utilizzare delle coordinate relative rispetto ad un punto di origine, dall'altra invece quella di utizzare dei formati specifici per questo tipo di applicazioni.
I vantaggi della prima sarebbero quelli di dare un risultato visivo complessivamente migliore, permettendo anche di evitare la definizione esplicita di ostacoli, scale, ascensori o vie di fuga (essendo tutto rappresentato a priori). Tuttavia, potrebbe trovare riscontro solo per i siti indoor, mentre per i luoghi all'aperto la situazione sarebbe diversa: basti pensare ad un parco o un sito archeologico (per cui basta dare un perimetro e visualizzarlo in una normale mappa). La volontà di generalizzare i siti culturali indoor e outdoor, e di non dover gestire logiche diverse per il tipo di coordinate o la gestione degli interni (oltre già a quelle di posizionamento che verranno discusse in seguito), ha condotto alla seconda opzione. Si è scelto di utilizzare il formato GeoJSON [3], ossia un formato di definizione di geometrie spaziali e features geografiche che proviene direttamente dal formato JSON. Permette di modellare punti nello spazio, linee spezzate, poligoni e collezioni di questi elementi, per cui si possono generare delle mappe più o meno complesse, a seconda della complessità globale del GeoJSON generato. Un esempio di GeoJSON, tratto da [17], è il seguente:

```
{
    "type": "FeatureCollection",
    "features": [
      { "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [102.0, 0.5]
          },
          "properties": {
            "prop0": "value0"
          }
        },
      { "type": "Feature",
        "geometry": {
          "type": "LineString",
          "coordinates": [
            [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
            ]
          },
        "properties": {
          "prop0": "value0",
          "prop1": 0.0
          }
        },
      { "type": "Feature",
         "geometry": {
           "type": "Polygon",
           "coordinates": [
             [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
               [100.0, 1.0], [100.0, 0.0] ]
             ]
         },
```

```
33            "properties": {
34              "prop0": "value0",
35              "prop1": {"this": "that"}
36              }
37            }
38      ]
39  }
```

Listato 4.2: Esempio di un GeoJSON [17]

Questo esempio, utilizzando il tool online [28], può essere visualizzato graficamente in Figura 4.1.

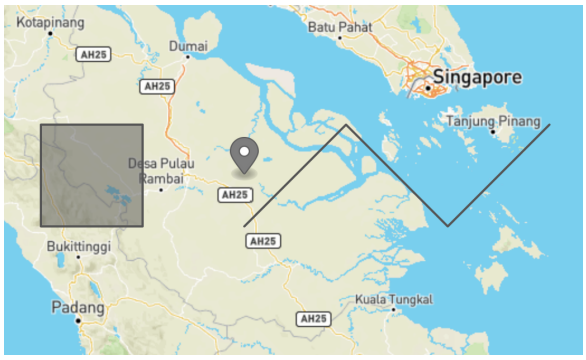

Figura 4.1: Visualizzazione delle informazioni formattate nel GeoJSON di esempio tramite il tool online gratuito geojson.io [28]

4.2 Librerie utilizzate

Vengono elencate di seguito le librerie open source utilizzate per l'implementazione della prova di fattibilità del modulo Smart Visit del progetto VASARI.

4.2.1 Leaflet

Leaflet [27] è una libreria open source, ottimizzata per lo sviluppo di applicazione mobile, che permette di gestire facilmente e con buone performance delle mappe interattive. Avendo alle spalle una documentazione e un codice sorgente ben documentato, sono stati creati una serie di plugin dalla vasta community capaci di personalizzare in vari aspetti (grafici e interattivi) l'utilizzo della mappa, in relazione al contesto applicativo. Parallelamente a questa libreria, se ne sono valutate altre con funzionalità simili, per cercare di trovare pro e contro di ognuna e fare una scelta mirata (essendo la mappa un elemento centrale di Smart Visit). La leggerezza di questa libreria, soprattutto in ambito di utilizzo in mobilità, ed il suo essere minimale in certe operazioni, l'ha resa ideale per il contesto applicativo di questa tesi. Inoltre, permette di utilizzare il formato GeoJSON per le rappresentazioni di geometrie spaziali, in modo tale di creare una mappa del sito culturale sia indoor che outdoor. La libreria permette l'inserimento di numerosi livelli in una mappa, che possono essere gestiti in maniera indipendente e con un propria logica, riuscendo così a generare delle strutture di mappe complesse, mantenendo un alto livello di modularità tra i vari livelli. In Figura 4.2, tratta da [18], viene riportato un esempio di mappa con un livello di GeoJSON.

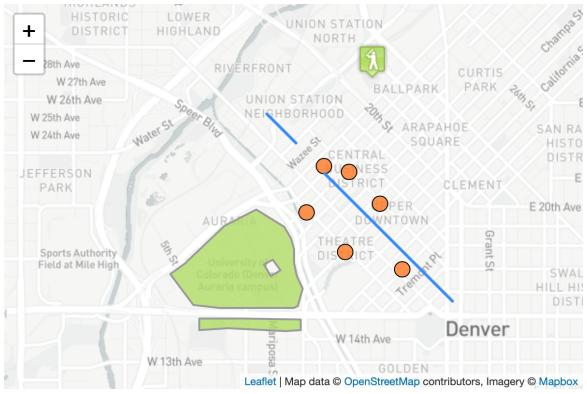

Figura 4.2: Esempio di mappa contenente dei GeoJSON creata con *Leaflet* [18]

4.2.2 Turf

Turf [41] è una libreria open source che serve ad analizzare strutture geospaziali, formattate in GeoJSON, al fine di estrapolarne informazioni utili. Permette ad esempio di calcolare l'area di un poligono, la lunghezza di una retta o la distanza tra due punti (scegliendo l'appropriata l'unità di misura). Un altro aspetto interessante della libreria è la sua capacità di generare GeoJSON a partire da altri: ad esempio, permette di generare una retta che interseca dei punti, oppure di trovare l'intersezione tra più poligoni.
Utilizzando questa libreria, si sono potuti eseguire dei calcoli che richiedono algoritmi più o meno complessi, tramite delle funzioni ben testate: ad esempio, è stato possibile capire facilmente se l'utente si trova all'interno del sito culturale (ossia, del perimetro del poligono che lo definisce), piuttosto che calcolare il cammino più breve, tramite l'algoritmo A*, tra la sua posizione e quella del prossimo punto di interesse da visionare (tenendo anche conto di eventuali ostacoli come pareti e arredamenti vari). Nelle Figure 4.3 e 4.4 sono rappresentati, rispettivamente, il centroide di un poligono e il percorso più breve tra due punti (tenendo conto di un ostacolo) calcolati con *Turf*.

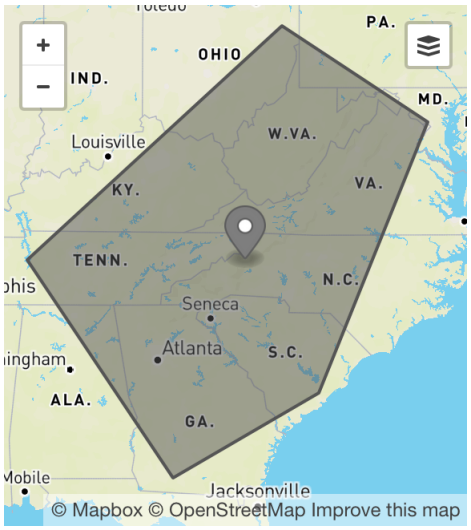

Figura 4.3: Centroide di un poligono calcolato con *Turf* [8]

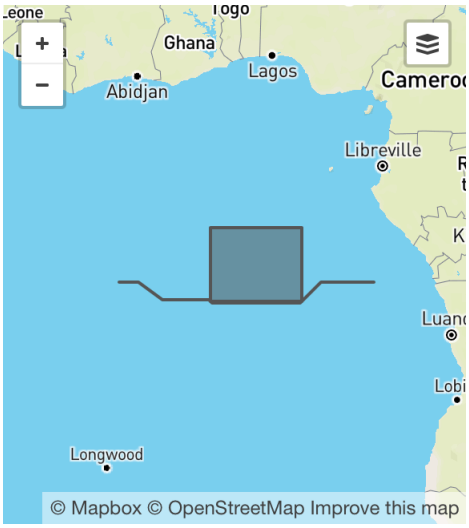

Figura 4.4: Shortest path con un ostacolo tra due punti generato con *Turf* [37]

4.2.3 TensorFlow

Più avanti in questo capitolo si parlerà di tecniche di posizionamento utente per quanto riguarda il contesto indoor: tra queste ci saranno tecniche basate sul Machine Learning (e in particolare, su Reti Neurali). Una libreria open source che permette una semplice gestione di modelli di intelligenza artificiale è *TensorFlow* [40], sia per la fase di training (offline), che per quella di predizione di nuovi sample (online). Nel secondo caso, è stata utilizzata una variante della libreria scritta in JavaScript, *TensorFlow.js*. Tramite un notebook scritto in Python, è stato studiato il dataset e sono state testate le varie configurazioni del modello prima di arrivare a quella definitiva. É stato creato un modello *Keras*, libreria specializzata nell'apprendimento automatico tramite reti neurali contenuta nella più generica libreria *TensorFlow*, che poi è stato convertito in due file. Il primo (.JSON), contiene una rappresentazione architetturale della rete, mentre il secondo (.bin), contiene i pesi effettivi del modello. Questi due file, dati in input alla libreria *TensorFlow.js*, permettono di caricare il modello precedentemente creato direttamente all'interno del progetto *Ionic/Angular* relativo all'applicazione mobile VASARI. A questo punto la nuova libreria permette, dato un nuovo sample (con lo stesso formato di quelli usati per l'addestramento), di predire la classe di appartenenza. Il caricamento e la gestione del modello di Machine Learning avviene perciò, all'interno del proof of concept, direttamente sull'applicazione. In un contesto reale si potrebbe considerare il fatto di spostare questa logica su un determinato server, in maniera tale da evitare di scaricare i due file, soprattutto il file binario, che potrebbe essere abbastanza pesante per reti complesse.

4.3 Analisi UI/UX

Lo scopo principale di questa sezione è quella di guidare l'utente lungo il percorso fisico nel sito culturale: a tal scopo, è stata progettata una pagina specifica all'interno dell'applicazione VASARI (collegata ad altre per la fruizione dei contenuti o la gestione del percorso).
La pagina in questione, oltre a contenere una mappa raffigurante il sito culturale di riferimento, deve dare dei feedback all'utente riguardo il proprio percorso, lo stato della visita, la posizione assoluta e relativa dei punti di interesse rispetto alla propria posizione, e fornire una serie di funzioni di varia utilità.

4.3.1 La mappa

Una delle funzionalità di maggior successo, al giorno d'oggi, per un'applicazione mobile è senza dubbio l'utilizzo di una mappa. Uno studio di mercato, riportato in [35] indica come, nel panorama globale, le applicazioni che fanno uso di una mappa vengono utilizzate quasi il doppio rispetto ad altre che non lo fanno. A differenza di una mappa cartacea, una mappa digitale offre più opportunità nell'utilizzo: permette all'utente di orientarsi e vedere in tempo reale la propria posizione/orientamento, e, grazie ad una serie di gestures ed eventi, offrire contemporaneamente informazioni aggiuntive.

Stato dell'arte

Dietro alla progettazione di un'applicazione mobile con mappa, si celano diversi punti critici e problematiche. Al di là del dispositivo, vi sono molti fattori che possono influenzare negativamente l'utilizzo di una mappa, come riporta [35]: ad esempio, condizioni climatiche avverse (pioggia, vento, alte temperature) o semplicemente condizioni di scarsa illuminazione e visibilità. Perciò, se da un lato l'app può essere il centro dell'attenzione da parte dell'utente, dall'altro può distrarlo dal contesto in cui si trova.
La ricerca per quanto riguarda il mobile mapping è ancora agli albori: sono presenti però

un insieme di aspetti interessanti da analizzare, sempre riportati in [35]. Il primo è il tipo di applicazione che si vuole avere:

- I *Location-based services* sono applicazioni che modificano la mappa e le informazioni che essa contiene in relazione alla positione corrente dell'utente. Esse abilitano una "visione egocentrica", ossia un focus in quella che è la locazione dell'utente e una cambio dinamico di orientamento in maniera tale che il suo puntamento sia sempre verso l'alto, ma non sia sempre il nord. Un esempio di questo tipo di applicazioni sono i sistemi di navigazione e ricerca di un percorso.

- l'*Adaptive cartography* corrisponde all'utilizzo di mappe che cambiano design a seconda del tipo di utilizzo e del contesto dell'utente, per cui sono rilevanti sia la posizione corrente, che l'ambiente circostante.

- La *Volunteered geographic information (VGI)* è un modello di applicazione che si basa totalmente sulle informazioni ambientali e spaziali, come il livello di crowdsourcing (la cooperazione volontaria di più individui al fine di raggiungere un determinato obiettivo) di un luogo, o altre informazioni che derivano comunque dall'utilizzo da parte di altri utenti.

Il design e le funzionalità di un'applicazione dipendono strettamente dalla categoria di appartenenza, oppure da più categorie se condivide funzionalità di ognuna. Ad esempio, per Smart Visit sarà necessario avere una mappa che si basi fondamentalmente sulla posizione dell'utente e dei punti di interesse, ma anche, in una futura implementazione, sul livello di "affollamento" che si ha in un punto di interesse (o all'ingresso di un sito culturale). Inoltre, il suo design potrebbe cambiare a seconda del tipo di sito culturale (indoor o outdoor), o di diversi elementi ambientali (come eventuali scale per raggiungere altri piani, servizi igienici o vie di fuga).
Di seguito vengono riportati, a partire da [35], alcuni elementi specifici del design di una mappa mobile:

- **Grandezza**
  Molte applicazioni mobile tendono a massimizzare la mappa (in relazione alla grandezza dello schermo del device), e piazzare eventuali funzionalità all'interno di un *Hamburger Menù* (un menù che si apre al click di un'icona, che in genere rappresenta tre linee orizzontali parallele), e renderla reattiva ad un cambiamento di aspect ratio verticale e orizzontale.

- **Caricamento**
  A causa dell'impatto che una mappa può avere in termini di peso, si consiglia il caricamento progressivo della mappa utilizzando dei tilesets e salvando le informazioni essenziali in cache (per velocizzare il recupero delle stesse). Un tileset è un'immagine che viene scomposta in altre immagini sotto forma di griglia, in maniera tale da velocizzarne il caching ed il caricamento e nel complesso la reattività di una mappa.

- **Punti di interesse**
  Una mappa che rappresenta solo lo spazio circostante e l'utente è fondamentalmente "nuda". Bisogna arricchirla con punti di interesse e altri elementi che diano un contesto applicativo e permettano all'utente di interagire con essa.

- **Simbolismo**
  L'utilizzo di icone, simboli o marker deve garantire un contrasto e una luminosità adeguata rispetto al tile sottostante ed a possibili fattori ambientali di disturbo. La posizione dell'utente deve essere più precisa possibile (più avanti si analizzeranno tecniche di posizionamento per migliorare questo aspetto), e quando non lo è bisognerebbe dare un

feedback all'utente (un range che si allarga e si restringe in caso di alta o bassa precisione). Anche la rappresentazione di elementi ambientali o relativi alla navigazione deve distinguersi dal resto, per evitare confusione nell'utente e un aumento di tempo di utilizzo della mappa (che deve mantenersi breve, sintomo di efficacia nel suo scopo).

- **Eventi**
  L'utente deve avere la possibilità di scatenare eventi sulla mappa, con dei click o altre azioni.

Implementazione

Analizziamo ora le scelte implementative relative alla mappa, includendo il design (UI), le funzionalità offerte e l'interazione con l'utente (UX).

- **Il tileset**
  Come detto in precedenza, e come d'altronde è implementato nella maggior parte delle librerie (tra cui *Leaflet*), per la gestione dello "sfondo" di una mappa si consiglia l'utilizzo di un set di tessere che vengono caricate quando effettivamente il livello di zoom globale è tale da doverle inserire. Inoltre, venendo salvate in cache, questa operazione risulta veloce ed efficiente. Si è deciso di utilizzare due diversi tileset, uno chiaro e uno scuro, forniti gratuitamente dal tile server *Stadia Maps* [1], chiamati *Alidade Smooth* e *Alidade Smooth Dark*, che sono mostrati nelle Figure 4.5 e 4.6. Si offre all'utente la possibilità

Figura 4.5: *Alidade Smooth* [1]

Figura 4.6: *Alidade Smooth Dark* [1]

di selezionare il tema della mappa a piacimento, tramite una delle opzioni di gestione mappa che vengono descritte in seguito.

- **Opzioni**
  Tramite un insieme di *FAB*, è stato creato un menù tramite il quale l'utente può eseguire alcune funzionalità sulla mappa. In particolare, oltre a fare uno switch del tile (da chiaro a scuro e viceversa), è possibile focalizzare la vista della mappa sulla propria posizione, piuttosto che sulla posizione del prossimo punto di interesse. Queste funzionalità sono rappresentate nelle Figure 4.7 e 4.8. Inoltre, come si descriverà più in avanti, sono state inserite altri due pulsanti posti in sovraimpressione alla mappa. Le due funzionalità offerte sono quella di visione di eventuali scale e ascensori per poter passare da un piano all'altro, mentre l'altra è la possibilità di abilitare il percorso più breve per raggiungere il prossimo punto di interesse. I due pulsanti sono rappresentati in Figura 4.9. Infine, essendo presente la possibilità di scannerizzare dei codici QR durante tutta la visita (anche di punti di interesse non appartenenti al proprio percorso), si è inserito un *FAB* in basso a destra della schermata (Figura 4.10). L'importanza di questa funzionalità è sottolineata dalla grandezza del pulsante e dalla sua posizione (ossia, la posizione

primaria che è presente in moltissime applicazioni mobile di svariato genere). In Figura

Figura 4.7: Floating Action Button tramite il quale si possono espandere delle funzionalità di gestione della mappa

Figura 4.8: Funzionalità di gestione della mappa

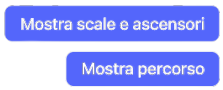

Figura 4.9: Funzionalità aggiuntive per la mappa

Figura 4.10: Floating Action Button per utilizzare lo scanner QR

4.11 è mostrata un'instantanea di cattura di codici QR tramite il plugin *BarcodeScanner*: Nel frammento di codice seguente è riportato un esempio di utilizzo del plugin:

Figura 4.11: Instantanea di cattura di codici QR

```
1    import {BarcodeScanner} from
     '@ionic-native/barcode-scanner/ngx';
2    .
3    .
4    .
5    constructor(private barcodeScanner: BarcodeScanner) {}
6    .
7    .
8    .
```

```
9       this.barcodeScanner.scan({
10          prompt: 'Inquadra un codice QR',
11          disableSuccessBeep: true,
12          resultDisplayDuration: 0
13      }).then(data => {
14          console.log(data.text);
15      });
16
```

- **I punti di interesse**
  I punti di interesse sono stati modellati con un layer apposito all'interno della mappa creata con *Leaflet*. Sono visualizzati come dei marcatori di posizione, che assumono diversi colori. Il colore rosso, di default, indica qualsiasi punto di interesse. Il colore verde, invece, è rappresentativo solamente del prossimo punto di interesse, secondo la logica del percorso (in ordine se il percorso è ordinato, il più vicino se il percorso è libero). Questi marker sono anche interattivi, per cui è possibile cliccare e navigare direttamente ai dettagli di quel punto di interesse (la cui pagina verrà descritta nel capitolo successivo). Si presuppone, a rigor di logica, che la posizione di un punto di interesse sia confinata all'interno del sito culturale a cui fa riferimento.

- **L'utente**
  La gestione dell'utente, della sua posizione e del suo orientamento è uno degli aspetti più critici del lavoro di tesi. Verranno approfonditi in seguito le varie tecniche con cui posizionare l'utente in maniera realistica a seconda del caso d'uso (indoor o outdoor). Così come per i punti di interesse, anche per l'utente è previsto un layer apposito all'interno della mappa, indipendente dagli altri e con la propria logica (ossia, il posizionamento). Per la scelta del tipo di icona da utilizzare, si sono valutate alcune problematiche espresse in [6]. In particolare, negli esperimenti condotti nel paper di ricerca, si è osservato come ci sia una grande misinterpretazione del simbolo della bussola, e in generale una confusione per quanto riguarda l'orientamento dell'utente. Nonostante la convenzione attuale sia l'utilizzo di un cerchio (la posizione), di un range (la precisione di localizzazione) e un arco pieno (la direzione verso cui punta l'utente), si è deciso di utilizzare un'icona quanto più significativa per dare un'interpretazione chiara di dove l'utente stia puntando con il proprio device. L'icona rappresenta una struttura formata da un cerchio, che sta ad indicare la posizione attuale, e un triangolo, con la punta posizionata coerentemente con la direzione dell'utente.

- **Il sito culturale**
  Come detto all'inizio del capitolo, si è deciso di utilizzare il formato GeoJSON per la modellazione di informazioni geometriche sullo spazio, che è facilmente integrabile nella libreria *Leaflet*. Per rappresentare il sito culturale, si è generato un GeoJSON che contiene all'interno la geometria del perimetro dei vari piani, eventuali scale e ascensori, servizi igienici e vie di fuga. A seconda del contesto, si visualizzeranno una o più caratteristiche del sito. Ad esempio, è stata inserita la possibilità di cambiare dinamicamente il piano di visualizzazione, tramite un *Segment Menù* posto immediatamente sopra la mappa (in Figura 4.12). Un'altra esempio è la comparsa di un *Popup* (in Figura 4.13), contenente una scheda generica del sito culturale (nome e immagine di default), che viene visualizzato quando lo zoom (controllato dall'utente) scende sotto una certa soglia. In Figura 4.14 è rappresentato il sito culturale, integrato con l'icona dell'utente e i marker dei punti di interesse, mentre in Figura 4.15 è rappresentato lo stesso sito nel momento in cui viene selezionata l'opzione per visualizzare gli strumenti di movimento tra piani (scale e ascensori), che assumono un diverso colore per essere distinguibili dalle pareti.

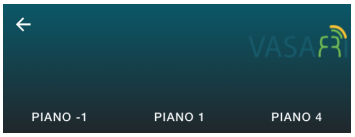

Figura 4.12: *Segment Menù* per il cambiamento dinamico di piano

Figura 4.13: Popup del sito culturale

Per concludere, vengono riportati nelle Figure 4.16 e 4.17 i rendering nei sistemi operativi Android e iOS della pagina fin'ora trattata, includendo tutti gli elementi descritti.

4.3.2 Stato della visita

Durante la visita, l'utente usufruirà dei contenuti culturali proposti per ogni punto di interesse, riuscendo così a "completarlo". La visita avrà fine quando tutti i punti di interesse saranno completati (anche in seguito a cambiamenti dinamici del percorso, come l'aggiunta o la rimozione di punti di interesse). Bisogna però dare all'utente una panoramica sulla situazione dello stato della visita in ogni singolo istante. Essendo un aspetto abbastanza specifico per l'applicazione, non vi è un vero e proprio stato dell'arte, quanto piuttosto dei componenti grafici o meccanismi consigliati.

Implementazione

Per l'implementazione di questa parte, si è deciso di procedere con un approccio gerarchico. Questo perchè si vuole evitare che, in ogni singolo istante, si forzi l'utente a dover visionare per forza informazioni che potrebbero essere superflue (implicando la navigazione verso altre pagine o l'apertura di popup ed elementi grafici). Se l'utente vuole sapere soltanto lo stato di avanzamento della visita, può bastare semplicemente una barra di caricamento. Se invece l'utente vuole vedere nel dettaglio quali punti di interesse sono stati completati, dovrebbe essere mostrata una breve lista spuntata (per indicare quali elementi sono stati completati e quali no). Infine, se l'utente vuole, oltre a visionare il progresso, anche modificare il percorso, allora si dovrà navigare opportunamente in una pagina dedicata dell'app.
Gli elementi principali di questa sezione sono:

- **La barra di caricamento**
  Posta al di sotto della mappa, si riempie coerentemente con le azioni dell'utente e con la visione dei vari punti di interesse (come mostrato nelle Figure 4.18, 4.19 e 4.20). Al click su di essa, si espande un *Pullup*, mostrando un pulsante (in Figura 4.21) con cui si può navigare verso i dettagli del percorso (potendolo perciò modificare durante la visita).

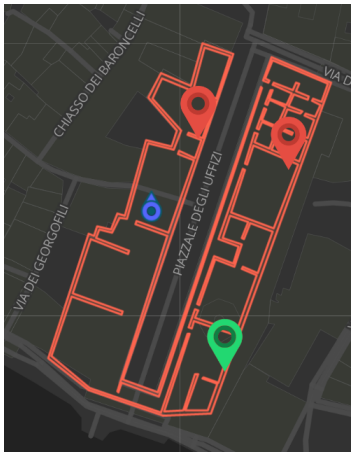

Figura 4.14: GeoJSON del perimetro del sito culturale, integrato con l'utente e i punti di interesse, rappresentato in *Leaflet*

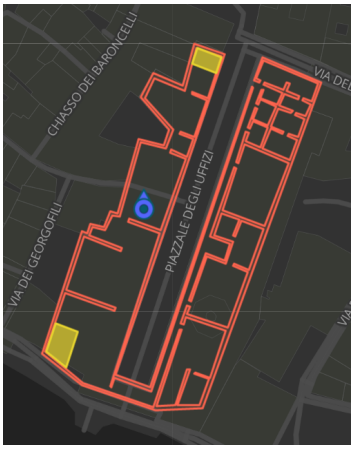

Figura 4.15: GeoJSON del perimetro del sito culturale con scale/ascensori, integrato con l'utente rappresentati in *Leaflet*

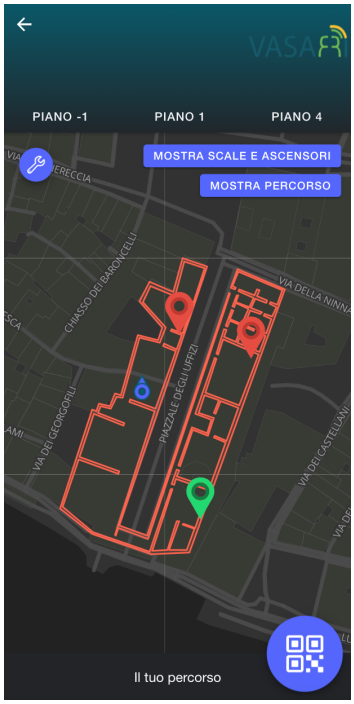

Figura 4.16: Pagina di guida dell'utente renderizzata nel sistema operativo Android

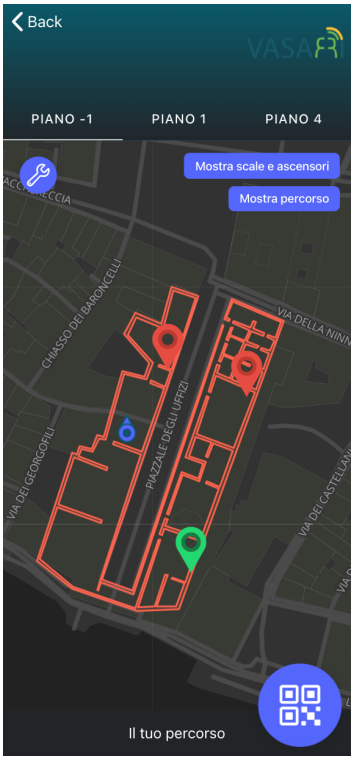

Figura 4.17: Pagina di guida dell'utente renderizzata nel sistema operativo iOS

- **Lista dei punti di interesse**
  Dopo aver cliccato sulla barra di caricamento, viene aperto un *Pullup* contenente i vari punti di interesse che formano il percorso, appartenenti per definizione al medesimo sito culturale. Essi sono disposti in ordine crescente di piano, e per ognuno è fornito un *Avatar* tondo della propria immagine, il nome e una spunta verde che rappresenta il completamento del punto di interesse. In Figura 4.22 è mostrata la lista spuntata dei

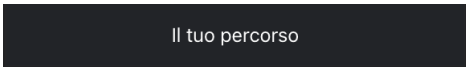

Figura 4.18: Barra di caricamento all'inizio della visita

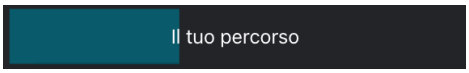

Figura 4.19: Progresso mostrato nella barra di caricamento

punti di interesse.

- **Prossimo punto di interesse**
  L'elemento che deve essere visionato secondo la logica del percorso scelto (il prossimo in ordine, piuttosto che quello più vicino in caso di percorso libero), viene rappresentato con una *Card* contenente un'immagine più grande, il nome, il riferimento alla distanza che si ha con esso, e delle frecce che permettono di indicare all'utente la direzione da prendere (discorso che verrà approfondito alla fine di questo capitolo). La *Card* è rappresentata in Figura 4.23.

Si noti che con questa scelta progettuale si è moltiplicata la modalità di visione dell'informazione sullo stato del percorso: se da un lato significa dover gestire opportunamente e coerentemente l'informazione su più livelli (gestione che però viene semplificata molto dal pattern MVC), dall'altro si è resa scalabile l'interazione dell'utente, che vorrebbe visionare, come detto, soltanto determinate informazioni (il progresso come barra di caricamento, un elenco con spunte, o i dettagli completi del percorso).

4.4 Posizionamento

Uno degli argomenti di maggior spicco all'interno della fase sperimentale di questa tesi è stata la ricerca di metodi di "positioning". Con questo termine intendiamo l'attribuzione di una posizione, in coordinate assolute o relative, all'utente in un determinato istante, con conseguente feedback sulla mappa (movimento dell'icona corrispondente). E' un processo cruciale per la visita culturale: dovendosi orientare per raggiungere i vari punti di interesse, il sistema con con cui l'utente può visualizzarsi in tempo reale sulla mappa deve essere quanto più possibilmente preciso. Vengono analizzate di seguito le tecniche di positioning per i contesti outdoor e indoor.

4.4.1 Outdoor

Si tratta della situazione per cui il sito culturale si trova in uno spazio aperto e libero da eventuali schermature per segnali di vario tipo. Un esempio di sito outdoor potrebbe essere uno scavo archeologico, piuttosto che una riserva o un frammento di città.
La tecnologia più in voga per questo contesto è il Global Positioning System (GPS) [43], ossia un sistema di navigazione satellitare globale che permette di fornire posizione, velocità e sincronizzazione nel tempo.
I tre elementi principali di questo sistema sono:

- I satelliti che orbitano intorno alla terra, in particolare un gruppo di 24 satelliti disposti i 6 piani orbitali.
- Dei segmenti di controllo, posti in tutti i continenti, che monitorano i satelliti e le trasmissioni.

Figura 4.20: Barra di caricamento quando la visita è stata completata (tutti i punti di interesse visionati)

Figura 4.21: Pulsante di modifica del percorso all'interno della barra di caricamento

- Il ricevitore on-board nel device utente.

Utilizzando una tecnica chiamata "trilaterazione", il device raccoglie i segnali provenienti da almeno 4 satelliti. Ogni volta che arriva un segnale da parte di un satellite, si genera una sfera (essendo in un mondo a tre dimensioni) con raggio misurato in base all'intensità del segnale. Viene scelto il punto terrestre più vicino all'intersezione tra le sfere per ottenere la vera e propria posizione del device.
L'accuratezza del GPS può dipendere da alcuni fattori, come il numero di satelliti disponibili, la ionosfera o l'ambiente in cui ci si trova. Alcuni fattori possono ridurre sensibilmente l'accuratezza, come ostacoli fisici, effetti atmosferici, interferenze artificiali. Essa risulta però essere abbastanza elevata in spazi aperti e con poche interferenze, motivo per il quale può essere considerata ottima per il contesto di visita culturale outdoor.
All'interno del lavoro di tesi è stato utilizzato il plugin *Geolocation* per la gestione delle informazioni riguardanti la posizione del device tramite GPS: di seguito è riportato un esempio di utilizzo di tale plugin: si vede come è possibile sia ottenere la coppia latitudine/longitudine in uno specifico istante di tempo, e sia iscriversi ai cambiamenti di tali coordinate per una gestione più elaborata (nel caso di Smart Visit, la gestione del movimento su mappa dell'icona dell'utente).

```
1  import {Geolocation} from '@ionic-native/geolocation/ngx';
2  .
3  .
4  .
5  constructor(private geolocation: Geolocation) {}
6  .
7  .
8  .
9  this.geolocation.getCurrentPosition({
10     enableHighAccuracy: true,
11     timeout: 5000
12 }).then(data => {
13     console.log(data.coords.latitude);
14     console.log(data.coords.longitude);
15 });
16 .
17 .
18 .
19 this.geolocation.watchPosition().subscribe(data => {
20     console.log(data.coords.latitude);
21     console.log(data.coords.longitude);
22 });
```

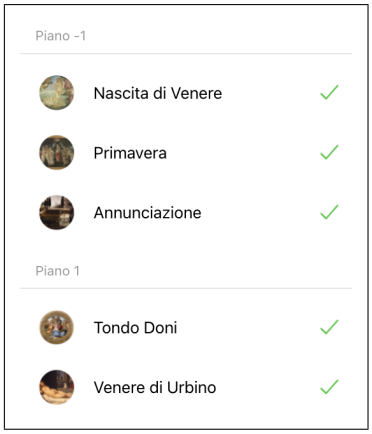

Figura 4.22: Lista dei punti di interesse che formano un percorso, con una spunta verde quando sono stati completati

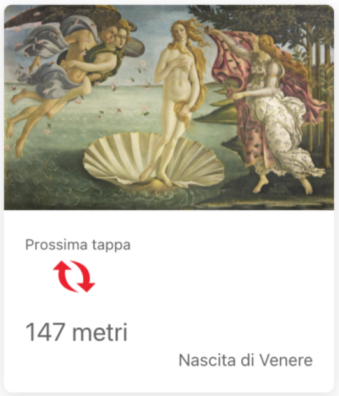

Figura 4.23: *Card* del prossimo punto di interesse da visionare

4.4.2 Indoor

Per quanto riguarda il contesto indoor (edificio o spazio al chiuso), la tecnologia GPS risulta essere non affidabile. Al contrario di un ambiente outdoor, un ambiente indoor è molto più complesso in termini di presenza, forma e grandezza di ostacoli (anche in movimento). Lo sviluppo di IPS (Indoor Positioning Systems) stabili e performanti è sempre più richiesto per moltissimi ambiti, soprattuto per applicativi che fanno della localizzazione in ambiente interno la propria funzionalità primaria. Lo sviluppo di questi sistemi implica però dei costi: è necessario adoperare ogni luogo con dell'attrezzatura specifica, ma soprattutto è difficile (se non impossibile) generalizzare alcune tecniche per più ambienti al chiuso. Inoltre, per tecniche basate sul Machine Learning (come vedremo più avanti), bisogna creare dei dataset, allenare e generare un modello per ogni singolo sito. Infine, avendo dall'altro lato il bisogno di interagire con dei dispositivi utente, bisogna progettare il tutto in maniera tale da non dover limitare le funzionalità per device vecchi o che non supportano l'utilizzo di determinati sensori.
Prima di parlare nel dettaglio di come si è deciso di implementare il sistema IPS del proof of concept di Smart Visit, introduciamo alcune nozioni di base per la discussione di questo argomento. Per prima cosa, bisogna introdurre alcuni concetti:

- **Line of sight (LOS)**
  Si ha una situazione di LOS (mostrata in Figura 4.24, tratta da [34]) quando vi è visibilità diretta tra trasmettitore e ricevitore: non vi sono, ossia, ostacoli nel mezzo che

possono interferire. Si tratta, per alcuni tipi di tecnologie, di una situazione ideale ma poco frequente in un caso reale.

- **Non-line of sight (NLOS)**
  Si ha una situazione di NLOS (mostrata in Figura 4.25, tratta da [34]) quando, invece, la presenza di ostacoli (anche mobili) rende difficile il passaggio del segnale, che viene oscurato parzialmente o totalmente. Alcune tecniche di "ranging" soffrono particolarmente di questo fenomeno, che è intrinseco di alcune tecnologie comuni. **Nodi ancora**

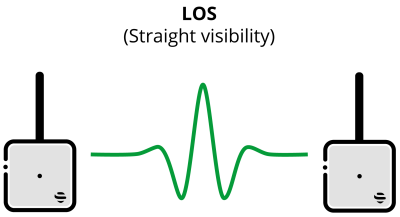

Figura 4.24: Line of sight (LOS) [34]

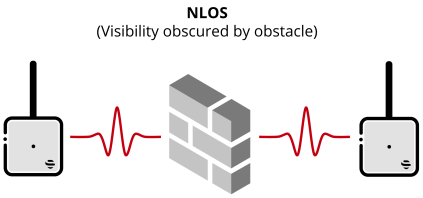

Figura 4.25: Non-line of sight (NLOS) [34]

Un'ancora è un nodo trasmettitore la cui posizione è globalmente nota a priori, tramite la quale si può effettuare un'analisi della distanza tra essa e il ricevitore. In seguito, dando come input queste distanze, è possibile effettuare la localizzazione dell'utente all'interno del sito indoor. In genere, l'accuratezza della localizzazione dipende dal numero di nodi ancora e da eventuali malfunzionamenti. Inoltre, se si ha un numero abbastanza ridotto di nodi ancora e, contemporaneamente, una distanza troppo elevata, l'errore della stima della posizione può essere veramente alto [13]. In Figura 4.26, tratta da [31], è mostrata un esempio di scenario formato da un nodo target (l'utente) e tre nodi ancora.

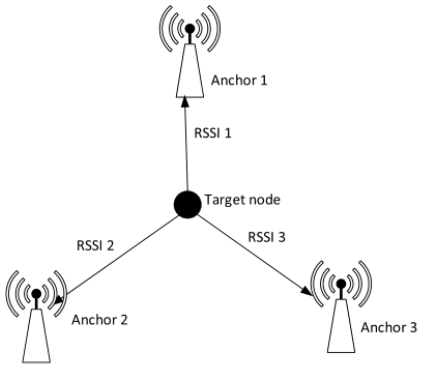

Figura 4.26: Diagramma schematico di posizionamento utilizzando nodi ancora [31]

In [31] vengono elencate le principali tecniche di "ranging", ovvero di calcolo della distanza tra il nodo target (l'utente) e un nodo ancora:

- **RSSI (Received Signal Strenght Indicator)**
  Rappresenta la forza con cui il segnale, partendo dal trasmettitore, arriva al ricevitore, e tramite la quale viene calcolata la distanza da esso. É in generale il parametro più facile da misurare, ma anche quello che porta più inaccuratezza, soffrendo intrinsecamente di fenomeni di rifrazione, riflessione, dissolvenza, dispersione, e in generale quando si hanno situazioni di NLOS. Risulta essere una tecnica a basso costo, perchè non richiede hardware aggiuntivo, e con una bassa complessità.

- **ToA (Time of Arrival)**
  Più accurato del precedente, utilizza il tempo di propagazione del segnale per calcolare la distanza fra il trasmettitore ed il ricevitore. Tuttavia, eventuali tempistiche di sincronizzazione o calcolo possono ridurre sensibilmente l'efficacia di questa tecnica. Risulta essere una tecnica costosa a causa di moduli aggiuntivi necessari per ridurre gli errori di sincronizzazione, ma tra le migliori per quanto riguarda la precisione nella determinazione della distanza verso il nodo ancora.

- **TDoA (Time Difference of Arrival)**
  Similmente al ToA, questa tecnica si basa sul tempo di propagazione per determinare la distanza. La differenza è che con il TDoA vengono coinvolti anche altri nodi ancora per il calcolo.

- **AoA (Angle of Arrival)**
  É una tecnica di "ranging" rinforzata, in cui basta avere due nodi per dare una stima della posizione. Utilizza l'angolo che il segnale forma con il ricevitore. Ha un'ottima precisione per trasmettitori posti a brevi distanze dai ricevitori, ma richiede dell'hardware extra.

- **CSI (Channel State Information)**
  Come la precedente, è una tecnica di "ranging" rinforzata. Può essere usata per avere una stima migliore del segnale ricevuto in tutta la lunghezza di banda del segnale, utilizzando sia l'informazione sull'ampiezza che sulla fase. É una tecnica più complessa rispetto alle altre.

In Tabella 4.1, tratta da [31], sono riassunti i vantaggi e gli svantaggi delle tecniche appena citate:

Technique	**Advantages**	**Disadvantages**
RSSI	A simple technique for the distance measurement as typically there is no need for extra hardware. Cost effective and less complex.	Provides low precision due to NLOS propagation of signal. Strong optimization and positioning techniques are required for accurate results.
CSI	This has high granularity over RSSI due to both amplitude and phase information of channel frequency.	Complexity is higher than that in RSSI-based and most other IPS techniques.
ToA	ToA provides the highest precision of the measurement among most range-based methods with strict clock synchronization between the transmitter and the receiver.	Expensive due to extra devices and modules needed to lower synchronization error.
TDoA	TDoA can provide better precision than ToA (especially when ToA has notable synchronization error between the clocks).	Is is expensive due to extra devices and modules added to lower synchronization error.
AoA	AoA provides more accuracy for the target node location estimation for short distance of signal propagation.	Requires antenna arrays and extra hardware. It is often hard to implement AoA due to multi path effects.

Tabella 4.1: Vantaggi e svantaggi di diverse tecniche di "ranging" [31]

I metodi di localizzazione a partire da queste tecniche possono essere, sempre secondo [31]:

- **Multilaterazione e trilaterazione**
  In questa tecnica si utilizzano le distanze tra il ricevitore e più nodi ancora, la cui posizione è nota a priori. La trilaterazione è un caso particolare di multilaterazione in cui il numero di ancore è uguale a tre. In uno spazio bidimensionale, vengono generati dei cerchi di raggio uguale alla distanza dalle ancore, e viene presa l'intersezione per determinare la posizione. Si presentano grossi errori in questa stima quando si verificano effetti di NLOS, per cui è necessario accompagnare questa tecnica con altre di manipolazione numerica e assestamento dei dati.

- **Triangolazione**
  Si può usare questa tecnica solo quando l'angolo di arrivo del segnale è disponibile, riducendo a due il numero di ancore necessarie per la stima della posizione. Aumentando il numero di nodi, aumenta anche la precisione. L'accuratezza nella localizzazione può soffrire però anche di piccoli errori nel calcolo dell'angolo.

- **Fingerprinting**
  É una tecnica piuttosto diffusa per il posizionamento indoor, che utilizza tecnologie wireless come il Wi-Fi o BLE combinate a pratiche di Machine Learning. Consiste in due fasi: quella di allenamento offline e quella di testing online. Durante la fase di allenamento, vengono raccolti dei dati (ad esempio, gli RSSI) di varie posizioni note nell'ambiente indoor, e viene generato un modello in grado di predire una posizione a partire da dati appartenenti alla stessa distribuzione. Nella seconda fase, i dati raccolti in tempo reale vengono utilizzati per stimare la posizione dell'utente. Dal punto di vista dell'accuratezza, risulta essere la tecnica migliore (in quanto è specifica per ogni singolo ambiente indoor) e non richiede hardware aggiuntivo. Tuttavia, non è robusta verso cambiamenti topografici, e richiede un costo in termini di tempo per la fase di raccolta dati e sviluppo del modello (fase di training offline).

- **Centroide**
  In questo metodo, viene stimata la posizione calcolando il centroide della struttura geometrica che si genera utilizzando l'informazione di distanza e angolo tra ogni nodo e il nodo target. Risulta essere semplice ma performante solo se accompagnata da particolari algoritmi.

- **DV Hop**
  In questa tecnica occorre trovarsi in un ambiente multi salto, ossia in cui ogni possibile posizionamento dista un salto dall'altro. Le coordinate dell'i-esimo nodo e del numero di salti minimo per raggiungerlo sono salvati in un distance-vector. Nel momento in cui viene trovato il numero di salti effettuati, si può risalire alla nuova posizione utilizzando questa tabella. Questa tecnica necessita di un meccanismo per capire se è stato effettuato un salto, ad esempio tramite il pedometro. Questo tipo di sensore potrebbe però non essere sempre presente a bordo di un device.

In Tabella 4.2, tratta da [31], sono riportati i vantaggi e gli svantaggi delle tecniche appena elencate.

Infine, [31] elenca le diverse tecnologie applicabili ai nodi ancora:

- **Wi-Fi**
  É uno degli IPS più utilizzati in relazione alla loro grande diffusione. Fornendo una grossa copertura, permettono di implementare tecniche di positioning basate su trilaterazione o fingerprinting.

- **RFID (Radio Frequency Identification Device)**
  É una tecnologia economica e durabile contro fattori ambientali come i cambiamenti topologici. Può essere associata a tecniche di fingerprinting.

Technique	Advantages	Disadvantages
Fingerprinting	Provides higher accuracy and it is fairly easy to use as no hardware and other devices are needed.	It is time and cost inefficient due to off-line measurement requirements. Also, not robust against topography changes.
Trilateration	Provides very higher accuracy when applied with other optimization techniques with comparatively less complex.	Higher location error in case of non-ideal case. Demands additional positioning algorithms to be applied.
Triangulation	Provides high accuracy if the measured AoA has good precision. Accuracy increases by increasing the number of anchor nodes.	Location accuracy highly suffers from even a small error in the angle calculations.
DV Hop	Simple and gives better accuracy with more nodes deployed.	Provides very low accuracy for less nodes and used only to estimate some rough localization measurement.
Centroid	Does not need direct measurement values and position can be estimated using geometric calculations.	Additional algorithms are needed for better accuracy.

Tabella 4.2: Vantaggi e svantaggi di diverse tecniche di localizzazione [31]

- **UWB (Ultra Wide Band)**
  É una tecnologia radio a corto raggio che trasmette brevi impulsi in un'estesa larghezza di banda, offrendo una grande precisione. Sistemi IPS basati su UWB permettono un'accuratezza addirittura di decine di centrimetri, e quindi sono considerate migliori rispetto a Wi-Fi o BLE. Non si tratta però di una tecnologia così diffusa da poter essere utilizzata universalmente. Vengono inoltre impattate da situazioni non ideali come l'NLOS.

- **Beacon BLE (Bluetooth Low Energy)**
  Sono dei piccoli trasmettitori radio Bluetooth, alimentati a batteria, che inviano in continuazione degli impulsi che possono essere captati, per determinare a quale distanza si trovano. A seconda della loro qualità, possono fornire una copertura che va dai 20-30 metri ai 70-100 metri, ma soffrono intrinsecamente di fenomeni di NLOS, in quanto viene utilizzato l'RSSI per la stima della distanza tra il ricevitore ed essi. Accompagnando l'utilizzo di questa tecnologia con altre tecniche di manipolazione numerica dei dati, è possibile creare dei sistemi di posizionamento performanti (ad esempio basati su fingerprinting).

Nel progetto VASARI si è deciso di allestire i siti culturali con dei Beacon BLE, di cui ne sono stati forniti 3 durante la fase sperimentale di questa tesi. Ho allestito un ambiente domestico di test, e ho condotto degli esperimenti atti a trovare, per il sito di riferimento, la tecnica migliore di posizionamento. Come detto in precedenza, nella maggior parte dei casi un sistema IPS è strettamente dipendente dal tipo di tecnologia che si decide di usare, e dal numero di nodi ancora. É perciò possibile che, in un futuro sviluppo commerciale di VASARI, si decida di adottare un sistema di posizionamento indoor diverso per ogni sito culturale, spostando ovviamente la logica del tutto su uno specifico server (dando all'app l'unico compito di raccogliere i dati trasmessi dai nodi, e tramite API specifiche ottenere la nuova posizione globale o relativa). La tecnologia utilizzata da ogni sito deve inoltre tenere conto delle esigenze dell'utilizzatore, perchè alcune tecnologie potrebbero non essere supportate da fasce più o meno grandi di device.

In Figura 4.27 è raffigurata la mappa (con una struttura GeoJSON volutamente semplificata) dell'ambiente domestico di test.

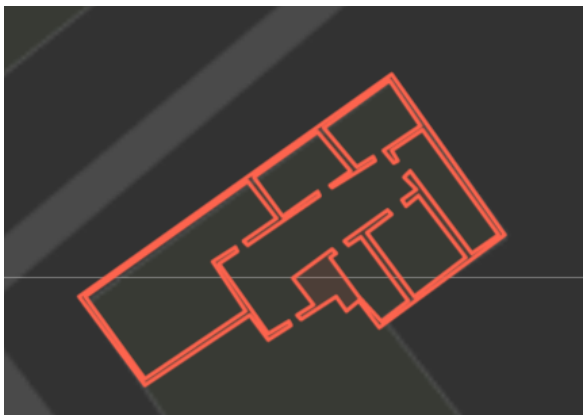

Figura 4.27: Ambiente domestico di test

Beacon BLE

I Beacon BLE che sono stati forniti (mostrati in Figura 4.28) appartengono ad una categoria di dispositivi abbastanza economici che, se da un lato permettono di mantenere bassi i costi (e quindi, a parità, poter essere presenti in numero più elevato), dall'altro soffrono di quella serie di problemi che sono stati discussi in precedenza (fenomeni di NLOS).
Dovendo utilizzare questi trasmettitori come dei nodi ancora (con posizione nota a priori),

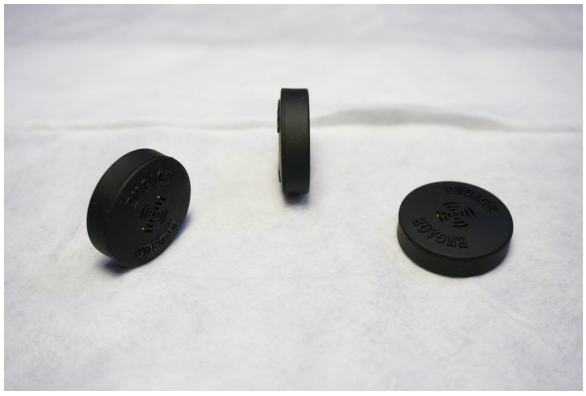

Figura 4.28: Beacon BLE usati durante la fase sperimentale

il primo problema che si è presentato è stato il come disporre tali dispositivi all'interno dell'ambiente di test. La disposizione fisica dei nodi ancora è strettamente dipendente da ogni ambiente indoor, e può influire sensibilmente sulle prestazioni di un IPS, come è analizzato in [36]. In particolare viene affermato come, per trovare un buon posizionamento dei Beacon, occorre studiare l'ambiente circostante per capire se i segnali siano ben separati tra di loro e se sia possibile coprire l'intera area (ossia, non vi siano aree non coperte da almeno uno di essi). In Figura 4.29, tratta da [36], è mostrato un esempio di disposizione dei Beacon in un ambiente di test. L'ambiente domestico di test utilizzato per lo sviluppo del proof of concept di Smart Visit ha un'area di circa 140 metri quadri, ed è suddiviso in varie stanze. Sapendo che eventuali ostacoli mobili sono abbastanza difficili da predire, si è cercato di ottimizzare la disposizione dei Beacon cercando di ridurre il più possibile il rumore dato dagli ostacoli fissi, ossia pareti, porte e arredamenti interni. La necessità di poter distinguere chiaramente i vari

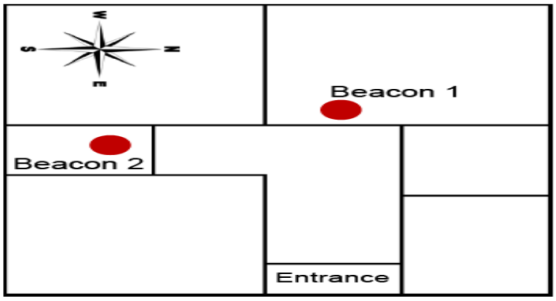

Figura 4.29: Disposizione dei beacon in un ambiente di test [36]

segnali (coprendo contemporaneamente tutta la superficie con la ricezione chiara di almeno un segnale), ha portato all'esecuzione di vari test, alla fine dei quali si è appurato come la disposizione in zone distanti circa 12-15 metri fosse ottimale per il sito demo. In Figura 4.30 è mostrata la disposizione finale dei Beacon all'interno dell'ambiente domestico di test.

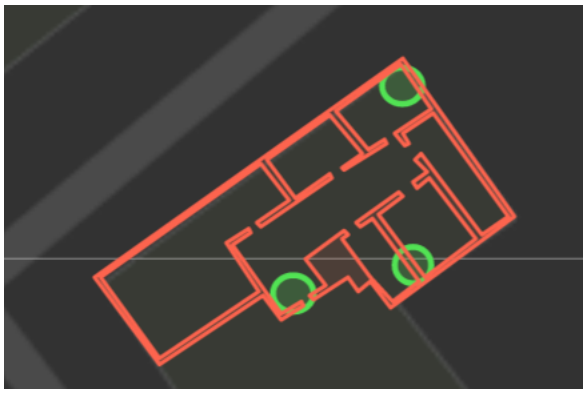

Figura 4.30: Posizione dei beacon all'interno dell'ambiente domestico di test

RSSI

Espresso in dB, rappresenta la relazione tra la potenza trasmessa e quella ricevuta. In riferimento a [13], se è presente un percorso diretto senza ostacoli (situazione di LOS), si ha una relazione di inversa proporzionalità quadratica tra la potenza del segnale ricevuta e la distanza tra trasmettitore e ricevitore:

$$P_r \propto d^{-2} \tag{4.1}$$

In una situazione reale però, saranno presenti ostacoli che comportano fenomeni di rifrazione, riflessione, attenuazione del segnale, ossia tutti quelli che possono portare alla situazione di NLOS. Ad esempio, sempre in riferimento a [13], si è trovato empiricamente che un muro può ridurre mediamente la potenza del segnale di circa 3dB. A causa quindi di una propagazione del segnale non più uniforme, la potenza ricevuta può degradarsi ad una velocità maggiore, per cui la (4.1) diventa:

$$P_r \propto d^{-n} \tag{4.2}$$

Attivando il Bluetooth del dispositivo utente, e utilizzando il plugin Cordova *IBeacon*, è stato possibile intercettare l'identificativo (dato dalla tripletta *UUID*, *major* e *minor*) e il segnale RSSI di ogni singolo trasmettitore. Il plugin permette, oltre che rilevare la presenza

di Beacon BLE in una determinata zona, anche di monitorare l'evoluzione del suo segnale nel tempo. Un esempio di utilizzo del plugin è il seguente, in cui si vuole semplicemente stampare nella console di debug ogni variazione dei segnali RSSI nel range di appartenenza (ossia, l'identificativo primario della famiglia di quei Beacons):

```
import {IBeacon} from '@ionic-native/ibeacon/ngx';
.
.
.
constructor(private ibeacon: IBeacon) {}
.
.
.
const id = '...';
const beaconsRange = '...';
this.ibeacon.requestWhenInUseAuthorization();
const delegate = this.ibeacon.Delegate();
delegate.didRangeBeaconsInRegion().subscribe(data => {
    console.log(data)
});
const beaconsRegion = this.ibeacon.BeaconRegion(id, beaconsRange);
this.ibeacon.startRangingBeaconsInRegion(beaconsRegion)
```

Il segnale non è trasmesso continuamente, ma con una cadenza che può andare da 100 millisecondi (segnale stabile) a 2 secondi (segnale instabile). Per calcolare la distanza in metri a partire dal segnale, si usa la seguente formula, tratta da [24]:

$$d = 10^{\left(\frac{MP-RSSI}{10 \cdot n}\right)} \tag{4.3}$$

Dove *MP* è la potenza misurata alla distanza di un metro (fattore di calibrazione), *RSSI* è la potenza del segnale ricevuto in un determinato istante di tempo, ed n è una costante il cui valore dipende da fattori ambientali e tipicamente assume valori tra 2 e 4. La potenza misurata, alla distanza di un metro, per i Beacon forniti, è di -69dB.
In relazione a quanto detto in [13], è utile applicare dei metodi matematici al flusso dei valori ricevuti di RSSI per migliorarne la stabilità (soprattutto in un contesto in cui il nodo target è in movimento, come per Smart Visit). Infatti, l'utilizzo dei valori grezzi non è affidabile perchè eventuali fenomeni di NLOS o rumore possono falsare la cognizione della distanza che si ha tra un nodo ancora e il nodo target, influenzando negativamente le tecniche di posizionamento. I principali metodi sono elencati in [13]:

- **Raw Data**
  É il metodo più semplice, nonchè quello più impreciso a causa delle fluttuazioni che i valori grezzi possono avere durante il movimento. Inoltre, un dato valore di RSSI può corrispondere a distanze diverse: ad esempio, secondo [13], il valore di -90dB può corrispondere a una distanza di 7 metri o di 26 metri. Perciò, l'utilizzo dei dati grezzi è una metodologia debole per determinare la distanza di un nodo ancora.

- **Moving Average**
  Per fare il modo di ridurre la fluttuazione del segnale, è possibile applicare una media mobile: invece di utilizzare il singolo valore di RSSI, viene utilizzato il valore medio dei precedenti n valori (insieme al nuovo valore). La sfida più grande è decidere il valore di n (la finestra temporale): un valore alto permette un'accuratezza maggiore (ma anche un'attesa maggiore, in quanto bisogna aspettare tutti i valori), mentre un valore basso permette di ottenere un risultato più rapido (al costo di un'accuratezza minore).

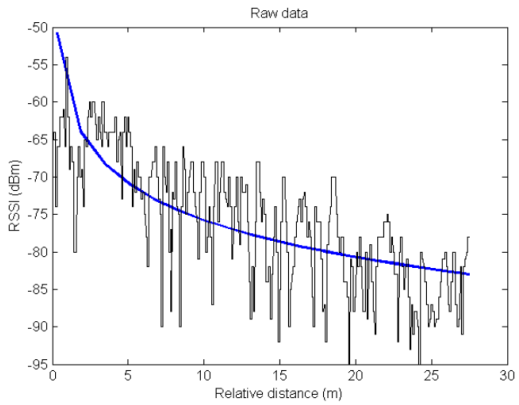

Figura 4.31: Utilizzo dei dati grezzi per la localizzazione [13]

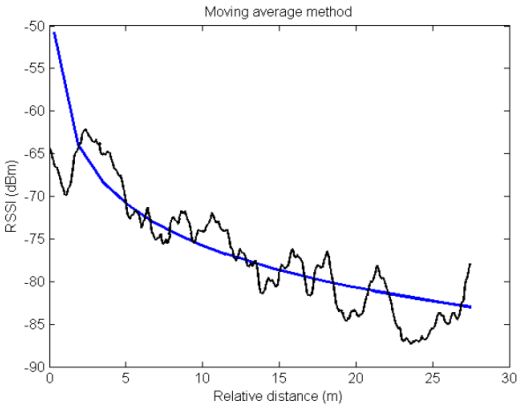

Figura 4.32: Utilizzo della media mobile per la localizzazione [13]

- **Weighted Average**
  In termini teorici, il cambiamento dei valori di RSSI dovrebbe essere graduale ma costante: questa tecnica, basandosi su questo principio, consiste nel dare un peso diverso ai samples raccolti in precedenza (mentre nella tecnica precedente viene assegnato lo stesso peso a tutti). La tecnica assegna un peso maggiore ai campioni che sono più vicini al nuovo valore di RSSI raccolto. In pratica però, a causa della fluttuazione e di fenomeni di NLOS, il risultato risulta essere molto simile alla semplice media mobile.

- **Curve Fitting**
  Questo metodo utilizza tutti i campioni raccolti per predire il valore di RSSI nel prossimo istante di tempo: più dati sono stati raccolti, più precisa sarà la predizione. Perciò, risulta essere una tecnica performante a lungo termine, mentre all'inizio ha un'accuratezza simile all'utilizzo dei dati grezzi.

Nelle Figure 4.31, 4.32, 4.33 e 4.34, tratte da [13], sono mostrate le varie tecniche di filtraggio dei valori di RSSI.
Nell'implementazione del proof of concept, si è deciso di utilizzare la tecnica della media mobile. Sono state eseguite delle analisi per vedere l'andamento nel tempo dei valori di RSSI dei Beacon che sono stati forniti durante la fase sperimentale. Essi (identificati con 517, 518 e 519, valori dei rispettivi *minor*) sono stati posizionati a diverse distanze (in una situazione di LOS) dal nodo target (fisso), per poi raccogliere i valori di RSSI ricevuti per circa 30

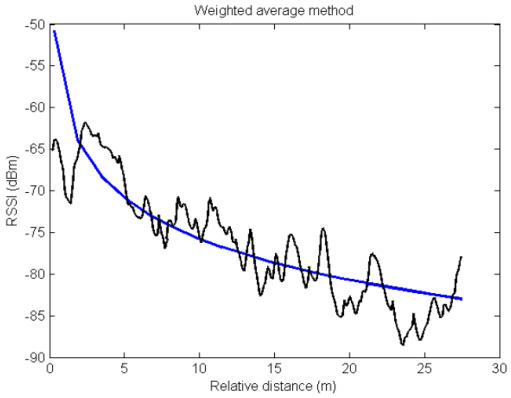

Figura 4.33: Utilizzo della media pesata per la localizzazione [13]

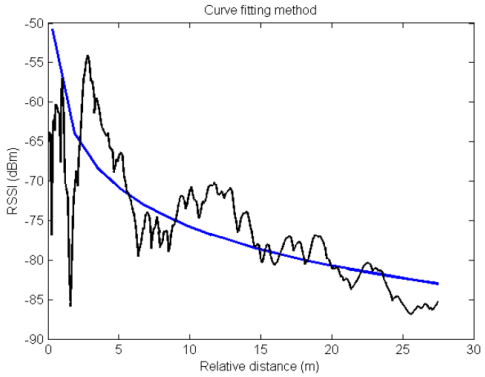

Figura 4.34: Utilizzo della curva di apprendimento per la localizzazione [13]

minuti. Nelle Figure 4.35, 4.36 e 4.37 sono mostrati gli andamenti nel tempo della distanza percepita dei tre Beacon: per ognuno di essi, è mostrato il valore reale, il valore medio finale, l'andamento della distanza calcolata tramite i dati grezzi e tramite media mobile. Nonostante l'intervallo di acquisizione del valori sia stato uguale per tutti e tre i Beacon, la quantità di valori effettivamente ricevuti dal nodo target differiscono l'uno dall'altro: per il Beacon 517 sono stati raccolti 344 valori, per il Beacon 518 ne sono stati acquisiti 141, mentre per il Beacon 519 soltanto 51. Perciò, anche in condizioni di staticità e di LOS, la quantità di valori che possono essere raccolti varia con la distanza a cui sono posti i Beacon, e inoltre la fluttuazione del segnale permane. Dall'analisi condotta si nota come il segnale del Beacon posto a circa 30 metri dal nodo target sia praticamente inutilizzabile (oltre a raccogliere nel complesso molti meno samples), mentre i segnali dei Beacon posti a 6 e 16 metri risultano avere un andamento più stabile, soprattutto a lungo termine. Perciò, in conclusione, la distanza massima a cui posizionare un Beacon dal nodo target è di massimo 20 metri: distanze superiori non rendono il segnale affidabile per il calcolo della distanza dal nodo target. La disposizione dei tre Beacon all'interno dell'ambiente domestico di test ha seguito anche queste considerazioni. Tuttavia, le analisi condotte si sono riferite esclusivamente ad un contesto statico (nodo target non in movimento) e in situazione di LOS (visibilità diretta con i Beacon e nessun ostacolo fisso o dinamico). Questo ha portato all'idea per cui l'utilizzo esplicito della misura della distanza possa non essere ottimale per il contesto applicativo, quanto piuttosto il valore di RSSI. Inoltre, eventuali valori inutilizzabili per questo tipo di analisi, potrebbero essere utili in tecniche basate sul Machine Learning, in quanto un modello potrebbe essere

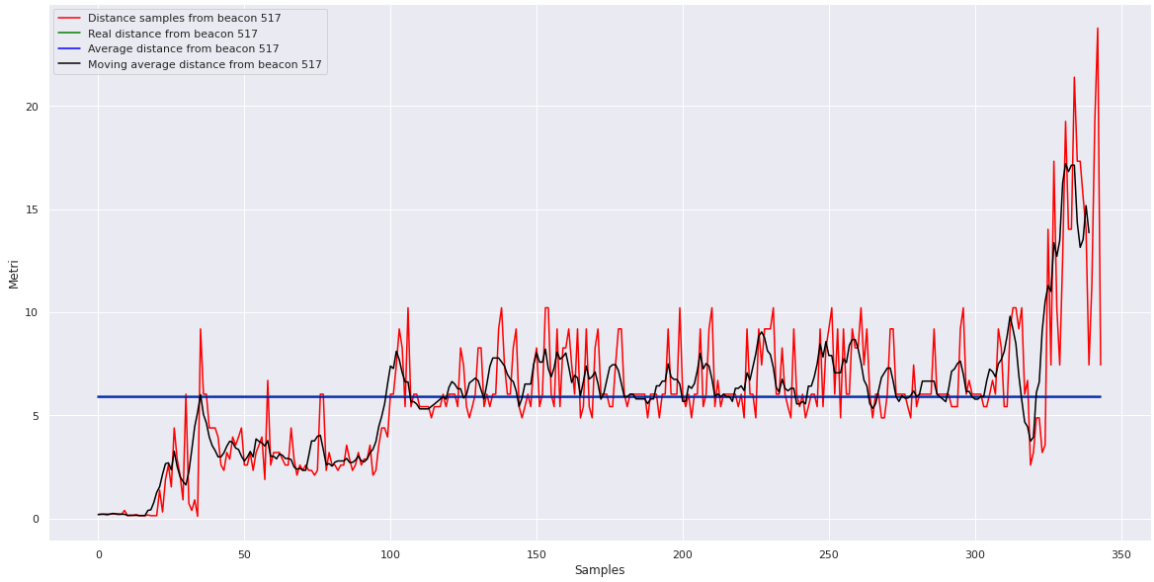

Figura 4.35: Analisi del Beacon 517, posto a circa 6 metri dal nodo target

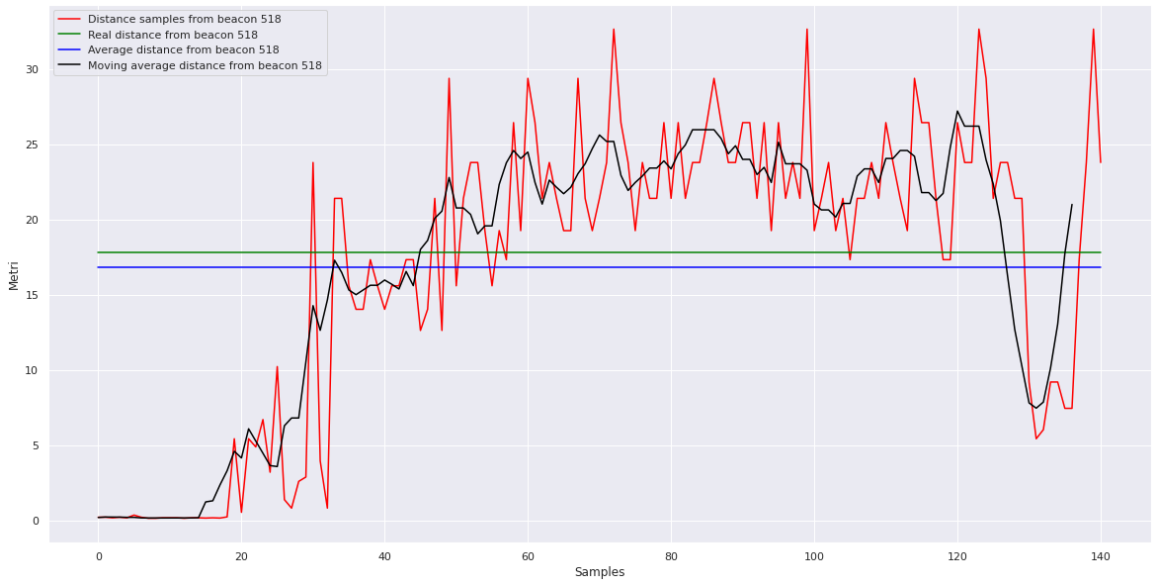

Figura 4.36: Analisi del Beacon 518, posto a circa 16 metri dal nodo target

addestrato su essi e estrapolare più informazioni in grado di posizionare meglio l'utente. Ad esempio, se in un determinato punto arriva solamente un segnale in un range di valori affidabili, mentre gli altri sono affetti da rumore (e quindi falsati), il modello potrebbe acquisire questa informazione e renderla utile ai fini del processo di addestramento.

Machine Learning per Indoor Positioning Systems (IPS)

Secondo [31], l'utilizzo di algoritmi di Machine Learning può risolvere alcuni problemi e limiti delle tecniche tradizionali di localizzazione all'interno di ambienti indoor. Esse, infatti, non sono molto scalabili a causa di una scarsa flessibilità a cambiamenti dinamici dell'ambiente. Il Machine Learning applicato a IPS può essere utilizzato per vari task, elencati in [31]:

Figura 4.37: Analisi del Beacon 519, posto a circa 30 metri dal nodo target

- **NLOS Error Minimization**
  Poichè nell'utilizzare determinate tecniche e strumenti (come la trilaterazione tramite Beacon BLE) si incorre in grossi errori di ranging dovuti a effetti di NLOS e propagazione errata del segnale, un'applicazione del Machine Learning può essere la mitigazione di questi effetti. L'obiettivo è cercare di distinguere il più possibile i segnali NLOS dai segnali LOS, per fare in modo di utilizzare solo gli ultimi nell'applicazione delle tecniche tradizionali (o altre tecniche di Machine Learning).

- **Fingerprinting**
  Un altro tipo di applicazione consiste nella predizione della nuova posizione, a partire da quelle precedenti. Il tutto si basa ovviamente sui dati, per cui occorre, in una fase offline, ottenere e catalogare il più possibile degli esempi con cui allenare dei modelli di intelligenza artificiale. Viene perciò generata in un primo momento una radio map del sito (samples contenenti i valori dei segnali e il punto geospaziale a cui si riferiscono). Questi modelli, a fronte di nuovi samples appartenenti alla stessa distribuzione, potranno predire la nuova posizione dell'utente.

- **Trajectory Learning**
  Si utilizza questa tecnica quando non è disponibile a priori una radio map del sito (come in precedenza). Si utilizza il contesto spaziale, che consiste nella conformazione dell'ambiente e dei suoi punti di riferimento (usando, ad esempio, il crowdsourcing di quel punto), per calibrare l'errore di localizzazione senza utilizzare hardware aggiuntivo.

Per quanto riguarda la creazione del dataset, per il proof of concept sono stati resi disponibili tre Beacon BLE, da ognuno dei quali si può ottenere un singolo RSSI. Questi attributi, che sono di carattere ambientale (uguali per tutti i device), potrebbero essere accompagnati da altri ottenuti dai singoli dispositivi (ossia, dai loro sensori). In [29], si valuta un approccio ibrido in cui il dataset finale consiste in 172 attributi (dati da diversi RSSI da vari access points WIFI, e sensori come accelerometro, magnetometro, rotazione e orientamento), per un totale di 3110 samples. Tuttavia, poichè l'entità del valore di un singolo sensore potrebbe variare in diversi device (o, addirittura, non essere del tutto presente), questo complicherebbe non poco la fase di data entry (in quanto bisognerebbe considerare vari dispositivi), oltre che la complessità globale del modello. Inoltre, l'utilizzo di molti sensori potrebbe compromettere

le prestazioni dell'applicazione.
Per la fase di data entry si è agito come segue. Essendo la superficie del sito demo di circa 120 metri quadri, sono state selezionate 9 sezioni principali (che nel complesso coprono interamente il sito), in ognuna delle quali sono stati raccolti dei samples (per un totale di 268 campioni). Ognuno di essi è stato etichettato utilizzando come target il punto fisico in cui si trovava il device al momento dell'acquisizione (al centro di quella sezione). In Figura 4.38 sono mostrati 10 samples di esempio del dataset raccolto, dove i valori di "Target" rappresentano l'identificativo della sezione del sito a cui fanno riferimento.

	Beacon517	Beacon518	Beacon519	Target
147	-87	-93	-88	3
2	-74	-98	-102	0
167	-102	-91	-70	4
78	-85	-89	-87	1
181	-102	-90	-72	4
190	-101	-92	-70	5
97	-98	-84	-92	2
216	-98	-75	-86	6
12	-72	-90	-100	0
214	-98	-74	-82	6

Figura 4.38: Frammento di dataset

Fingerprinting

Data la ridotta grandezza del sito, per il proof of concept si è deciso di procedere con un approccio di classificazione, piuttosto che di regressione. La differenza tra i due approcci sta nel tipo di predizione, che nel primo caso è discreta (una tra i target possibili, ossia una tra le 9 posizioni fisiche), mentre nel secondo è continua (per cui la nuova posizione potrebbe essere del tutto nuova).
Come primo tentativo si è cercato di allenare un algoritmo di Machine Learning chiamato K-Nearest Neighbor (KNN), il quale utilizza i K samples più vicini (secondo una metrica di distanza prestabilita) per predire il prossimo sample (basandosi sulla maggioranza dei voti). In letteratura è molto utilizzato come termine di confronto con altri algoritmi per vari tasks, tra cui anche il posizionamento indoor (in relazione a [31] e [29]), ma risente intrinsecamente di alcuni problemi. Per esempio, soffre della cosiddetta "Curse of dimensionality": all'aumentare della dimensionalità (numero di attributi) del dataset, la performance diminuisce (per ovviare a ciò, si può intervenire applicando tecniche di riduzione della dimensionalità come la Principal Component Analysis o la Linear Discriminant Analysis). Inoltre per grandi dataset risulta essere abbastanza lento: la predizione di un nuovo sample prevede il calcolo della distanza da ogni altro sample.
Utilizzando il dataset raccolto (standardizzato), si è allenato un modello basato su KNN (validato con un 10-Fold Cross Validation) utilizzando vari valori di K, ossia di campioni su cui basarsi per predire la nuova posizione. A tal scopo, si è utilizzato il linguaggio di programmazione Python, molto utilizzando nell'ambito della Data Science e del Machine Learning. I risultati sono riassunti in Tabella 4.3.

K	Accuratezza
1	81.07%
3	74.35%
5	72.47%
7	58.06%
9	56.68%
11	53.20%
13	53.18%
15	53.64%
17	53.66%
19	52.35%

Tabella 4.3: Accuratezza del K-Nearest Neighbor per vari valori di K

Si noti come la performance del modello crolli all'aumentare del parametro K (il numero di samples vicini da considerare). In seguito si è deciso di procedere con un altro approccio. Una particolare sotto-categoria del Machine Learning molto in voga in ricerca e con applicazione in vari ambiti è il Deep Learning (apprendimento profondo). Esso utilizza come modelli le Reti Neurali, che si ispirano a quelle biologiche come struttura e funzionamento. In particolare, il Deep Learning prevede l'utilizzo di reti profonde, con una quantità di parametri (connessioni tra neuroni, chiamate "pesi") molto superiore ad altri algoritmi di intelligenza artificiale. La combinazione lineare dei pesi, insieme alle varie funzioni di attivazione, permette di predire il risultato a partire dall'input (dove ogni attributo è modellato come un neurone al primo livello). Ciò che rende il Deep Learning uno strumento potente è il fatto che, aumentando la complessità del modello, è possibile far imparare determinati pattern nascosti dei dati (che con altri algoritmi sarebbero difficili da estrapolare).
In [30], si discute l'utilizzo di una ANN (Artificial Neural Network) per questo tipo di applicazioni. In particolare, viene utilizzata una rete con 4 neuroni di input, 2 neuroni di output e 4 layer nascosti: basandosi su questo esempio, si è cercato di stabilire una possibile architettura di ANN per il sito indoor di test. Per capire quanti layer nascosti (che definiscano la profondità di questo modello di Deep Learning) e quanti neuroni assegnare ad ognuno, si è fatto riferimento all'articolo online [20]. Per quanto riguarda il numero di livelli (oltre a quello di input e quello di output), un numero alto è consigliato per datasets abbastanza complessi (per problemi di Computer Vision o che coinvolgono un ordine temporale), mentre per datasets di scarsa complessità (come quello di riferimento), è possibile utilizzare un numero basso (2-3). É stato deciso di utilizzare 3 layer nascosti. La scelta del numero di neuroni per ciascun layer è più delicata: essi hanno un grande impatto verso l'output finale. Un numero troppo basso può portare a situazioni di underfitting, ossia delle performance molto basse sul training set. Di contro, un numero troppo alto potrà causare overfitting, ossia una situazione in cui il modello performa sin troppo bene sul training set, ma male sul test set (è come se il modello imparasse a memoria, senza estrapolare pattern utili alla predizione). Si è deciso di utilizzare una delle regole consigliate, ossia un numero di neuroni nascosti in ogni layer intermedio pari a 2/3 la grandezza del layer di input, più la grandezza del layer di output (nel caso del dataset di riferimento, 11 neuroni per ogni layer intermedio). I vari livelli sono fully-connected, perciò il numero totale di parametri del modello sarà di 416. Altre caratteristiche della rete progettata sono l'utilizzo della funzione di attivazione ELU alla fine di ogni layer e un regolarizzatore di tipo L2, un ottimizzatore Adam (rispetto alla classica Discesa Stocastica del Gradiente), un'inizializzazione di Xavier dei pesi della rete e 1000 epoche di training (iterazioni lungo il training set, il 75% del dataset originario standardizzato). Il livello finale, infine, è terminato con una funzione di attivazione Softmax, la quale specifica la probabilità di appartenenza ad ogni singolo target. Prendendo la classe con probabilità massima, si otterrà infine la predizione finale del modello. In Figura 4.40 è mostrata una rappresentazione ad alto livello del

modello progettato (generata con il tool online gratuito [32]), mentre in Figura 4.39 sono mostrati i dettagli dei singoli livelli.
L'accuratezza della rete proposta si attesta intorno al 94-96% (in varie esecuzioni del codi-

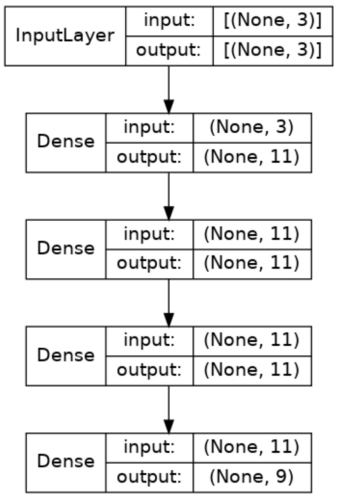

Figura 4.39: Layers della Rete Neurale nel dettaglio

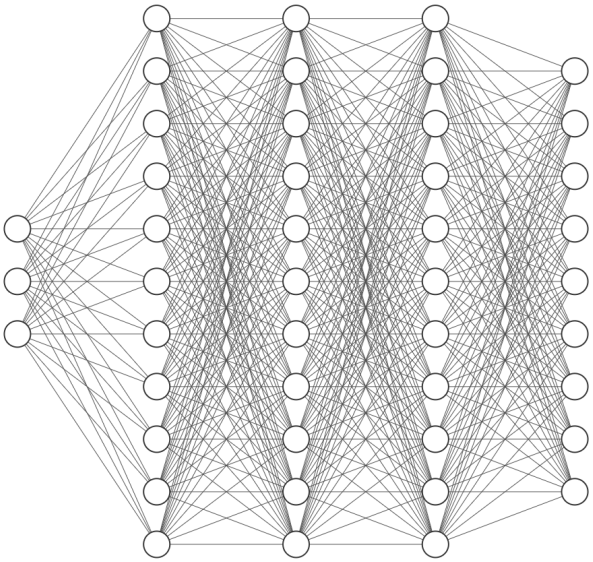

Figura 4.40: Rete Neurale utilizzata per l'ambiente indoor di test, generata con il tool online gratuito [32]

ce). Confrontando con il modello basato su KNN, questo rappresenta un grosso miglioramento in termini di accuratezza del posizionamento dell'utente. Nel seguente frammento di codice viene riportata la definizione di una classe che permette di caricare il modello (data una risorsa, locale o remota) e eseguire delle predizioni (dato un array numerico che viene convertito in un tensore, l'oggetto principale della libreria *TensorFlow*). Nel caso specifico, la predizione consiste in un numero (l'identificativo del target che massimizza la probabilità dell'output dato l'input) che può essere utilizzato come indice in un array di punti (coppie latitudine/longitudine) per ottenere quelle specifiche coordinate GPS.

```
import * as tf from '@tensorflow/tfjs';

export class ArtificialIntelligenceModel {

    private _model;

    constructor() {}

    /**
     * Load the model from an URI
     * @param uri URI
     */
    public async load(uri: string) {
        this._model = await tf.loadLayersModel(uri);
    }

    /**
     * Make a prediction with a given input
     * @param input Input
     */
    public predict(input: number[]): number {
        const tensor = tf.tensor([input]);
        const predictions = this._model.predict(tensor);
        const result = tf.argMax(predictions, 1).dataSync();
        return Array.from(result)[0];
    }

}
```

4.5 Navigazione

Dopo aver discusso come posizionare l'utente all'interno del sito culturale (dando un feedback visivo sulla mappa), parliamo ora del sistema di navigazione implementato che permette di dare un'indicazione sulla strada da percorrere per arrivare al prossimo punto di interesse. Questo aspetto si sviluppa solamente nel contesto di un percorso ordinato, e non libero (perchè, sostanzialmente, il prossimo punto di interesse sarebbe quello più vicino). Lo stesso principio è stato adottato anche per mostrare il percorso verso un sistema di movimento tra piani (scale o ascensori), nel momento in cui l'utente ne abilita la visione (tramite opzione "Mostra scale e ascensori") e clicca su uno dei riquadri corrispondenti.
Il meccanismo implementato consiste nella generazione, tramite la libreria *Turf*, di uno "shortest path" che va dalla posizione corrente dell'utente fino alla posizione del prossimo punto di interesse (nota a priori). La libreria offre una funzione, tra le varie presenti, che calcola lo shortest path (formattato in GeoJSON) tenendo conto di ostacoli e di una determinata risoluzione (ossia, la precisione con cui viene calcolato). in Figura 4.41 sono mostrati degli esempi di shortest path tra la posizione corrente dell'utente e il prossimo punto di interesse (indicato in verde), tenendo conto di ostacoli (ossia, il perimetro del sito culturale). Scendiamo ora nel dettaglio di come questo meccanismo è stato implementato in relazione al contesto applicativo di Smart Visit.

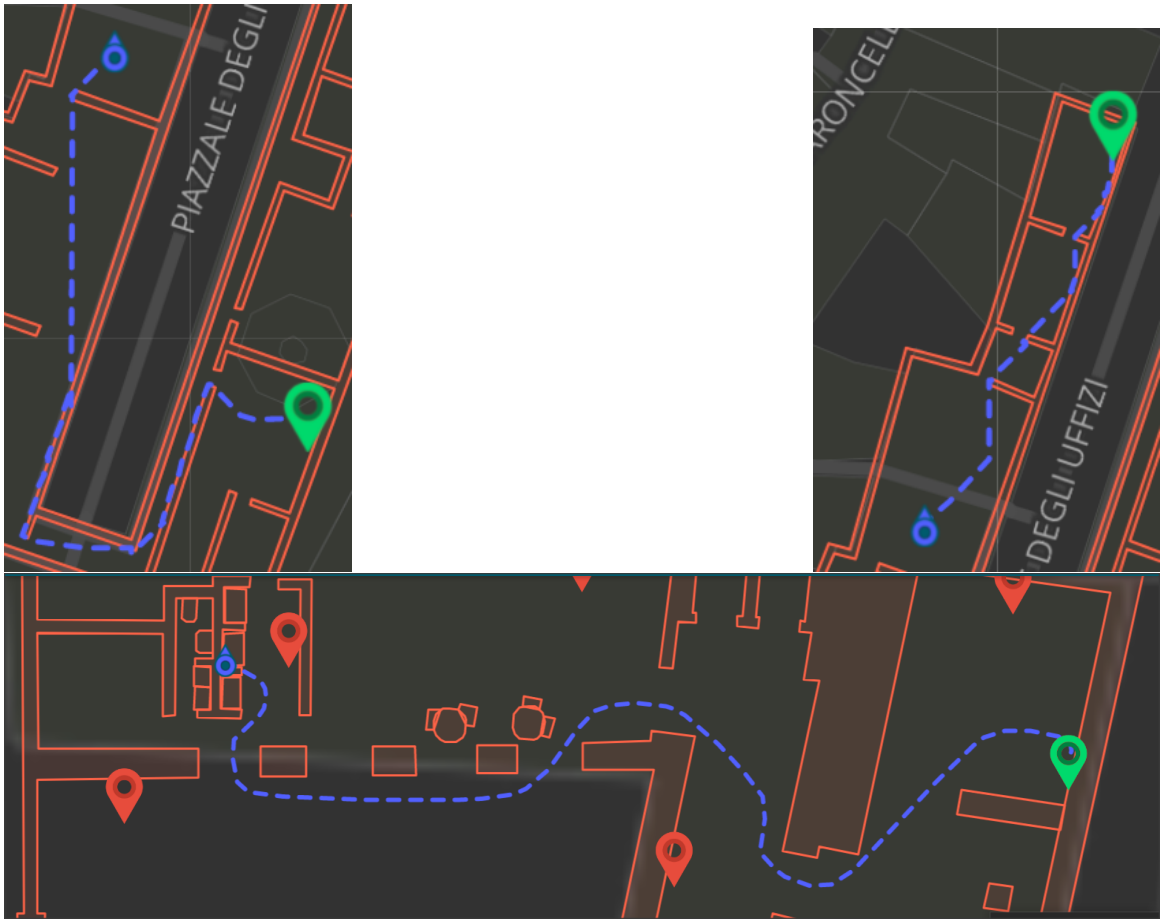

Figura 4.41: Esempi di shortest path tracciati tramite le libreria *Turf*

4.5.1 Algoritmo A*

In riferimento a [4], discutiamo dei principali algoritmi utilizzati per la risoluzione del problema del "path finding". Esso consiste nel trovare un percorso, inteso come sequenza di nodi all'interno di un grafo, da un nodo sorgente fino ad un nodo destinazione.
Uno degli algoritmi più utilizzati per la risoluzione di questo problema è l'algoritmo di Dijkstra, il quale cerca ogni possibile percorso (a partire dal nodo sorgente) per trovare il più breve. Se da un lato questo algoritmo riesce a trovare la soluzione migliore al problema, dall'altro può richiedere molto tempo per farlo e, perciò, essere poco performante. Nel contesto applicativo di Smart Visit questo può essere problematico in quanto la visualizzazione del percorso deve essere rapida: nel momento in cui arriva la soluzione, l'utente potrebbe essere già stato riposizionato, rendendo inutile il calcolo precedente.
L'algoritmo A*, implementato dalla libreria *Turf*, è un algoritmo best-first che nasce come generalizzazione dell'algoritmo di Dijkstra, ma risulta essere più efficiente (in termini di tempo) in quanto utilizza delle informazioni extra per guidare la ricerca: infatti, non considera solamente il numero di steps dal nodo sorgente, ma anche una funzione euristica che da un'indicazione della direzione da preferire durante la ricerca. La funzione di stima dell'algoritmo A* è espressa in 4.4.

$$f(n) = g(n) + h(n) \tag{4.4}$$

Dove $g(n)$ rappresenta il costo del percorso più breve dal nodo sorgente fino al nodo n, lungo il percorso trovato fino a quel momento. La funzione euristica $h(n)$ è invece una stima del costo del percorso più breve dal nodo n fino al nodo destinazione. L'algoritmo di Dijkstra è un caso particolare dell'algoritmo A*, in cui $h(n) = 0$ per ogni nodo n.

L'algoritmo A* mantiene due set di nodi: *CLOSED* e *OPEN*. L'algoritmo rimuove iterativamente il nodo n per cui *f(n)* è minimo dal set *OPEN* e lo inserisce nel set *CLOSED*. Se n è il nodo destinazione, la soluzione è stata trovata, altrimenti viene generato un set di nodi contenente tutti i nodi vicini al nodo n: per ognuno di essi (n') è calcolata la funzione di stima *f(n')*. Infine, ogni nodo n' che non è già in *OPEN* o *CLOSED* viene inserito in *OPEN*. Il seguente frammento, tratto da [4], rappresenta lo pseudocodice dell'algoritmo A*:

```
1  (1)      Put the start node s into OPEN.
2  (2)      IF OPEN is empty THEN exit with failure.
3  (3)      Remove from OPEN and place in CLOSED a node n
4           for with f is minimum.
5           (Resolve ties for minimal f value,
6           but always in favor of any goal node)
7  (4)      IF n is a goal node THEN exit successfully with
8           the solution obtained by tracing back
9           the pointers from n to s.
10 (5)      ELSE expand n, generating all its successors,
11          and attach to them pointers back to n.
12          FOR every successor n' of n DO
13              (5.1)   IF n' is not already in OPEN or CLOSED
14                      THEN estimate h(n'),
15                      and calculate f(n') = g(n') + h(n'),
16                      where g(n') = g(n) + c(n, n') and g(s) = 0,
17                      and put n' into OPEN.
18              (5.2)   IF n' is already in OPEN or CLOSED
19                      THEN direct its pointer along the path
20                      yielding the lowest g(n').
21              (5.3)   IF n' required pointer adjustment
22                      and was in CLOSED
23                      THEN reopen it.
24          END FOR
25 (6)      GO TO step 2.
```

La Figura 4.42, tratta da [4], mostra un esempio del processo di ricerca dell'algoritmo A* in una griglia rettangolare contenente 3 ostacoli (in nero). I nodi neri sono inseriti nel set *CLOSED*, mentre i nodi bianchi nel set *OPEN*. Il numero all'interno di un singolo nodo rappresenta il valore di g, misurato tramite distanza di Manhattan dal nodo sorgente (etichettato come 0). Il costo di un percorso ottimale dal nodo sorgente al nodo destinazione coincide con l'etichetta data al nodo destinazione, ossia 33.

4.5.2 Definizione della griglia e degli ostacoli

Nell'applicazione di questo algoritmo, come anche in quello di Dijkstra, bisogna trasformare lo spazio continuo (la mappa, intesa come insieme di coppie latitudine-longitudine) in un grafo discreto (insieme di nodi e archi con costo unitario). Quando viene richiamata la funzione di *Turf* che calcola lo shortest path, è possibile inserire delle opzioni: tra di queste ve ne sono due, chiamate *units* e *resolution* che permettono di stabilire il grado di precisione (inteso come granularità dello spazio) e l'unità di misura (ad esempio, metri o chilometri). Più il livello di risoluzione è alto, più il percorso sarà preciso e terrà conto di eventuali ostacoli: tuttavia, ciò si ripercuoterà ovviamente sulle prestazioni. É stato trovato un trade off durante la fase sperimentale di circa 1/3 di metro, per cui lo spazio fisico viene modellato con una matrice di nodi, distanti 0.33 metri l'uno dall'altro. Tuttavia, il valore di questo parametro (e la sua unità di misura) potrebbe essere strettamente dipendente dal tipo di sito culturale (e dalle sue caratteristiche, come la superficie o la presenza più o meno alta di ostacoli fissi): si potrebbe inserire, a tal proposito, un campo all'interno del JSON che modella il sito culturale (formato

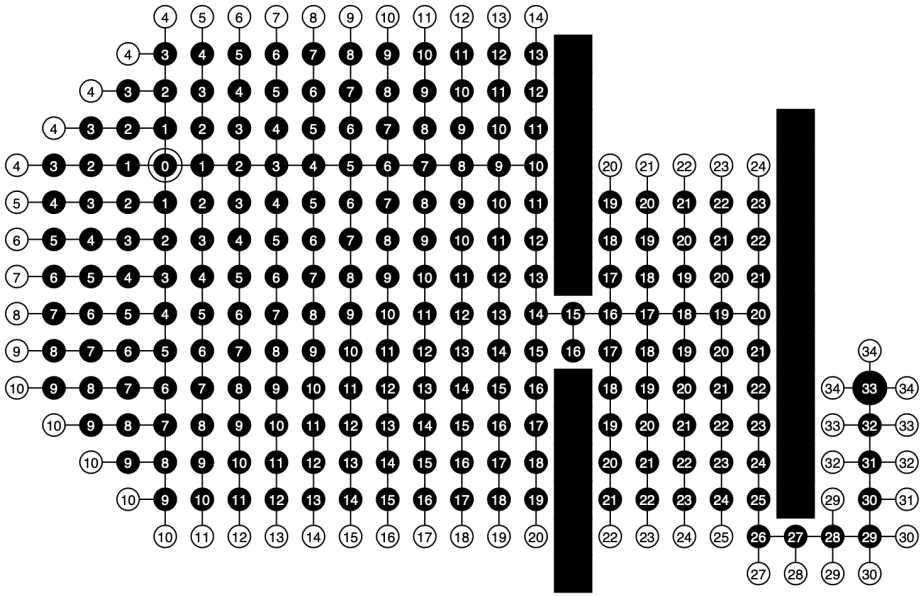

Figura 4.42: Griglia 2-D con 3 ostacoli [4]

da più GeoJSON) che indichi i valori di *units* e *resolution*.
Per quanto riguarda invece gli ostacoli da considerare per tracciare il percorso, essi possono ridursi sostanzialmente alle pareti del sito culturale. La rappresentazione dello stesso potrebbe essere più complessa, ossia contenere ostacoli interni che non toccano le pareti (altrimenti basterebbe inglobare tali elementi nelle pareti) o aperture secondarie nella geometria (magari per rappresentare finestre) che possano confondere l'algoritmo e tracciare un percorso attraverso di esse. La fase di sviluppo di questi file è perciò abbastanza delicata, e va oltre le finalità di questo lavoro di tesi (in cui i GeoJSON dei siti culturali sono stati mantenuti semplici).

4.5.3 Indicazioni tramite frecce

Come mostrato in precedenza, in Figura 4.23, è stato implementato un meccanismo di navigazione tramice frecce tramite il quale l'utente, a seconda del proprio orientamento, riceve un feedback visivo sulla direzione verso cui procedere per seguire lo shortest path verso il prossimo punto di interesse. Quando viene definito, in un determinato istante, il percorso come oggetto GeoJSON, viene preso come riferimento il primo tratto di spezzata (essendo, fondamentalmente, una linea spezzata composta da tante linee rette). Utilizzando *Turf* si è calcolato il bearing (l'angolo misurato in gradi a partire dal nord, corrispondente a 0) tra la posizione dell'utente e quella del prossimo punto di interesse, per poi confrontarlo effettivamente con l'orientamento (heading) in quell'istante dell'utente (un angolo misurato sempre in corrispondenza del nord). Dalla differenza tra questi due valori, sono stati ricavati una serie di intervalli che corrispondono ad un'indicazione ben precisa. Ad esempio, per valori di differenza che vanno da 63 a 140 (valori approsimativi), si notifica all'utente di procedere verso destra. Nella stessa figura, sotto l'indicazione, è presente anche la distanza in metri dal prossimo punto di interesse. Essa è calcolata come somma delle lunghezze delle varie spezzate che compongono lo shortest path. Poichè il calcolo dello shortest path è oneroso (per quanto l'implementazione dell'algoritmo A* sia ottimale), occorre trovare un modo per limitare l'aggiornamento troppo frequente di queste informazioni: si può impostare una cadenza di aggiornamento di qualche secondo, oppure aggiornare solamente quando la nuova posizione dell'utente supera una certa soglia di distanza dalla precedente o si allontana sensibilmente

dallo shortest path. Nello sviluppo, per ottenere l'orientamento del device rispetto al nord si è utilizzato il plugin *DeviceOrientation*, di cui viene mostrato un esempio di seguito:

```
1  import {DeviceOrientation} from
       '@ionic-native/device-orientation/ngx';
2  .
3  .
4  .
5  constructor(private deviceOrientation: DeviceOrientation) {}
6  .
7  .
8  .
9  let heading;
10 this.deviceOrientation.watchHeading({
11     frequency: 2000
12 }).subscribe(data => {
13     heading = data.magneticHeading;
14 });
```

Questo plugin è stato utilizzato per il feedback visivo sulla mappa (la rotazione dell'icona dell'utente) e per la gestione del sistema di navigazione insieme ad un altro parametro, ossia il bearing tra la posizione utente e quella del prossimo punto di interesse. Questo è ottenuto utilizzando *Turf* come segue:

```
1 import * as turf from '@turf/turf';
2 .
3 .
4 .
5 const source = turf.point([..., ...]);
6 const destination = turf.point([..., ...]);
7 const bearing = turf.bearing(destination, source);
```

In Tabella 4.4 sono rappresentati gli intervalli e le conseguenti indicazioni da dare all'utente per guidarlo verso il prossimo punto di interesse, in cui d = *heading* - *bearing*:

Dritto	d > 140 OR d < 220
Indietro	d < 63 OR (d > 310 AND d < 415)
Sinistra	d > 220 AND d < 310
Destra	(d > 63 AND d < 140) OR d > 415

Tabella 4.4: Intervalli di d = *heading* - *bearing* con conseguenti indicazioni da dare all'utente

4.5.4 Curve di Bèzier

Lo shortest path generato è un insieme di linee spezzate, che partono dalla posizione dell'utente fino ad arrivare alla posizione del prossimo punto di interesse. Per migliorare il feedback visivo del percorso, si è deciso di approssimare questo insieme di linee spezzate in un'unica curva di Bèzier. Si è scelto di utilizzare questa approssimazione in quanto l'unica supportata all'interno di *Turf* per la gestione di questa problematica. Prendendo riferimento da [16], una curva di Bèzier, di grado n in un intervallo $[a,b]$ è un polinomio parametrico $\mathbf{p}$ tale che:

$$\mathbf{p} : [a,b] \rightarrow \mathbb{R}^d \tag{4.5}$$

Dato dalla formula:

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{c}_i B_i^n(u), \quad t \in [a, b] \tag{4.6}$$

Dove $u = (t - a)/(b - a)$ e i punti $\mathbf{c}_i$ sono i punti di controllo di $\mathbf{p}$. Definiamo anche i polinomi di Bernstein come:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-1}, \quad u \in [0,1] \tag{4.7}$$

Ad esempio, la Figura 4.43 mostra una curva di Bèzier cubica in cui polinomi di Bernstein sono:

$$B_0^3(u) = (1-u)^3, \;\; B_1^3(u) = 3u(1-u)^2, \;\; B_2^3(u) = 3u^2(1-u), \;\; B_3^3(u) = u^3 \tag{4.8}$$

Questo concetto, applicato al contesto di Smart Visit, permette di rendere lo shortest path

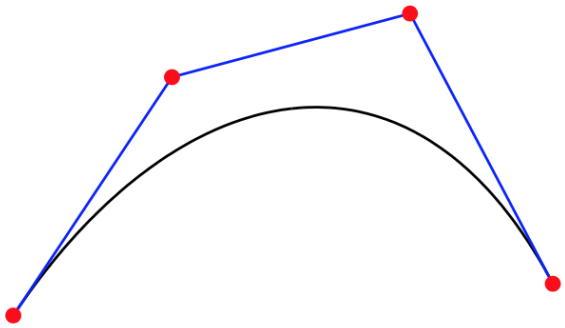

Figura 4.43: Curva di Bèzier cubica [16]

meno spigoloso ed evitare effetti di zig-zag. In Figura 4.44 viene mostrato il rendering su mappa senza e con l'utilizzo delle curve di Bèzier. Per concludere, nel seguente frammento

Figura 4.44: Shortest path tracciato senza (a sinistra) e con (a destra) approssimazione di Bèzier

di codice è riportato il calcolo dello shortest path per il contesto applicativo di Smart Visit. Dato un punto sorgente e un punto destinazione, viene calcolato il percorso tenendo conto di un ostacolo (il perimetro del sito) e di una determinata risoluzione (0.33 metri) per la generazione della griglia usata dall'algoritmo A*. Il risultato finale è la rappresentazione GeoJSON dell'approssimazione di Bèzier dello shortest path, che può essere visualizzato sulla mappa tramite la libreria *Leaflet*.

```
import * as turf from '@turf/turf';
.
.
.
const siteGeoJSON = ...;
const source = turf.point([..., ...]);
const destination = turf.point([..., ...]);
const site = turf.polygon(siteGeoJSON);
const shortestPath = turf.shortestPath(source, destination, {
    obstacles: site,
    units: 'meters',
    resolution: 0.33
});
const bezierCurve = turf.bezierSpline(shortestpath, {
    sharpness: 0.5
});
```

Capitolo 5

Fruizione di contenuti culturali

Durante la visita, l'utente avrà modo di essere guidato, tramite mappa e indicazioni, verso i vari punti di interesse del percorso scelto. Una volta che si troverà di fronte ad uno di essi, potrà aprire un'apposita sezione di dettagli in cui potrà usufruire di contenuti culturali di vario genere. Per Smart Visit è stata definita una particolare pagina che può essere visionata sia in loco (l'ultima situazione descritta), che in qualsiasi momento precedente alla visita (come durante la creazione di percorsi). Queste due situazioni vengono opportunamente differenziate, dando la possibilità di visionare già alcuni contenuti senza essere sul posto, ma presupponendo la possibilità di farlo per altri solo nel momento in cui si sta progredendo in una visita on site.

5.1 Analisi UI/UX

Questa sezione è tra le più importanti di tutta l'applicazione VASARI, in quanto è il momento in cui l'utente usufruisce della vera e propria esperienza artistica che un museo o un sito culturale può offrirgli. La visione dei contenuti deve essere chiara e immersiva, per contribuire al meglio all'esperienza globale del percorso. Il singolo punto di interesse è stato rappresentato tramite una *Card*: in essa appaiono subito dei contenuti visivi e testuali, ed è possibile approfondire il singolo punto di interesse mediante ulteriori pagine. La stessa struttura, meno ricca di contenuti, viene utilizzata anche per visionare i dettagli del punto di interesse off site. Nelle Figure 5.1 e 5.2 sono mostrati i rendering nei sistemi operativi Android e iOS per la pagina in questione in modalità on site, mentre nelle Figure 5.3 e 5.4 sono mostrate le stesse pagina in modalità off site.

5.1.1 Contenuti culturali

Vediamo nel dettaglio quali tipi di contenuti sono possibili visionare per un punto di interesse, nel momento in cui viene aperta la corrispondente pagina di dettagli.

Immagini

La prima cosa che appare nella *Card* è l'immagine primaria del punto di interesse. Utilizzando il plugin *PhotoViewer* è stata inserita la possibilità di visionare la stessa immagine a tutto schermo con gestures di zoom in e zoom out (in Figura 5.5). Oltre all'immagine primaria ne sono disponibili altre nella sezione "Contenuti associati". Di seguito è riportato il codice di gestione dell'anteprima tutto schermo di un'immagine, tramite il plugin descritto. Oltre alla risorsa (URL) dell'immagine, è possibile aggiungere un titolo all'anteprima (come in Figura 5.5) e delle opzioni a scelta.

```
1 import {PhotoViewer} from '@ionic-native/photo-viewer/ngx';
2 .
```

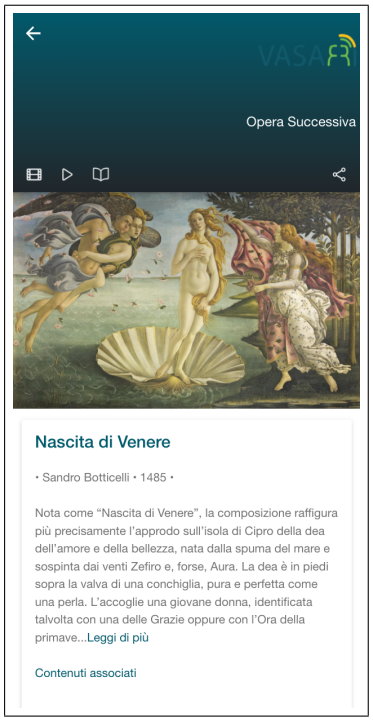

Figura 5.1: Pagina con dettagli di un punto di interesse on site renderizzata nel sistema operativo Android

Figura 5.2: Pagina con dettagli di un punto di interesse on site renderizzata nel sistema operativo iOS

```
3  .
4  .
5  constructor(private photoViewer: PhotoViewer) {}
6  .
7  .
8  .
9  const url = '...';
10 const title = '...';
11 const options = {
12     share: true
13 };
14 this.photoViewer.show(url, title, options);
```

Testo

Immediatamente sotto l’immagine è presente il titolo del punto di interesse e una sua descrizione. Essendoci la possibilità che sia abbastanza lunga, si è deciso di contrarla e di permettere all’utente di espanderla. In generale, la fruizione di descrizioni o altre fonti testuali non permette una totale immersione nel contesto culturale da parte dell’utente: questo perchè si rischia che la lettura sia troppo lunga e alienante nei confronti del percorso culturale che si è deciso di intraprendere. Per questo motivo, si è deciso di mantenere un’unica descrizione del bene (tradotta in varie lingue) e di dare più spazio ai contenuti di carattere multimediale.

Figura 5.3: Pagina con dettagli di un punto di interesse off site renderizzata nel sistema operativo Android

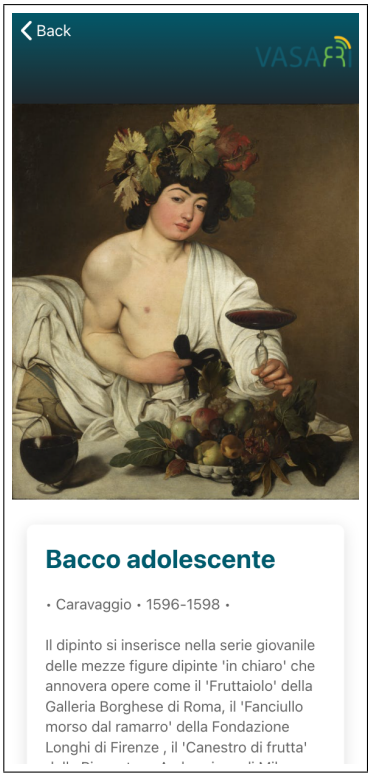

Figura 5.4: Pagina con dettagli di un punto di interesse off site renderizzata nel sistema operativo iOS

Figura 5.5: Anteprima a tutto schermo di un'immagine

Audio

Nel menù superiore è presente un pulsante che permette la riproduzione di una traccia audio, che non viene interrotta nel caso in cui l'utente decida di lasciare la pagina per fare altro (ad esempio, continuare l'esplorazione del sito). L'icona cambia aspetto quando la riproduzione è in corso, e cliccando nuovamente è possibile interromperla.
All'interno del progetto VASARI potrà capitare che, per un dato punto di interesse, non siano disponibili tracce audio o altri contenuti multimediali. Nel caso di contenuti udibili, sono stati utilizzati i seguenti plugin a seconda del contesto:

- *TextToSpeech*
  In mancanza di tracce audio, viene effettuata una sintesi vocale della descrizione testuale, che è sempre presente. Tramite questo plugin, viene riprodotta artificialmente una voce umana che può essere impostata in varie lingue e velocità di lettura. Un grosso vantaggio di questa tecnica è la scalabilità, ossia la possibilità di dare questa funzionalità a praticamente tutti i punti di interesse (compresi quelli che di default non avrebbero questo tipo di contenuti, utilizzando come sorgente la descrizione testuale, che è invece sempre presente). Lo svantaggio principale invece è che il livello di personalizzazione dato al contenuto è veramente ridotto, essendo il risultato molto artificiale e, per lunghe descrizioni, a tratti fastidioso. Si consiglia perciò di non utilizzare, a tale scopo, delle descrizioni troppo lunghe, anche in relazione a quanto detto in precedenza riguardo il rischio di alienazione dal percorso. Nel seguente frammento di codice, si abilita il sintetizzatore vocale per la riproduzione di un determinato testo. É possibile impostare la lingua di riferimento e la velocità di lettura.

```
1     import {TextToSpeech} from
     '@ionic-native/text-to-speech/ngx';
2     .
3     .
4     .
5     constructor(private tts: TextToSpeech) {}
6     .
7     .
8     .
9     const description = '...';
10    this.tts.speak({
11        text: description,
12        locale: 'it-IT',
13        rate: 1
14    });
15
```

- *NativeAudio*
  Questo plugin permette la riproduzione di tracce audio, accessibili via URL. Tra i vari plugin disponibili, si è scelto di utilizzare questo in quanto gli altri permettevano la riproduzione tramite player nativo, funzionalità che invece è più adatta ai video. É riportato di seguito il codice di riferimento al plugin. Bisogna in primo luogo caricare la traccia audio (tramite URL) e assegnare un identificativo, tramite il quale il plugin riconosce quale traccia riprodurre (infatti, è possibile caricare più tracce in contemporanea).

```
1     import {NativeAudio} from '@ionic-native/native-audio/ngx';
2     .
3     .
4     .
5     constructor(private nativeAudio: NativeAudio) {}
```

```
6       .
7       .
8       .
9       const url = '...';
10      this.nativeAudio.preloadSimple('Test', url);
11      .
12      .
13      .
14      this.nativeAudio.play('Test');
15
```

Nel caso di visite totalmente immersive, ossia con fruizione di contenuti che vanno di pari passo all'esplorazione del sito, sono presenti delle tracce audio più lunghe, la cui riproduzione deve continuare anche quando il dispositivo è in stand-by. Perciò è stato inserito un altro plugin, chiamato *BackgroundMode*, che permette di monitorare il flusso della modalità background, e di iscrivere degli handler quando essa viene attivata. La gestione della modalità background (e del suo ciclo di vita) è riportata di seguito:

```
1  import {BackgroundMode} from '@ionic-native/background-mode/ngx';
2  .
3  .
4  .
5  constructor(private backgroundMode: BackgroundMode) {}
6  .
7  .
8  .
9  this.backgroundMode.enable();
10 .
11 .
12 .
13 this.backgroundMode.on('activate').subscribe(() => {
14     console.log('Background mode on');
15 });
```

Video

Sopra l'immagine del punto di interesse, è presente un menù con varie opzioni: la prima di esse permette l'apertura di un player nativo (in Figura 5.6) per la riproduzione di video, di cui vengono opportunamente forniti gli URL insieme a tutti gli altri campi. É stato utilizzato il plugin *StreamingMedia*, che permette di personalizzare il player con varie opzioni. In particolare, l'orientamento, i controlli, eventi automatici come la chiusura del player alla fine del video, o delle callback che si azionano in caso di errore o di terminazione del video. Nel seguente frammento di codice, si riporta il funzionamento del plugin citato per la riproduzione di un video con il player nativo del sistema operativo. Oltre all'URL che contiene il video, è possibile aggiungere delle opzioni: ad esempio, si vuole un orientamento landscape, l'autochiusura del player alla fine della riproduzione, e l'abilitazione dei controlli tramite i quali si può navigare nella riproduzione del video (come in Figura 5.6). Inoltre, è possibile impostare delle callback di successo (richiamata alla fine della riproduzione) o fallimento (per gestire eventuali errori).

```
1  import {StreamingMedia} from '@ionic-native/streaming-media/ngx';
2  .
3  .
```

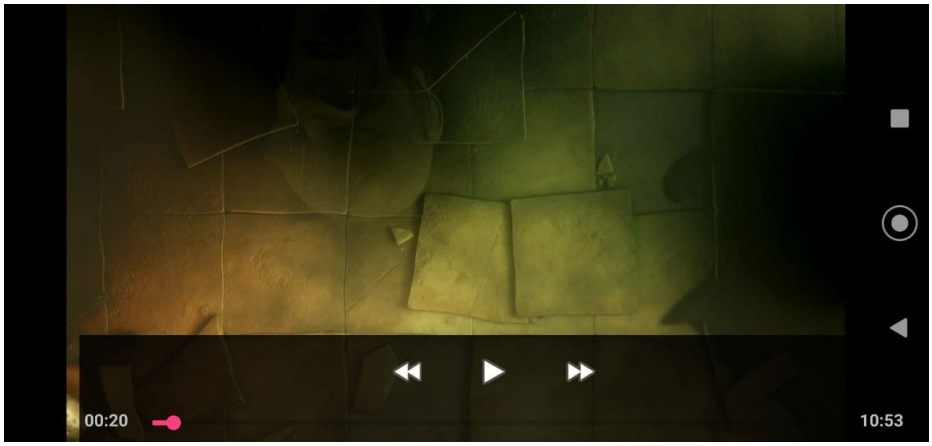

Figura 5.6: Player per la riproduzione di video

```
4  .
5  constructor(private streamingMedia: StreamingMedia) {}
6  .
7  .
8  .
9  const url = '...';
10 const options = {
11     successCallback: () => { console.log('Video played'); },
12     errorCallback: () => { console.log('Streaming error'); },
13     orientation: 'landscape',
14     shouldAutoClose: true,
15     controls: true
16 };
17 this.streamingMedia.playVideo(url, options);
```

Realtà aumentata

Per la visione di questo tipo di contenuti, all'interno del progetto VASARI è stato deciso l'utilizzo di *Wikitude* [44]. Si tratta di un SDK (Support Development Kit) con supporto cross-platform che permette sia di effettuare tracking offline di oggetti o scenari, sia di creare esperienze di realtà aumentata (di vario genere). Permette la fruizione di questi contenuti sia in maniera classica (tramite Smartphone o Tablet), sia in maniera più immersiva (tramite visori VR o Smart Glasses). Tramite *Wikitude*, in ambito culturale sono state sviluppate numerose applicazioni basate sul riconoscimento immagini e sulla realtà aumentata. Ad esempio, tra i casi d'uso mostrati in [44], vi è l'applicazione mobile cross-platform [39], che permette ai visitatori nella Galleria degli Uffizi di scannerizzare e identificare immediatamente più di 500 opere d'arte, sbloccando dei contenuti audio atti ad arricchire la classica esperienza culturale. Un altro esempio si ha con [21], un'applicazione mobile del Franklin Institute per la visualizzazione di elementi in realtà aumentata in sovraimpressione ad immagini reali di alcune statue in argilla. In Figura 5.7 sono mostrati alcuni esempi tratti da [21]. In particolare, l'esperienza in realtà aumentata per queste opere consiste nella visualizzazione di modelli 3D di armi che si vanno a posizione nelle mani dei soldati. Nel contesto applicativo di Smart Visit, vi è stato bisogno di abilitare l'utilizzo di esperienze in realtà aumentata con questo SDK. Questo avviene tramite un plugin che però non è più in versione *Capacitor* come tutti quelli mostrati fino ad ora, ma è nella versione *Cordova*. Il plugin (*WikitudePlugin*) permette, dato un URL contenente l'esperienza da visionare, di caricare l'esperienza a tutto schermo e in

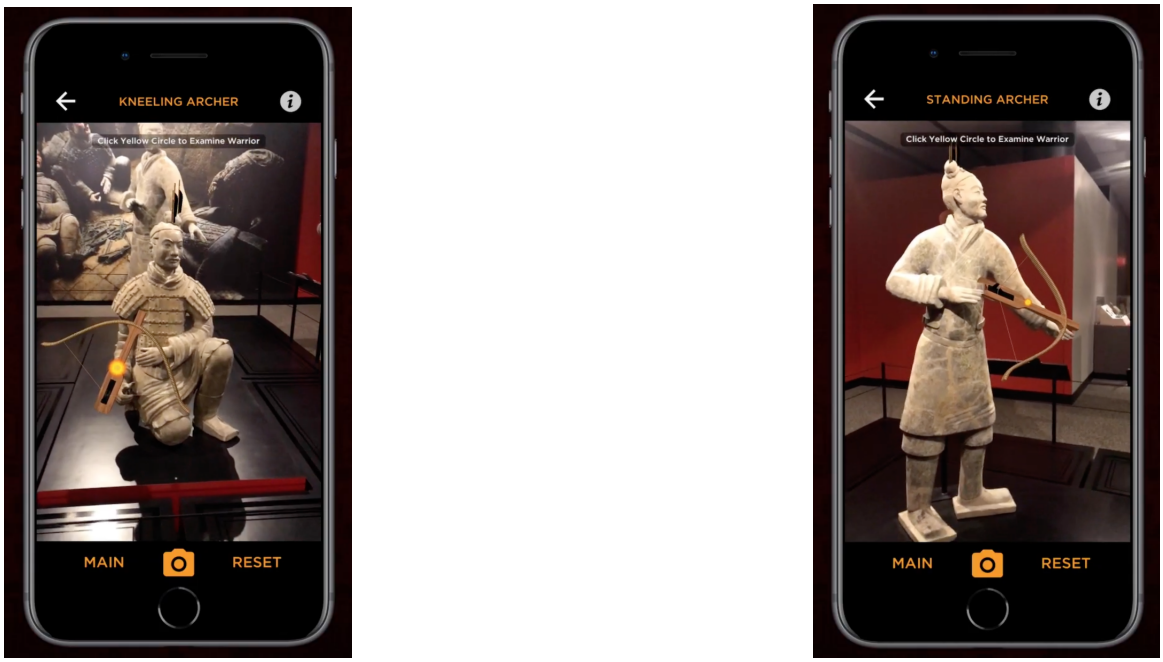

Figura 5.7: Esempi di contenuti culturali in realtà aumentata [21]

maniera interattiva (utilizzando la fotocamera per scansionare eventuali oggetti o marker utili all'attivazione del contenuto). Di seguito è riportato un frammento di codice dell'integrazione di *Wikitude* in un progetto *Ionic/Angular*:

```
const arExperienceURL = "...";
const requiredFeatures = ['image_tracking', 'geo'];
const startupConfiguration = {
    camera_position = "back"
}
const wikitudePlugin =
    cordova.require('com.wikitude.phonegap.wikitudeplugin.WikitudePlugin');
wikitudePlugin.isDeviceSupported(
    () => {
        console.log('Device supported');
        wikitudePlugin.loadARchitectWorld(
            () => {
                console.log('ARchitect World loaded');
            },
            (err) => {
                console.log(err);
            },
            arExperienceURL,
            requiredFeatures,
            startupConfiguration
        );
    },
    (err) => {
        console.log(err)
    },
    requiredFeatures
)
```

In Figura 5.8 sono riportati alcuni esempi di realtà aumentata inseriti tramite *Wikitude*: nel primo viene renderizzato un modello del sistema solare in rotazione, mentre nel secondo un modellino d'auto (collegato ad un target mostrato in uno schermo) su cui si può azionare un evento (tramite un pulsante) che lo fa cominciare a ruotare su se stesso. Utilizzando lo

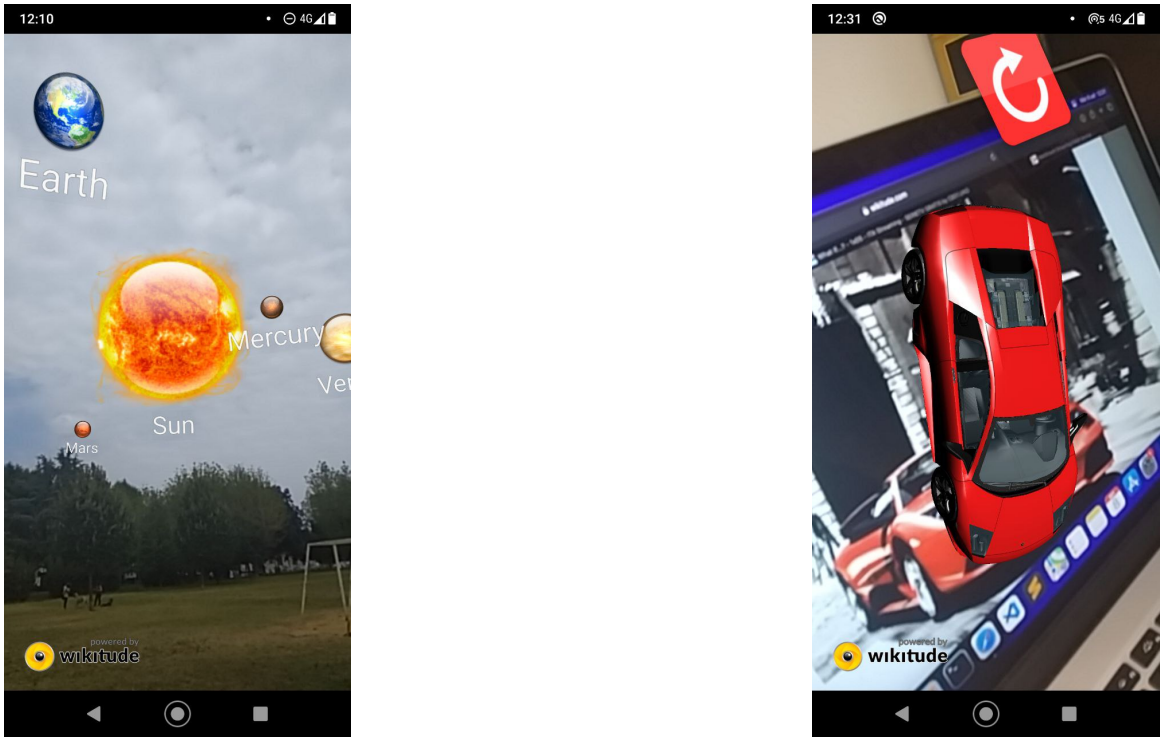

Figura 5.8: Esempi di realtà aumentata con *Wikitude*

strumento [42] messo a disposizione da *Wikitude*, si è creata un'esperienza in realtà aumentata specifica per un punto di interesse del proof of concept di Smart Visit. In particolare è stata utilizzata un'immagine 2D dell'opera per generare, tramite *Wikitude Studio*, un file contenente tutte le informazioni necessarie per il riconoscimento e il tracking della stessa, per poi inserire elementi di realtà aumentata (etichette e modelli 3D). L'esperienza in realtà aumentata di esempio è mostrata in Figura 5.9, con il rendering su device mostrato in Figura 5.10: essa arricchisce la *Nascita di Venere* con i nomi dei vari personaggi ed elementi ambientali 3D (cespugli e farfalle). *Wikitude Studio* permette di esportare l'esperienza sotto forma di progetto HTML/CSS/JS, per poi attivare il file *index.html* tramite il plugin riportato in precedenza.

5.1.2 Galleria multimediale

Nel momento in cui un punto di interesse sia accompagnato da più contenuti visivi o multimediali, è stata inserita una sezione aggiuntiva (al di fuori della *Card*) che permette la visualizzazione di tutti i contenuti (la voce "Contenuti associati"). La pagina è stata predisposta con un *Tabs Menù*, tramite il quale si accede ai diversi generi di contenuti (di cui ne è contrassegnato il numero). Per ogni genere, vi è l'elenco scrollabile dei contenuti, che possono essere rappresentati in maniera più vistosa essendo specifici per quella sezione. Ad esempio, per un video o un audio sono presenti delle immagini di anteprima (e non soltato l'icona dello specifico pulsante, come nella pagina precedente). Ogni contenuto è poi trattato secondo le modalità e i plugin descritti in precedenza. É disponibile infine un pulsante (con l'icona *i*) con cui accedere ad approfondimenti esterni ed essere reindirizzati, tramite browser, ad una pagina web autonoma e indipendente dal contesto di VASARI (ad esempio, la pagina web della Galleria degli Uffizi). Nelle Figure 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 e 5.18 è mostrata la pagina con i contenuti associati renderizzata nei sistemi operativi Android e iOS,

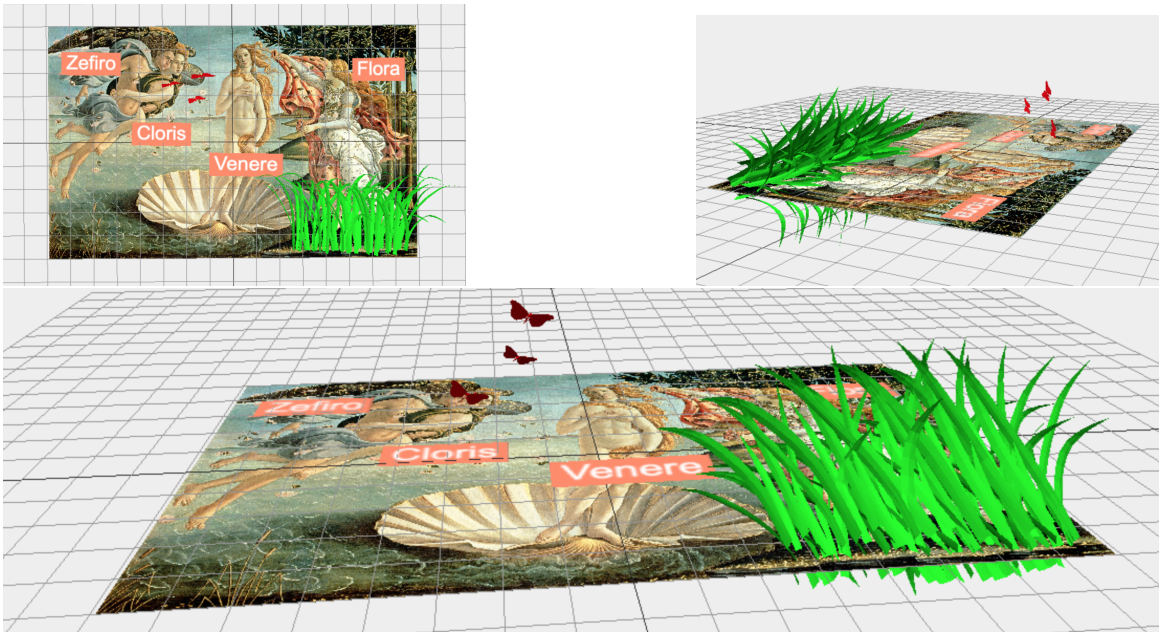

Figura 5.9: Esperienza in realtà aumentata generata con *Wikitude Studio*

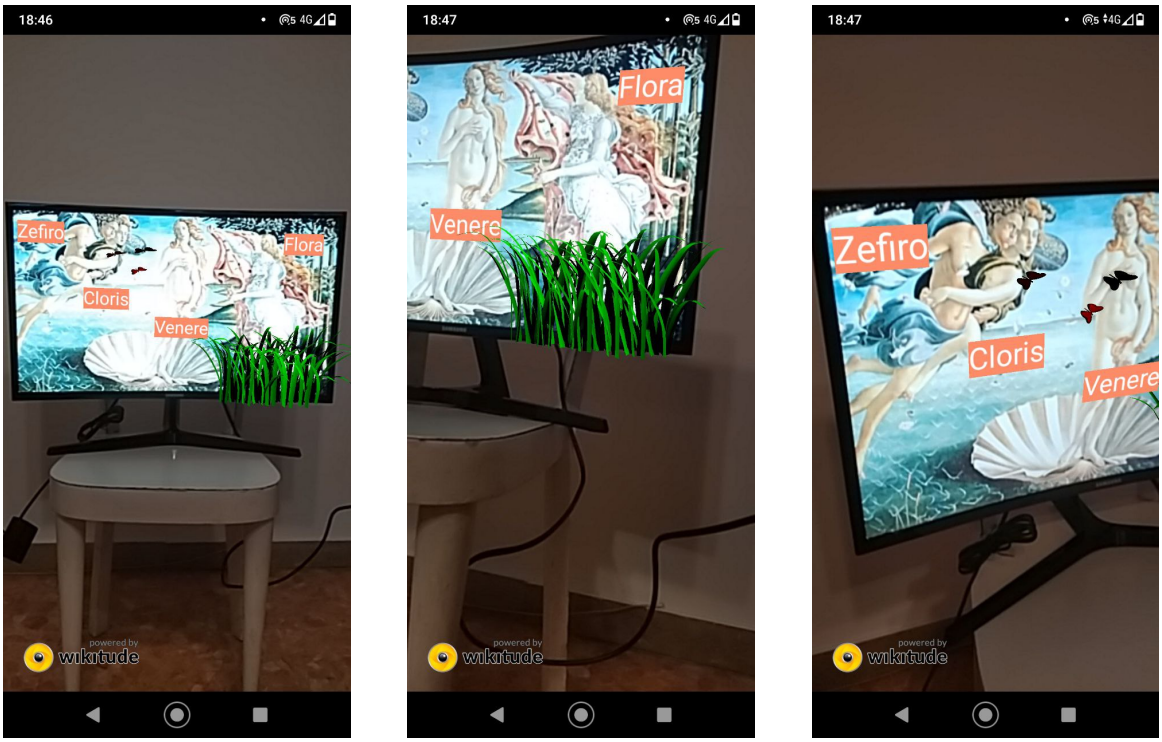

Figura 5.10: Esperienza in realtà aumentata generata renderizzata sul dispositivo fisico

nelle varie sezioni (immagini, audio, video, altro). Per la sezione immagini si è implementata una griglia che favorisce lo scorrimento rapido tra di esse, mentre per gli audio e i video si è creata una *Card* con titolo, durata ed eventualmente l'anteprima (per i video). Infine, per la sezione "altro", ogni contenuto massimizza lo spazio disponibile (in altezza e larghezza) tramite delle *Slides*.

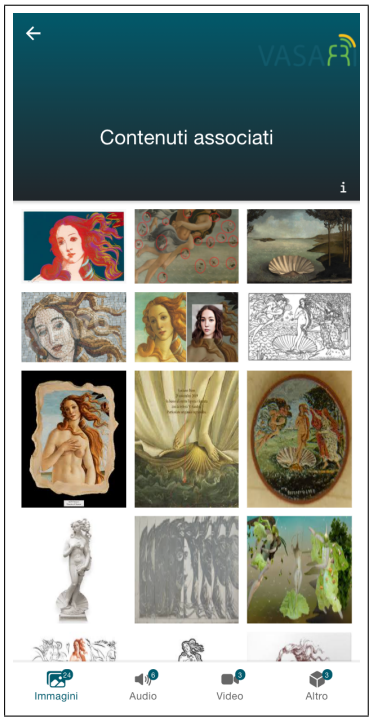

Figura 5.11: Pagina con galleria (sezione immagini) renderizzata nel sistema operativo Android

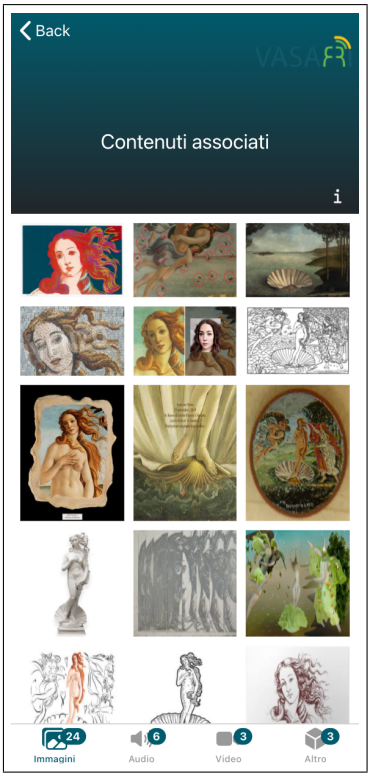

Figura 5.12: Pagina con galleria (sezione immagini) renderizzata nel sistema operativo iOS

5.1.3 Interazione con l'utente

Per finire, vediamo quali sono le funzionalità dedicate esclusivamente al punto di interesse nel contesto di appartenenza al percorso culturale.

Azioni principali

Nel momento in cui l'utente visiona i dettagli di un'opera, ha la possibilità di navigare tra le opere del percorso direttamente nella stessa pagina (tramite i pulsanti "Opera precedente" e "Opera successiva"), di visionare i contenuti associati o i collegamenti tematici, di condividere sui social o di confermare il completamento della stessa tramite un pulsante posto in fondo alla pagina. Per quanto riguarda la decisione di rendere alcuni contenuti obbligatori e altri facoltativi (al fine di completare il punto di interesse), essa è totalmente demandata alla logica e alla libertà del percorso che si è deciso di intraprendere. Prima di confermare un punto di interesse, è possibile lasciare un feedback con uno specifico *Popover* (tramite valutazione da 0 a 5 stelle più eventuali commenti, in Figura 5.19). Il menù superiore è mostrato in Figura 5.20, mentre le opzioni di completamento e visione dei contenuti associati sono mostrate in Figura 5.21.

Figura 5.13: Pagina con galleria (sezione audio) renderizzata nel sistema operativo Android

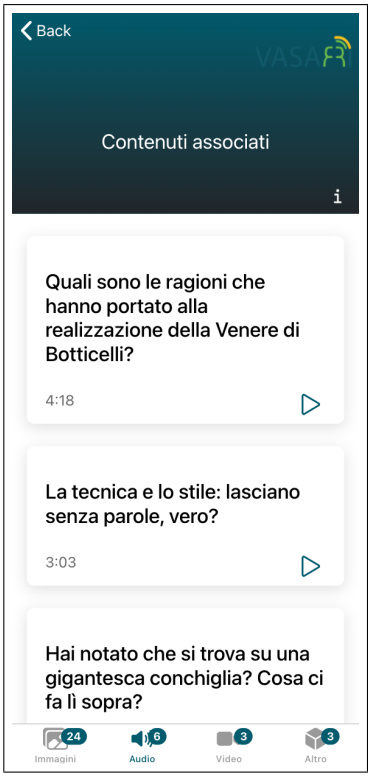

Figura 5.14: Pagina con galleria (sezione audio) renderizzata nel sistema operativo iOS

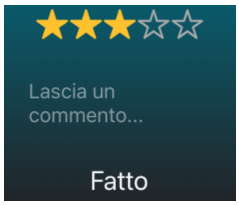

Figura 5.19: *Popover* per lasciare un feedback su un punto di interesse

Figura 5.21: Opzioni di completamento e visione dei contenuti associati

Condivisione social

All'interno del menù posto sopra l'immagine, sono disponibili altre funzionalità aggiuntive (oltre alla riproduzione video). Una di esse permette la condivisione tramite social del punto di interesse, implementata tramite un *Modal* che si apre dal basso e che presenta più vie di condivisione (Facebook, Instagram, Whatsapp, Twitter, Email). Questo componente è mostrato nell Figure 5.22 e 5.23 . Ognuna delle opzioni del *Modal* è gestita tramite il plugin

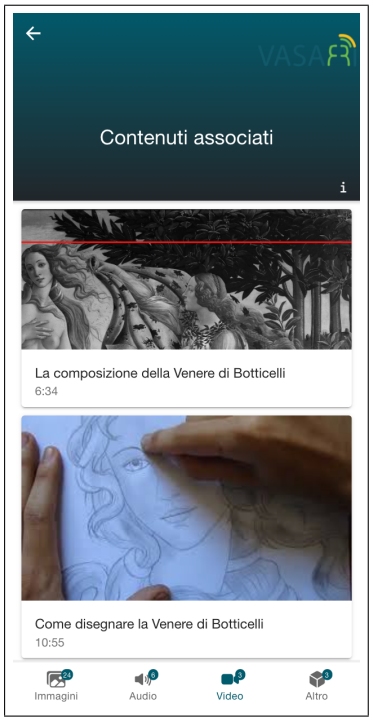

Figura 5.15: Pagina con galleria (sezione video) renderizzata nel sistema operativo Android

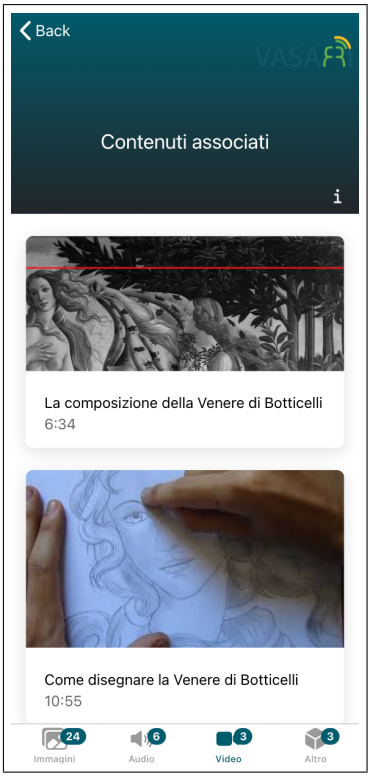

Figura 5.16: Pagina con galleria (sezione video) renderizzata nel sistema operativo iOS

SocialSharing. Nel seguente frammento di codice, si condivide tramite Facebook (tra le tante opzioni disponibili) la descrizione di un punto di interesse e l'URL della sua immagine:

```
1  import {SocialSharing} from '@ionic-native/social-sharing/ngx';
2  .
3  .
4  .
5  constructor(private socialSharing: SocialSharing) {}
6  .
7  .
8  .
9  const description = '...';
10 const url = '...';
11 this.socialSharing.shareViaFacebook(description, url);
```

Collegamenti tematici

Con un'altra opzione si può accedere ad una sezione di collegamenti tematici: in essa sono presenti dei punti di interesse che, per un motivo e per un altro, sono associati e consigliati: ad esempio, perchè appartenenti allo stesso autore, alla stessa corrente artistica, hanno la stessa tematica e così via. Cliccando su uno dei collegamenti tematici, è possibile visionare i dettagli di quel punto di interesse, con le stesse modalità fin'ora descritte. Nelle Figure 5.24 e 5.25 sono mostrati i rendering nei due sistemi operativi della suddetta pagina.

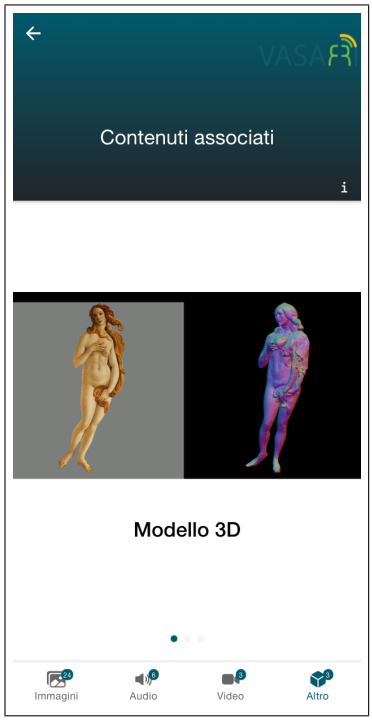

Figura 5.17: Pagina con galleria (sezione altro) renderizzata nel sistema operativo Android

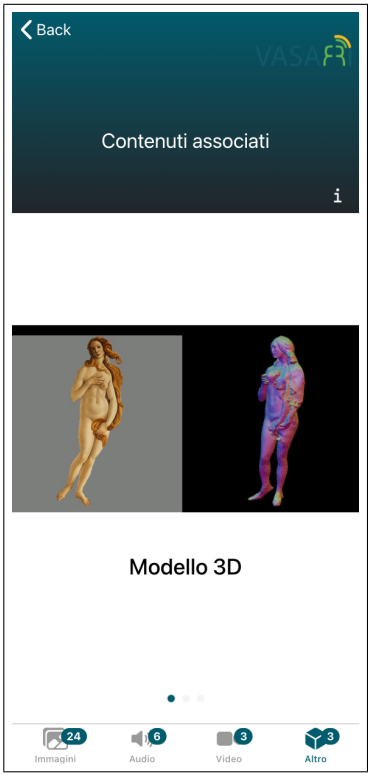

Figura 5.18: Pagina con galleria (sezione altro) renderizzata nel sistema operativo iOS

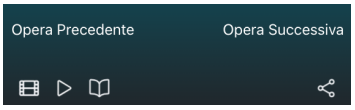

Figura 5.20: Menù superiore nella pagina di dettagli di un punto di interesse

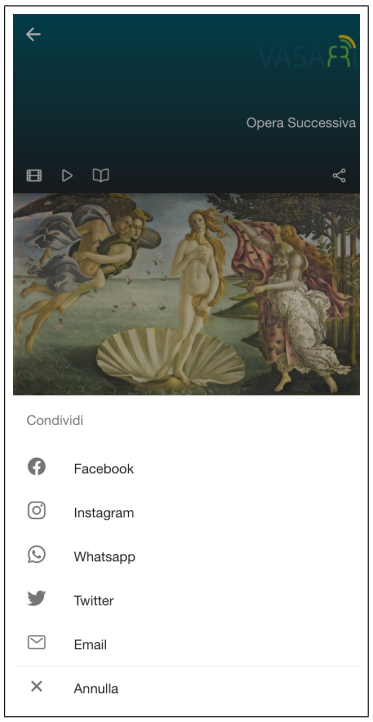

Figura 5.22: *Modal* con opzioni di condivisione social del punto di interesse renderizzato nel sistema operativo Android

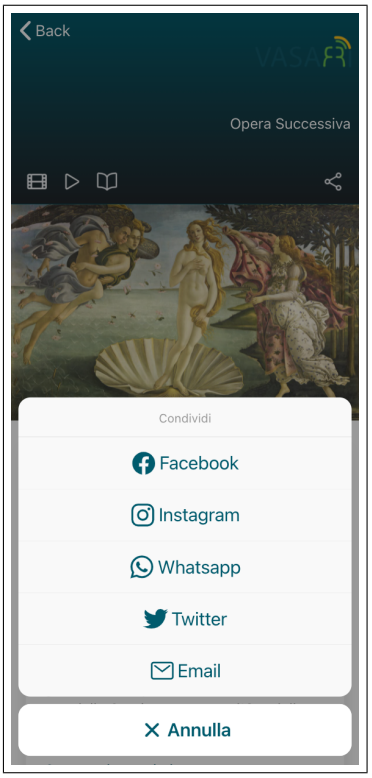

Figura 5.23: *Modal* con opzioni di condivisione social del punto di interesse renderizzato nel sistema operativo iOS

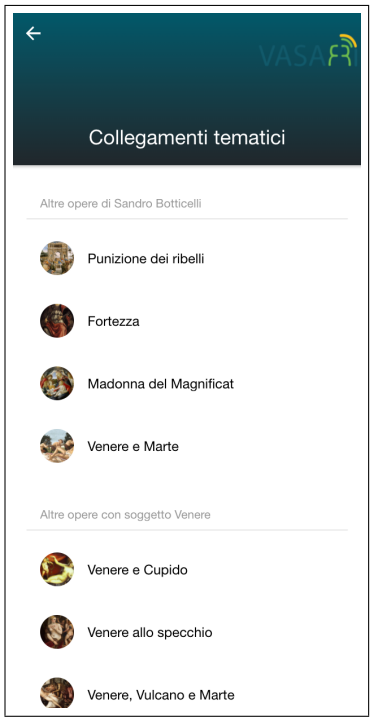

Figura 5.24: Pagina con collegamenti tematici renderizzata nel sistema operativo Android

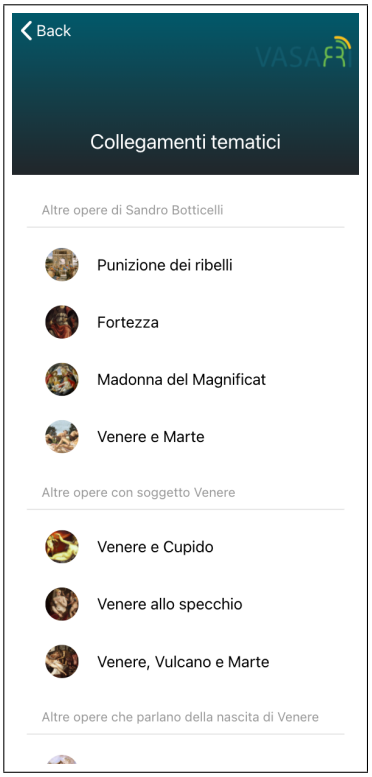

Figura 5.25: Pagina con collegamenti tematici renderizzata nel sistema operativo iOS

Capitolo 6

Conclusioni

In questo lavoro di tesi si è parlato dell'implementazione del modulo Smart Visit all'interno dell'applicazione mobile VASARI. La decisione di optare per un approccio cross-platform, piuttosto che nativo, ha portato il vantaggio di avere un'unica base di codice per lo sviluppo su più piattaforme. Questo sviluppo, inoltre, si è basato su tecnologie client-side e linguaggi di programmazioni tipici della programmazione Web, molto meno specialistici rispetto a quelli utilizzati per lo sviluppo nativo. I principali problemi, tuttavia, si sono manifestati proprio in quelli che sono i difetti dell'approccio cross-platform: la necessità di utilizzare funzionalità native (come i dati derivanti da vari sensori o il monitoraggio di alcuni stati del device) ha portato all'impiego di plugin che, se presenti in quantità significative, appesantiscono nel complesso l'applicazione. In un possibile sviluppo futuro, al di fuori del contesto di questa tesi, potrebbe essere valutato un approccio nativo: esso risulterebbe senza dubbio più performante, anche in device più datati, ma porterebbe inevitabilmente ad un aumento dei costi e del tempo di sviluppo.
Parlando dei principali aspetti descritti nel corso dei vari capitoli, vi è stata un'analisi della User Experience e della User Interface dell'applicazione, in relazione anche ad indicazioni (generiche o mirate) fornite da paper di ricerca o articoli. Dal punto di vista visivo, si è cercato di mantenere un aspetto minimale e omogeneo che possa facilitare l'utente nell'utilizzo, senza togliere alcuni elementi di personalizzazione e di branding che il progetto VASARI vuole portare avanti. Stesso discorso per l'esperienza di utilizzo: unificare tramite i medesimi meccanismi (gestures, componenti, menù e così via) le varie pagine dell'app permette all'utente di cogliere in fretta la filosofia di utilizzo della stessa e padroneggiarla nel più breve tempo possibile. Una possibile futura aggiunta potrebbe essere l'inserimento degli altri principali moduli (oltre Smart Visit) che sono diversi (ognuno con le proprie caratteristiche e obiettivi) ma complementari tra di loro: un design (UI/UX) comune a tutti i principali moduli sarebbe un pilastro fondamentale per il successo del progetto, per evitare di dare all'utente un prodotto internamente poco collegato ed esternamente confusionario.
Un altro aspetto principale, più specifico di Smart Visit, è stato il meccanismo di guida lungo il percorso fisico. L'utilizzo della mappa ricopre un aspetto primario: deve essere interattiva ma allo stesso tempo non deve alienare l'utente dall'ambiente circostante. Inoltre, i meccanismi di localizzazione e posizionamento devono essere basati sul tipo di tecnologia che si decide di utilizzare in ogni sito culturale: se per il contesto outdoor l'utilizzo del GPS è predominante (data la sua accuratezza all'aperto), per quello indoor sorgono più problemi. Laddove il tipo di tecnologia utilizzata non sia molto affidabile, possono essere valutati dei meccanismi basati sul Machine Learning in grado di utilizzare anche informazioni corrotte per un buon posizionamento dell'utente all'interno dell'ambiente. Per il proof of concept ci si è basati su un piccolo ambiente indoor domestico con un ridotto numero di nodi ancora (3 Beacon BLE): un possibile futuro oggetto di studio potrebbe essere l'implementazione di tali tecniche in ambienti più grandi e dispersivi con un numero di nodi maggiore, in maniera tale da allenare dei modelli di intelligenza artificiale su dataset più grandi e dimensionalmente più ricchi.

Infine, un altro punto cardine del lavoro di tesi è stata la gestione dei contenuti culturali. Le descrizioni testuali, forma classica di fruizione, sono state accompagnate da altre forme multimediali (immagini, audio e video) e arricchite tramite realtà aumentata. L'ordine con cui questi diversi tipi di contenuti vengono forniti all'utente è molto importante: partendo da quelle più basilari (il testo) si arriva a quelle più complesse (la realtà aumentata), in maniera tale da permettere all'utente sia una fruizione generica che approfondita. Uno sviluppo futuro, al di fuori di questo proof of concept, potrebbe essere l'analisi di altre forme di fruizione culturale come la realtà virtuale o basate sull'interazione tra utenti e personale specializzato.

Elenco delle figure

1.1 Architettura del progetto VASARI [33] . . . 2
1.2 Flusso logico del modulo di Smart Visit . . . 3
1.3 Dettagli di un punto di interesse dell'applicazione su Malta [5] . . . 6
1.4 Fruizione di un contenuto multimediale dell'applicazione su Malta [5] . . . 6
1.5 Guida dell'utente tramite mappa e informazioni geospaziali dell'applicazione su Malta [5] . . . 6
1.6 Listato di punti di interesse dell'applicazione su Malta [5] . . . 6
1.7 Flow Chart dello sviluppo della User Interface [12] . . . 7
1.8 Fattori da prendere in considerazione per il design della UX [15] . . . 8
1.9 Complementarità tra User Interface e User Experience (UI/UX) [10] . . . 9
1.10 Situazione attuale della distribuzione sul mercato dei sistemi operativi per smartphone, secondo IDC (International Data Corporation) [38] . . . 10
1.11 Confronto tra sviluppo nativo e cross-platform [11] . . . 11

2.1 Pattern MVC [7] . . . 14
2.2 Componente renderizzato nel sistema operativo Android . . . 19
2.3 Componente renderizzato nel sistema operativo iOS . . . 19
2.4 Pagina principale renderizzata nel sistema operativo Android . . . 21
2.5 Pagina principale renderizzata nel sistema operativo iOS . . . 21
2.6 Pagina di dettagli renderizzata nel sistema operativo Android . . . 22
2.7 Pagina di dettagli renderizzata nel sistema operativo iOS . . . 22

3.1 Pagina iniziale renderizzata nel sistema operativo Android . . . 26
3.2 Pagina iniziale renderizzata nel sistema operativo iOS . . . 26
3.3 Pagina con menù renderizzata nel sistema operativo Android . . . 27
3.4 Pagina con menù renderizzata nel sistema operativo iOS . . . 27
3.5 Pagina di ricerca renderizzata nel sistema operativo Android . . . 28
3.6 Pagina di ricerca renderizzata nel sistema operativo iOS . . . 28
3.7 Pagina con risultati della ricerca renderizzata nel sistema operativo Android . . . 29
3.8 Pagina con risultati della ricerca renderizzata nel sistema operativo iOS . . . 29
3.9 Pagina con dettagli di un risultato della ricerca renderizzata nel sistema operativo Android . . . 30
3.10 Pagina con dettagli di un risultato della ricerca renderizzata nel sistema operativo iOS . . . 30
3.11 Componente di selezione bozza renderizzato nel sistema operativo Android . . . 30
3.12 Componente di selezione bozza renderizzato nel sistema operativo iOS . . . 30
3.13 Componente di selezione percorso renderizzato nel sistema operativo Android . . . 30
3.14 Componente di selezione percorso renderizzato nel sistema operativo iOS . . . 30
3.15 Pagina delle bozze renderizzata nel sistema operativo Android . . . 31
3.16 Pagina delle bozze renderizzata nel sistema operativo iOS . . . 31
3.17 Componente di creazione bozza renderizzato nel sistema operativo Android . . . 31
3.18 Componente di creazione bozza renderizzato nel sistema operativo iOS . . . 31

3.19 Pagina di dettagli di una bozza renderizzata nel sistema operativo Android . . 32
3.20 Pagina di dettagli di una bozza renderizzata nel sistema operativo iOS 32
3.21 Pagina dei percorsi renderizzata nel sistema operativo Android 32
3.22 Pagina dei percorsi renderizzata nel sistema operativo iOS 32
3.23 Pagina di dettagli di un percorso renderizzata nel sistema operativo Android . 33
3.24 Pagina di dettagli di un percorso renderizzata nel sistema operativo iOS 33
3.25 Pagina iniziale con collegamento rapido al percorso renderizzata nel sistema operativo Android 34
3.26 Pagina iniziale con collegamento rapido al percorso renderizzata nel sistema operativo iOS 34

4.1 Visualizzazione delle informazioni formattate nel GeoJSON di esempio tramite il tool online gratuito geojson.io [28] 38
4.2 Esempio di mappa contenente dei GeoJSON creata con *Leaflet* [18] 39
4.3 Centroide di un poligono calcolato con *Turf* [8] 39
4.4 Shortest path con un ostacolo tra due punti generato con *Turf* [37] 39
4.5 *Alidade Smooth* [1] 42
4.6 *Alidade Smooth Dark* [1] 42
4.7 Floating Action Button tramite il quale si possono espandere delle funzionalità di gestione della mappa 43
4.8 Funzionalità di gestione della mappa 43
4.9 Funzionalità aggiuntive per la mappa 43
4.10 Floating Action Button per utilizzare lo scanner QR 43
4.11 Instantanea di cattura di codici QR 43
4.12 *Segment Menù* per il cambiamento dinamico di piano 45
4.13 Popup del sito culturale 45
4.14 GeoJSON del perimetro del sito culturale, integrato con l'utente e i punti di interesse, rappresentato in *Leaflet* 46
4.15 GeoJSON del perimetro del sito culturale con scale/ascensori, integrato con l'utente rappresentati in *Leaflet* 46
4.16 Pagina di guida dell'utente renderizzata nel sistema operativo Android 46
4.17 Pagina di guida dell'utente renderizzata nel sistema operativo iOS 46
4.18 Barra di caricamento all'inizio della visita 47
4.19 Progresso mostrato nella barra di caricamento 47
4.20 Barra di caricamento quando la visita è stata completata (tutti i punti di interesse visionati) 48
4.21 Pulsante di modifica del percorso all'interno della barra di caricamento 48
4.22 Lista dei punti di interesse che formano un percorso, con una spunta verde quando sono stati completati 49
4.23 *Card* del prossimo punto di interesse da visionare 49
4.24 Line of sight (LOS) [34] 50
4.25 Non-line of sight (NLOS) [34] 50
4.26 Diagramma schematico di posizionamento utilizzando nodi ancora [31] 50
4.27 Ambiente domestico di test 54
4.28 Beacon BLE usati durante la fase sperimentale 54
4.29 Disposizione dei beacon in un ambiente di test [36] 55
4.30 Posizione dei beacon all'interno dell'ambiente domestico di test 55
4.31 Utilizzo dei dati grezzi per la localizzazione [13] 57
4.32 Utilizzo della media mobile per la localizzazione [13] 57
4.33 Utilizzo della media pesata per la localizzazione [13] 58
4.34 Utilizzo della curva di apprendimento per la localizzazione [13] 58
4.35 Analisi del Beacon 517, posto a circa 6 metri dal nodo target 59
4.36 Analisi del Beacon 518, posto a circa 16 metri dal nodo target 59

4.37 Analisi del Beacon 519, posto a circa 30 metri dal nodo target 60
4.38 Frammento di dataset 61
4.39 Layers della Rete Neurale nel dettaglio 63
4.40 Rete Neurale utilizzata per l'ambiente indoor di test, generata con il tool online gratuito [32] 63
4.41 Esempi di shortest path tracciati tramite le libreria *Turf* 65
4.42 Griglia 2-D con 3 ostacoli [4] 67
4.43 Curva di Bèzier cubica [16] 69
4.44 Shortest path tracciato senza (a sinistra) e con (a destra) approssimazione di Bèzier 69

5.1 Pagina con dettagli di un punto di interesse on site renderizzata nel sistema operativo Android 72
5.2 Pagina con dettagli di un punto di interesse on site renderizzata nel sistema operativo iOS 72
5.3 Pagina con dettagli di un punto di interesse off site renderizzata nel sistema operativo Android 73
5.4 Pagina con dettagli di un punto di interesse off site renderizzata nel sistema operativo iOS 73
5.5 Anteprima a tutto schermo di un'immagine 73
5.6 Player per la riproduzione di video 76
5.7 Esempi di contenuti culturali in realtà aumentata [21] 77
5.8 Esempi di realtà aumentata con *Wikitude* 78
5.9 Esperienza in realtà aumentata generata con *Wikitude Studio* 79
5.10 Esperienza in realtà aumentata generata renderizzata sul dispositivo fisico . . . 79
5.11 Pagina con galleria (sezione immagini) renderizzata nel sistema operativo Android 80
5.12 Pagina con galleria (sezione immagini) renderizzata nel sistema operativo iOS 80
5.13 Pagina con galleria (sezione audio) renderizzata nel sistema operativo Android 81
5.14 Pagina con galleria (sezione audio) renderizzata nel sistema operativo iOS . . . 81
5.19 *Popover* per lasciare un feedback su un punto di interesse 81
5.21 Opzioni di completamento e visione dei contenuti associati 81
5.15 Pagina con galleria (sezione video) renderizzata nel sistema operativo Android 82
5.16 Pagina con galleria (sezione video) renderizzata nel sistema operativo iOS . . . 82
5.17 Pagina con galleria (sezione altro) renderizzata nel sistema operativo Android 83
5.18 Pagina con galleria (sezione altro) renderizzata nel sistema operativo iOS . . . 83
5.20 Menù superiore nella pagina di dettagli di un punto di interesse 83
5.22 *Modal* con opzioni di condivisione social del punto di interesse renderizzato nel sistema operativo Android 84
5.23 *Modal* con opzioni di condivisione social del punto di interesse renderizzato nel sistema operativo iOS 84
5.24 Pagina con collegamenti tematici renderizzata nel sistema operativo Android . 85
5.25 Pagina con collegamenti tematici renderizzata nel sistema operativo iOS 85

Elenco delle tabelle

4.1 Vantaggi e svantaggi di diverse tecniche di "ranging" [31] 51
4.2 Vantaggi e svantaggi di diverse tecniche di localizzazione [31] 53
4.3 Accuratezza del K-Nearest Neighbor per vari valori di K 62
4.4 Intervalli di d = *heading* - *bearing* con conseguenti indicazioni da dare all'utente 68

Bibliografia

[1] URL: https://stadiamaps.com/themes/.

[2] URL: https://angular.io/.

[3] URL: https://geojson.org/.

[4] Antti Autere. *Extensions and applications of the A* algorithm*. English. Doctoral thesis. 2005.

[5] Stefania Boiano, Jonathan Bowen e Giuliano Gaia. «Usability, Design and Content Issues of Mobile Apps for Cultural Heritage Promotion: The Malta Culture Guide Experience». In: lug. 2012. DOI: 10.14236/ewic/EVA2012.12.

[6] Anders Bouwer, Frank Nack e Abdallah El Ali. «Lost in navigation: Evaluating a mobile map app for a fair». In: ott. 2012, pp. 173–180. ISBN: 9781450314671. DOI: 10.1145/2388676.2388712.

[7] di Francesco Camarlinghi e Francesco Camarlinghi. *Il pattern MVC*. Gen. 2019. URL: https://www.html.it/pag/18299/il-pattern-mvc/.

[8] *Centroid*. URL: https://turfjs.org/docs/#centroid.

[9] *Cross-platform native runtime for web apps*. URL: https://capacitorjs.com/.

[10] CubeProjects. *User Experience (UX), User Interface (UI)*. Dic. 2020. URL: https://cubeprojects.it/2020/12/28/user-experience-ux-user-interface-ui/.

[11] Megan Dennis, Susan May e Ruslan Bragin. *Native VS Cross-Platform Apps: Differences Between Native and Cross-Platform Apps*. Dic. 2018. URL: https://www.zeolearn.com/magazine/native-vs-cross-platform-apps-youll-be-the-winner.

[12] Pradip Dey et al. «Best Practices for Improving User Interface Design». In: *International Journal of Software Engineering & Applications* 10 (set. 2019), pp. 71–83. DOI: 10.5121/ijsea.2019.10505.

[13] Qian Dong e Waltenegus Dargie. «Evaluation of the reliability of RSSI for indoor localization». In: ago. 2012, pp. 1–6. ISBN: 978-1-4673-1290-5. DOI: 10.1109/ICWCUCA.2012.6402492.

[14] Daphne Economou et al. «Cultural applications for mobile devices: Issues and requirements for authoring tools and development platforms». In: *Mobile Computing and Communications Review* 12 (lug. 2008), pp. 18–33. DOI: 10.1145/1462141.1462145.

[15] www.extradigital.co.uk ExtraDigital Ltd. *User Experience (UX)*. URL: https://www.extradigital.co.uk/development/user-experience-ux.html.

[16] Michael Floater. *Bezier Curves and Surfaces*. Gen. 2015. DOI: 10.1007/978-3-540-70529-1_317.

[17] *GeoJSON*. URL: https://doc.arcgis.com/it/arcgis-online/reference/geojson.htm.

[18] *GeoJSON tutorial*. URL: https://leafletjs.com/examples/geojson/example.html.

[19] *Get Started Fast*. URL: https://cordova.apache.org/.

[20] Jeff Heaton. *The number of hidden layers*. Lug. 2020. URL: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[21] Larry Dubinski | President & CEO | The Franklin Institute. *Terracota warriors augmented reality experience at the Franklin Institute*. Lug. 2021. URL: https://www.wikitude.com/showcase/terracotta-warriors-augmented-reality-at-the-franklin-institute/.

[22] *Introduzione a JSON*. URL: https://www.json.org/json-it.html.

[23] Ionicframework. *Cross-Platform Mobile App Development*. URL: https://ionicframework.com/.

[24] Iotbymukund e Luqman Hakem says: *How to calculate distance from the RSSI value of the Ble Beacon*. Ott. 2016. URL: https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon/.

[25] *JavaScript with syntax for types*. URL: https://www.typescriptlang.org/.

[26] H. Joo. «A study on understanding of UI and UX, and understanding of design according to user interface change». In: *International Journal of Applied Engineering Research* 12 (gen. 2017), pp. 9931–9935.

[27] *Leaflet: an open-source JavaScript library for interactive maps*. URL: https://leafletjs.com/.

[28] MapBox. URL: https://geojson.io/.

[29] David Mascharka e Eric Manley. «LIPS: Learning Based Indoor Positioning System using mobile phone-based sensors». In: *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (gen. 2016). DOI: 10.1109/ccnc.2016.7444919. URL: http://dx.doi.org/10.1109/CCNC.2016.7444919.

[30] Hamid Mehmood, Nitin Tripathi e Taravudh Tipdecho. «Indoor Positioning System Using Artificial Neural Network». In: *Journal of Computer Science* 6 (ott. 2010). DOI: 10.3844/jcssp.2010.1219.1225.

[31] Ahasanun Nessa et al. «A Survey of Machine Learning for Indoor Positioning». In: *IEEE Access* 8 (2020), pp. 214945–214965. DOI: 10.1109/ACCESS.2020.3039271.

[32] *Nn-svg*. URL: https://alexlenail.me/NN-SVG/.

[33] *PROGETTO*. URL: https://www.vasariartexperience.it/progetto/.

[34] *PUB*. URL: https://docs.sewio.net/docs/los-vs-nlos-25593229.html.

[35] Britta Ricker e Robert Roth. «Mobile Maps and Responsive Design». In: *Geographic Information Science and Technology Body of Knowledge* 2018 (apr. 2018). DOI: 10.22224/gistbok/2018.2.5.

[36] Azhar Sehto. *Beacon Based Indoor Positioning System*. Dic. 2018. DOI: 10.13140/RG.2.2.17361.79200.

[37] *Shortest Path*. URL: https://turfjs.org/docs/#shortestPath.

[38] *Smartphone Market Share - Market Share*. URL: https://www.idc.com/promo/smartphone-market-share.

[39] Daniele Ponte | CTO | ComPart Multimedia srl. *Augmented reality in museums to identify artwork*. Mag. 2021. URL: https://www.wikitude.com/showcase/augmented-reality-in-museums-to-identify-artwork/.

[40] *TensorFlow*. URL: https://www.tensorflow.org/.

[41] *TURF*. URL: https://turfjs.org/.

[42] *User manual*. URL: https://www.wikitude.com/external/doc/documentation/studio/introduction.html#introduction-to-studio.

[43] *What Is GPS?* URL: https://www.geotab.com/blog/what-is-gps/.

[44] *Wikitude augmented reality: The world's leading cross-platform ar sdk*. Gen. 2021. URL: https://www.wikitude.com/.