Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze informatiche

SVILUPPO DELLA COMPONENTE CLIENT DI UN SISTEMA DI WAYFINDING

Tesi di Laurea in Programmazione ad Oggetti

Relatore:
Prof. Mirko Viroli

Presentata da:
Linda Farneti

II Sessione
Anno Accademico 2018/2019

Sommario

Il lavoro presentato in questa tesi tratta la progettazione e l'implementazione di un'applicazione mobile utilizzabile come wayfinding per ambienti esterni non mappati. Nato su richiesta da parte di un committente, il sistema si pone come obiettivo quello di agevolare l'utente nell'orientamento e negli spostamenti effettuati all'interno di spazi, anche di grandi dimensioni, per i quali non è possibile ricorrere all'utilizzo dei sistemi di navigazione già esistenti. Il software trova applicazione in contesti quali parchi divertimento o grandi eventi all'aperto e agisce da navigatore mostrando i punti di interesse presenti nell'area e permettendo di attivare la navigazione verso di essi. L'interfaccia grafica è stata realizzata in modo da mostrare un design simile ad applicazioni del settore già presenti sul mercato, questo allo scopo di privilegiare un approccio intuitivo all'ambiente ed offrire una piacevole esperienza d'uso anche da parte di utenti inesperti.

Indice

Capitolo 1

Introduzione

Questa tesi espone la progettazione e l'implementazione di un sistema di wayfinding per ambienti esterni non mappati ed è frutto della continuazione di uno studio iniziato durante le ore di tirocinio curricolare svolte in azienda. Il progetto, realmente commissionato dal proprietario di un resort di lusso, richiedeva la realizzazione di un'applicazione mobile per agevolare i propri ospiti negli spostamenti attraverso i numerosi ettari di terreno e i servizi messi a disposizione dal resort stesso. L'idea prende dunque ispirazione dai sistemi di navigazione già esistenti e si pone l'obiettivo di voler estendere il servizio offerto in modo da poter essere utilizzato anche in contesti di proprietà private e strade non ancora mappate che, in un contesto più generale, potrebbero essere rappresentati da parchi divertimenti, fiere ed eventi all'aperto anche di grandi dimensioni.

I requisiti di sistema, formulati dal committente e dall'azienda stessa, prevedevano l'implementazione di un'applicazione Android che potesse essere messa a disposizione dei clienti in modo che, ad installazione avvenuta, permettesse loro di visualizzare la mappa del luogo e i punti di interesse in essa presenti, corredati eventualmente delle relative descrizioni o di informazioni ritenute rilevanti, e di evidenziare il percorso minimo da intraprendere per raggiungerli, fornendo la possibilità di attivare la modalità di navigazione. La realizzazione di un progetto ex-novo di questo tipo ha comportato una fase di studio delle applicazioni simili già presenti sul mercato principalmente finalizzato alla creazione di interfacce grafiche che potessero garantire una User Experience adeguata. La produzione del sistema ha visto la collaborazione con un altro studente e il lavoro da svolgere è stato a tal proposito suddiviso in parte server, relativa alla realizzazione di un editor per la creazione e la modifica delle mappe da parte degli amministratori, e parte client, rappresentata dall'implementazione dell'applicazione per la visualizzazione delle mappe e la

navigazione utente. In questa tesi si espongono quindi gli studi relativi alla progettazione e all'implementazione del lato client del sistema, per la descrizione del progetto lato server si rimanda alla tesi "Sviluppo della componente server di un sistema di wayfinding"[1]

Il processo di sviluppo ha avuto come esito la realizzazione di un software funzionante che mediante interfacciamento con i sensori GPS presenti sul dispositivo fornisce all'utente la possibilità di visualizzare la mappa relativa alla sua posizione, permettendogli di visionare i punti di interesse presenti e di attivare la navigazione verso di essi. Per una corretta esecuzione dell'applicazione è richiesta la presenza di una connessione attiva con il lato server, necessaria per la ricezione dei dati relativi alle informazioni da mostrare e al percorso minimo individuato dal sistema. Lo sviluppo ha visto la concretizzazione di tutti i requisiti formulati in fase di analisi, ciononostante si ritiene che il software manchi ancora di alcune funzionalità ritenute necessarie per un'applicazione di questo tipo, quali per esempio le notifiche all'utente relative alle indicazioni da seguire e ai metri che lo separano dalle svolte e le modalità di ricalcolo del percorso.

Le attività di validazione e testing hanno rappresentato un valido supporto in fase di sviluppo permettendo di verificare il corretto funzionamento dell'applicazione, non hanno inoltre evidenziato la presenza di bug causa di interruzione o malfunzionamenti del sistema. Il progetto di tesi si è concluso positivamente, ha visto la realizzazione di un software funzionante che ha ricevuto l'approvazione anche da parte dei tutor aziendali.

La trattazione seguente si articola in sette capitoli: nel Capitolo 2 viene fornita una panoramica del background tecnico ritenuto utile alla comprensione del processo che ha portato allo sviluppo dell'applicazione e si introduce il contesto generale di wayfinding all'interno del quale si colloca questo progetto; il Capitolo 3 è inerente alla fase di analisi dei requisiti ed espone dettagliatamente le richieste formulate dal committente e dall'azienda in merito alle caratteristiche di sistema; nel Capitolo 4 si illustra la fase di design effettuata: nella prima parte si ha la descrizione delle componenti architetturali del sistema e delle relative funzionalità mentre la seconda presenta gli studi svolti sull'interfaccia grafica; il Capitolo 5 tratta la fase di sviluppo ed espone in modo dettagliato il funzionamento delle classi principali e l'implementazione dei metodi più rilevanti del progetto; il Capitolo 6 è relativo alla fase di testing effettuata sul sistema, descrive le modalità di validazione svolte e fornisce un breve commento relativo ai test implementati; infine, il Capitolo 7 comprende un giudizio generale sul progetto realizzato ed introduce alcune possibili funzionalità integrabili negli sviluppi futuri dell'applicazione.

[1] Bacchilega G., Sviluppo della componente server di un sistema di wayfinding. Tesi. 2018

Capitolo 2

Background

In questo capitolo si fornisce una panoramica delle nozioni tecniche necessarie alla comprensione della descrizione dello sviluppo del progetto di tesi.

La Sezione 1 tratta le tecnologie Web utilizzate per l'implementazione dell'applicazione mediante approfondimento dei linguaggi di programmazione, degli ambienti di sviluppo e degli strumenti di supporto utilizzati.

La Sezione 2 presenta le due librerie impiegate, fornendo per ognuna di essere una breve descrizione.

La Sezione 3 descrive lo scenario di riferimento del progetto.

2.1 Tecnologie Web

Si espongono di seguito le principali tecnologie Web utilizzate per la realizzazione del progetto.

2.1.1 JavaScript

JavaScript è un linguaggio di scripting lato client basato su una logica weakly typed e prototype-based e che costituisce, insieme ad HTML e a CSS, una delle 3 tecnologie fondamentali per la produzione di contenuti Web.

Originariamente sviluppato da Netscape con il nome Mochan, divenne LiveScript per poi assumere il nome JavaScript solo quando Netscape iniziò ad includere il supporto per Java nel proprio browser. La prima standardizzazione del linguaggio avvenne nel 1997, attualmente esso fa parte dello standard ECMAScript ed è uno standard ISO.

Nei linguaggi di scripting lato client l'interpretazione del codice è demandata non al server ma al client, in ambito Web rappresentato dal browser, e questa caratteristica rende possibile scrivere veri e propri script in JavaScript attraverso i quali l'utente può interagire con il browser. Poichè compito del server è solamente quello di trasmettere il sorgente, l'esecuzione di programmi complessi nel caso dei linguaggi lato client non comporta alcun sovraccarico per la capacità computazionale del server stesso e perciò risulta essere agevolata.

Di contro, questo approccio si traduce in tempi di esecuzione elevati e nell'impossibilità di eseguire direttamente operazioni che coinvolgono il server, quali per esempio la scrittura di dati all'interno del database.

Altra caratteristica di JavaScript è quella di essere un linguaggio weakly typed: il fatto che il controllo del tipo di variabile non avvenga in fase di compilazione bensì a runtime conferisce estrema flessibilità al linguaggio, per esempio permettendo il cambio del tipo di variabile dinamicamente durante l'esecuzione di un programma, ma allo stesso tempo può portare alla generazione di codice ambiguo e causare errori individuabili solamente in fase di esecuzione.

Gli approcci alla programmazione offerti da JavaScript sono sia quello procedurale e sia quello orientato agli oggetti, il fatto che si tratti però di un linguaggio debolmente tipizzato e che ignori i concetti di classe, di interfaccia e di polimorfismo, lo porta ad essere piuttosto limitato in questa seconda modalità.

2.1.2 Typescript

L'esigenza di una maggior sicurezza e robustezza spinse Microsoft ad implementare un'estensione di JavaScript che includesse il supporto per il controllo statico dei tipi e che mettesse a disposizione funzionalità aggiuntive per la scrittura di applicazioni complesse: nel 2012 nacque quindi TypeScript.

Si tratta di un linguaggio di programmazione libero ed open source costituito da un super set di JavaScript del quale estende sintassi e semantica aggiungendo concetti quali classi, interfacce e moduli e questo permette l'utilizzo di codice JavaScript esistente, librerie comprese, senza dover effettuare modifiche di alcun tipo.

La compilazione di un codice TypeScript porta alla realizzazione di codice JavaScript eseguibile su qualsiasi browser, in Node.js o in un qualsiasi motore JavaScript che supporti ECMAScript 3 (o più recente) e viene effetuata grazie all'utilizzo di un transpiler, cioè un apposito tool necessario per convertire il codice sorgente da un linguaggio ad un altro.

TypeScript mette a disposizione dello sviluppatore un sistema opzionale di annotazioni dei tipi che consente di specificare esplicitamente il tipo di variabile, il controllo statico, il refactoring del codice e altre pratiche che conferiscono una maggior comprensibilità e leggibilità; tutte queste caratteristiche rappresentano notevoli vantaggi per quanto riguarda la realizzazione di applicazioni complesse.

L'editor utilizzato per la programmazione JavaScript e TypeScript relativa al progetto è Visual Studio Code.

Visual Studio Code [1] Annunciato da Microsoft il 29 aprile 2015, Visual Studio Code è un editor di codice leggero ma potente con supporto integrato per JavaScript, TypeScript e Node.js ma che comprende anche una serie di estensioni per altri linguaggi quali C++, C#, Java, Python e PHP.

Si tratta di un software libero sviluppato per Windows, Linux e macOS che si basa su Electron, framework con il quale è possibile sviluppare applicazioni Node.js, ed include il supporto per il debugging, un controllo per Git integrato, Syntax highlighting, IntelliSense, Snippet e refactoring del codice.

2.1.3 Sass

Sass[2] (Syntactically Awesome StyleSheets) è un'estensione del linguaggio CSS di supporto alla realizzazione di file CSS ottimizzati che mette a disposizione funzionalità aggiuntive quali variabili, mixin, nesting, ereditarietà e altro ancora.

Poichè le features offerte da Sass non sono direttamente interpretabili dal browser, occorre ricorrere all'utilizzo di un compilatore che, prendendo in ingresso uno o più file Sass, ne restituisce l'equivalente in CSS, direttamente utilizzabile nel proprio sito web, e ciò rende possibile realizzare fogli di stile con una forma più semplice ma anche più potente e completa rispetto a CSS.

2.1.4 Client API

In ambito informatico, le API (Application Programming Interface) costituiscono un insieme di procedure necessarie allo svolgimento di un determinato compito.

Si tratta di un valido strumento di supporto messo a disposizione dello sviluppatore che gli consente di effettuare un ampio riuso di codice e di ottenere un'astrazione più ad

[1] `https://code.visualstudio.com/`
[2] `https://sass-lang.com/`

alto livello, per esempio tra hardware e software ma anche tra software a basso e ad alto livello.

Nelle architetture Client-Server, ci si serve di client API per fare in modo che il client possa richiedere uno specifico servizio al server mediante l'invio di un apposito messaggio.

In ambito Web le applicazioni vengono realizzate per funzionare su un browser o un server Web e, trattandosi di un concetto solitamente limitato al lato client, per utilizzare le client API si implementano apposite interfacce di programmazione, chiamate Web API, che nascondono i dettagli implementativi della piattaforma sulla quale girano.

Una Web API:

- lato client: è un'interfaccia di programmazione realizzata per estendere le funzionalità all'interno di un browser web o di un altro client HTTP;
- lato server: è un'interfaccia di programmazione costituita da uno o più endpoint esposti pubblicamente e da un sistema di messaggi richiesta-risposta definito che viene mostrato tramite Web, normalmente mediante una base HTTP Server Web.

2.1.5 Web App

Il termine Web application identifica un ibrido tra un tradizionale sito Web e un'app mobile e in particolare si tratta di un tipo di applicazione che non risiede direttamente sulle macchine che la utilizzano bensì su Server remoti.

Sostanzialmente, una Web app non è altro che un programma che viene scritto per essere eseguito su Internet o su una rete locate e che viene distribuito agli utilizzatori in modo che essi ne possano usufruire senza però disporre del codice stesso.

Per creare un'app mobile tradizionale con due diversi target, per esempio iOS e Android, si rende necessaria la realizzazione di due sviluppi nativi, poichè invece le Web application possono essere utilizzate su qualsiasi dispositivo dotato di connessione Internet, per implementare la medesima funzionalità utilizzando questa tecnologia è sufficiente un unico sviluppo e questo si traduce in vantaggi economici e dal punto di vista delle tempistiche di realizzazione.

Le Web app vengono sviluppate a partire dal linguaggio Web lato client e questo permette di gestire l'interfaccia utente di ogni singola pagina, inoltre esse emulano la navigazione delle app native con l'obiettivo di fornire una navigazione più fluida possibile ed un'ottima User Experience e trattandosi di pagine Web, non causano problemi riguardanti lo spazio di archiviazione sullo smartphone.

2.1.6 Bitbucket

Bitbucket[3] è un servizio di hosting Web-based realizzato in Python che viene utilizzato per progetti basati su sistemi di controllo versione Mercurial o Git e che è stato di supporto all'intero sviluppo del progetto di tesi.

Grazie a questa soluzione è possibile scegliere se realizzare repository online pubblici e privati, in base a se devono essere utilizzati per lo sviluppo di software open source oppure per progetti che devono rimanere privati ad un team di persone, e permette non solo di gestire codice Git ma offre anche un supporto ai team fornendo loro un luogo virtuale per collaborare sul codice, implementarlo ed eseguire testing.

2.2 Librerie

Di seguito, si fornisce una breve descrizione delle due principali librerie utilizzate nel progetto di tesi: Leaflet e JQuery.

2.2.1 Leaflet

Leaflet[4] è una libreria JavaScript open source di supporto alla realizzazione e alla gestione di mappe interattive ottimizzate per i dispositivi mobile che permette di mostrare punti di interesse, linee, aree o strutture dati ma anche livelli interattivi su mappe costruite a tasselli.
Mettendo a disposizione gran parte delle funzionalità richieste per lo scopo, Leaflet è considerata la principale libreria JavaScript nel suo ambito, è estensibile con vari plugin ed è stata progettata in modo semplice, con particolare attenzione alle prestazioni e all'usabilità.

2.2.2 JQuery

JQuery[5] è una libreria JavaScript per applicazioni web, nata con l'obiettivo di semplificare la selezione, manipolazione, gestione degli eventi e l'animazione di elementi DOM (Document Object Model) in pagine HTML, nonchè per implementare funzionalità AJAX.

[3] `https://bitbucket.org/`
[4] `https://leafletjs.com/`
[5] `https://jquery.com/`

Si tratta della libreria JavaScript più utilizzata su Internet e i principi sui quali si basa sono: separazione di JavaScript e HTML, brevità e chiarezza, eliminazione di incompatibilità cross-browser ed estensibilità.

2.3 Wayfinding

Il termine wayfinding apparve per la prima volta negli anni '60 nel libro "The image of the city" di Kevin Lynch come parola che esprimeva il legame fra il cittadino e l'ambiente urbano. Non potendo tradurlo letteralmente in modo corretto, Salvatore Zingale in un'intervista[6] usò l'espressione "cognizione spaziale" per indicare le modalità con cui le persone si orientano all'interno di uno spazio fisico navigando da un luogo ad un altro.

Le fasi che caratterizzano il wayfinding sono quattro:

- orientamento: il soggetto cerca di determinare la propria posizione in relazione agli oggetti vicini e alla destinazione desiderata;
- decisione del percorso: si seleziona un corso di direzione verso la destinazione;
- monitoraggio del percorso: questa fase prevede la verifica del percorso per assicurarsi che quello selezionato sia il giusto tragitto diretto verso la destinazione scelta;
- riconoscimento della destinazione: il percorso si considera terminato quando viene riconosciuta la destinazione finale.

Uno dei campi di riferimento per il progresso tecnologico degli ultimi anni è proprio quello del wayfinding: sono numerose le applicazioni di navigazione outdoor che permettono agli utenti di muoversi efficacemente all'interno e all'esterno dei centri urbani, prima fra tutte Google Maps[7], una piattaforma che consente di orientarsi ovunque nel mondo, indipendentemente dal mezzo utilizzato. Fra le altre applicazioni che offrono questo tipo di servizio c'è Waze[8], recentemente acquisita da Google, che, a differenza di Google Maps, basa il suo funzionamento sulla cooperazione tra gli utenti che svolgono un'attività di crowdsourcing segnalando in prima persona le informazioni sul traffico e sulle condizioni stradali in tempo reale tramite l'applicazione stessa.

Diverso è il caso di OpenStreetMap[9] che invece ha lo scopo di raccogliere tutti i dati

[6] http://www.salvatorezingale.it/download/ZINGALE-Wayfinding-e-cognizione-spaziale.pdf

[7] https://www.google.com/maps

[8] https://www.waze.com/it/

[9] https://www.openstreetmap.org/

geografici mondiali come base per la realizzazione di mappe e cartografie liberamente utilizzabili, modificabili e controllabili da chiunque. Di solito nel campo della cartografia si utilizzano le ortofoto, cioè fotografie geometricamente corrette e georeferenziate alle quali vengono poi successivamente aggiunte informazioni come rilievi topografici che permettono di misurare le distanze reali tra i vari punti rappresentati. Nel caso di OpenStreetMap, le ortofoto vengono donate da aziende ed enti governativi mentre i rilievi sul territorio vengono fatti da parte di volontari che viaggiano all'interno delle zone da mappare. La libreria Leaflet utilizzata per l'implementazione del software oggetto di questo trattato si serve proprio delle mappe base messe a disposizione da OpenStreetMap.

Il progetto di tesi parte dunque dallo studio delle applicazioni citate e si pone come obiettivo la realizzazione di un servizio simile ma utilizzabile in contesti differenti, in particolare nel caso di ambienti di grandi dimensioni, quali parchi divertimenti, fiere all'aperto o eventi, non mappati precedentemente.

2.3.1 Algoritmo di Dijsktra

L'algoritmo di Dijkstra fu inventato nel 1956 dall'informatico olandese Edsger Dijsktra e permette di cercare i cammini minimi in un grafo, ordinato oppure non ordinato, ciclico e con pesi non negativi sugli archi.

Gli ambiti della vita reale in cui questo algoritmo trova applicazione sono molteplici, per esempio consente di ottimizzare la realizzazione di reti stradali e ferroviare ma anche di organizzare e valutare percorsi runtime nel campo della robotica.

L'algoritmo visita i nodi del grafo sul quale viene applicato in un modo simile ad una ricerca in ampiezza o in profondità e li suddivide in due insiemi contenenti rispettivamente i nodi già etichettati e quelli ancora da etichettare. Il funzionamento generale è il seguente:

- ogni nodo all'inizio ha potenziale $+\infty$;
- il nodo di partenza ha potenziale 0 (cioè dista zero da se stesso);
- ogni volta che si sceglie il nodo con potenziale minore e lo si rende definitivo si aggiornano i nodi adiacenti;
- il potenziale di un nodo è dato dal potenziale del nodo precedente sommato al costo del collegamento;
- i potenziali dei nodi resi definitivi non vengono più aggiornati ed indicano la distanza di quello specifico nodo da quello di partenza;

– trattandosi di un problema di ricerca del percorso minimo, aggiornando il potenziale di un nodo si lascia quello minore.

Il tempo di esecuzione di questo algoritmo può essere espresso in funzione del numero |V| dei vertici e |E| degli archi appartenenti al grafo sul quale viene eseguito ed equivale a

$$O(|E| + |V^2|)$$

che, nel caso di un grafo generico, può essere approssimato a

$$O(|V^2|)$$

e questo significa che è proporzionale al quadrato dei vertici del grafo.

Capitolo 3

Analisi dei requisiti

Obiettivo del progetto di tesi è la realizzazione di un'applicazione Android che permetta all'utente di orientarsi all'interno di uno spazio outdoor, non mappato da satelliti o simili.

Dopo aver completato l'installazione dell'app sullo smartphone, l'utente potrà utilizzarla per ricercare i luoghi che giudica interessanti, sia attraverso le icone sulla mappa e sia attraverso una barra di ricerca, e attivare la navigazione verso di essi. Il percorso verrà calcolato utilizzando come punto di partenza la posizione stessa dell'utente in quell'istante e come destinazione il luogo da lui scelto, che dovrà necessariamente corrispondere alle coordinate di un punto di interesse.

Dopo aver attivato la navigazione, sulla mappa verrà segnalato il percorso da seguire, dotato di un apposito cursore posto in corrispondenza della posizione dell'utente, in modo che egli possa in ogni momento consultarlo per capire dove si trova e dove deve dirigersi.

Per il momento, non è richiesta l'implementazione di un sistema che comunichi all'utente come muoversi, per esempio dopo quanti metri girare a destra, ma non si esclude che possa essere argomento di sviluppi futuri del progetto.

3.1 Requisiti funzionali

Visualizzazione della mappa. L'applicazione deve consentire la visualizzazione di mappe graficamente sensate, per esempio in sovraimpressione rispetto ad Open Street Maps[1] o Google Maps, che permettano di prendere visione dei punti di interesse presenti nell'intorno dell'utente.

[1] `https://www.openstreetmap.org/`

Le destinazioni devono essere segnalate (in modo chiaro ed efficace) per l'utente finale, per esempio tramite l'apposizione di icone, con la possibilità di visualizzarne il nome e l'eventuale descrizione associata e di attivare la navigazione verso di esse.

Per garantire una migliore esperienza d'uso, è inoltre richiesto che la mappa sia navigabile e che sia possibile effettuare su di essa le operazioni di zoom-in e zoom-out.

Le mappe da visualizzare devono essere ottenute mediante comunicazione con il Server, realizzata con l'implementazione di apposite richieste API.

Localizzazione dell'utente. Il corretto funzionamento si basa sulla capacità da parte del sistema di localizzare l'utente durante la navigazione, è perciò necessario prevedere l'integrazione del progetto con il rilevatore del segnale GPS presente sul dispositivo mobile.

La posizione dell'utente deve essere aggiornata ad ogni spostamento e segnalata con l'inserimento di un'apposita icona, questo accorgimento darà origine ad un sistema reattivo caratterizzato da un cursore che si muoverà sulla mappa in relazione agli spostamenti reali compiuti dall'utente stesso.

Infine, deve essere prevista una funzionalità che permetta all'utilizzatore del sistema di localizzare la propria posizione anche con la modalità di navigazione disabilitata in modo da permettergli di capire immediatamente dove si trova al lancio dell'applicazione.

Motore di ricerca. Il sistema dovrà essere dotato di una barra per la ricerca in modo che, digitanto il nome di un punto di interesse o parte di esso, si attivi una funzionalità che analizzi i vari punti di interesse presenti sulla mappa e fornisca come risultato solo quelli per i quali è stato trovato il matching corrispondente.

La ricerca potrà essere effettuata considerando unicamente il nome proprio dei punti di interesse e dovrà essere svolta senza fare distinzioni tra lettere maiuscole e minuscole, in modo da diminuire le probabilità di errore dovute ad un errato inserimento da parte dell'utente. L'elenco dei risultati ottenuti dovrà prevedere la visualizzazione del nome di ogni punto di interesse trovato, eventualmente corredato dalla distanza in metri (l'utente ideale del sistema è una persona che si muove a piedi) da percorrere per raggiungerli.

Per una migliore esperienza utente, alla ricerca potrebbe inoltre essere associata un'azione che nasconda dalla mappa tutti i punti di interesse non appartenenti all'e-

lenco dei risultati, in modo da conferire immediatezza ed una maggior efficacia alla visualizzazione utente nell'individuazione delle varie località trovate.

Client API. La comunicazione con il Server, necessaria per esempio per la restituzione della mappa da visualizzare o del percorso da evidenziare, deve essere realizzata con l'implementazione di chiamate API lato Client.

3.2 Requisiti non funzionali

Affidabilità. Si richiede l'implementazione di un sistema affidabile, che limiti quanto più possibile le situazioni critiche di malfunzionamento. Il processo di sviluppo, sulla base di questo particolare requisito, potrebbe includere una serie di accorgimenti che permetterebbero al sistema Client-side di funzionare anche in casi di irraggiungibilità o malfunzionamento del server.

Non disponendo di un collegamento diretto con il database nel quale sono memorizzate le mappe, il client dovrà necessariamente affidarsi al server per ottenerle, l'implementazione lato client dello stesso algoritmo di ricerca del percorso minimo già presente sul server, garantirebbe però all'applicazione di funzionare anche in casi in cui, dopo aver ricevuto le mappe da visualizzare, il server diventasse irraggiungibile.

User Experience. Il sistema deve essere caratterizzato da una buona User Experience che garantisca una piacevole usabilità dell'applicazione anche da parte dell'utente non esperto nonchè un'ottimale fruizione dei contenuti e delle funzionalità. A tal proposito, si richiede la realizzazione di un'interfaccia grafica che sia quanto più possibile efficace, minimale e che permetta una comprensione immediata da parte dell'utilizzatore di ciò che viene mostrato e di come agire per ottenere la funzionalità desiderata.

L'intera schermata deve essere studiata in modo che il posizionamento degli elementi grafici in essa presenti si adatti agevolmente alle diverse dimensioni degli smartphone presenti in commercio, deve inoltre risultare idonea ad un utilizzo con orientamento sia verticale e sia orizzontale.

Uso efficiente delle risorse. Allo scopo di limitare quanto più possibile i casi di impossibilità di utilizzo dell'applicazione su dispositivi con capacità di archiviazione limitata o ridotta, l'app mobile deve essere realizzata prestando particolare attenzione alla

memoria necessaria per l'installazione. Inoltre, poichè spesso uno spazio di archiviazione libero insufficiente si traduce in una potenza di calcolo ridotta, essa dovrà richiedere un utilizzo di risorse limitato, in modo da non necessitare di lunghi tempi di attesa per il caricamento e il funzionamento.

Portabilità. Nonostante i requisiti del progetto attualmente commissionato prevedano l'implementazione di un'applicazione per dispositivi Android, va prevista la possibilità che questa stessa applicazione debba essere sviluppata in futuro anche per sistemi iOS. A tal proposito, si richiede che la fase di design venga realizzata in modo da agevolare questo probabile successivo sviluppo.

3.3 Requisiti tecnologici

Implementazione su Android. L'implementazione richiesta prevede il funzionamento su dispositivi con Sistema Operativo Android 6 o superiori (versione 23 delle API)

3.4 Requisiti dell'implementazione

Servizio mock. In previsione del forte disaccoppiamento che vi sarà nello sviluppo delle parti client e server, è richiesta l'implementazione di un sistema di chiamate mock al server allo scopo di simulare l'interazione con esso per la ricezione dei dati. Questo servizio rappresenterà un valido supporto alla fase di testing, permettendo allo sviluppatore del sistema lato client, argomento di questa tesi, di poterne verificare la corretta implementazione indipendentemente dalla presenza del server.

I dati mock forniti non dovranno essere necessariamente verosimili, ma si richiede che siano tali da poter permettere un testing completo del codice prodotto, andando a coprire il maggior numero di situazioni diverse.

3.5 Diagramma dei casi d'uso

Lo studio dei requisiti di sistema formulati ha permesso la creazione del seguente diagramma dei casi d'uso, utilizzato per verificare con il committente che non fossero sorte incomprensione durante la fase di analisi dei requisiti.

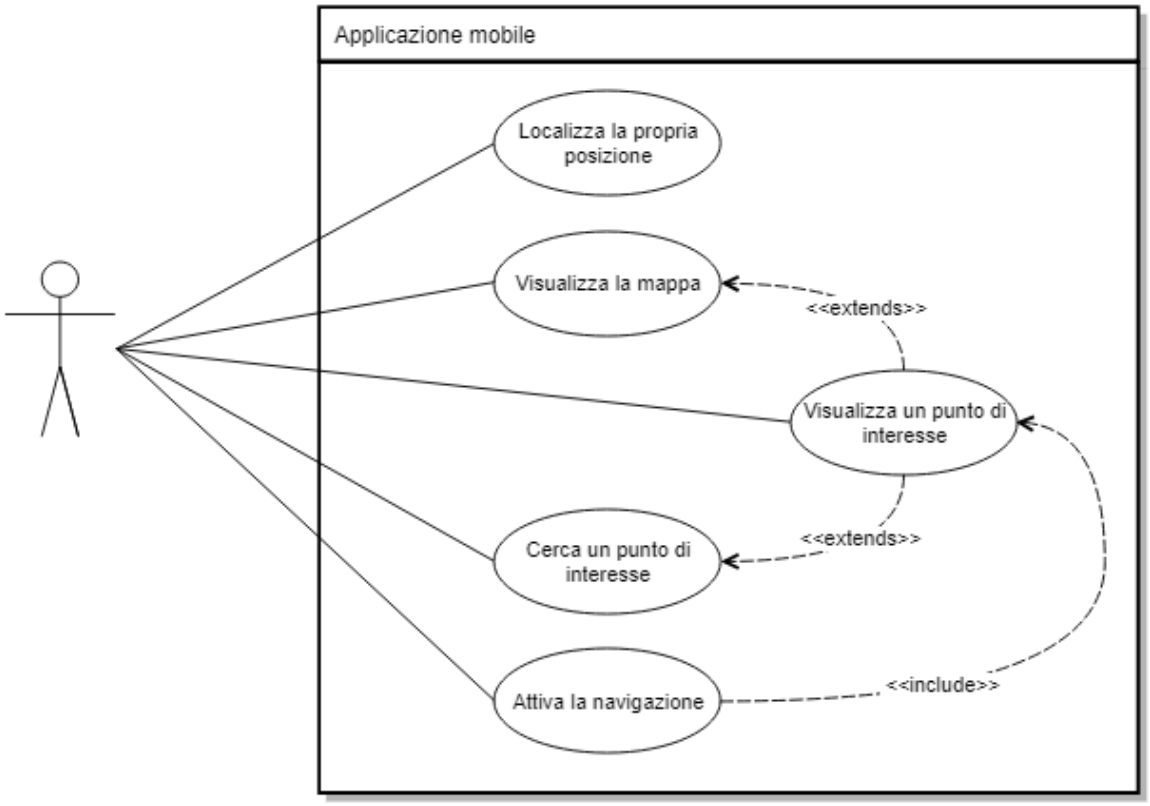


Figura 3.1: Sistema - Diagramma dei casi d'uso

Capitolo 4

Design e progettazione

Obiettivo di questo capitolo è la descrizione della fase di design e progettazione effettuata sul progetto.

Nella Sezione 1 viene presentato il sistema nella sua interezza, si fornisce una spiegazione in merito alla suddivisione in lato client e lato server e vengono presentate le funzionalità implementate dai due componenti.

Nella Sezione 2 viene descritto il pattern architetturale adottato per l'implementazione lato client, vi è la spiegazione del design generale del progetto e la descrizione delle funzionalità dei componenti principali.

Nella Sezione 3 si espone lo studio effettuato sull'interfaccia grafica mediante spiegazione dettagliata dei mockup prodotti.

4.1 Componenti architetturali del sistema

Per la realizzazione di un progetto sulla base delle specifiche formulate, si è resa necessaria la suddivisione del sistema in macro componenti e, a tal proposito, si è scelto di adottare quella già piuttosto evidente basata sulle differenti entità fisiche sulle quali dovrà avvenire l'esecuzione.

All'interno del sistema generale, è stata quindi individuata un'architettura composta da due componenti con distinti ruoli e funzionalità:

– client: è il software che dovrà essere eseguito sui dispositivi con sistema operativo Android, quali smartphone e tablet, e rappresenta l'argomento di questa tesi. Il lato client del sistema dovrà provvedere alla visualizzazione delle mappe ricevute dal server, fornire la possibilità all'utente di prendere visione dei punti di interesse

presenti nel suo intorno e guidarlo nella navigazione verso uno di essi, sarà quindi necessario prevedere l'interfacciamento con i sensori GPS presenti sul dispositivo per permetterne la localizzazione;

– server: è il componente che dovrà fornire al client, su sua esplicita richiesta, i dati di cui necessita per lo svolgimento delle sue funzioni. Si tratta della parte amministrativa del sistema che dovrà mettere a disposizione dell'utente, in questo caso rappresentato da un amministratore, un editor per la realizzazione di mappe personalizzate dotate di strade e punti di interesse. Andranno inoltre previsti collegamenti con un database, necessario per la realizzazione di una storicizzazione dei dati creati dagli amministratori, e con un server di supporto all'implementazione delle chiamate relative al popolamento e alla modifica delle mappe create.

La comunicazione fra i componenti client e server del sistema verrà realizzata mediante implementazione di API.

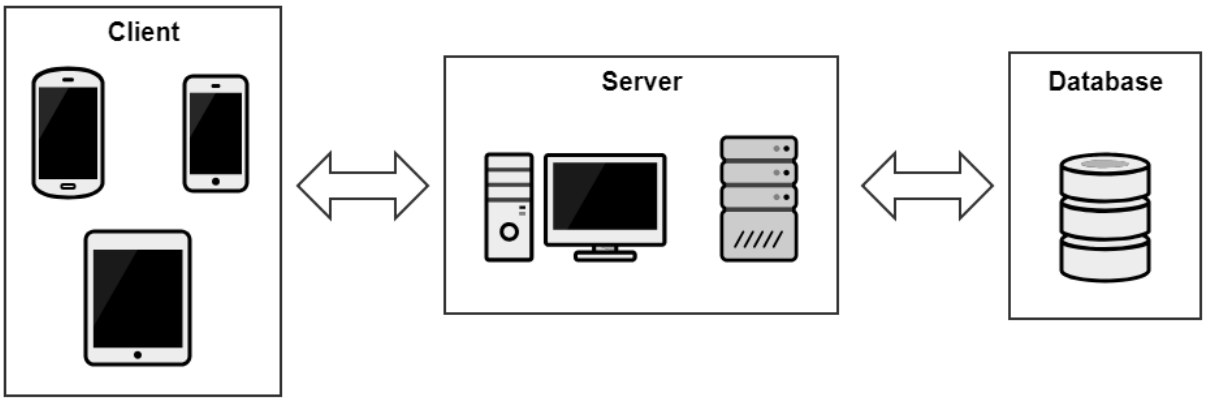


Figura 4.1: Architettura generale del sistema

4.2 Design architetturale

La progettazione architetturale del sistema lato client ha visto l'individuazione di 4 componenti distinti:

– Model: contiene le interfacce e le strutture dati relative agli elementi da memorizzare;

– View: implementa i metodi necessari per il disegno e la modifica della mappa. Questo componente racchiude tutta la parte relativa alla presentazione e questo significa che,

nel caso in cui si volesse modificare il look-and-feel del sito, sarebbe possibile farlo agendo solamente su questa parte;

- Gestore dei sensori: permette l'interfacciamento del sistema con il sensore GPS presente sul dispositivo e notifica i dati ricevuti al Controller;
- Controller: comprende tutta e sola la logica del business-logic, esso riceve i comandi dell'utente e li attua modificando lo stato del componente relativo alla View

All'interno di questa architettura, i componenti denominati Model, Controller e View rispecchiano l'implementazione del più generale pattern architetturale MVC e ne implementano le funzionalità e le modalità di comunicazione.

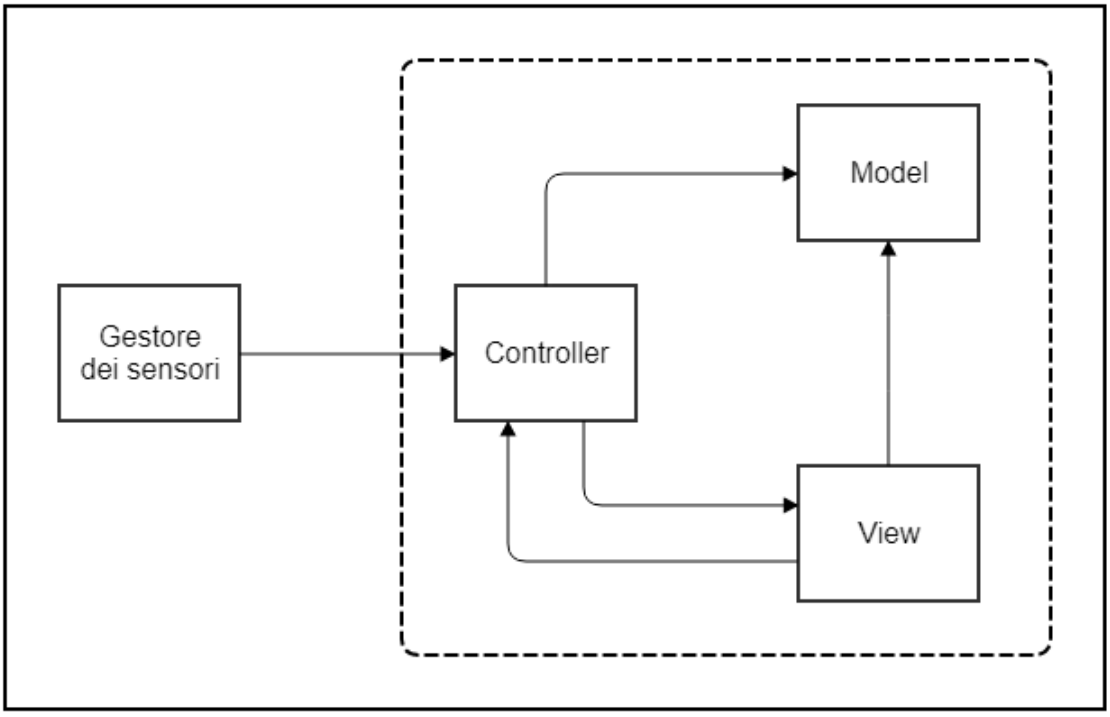


Figura 4.2: Architettura generale del sistema lato Client

Di seguito si presentano le funzionalità proprie di ciascun componente dell'architettura.

4.2.1 Model

Lato client, non è prevista l'implementazione di un database locale all'applicazione e la ricezione dei dati avviene solo mediante connessione con il server, le funzionalità relative al

Model si limitano dunque alla dichiarazione delle strutture dati necessarie al funzionamento degli altri componenti.

A livello grafico, si è deciso di assimilare la mappa da realizzare ad un grafo in cui gli archi rappresentano le strade e i nodi possiedono un determinato indice e possono essere associati ai relativi punti di interesse. Sulla base di questa rappresentazione, i dati da richiedere al server saranno necessariamente: la mappa da visualizzare, in particolare le coordinate di latitudine e longitudine di riferimento, le coordinate dei nodi del grafo e i relativi archi che li congiungono e le informazioni in merito ai punti di interesse da disegnare, in primo luogo l'identificatore del nodo corrispondente e l'icona ad essi associata.

Il Model consisterà unicamente nella dichiarazione delle strutture dati relative alla memorizzazione di queste informazioni, corredate degli attributi necessari per la realizzazione dei corrispondenti elementi grafici.

4.2.2 View

La View dovrà contenere l'implementazione di tutti i metodi necessari alla visualizzazione e manipolazione della mappa.

In questo componente sarà gestita la creazione di tutti gli elementi grafici mostrati dall'applicazione (mappa, nodi, strade e punti di interesse) ma anche le funzionalità relative alla modifica della mappa, quali per esempio la segnalazione del percorso trovato e la visualizzazione/cancellazione delle icone sulla base dei risultati della ricerca effettuata dall'utente.

In merito alla caratteristica di manutenibilità del codice, è stato deciso di mantenere in questo componente ogni riferimento e collegamento con la libreria grafica. Questa scelta, oltre a garantire una maggior chiarezza del codice prodotto, rappresenterà un vantaggio soprattutto per la realizzazione degli sviluppi futuri del progetto per i quali sarà richiesto un ampliamento delle funzionalità e dei requisiti, la libreria grafica individuata potrebbe infatti non rappresentare più la scelta migliore ma con questo tipo di implementazione potrà essere agevolmente sostituita andando ad agire unicamente su questa classe e lasciando inalterato il resto del progetto.

4.2.3 Controller

Il Controller dovrà implementare gran parte delle funzionalità del progetto.Come scelta implementativa, si è deciso di organizzare il tutto realizzando classi separate in cui ver-

ranno raggruppati i metodi relativi a specifiche e distinte funzionalità, questo allo scopo di realizzare un codice modulare, ben organizzato e caratterizzato da un alto livello di chiarezza e leggibilità.

Le principali funzionalità da implementare all'interno di questo componente saranno:

- la gestione dell'interazione dell'utente con l'interfaccia grafica del sistema, l'intercettazione dei click effettuati e l'attivazione delle relative funzionalità;
- l'inizializzazione del sistema e l'instanziazione di tutte le classi necessarie al suo funzionamento;
- la gestione della connessione con il Server, l'implementazione delle richiesta da effettuare e il passaggio dei dati ricevuti alle classi corrispondenti.

4.2.4 Gestore dei sensori

Il componente relativo alla gestione dei sensori avrà come unica funzionalità quella di interfacciarsi con i sensori presenti sul dispositivo mobile per poter trasferire le informazioni rilevate al sistema e permetterne dunque il funzionamento.

I dati ricevuti dal sensore GPS dovranno essere notificati al Controller che provvederà ad aggiornare le coordinate utente memorizzate dal sistema con quelle relative all'ultima posizione rilevata.

4.3 Design dell'interfaccia

Per la progettazione dell'interfaccia grafica è stato svolto un attento studio delle applicazioni del settore già presenti sul mercato, conclusosi con la realizzazione di 4 mockup.

Considerati i requisiti, si è cercato di dare all'applicazione un design tale da essere facilmente compreso anche dall'utente inesperto, essenziale ma al tempo stesso autoesplicativo.

Il sistema lato Client si compone di un'unica schermata che viene modificata in relazione alle azioni compiute dall'utente.

Si presentano di seguito i mockup realizzati, gli elementi grafici utilizzati e le relative funzionalità.

Figura 4.3: Schermata iniziale dell'applicazione

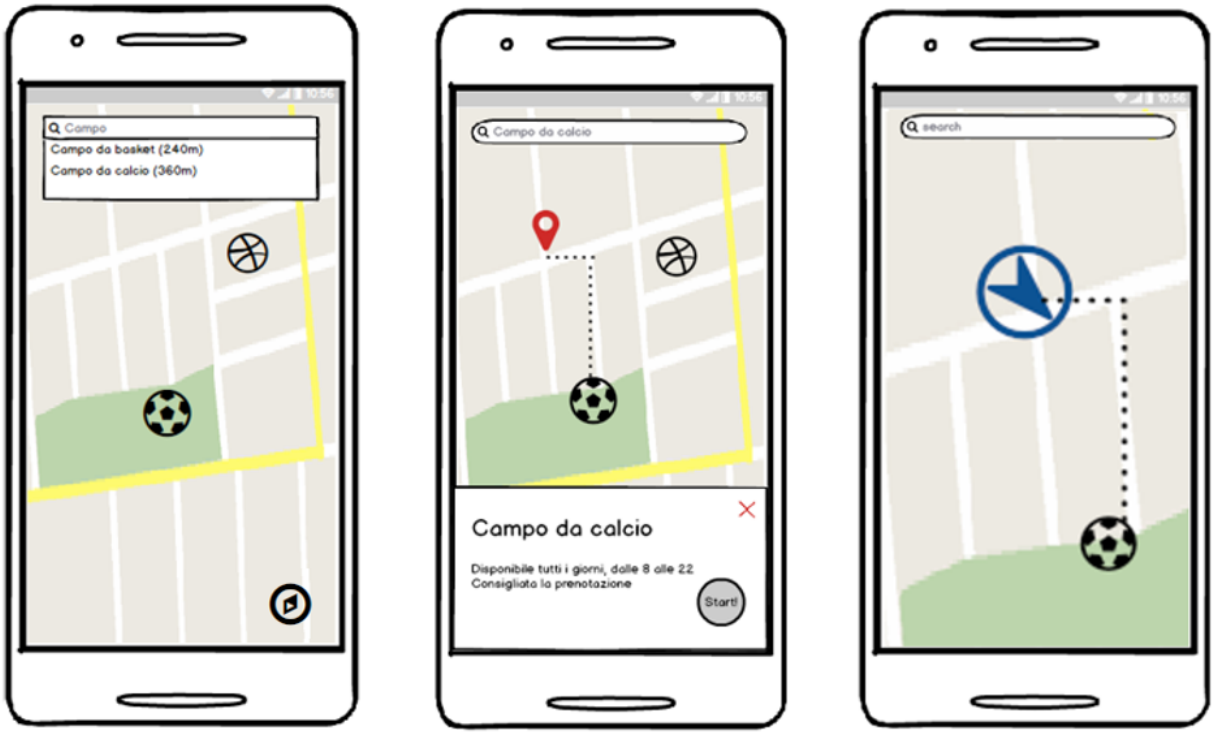
4.3.1 Avvio dell'applicazione

All'avvio dell'applicazione avverrà il caricamento di una schermata (Figura 4.3) caratterizzata da una mappa estesa a tutte le dimensioni dello schermo del dispositivo e, sovrapposti ad essa, 3 elementi grafici:

- barra di ricerca: posizionata nella parte alta dello schermo e centrata, permetterà all'utente di effettuare ricerche in merito ai punti di interesse da visualizzare sulla mappa;
- punti di interesse: rappresentati da icone posizionate sulla mappa, saranno cliccabili e consentiranno all'utente di attivare la navigazione verso una specifica destinazione;
- pulsante di localizzazione: situato nella parte bassa della schermata e allineato a destra, se cliccato, permetterà all'utente di identificare immediatamente la propria

posizione sulla mappa facendo apparire l'apposito cursore alle coordinate corrispondenti.

4.3.2 Funzionamento dell'applicazione



(a) Ricerca di un punto di interesse

(b) Selezione di un punto di interesse

(c) Navigazione verso un punto di interesse

Figura 4.4: Funzionamento dell'applicazione

Come richiesto, il sistema dovrà offrire la possibilità di effettuare una ricerca sui punti di interesse mostrati sulla mappa. In Figura (a) si mostra un esempio di tale funzionalità, in particolare ciò che si otterrebbe digitando la parola "campo" nella barra di ricerca e premendo il pulsante contrassegnato dall'icona raffigurante una lente d'ingrandimento. Il click produce come risultato quello di far "scomparire" dalla mappa tutte le icone relative alle destinazioni non inerenti alla ricerca effettuata, in questo caso si mostrano solo quelle del "campo da basket" e del "campo da tennis", e contemporaneamente i risultati trovati vengono mostrati mediante apparizione di un pannello sottostante alla barra di ricerca, in cui al nome del punto di interesse è associata la corrispondente distanza che lo separa dall'utente. Trattandosi di un'applicazione sviluppata principalmente per essere un sistema

di navigazione dovrà inoltre essere prevista la possibilità di selezionare la destinazione di proprio interesse, e ciò attuabile cliccando nel pannello contenente i risultati della ricerca ma anche direttamente sull'icona presente sulla mappa, il risultato sarà il medesimo.

In Figura (b) si ha il mockup relativo a ciò che si dovrà ottenere effettuando un click su un punto di interesse presente sulla mappa, in particolare vengono mostrate:

- l'apertura di un pannello nella parte bassa dello schermo contenente il nome della destinazione selezionata, la descrizione e, posizionato in basso a destra, un pulsante contrassegnato dalla parola "Start" che permetterà di avviare la modalità navigazione;
- la visualizzazione del percorso da intraprendere per raggiungere il punto di interesse cliccato, visualizzato mediante opportuna segnalazione delle strade coinvolte (in figura le strade sono contrassegnate da un tratteggio di colore differente rispetto a quello della strada stessa).

Infine, in Figura (c) viene mostrato l'avvio della modalità navigazione: è prevista la scomparsa del pannello descrittivo e il ritorno ad una schermata interamente occupata dalla mappa, che in questo caso avrà subito un leggero zoom centrato sulla posizione dell'utente. Durante la navigazione sarà possibile attivare/disattivare la modalità localizzazione mediante il pulsante posizionato in basso a destra: disattivata, permetterà all'utente di scorrere la mappa pur continuando a localizzarlo mentre si muove, attivandola invece si avrà una mappa la cui visuale rimarrà centrata sull'utente "seguendolo" durante la navigazione.

Capitolo 5

Sviluppo del sistema

Questo capitolo espone le scelte effettuate in merito allo sviluppo del sistema:

La Sezione 1 è relativa all'organizzazione e all'implementazione generale del progetto, vi è quindi una breve presentazione delle classi principali e delle loro relazioni.

La Sezione 2 analizza nel dettaglio l'implementazione delle classi più rilevanti del sistema ed espone le motivazioni di alcune scelte prese in fase di sviluppo. Il tutto è organizzato suddividendo le classi in relazione al componente di appartenenza, in riferimento al pattern architetturale, vi è poi la spiegazione di alcuni metodi di utility ritenuti significativi.

La Sezione 3 è relativa allo sviluppo dell'interfaccia grafica del sistema e mostra alcune stampe schermo esemplificative

5.1 Implementazione generale

La realizzazione del sistema si basa sull'implementazione di 4 classi fondamentali:

- Setup: ha il compito di inizializzare il sistema istanziando tutte le classi necessarie al suo funzionamento;
- BrowserObservable: contiene tutte le funzioni che permettono di intercettare le azioni svolte dall'utente sull'interfaccia grafica;
- Controller: collega le azioni compiute dall'utente sull'interfaccia grafica alle relative funzionalità sulla view;
- Drawer (View): comprende tutte le funzioni relative a visualizzazione e modifica della mappa.

Per una maggior comprensibilità, è mostrato in figura il diagramma di sequenza relativo all'implementazione di queste 4 classi e alle loro interazioni, basate sul pattern Observer.

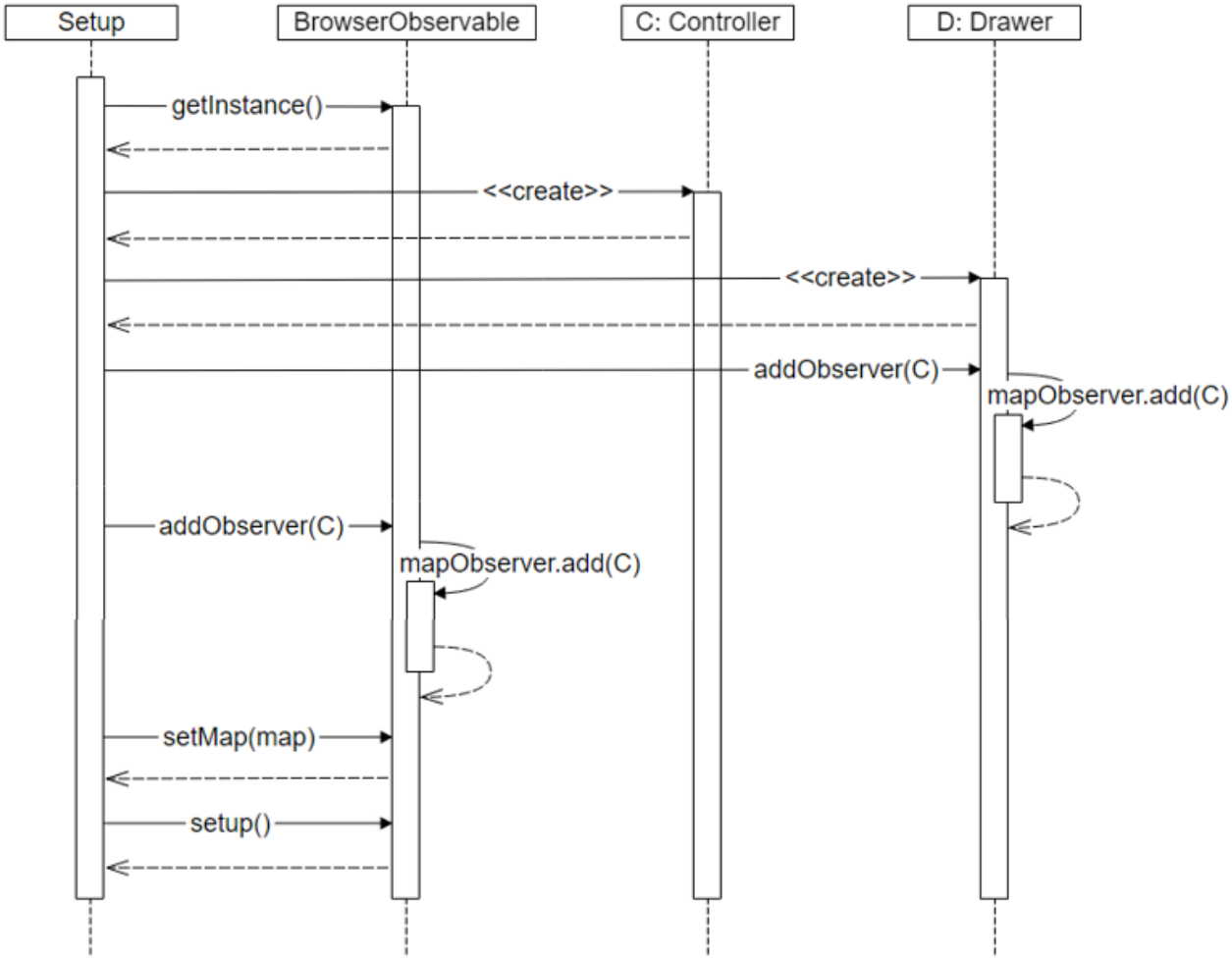


Figura 5.1: Implementazione generale del sistema - Diagramma di sequenza

Pattern Observer. Il pattern Observer è un design pattern che viene utilizzato come base architetturale di molti sistemi di gestione di eventi.

Si basa su uno o più oggetti, chiamati osservatori o observer, che vengono registrati per gestire un evento che potrebbe essere generato dall'oggetto osservato, che prende il nome di soggetto. Grazie a questo pattern è possibile definire una dipendenza uno a molti tra oggetti in modo tale che, se uno di questi cambia il suo stato in-

terno, ciascuno degli oggetti dipendenti da esso riceva la notifica o venga aggiornato automaticamente.

L'Observer nasce dall'esigenza di mantenere un alto livello di consistenza tra classi correlate senza però produrre situazioni di forte dipendenza o di accoppiamento elevato e prevede l'implementazione di due classi:

- Subject: è la classe che può essere soprannominata Observable, quella che conosce i propri observer, che possono anche essere in numero illimitato, e fornisce operazioni per effettuarne l'aggiunta e la rimozione oltre che metodi per notificar loro gli eventi;
- Observer: è l'interfaccia Observer, specifica per la notifica degli eventi agli oggetti interessati ad una Subject.

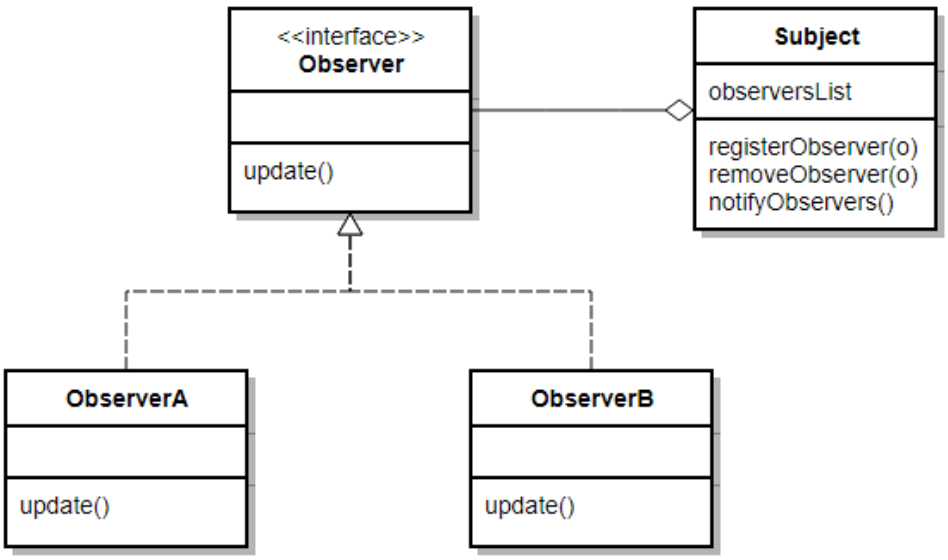


Figura 5.2: Struttura del pattern Observer

5.2 Sviluppo

Sulla base dei requisiti formulati dal committente in merito alle dimensioni ridotte e alla portabilità richieste dall'applicazione e considerando i vantaggi ottenuti dall'utilizzo di questa pratica, si è deciso di realizzare il sistema lato Client mediante implementazione

di una Web App. L'applicazione è stata sviluppata nei linguaggi di programmazione TypeScript e Android, quest'ultimo solamente per la parte relativa all'interfacciamento con i sensori del dispositivo mobile.

Di seguito, organizzati in relazione al componente di appartenenza, si presentano le classi del sistema e le relative funzionalità implementate.

5.2.1 Model

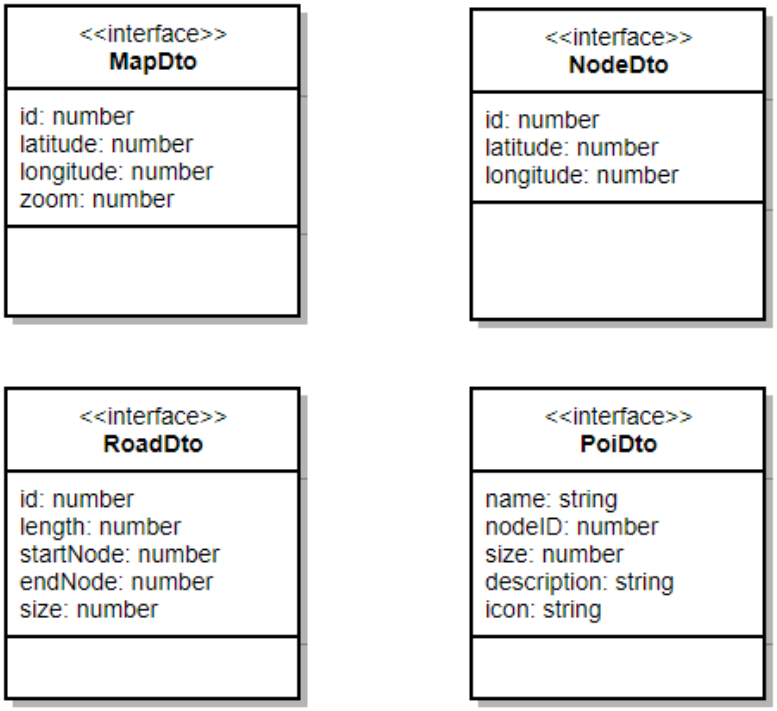


Figura 5.3: Model - Diagramma delle classi

Per l'implementazione del model, sulla base di quanto già descritto nel capitolo relativo allo sviluppo, la scelta è ricaduta sull'utilizzo di interfacce DTO (Data Transfer Object), in riferimento al design pattern omonimo. I DTO vengono utilizzati con gli oggetti di trasferimento dati MVC per associare modelli di dominio ad oggetti più semplici che verranno poi visualizzati dalla view, sono utili quindi per effettuare il trasferimento di parametri ai metodi o da usare come tipi di ritorno, e la particolarità consiste nel fatto di non avere alcun comportamento ma di limitarsi ad archiviare e recuperare i dati.

In Figura 5.3 è mostrato il diagramma delle classi relativo all'implementazione del componente model in cui sono definite le 4 strutture dati utilizzate per la memorizzazione

dei dati ricevuti dal server con gli attributi necessari per la realizzazione dei relativi elementi grafici.

5.2.2 View

Nel rispetto del design effettuato, la view è rappresentata da una sola interfaccia, Drawer, e dichiara tutti i metodi necessari al disegno di mappa, nodi, strade e punti di interesse ma anche quelli che permettono di agire sulla mappa quali la segnalazione del percorso e la modifica della visuale.

Gli elementi grafici che vengono ricevuti dal server sono di 4 tipologie:

- mappa
- strade
- punti di interesse
- nodi

Di questi, gli unici a dover essere memorizzati ma non implementati graficamente sono i nodi, necessari unicamente alla rappresentazione e gestione delle strade, mentre per tutti gli altri tipi di dato sarà necessario cercare un elemento grafico adatto alla rappresentazione sulla mappa.

Iniziata la fase di sviluppo del sistema, è stata effettuata una ricerca allo scopo di individuare una libreria grafica che supportasse l'implementazione delle funzionalità necessarie a soddisfare i requisiti di progetto e la scelta è ricaduta su Leaflet. Principale libreria grafica per quanto riguarda la gestione di mappe interattive in JavaScript, Leaflet consente non solo di visualizzare mappe coerenti e realistiche ma fornisce anche la possibilità di agire su di esse creando nuovi elementi grafici assimilabili a strade ed icone di punti di interesse.

I dati ricevuti tramite API client vengono passati al MapDrawer in fase di setup ed esso procede con la separazione ed interpretazione degli stessi: per ogni tipo di dato ricevuto in formato JSON viene memorizzato il relativo oggetto e, fatta eccezione per i nodi, si ha la creazione degli elementi grafici corrispondenti, concretizzati nei tipi di dato Map, Polyline, Marker di Leaflet.

Pur avendo organizzato l'intero progetto in modo che tutti gli eventi fossero intercettati da un'unica classe rappresentata da BrowserObservable, per realizzare un'interfaccia grafica in cui tutte le icone posizionate sulla mappa fossero cliccabili è stato necessario

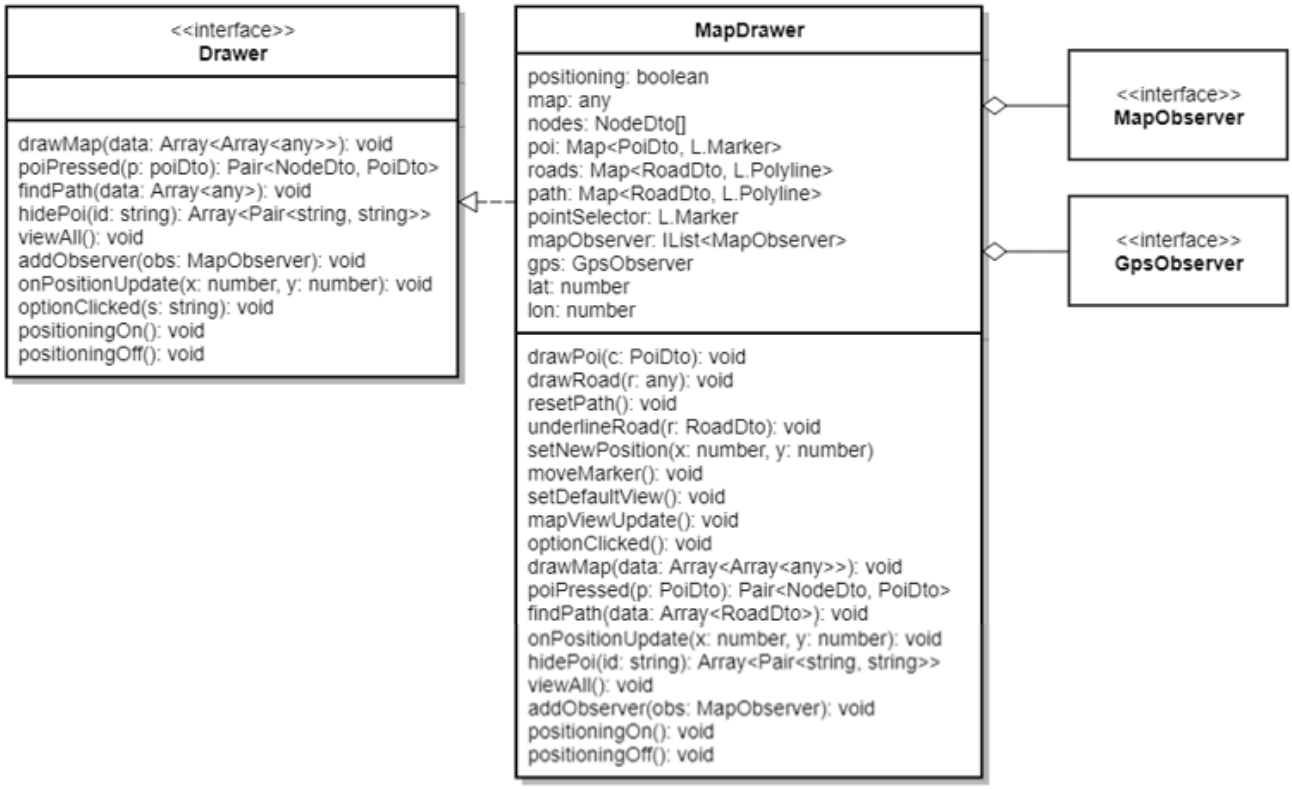


Figura 5.4: View - Diagramma delle classi

memorizzare nella classe MapDrawer una lista di MapObserver, interfaccia implementata dalla classe Controller, ai quali notificare i click effettuati poichè, trattandosi di elementi grafici inseriti in linguaggio TypeScript, l'utilizzo di JQuery in questo caso sarebbe risultato poco immediato e molto dispendioso in termini di calcoli da effettuare sulle coordinate cliccate.

MapDrawer vede inoltre la presenza di due attributi, latitudine e longitudine, che vengono aggiornati in tempo reale durante lo spostamento dell'utente per permettere un corretto funzionamento della modalità di localizzazione (positioning = true) che prevede che la visuale della mappa venga modificata in ogni momento, in modo che risulti essere sempre centrata sulla posizione dell'utente (in caso contrario, non memorizzando questi attributi, per modificare la visuale della mappa sarebbe stato necessario attendere l'invio di coordinate successivo).

La rappresentazione della posizione dell'utente mediante cursore viene realizzata con l'apposizione di un'icona sulla mappa le cui coordinate vengono aggiornate in relazione a quelle ricevute da parte del rilevatore GPS e, nel caso di localizzazione dell'utente attiva,

al termine di questa procedura avviene anche il setting del centro della mappa alle stesse coordinate.

Per implementare la funzionalità di ricerca dei punti di interesse, MapDrawer riceve dal Controller la stringa digitata dall'utente e procede analizzando i nomi di tutte le icone presenti sulla mappa, nascondendo quelle per le quali non viene trovato un matching e restituendo un array contenente le destinazioni correlate trovate e le relative distanze che le separano dall'utente, calcolate mediante l'utilizzo del metodo di utility presente nell'apposita classe DistanceCalculator.

5.2.3 Controller

Come deciso in fase di design, il componente relativo al Controller si occupa di realizzare varie funzionalità che possono essere considerate distinte e che quindi sono state implementate in classi differenti. Si presentano di seguito quelle ritenute più significative per la realizzazione del progetto.

IBrowserObservable

La classe BrowserObservable, implementazione della relativa interfaccia IBrowserObservable, racchiude quasi tutte le interazioni fra il sistema e l'utente allo scopo di conferire una maggior chiarezza al progetto, contiene quindi una funzione per ogni elemento grafico presente sulla schermata che permette l'attivazione della funzionalità corrispondente.

In questa classe è mantenuta una lista di oggetti di tipo CommandObserver, interfaccia implementata dal Controller, ai quali vengono notificati tutti i click compiuti dall'utente, mediante eventi registrati nel metodo setup(). Un caso particolare è rappresentato dall'evento relativo alla pressione del pulsante "Cerca" poichè l'invocazione del relativo metodo di notifica sull'observer restituisce in output una lista di punti di interesse che vengono creati dinamicamente come <input> ed aggiunti alla pagina HTML, in modo tale da creare un menu a tendina contenente i risultati della ricerca.

In questa classe sono inoltre presenti funzioni relative all'icona di caricamento visualizzata al lancio dell'applicazione e il metodo openDescriptionPanel() che viene invocato dal Controller alla ricezione della notifica relativa ad un click su un'icona, e che permette il setting del pannello relativo al punto di interesse cliccato con il nome e la descrizione corrispondenti.

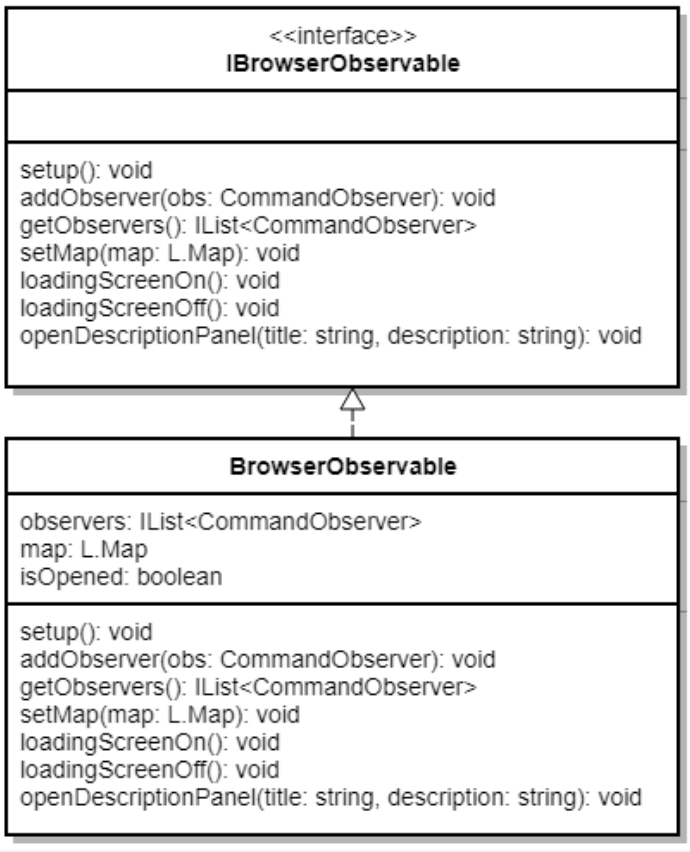


Figura 5.5: BrowserObservable - Diagramma delle classi

MapObserver e CommandObserver

La classe Controller implementa due differenti interfacce: MapObserver e CommandObserver. La prima contiene solo un metodo, onPoiClicked(), che viene utilizzato dalla view per notificare l'avvenuto click su un'icona presente sulla mappa, la seconda dichiara invece tutti i metodi che vengono invocati dalla classe BrowserObservable per notificare i click effettuati sugli elementi HTML della pagina Web. Il metodo onPoiClicked() viene chiamato dalla classe MapDrawer passando come parametro in ingresso una coppia di dati costituita dall'oggetto relativo alla destinazione selezionata e dal nodo corrispondente, ciò permette di comunicare alla classe Controller che l'utente ha effettuato il click su un'icona e di propagare la notifica a BrowserObservable, che procederà facendo apparire il relativo pannello descrittivo. Inoltre, poichè il click sulle icone non si limita a fare apparire un pannello nella parte bassa dello schermo ma causa anche la segnalazione del percorso che unisce l'utente e il punto cliccato, all'interno di questo metodo è presente anche l'invoca-

zione del metodo, anch'esso presente nel Controller, che si occupa di effettuare la chiamata API al server necessaria per ottenere il percorso da evidenziare.

Le altre funzioni implementate in questa classe riguardano la propagazione dei click effettuati sugli elementi HTML sulla classe Drawer, per esempio quelli relativi all'attivazione/disattivazione della modalità di localizzazione, la pressione dei pulsanti "Start" o "Search" e la visualizzazione di tutte le icone determinata dall'annullamento di una ricerca precedentemente effettuata.

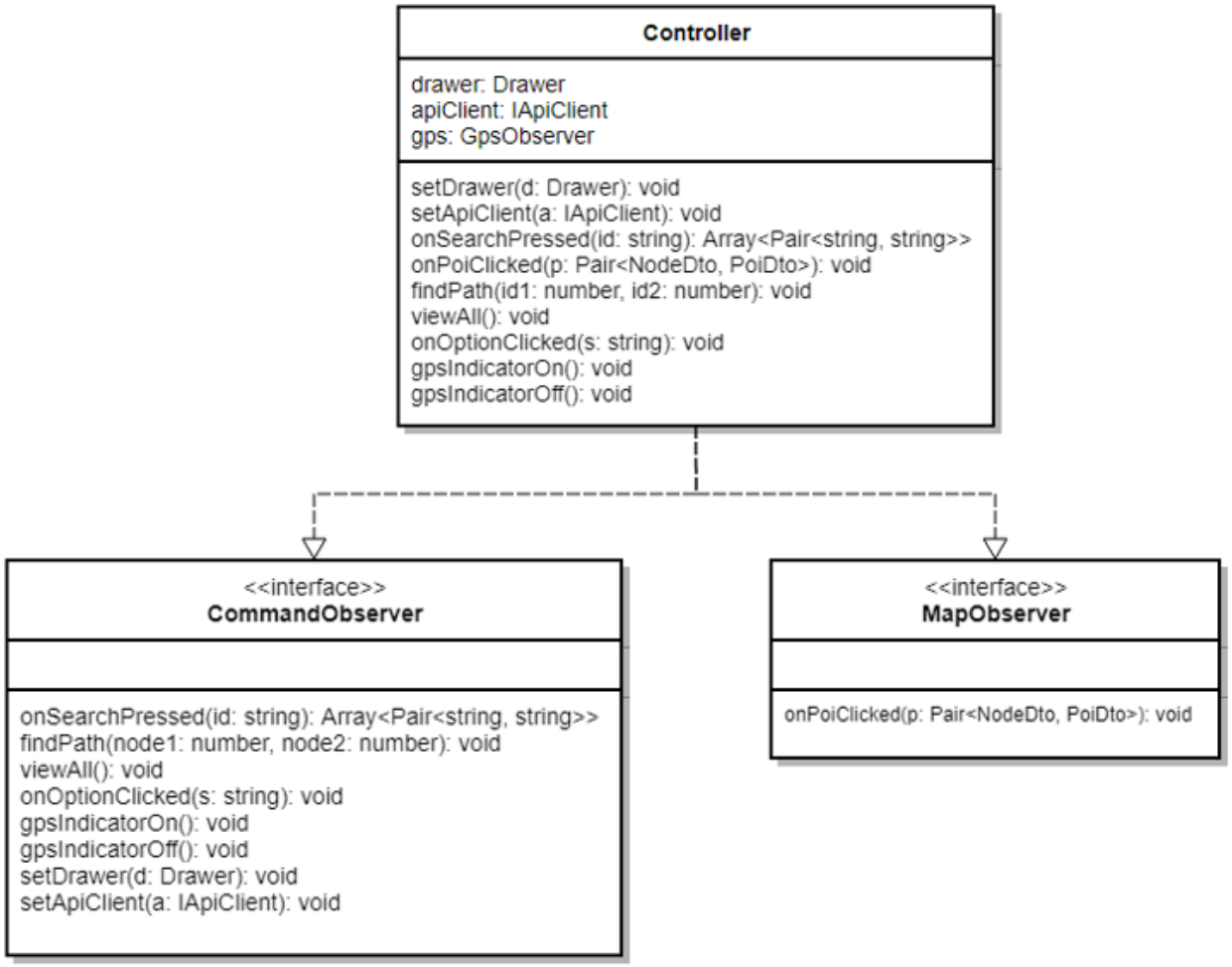


Figura 5.6: Controller - Diagramma delle classi

GpsObserver

L'interfaccia GpsObserver viene implementata dalla classe GpsObservable che memorizza come attributi privati le coordinate GPS rilevate (latitudine e longitudine) e una

lista di oggetti di tipo Drawer come observers ai quali viene notificato l'aggiornamento della posizione mediante passaggio dei relativi parametri.

Con l'invocazione del metodo addObserver() si ha l'aggiunta di Drawer alla lista e il passaggio delle coordinate GPS attuali, questo allo scopo di permettere la corretta modifica della visuale della mappa al click sull'icona di localizzazione avvenuto precedentemente all'avvio della navigazione.

L'utilizzo di appositi attributi della classe per la memorizzazione di latitudine e longitudine è dovuta alla presenza del metodo getActualPosition() che permette di passarli come parametri di input a getPosition(), implementato nella classe di utility Position e necessario per rendere utilizzabili le coordinate utente per lo svolgimento dei calcoli dell'algoritmo.

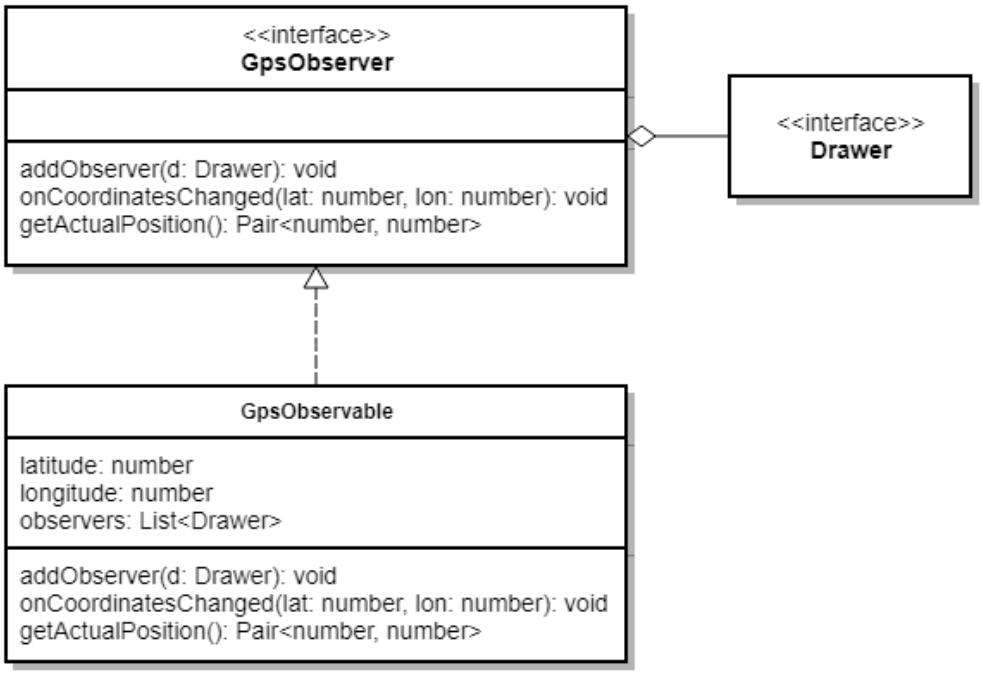


Figura 5.7: GpsObserver - Diagramma delle classi

Setup

La classe di Setup si compone di un solo metodo, denominato anch'esso setup(), che viene invocato al termine del caricamento della pagina Web e che ha il compito di inizializzare tutte le istanze necessarie al funzionamento del sistema (mappa, Controller, IApiClient, Drawer e GpsObserver) e di effettuare la registrazione dei relativi Observer.

All'interno del metodo è inoltre presente un'istanza locale di IBrowserObservable che permette l'aggiunta alla pagina Web dell'icona di caricamento e la sua successiva rimozione ad inizializzazione del sistema completata, ma causa anche l'invocazione del metodo setup() sullo stesso BrowserObservable.

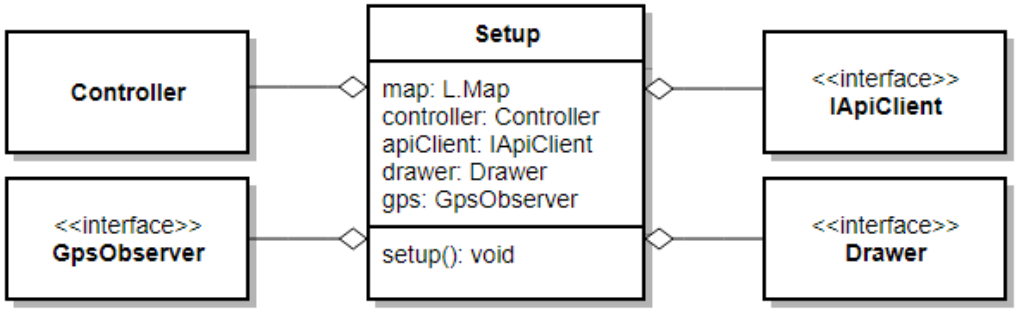


Figura 5.8: Setup - Diagramma delle classi

IApiClient

Dovendo instaurare una connessione fra client e server, è stato implementato un sistema di chiamate API che permettesse il passaggio di dati fra i due componenti.

L'interfaccia IApiClient dichiara i metodi necessari per effettuare le API al Server allo scopo di ricevere dati in ingresso; in particolare, i tipi di dato sono 5 (nodi, strade, punti di interesse, percorso, mappa) e corrispondono 5 differenti chiamate API.

L'interfaccia viene implementata da due differenti classi che sono HttpApiClient e MockApiClient: la prima realizza effettivamente la comunicazione con il server, la seconda è invece una classe mock implementata per realizzare il testing del sistema unicamente lato Client e quindi per permettere lo sviluppo autonomo delle due parti di progetto.

La classe HttpApiClient si compone di due attributi privati: path, stringa che memorizza il percorso necessario per effettuare le chiamate al server, e id, corrispondente all'id della mappa da richiedere, ed entrambi vengono settati nel costruttore mediante passaggio di parametri dalla classe Setup. Le chiamate al server vengono realizzate grazie all'utilizzo di richieste AJAX che forniscono in ingresso i dati sottoforma di testo JSON, questi vengono poi passati al costruttore della classe Drawer per poter essere interpretati; differente è il caso della richiesta relativa alle strade che appartengono al percorso da segnalare, in questo caso infatti l'invocazione del metodo corrispondente non avviene in fase di inizializzazione del sistema ma in seguito ad una richiesta specifica da parte dell'utente.

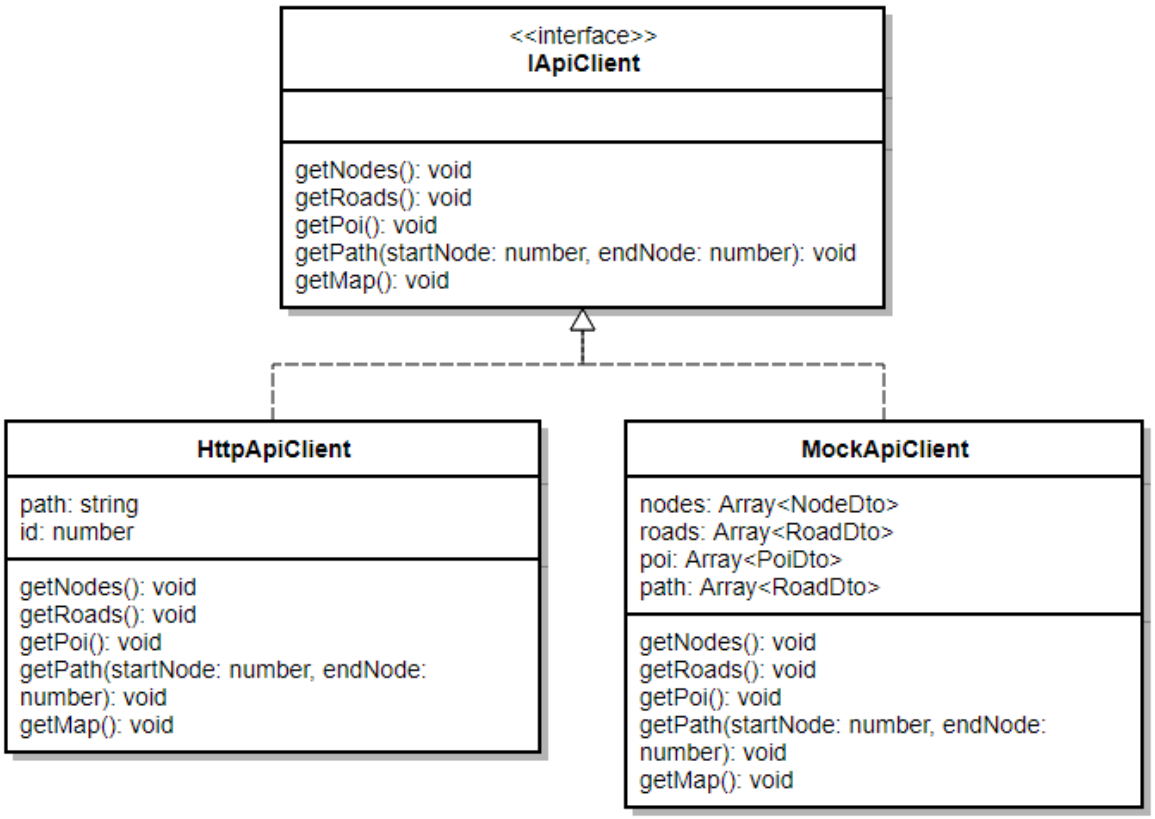


Figura 5.9: IApiClient - Diagramma delle classi

Chiamate AJAX e funzioni di callback. AJAX (Asynchronous JavaScript and XML) è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive che permette di realizzare chiamate di tipo asincrono ad una risorsa esterna in modo che queste non vadano ad interferire con la sua esecuzione. Il caricamento della risorsa esterna avviene in background e i risultati forniti da essa saranno utilizzabili solo dopo essere stati resi disponibili dalla risorsa stessa, senza causare tempi morti per l'utilizzatore.

Mediante AJAX è possibile cambiare dinamicamente il contenuto di una pagina o di una sua porzione attraverso una chiamata HTTP che viene lanciata tramite JavaScript, ed è quest'ultimo ad occuparsi della gestione di eventuali errori e della manipolazione del risultato ricevuto in risposta interagendo con il DOM del documento.

Le funzioni di callback vengono tipicamente eseguite dopo la routine principale, cioè al termine dell'esecuzione primaria lanciata dalla funzione chiamante e questo permette

a funzioni specifiche di svolgere compiti ulteriori ai propri, normalmente non noti al momento della scrittura del codice della funzione principale.

Nel caso specifico del progetto di tesi, l'utilizzo di callback in aggiunta alle chiamate AJAX ha reso possibile l'implementazione di chiamate al Server di tipo sincrono, in modo tale da evitare il lancio di eccezioni dovute al tentativo di visualizzare la pagina Web prima di aver effettivamente ricevuto i dati da visualizzare.

5.2.4 Gestore dei sensori

In Android, l'accesso ai servizi di localizzazione viene fornito tramite istanze di oggetti definiti nel package android.location.*, vi è inoltre un LocationManager mediante il quale è possibile istanziare i componenti necessari per ottenere informazioni sulla posizione dell'utente. Ottenuta l'istanza del LocationManager, è quindi possibile:

- registrare Intent per essere notificati quando il device si trova in prossimità di specifiche posizioni, indicate con latitudine e longitudine;
- interrogare l'istanza di un oggetto di tipo LocationProvider per conoscere l'ultima posizione nota determinata dal sistema di geolocalizzazione;
- registrarsi per ottenere aggiornamenti periodici.

L'ultima opzione è stata riconosciuta come quella più adatta per essere utilizzata in questo progetto di tesi. A tal proposito, è stata modificata la classe Gps di Android in modo da poter memorizzare una lista di observers ai quali, mediante override del metodo onLocationChanged(), viene notificata ogni variazione di posizione rilevata dal GPS mediante passaggio dei parametri di latitudine e longitudine.

Per consentire l'utilizzo del rilevatore GPS e permettere il funzionamento dell'applicazione, è stato inoltre necessario dichiarare il relativo permesso nel file Manifest del progetto e questo significa che, al primo avvio dell'applicazione, viene esplicitamente richiesto all'utente di poter utilizzare i servizi di geolocalizzazione presenti sul dispositivo.

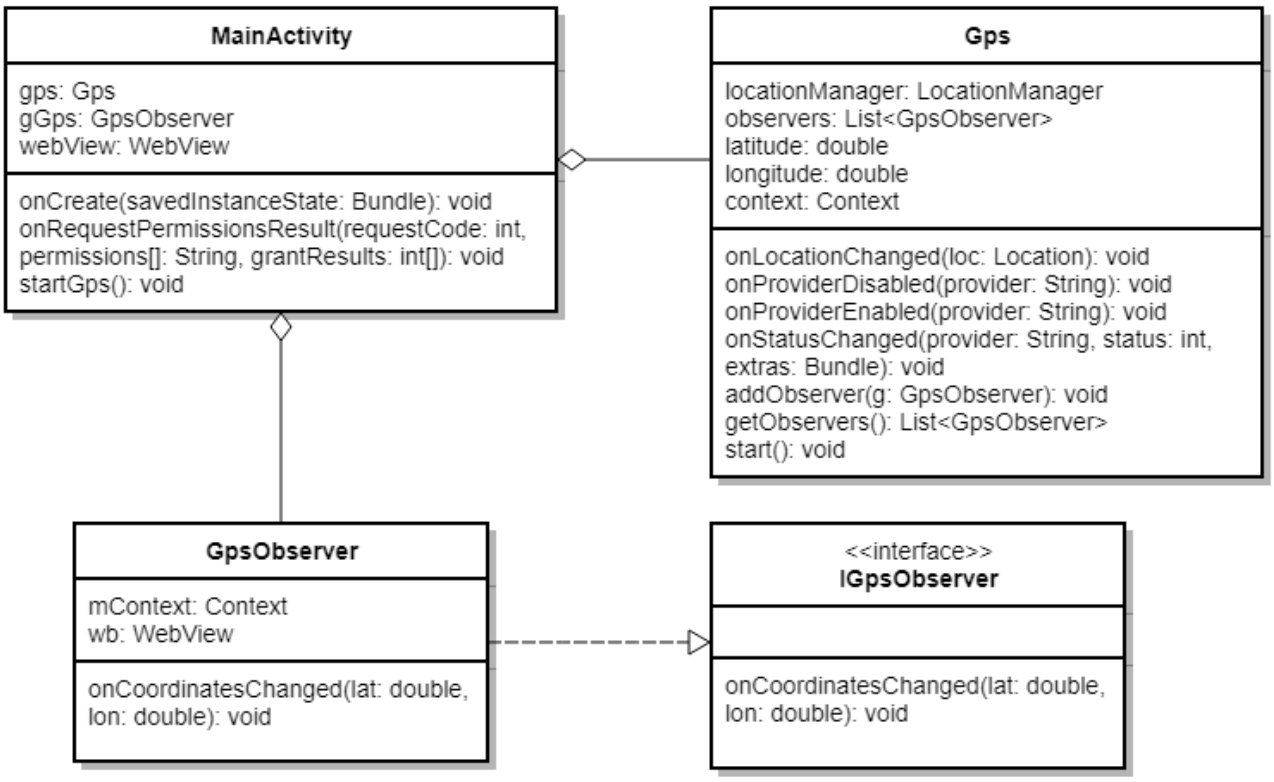


Figura 5.10: Gestore dei sensori - Diagramma delle classi

5.2.5 Utility

DistanceCalculator

Per il calcolo della distanza tra i nodi del grafo è stata utilizzata una formula che permette di calcolarla considerando anche il raggio di curvatura della Terra, in modo che il risultato sia quanto più possibile veritiero. Di seguito viene presentato lo pseudocodice di riferimento per l'implementazione del metodo.

```
1  public getDistanceNodeToNode(LatNode1, LonNode1, LatNode2, LonNode2) {
2          /* Raggio della Terra in km */
3          var R = 6371;
4          var distanceLat = (LatNode2-LatNode1) * (PI/180);
5          var distanceLon = (LonNode2-LonNode1) * (PI/180);
6          var a =  sin(distanceLat/2) * sin(distanceLat/2) +
7                  cos(LatNode1 * (PI/180)) * cos(LatNode2 *
```

```
                    (PI/180)) * sin(distanceLon/2) * sin(distanceLon/2);
            var c = 2 * arctan(sqrt(a), sqrt(1-a));
            /* Distanza in km */
            var distance = R * c;
            /* Distanza in m */
            return distance * 1000;
}
```

Container

Il progetto è stato realizzato in modo da implementare il design pattern Dependency injection per tutte le classi le cui istanze vengono utilizzate nel codice, questo allo scopo di poter passare agevolmente dall'utilizzo del sistema mock a quello di reale connessione con il server e semplificare la gestione delle classi che implementano il pattern Singleton. Il framework utilizzato, InversifyJS, richiede a tal proposito la dichiarazione di una costante che permetta di definire gli identificatori da utilizzare a runtime, in questo caso è stata creata TYPES come mostrato.

```
let TYPES = {
        IApiClient: Symbol("IApiClient"),
        CommandObserver: Symbol("CommandObserver"),
        MapObserver: Symbol("MapObserver"),
        GpsObserver: Symbol("GpsObserver"),
        Drawer: Symbol("Drawer"),
        BrowserObservable: Symbol("BrowserObservable")
};
export default TYPES;
```

Ogni classe che implementa un'interfaccia presente in TYPES è stata quindi dichiarata utilizzando il decoratore @injectable e, nel caso in cui fosse presente all'interno della classe una dipendenza con un'altra interfaccia, è stato utilizzato il decoratore @inject. Di seguito è inserito un esempio di dichiarazione di una classe injectable contenente una dipendenza.

```
@injectable()
export class Controller implements CommandObserver, MapObserver {
        private drawer: Drawer;
        private apiClient: IApiClient;
        private _gps: GpsObserver;
```

```
6
7           public constructor (@inject(TYPES.GpsObserver)
8                   _gpsObs: GpsObserver) {
9                   this._gps = _gpsObs;
10          }
11  }
```

Fondamentale nell'implementazione della Dependency injection è però la creazione del Container, che nel caso specifico di InversifyJS deve essere presente all'interno di un file denominato "inversify.config.ts". All'interno del Container è stata stabilita l'associazione tra le interfacce e le relative classi che le implementano e ciò ha reso possibile l'eliminazione dal restante codice del progetto di ogni riferimento alle classi iniettate, rendendo sufficiente l'utilizzo del Container e dell'interfaccia di riferimento.

```
1  var container = new Container();
2  container.bind<IApiClient>(TYPES.IApiClient).to(MockApiClient);
3  container.bind<CommandObserver>(TYPES.CommandObserver).to(Controller);
4  container.bind<MapObserver>(TYPES.MapObserver).to(Controller);
5  container.bind<Drawer>(TYPES.Drawer).to(MapDrawer);
6  container.bind<GpsObserver>(TYPES.GpsObserver).to(GpsObservable)
7          .inSingletonScope();
8  container.bind<IBrowserObservable>  TYPES.BrowserObservable)
9          .to(BrowserObservable).inSingletonScope();
10 export default container;
```

Ciò che si ottiene con la Dependency Injection è un codice interamente privo di riferimenti alle classi iniettate, fatta eccezione per l'associazione presente nel Container, e questo ha semplificato lo sviluppo, in particolare relativamente all'utilizzo dell'interfaccia IApiClient alla quale è stata associata l'implementazione presente nella classe MockApiClient per il testing del sistema, sostituita poi da HttpApiClient a progetto ultimato.

Dependency injection. Si tratta di un design pattern della programmazione orientata agli oggetti, più nello specifico di una particolare forma del pattern Inversion of Control (IoC) grazie al quale è possibile semplificare lo sviluppo del software e migliorarne la testabilità.

L'implementazione di questo pattern prevede la presenza di un componente esterno, rappresentato da un assembler, che crea gli oggetti e le relative dipendenze e che le assembla mediante l'utilizzo dell'injection che può essere realizzata in 3 modi diversi:

- constructor injection: la dipendenza si inietta come argomento del costruttore;
- setter injection: la dipendenza si inietta attraverso un metodo "set";
- interface injection: si basa sul mapping tra interfaccia e relativa implementazione.

Utilizzando la Dependency injection, le classi non sono quindi responsabili dell'inizializzazione delle proprie dipendenze e l'oggetto ha in sè solamente una proprietà che può ospitare un riferimento a quel servizio, iniettato all'insaputa del programmatore per quanto riguarda il posizionamento o i dettagli dello stesso. In generale, un container è un componente esterno che svolge una serie di compiti esonerando così lo sviluppatore dall'occuparsene, nel caso più specifico di un IoC Container si tratta di uno specifica tipologia di container per la Dependency injection che è in grado di compiere determinate operazioni di injection grazie ad apposite configurazioni definite dall'utente.

Il pattern si compone di 3 elementi che sono:

- componente dipendente,
- dichiarazione delle dipendenze del componente (interface contracts),
- injector (è chiamato anche provider o container).

L'injector ha il compito di creare, su richiesta, le istanze delle classi che implementano le dependency interfaces e perciò, quando il componente viene istanziato, esso si prende carico di risolvere le dipendenze, in particolare:

- se è la prima volta che si tenta di risolverla, l'injector istanzia il componente dipendente, lo memorizza in un container e lo restituisce;
- se non è la prima volta che si tenta di risolverla, l'injector si limita a restituirne la copia salvata nel contenitore.

Risolte tutte le dipendenze, il controllo può tornare al componente applicativo.

Il container di IoC utilizzato in questo specifico progetto è InversifyJS[1], un framework molto leggero (4 KB) utilizzabile su applicazioni TypeScript e JavaScript, creato per supportare gli sviluppatori nella scrittura di codice caratterizzato da un buon design Object Oriented.

PathCalculator

La scelta di voler implementare l'algoritmo di ricerca del percorso minimo anche lato client è stata fatta considerando i due principali vantaggi che ne sarebbero derivati:

- poter sopperire agli eventuali malfunzionamenti del server;
- poter effettuare un testing più completo del sistema lato client in concomitanza con l'utilizzo di richieste mock per simulare la comunicazione con il server.

A tal proposito, il sistema è stato implementato in modo che la ricerca del percorso, quella innescata dal click sull'icona corrispondente, venga effettuata dal server che restituisce le strade da evidenziare mediante chiamata AJAX, si ricorre però all'invocazione dell'algoritmo implementato lato client in fase di ricerca, per fornire le distanze che separano l'utente dai punti di interesse individuati. Per realizzare un software il più possibile modulare e riusabile, le funzioni relative al calcolo del percorso sono state raccolte in un'unica classe denominata "PathCalculator". Il metodo statico getPath() ha visibilità pubblica poichè deve poter essere invocato dalle altre classi del sistema e prende in ingresso i nodi di partenza ed arrivo ma anche due array contenenti rispettivamente l'elenco dei nodi e degli archi presenti sulla mappa, ciò che restituisce in output è invece un array contenente le strade di cui si compone il percorso individuato. L'algoritmo di ricerca del percorso minimo implementato in questo sistema si basa sullo pseudocodice dell'algoritmo di Dijkstra, applicabile per ricercare i cammini minimi in grafi ciclici con pesi non negativi sugli archi.

Di seguito viene presentato lo pseudocodice dell'algoritmo implementato, per agevolarne la comprensione sono stati aggiunti commenti alle righe ritenute significative.

```
1  public GetPath(FirstNode, SecondNode, Nodes, Roads) {
2          foreach Node n in Nodes :
3                  /* Potenziale di ogni nodo posto a infinito */
4                  f.set(n, +Infinity)
5                  /* Predecessore di ogni nodo sconosciuto */
```

[1] http://inversify.io/

```
6                   j.set(n, null)
7           end for
8           f.Remove(FirstNode)
9           /* Potenziale del nodo di partenza posto a 0 */
10          f.Add(FirstNode, 0)
11          foreach Neighbour n of FirstNode :
12                  f.Remove(n)
13                  /* Distanza dai vicini conosciuta */
14                  f.Add(n, distanceBetween(FirstNode, n))
15          end for
16          t.addAll(Nodes)
17          t.Remove(FirstNode)
18          while (t is not Empty) :
19                  /* Nodo con la distanza f minima */
20                  nearest = getNearest(f, t)
21                  if (neighbours of Node contains nearest)
22                          j.Remove(nearest)
23                          j.Add(nearest, FirstNode)
24                  end if
25                  t.Remove(nearest)
26                  s.add(nearest)
27                  neighbours = getNeighbours(nearest, Nodes, Roads)
28                  foreach Node in s :
29                          neighbours.Remove(Node)
30                  end for
31                  foreach Node in neighbours :
32                          distance = f[nearest] +
33                                  distanceBetween(nearest, Node, Roads)
34                          if (distance < f[Node])
35                                  f.Remove(Node)
36                                  f.Add(Node, distance)
37                                  j.Remove(Node)
38                                  j.Add(Node, nearest)
39                          end if
40                  /*
41                       Calcolo del potenziale di ogni nodo come somma del
42                       potenziale precedente e del costo di collegamento
43                  */
44                  end for
45          end while
46          Next = SecondNode
```

```
47          do :
48                  prev = j[Next]
49                  /* Aggiunta della strada piu' corta */
50                  path.Add(getRoadsBetween(Next, Prev, Roads))
51                  Next = Prev
52          while (Next != StartNode)
53          return path
54  }
```

Position

Nella classe di Position è presente un solo metodo, getPosition(), fondamentale per il funzionamento del sistema poichè permette di calcolare, date le coordinate GPS inviate dal dispositivo Android, il nodo del grafo più vicino, mediante un metodo presente nella classe Nodes.

Per la gestione del calcolo del percorso, si è infatti deciso di approssimare sempre la posizione dell'utente a quella del nodo a minor distanza allo scopo di rendere più agevoli i calcoli e l'utilizzo dell'algoritmo.

Web App

Per realizzare la Web App, rappresentata in questo specifico caso da una singola pagina Web, è stata utilizzata un'estensione della classe View di Android chiamata WebView che consente la visualizzazione di pagine Web come parte del layout dell'attività.

Per aggiungere la WebView nell'applicazione è stata impostata l'intera finestra MainActivity come una WebView mediante il metodo onCreate() e, poichè la pagina da caricare è stata scritta in TypeScript, che non è altro che un'estensione di JavaScript, è stato inoltre necessario abilitare JavaScript per WebView (disabilitato di default).

Successivamente è stato realizzato il bridge tra Android e JavaScript implementando un'interfaccia apposita tra il codice relativo all'applicazione mobile e alla pagina web.

5.3 Implementazione dell'interfaccia

Nella realizzazione dell'interfaccia grafica dell'applicazione si è deciso di concretizzare i mockup realizzati in fase di design e, a dimostrazione di questo, in Figura 5.16 sono riportati alcuni screen relativi all'applicazione implementata. Un'ulteriore funzionalità di

supporto alla navigazione sarebbe stata la possibilità di modificare l'orientamento della mappa in relazione alla posizione dell'utente ma non essendo implementabile mediante la libreria grafica scelta e trattandosi di una funzionalità ritenuta non fondamentale per il buon funzionamento del progetto, data la notevole mole di lavoro che avrebbe richiesto, si è deciso di non implementarla all'interno di questo progetto di tesi. Ciononostante, considerato il valore in termini di User Experience, non si esclude la possibilità che essa venga integrata in future versioni del sistema.

(a) Schermata iniziale dell'applicazione

(b) Ricerca di un punto di interesse

(c) Selezione di un punto di interesse

(d) Navigazione verso un punto di interesse

Figura 5.11: Esempi relativi al funzionamento dell'applicazione

Capitolo 6

Testing del sistema

La fase di testing e validazione effettuata sul sistema ha rappresentato un valido supporto allo sviluppo e ha permesso di verificarne la corretta implementazione. L'applicazione è stata testata in modo piuttosto esaustivo, coprendo gran parte delle funzionalità presenti e ciò ha dato origine ad un software robusto e stabile. La validazione del sistema realizzata mediante Unit Testing si è concentrata sui metodi relativi alla categoria di utility, prediligendo una modalità di testing di tipo visivo per le componenti grafiche, effettuata mediante un utilizzo massiccio dell'applicazione che ha permesso di verificare che le funzionalità venissero attivate ed eseguite in modo corretto. La fase di testing effettuata rappresenta dunque un valido e significativo esempio delle modalità di validazione possono essere effettuate sul sistema, demandando a futuri sviluppi del software la ricerca di ulteriori test-case.

6.1 Unit testing

L'attività di testing svolta sul sistema si identifica con il nome di unit testing ed è caratterizzata dal fatto di essere eseguita su singole unità software, cioè su minimi componenti di un programma dotati di funzionamento autonomo, che nel caso di una programmazione orientata agli oggetti sono rappresentati dai metodi.

L'implementazione di unit testing permette di verificare il corretto funzionamento di determinate parti del programma e conseguentemente la precoce individuazione di eventuali bug. I test vengono solitamente scritti dagli sviluppatori stessi e sono dotati di uno o pochi input e di un singolo output.

Utilizzare unit testing nel proprio sistema software offre i seguenti vantaggi:

- maggiore affidabilità: la corretta implementazione del codice può essere testata e verificata per una molteplicità di casi possibili;
- maggiore riusabilità: dover eseguire i test su singole unità software si ha la realizzazione di codici modulari caratterizzati da una maggior riusabilità;
- minor tempo di sviluppo: nonostante il tempo impiegato per la scrittura di questi test, la loro esecuzione porta ad un notevole risparmio in termini di tempo di sviluppo poichè esenta lo sviluppatore dal dover testare manualmente il codice;
- maggior modificabilità: scrivere buoni unit test porta all'individuazione immediata di eventuali bug dovuti all'introduzione di modifiche e permette di porvi rimedio più agevolmente;
- maggior manutenibilità: dover eseguire i test su singole unità software comporta la realizzazione di codici meno interdipendenti tra loro in cui l'impatto legato alle modifiche risulterà sicuramente minore.

Come tutti i testing, l'applicazione di Unit Testing non certifica l'assenza di errori nel codice ma ne evidenzia la presenza, senza però identificarli. Inoltre, questa specifica attività non permette di individuare la presenza di errori di integrazione, i problemi legati alle performance o altri problemi relativi al sistema generale ed è per questo motivo che normalmente viene utilizzata insieme ad altre tecniche di testing del software. Nel progetto di tesi è stata quindi realizzata una prima fase di testing a scopo principalmente dimostrativo mediante utilizzo del framework Mocha[1] sull'ambiente di sviluppo Visual Studio Code.

Per definizione, i test effettuati devono essere riproducibili e ciò significa che i test case scelti non possono dipendere dai dati ricevuti dal server che potrebbero variare ad ogni avvio dell'applicazione, a tal proposito sono quindi stati appositamente dichiarati dati mock locali per una corretta validazione dei vari test case.

I test implementati sono stati suddivisi in 3 categorie in relazione all'elemento grafico di riferimento: nodi, strade e percorso.

6.1.1 Testing relativo ai nodi

Distanza tra nodi. Il calcolo della distanza che intercorre tra due nodi distinti viene realizzato mediante invocazione di un apposito metodo denominato getDistanceNo-

[1] `https://mochajs.org/`

deToNode() che utilizza le coordinate ricevute come parametri di input per svolgere le operazioni matematiche necessarie.

La correttezza del calcolo è stata testata utilizzando le coordinate di due nodi con distanza nota l'uno dall'altro passati come parametri in ingresso e verificando la correttezza del risultato ottenuto. Inoltre, l'implementazione del metodo è stata testata anche nel caso particolare in cui le coordinate di latitudine e longitudine passate in input siano le medesime, il test ha in questo caso restituito correttamente il valore della distanza pari a 0.

Ricerca di un nodo. Il metodo getNodeFromID() restituisce il nodo corrispondente all'identificatore numerico ricevuto come parametro in ingresso.

Questa funzionalità è stata testata mediante implementazione di unit test sia nel caso in cui l'id in input corrispondesse a quello di un nodo fra quelli memorizzati e sia nel caso in cui invece il nodo non fosse contenuto nell'array sul quale avviene la ricerca, entrambi i test sono stati superati e hanno fornito risultati corrispondenti a quelli effettivamente attesi.

Nodo a minor distanza. Il metodo che permette, date due coordinate di latitudine e longitudine, di trovare il nodo più vicino ad esse fra quelli memorizzati nel sistema si chiama nearestNode() e viene utilizzato per poter approssimare la posizione dell'utente a quella del nodo a minor distanza per applicare l'algoritmo di ricerca del percorso.

Il test effettuato ha in questo caso permesso di verificare che, passando in ingresso coordinate per le quali era già noto il nodo più vicino, il risultato ottenuto fosse corretto.

Coordinate di un nodo. Le coordinate di un nodo presente sulla mappa vengono restituite mediante il metodo getCoordinates() al quale deve essere passato in ingresso l'identificatore numerico corrispondente.

In questo caso, i test implementati hanno verificato che passando in input l'id di un nodo della mappa, le coordinate restituite fossero corrette, ma è stato anche testato il caso in cui l'id in ingresso non corrispondesse ad alcun nodo memorizzato dal sistema.

6.1.2 Testing relativi alla ricerca del percorso

Ricerca vicini. Per facilitare la ricerca del percorso minimo, è stato implementato un metodo denominato getNeighbor() che, ricevuta in input l'istanza di un nodo, restituisce una lista contenente tutti i vertici direttamente collegati ad esso mediante gli archi della mappa.

Avendo a disposizione un nodo memorizzato nel sistema e la corrispondente lista di nodi ad esso adiacente, mediante unit testing è stato verificato che i vicini restituiti dall'invocazione del metodo fossero corretti. Un altro test case effettuato su questo metodo ha inoltre riguardato il caso in cui il parametro passato non appartenesse a quelli presenti sulla mappa, la restituizione di una lista vuota ha nuovamente confermato il superamento del test.

6.1.3 Testing relativi alle strade

Strade collegate ad un nodo. La ricerca delle strade collegate ad uno specifico nodo viene effettuata dal metodo getRoadsFromNode() al quale deve essere passato come parametro in ingresso l'id numerico del vertice sul quale deve essere svolta l'analisi.

L'applicazione di unit testing a questo metodo ha visto l'individuazione di due test case: il primo ha confermato che passando in input l'id di un nodo fra quelli memorizzati dal sistema, la funzione restituisce correttamente la lista degli archi ad esso collegati, il secondo ha invece verificato che l'invocazione del metodo con id numerico non corrispondente ad alcun nodo della mappa non restituisce alcuna strada in output.

Ricerca di una strada. Utilizzando come parametri in ingresso due nodi fra quelli presenti sulla mappa, mediante l'invocazione del metodo getRoadBeetweenNodes() si ha la restituzione dell'arco corrispondente che li congiunge.

Il testing di questa funzionalità ha permesso di verificare che invocando il metodo con parametri di ingresso corrispondenti a due differenti nodi fra quelli rappresentati nella mappa, venga correttamente restituita la strada che li unisce. Inoltre, si è testato il buon funzionamento del metodo, individuato dalla non restituzione di una strada, anche nei casi in cui:

- uno dei nodi passati in ingresso sia inesistente;
- entrambi i nodi passati in ingresso siano inesistenti.

Capitolo 7

Conclusioni

Obiettivo del progetto era l'implementazione di un'applicazione mobile che agisse da navigatore nel caso specifico di ambienti esterni non mappati dagli attuali sistemi del settore presenti in commercio. Il processo che ha portato alla realizzazione del software è stato suddiviso in numerosi fasi, sia periodiche e sia in corrispondenza dell'implementazione di funzionalità complesse, al termine delle quali sono state svolte attività di revisione e ottimizzazione del codice prodotto e a tal proposito si è rivelato fondamentale il supporto offerto dei tutor aziendali che hanno contribuito a constatare la correttezza funzionale del lavoro svolto e hanno offerto un valido aiuto nella scelta delle migliori strategie da adottare. L'attento studio effettuato per la realizzazione delle fasi di analisi e progettazione ha avuto come esito l'implementazione di un sistema funzionante, sul quale sono state svolte numerose attività di testing sia a livello di funzionamento dell'interfaccia grafica, per il quale è stato svolto un utilizzo massiccio dell'applicazione, che mediante implementazione di test che hanno coperto gran parte delle funzionalità implementate. Il software risulta efficiente e rispetta i requisiti di sistema stabiliti in fase di analisi e la valutazione positiva espressa in merito anche da parte dell'azienda rappresenta un uleriore constatazione del raggiungimento dell'obiettivo di partenza.

7.1 Sviluppi futuri

In futuro, potranno essere svolte sul sistema attività di ottimizzazione dell'algoritmo di ricerca del percorso utilizzato ed implementati meccanismi di correzione dei dati errati provenienti dal sensore GPS.

Durante la fase di progettazione si è prestata particolare attenzione alla realizzazione di un architettura che agevolasse l'implementazione di ulteriori funzionalità aggiuntive, integrabili in future versioni del sistema stesso. A progetto ultimato sono stati dunque individuati alcuni possibili contributi futuri da dare all'applicazione che vengono esposti di seguito.

Comunicazione delle indicazioni. Il sistema realizzato si limita ad evidenziare sulla mappa il percorso più breve che permette di raggiungere il luogo di interesse selezionato e ad aggiornare il cursore corrispondente all'utente sulla mappa in tempo reale con gli spostamenti, questo obbliga a dover controllare costantemente l'applicazione per avere un riscontro sulla correttezza o non correttezza della strada che si sta percorrendo. Per risolvere questo tipo di problematica e migliorare l'esperienza d'uso, si valuta la possibilità di integrare il software con un sistema di notifiche all'utente che preveda la comunicazioni di informazioni quali per esempio l'anticipazione delle svolte da effettuare e i metri di distanza, sull'esempio dei navigatori GPS delle automobili.

Ricalcolo del percorso. Per poter rappresentare un valido strumento per il wayfinding degli ambienti, l'applicazione dovrà implementare un sistema di ricalcolo del percorso trovato, dovuto agli spostamenti sbagliati dell'utente. Per la realizzazione di questa funzionalità sarà necessario effettuare uno studio relativo ad una modalità matematica per l'individuazione della strada sulla quale si trova il dispositivo, in modo da poter verificare se essa appartiene al percorso segnalato oppure no, ed attivare in questo caso il ricalcolo del percorso utilizzando come punto di partenza le nuove coordinate GPS.

Orientamento della mappa. Come già accennato, si ritiene rilevante fornire all'utente la possibilità di scegliere di far ruotare la visuale della mappa in relazione al proprio orientamento. Per implementare questa specifica funzionalità si rende necessaria la sostituzione della libreria grafica individuata per la realizzazione del progetto poichè si tratta di una modalità non supportata da quella utilizzata. Una buona fase di progettazione ha permesso di considerare questa eventualità limitando i riferimenti alla libreria grafica ad una sola classe del sistema, agevolandone quindi l'eventuale sostituzione. Si tratta dunque di una funzionalità che non richiederebbe grossi impieghi di risorse per la sua realizzazione.

Funzionamento in background. La possibilità di mantenere la navigazione attiva in background e l'integrazione di un sistema di notifiche all'utente contenenti le indicazioni da seguire, permetterebbe l'utilizzo del dispositivo per scopi diversi dalla navigazione anche durante l'esecuzione dell'applicazione stessa, si tratta quindi di una funzionalità che incrementerebbe l'usabilità generale del sistema.

Navigazione indoor. Un'ulteriore estensione del software sarebbe data dalla possibilità di utilizzarlo anche per la navigazione interna degli edifici, in modo da poter aumentare le applicazioni d'uso anche ad ospedali, aeroporti, centri commerciali eccetera. In questo caso, non sarebbe però possibile utilizzare i dati rilevati dal sensore GPS ma si dovrebbe studiare un metodo alternativo di individuazione della posizione dell'utente, per esempio basato su una triangolazione del segnale rilevato dalle reti Wi-Fi presenti nell'ambiente.

Gestione mappe con grafica 3D. Attualmente l'applicazione utilizza una grafica a due dimensioni ma potrebbe essere ulteriormente estesa in modo tale da fornire la possibilità di scegliere la modalità di visualizzazione 3D delle mappe. Questo rappresenterebbe un enorme progresso per l'interfaccia grafica del sistema ma comporterebbe anche notevoli costi dovuti alla ricerca di appositi tool grafici e librerie necessari per la realizzazione di una grafica 3D nonchè per la creazione delle mappe ricevute in input.

Bibliografia

[1] Brown E., *Learning JavaScript: add sparkle and life to your Web pages.* 3ªed. O'Reilly Media, Sebastopol CA. 2016

[2] Cormen T. H., Leiserson C. E., Rivest R. L, Stein C., *Introduction to algorithms.* 3ªed. MIT Press Ltd, Cambridge MA. 2009

[3] Nance C., *TypeScript essentials: develop large scale responsive Web application with TypeScript*, ebook. 2014

[4] Osmani A., *Learning JavaScript design patterns: A JavaScript and jQuery developer's guide.* O'Reilly Media, Sebastopol CA. 2012

[5] Zingale S. *Wayfinding e cognizione spaziale.* Melzani L., 2016. `http://www.salvatorezingale.it/wordpress/wpcontent/uploads/2017/05/ZINGALE-Wayfinding-e-cognizionespaziale.pdf`

[6] Web App: `https://developer.android.com/guide/webapps/`, consultato il 24.04.2018

Ringraziamenti

In conclusione voglio ringraziare tutti coloro che mi hanno aiutata nel raggiungimento di questo traguardo.

Il primo ringraziamento va al professor Mirko Viroli per il supporto fornitomi nella realizzazione di questo trattato di tesi.

Grazie a tutti i ragazzi di myDev per avermi accolta in azienda senza mai farmi sentire fuori posto, ringrazio Stefano e in particolare Giacomo per essere stato un formidabile tutor aziendale ed avermi aiutata e consigliato durante tutta la realizzazione del progetto.

Un sincero ringraziamento va ai miei genitori per avermi sempre sostenuta ed incoraggiata a compiere le mie scelte in totale autonomia, permettendomi di arrivare dove sono ora, e a mio fratello, per avermi dato la possibilità di ritagliarmi il tempo per terminare questa tesi.

Ringrazio i miei nonni, Eleonora e Dino, che sono stati i miei fedeli compagni durante tutte le sessioni d'esame di questi 3 anni. Grazie per le chiacchiere, gli squisiti piatti romagnoli e per i pranzi alle 11.40.

Un ringraziamento speciale va al mio amico Giulio: senza di lui probabilmente questo progetto non sarebbe mai venuto alla luce. Grazie di essere così ambizioso e di spronarmi costantemente a mettermi in gioco e a dare il massimo, grazie per avermi aiutata e sopportata in tutti questi mesi e grazie per le lunghissime giornate trascorse insieme.

E infine, ringrazio con tutto il mio cuore Francesco: il mio ragazzo, mio migliore amico e compagno di vita. Grazie di essere la persona che chiunque vorrebbe accanto, gran parte di ciò che ho ottenuto in questi anni lo devo a te e a ciò che abbiamo creato insieme.