Alma Mater Studiorum - Università di Bologna
Campus di Cesena

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DELLA COMPONENTE SERVER DI UN SISTEMA DI WAYFINDING

Tesi di Laurea in Programmazione ad Oggetti

Presentata da:
Giulio Bacchilega

Relatore:
Prof. Mirko Viroli

II sessione di laurea
Anno Accademico 2018 - 2019

Sommario

La disciplina del wayfinding rappresenta un campo di studio abbastanza complesso, il cui termine viene introdotto per la prima volta nella sfera architetturale e ingegneristica durante gli anni Sessanta da Kevin Lynch nel libro “The Image of City”, e utilizzato in seguito in tutti quei contesti in cui si ha la necessità di muoversi per poter trovare la destinazione desiderata. Il tema del wayfinding si basa fortemente sul concetto di “orientamento" all'interno di un qualsiasi spazio di cui si conoscono una quantità limitata di informazioni, che potrebbero ad esempio essere la posizione degli edifici, delle aree urbane, e sicuramente delle strade che li collegano.

Il professore Salvatore Zingale, docente di Semiotica presso il Politecnico di Milano, evidenzia in un'intervista le tipologie di luoghi in cui maggiormente si potrebbe necessitare di un supporto di wayfinding: spiega a tal scopo che sono svariate le strutture pubbliche o gli edifici la cui struttura non è immediatamente riconoscibile, e per i quali di conseguenza non è possibile in tempi brevi reperire i percorsi migliori relativi ad una destinazione che si vuole raggiungere.

Per tutte queste classi di ambienti, quindi, un sistema di wayfinding potrebbe sicuramente essere apprezzato e utilizzato da eventuali clienti e/o visitatori. Il presente lavoro consiste nello sviluppo della componente server di un sistema di navigazione, che possa consentire agli utenti che usufruiscono di tale servizio di potersi orientare all'interno di un ambiente privato e relativamente grande. L'applicazione è stata sviluppata dopo un'attenta valutazione di quelle che sarebbero potute essere le tecnologie e gli strumenti adatti allo sviluppo, cercando di soddisfare i requisiti progettuali e funzionali commissionati, e creando al tempo stesso un sistema estendibile e portabile.

Indice

Sviluppo della Componente Server di un Sistema di Wayfinding

3 ottobre 2018

Capitolo 1

Introduzione

Il lavoro presentato in questa tesi consiste nella realizzazione di un sistema di way-finding che possa supportare eventuali utenti che utilizzano l'applicazione a reperire la destinazione prestabilita attraverso la ricerca del percorso minimo, tra quelli disponibili all'interno dell'ambiente in cui essi si trovano.

Il software prende ispirazione dai già presenti sistemi di navigazione, quali ad esempio Google Maps, anche se la sua ideazione e realizzazione hanno lo scopo di fornire assitenza in ambienti interni privati, quali ad esempio aeroporti, grandi centri commerciali o edifici pubblici i quali non sono, come evidenziato dal professor Zingale durante un'intervista[1], immediatamente riconoscibili e interpretabili.

Tale applicazione è stata commissionata dal village resort di Varignana (Castel San Pietro - BO) all'azienda in cui è stata svolta l'attività di tirocinio, Vem Sistemi s.r.l., allo scopo di poter offrire ai propri clienti un sistema in grado di guidarli attraverso l'intera mappa del resort, senza bisogno di volantini o mappe cartacee.

I moderni sistemi di navigazione offrono una mappatura di percorsi e punti di interesse appartenenti ad ambienti pubblici e comprendenti strade provinciali, autostrade, sentieri e rotte commerciali. Sono tuttavia esclusi da tale servizio le zone interne, per le quali non esiste un comune strumento in grado di guidare utenti all'interno di tale ambiente: è stato quindi necessario fin da subito pensare ad una soluzione che consentisse, attraverso diversi layer, di sovrapporre a mappe preesistenti elementi grafici che potessero simulare archi e nodi di un grafo non orientato: una volta mappato il grafo, sarà poi possibile attraverso appositi tool e funzionalità inserire i punti di interesse raggiungibili, i quali verranno poi

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

presentati ai clienti che richiedono il servizio. L'utente avrà quindi la possibilità di inserire o ricercare la destinazione prescelta, allo stesso modo in cui potrebbe, tramite i servizi di wayfinding già esistenti, cercare la strada più breve per raggiungere un paese, un hotel o un edificio pubblico. Essendo poi il sistema sviluppato per essere potenzialmente distribuito a diversi gestori, viene posto come obiettivo aggiuntivo la realizzazione di una componente di editing, attraverso la quale l'amministratore di sistema avrà la possibilità di creare, sovrapponendolo alla piantina della struttura interessata, un grafo totalmente personalizzato e contenente i punti di interesse da lui stabiliti.

Un ulteriore requisito emerso è quello relativo alla gestione degli account privati di ogni amministratore: ogni account infatti, oltre ad essere accessibile secondo particolari criteri di sicurezza, deve consentire la visualizzazione delle sole mappe rappresentate dall'utente interessato, il mantenimento delle impostazioni di editing personalizzate e la gestione di icone o raffigurazioni grafiche scelte dall'utente per evidenziare i punti di interesse da lui aggiunti. Dopo una dettagliata fase di raccolta e analisi dei requisiti, è iniziata la progettazione del sistema, durante la quale sono stati messi in evidenza gli aspetti architetturali dell'applicazione, evidenziando la necessità di servirsi di un apposito database per il mantenimento dei dati, realizzare un server tramite il quale poter accedere e modificare il contenuto di tale database e sviluppare una componente client per la navigazione utente all'interno dei percorsi. L'intero processo di sviluppo è stato svolto in collaborazione con il tutor aziendale e con Linda Farneti, una collega universitaria: in particolare, riguardo alla prima fase di implementazione dell'editor, la progettazione è stata svolta parallelamente suddividendo il dominio in parte grafica e parte di modellazione, mentre successivamente la suddivisione ha coinvolto la creazione separata della parte client e la parte server. L'implementazione e la rappresentazione del modello del dominio e lo sviluppo della parte server-side dell'applicazione sono i temi trattati in questa tesi, mentre il lavoro trattato nella tesi di Linda Farneti[2] comprende la realizzazione della parte grafica iniziale e successivamente l'implementazione dell'applicazione client.

Per effettuare i test applicativi ai fini di stabilire la correttezza delle funzionalità implementate si è da subito deciso di utilizzare gli Unit Test, mentre per testare la correttezza delle chiamate API al server è stato utilizzato il software Postman, descritto nel capitolo relativo alle tecnologie utilizzate. I testi sono stati realizzati sia per controllare l'effettivo soddisfacimento dei requisiti e la corretta implementazione delle funzionalità, sia per poter tenere monitorati i costi e gli impatti di manutenzione perfettiva dovuti alla realizzazione

[2]L.Farneti - "Sviluppo della Componente Client di un Sistema di Wayfinding"

o aggiunta di nuove funzionalità al sistema.

Proseguendo nella lettura del documento, precisamente nel secondo capitolo, sono trattate le tecnologie utilizzate per la realizzazione del sistema e gli scenari applicativi, focalizzati principalmente sul tema del wayfinding. Successivamente nel quarto capitolo sono descritte le diverse fasi di progettazione del sistema, concentrandosi dapprima sull'architettura della componente amministrativa, per poi passare alla definizione strutturale della componente server.

Per quanto riguarda invece la parte di sviluppo, descritta nel quinto capitolo, sono presentati gli aspetti implementativi più rilevanti, assieme alle motivazioni che hanno portato a particolari decisioni riguardanti la realizzazione di alcuni requisiti non funzionali, come ad esempio il miglioramento delle performance e la realizzazione di un algoritmo euristico per la ricerca dei percorsi ottimi. Vengono poi descritte, nel capitolo finale, le metodologie con le quali è stata testata l'applicazione, focalizzandosi sulla loro copertura e sull'impatto valutativo ottenuto in seguito alla loro implementazione. La realizzazione del sistema trattato in questa tesi è stato portato a termine in seguito ad una prima fase di sviluppo, la quale è stata completata durante il periodo di tirocinio svoltosi nella stessa struttura lavorativa. Tale applicazione è stata poi riadattata e perfezionata in modo da poter comprendere al suo interno la logica client-server, potendola definire al termine dello sviluppo una web-app a tutti gli effetti. Come già accennato, il sistema tratta solamente degli aspetti riguardanti la parte amministrativa e server-side dell'intero sistema, dato che la parte client è stata sviluppata parallelamente da un altro componente, suddividendo il lavoro di progettazione e sviluppo.

Capitolo 2

Background

Il sistema è stato realizzato seguendo particolari direttive di progettazione e sviluppo dettate dai tutor aziendali o apprese durante il corso di studi. Inoltre sono state utilizzate diverse tecnologie e librerie per l'implementazione di ogni componente, scelte dopo una valutazione di quelli che precisamente fossero gli obiettivi, e optando poi per le soluzioni migliori e più efficaci in base al contesto. Sono in seguito elencate e descritte le tecnologie utilizzate, scelte sia per la necessità di trovare strumenti necessari al raggiungimento di un particolare scopo che per migliorare e rendere più comoda la fase di sviluppo e testing. Verranno poi messi in evidenza gli scenari applicativi di riferimento.

2.1 Linguaggi e Tecnologie di Sviluppo

In base ai requisiti dettati in fase di analisi e alle necessità evidenziate focalizzandosi sulla natura del sistema da creare, sono state scelte le seguenti soluzioni, che comprendono sia linguaggi di programmazione che l'utilizzo di particolari framework, i quali hanno consentito di semplificare notevolmente la fase di implementazione e creazione del sistema, per la loro interoperabilità e facilità di utilizzo.

2.1.1 Javascript

Dovendo realizzare un sistema che si collocasse all'interno del panorama delle web application, è stato ovviamente necessario valutare l'utilizzo di un linguaggio di programmazione che permettesse con relativa semplicità l'interazione e l'approccio alle diverse componenti presenti nel sistema. Queste necessitudini hanno portato alla decisione di realizzare la

parte di amministrazione servendosi di JavaScript, uno dei linguaggi di programmazione più conosciuti e utilizzati al mondo, impiegato principalmente per la realizzazione e sviluppo di pagine web client-side. Non si tratta di un vero e proprio linguaggio ad oggetti: infatti in JavaScript non esiste il concetto di classe o interfaccia e per simulare il modello di programmazione ad oggetti vengono utilizzate particolari funzioni (*Function Constructor*). Anche per quanto riguarda i meccanismi di ereditarietà, in JavaScript questi ultimi vengono simulati attraverso il concetto di *prototipo*, secondo il quale un oggetto contiene una proprietà *prototype* che punta ad un secondo oggetto, il quale punta a un terzo e così via fino a giungere all'oggetto base, definendo quella che viene chiamata Prototype Chain.

Ideato nel 1995 da Netscape e avente inizialmente il nome di LiveScript, questo linguaggio permise sin dall'inizio di aggiungere alle pagine HTML la possibilità di essere modificate a seconda delle diverse iterazioni dell'utente, in maniera totalmente dinamica e offrendo la possibilità di interoperare con i vari oggetti presenti nel documento, come immagini, form o link.

Dopo una prima fase di declino, dovuta alla concorrenza esterna di tecnologie come Flash o ActiveX, JavaScript tornò sulla cresta dell'onda con l'avvento della tecnologia Ajax, che forniva la possibilità di comunicare in maniera asincrona col server tramite opportuni script. A partire dallo standard ECMAScript 2015 in tale linguaggio è stato introdotto il costrutto *class*, il quale però rappresenta solamente una semplificazione dal punto di vista sintattico. JavaScript, all'interno dell'applicazione, è stato utilizzato prevalentemente per il mapping delle iterazioni tra utente e interfaccia grafica, e la definizione della conseguente architettura observer - observable, tramite la quale gli eventi generati da ogni iterazione sono mappati e interpretati dal Controller, secondo la logica MVC (Model - View - Controller), descritta nel capitolo relativo alla progettazione architetturale.

Trattandosi di un linguaggio di programmazione a tipizzazione dinamica debole ed essendo il controllo del tipo di una variabile effettuato a run-time, JavaScript è un linguaggio estremamente flessibile, ma probabilmente anche scomodo per certi punti di vista: è infatti più difficile stabilire la correttezza del codice se non a run-time, dato che alcune classi di errore sono evidenziate solamente in fase di esecuzione.

2.1.2 TypeScript

Il linguaggio di programmazione Typescript, ideato da Microsoft per supportare e facilitare lo sviluppo di applicazioni front-end, nonchè per garantire sufficiente robustezza e sicurezza applicativa, permette di seguire più intuitivamente la logica progettuale del

paradigma ad oggetti, consentendo quindi di creare interfacce e classi in maniera molto simile rispetto ad altri linguaggi Object-Oriented. Ogni file scritto in Typescript, per poter essere eseguito, deve essere convertito tramite appositi tool compilativi, chiamati transpiler, in codice JavaScript. Trattandosi di un'estensione di quest'ultimo, si può con certezza affermare che una qualsiasi applicazione scritta in JavaScript è considerabile anche un'applicazione TypeScript: di conseguenza, la conversione di un file *.js* in un file *.ts* può essere fatta gradualmente e in maniera totalmente trasparente rispetto al compilatore.

Rispetto a JavaScript, ciò che viene aggiunto nella sintassi di TypeScript sono le classi, le interfacce, i moduli, le enumerazioni e i tipi di dato, i quali però sono considerabili opzionali, dato che in JS non esistono. Utilizzando questo linguaggio,le problematiche che emergono nell'utilizzo di JavaScript vengono minimizzate: il controllo dei tipi viene eseguito in fase di compilazione, limitando la flessibilità e il tasso di errore, mentre il sistema opzionale delle annotazioni di questi ultimi consente di specificare in maniera esplicita il tipo di una variabile, col risultato di ottenere un codice più leggibile e comprensibile.

2.1.3 API Rest

Dovendo il server gestire numerosi richieste, provenienti sia dall'applicazione client che dalla componente amministrativa, si è da subito stabilito di dover trovare una soluzione portabile, semplice ed estendibile che consentisse di raccogliere e organizzare al meglio tutta la logica relativa a tali richieste. Tramite l'utilizzo di API (*Application Programming Interface*), è possibile creare ed organizzare tutte quelle che saranno le richieste effettuate dal client verso il server. L'applicativo infatti deve consentire, lato admin, di poter interagire col database, ad esempio nel caso in cui si volesse aggiungere una nuova mappa o modificarne una già esistente.

Sarà invece necessario, per soddisfare le richieste di ogni utente, creare interfacce funzionali che permettano di ricevere informazioni riguardanti l'ambiente che lo circonda, in modo da poterlo guidare verso la destinazione da lui scelta.

Queste API, quindi, non sono altro che un insieme di tecniche che consentono di collegarsi ad un server esterno e eseguire alcune operazioni sui dati che esso contiene.

Per REST (*REpresentational State Transfer*), invece, si intende una serie di linee guida e di approcci che definiscono lo stile con cui i dati sono trasmessi dal client al server, o viceversa. Spesso questo insieme di chiamate che rispettano i criteri e la logica imposta da REST viene raccolto in una collezione, definita come un set di RESTful API. In realtà definire un vero e proprio standard per la definizione delle interazioni client-server non è per

niente facile, ma è possibile seguire quelle che sono le principali linee guida e best-practices utilizzate dalla maggior parte degli sviluppatori, le quali definiscono i seguenti concetti:

- Occorre che ci sia una determinata consistenza e uno specifico compito per ogni API che viene definita;
- Utilizzando questo tipo di API non si è vincolati nel creare sessioni all'interno dei server, definendo applicazioni stateless dove il server non salva informazioni di login tra un'operazione e l'altra;
- Possono essere utilizzati status code HTTP che permettono di velocizzare le risposte del server;

Le API, se ben definite, consentono quindi di effettuare una qualsiasi delle operazioni di GET, POST, PUT o DELETE in maniera trasparente rispetto al database sul quale vengono inviate. All'interno dell'applicazione, sono state inserite delle API per tutte le operazioni di editing delle mappe, oltre che per la restituzione dei dati utili al client nel momento della ricerca del percorso punto-punto.

2.2 Ambiente .NET

La tecnologia lanciata sul mercato da Microsoft denominata .NET (*dot NET*) unisce all'interno dello stesso ambiente svariati supporti per la programmazione di diversi processi informativi, che siano essi applicativi Web, mobile app o WUI(*Windows User Interface*).

Tramite .NET è quindi possibile far interagire tra loro applicazioni e software di diversa struttura e tecnologia, come ad esempio un'applicazione stand-alone e una Web App, che nel caso in cui fosse presente un database condiviso potrebbero fornire informazioni anche a dispositivi mobili. Il livello di astrazione e trasparenza offerto dalla piattaforma .NET consente inoltre di utilizzare diversi tipi di linguaggi a seconda delle esigenze, potendo associare ad ogni scenario applicativo la corretta sintassi e metodologia di sviluppo. L'interoperabilità e le modalità di interfacciamento delle tecnologie .NET, semplici e funzionali, sono state di fondamentale importanza per la realizzazione delle diverse componenti del sistema. Tali tecnologie sono di seguito elencate.

2.2.1 Microsoft Visual Studio Code

Editor di codice ideato da Microsoft con supporto di debugging integrato e in grado di supportare diversi linguaggi di programmazione dell'ambiente .NET. Il progetto è stato inizialmente svilupato in questo editor, compilando i file sorgenti attraverso il pacchetto *npm* della piattaforma event-driven Node.js. Questo editor è disponibile anche per sistemi operativi diversi da quelli di casa Microsoft, quali ad esempio Linux o Mac.

2.2.2 Microsoft Visual Studio Enterprise

Ambiente di sviluppo integrato multipiattaforma che permette la realizzazione di applicazioni, siti e servizi web. La versione utilizzata per lo sviluppo del sistema è Visual Studio 2017 - Enterprise. Questo IDE è stato utilizzato per la l'implementazione della parte server dell'applicazione e per effettuare i successivi Unit Test.

2.2.3 Microsoft SQL Server Management Studio

Microsoft SQL Server è un DBMS relazionale utilizzato per lo sviluppo di database di qualsiasi dimensione. Permette di eseguire query e progettare database e datawarehouse sia in locale che sul cloud. Le informazioni che sono state gestite a livello di database sono la memorizzazione delle mappe create, oltre che naturalmente quelle riguardanti i punti di interesse e i percorsi da cui erano collegati. Le API definite nella componente server-side consentono perciò di interagire direttamente con il database a cui il software fa riferimento.

2.2.4 Linq Framework

Acronimo di "Language Integrated Query", Linq rappresenta una delle novità del framework .NET e consente, indipendentemente dall'architettura e dalle strutture dati alle quali si vuole accedere di eseguire query su uno specifico database. Con questa tecnologia è possibile manipolare diverse tipologie di dati reperiti da varie fonti, come file di testo, file XML, file Excel, array e qualsiasi altro tipo di oggetto enumerabile.

Tramite apposite keyword è possibile manipolare strutture dati complesse, similmente agli stream forniti dal linguaggio Java, senza dover ricorrere a query testuali.

Adottando questo framework per interrogare il database, è stata riscontrata una notevole comodità di utilizzo, data dall'estrema semplicità di composizione delle query e dalla loro intuitività. I meccanismi di astrazione e forniti da questo framework e la generalità

delle Linq API hanno permesso la creazione di query utilizzate sia per l'interazione con il database SQL che per manipolare strutture dati "mock", create appositamente per lanciare i primi test applicativi.

2.2.5 Razor

Razor è una view engine sviluppata per l'ambiente .NET che permette di visualizzare pagine web dinamicamente, inserendo blocchi di codice scritti in C# o Visual Basic all'interno del codice HTML, il modo tale che il server lo possa interpretare ed eseguire. La sintassi da utilizzare è piuttosto semplice, siccome il cambio di linguaggio all'interno del documento non deve essere specificato. Questa caratteristica consente di dichiarare variabili e costrutti di iterazione in maniera piuttosto semplice, in modo da poter rendere la pagina web più interattiva con relativa semplicità.

2.3 Tecnologie Web

2.3.1 Postman

Postman[1] è un'applicazione di Google Chrome che permette di effettuare chiamate API senza dover mettere mano al codice sorgente. Il software potrebbe essere integrato con vari plug-in, ad esempio:

- *Newman*: utile per l'automatizzazione dei test;

- *Interceptor*: utile per la personalizzazione degli header e la gestione più semplice dei cookies e delle richieste inviate dal browser;

Tale applicazione è stata utilizzata per effettuare test di debug sulle chiamate API: il software permette infatti di simulare le chiamate al server di tipo GET, POST, PUT e DELETE e visualizzare i dati restituiti in formato JSON o XML. Questa applicazione è stata ritenuta piuttosto utile per il testing delle API, in quanto in precedenza interpretare la risposta del server poteva essere piuttosto difficoltoso.

2.3.2 Bitbucket

Nato inizialmente come una start-up indipendente e inseguito acquistato da Atlassian, Bitbucket[2] è un servizio di hosting per progetti che utilizzano sistemi di controllo versione

[1] `https://www.getpostman.com`
[2] `https://www.openstreetmap.org`

distribuiti come Mercurial o Git. Tramite questa piattaforma virtuale è stato possibile creare e implementare passo a passo l'applicazione, tenendo sotto controllo le modifiche effettuate volta per volta e poter di conseguenza stabilire con più facilità su quale via di sviluppo fosse più conveniente concentrarsi.

2.3.3 Open Street Map

Molto spesso le mappe reperibili on-line non possono essere personalizzato o utilizzate per scopi commerciali perchè coperte da copyright. Per questo e altri motivi nel 2004 nasce Open Street Map (OSM)[3] che ben presto diventa il più grande progetto di raccolta di dati geografici liberi. Giornalmente, centinaia di volontari sparsi nel mondo aggiungono milioni di dati al progetto OSM, come informazioni riguardanti aree amministrative, strade, conformazioni fisiche del territorio, ma anche dati più dettagliati come curve di livello, edifici e numeri civici. Le informazioni sono in continuo aumento, come del resto anche i volontari che contribuiscono ad arricchire il progetto. I dati possono essere presentati in diverse modalità, potendo ad esempio scegliere tra la visualizzazione standard, quelle della rete del trasporto pubblico oppure una configurazione una più pulita e con meno elementi. La banca dati può essere acceduta per svariati motivi, e gli strumenti messi a disposizione dal team OSM, come in questo caso, possono essere utilizzati anche all'interno di applicazioni.

2.3.4 Google Maps

Servizio accessibile dal relativo sito web che permette di visualizzare carte geografiche di buona parte del Pianeta e ricercare luogh il percorso dal punto di origine, dal quale poter poi raggiungere la meta previssata grazie al supporto per la navigazione. L'idea del progetto spiegato in questa tesi deriva proprio da questo software: si è cercato di ottenere in maniera più modesta gli stessi risultati,con una differenza sostanziale: le mappe dei luoghi saranno piantine rese disponibili dal servizio OSM (Open Street Map), e sarà compito dell'utente creare in un layer sovrapposto un grafo, che con l'aiuto di opportune features potrà essere editabile e navigabile. GoogleMaps[4] è stato utilizzato in alcuni scenari applicativi, per poter confrontare se un percorso minimo trovato da questo sistema corrispondesse a quello calcolato all'interno dell'applicazione.

[3] `https://bitbucket.org`
[4] `https://www.google.com/maps`

2.4 Librerie

2.4.1 JQuery

Allo scopo di semplificare la sintassi del codice JavaScript all'interno di un progetto, è stata utilizzata la libreria JQuery[5], ideata appositamente per applicazioni web. Nasce con l'obiettivo di facilitare la gestione di eventi e l'iterazione con i vari elementi della pagina html, usufruire più semplicemente delle funzionalità di Ajax e rendere più intuitiva e veloce la manipolazione degli oggetti della pagina web, offrendo la possibilità di modificarne lo stile e tracciare eventi scatenati dalle interazioni con questi ultimi. L'architettura e il pattern observer per la definizione del comportamento dell'applicazione in seguito alle azioni dell'utente sulla GUI sono stati definiti appoggiandosi proprio a questa libreria.

2.4.2 Leaflet

Per la rappresentazione delle mappe e per l'inserimento degli elementi principali, quali nodi, strade e punti di interesse, è stata utilizzata in typescript una opportuna libreria grafica, chiamata Leaflet[6].

Tale libreria permette infatti lo sviluppo di mappe geografiche interattive, da poter poi integrare con marker e layers, da aggiungere in seguito. Sviluppando all'interno del sistema la parte amministrativa, la libreria è stata utilizzata per poter permettere agli amministratori di sistema di aggiungere all'occorrenza nuove mappe e/o modificarne alcune già esistenti.

Ciò è stato reso possibile dal fatto che Leaflet permette di mostrare punti di interesse, marker, linee o figure geometriche con file GeoJSON, servendosi di particolari mappe a tasselli.

2.5 Scenario

2.5.1 Wayfinding

Il tema del wayfinding viene adottato ogniqualvolta le persone necessitino di orientarsi in uno spazio fisico, per reperire il percorso che le possa portare alla destinazione prescelta.

[5] `https://jquery.com`
[6] `https://leafletjs.com/`

Questo processo che coinvolge letteralmente la "ricerca del percorso", può articolarsi in quattro stadi:

- **Orientamento**: tentativo di determinare il punto in cui ci si trova in base all'ambiente circostante e poter di conseguenza immaginare come raggiungere la destinazione;
- **Scelta del percorso**: decisione su quali direzioni seguire e quali punti attraversare per raggiungere la destinazione;
- **Controllo del percorso**: verifica, proseguendo lungo il percorso scelto, se tale soluzione può effettivamente portare alla destinazione finale;
- **Raggiungimento della destinazione**: ultima fase, che consiste nell'identificare la destinazione che si voleva raggiungere e terminare la ricerca;

Ognuno di questi passi, per quanto riguarda la creazione di un'applicazione o sistema informativo, dovrà essere prevalentemente a carico del sistema, risparmiando all'utente che beneficia del servizio un numero di operazioni relativamente ampio e in alcuni casi piuttosto stressanti e impegnative. Dato un grafo, perciò, è necessario utilizzare un apposito algoritmo di ricerca che possa condurre l'utente alla destinazione preimpostata e guidarlo seguendo il percorso più veloce, inteso come quello la cui somma dei pesi degli archi che lo compongono è la minore tra tutte quelle disponibili.

2.5.2 Djikstra e A*

Proposto nel 1956 da Edsger Wybe Dijkstra, l'omonimo algoritmo risolve in maniera elegante il problema della ricerca del cammino minimo in un grafo orientato e pesato. L'algoritmo trova tutt'ora svariate applicazioni nel mondo reale, ad esempio nell'ottimizzazione di reti idriche, stradali o di telecomunicazioni. Il costo computazionale dipende dal numero di nodi e di archi presenti nel grafo, e può essere espresso come

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

ma se si assume che

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

è possibile approssimare il costo con

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

L'idea generale dell'algoritmo è la seguente:

- Ogni nodo ha un costo iniziale pari a $+\infty$;
- Il nodo di partenza ha un costo iniziale pari a 0;
- Ad ogni iterazione viene considerato il nodo tale per cui il suo raggiungimento ha il costo minore rispetto ai rimanenti nodi adiacenti;
- Il costo per il raggiungimento di un nodo è dato dalla somma del costo del nodo precedente lungo il percorso minimo più il costo del collegamento con tale nodo;
- Non vengono aggiornati i nodi presenti all'interno dell'insieme ottimo;
- Il costo per giungere ad un nodo, se conosciuto, è il minor costo per il raggiungimento di tale nodo;
- Nel momento in cui si aggiorna il costo di un nodo, si prende sempre quello minore;

L'idea dell'algoritmo di Dijkstra è anche alla base dei protocolli principali di routing, ad esempio:

- *OSPF (Open Shortest Path First)*: i protocolli OSPF necessitano di conoscere approfonditamente la rete su cui operano, e possono "esplorarla" iterando l'algoritmo di Dijkstra, gestendo anche casi in cui ci sono cammini con costi molto diversi verso la destinazione;
- *IS-IS (Intermediate System to Intermediate System)*: che procede in modo simile, fa sì che i router inondino la rete con informazioni link state. Ogni router accresce le informazioni ricevute aggiungendovi le proprie, portando così alla creazione di un database contenente tutti i percorsi possibili, e potendo successivamente adottare svariate strategia algoritmiche per stabilire quale tra essi sia il cammino ottimo;

Nel caso in cui si disponesse di grafi di grandi dimensioni, il problema della ricerca del cammino minimo, per essere risolto, potrebbe richiedere una grande quantità di tempo e risorse con la semplice implementazione dell'algoritmo.

A questo punto può esswere presa in considerazione l'idea di adottare meccanismi di euristica per il miglioramento computazionale: l'algoritmo A* prevede, sulla base di Dijkstra, di considerare i percorsi che sembrano costare meno, secondo una stima euristica, il cui procedimento di calcolo sarà anche il fulcro della complessità risultante.

Capitolo 3

Requisiti

Come nella maggiorparte dei procesi di progettazione, lo sviluppo dell'applicazione è iniziato con una stesura di un documento dei requisiti. Per "requisito" si intede una proprietà richiesta che deve essere presente nel prodotto finito, ossia una funzionalità desiderata dal committente.

Ogni requisito potrebbe esprimere una proprietà obbligatoria oppure solamente auspicabile ma non strettamente necessaria per determinare la garanzia del prodotto finale.

Per questo sistema, l'analisi dei requisiti è stata scomposta in due fasi:

- **Analisi degli utenti**: definizione della categoria di utenti che potrebbero beneficiare del servizio. Tale analisi consente di poter più facilmente ideare e progettare l'interfaccia grafica e definirne i criteri di usabilità. Per quanto concerne questa applicazione, l'interfaccia grafica sarà suddivisa in due componenti principali: la parte amministrativa a la parte applicativa lato client, la quale rispetto alla prima avrà maggior necessità di essere intuitiva e usabile. Per quanto riguarda invece la GUI amministrativa, che per sua natura è più complessa, si è cercato di rappresentare i comandi in maniera esplicativa e di mantenerne solamente un numero ridotto, in modo da semplificarne l'utilizzo e migliorare anche in questo caso la user-experience;

- **Analisi dei contesti d'uso**: definizione dei diversi contesti d'uso del prodotto in base alla diversa categoria di utenti. In particolare è stata da subito evidenziata una netta distinzione tra le componenti relative alla parte amministrativa e quelle orientata ai clienti che dovranno beneficiare del servizio;

Ciò che è emerso da tale analisi consiste nello sviluppo di un sistema in cui fosse possibile creare e manipolare mappe composte da punti di interesse e percorsi che li collegano,

potendo operare sia in ambienti indoor che outdoor, e consentendo ad eventuali utenti di poter reperire il percorso più veloce tra la posizione originale e la destinazione da questi ultimi scelta.

Sarà presente poi un sistema di navigazione che in base alle coordinate GPS potrà guidare l'utente durante il tragitto, similmente al servizio offerto dagli attuali navigatori satellitari.

3.1 Requisiti funzionali

I requisiti funzionali all'interno di un'applicazione descrivono gli aspetti comportamentali che devono essere presenti nell'applicazione e che evidenziano ciò che il sistema deve offrire e come si deve comportare, in casi particolari o semplicemente in seguito a semplici input da parte dell'utente. Vengono definiti requisiti funzionali anche l'insieme dei vincoli che l'applicazione deve rispettare, sia in fase di sviluppo che in fase di operatività. All'interno del progetto, precisamente focalizzandosi sulla parte di modellazione e server-side, i requisiti funzionali che sono stati messi in evidenza sono i seguenti:

Content Management System
Componente del sistema che consentirà la modifica dei dati presenti sul database, l'editing di nuove mappe e l'aggiunta di nuovi percorsi e punti di interesse a mappe già esistenti. Il CMS rappresenta da questo lato l'intera componente amministrativa del sistema, con associata la relativa interfaccia grafica.

Database System
Il sistema deve poter mantenere i diversi dati relativi ai punti di interesse, alle strade e ai nodi delle varie mappe. Tramite opportuni controlli e meccanismi di interazione il database verrà aggiornato ogniqualvolta l'utente esegue operazioni di editing o modifica dei contenuti già presenti.

I dati contenuti nel database, all'occorrenza, dovranno essere disponibili sia per il supporto alla navigazione lato client, mantenendo informazioni relative ad ogni punto di interesse presente nella mappa, che per il salvataggio e la storicizzazione dei dati caricati dagli utenti che amministreranno il sistema.

Deve inoltre essere possibile la gestione di più amministratori, ognuno dei quali potrà creare le proprie mappe e personalizzare le impostazioni di editing. Sarà quindi necessario registrare ogni utente all'interno del database tramite opportuni account, con password criptate per garantire un certo livello di sicurezza.

Supporto alla Navigazione

Il sistema dovrà essere in grado di fornire all'applicazione client l'insieme delle informazioni necessarie per il raggiungimento della destinazione prestabilita. A tal scopo, è necessario che un apposito algoritmo trovi all'interno del grafo pesato e precedentemente creato il percorso ottimale punto-punto, restituendo una struttura dati che contenga i nodi da attraversare, oppure le strade da percorrere. Tale sistema dovrà consentire agli amministratori di poter controllare quali percorsi sono disponibili per giungere alle varie destinazioni inserite nella mappa, e constatare di conseguenza quali saranno i cammini ottimi che verranno presentati agli utenti finali.

3.2 Requisiti non funzionali

I requisiti non funzionali di un sistema costituiscono l'insieme delle funzionalità e caratteristiche che non sono esplicitamente richieste dal committente, ma che potrebbero essere prese in considerazione per il miglioramento del prodotto, in termini di performance e manutenibiltià del sistema.

Ottimizzazione dell'algoritmo di ricerca

Essendo il funzionamento dell'algoritmo di ricerca scelto per il calcolo del percorso minimo piuttosto semplice e basilare, considerando di adottare tecniche e strategie di euristica è possibile migliorarlo cercando di ottenere i medesimi risultati cercando di equilibrare ottimizzazione, completezza, accuratezza e velocità. Gli algoritmi euristici vengono psesso utilizzati per la risoluzione di tutti quei problemi troppo complessi da essere risolti con i metodi classici: il risultato che si ottiene attraverso calcoli matematici viene ottenuto più velocemente e costituisce una soluzione più o meno vicina a quella ottima. Nel caso dell'algoritmo di ricerca impiegato, l'euristica comporterebbe una stima del costo tra il punto di origine e il punto di destinazione, ottenendo nel caso peggiore il costo standard dell'algoritmo Dijkstra. L'algoritmo A* non è tuttavia un algoritmo *greedy* dato che i risultati conseguiti non dipendono esclusivamente dai parametri euristici adottati.

Miglioramento Performance

Algoritmi di parallelizzazione e particolari strategie di storicizzazione si rivelano spesso utili per quanto riguarda il miglioramento delle performance e l'ottimizzazione dei tempi computazionali. La distribuzione delle richieste e del calcolo a più processori o macchine

indipendenti richiede però di essere considerata solo dopo un'attenta analisi, durante la quale devono essere valutati aspetti determinanti, quali i costi di manutenzione perfettiva e adattativa e la complessità aggiunta al sistema, da paragonare con i benefici apportati in seguito all'adozione di queste tecniche. L'analisi dovrà incentrarsi perciò sul numero di utenti mediamente attivi contemporaneamente, sulla dimensione dei grafi sui quali calcolare i percorsi e sul numero di volte che uno stesso percorso potrebbe essere scelto in un lasso di tempo relativamente breve.

Usabilità dell'Interfaccia Utente

Uno degli aspetti richiesti e messi in evidenza durante la fase di analisi dei requisiti riguarda l'usabilità e l'intuitività che devono caratterizzare l'interfaccia utente: la GUI dovrà apparire semplice, essenziale e soprattutto intuitiva. I comandi messi a disposizione per l'editing verranno quindi ideati appositamente per essere funzionali ed autoesplicativi. In generale, l'interfaccia grafica che si presenterà dovrà avere le seguenti caratteristiche:

- L'utente potrà utilizzare facilmente il servizio anche dopo svariato tempo, senza dover riprendere da alcun tutorial;
- I comandi dovranno essere tali da permettere di eseguire ogni tipo di operazione in tempi relativamente brevi;
- L'utente dovrà essere in grado di svolgere le principali operazioni base in tempi di apprendimento abbastanza brevi;
- Il tasso di errore dovuto allo svolgimento delle operazion di editing dovrà essere minimizzato. Ciò è possibile scegliendo di inserire popup o messaggi che indichino all'utente cosa effettivamente sta accadendo all'interno della applicazione e lo renda partecipe dei risultati conseguiti;
- La GUI dovrà inoltre essere il più possibile piacevole da utilizzare;

A tali caratteristiche vanno poi aggiunti i requisiti collegati alla parte di interfaccia grafica, che potrà essere migliorata aumentando l'usabilità e la facilità di utilizzo della stessa, come viene descritto in seguito.

Sarà possibile inserire e rimuovere elementi grafici, quali nodi e archi attraverso gli opportuni comandi, inserire icone simboliche per la rappresentazione figurata di un luogo e aggiungere una descrizione testuale, la quale sarà poi messa a disposizione dell'utente.

Capitolo 4

Design Architetturale

Vengono di seguito trattati aspetti riguardanti la progettazione architetturale del sistema e lo sviluppo delle singole parti che lo compongono. Partendo dalla progettazione dell'interfaccia grafica, il cui aspetto e la presentazione delle diverse componenti della parte amministrativa sono state studiate e realizzate basandosi sulle linee guida prodotte grazie all'analisi dei requisiti, si procede con aspetti puramente architetturali, focalizzandosi sulla struttura e tecniche di interazione delle varie componenti del sistema. Infine, si analizzeranno le strategie di sviluppo adottate per la realizzazione delle componenti precedentemente identificate nella fase di progettazione.

4.1 Interazioni tra le Componenti Architetturali

Per poter realizzare un sistema che comprenda le funzionalità e i meccanismi riportati in seguito all'analisi dei requisiti è stato necessario pensare a quali componenti fisiche inserire all'interno dello schema architetturale del sistema. Essendo il sistema realizzato in una web-app, da presentare agli utenti e che fosse fornita da un'entità separata, si è da subito stabilito che le due linee di sviluppo all'interno del team comprendessero le due seguenti figure:

– **Client**: la componente client del sistema avrà lo scopo ben preciso di presentare all'utente le mappe fornite dal server e di guidarlo all'interno del percorso punto-punto stabilito. Il supporto per la visualizzazione della web-app sarà un qualsiasi dispoditivo mobile, ad esempio smartphone o tablet con sistema operativo Android, secondo quanto specificato in seguito alla commissione;

– **Server**: il server, di cui verrà trattato in questa tesi, avrà invece il compito di fornire dati e strutture su richiesta del client. La componente di amministrazione potrà appoggiarsi al server per effettuare tutte le chiamate necessarie al popolamento e modifica delle mappe create;

In seguito alla definizione delle due entità citate sopra, si è reso necessario pensare anche ad un supporto di storicizzazione dei dati creati dagli amministratori: a questo scopo è stata aggiunto un database nel quale registrare tutte le strutture dati che potessero essere modificate o richieste dalla componente amministrativa o client.

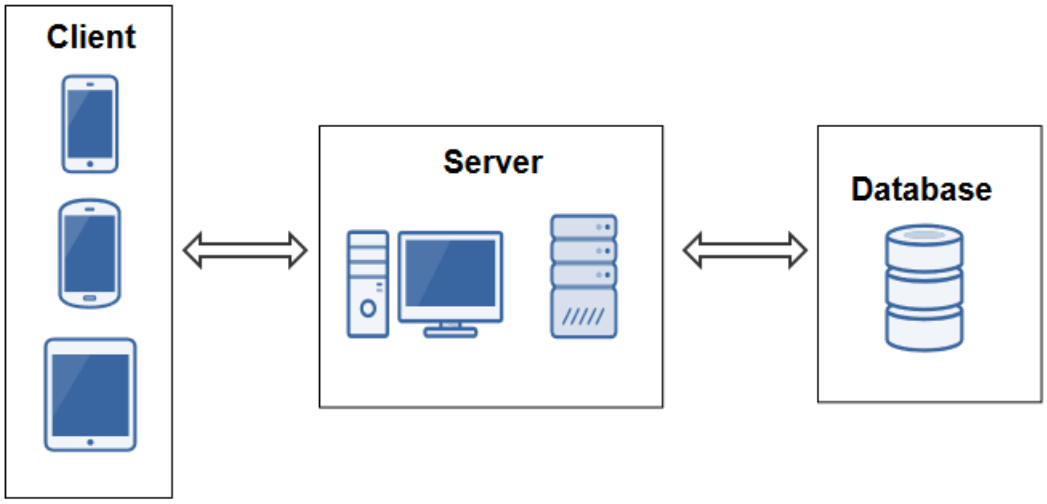

Figura 4.1: Componenti del sistema

Il server implementerà chiamate API indirizzate al server, utilizzabili anche per il client che serviranno a fornire i dati utili su richiesta. Altre chiamate saranno utilizzate per la modifica de dati storicizzati.

4.2 Design dell'Interfaccia Grafica

4.2.1 Login

Come prima pagina iniziale l'applicativo sarà caratterizzato da una schermata di login, all'interno della quale i nuovi amministratori possono registrarsi all'interno del sistema oppure accedere alla propria pagina privata. La suddivisione dell'applicazione e la gestione interna di più account si è resa necessaria siccome ogni amministratore o organizzazione

avrebbero dovuto mantenere e gestire le mappe dei propri ambienti privati, potendo inserire icone da loro scaricate e interagire con la mappa in base alla personalizzazione delle opzioni di settings.

Per questi motivi si è pensato di adattare il database per andare incontro a queste esigenze. Dovendo gestire la registrazione e l'inserimento di account con password e nome utente, è stato necessario anche mettere in piedi strategie che consentissero di crittografare le password inserite, in modo che sul database venissero salvate solamente stringhe a caratteri cifrati.

4.2.2 Get Started

La parte amministrativa avrà come punto di snodo centrale la pagina "GetStarted", dalla quale sarà possibile scegliere le operazioni da effettuare all'interno dell'applicazione. Non essendo la parte server-side così complessa e articolata dal punto di vista dell'interfaccia, la pagina principale metterà a disposizione 3 opzioni principali per la navigazione e l'interazione interna all'applicativo, ovvero:

- possibilità di creare una nuova mappa settando i parametri fondamentali come latitudine, longitudine e zoom;
- navigare tra le mappe già create e caricarne una per poter procedere alla sua modifica o cancellazione;
- possibilità di configurare alcuni parametri dell'editor e manipolare le icone, caricandone di nuove da poter poi inserire nelle mappe come punti di interesse o rimuovendone alcune già esistenti nel caso non dovessero più servire o dovessero essere aggiornate.

Si era inizialmente ideato un layout simile al seguente per quanto riguardava la pagina principale: Come accennato in precedenza, tramite i tre pannelli sarà possibile svolgere le diverse operazioni per l'editing e la modifica dellle mappe:

- **New Map:** tramite questo comando si entrerà nella modalità di creazione di una nuova mappa. La configurazione iniziale prevede che la mappa presentata all'utente sia quella relativa alla penisola italiana, ma con i vari comandi di interazione l'utente potrà localizzare un punto specifico sopra al quale creare il grafo;
- **Maps:** tale pannello aprirà una schermata in cui saranno presenti tutte le mappe create dall'account con il quale ci si è loggati all'interno del sistema, con relativa

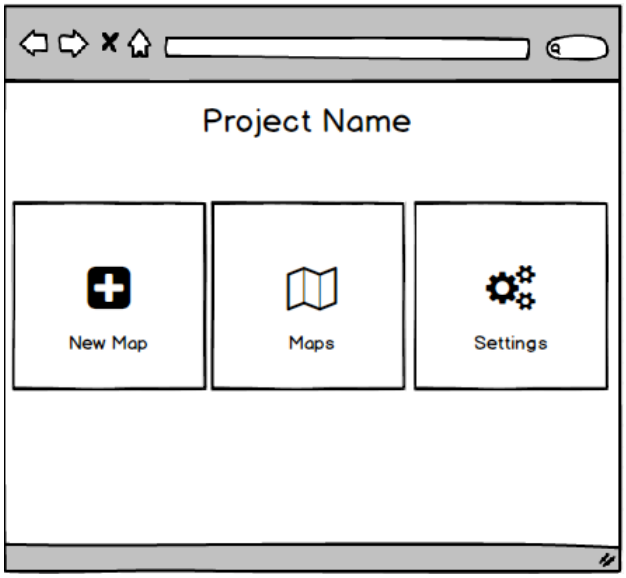

Figura 4.2: Pagina iniziale - GetStarted

anteprima. Selezionando una delle mappe presenti, questa verrà caricata e l'utente verrà indirizzato nella scheramata di editing, dalla quale potrà modificare o eliminare la mappa selezionata;

– **Settings:** il terzo e ultimo comando permetterà all'utente di configurare parametri utili all'editing delle mappe, come ad esempio la possibilità di caricare nuove icone da inserire come punti di interesse e di selezionare diversi colori e stili per la rappresentazione degli elementi all'interno dell'editor. La possibilità di selezionare i colori relativi alla rappresentazione e selezione di strade e nodi costituisce un secondo metodo di configurazione: tale funzionalità è stata aggiunta in modo tale che in base al background delle diverse mappe, gli elementi grafici possano essere comunque ben visibili e distinti dal layer OSM sottostante.

4.2.3 New Map

Tramite il primo pannello della pagina "Get Started" sarà possibile entrare nella modalità di creazione di una nuova, mappa da poter in seguito utilizzare per mappare un grafo inserendo nodi, archi e punti di interesse. Tale pagina si presenterà all'incirca nella forma seguente:

Figura 4.3: Creazione di una nuova mappa - New map

Tramite il drag del mouse e la mouse wheel i parametri relativi allo zoom, alla latitudine e alla longitudine verranno settati in maniera automatica, in modo tale da evitare che tale compito sia a carico dell'utente. Questi parametri verranno poi storicizzati al momento del salvataggio e in seguito recuperati una volta che occorrerà caricare la mappa per il conseguente editing. L'utente dovrà quindi posizionarsi nel punto della mappa prescelto e in seguito digitare un nome con il quale potrà essere salvata la nuova mappa.

4.2.4 Maps

Il pannello "Maps" permetterà all'utente di visualizzare tutte le mappe fino a quel momento create, che saranno presentate attraverso un nome e un'anteprima. Selezionando una di queste mappe, si entrerà nella pagina di editing dalla quale l'utente potrà appportare modifiche alla mappa caricata. Tale pagina avra un aspetto simile al seguente:

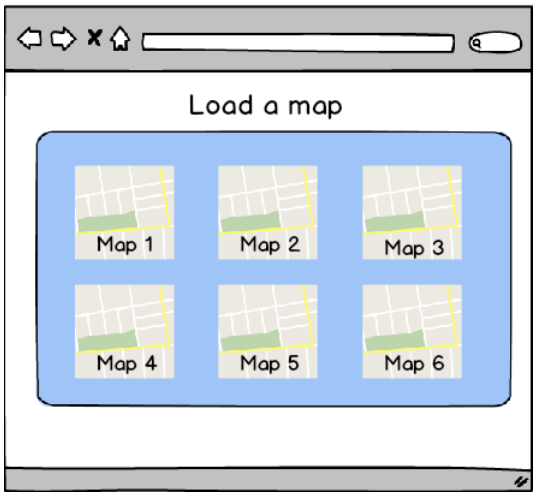

Figura 4.4: Caricamento di una mappa precedentemente creata - LoadMap

4.2.5 Settings

L'interfaccia, piuttosto semplice e facile da utilizzare, si presenterà nel modo seguente:

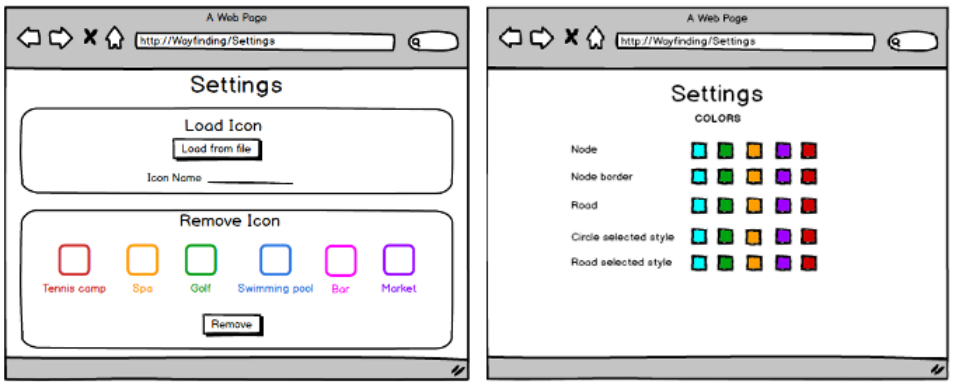

Figura 4.5: Setting dei parametri e manipolazione delle icone - Settings

Il pannello "Settings" permette all'amministratore di caricare nuove icone o rimovere quelle esistenti, selezionandole e in seguito premendo il pulsate "Remove". Tramite questo

pannello sarà inoltre possibile, per l'utente, settare i diversi colori per la rappresentazione grafica degli elementi della mappa, in modo da poter scegliere sempre la combinazione di colori più visibile per una certa mappa di sfondo.

4.2.6 Editor

La parte più consistente dell'interfaccia amministrativa è rappresentata dalla parte di editing della mappa: le funzionalità implementate, che permettono all'utente di creare grafi in un layer superiore a quello rappresentato dalla mappa, inserire punti di interesse per gli utenti e controllare percorsi ottimi tra le strade disegnate, sono concretizzate all'interno di una menù bar, la quale è stata disegnata appositamente per essere intuitiva ed efficace. Inizialmente, da una prima analisi dei requisiti riguardanti la progettazione dell'interfaccia di editing, è emerso che la GUI amministrativa dovesse avere un aspetto e un insieme di controlli e comandi simili a quelli presentati nella seguente bozza:

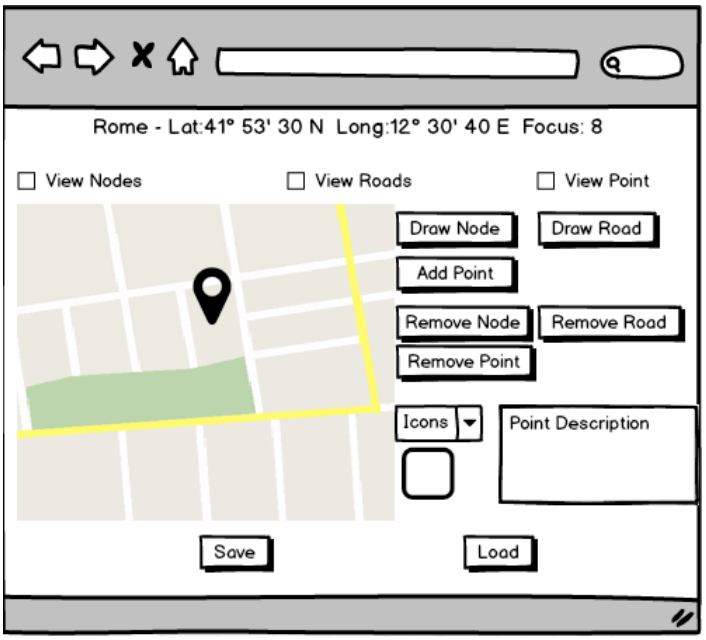

Figura 4.6: Interfaccia grafica per l'editing delle mappe

Successivamente, dopo aver creato tale interfaccia e avere testato il funzionamento dei diversi comandi disponibili, è iniziato un procedimento di refactoring dei comandi e perfezionamento GUI, il quale è stato intrapreso allo scopo di creare un'interfaccia più essenziale, intuitiva e funzionale. Da un procedimento di analisi iterativo è stata poi ideata e rappresentata un'interfaccia che fosse basata sul seguente mock-up:

Figura 4.7: Interfaccia finale per l'editing delle mappe

Come accennato in precedenza, la menù bar si compone dei seguenti controlli:

- **OpenEditor / CloseEditor**: comando che permette di aprire un pannello per l'editing delle strade e delle icone: selezionando una strada si potrà modificare la relativa larghezza, così come per le icone. Per queste ultime inoltre sarà possibile anche aggiungere o modificare la relativa descrizione, che sarà presentata poi all'utente che vorrà giungere a tale destinazione. In questo pannello sono poi presentate le icone attualmente presenti nell'applicativo, le quali potranno essere inserite in sovrapposizione dei nodi creati e presenti nella mappa;

- **View**: insieme di checkbox che permettono di scegliere se visualizzare o meno gli elementi grafici della mappa, quali icone, strade e nodi;

- **Draw Node**: pulsante che permette di passare in modalità "Draw Node": in questo stato, ogni click sulla mappa porterà alla creazione di un nuovo nodo;

- **Draw Road**: alla pressione di tale pulsante, nel caso siano stati selezionati due o più nodi, verranno create le strade rispetto all'ordine di selezione, e verrà instaurato un nuovo percorso che collegherà tali nodi;

- **Move Node**: alla pressione di tale pulsante, si entrerà in modalità "Move Node". In questo stato sarà possibile selezionare un nodo e trascinarlo all'interno della mappa, cambiando quindi la sua posizione: le strade verranno ridisegnate seguendo volta per volta il percorso segnato dal nodo in movimento. Al termine, i parametri modificati, come le coordinate del nodo e la lunghezza delle strade ad esso collegate verranno sovrascritti nel database;

- **Find Path**: selezionando due nodi casuali e collegati all'interno della mappa, verrà calcolato e visualizzato il percorso ottimo, in maniera tale da rendere partecipe l'amministratore della via più breve punto-punto;

- **Remove Node**: selezionando uno o più nodi e premendo questo pulsante, tali nodi saranno eliminati dalla mappa e di conseguenza anche dal database. Oltre ai nodi verranno eliminate anche eventuali icone ad essi sovrapposte e ovviamente anche le strade ad essi collegate;

- **Delete icon :** comando che permette, una volta selezionato un nodo, di eliminare l'icona ad esso sovrapposta. A differenza del comando precedente l'eliminazione coinvolge unicamente l'icona, e non il nodo o le strade collegate.

Il pulsante "Delete Map" consente di eliminare dal database l'attuale mappa presentata e di conseguenza anche tutti gli elementi grafici presenti. Al termine dell'operazione l'utente verrà riportato alla schermata "Get Started". Il refactoring dei pulsanti è stato reso possibile in seguito all'inserimento della logica di una macchina a stati: la maggior parte dei pulsanti infatti comporterà un passaggio di stato il quale è tradotto in una diversa interpretazione dei "click" che avverranno sulla mappa.

4.3 Progettazione della Componente Amministrativa

Vengono di seguito presentate le principali componenti responsabili del funzionamento del sistema lato server. La trattazione si focalizza sui componenti relativi all'editor della parte amministrativa.

4.3.1 Setup

Setup.ts si occupa di caricare i dati relativi alla mappa sulla quale si vogliono effettuare le modifiche.

```
1     class Setup {
2
3     private controller: Controller;
4     private graph: Graph;
5     private drawers: IList<Drawer>;
6     private iconManager: IconManager;
7
8     public setup(): void {
9           this.getMapInfo((longitude, latitude, zoom, name) => {
10                this.getIcons(() => {
11                      this.initComponents(longitude, latitude, zoom);
12                })
13          });
14    }
15    . . .
```

Figura 4.8: Classe di setup del sistema

Questa classe si occupa di istanziare e settare gli oggetti di View (Drawer) e Model (Graph) oltre che creare un'istanza della mappa tramite i parametri di latitudine, longitudine e zoom ottenuti tramite un'apposita chiamata al server. Il metodo di setup si avvale di 3 chiamate a metodi specifici, eseguiti in ordine temporale tramite utilizzo di callback per evitare errori di inizializzazione:

- *initComponents(...)*: metodo che si occupa di istanziare gli oggetti relativi al SettingsManager, IconManager, Controller, View e Model;
- *getIcons(...)*: permette di ricavare, tramite un'opportuna chiamata al server, tutte le icone inserite nel database e utilizzabili per la creazione dei punti di interesse all'interno della mappa;
- *getMapInfo(...)*: metodo che, dato l'ID della mappa da caricare ottenuto precedentemente tramite la schermata *LoadMap*, effettua una chiamata al server per ottenere tutti i parametri di configurazione della mappa da visualizzare, come ad esempio la latitudine, la longitudine e lo zoom con il quale si era deciso di settare inizialmente tale mappa.

4.3.2 Browser Observable

Browser Observable è la componente software che si occupa di catturare gli eventi associati alle interazioni dell'utente con l'interfaccia grafica e di notificarli al Controller del

sistema, che assumerà in questo caso il ruolo di "osservatore". In particolare, in questa classe sono stati implementati, utilizzando l'apposita libreria di JQuery, tutti gli eventi che sarebbero potuti scaturire dall'interazione con i pulsanti della schermata di Editor. Tale classe, implementata seguendo il pattern Singleton, incamera metodi utili per la registrazione dei relativi Observer, i quali potranno di conseguenza notificare eventi e cambiamenti alla componente grafica e a quella di modellazione.

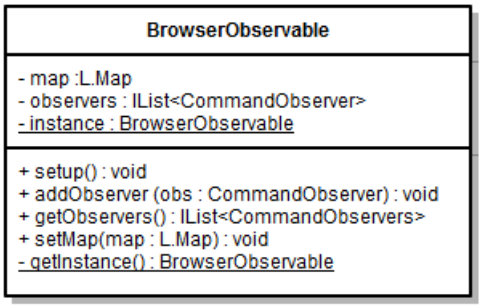

Figura 4.9: Classe BrowserObservable

4.3.3 Map Drawer

La classe MapDrawer, la quale implementa l'interfaccia *Drawer* costituisce la componente principale della View all'interno dell'Editor. La logica implementativa e i dettagli strutturali non saranno trattati in questo documento, dato che la realizzazione della parte grafica è stata portata a termine da un altro componente del gruppo. Occorre però precisare che è in questa classe che sono stati implementati i metodi collegati all'interazione dell'utente con le componenti grafiche quali nodi, icone e strade: tale soluzione è stata scelta per pura comodità implementativa siccome era in questa parte di codice che più facilmente si potevano catturare gli eventi relativi alla selezione degli elementi grafici della libreria Leaflet.

Il comportamento di tale classe è quindi per certi versi simile a quello della classe BrowserObservable, e la suddivisione di eventi tra queste due ha semplificato di molto l'implemetazione degli handlers e ha permesso di scomporre ulteriormente la logica di selezione dei componenti grafici e la loro manipolazione.

4.3.4 Controller

La classe Controller contiene la logica implementativa che consente di mappare ogni evento scaturito dall'interazione con l'interfaccia grafica e di utilizzarlo per apportare i dovuti cambiamenti alle classi che compogono la View e il Model. In questa classe sono contenute strutture dati relative a nodi, archi e punti di interesse: in particolare, vengono mantenute le corrispondenze tra elementi del modello ed elementi grafici, tramite strutture dati quali mappe e liste, come ad esempio l'HashMap *Map<Circle, Node>* oppure *Map <Line, Road>*. Tali corrispondenze verranno poi completate all'interno delle classi GraphCreator (Model) e MapDrawer(View) come di seguito precisato. La suddivisione degli handlers tra View e BrowserObservable ha reso necessario l'implementazione, da parte del Controller di due diverse, specifiche interfacce: *CommandObserver* e *MapObserver*.

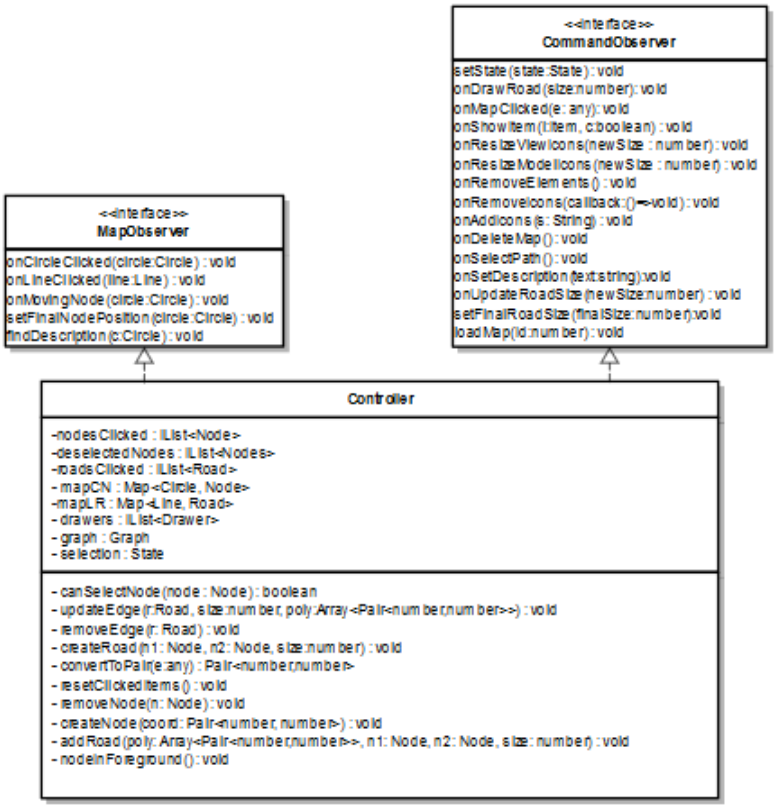

Figura 4.10: Classe Controller e interfacce implementate

Il Controller è stato ideato inizialmente seguendo l'idea di una macchina a stati: un semplice "click" sulla mappa o su un altro elemento grafico potrebbe infatti avere impatti differenti a seconda dello stato in cui l'automa si trova: i pulsanti della GUI *DrawNode*,

Remove, *MoveNode* e *DrawRoad* permettono al Controller di passare da uno stato all'altro tramite il metodo *setState(state : State)* e di interagire in maniera differente con la mappa che si sta modificando.

4.3.5 GraphCreator

Classe rappresentante la componente di modellazione del sistema, contiene metodi che permettono, tramite chiamate Ajax, di interagire col database per quanto riguarda le operazioni di inserimento, modifica e cancellazione degli elementi del dominio. Di seguito è presentato il diagramma delle classi relativo alla componente di modellazione del sistema, e sono riportati per la classe GraphCreator i metodi principali e più significativi.

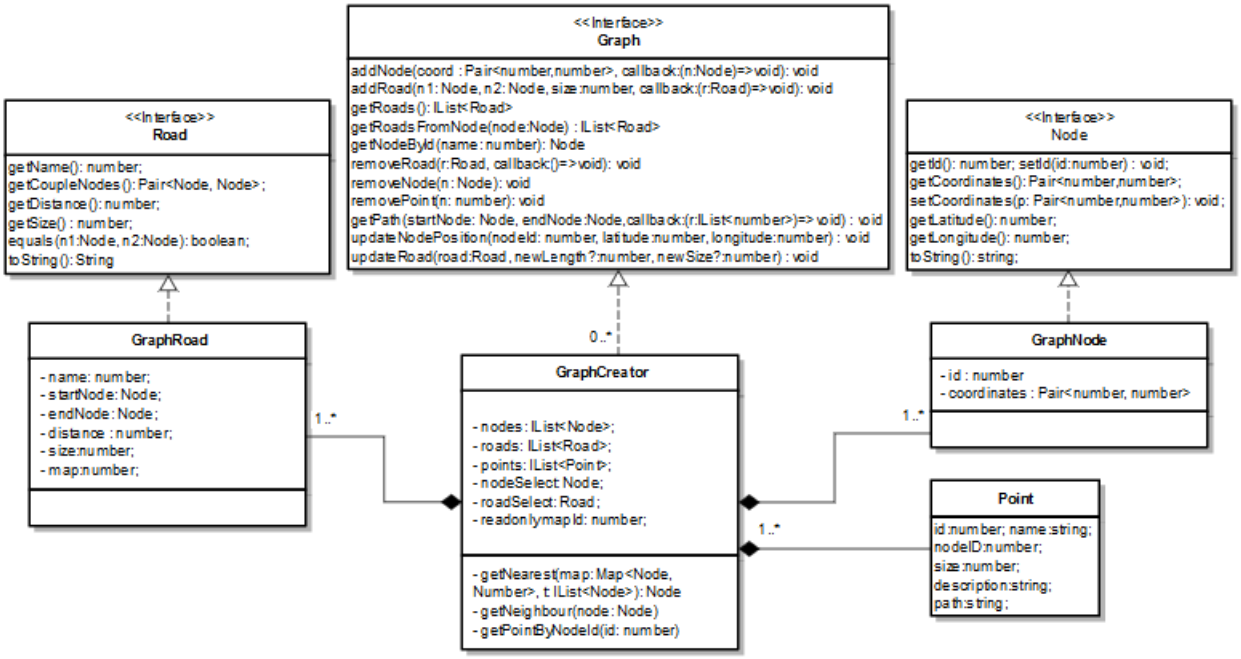

Figura 4.11: Architettura Model

4.4 Suddivisione e Gestione delle Entità del dominio

La logica di progettazione e di interazione delle entità presenti all'interno della componente amministrativa sono state ideate seguento il pattern architetturale MVC. Questo pattern è piuttosto diffuso nell'ambito dello sviluppo dei sistemi software e viene considerato funzionale in quanto consente di suddividere il codice in blocchi, i quali racchiudono al

loro interno un'insieme di funzionalità comuni. Il suo utilizzo consente di migliorare l'organizzazione del codice, incrementando la manutenibilità e velocizzando le sessioni di debug. L'idea generale di questo pattern è quella di mantenere separati e indipendenti le parti di modello (Model) e presentazione(View) in modo tale che le modifiche apportate a uno di questi due componenti non rischi di apportare malfunzionamenti all'altro: se in un'applicazione con architettura MVC si decidesse ad esempio di utilizzare una nuova libreria grafica, tutte le modifiche apportate alla View saranno percepite in maniera completamente trasparente dalle altre due componenti. In tale architettura è il Controller a rappresentare il collegamento tra la View e il Model: in particolare, le interazioni dell'utente sull'interfaccia grafica verranno percepite dal Controller come eventi, i quali avranno come conseguenza la modifica delle strutture dati presenti nella parte di modellazione. Queste modifiche verranno poi ripresentate all'utente tramite il Controller stesso, che andrà questa volta ad interagire direttamente con la componente grafica.

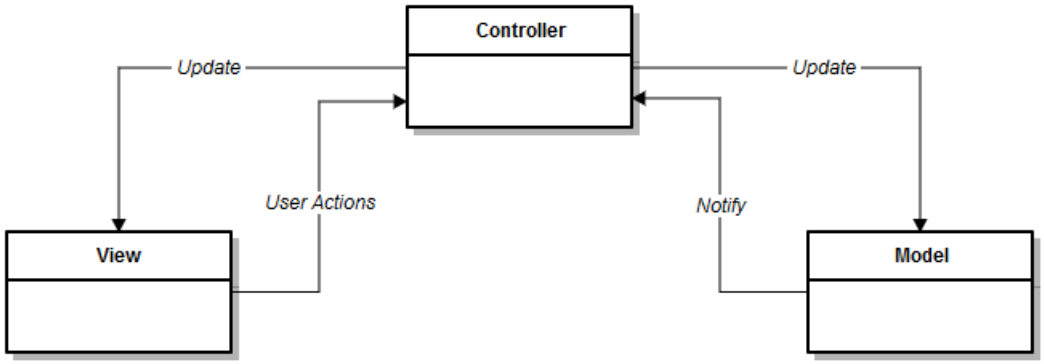

Figura 4.12: Interazioni tra le componenti nel pattern MVC

Nella prima fase di sviluppo del sistema, l'utilizzo del pattern è servito principalmente per disaccoppiare le diverse componenti del sistema, in termini di parte grafica (View) e parte logica (Controller - Model): i dati erano infatti storicizzati all'interno della parte di modellazione, e le iterazioni dell'utente con la GUI consentivano, seguendo la logica del pattern Observer, di modificare le strutture dati, quali liste, mappe e HashSet. Successivamente, avendo effettuato la suddivisione client-server all'interno del sistema, le strutture dati della parte di modellazione sono state storicizzate in un database apposito, e le funzioni di aggiunta, modifica e cancellazione presenti nel Model sono state trasformate in chiamate GET/POST che permettessero di interagire con il sistema di storicizzazione prescelto. Occorre precisare che le diverse componenti di modellazione rappresentabili (strade, punti

di interesse e nodi) sono state gestite separatamente all'interno di Model, View e Controller: nel Model ad esempio venivano storicizzati e considerati gli oggetti appartenenti alla classe Node, mentre nel Controller e nella View, rispettivamente, sono stati gestiti Circle e LCircle (Leaflet Circle). La stessa entità astratta (in questo caso il nodo di un grafo non orientato), era rappresentata e storicizzata diversamente all'interno di Model, View e Controller, in base alla semantica assunta all'interno di tali contesti. Questa separazione ha permesso di gestire indipendentemente i vari oggetti, traendone numerosi benefici, come ad esempio il fatto di non dover modificare le strutture dati di Model e Controller qualora si dovesse cambiare libreria grafica. L'unico svantaggio che si è presentato adottando questa strategia è stata la difficoltà nel mantenere coerenti le diverse strutture dati a fronte di nuove aggiunte o eliminazioni: è stato infatti necessario implementare un metodo di creazione o rimozione per ogni componente del pattern MVC, invocando metodi funzionalmente identici nel Controller, nel GraphCreator e nel MapDrawer. A tal scopo sono state create apposite strutture dati (mappe e liste) in ogni componente di questo pattern. Consideriamo ad esempio l'elemento "nodo":

- Model : storicizzazione dei nodi in una lista di oggetti "Node" (List<Node>);

• Controller : associazione tra oggetti di tipo"Node" del Model e oggetti di tipo "Circle", tramite apposite mappe (Map<Circle,Node>);

• View: associazione tra oggetti di tipo "Circle" e oggetti di tipo "LCircle (Leaflet Circle)";

Tali associazioni, come già accennato, sono state create anche per le strade, mentre i punti di interesse sono stati mappati solamente nella View e nel Model, dato che non è stato ritenuto necessario e utile mantenere una strruttura dati per tali elementi nel Controller.

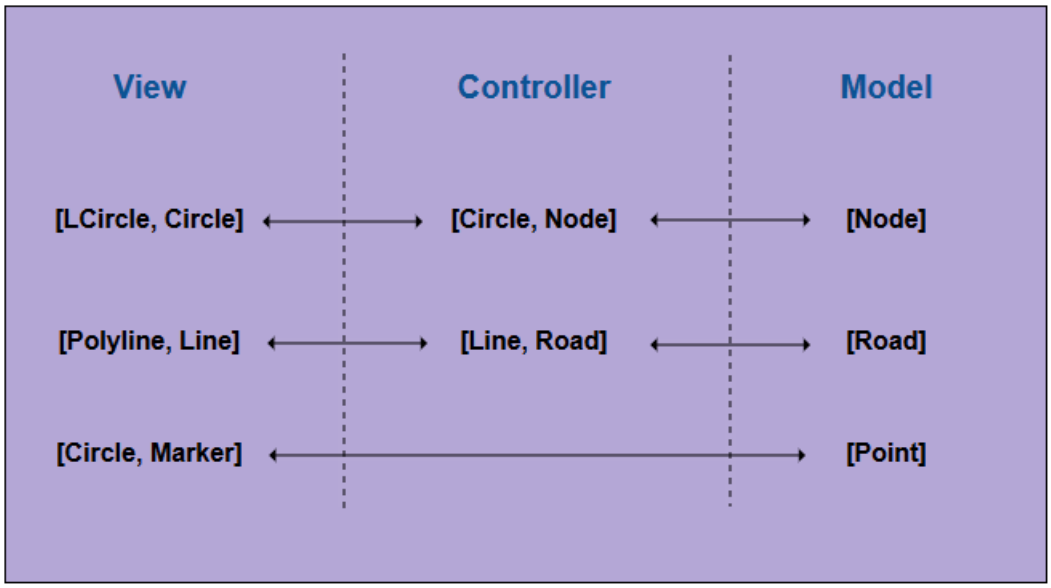

Figura 4.13: Separazione dei tipi di Entità nel pattern MVC

Capitolo 5

Sviluppo

In questo capitolo vengono messi in evidenza quelle che sono state le principali scelte e strategie di sviluppo dell'applicazione, focalizzandosi in particolar modo sulla realizzazione e implementazione delle funzionalità riguardanti sia la componente di dominio che la componente server.

5.1 Realizzazione della Componente di Editing

Realizzata precedentemente alla configurazione del sistema server-side, la componente di editing è la parte fondamentale che permette agli amministratori del sistema di realizzare e che verranno poi a disposizione dei clienti che ne richiedono l'utilizzo.

5.1.1 Concretizzazione dei Mock-Up

Vengono di seguito presentate le interfacce grafiche al termine del loro sviluppo. Tale realizzazione si è basata fortemente sui mock-up creati in precedenza, i quali hanno consentito di stabilire solide basi di progettazione che hanno permesso di diminuire i tempi di refactoring e manutenzione perfettiva.

GetStarted Page

Figura 5.1: Schermata di Home - GetStarted

Maps Page

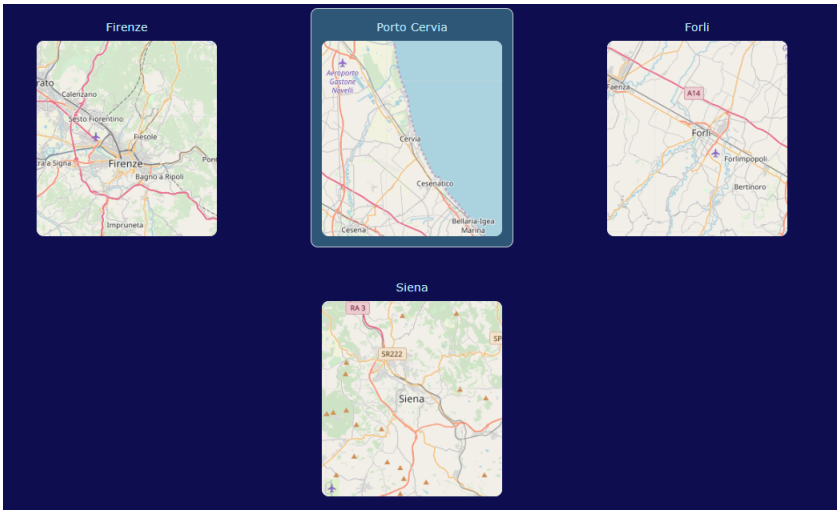

Figura 5.2: Schermata di Visualizzazione Mappe - Maps

NewMap Page

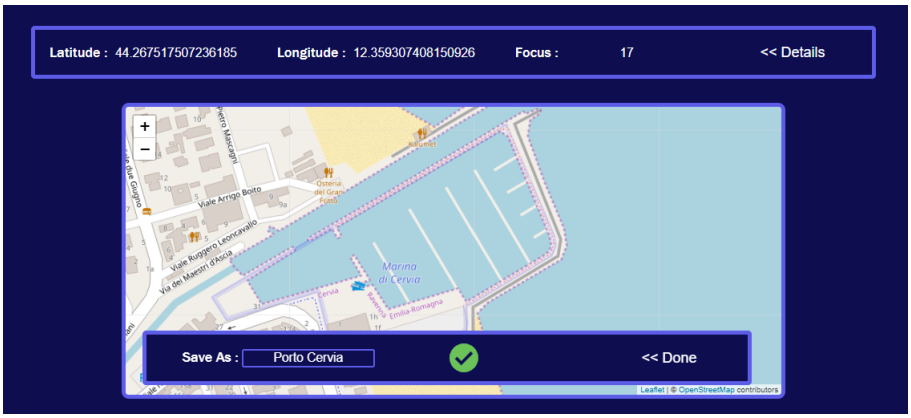

Figura 5.3: Schermata di Creazione di una Nuova Mappa - NewMap

EditingPage

Figura 5.4: Schermata d iEditing di Mappe Esistenti- Editor

5.1.2 Meccanismi di Interattività dell'Interfaccia Utente

Uno degli obbiettivi principali che sono stati prefissati all'inizio della progettazione di tale sistema è stato quello di creare un'interfacca di editing reattiva e che venisse aggiornata in tempo reale conseguentemente alle operazioni effettuate dall'utente. L'impatto visivo è stato realizzato in modo tale da essere percepito in maniera comprensibile, portando

l'utente a constatare il successo dell'operazione. Una particolare forma di aggiornamento è stata quella riguardante lo spostamento di un nodo collegato a più strade: inizialmente si era ipotizzato di aggiornare la posizione delle strade solamente dopo il rilascio del nodo sul nuovo punto di posizionamento, ossia al termine del trascinamento. In seguito tale ipotesi è stata scartata per il deludente impatto grafico e la poca intuitività di ciò che effettivamente stava accadendo. Si è proceduto quindi seguendo un altra strada: durante il trascinamento, le strade venivano ridisegnate ad intervalli di tempo regolari, in modo da ottenere un effetto di trascinamento continuativo anche per le strade stesse. Al termine del trascinamento vengono chiamate le API necessarie per l'update delle diverse strutture. Si è reso quindi necessario separare, unicamente per questa operazione, l'aggiornamento dell'interfaccia grafica con la modifica delle strutture dati della parte di modellazione, la quale avviene solo al termine dell'interazione.

Optando per questa strategia e desincronizzando View e Model, l'impatto visivo di tale operazione si è rivelato molto più soddisfacente e interessante, evitando per altro di sovraccaricare il server con chiamate di update, effettuando solamente l'aggiornamento finale. Tale soluzione è stata utilizzata anche per quanto riguarda la modifica della dimensione delle strade e dei punti di interesse. A seguito viene mostrato il diagramma di sequenza relativo alla funzione di spostamento di un nodo:

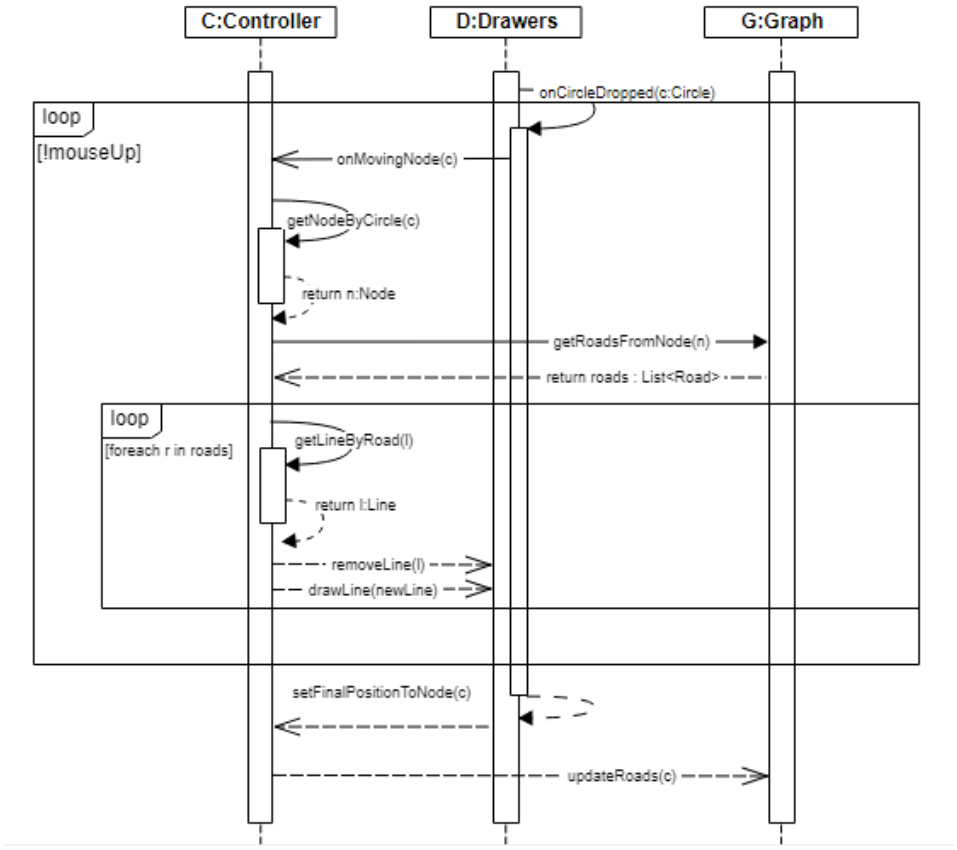

Figura 5.5: Diagramma di Sequenza- Spostamento di un nodo

5.1.3 Callback e Gestione degli Errori di Rimozione

L'introduzione all'interno del sistema delle chiamate API GET/POST che permettessero di interagire con gli elementi presenti nel database ha comportato una ristrutturazione della logica di chiamata, la quale potesse consentire di mantenere saldi i vincoli di integrità delle strutture dati presenti senza causare errori.

Tuttavia, le uniche modifiche necessarie al Controller sono quelle derivanti dalle operazioni di cancellazione: le strade ad sempio, sono state mappate nel database in maniera tale da mantenere la dipendenza funzionale con i due nodi presenti alle estremità. Cancellare un nodo senza aver prima eliminato la strada ad esso collegata rappresentava quindi una violazione del vincolo di integrità, con conseguente errore del sistema.

Benchè inizialmente questo non rappresentasse un problema, siccome i vincoli di integrità non sono contemplati nelle semplici strutture dati mantenute nel Model, aggiungendo la logica relazionale di un database è stato necessario garantire un "ordine temporale" se-

condo il quale effettuare tali modifiche: era necessario quindi che il Controller invocasse i metodi di cancellazione del singolo nodo in un momento successivo all'eliminazione delle strade collegate. Tale ordine temporale è stato implementato e gestito tramite l'utilizzo di callback, le quali offrivano una buona soluzione per controllare che le operazioni di eliminazione precedenti fossero terminate e andate a buon fine. L'esempio di come erano svolte le operazioni di cancellazione di un nodo tra Model, View e Controller sono rappresentate nel seguente diagramma di sequenza

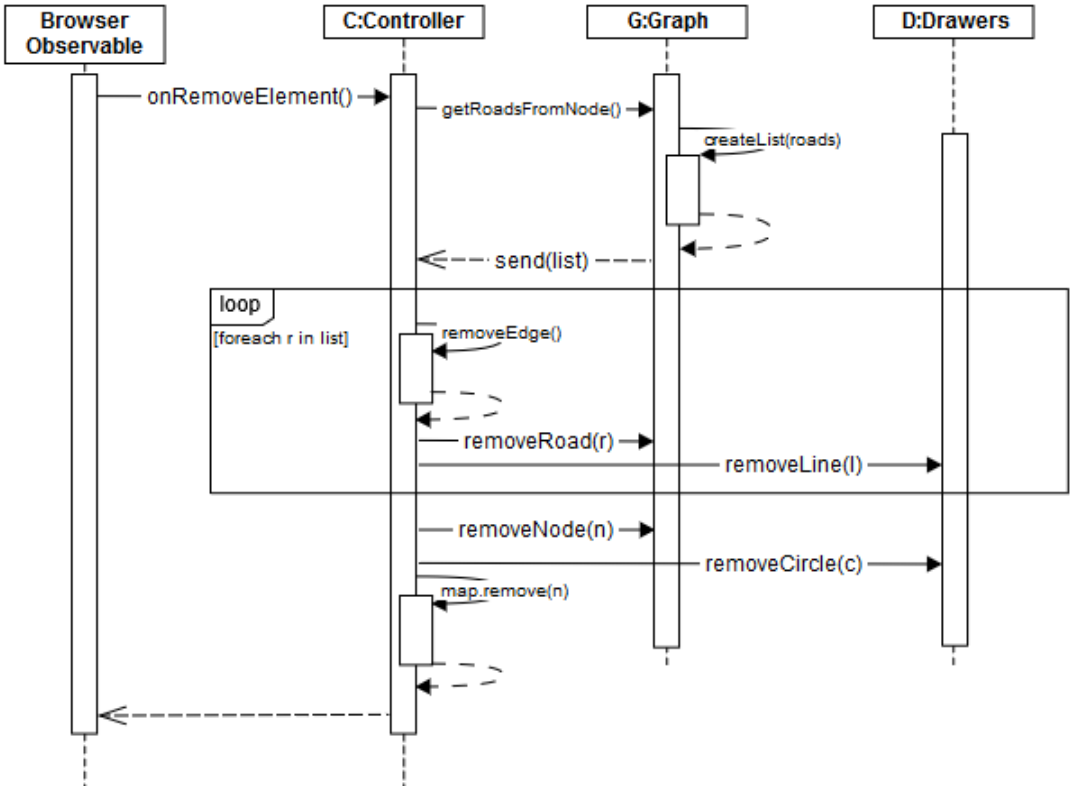

Figura 5.6: Diagramma di sequenza - Rimozione di un nodo

In tale diagramma, l'evento associato alla cancellazione di un nodo viene intercettato dalla classe BrowserObservable, responsabile di comunicare al Controller le interazioni dell'utente sull'interfaccia grafica. Il Controller a tal punto, dopo aver reperito l'insieme delle strade associate al nodo da eliminare, procede alla loro eliminazione. I metodi *Graph.removeRoad(r)* e *Drawer.removeLine(l)* servono rispettivamente per eliminare le due componenti relative al medesimo oggetto: la prima riferita alla rappresentazione grafica della strada all'interno della View, mentre la seconda relativa alla rappresentazione logica del percorso all'interno del Model.

Solo nel momento in cui tutte le strade collegate al nodo da rimuovere sono state eliminate, il Controller procede all'eliminazione del nodo in maniera pressochè identica: dopo aver invocato i metodi *Graph.removeNode(n)* e *Drawer.removeCircle(c)* viene eliminato il riferimento all'oggetto anche nelle strutture dati del Controller, permettendo di cancellare le diverse istanze del medesimo elemento da tutte le componenti associate all'architettura MVC.

5.2 Sviluppo della Componente Server

Di seguito è descritta la seconda principale componente del sistema, ossia la parte server-side, la quale è stata creata e progettata per permettere all'applicazione client di interagire e popolare il database, ricavare informazioni e storicizzare i dati utili. Tali strutture dati sono coinvolte in un meccanismo di astrazione resa tramite l'utilizzo dei Data Trasfer Object (DTO) i quali come viene spiegato nel primo paragrafo di questa sottosezione consentono di operare in maniera più semplice e sicura sui dati contenuti all'interno del database.

5.2.1 Data Transfer Objects

I "Data Transfer Object" o DTO sono rappresentazioni concettuali dei dati presenti su un database. Questi nuovi dati, che posseggono esclusivamente gli attributi corrispondenti a quelli registrati nelle tabelle del database vengono quindi serializzati per essere disponibili al codice del server.

I motivi per cui si sceglie di utilizzare i Dto sono svariati, ad esempio:

- necessità di dover rimuovere i riferimenti circolari e i vincoli di integrità che caratterizzano le istanza di un database;
- nascondere proprietà che non dovrebbero essere visualizzate dai client, oppure decidere di ometterle per alleggerire il payload del trasferimento;
- disaccoppiare il livello di servizio dalla logica del database.

I DTOs utilizzati all'interno del sistema per mappare ed interpretare i dati presenti sul database sono relativi ad ogni entità rappresentabile all'interno del sistema, ad esempio nodi, strade e punti di interesse. Anche per le icone, da passare all'applicazione client è stata utilizzata la medesima logica.

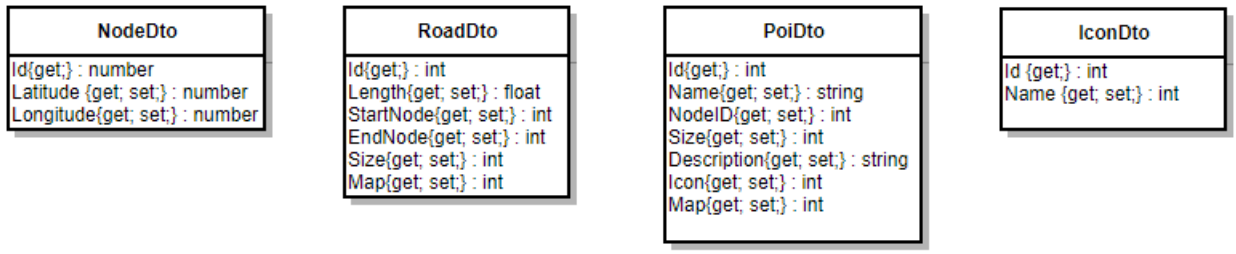

Figura 5.7: Data Transfer Object

5.2.2 RESTful API e Implementazione dei Controllers

All'interno dei Controllers sono stati implementati i metodi che corrispondono alle chiamate API da effettuare al server. In particolare è stato creato un Controller apposito per ogni entità del sistema, come i nodi o le strade. Le funzionalità di ogni controller, nel caso quest'ultimo fosse piuttosto corposo, sono state poi suddivise in due gruppi distinti di servizi:

- *Service*: chiamate (POST) che consentono l'aggiunta, la modifica e la rimozione di particolari tipi di oggetti all'interno del database;
- *QueryService*: chiamate (GET) che restituiscono collezioni di dati, che siano essi nodi, punti di interesse oppure insieme di strade, qualora ad esempio venisse richiesto il percorso ottimo tra due nodi.

Ad ogni Controller è stato poi affiancato un ulteriore Controller di tipo "Mock", il quale è stato realizzato per poter eseguire le stesse operazioni ma su un database fittizio. I "MockController" sono stati quindi implementati ed utilizzati per simulare e testare il funzionamento delle chiamate API servendosi di un database non reale. Questo ha permesso, precedentemente all'implementazione dei Controllers veri e propri, di testare in modo sicuro il comportamento delle API, senza rischiare di compromettere l'integrità e la struttura del reale database.

5.2.3 NodeController

La classe NodeController implementa tutti i servizi per la manipolazione dei nodi, come la restituzione di tutti i nodi di una mappa, l'update di un singolo nodo, l'aggiunta e la rimozione. Queste operazioni sono svolte appoggiandosi a due oggetti di tipo *NodeQueryService* e *NodeService* che come accennato in precedenza separano le due diverse tipologie

di operazioni da portare a termine. La classe implementa le due interfacce *ApiController* e *IDbNodeController*, di cui la seconda è stata utilizzata per creare un tipo di dato condivisibile con la classe MockNodeController, col vantaggio di poter facilmente interscambiare le due classi tramite Dependency Injection e passare dal database fittizio a quello reale.

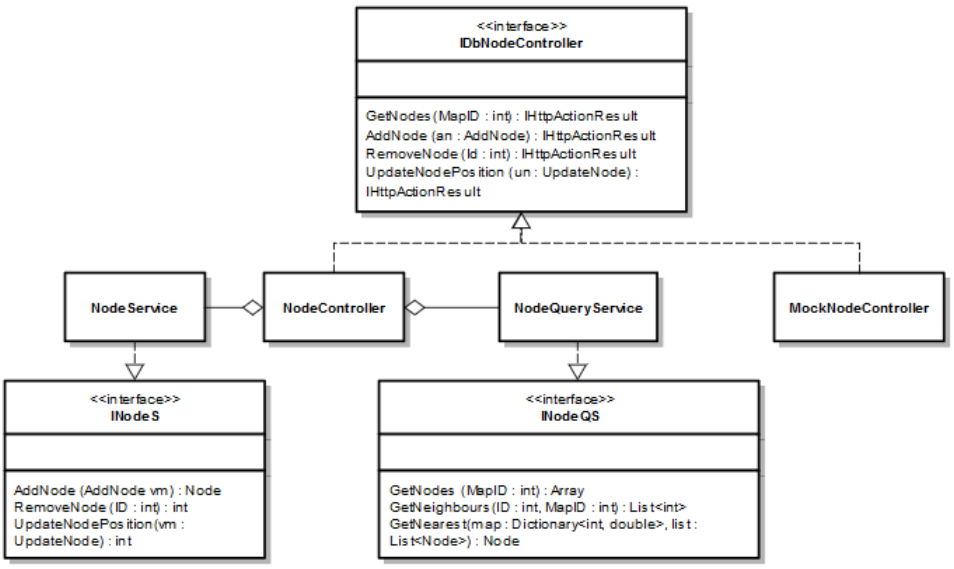

Figura 5.8: Suddivisione servizi “Service” e “QueryService”

Di seguito è riportata una vista del codice della classe NodeController. I metodi implementati richiamano le funzionalità degli oggetti di tipo *NodeService* e *NodeQueryService*.

La stessa suddivisione di funzionalità è stata la medesima anche per quanto riguarda le operazioni sul database che coinvolgono le strade, mentre per i punti di interesse, le mappe e le icone da caricare, essendo il numero di chiamate API più limitato, la totalità dei metodi è stata inserita nelle opportuni classi *Controller*, sia che si trattasse di query di ricerca che di query di modifica, inserimento o cancellazione.

```
1     using System.Net;
2     using System.Net.Http;
3     using System.Web.Http;
4     using Wayfinding.Services;
5     using Wayfinding.ViewModels;
6     using Wayfinding.ViewModels.Node;
7
8     namespace Wayfinding.Controllers.WebApi
9     {
10        public class NodeController : ApiController, IDbNodeController
11        {
12            private NodeQueryService nqs;
13            private NodeService ns;
14
15            public NodeController()
16            {
17                nqs = new NodeQueryService();
18                ns = new NodeService();
19            }
20
21            [System.Web.Http.HttpGet]
22            public IHttpActionResult GetNodes(int MapID)
23            {
24                return Ok(this.nqs.GetNodes(MapID));
25            }
26
27            [System.Web.Http.HttpPost]
28            public IHttpActionResult AddNode(AddNode vm)
29            {
30                return Ok(this.ns.AddNode(vm));
31            }
32    . . .
```

Figura 5.9: Vista implementativa della classe NodeController

5.2.4 Sviluppo dell'Algoritmo di Ricerca

La parte dei Controllers relativa alle strade si occupa della loro gestione all'interno del database. Ogni strada è storicizzata memorizzando i nodi agli estremi, la larghezza con la quale viene rappresentata graficamente e la lunghezza, calcolata nel momento della creazione tramite il metodo di utility *getDistanceNodeToNode(...)* :

```
1     static getDistanceNodeToNode(n1: Node, n2: Node) : number {
2           let lat1 = n1.getLatitude();
3           let lat2 = n2.getLatitude();
4           let lon1 = n1.getLongitude();
5           let lon2 = n2.getLongitude();
6           var R = 6371; // Radius of the earth (km)
7           var dLat = (lat2-lat1) * (Math.PI/180);
8           var dLon = (lon2-lon1) * (Math.PI/180);
9           var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
10                  Math.cos(lat1 * (Math.PI/180)) *Math.cos(lat2 * (Math.PI/180))*
11                  Math.sin(dLon/2) * Math.sin(dLon/2);
12          var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
13          var d = R * c; // Distance (km)
14          return d*1000; // Distance (m)
15    }
```

Figura 5.10: Metodo di utility per il calcolo della distanza tra due nodi

Dopo la creazione, i parametri relativi alla nuova strada sono caricati sul database, in

modo tale da poter essere richiesti al momento del caricamento della mappa.

Uno dei metodi principali della classe RoadController è sicuramente la chiamata API *getPath(...)*, la quale presi in ingresso due nodi permette di ricavare il percorso ottimo tra quelli disponibili. Il tutto è reso possibile grazie all'esecuzione dell'algoritmo di ricerca Dijkstra, di cui è riportato lo pseudocodice:

```
public GetPath(FirstNode, SecondNode, Map){
             foreach Node in Map :
                f.set (Node, Distance(Infinity))          // distanza da ogni nodo posta a +∞
                j.set (Node, null)                        // predecessore di ogni nodo sconosciuto
            end for
            f.Remove(FirstNode)
            f.Add(FirstNode, 0)                           // distanza dal nodo di partenza posta a 0
             foreach Neighbour n of FirstNode :
                f.Remove(n);
                f.Add(n, distanceBetween(FirstNode, n))   // distanza dai vicini conosciuta
            end for
            t = 'List of all graph nodes'
            t.Remove(RemoveNode)
            while (t is not empty) :
                nearest = GetNearest(f, t)                // nodo con la più breve distanza f
                j.Remove(nearest)
                j.Add(nearest, FirstNode)
                t.Remove(nearest)
                s.Add(nearest)
                neighbours = GetNeighbours(nearest, Map);
                foreach Node in s :
                   neighbours.Remove(Node)
                end for
                foreach Node in neighbours :
                    distance = f[nearest] + distanceBetween(nearest, Node)
                    if (distance < f[Node]) :
                        f.Remove(Node)
                        f.Add(Node, distance)
                        j.Remove(Node)
                        j.Add(Node, nearest)
                    end if
                 end for    // distanza da ogni vicino calcolata e sommata a quella che porta a FirstNode
             end while
             Next = database.Node.Find(SecondNode)
            do :
               Prev = j[Next]
               path.Add(getRoadsBetween(Next, Prev))       // strada più breve aggiunta al percorso
               Next = Prev
            while (Next != StartNode)
            return path
}
```

Figura 5.11: Algoritmo di Dijkstra - Pseudocodice

L'algoritmo di Dijkstra consente di ricavare il percorso ottimo tra un insieme di nodi collegati da archi in un tempo pari a

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

dove V indica il numero di nodi o vertici del grafo. Non si tratta dell'algoritmo di ricerca migliore: infatti nel caso i grafi fossero piuttosto grandi e non venissero adottate strategie

di euristica, la computazione potrebbe riverlarsi piuttosto pesante e verrebbe completata in tempi più lunghi. Ciònonostante, essendo le dimensioni dei grafi rappresentabili nel sistema di dimensioni ridotte, tale algoritmo si presta bene al calcolo del percorso migliore punto-punto. L'algoritmo è stato implementato partendo dallo pseudocodice, il quale è stato opportunamente adattato alle strutture dati presenti nel sistema.

Il metodo associato alla ricerca del percosto ottimo viene utilizzato per soddisfare le richieste API provenienti sia dall'applicazione client che dalla componente amministrativa, qualora si decidesse di effettuare controlli relativi al percorso ottimo prescelto.

Durante la fase di sviluppo è stato constatato che l'adozione di meccanismi e strategie di euristica da incorporare nell'algoritmo di ricerca si sarebbero rivelati poco efficaci: infatti i paramtetri euristici all'interno di problemi che rientrano nella categoria NP-hard consentono di apportare significativi miglioramenti solamente nel caso in cui la dimensione del problema sia sufficientemente grande. In questo scenario applicativo, le dimensioni dei grafi rappresentati rimarrebbero pressochè relativamente ridotte, mantenendo un numero di nodi e archi probabilmente molto inferiore a un centinaio. Eseguire l'algoritmo A*, ovvero Dijkstra migliorato con strategie euristiche, potrebbe causare overhead nella computazione: infatti i tempi di calcolo in grafi piuttosto ridotti, essendo costituiti da un numero più alto di parametri e operazioni, potrebbero paradossalmente essere più lunghi rispetto che quelli ottenuti dalla versione base dell'algoritmo. Oltre ciò, l'implementazione dell'algoritmo A* a completamento di Dijkstra andrebbe ad aggiungere un'inutile complessità al sistema.

Per questi motivi tali manutenzioni perfettive dell'algoritmo sono state scartate.

5.2.5 Valutazioni sul Miglioramento delle Performance

In alcune situazioni particolari, adottare meccanismi di parallelizzazione e caching potrebbe portare a significativi miglioramenti delle computazioni in termini di tempi di esecuzione e diminuzione dei periodi di latenza. In particolare nel sistema descritto, è stato valutato se effettivamente potesse convenire aggiungere tali soluzioni, parallelizzando operazioni indipendenti e complesse e introducendo meccanismi di caching che possano ad esempio consentire di reperire i percorsi scelti dagli utenti in base alle precedenti navigazioni dei percorsi nella stessa mappa, senza quindi avere la necessità di ricalcolare il percorso.

Nonostante ciò, i grafi creati all'interno del sistema, essendo esso progettato per ambienti relativamente piccoli, rimarrebbero di dimensioni piuttosto ridotte. Ci sarebbe quindi un eccessivo overhead e si velocizzerebbero richieste che nella maggiorparte dei casi sono

soddisfatte in tempi già molto brevi, aggiungendo complessità al sistema e dovendo gestire ulteriori problematiche, come ad esempio:

- **aggiornamento dei percorsi**: i percorsi storicizzati e resi disponibili agli utenti successivi che richiedono la stessa navigazione punto-punto sarebbero da aggiornare nuovamente ogniqualvolta la mappa viene modificata da un amministratore. Si potrebbe a tal scopo decidere di riaggiornare ogni percorso storicizzato relativo a una data mappa nel momento in cui tale mappa è modificata dalla componente amministrativa, oppure di effettuare l'aggiornamento relativo ad un percorso dopo la prima richiesta di un utente successiva alla modifica;
- **dimensioni del database**: nel caso in cui si volessero mantenere i percorsi registrati, il relativo database potrebbe dover contenere un numero molto alto di percorsi, pari a tutte le possibili rotte punto-punto presenti nelle varie mappe. Se un utente effettua una richiesta di navigazione, occorrerebbe ricercare il percorso tra quelli già storicizzati, rischiando paradossalmente di impiegare ancora più tempo e risorse rispetto che reperire il medesimo risultato tramite esecuzione dell'algoritmo.

Per suddetti motivi, l'idea di adottare meccanismi di caching server-side è stata scartata.

5.2.6 Dependency Injection

La Dependency Injection è un pattern di design definito nei paradigmi della programmazione ad oggetti e derivante dal pattern IoC (Inversion of Control), in cui un oggetto "iniettore" si occupa di risolvere le dipendenze relative agli altri oggetti istanziando di volta in volta i componenti di cui le relative classi necessitano.

In questo modo la creazione degli oggetti e degli elementi strutturanti dell'applicazione viene raccolta in un unico punto, in modo da rifattorizzare il codice e rendere più veloci ed efficaci le operazioni di testing: è possibile infatti creare classi "mock" con le quali effettuare alcuni test nel caso in cui le classi vere e proprie non fossero ancora state implementate. Sarà infatti sufficiente modificare la struttura dell'oggetto "iniettore" in modo che passi come argomento oggetti appartenenti alle classi fittizie. Questo pattern può essere interpretato e realizzato in maniera differente in base alle esigenze che emergono dalla progettazione. Esistono infatti tre principali forme di Dependency Injection:

- *Contructor Injection*: in questo scenario l'iniettore di occupa di istanziare e risolvere le dipendenze delle altre classi attraverso gli opportuni costruttori;

– *Setter Injection*: similmente al caso precedente, le dipendenze vengono risolte attraverso i metodi "setter" messi a disposizione delle varie classi tramite le relative interfacce;

– *Interface Injection*: questo tipo di implementazione consente di risolvere le dipendenze associando ad una particolare interfaccia una o più classi che la implementano: in questo modo, mantenendo la stessa interfaccia e scambiando di volta in volta le classi che devono essere istanziate per risolvere le dipendenze, è possibile passare da una configurazione progettuale all'altra in maniera piuttosto veloce e comoda.

La forma del pattern che è stata realizzata in questo progetto è stata la Interface Injection, la quale ha avuto applicazione nel momento in cui si è presentata la necessità di effettuare test su un database "mock": come verrà spiegato successivamente, creando due tipologie di controller e potendoli interscambiare all'interno dell'injector, è stato possibile eseguire test con classi fittizie e reali con relativa semplicità.

Tale soluzione progetturale è stata utilizzata per quanto riguarda i test effettuati sul database: come verrà spiegato in seguito, la classe TestManager, responsabile di risolvere le dipendenze ai TestControllers, potrà appoggiarsi a diverse tipologie di questi ultimi. Si prenda come esempio il caso del testing gei nodi:

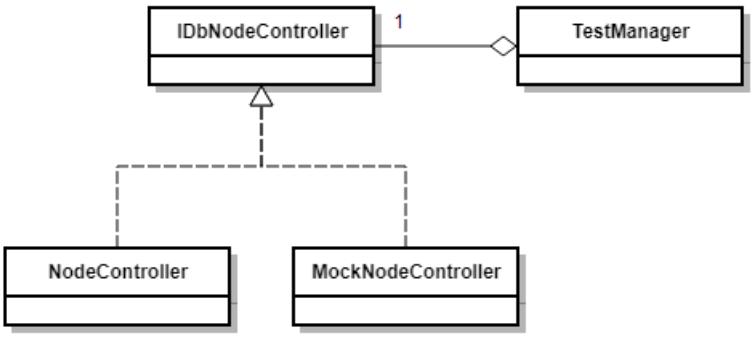

Figura 5.12: NodeController - Dependency Injection Class Diagram

Il TestManager, che in questo caso rappresenteà l'iniettore del pattern, utilizzerà sempre un oggetto di tipo *IDbNodeController*, ma sarà la classe che implementa l'interfaccia a stabilire se le operazioni effettuate avverranno sul database reale oppure sulla componente "mock". Allo stesso modo dei nodi, il procedimento si ripercuote anche su strade e punti di interesse. A livello di codice, la Dependency Injection viene implementata nel modo seguente:

```
1 // TestManager
2 private void Setup()
3 {
4    var container = new UnityContainer();
5    Mock = System.Configuration.ConfigurationManager.AppSettings["UseMock"].Equals("True");
6    if (Mock)
7    {
8         container.RegisterType<IDbNodeController, MockNodeController>();
9         container.RegisterType<IDbRoadController, MockRoadController>();
10        container.RegisterType<IDbPointController, MockPointController>();
11   }else{
12        container.RegisterType<IDbNodeController, NodeController>();
13        container.RegisterType<IDbRoadController, RoadController>();
14        container.RegisterType<IDbPointController, PointController>();
15   }
16   NodeController = container.Resolve<IDbNodeController>();
17   RoadController = container.Resolve<IDbRoadController>();
18   PointController = container.Resolve<IDbPointController>();
19 }
```

Figura 5.13: Risoluzione Dipendenze - Classe Test Manager

I vantaggi dati dall'utilizzo di questo pattern sono diversi: oltre a mantenere il codice più pulito e semplificare il debug dell'applicazione, la Dependency Injection può rivelarsi utile, nel caso di software di grandi dimensioni, per mantenere sotto controllo la quantità di memoria che viene utilizzata, dato che ogni oggetto verrà inizializzato in un solo punto del codice.

5.2.7 Sicurezza e Sistema di Cifratura delle Password

Come accennato in precedenza, i vari utenti possono loggarsi all'interno del sistema attraverso l'utilizzo di username e password, i quali possono essere settati opportunamente tramite la schermata di registrazione. Naturalmente, il salvataggio delle password all'interno del database non poteva essere risolto con la semplice storicizzazione delle stringhe. Si è quindi reso necessario inserire un sistema di cifratura, che consentisse, durante la registrazione e il login, di cifrare le stringhe da inserire all'interno delle chiamate API. La chiave di cifratura, seppur semplice, consente di sostituire ogni carattere inserito e contenuto nella password con un opportuno caratteri cifrato.

La stessa operazione viene eseguita durante il login: la password viene crittografata con lo stesso algoritmo e la stringa cifrata viene poi inviata al database e confrontata con quella già presente. Le operazioni di cifratura, per avere un livello di sicurazza maggiore, vengono eseguite all'interno dell'applicativo piuttosto che dalla componente server, in modo da avere già stringhe cifrate al momento dell'invio dei dati al server.

Capitolo 6

Validazione e Testing

La validazione del sistema è stata portata a termine in più step, verificando volta per volta se le funzionalità implementate soddisfacessero effettivamente i requisiti stabiliti. Gran parte del testing del sistema si è tradotta in un pesante utilizzo dell'editor, valutando se le operazioni messe a disposizione fossero portate a termine in maniera corretta.

Al termine dell'implementazione dei vari test-case, si è potuto constatare che nel loro insieme i test coprono gran parte del sistema, in termini di funzionalità e operazioni che necessitano di validazione. L'implementazione della componente di testing è suddivisibile in diversi campi, a seconda della tipologia di test effettuati:

6.1 Test delle funzionalità

I seguenti test sono stati implementati per verificare il corretto funzionamento delle feature implementate nel sistema:

- **aggiornamento della mappa e del database**: ogni operazione di inserimento, cancellazione o modifica degli elementi grafici all'interno della mappa avrebbe dovuto avere il conseguente e corretto impatto visivo. Inoltre, ovviamente, le modifiche relative ai parametri degli oggetti, come la loro posizione, dimensione e presenza all'interno della mappa avrebbero dovuto essere coerenti con tali cambiamenti grafici;
- **correttezza dei controlli utente e cambiamento di stato**: tramite appositi tool messi a disposizione dal browser, si è potuto constatare se effettivamente i controlli messi a disposizione per l'utente garantissero le funzionalità per le quali erano stati

inseriti. In particolare, è stato ampiamente testato il cambiamento di stato che portava l'utente ad interagire diversamente con la mappa;

- **correttezza dell'algoritmo di navigazione**: eseguendo test su carta e simulando la ricerca del percorso minimo è stato possibile constatare se effettivamente il percorso ottimo restituito dall'algoritmo di Dijkstra fosse corretto e se fosse sensibile alle modifiche della mappa, ad esempio per quanto riguarda lo spostamento di nodi e la conseguente modifica dei pesi delle strade del grafo;
- **correttezza delle chiamate API al server**: utilizzando il software di simulazione Postman è stato possibile effettuare un testing completo di tutte le chiamate che sarebbero potute pervenire al server, potendo prendere atto della composizione e sintassi delle strutture dati restituite e del fatto che il server fosse effettivamente raggiungibile.

6.1.1 Validazione dell' Interfaccia Utente

Le componenti grafiche e il layout delle pagine web componenti l'applicativo sono stati realizzati partendo da alcune direttive iniziali, ideate al momento dell'analisi dei requisiti. Sono stati poi organizzati e stabiliti "checkpoint" temporali, durante i quali l'interfaccia utente è stata testata e valutata dai componenti responsabili dello sviluppo, dal tutor aziendale e da altre fonti esterne all'azienda. I checkpoint e i pareri raccolti hanno permesso di creare una GUI che fosse funzionale e intuitiva anche per chi non fosse mai venuto a conoscenza di questo applicativo. Sono state inoltre seguite le direttive apprese durante il corso di Web, che hanno portato alla realizzazione di pagine il più possibile user-friendly e caratterizzate da un corretto bilanciamento dei colori, che rientra più generalmente nel rispetto delle principali direttive di validazione.

6.2 Unit Test

In principio, tutte le operazioni di inserimento, cancellazione e modifica da indirizzare sul database sono state testate in strutture dati create appositamente, le quali erano eseguite automaticamente su una mappa adibita al test. Tali funzioni sono state implementate all'interno dei "MockControllers", i quali simulavano le chiamate API per verificarne il corretto funzionamento. I controllers "mock" e quelli reali sono stati quindi creati in maniera tale da implementare la medesima interfaccia, e avere in secondo luogo la possibilità

di interscambiarli tramite Dependency Injection. La classe TestManager, strutturata in maniera tale da rappresentare un Singleton, è l'entità che si occupa, oltre che a risolvere le dipendenze, di fornire alle classi di test le strutture dati con le quali interagire, oltre che metodi utili per la creazione e la storicizzazione sul database.

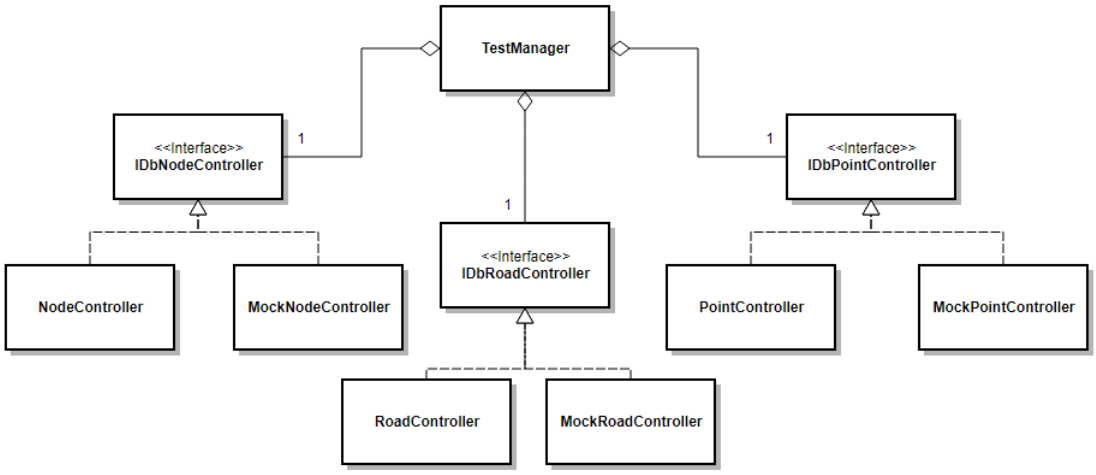

Figura 6.1: TestManager - Diagramma delle Classi

La classe TestManager si compone di tre oggetti che rispettivamente incamerano i metodi per eseguire il testing su entità specifiche, ossia nodi, archi e punti di interesse. I tre controllers, come già accennato, possono essere scambiati con le rispettive versioni mock senza che ciò possa apportare modifiche alle altre classi. Ogni controller incamera le funzionalità per poter eseguire tutti i tipi di operazioni sul database, nello specifico

- **operazioni di inserimento**: tali operazioni vengono testate controllando se effettivamente, sia nel database "mock" che in quello reale gli oggetti creati ed inseriti siano effettivamente stati aggiunti;
- **operazioni di eliminazione**: viene simulata l'eliminazione degli elementi creati appositamente sulla mappa di test. Le operazioni di cancellazione sono state svolte in maniera indipendente per quanto riguarda i singoli elementi grafici: non è stato quindi simulato l'evento di eliminazione di un nodo collegato a più strade, siccome nella parte applicativa le chiamate al server sono effettuate nell'ordine corretto per evitare la violazione dei vincoli. Un test del genere si sarebbe rivelato utile nel caso i meccanismi di controllo fossero stati implementati nella parte del server, mentre in questo caso avrebbero aggiunto complessità senza effettivamente apportare nessun contributo alla validazione del sistema;

– **operazioni di aggiornamento**: in questo caso i test si svolgono creando un elemento del dominio applicativo, inserirlo nel database e in seguito richiamare le API necessarie alla sua modifica. Al termine, i nuovi paramentri sono confrontati a quelli stabiliti per la modifica, in modo da verificare se tali operazioni sono state effettivamente portate a termine in maniera corretta. Ogni elemento del dominio ha avuto come parametri di modifica quelli che sarebbero potuti essere modificabilli in seguito all'interazione dell'utente sulla mappa, fatta eccezione per la lunghezza delle strade, il cui valore non può essere stabilito dall'utente ma viene settato in automatico in seguito al collegamento con due nodi agli estremi.

6.2.1 Test sui Grafi

Oltre ai test riportati sopra, ne sono stati eseguiti altri più corposi per verificare che le varie entità o elementi rappresentati in una stessa mappa fossero effettivamente visti logicamente come appartenenti allo stesso grafo. I due test realizzati per questo scopo sono stati:

– **checkNeighbours**: il test viene eseguito creando un nodo "master" al quale vengono poi collegati tramite altre strade un numero prestabilito di nodi. Richiamando poi il metodo di utility si verifica che effettivametne i vicini del nodo master sono i punti a lui collegati;

– **checkRoads**: il test prende come modello quello precedente, creando un nodo master che viene collegato a più nodi. In questo caso viene però verificato che il numero di strade aventi un estremo in quel nodo sono effettivamente nello stesso numero dei vicini;

– **GoogleMaps tests**: utilizzando il supporto di navigazione del software di wayfinding GoogleMaps, si è potuto effettivamente constatare che i percorsi ottimi trovati dall'algoritmo di Dijkstra fossero effettivamente i più brevi. Tali test sono stati eseguiti creando percorsi ad-hoc, e in seguito confrontando i risultati di ricerca con quelli di GoogleMaps.

Capitolo 7

Conclusioni

Il sistema di Wayfinding realizzato rappresenta una soluzione a supporto della navigazione originale: sono poche infatti le applicazioni simili che possano portare tale servizio su ambienti interni e privati, come village resort, aeroporti o grandi supermercati, permettendo ad eventuali clienti che richiedono il servizio di orientarsi in posti a loro completamente nuovi e sconosciuti. Lo sviluppo di quello che inizialmente pareva un sistema complesso e piuttosto corposo è stato seguito e pianificato in maniera efficace: seguendo a grandi linee i principi di analisi, progettazione e sviluppo il team è stato in grado di pianificare e organizzare al meglio le tempistiche e le priorità da assegnare ai requisiti.

Durante il periodo di sviluppo, le diverse funzionalità, rappresentate come milestone all'interno del ciclo di realizzazione del sistema, sono state di volta in volta verificate e validate durante opportune fasi di controllo, pianificate ad intervalli di tempo regolari, che potessero permettere di controllare il lavoro svolto all'interno di quello slot temporale, se necessario riorganizzarlo e poter con più calma stabilire come proseguire con la realizzazione.

Al termine di questo percorso si può affermare che il sistema portato a termine risulta funzionale, efficace e piuttosto robusto. Le caratteristiche e proprietà interne al sistema che riguardano ad esempio performance, portabilità e manutenibilità risultano presenti nel progetto in maniera visibile e pienamente soddisfacente. L'applicazione offre un supporto all'editing, all'organizzazione dei dati gestiti dai vari utenti e un supporto client alla navigazione: si può quindi dire che gli obiettivi inizialmente stabiliti e concordati sono stati raggiunti, avendo realizzato un sistema che soddisfa appieno e totalmente le funzionalità base richieste, ma che comunque potrebbe essere migliorato ed esteso con altre features. La creazione dei test funzionali, che per la maggiorparte servivano a garantire la consistenza

dei dati all'interno del database e la correttezza delle operazioni su di essi, hanno permesso di garantire una certa robustezza del sistema anche a fronte di un utilizzo pesante e continuo della componente amministrativa.

La totalità dei test, anche quelli che riguardano la parte "mock" del sistema, interscambiabili facilmente tramite Dependency Injetion, possono facilmente essere estesi parallelamente allo sviluppo di nuove funzionalità consentendo un maggiore controllo riguardo agli errori che si potrebbero presentare nel corso del processo di manutenzione evolutiva.

I test realizzati, congiuntamente a quelli visivi rappresentati dal semplice giudizio obiettivo che riguarda la correttezza delle operazioni, hanno quindi permesso di stabilire in maniera qualitativa e quantitativa quali fossero le funzionalità e features portate a termine: dal punto di vista quantitativo, i checkpoint raggiunti e i punti di sviluppo portati a termine sono da considerare soddisfacenti e in linea con ciò che era stato stabilito all'inizio della fase di analisi. I test realizzati sono piuttosto esaustivi per poter efffettivamente stabilire che le operazioni sono sicure e robuste.

I bug riscontrati e i vari problemi di funzionamento emersi durante il processo di realizzazione, anche se non impattavano in maniera incisiva sul funzionamento del sistema, sono stati per la maggiorparte risolti, ottenendo un sistema più corretto, prevedibile e sicuro. Dal punto di vista qualitativo si può invece dire che, dopo svariate fasi di miglioramento dell'interfaccia, l'utente si potrà trovare davanti ad un'applicazione dalla GUI semplice, intuitiva e soggettivamente stimolante da un punto di vista prettamente estetico. Gli strumenti di validazione utilizzati sono stati utili a garantire la correttezza dell'interfaccia in questi termini, aiutando a stabilire il giusto contrasto tra i diversi elementi grafici e altre caratteristiche meno rilevanti, come i vari stili del font adottato o le sue dimensioni. Inoltre, tutte le caratteristiche quantitative e qualitative, come già spiegato, sono state sempre controllate durante i diversi checkpoint, consentendo di apportare miglioramenti e modifiche a costi ridotti e potendo soprattutto stabilire se le strategie adottate fossero effettivamente quelle migliori. A tal scopo sono stati consultati i tutor aziendali, che hanno potuto stabilire la correttezza funzionale del sistema, mentre pareri esterni sono stati raccolti riguardo alla parte di grafica e interfaccia. Si può quindi stabilire che il processo di realizzazione e sviluppo del sistema si sia concluso in maniera soddisfacente e con successo.

7.1 Sviluppi Futuri

Le caratteristiche architetturali del sistema offrono un buon supporto alla realizzazione di tutte quelle funzionalità che potrebbero essere applicate al sistema, estendendolo. L'implementazione e sviluppo del sistema, svolti seguendo i vincoli posti da alcune best-practices, hanno consentito di poter ridurre i costi di manutenzione perfettiva, evolutiva e correttiva che si potrebbero eventualmente avere in fasi successive.

Riguardo agli sviluppi futuri, si possono evidenziare le seguenti linee di estensione:

- **navigazione indoor**: il sistema potrebbe essere esteso integrando le funzionalità, sia per la parte client che per quella server, riguardanti la creazione di mappe e il supporto alla navigazione per ambienti indoor. Questa estensione può essere messa a punto migliorando le strategie di posizionamento GPS e utilizzando diverse librerie grafiche che permettano di avere un focus maggiore, il quel permetta di portare il processo di editing ad un livello più preciso e in una zona più ristretta;

- **integrazione per ambienti su più piani**: per ora il sistema garantisce un corretto funzionamento per ambienti esterni e sviluppati solamente su un piano. Non sarebbe complicato tuttavia aggiungere funzionalità che possano mappare e guidare l'utente all'interno di ambienti strutturalmente realizzati su più piani, come grattacieli o palazzi. Nel momento in cui ogni piano venisse mappato in maniera separata e collegato logicamente agli altri tramite elementi strutturali come scale o ascensori, non servirebbero molti altri accorgimenti per poter sviluppare questa funzionalità;

- **mantenimento dati utente**: utilizzando cookie o altri meccanismi di storicizzazione, si potrebbero stilare, per ogni utente, i percorsi più richiesti e temporalmente più prossimi all'attuale utilizzo. Ovviamente tali percorsi sarebbero già calcolati come ottimi, e i risultati potrebbero essere di conseguenza presentati all'utente in tempi molto più rapidi e soprattutto senza il necessario supporto del server. Occorrerebbe però introdurre meccanismi di aggiornamento dei tali percorsi, a fronte delle modifiche che potrebbero essere apportate dagli amministratori e che coinvolgerebbero la modifica dei pesi dei vari archi;

- **gestione ambienti e grafica 3D**: il supporto alla navigazione e l'editing delle mappe viene supportato da un sistema di grafica a due dimensioni, ma con l'utilizzo di appositi tool e librerie grafiche l'intero procedimento potrebbe essere realizzato per ambientazioni in 3D, migliorando l'impatto visivo e la user-experience. Tale

miglioramento però non si rivela particolarmente semplice da sviluppare, soprattutto a livello grafico. Per quanto riguarda la parte di modellazione invece, l'impatto perfettivo sarebbe contenuto, dato che lo scorporamento ottenuto dall'utilizzo del pattern MVC permette di portare a termine tali estensioni a costi piuttosto ridotti.

Ringraziamenti

Volevo ringraziare i miei genitori e i parenti tutti che mi hanno sempre durante il percorso svolto, supportato e aiutato, offrendomi un ampio e fondametale sostegno.

Ringrazio il professor Viroli, il quale mi ha seguito in tutta la fase di sviluppo della tesi e il tutor aziendale Giacomo Dradi, il cui supporto nella realizzazione di questo progetto è stato decisivo, e che mi ha aiutato a conoscere nuovi metodi e strategie di sviluppo, che potranno sicuramente tornare utili per esperienze future.

Un rigraziamento particolare va a Linda, compagna di sviluppo e ottima amica, la quale si è sempre mostrata ben disposta a collaborare e il cui aiuto e partecipazione sono stati fondamentali, sia per lo sviluppo di questo progetto sia per tutte le altre atttività universitarie svolte assieme.

Ringrazio infine i miei amici, i quali hanno rappresentato un'ottima valvola di sfogo, e che hanno aiutato a rendere questa esperienza, congiuntamente a tutto il percorso di studi, meno stressanti e opprimenti.

Bibliografia

[1] K.Lync, *"The Image of the City"*. The MIT Press, Massachussets, 1960.

[2] E.Brown, *"Learning JavaScript: Add Sparkle and Life to Your Web Pages"*, 3ªed. O'Reilly Media, Inc., 2016.

[3] A. Osmani, *"Learning JavaScript Design Patterns: a JavaScript and JQuery Developer's Guide"*
O'Reilly Media, Inc., 2012

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest e Clifford Stein, *"Introduction to Algorithms"*, 3ªed.
The MIT Press Ltd, Massachussets, 2009

[5] Kanjilal, Joydip, *"ASP.NET Web API: build RESTful web applications and services on the .NET framework "*
eBook, 2013

[6] Nance C., *"TypeScript essentials: develop large scale responsive web applications with TypeScript "*
eBook, 2014

[7] Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm, *"Design Patterns: Elements of Reusable Object-Oriented Software"*
Addison-Weasley, California, 2002