UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

PROGETTO E SVILUPPO DI UNA APP CONTEXT-AWARE PER LA FRUIZIONE DI SERVIZI IN BASE AL PROFILO DELL'UTENTE

Relatori:

Chiar.mo Prof. Giacomo Cabri

Chiar.mo Prof. Riccardo Martoglia

Candidato:

Chiara Bruschi

Anno Accademico 2015-2016

Indice

Introduzione

Al giorno d'oggi l'offerta di servizi è grande e variegata e orientarsi nella scelta può essere a volte difficile. Un modo per offrire supporto all'utente è sviluppare applicazioni in grado di analizzare le richieste e adattare le risposte secondo il suo profilo e il contesto in cui si trova.
Spesso una richiesta di servizio comprende più aspetti. Per questo motivo un servizio può essere definito anche come composizione di più servizi.
All'utente viene proposta una composizione di servizi che rispecchia le sue esigenze, costruita sulla base dei dati acquisiti e, perciò, con alte possibilità di soddisfare la ricerca. Tutto ciò senza che l'utente debba cercare autonomamente il servizio migliore per sé.

Obiettivo

In questo lavoro di tesi si descrivono l'analisi, la progettazione e gli strumenti che hanno portato allo sviluppo di una app context-aware per la fruizione di servizi.
L'idea è nata dal problema di offrire all'utente un servizio completo, che sia individuato in modo automatico e senza necessità di fare più ricerche.

In particolare l'applicazione si pone come obiettivo di fornire informazioni sui servizi resi disponibili in relazione a ciò che si può visitare in una città, ad esempio servizi di ticketing dei musei. Sfruttando il contesto, l'applicazione è in grado di adattarsi e guidare l'utente verso la scelta di servizi vicini a lui e correlati alla sua ricerca.
La proposta fatta all'utente non è di un unico servizio, ma di una composizione che comprende un servizio principale e altri servizi secondari correlati al primo.

App context-aware

Cosa si intende per “app context-aware” ? Una applicazione “consapevole del contesto”, dove per “contesto” si intende l'insieme delle caratteristiche di ciò che circonda l'utente al momento dell'utilizzo dell'app, oltre alle informazioni sull'utente stesso.

Il contesto comprende principalmente il luogo in cui l'utente si trova, ma anche il suo profilo, come per esempio i suoi gusti personali e le preferenze di ricerca.
L'applicazione è "consapevole" di tale contesto nel senso che si adatta ad esso, cerca di individuarne le caratteristiche e di selezionare i risultati in base a cosa si conforma di più ad esse.

Struttura della tesi

Nel primo capitolo sono forniti i requisiti principali dell'app e le sue caratteristiche. In seguito è descritta l'analisi del funzionamento di base. Viene studiato il metodo di classificazione dei servizi in base al profilo dell'utente e al contesto e l'algoritmo di scelta con cui viene realizzata la composizione finale.

Il secondo capitolo si compone di tre parti interdipendenti: lo studio e la descrizione delle tecnologie di sviluppo dell'app, la descrizione del funzionamento e l'implementazione di quest'ultima sulla base dei requisiti.

In ultima analisi viene presentato il lavoro di test fatto sull'applicazione.
Seguono le conclusioni sul lavoro svolto e possibili sviluppi futuri.

1 Requisiti e progetto

La fase iniziale del progetto dell'applicazione ha compreso la ricerca e la specifica dei requisiti. Quest'ultima è servita per descrivere in modo completo e non ambiguo le funzionalità principali del software che è stato realizzato e le rispettive caratteristiche. Essa illustra le scelte fatte riguardo ai compiti che il software svolge, senza entrare in dettagli implementativi.

1.1 Specifica dei requisiti

Il risultato del progetto è una app *context-aware*. Essa mira ad essere utilizzata nei casi in cui è richiesto un insieme di servizi. Sfruttando dati relativi alla ricerca dell'utente, al suo profilo e al contesto si può raggiungere un risultato completo e "su misura". Uno dei vantaggi di questo approccio è che l'utente può ottenere risposta a ciò che cerca in modo veloce e soddisfacente senza dover fare più ricerche: tutti i servizi che rispecchiano la sua richiesta vengono forniti in un unico insieme.

Le problematiche principali prese in considerazione riguardano il metodo da utilizzare per avere un risultato che rispecchi nel modo più preciso possibile la richiesta dell'utente e il metodo di scelta della composizione di servizi. In altre parole, il problema principale è come trovare ciò che si collega di più alle caratteristiche dell'utente e a quelle del contesto.

Nella specifica dei requisiti si utilizza il termine *utente* per indicare l'utilizzatore del software, colui che richiede servizi facendo una ricerca con certe caratteristiche. Il *profilo utente* è l'insieme delle informazioni che caratterizza l'utente.
I *servizi* proposti dall'applicazione indicano le attività a disposizione dell'utente riguardanti il luogo che è stato inserito nella ricerca.
Infine, per *contesto* si indica tutto ciò che riguarda il mondo esterno che circonda l'utente, come il luogo in cui si trova. Nel progetto i termini contesto e profilo utente sono stati spesso utilizzati in correlazione.

1.1.1 Specifiche generali

Il software permette all'utente la fruizione di servizi adeguatamente scelti secondo le sue esigenze e che si adattano al contesto dove si trova.
Esso sfrutta componenti già esistenti per analizzare e comporre l'offerta dei servizi, quindi non ha una struttura indipendente. Tali componenti sono principalmente uno strumento Thesaurus e librerie esterne. Inoltre vengono sfruttati servizi come siti web, per ottenere informazioni.
Il software non è un semplice search engine, ma cerca ed elabora precisi risultati adatti al profilo utente e al contesto. Questo è il suo punto di forza e ciò permette un adeguato match con l'utente.

L'interfaccia è pensata per essere *user-friendly*, per andare incontro anche agli utenti meno esperti nell'utilizzo delle applicazioni software. Essa presenta componenti essenziali, ma intuitive che permettono una buona interattività con l'utente. La grafica esclude ogni ambiguità e guida in modo semplice l'utente nella scelta del servizio, con icone e pulsanti.

Il software mira ad essere *cross-platform*, quindi adatto a dispositivi diversi.
Per garantire una risposta completa alla richiesta dell'utente si sfruttano pacchetti software esterni come librerie, strumenti per il Thesaurus e servizi disponibili in siti web, ad esempio portali di ticketing online.

Per poter utilizzare certe funzionalità è necessario sfruttare un sistema server esterno all'applicazione, che comunica con essa utilizzando il protocollo HTTP.
Il programma memorizza eventuali dati in locale e non è necessario utilizzare ulteriori supporti di memoria esterni.

Il software ha il compito di:

- elaborare la richiesta dell'utente, sfruttando la comunicazione con il server esterno;
- trovare servizi che rispecchiano la richiesta in modo da soddisfarla nel miglior modo possibile;

- comporre i servizi che si adattano di più alle caratteristiche della richiesta e al contesto;
- fornire la composizione all'utente, attraverso un'interfaccia grafica;
- fornire all'utente la possibilità di modificare la composizione proposta;
- fornire all'utente la possibilità di accedere ad una *history* delle sue richieste passate.

Il software fornisce una risposta che si avvicina il più possibile a ciò che chiedono gli utenti, ma che può anche non essere del tutto soddisfacente.

1.1.2 Requisiti funzionali

I requisiti funzionali descrivono il comportamento del sistema in risposta a determinati input, ciò che esso deve fare in certe situazioni.

RF01	Parole chiave	Elaborazione
Input	Richiesta, contesto, utente e servizi	
Processo	L'applicazione individua e analizza semanticamente le parole chiave che caratterizzano ogni componente.	
Output	Insieme di parole chiave e concetti correlati che rispecchiano i componenti, da passare come input all'operazione successiva.	

Tabella 1.1: Requisito funzionale 1

RF02	Parole chiave	Ricerca servizi
Input	Parole chiave relative alla richiesta, al profilo utente e al contesto	
Processo	Tra i tanti servizi disponibili, l'applicazione cerca quelli che si collegano ai concetti raccolti.	
Output	Servizi che rispondono alla richiesta, da passare come input all'operazione successiva.	

Tabella 1.2: Requisito funzionale 2

RF03	Elaborazione servizi	Scelta servizi adatti
Input	Servizi trovati nella ricerca secondo le parole chiave	
Processo	L'applicazione analizza i servizi e li classifica rispetto a quanto si avvicinano al profilo dell'utente e alle caratteristiche del contesto.	
Output	Insieme di servizi con diversi gradi di importanza (pesi), in input all'operazione successiva.	

Tabella 1.3: Requisito funzionale 3

RF04	Elaborazione servizi	Composizione
Input	Insieme di servizi classificati	
Processo	L'applicazione crea associazioni tra i servizi, realizzando un percorso che forma una composizione. Viene scelta la composizione con punteggio più alto, moltiplicando i vari servizi secondo il peso dato loro dalla classificazione fatta in precedenza.	
Output	Composizione di servizi che rispecchia maggiormente la richiesta dell'utente e il contesto, viene proposta all'utente in risposta a ciò che cercava.	

Tabella 1.4: Requisito funzionale 4

1.1.3 Requisiti non funzionali

I requisiti non funzionali analizzati per il progetto riguardano le proprietà del sistema e le caratteristiche dei servizi che offre. Essi definiscono i vincoli di sviluppo del sistema.

Dal punto di vista delle prestazioni, il sistema gestisce gli utenti in modo indipendente l'uno dall'altro, quindi sono possibili più accessi concorrenti, purché fatti da dispositivi diversi. Le risposte alle richieste provenienti da utenti diversi vengono gestite in modo sequenziale rispetto all'ordine di arrivo.

Il sistema non può gestire contemporaneamente più ricerche fatte dallo stesso dispositivo, quindi l'utente può fare solo una ricerca alla volta.
Il sistema si basa sulla memorizzazione di dati in locale, quindi non utilizza DBMS.

Il grado di affidabilità del sistema dipende da quanto la richiesta dell'utente è chiara e non ambigua. Questo permette una risposta veloce e completa.
Il sistema è modulare e diviso in base al tipo di operazione, in modo da rendere più facili la modifica e il controllo delle singole parti.
Il software sfrutta in gran parte dati esterni riguardo le informazioni acquisite. Inoltre esso si basa su un *framework* che lo rende *cross-platform*, cosi da poter essere correttamente utilizzato su dispositivi diversi (mobile o computer).

Il sistema non risente in modo significativo dell'aumento del numero di utenti e le performance rimangono abbastanza stabili poiché ognuno è gestito in modo indipendente dagli altri. É possibile che un utente debba attendere qualche secondo in più del previsto prima di ricevere una risposta, in quanto il server che comunica con l'applicazione gestisce solo una richiesta alla volta. É anche possibile che, a causa di un grande flusso di richieste verso il server, l'utente debba inviare più volte la propria ricerca, finché questa non viene acquisita.

Riguardo la scalabilità, essa può interessare il caso in cui ci siano molti servizi da analizzare per generare la risposta alla richiesta o molte richieste inviate al server. In questo caso il sistema potrebbe subire rallentamenti, ma solo di pochi secondi.

1.2 Struttura base e algoritmo

Ciò che permette all'applicazione di essere precisa e completa nella risposta è l'utilizzo di strumenti di analisi del testo nell'elaborazione della richiesta dell'utente. Di fondamentale importanza sono le operazioni di analisi semantica e lo studio delle relazioni tra concetti simili.
In questa sezione viene mostrato lo studio fatto per trovare il corretto metodo di funzionamento dell'applicazione, sfruttando proprio l'analisi semantica.

1.2.1 Keyword come base del funzionamento

Il funzionamento dell'applicazione è basato su keyword.
L'idea è che l'utente inserisca come ricerca una parola chiave che sintetizza ciò che sta cercando. Questa viene elaborata dal sistema e utilizzata come base per individuare la risposta. Ad essa vengono associate altre keyword che identificano l'utente. Queste ultime rappresenteranno il profilo dell'utente nella ricerca dei servizi migliori.

Allo stesso modo anche il contesto, collegato all'utente, e i servizi disponibili sono rappresentati da un insieme di keyword.
Il contesto comprende solo il nome della città in cui si trova l'utente.
Le keyword dei servizi si possono ottenere facendo una analisi testuale della descrizione delle loro caratteristiche (es. per il servizio "Uffizi", si possono trovare le caratteristiche in un sito d'informazione culturale o sul sito ufficiale del museo). Questa analisi comprende operazioni di analisi lessicale, eliminazione di stopwords[1] e stemming[2].

1.2.2 Struttura

La struttura dell'applicazione è stata pensata come un grafo pesato senza cicli.
La rappresentazione grafica di un grafo è semplice e descrive in modo chiaro le relazioni tra i servizi compresi nei nodi. Inoltre è facile mostrare come ognuno di essi abbia un peso, necessario per la costruzione della composizione finale.

Nel grafo ogni nodo è un servizio disponibile e ogni arco rappresenta un collegamento tra servizi. Ad ogni nodo sono associati dei pesi, calcolati secondo quanto i servizi rispecchiano profilo utente e contesto.

1 *Stopwords*: parole appartenenti al lessico che non sono utili ai fini dell'analisi lessicale e dell'analisi semantica, ad esempio articoli, congiunzioni e avverbi

2 *Stemming*: procedimento che permette di ottenere la radice di una parola, eliminando prefissi e suffissi e riportandola alla forma base nel caso di verbi coniugati

La struttura è a livelli, ognuno con nodi paralleli.
Al primo livello sono presentati servizi equivalenti che rispecchiano (con pesi diversi) la richiesta principale dell'utente, cioè la parola chiave inserita. Questi permettono una prima selezione; ad esempio, se l'utente ha cercato “museo”, vengono proposti parallelamente i musei che si possono visitare nella città dove si trova. Viene messo in evidenza e consigliato il servizio che soddisfa più di altri la richiesta e le caratteristiche del contesto.

Ai livelli successivi seguono altre proposte di servizi che si collegano a quelli di primo livello e offrono ulteriori “approfondimenti” per la richiesta dell'utente, ad esempio un servizio di ticketing per il museo.

Per ogni ricerca dell'utente, l'applicazione sceglie un servizio ad ogni livello seguendo precise condizioni e lo mette in evidenza, consigliandolo all'utente. Come risultato viene fornita una composizione di tali servizi, rappresentata da un grafo.

1.2.3 Pesi del grafo

Ad ogni nodo del grafo è associato un peso che indica il grado di vicinanza del servizio nel nodo con ciò che ha richiesto l'utente.
Un peso si può intendere come il grado di somiglianza tra le keyword del servizio nel nodo e le keyword di utente e contesto.

Il calcolo viene fatto mettendo a confronto le keyword, non solo a livello di somiglianza semantica, ma anche a livello di relazione. Ad esempio si può dire che “statua” è un “monumento”, seguendo un rapporto di ipernimia, ma “statua” è anche in relazione con “piazza”.

Si cerca un match tra le parole chiave, analizzate a coppie.
Per l'analisi vengono utilizzate le funzionalità di WordNet, servizio di Thesaurus che fornisce relazioni semantiche tra parole.

Nel calcolo, le caratteristiche del profilo utente vengono inserite in un unico concetto "contesto". Il luogo inserito dall'utente è determinante nella scelta dei servizi e, se compreso nel "contesto", non permette un calcolo corretto ed efficiente. Per questo motivo è stato elaborato all'esterno delle formule, in modo da fornire una scelta di servizi più precisa.

Il peso di un nodo può essere definito come

$$W_{servizio} = \alpha * dsim(richiesta, servizio) + \beta * dsim(contesto, servizio)$$

dove α e β sono parametri fissati la cui somma vale 1.
dsim è una funzione di similarity che ha come valore un numero tra 0 e 1.

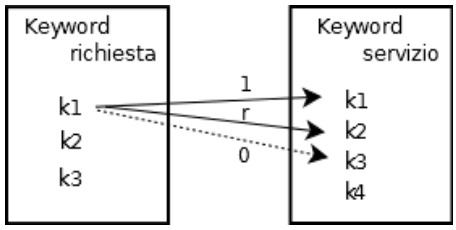
dsim è definita come la somma dei prodotti tra le keyword di richiesta/contesto e le keyword del servizio che si adatta di più ad esse, rispettivamente.
Il prodotto è calcolato per ogni keyword della richiesta/contesto e per ogni keyword del servizio che si adatta maggiormente ad essa, prese a coppie. I prodotti vengono poi sommati tra loro.

$$dsim = \frac{\sum k_1 * max\, k_2 [ksim(k_1, k_2)]}{n° keywords\ richiesta/contesto}$$

Per normalizzare il risultato, la somma viene divisa per il numero di keyword della richiesta/contesto di riferimento.

ksim è la funzione di confronto ed è chiamata per ogni coppia di keyword k1 della richiesta/contesto e k2 del servizio.

$$ksim(k_1, k_2) = \begin{cases} 1, & se \quad k_1 = k_2 \quad o \quad k_1 SYN\, k_2 \\ r, & se \quad k_1 REL\, k_2 \\ 0, & altrimenti \end{cases}$$



SYN è una relazione di sinonimia calcolata sfruttando le strutture *synsets*[3] di WordNet.

r è un parametro fissato con valore intermedio tra 0 e 1.

REL è una relazione calcolata sull'*hypernym tree* di WordNet, che indica la distanza tra parole (più due parole sono vicine, più sono correlate).

Per il calcolo della distanza viene utilizzata *hsim*, basata su un algoritmo di *Path-based Similarity*, quello di Leacock e Chodorow.

hsim è presa in considerazione solo se maggiore o uguale di una certa soglia fissata.

$$hsim(k_{1,}k_2)=\begin{cases} -\ln\left(\frac{len(k_{1,}k_2)}{2\mathrm{H}}\right), & se\ \ esiste\ il\ Least\ Common\ Subsumer\ di\ k_1\ e\ k_2 \\ 0, & altrimenti \end{cases}$$

len è la distanza tra le keyword considerate.

H è la massima profondità dell'hypernym tree di WordNet.

Il Least Common Subsumer è l' ”antenato” comune alle due keyword più vicino a loro nell'hypernym tree.

1.2.4 Algoritmo di scelta

L'obiettivo dell'applicazione è fornire una composizione dei servizi più adatti per l'utente. Questa è scelta in base al massimo prodotto ottenuto combinando i pesi dei nodi, ad ogni livello. Si deve quindi trovare la sequenza di servizi che dà il punteggio più alto.

La formula è $W_{percorso}=\prod W_{nodo}$

Si utilizza la produttoria per evidenziare l'effettiva differenza tra i pesi dei nodi.

3 *Synset*: insieme di sinonimi e termini correlati al termine principale al quale appartiene l'insieme

Se si utilizzasse la sommatoria e si sommasse un nodo con un peso piccolo, questo non cambierebbe molto il risultato finale. Diversamente, facendo il prodotto, il peso piccolo ha più influenza.

L'idea da seguire per scegliere il percorso si basa sull'algoritmo *shortest paths*: partendo da un nodo *source* si deve arrivare ad un nodo *destination* percorrendo meno archi possibili e componendo il punteggio più basso.
Nel nostro caso si dovrà adattare il prodotto dei nodi toccati in modo che sia il più grande possibile. Inoltre non ci saranno source e destination, ma si proveranno tutte le composizioni possibili partendo dai servizi di primo livello e arrivando ai servizi di ultimo livello.

In *Figura 1.1* viene riportato un esempio di costruzione della struttura e di assegnazione dei pesi.

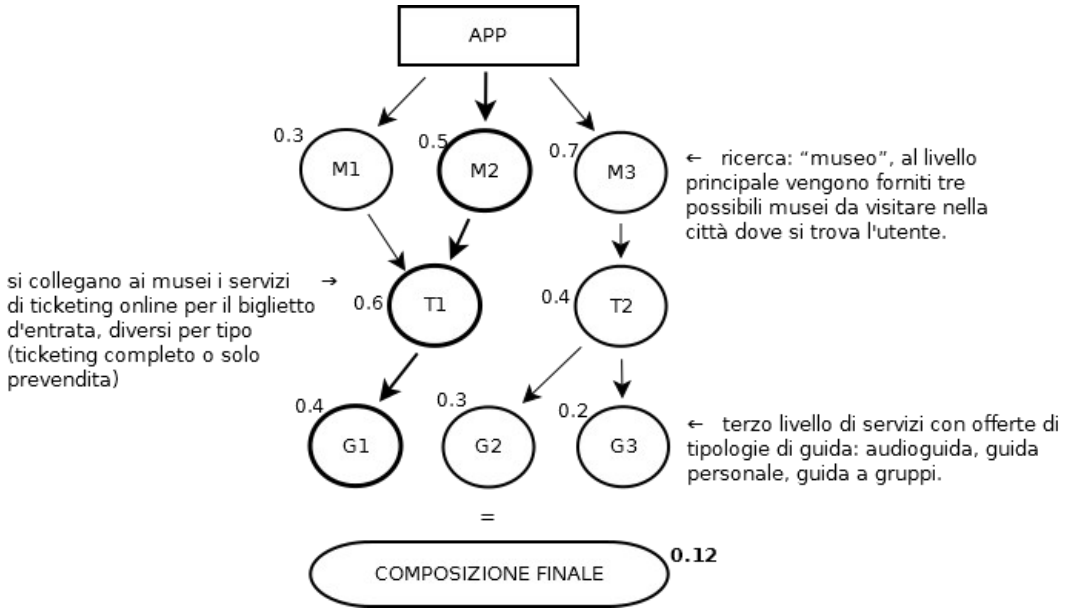


Figura 1.1 : esempio di struttura

1.2.5 Considerazioni

Durante l'implementazione è stato individuato un aspetto della struttura del grafo da evidenziare.

Il numero di livelli corrispondenti al servizio principale deve essere lo stesso per tutti i servizi, altrimenti il calcolo della composizione finale non risulta equo. Viene infatti scelta la composizione che raggruppa il minor numero di livelli e ha il punteggio più alto: scegliendo un servizio con un numero maggiore di livelli e facendo il prodotto con i pesi dei livelli in più, il punteggio finale viene diminuito.

Nonostante una certa composizione sia "arricchita" da livelli in più, non verrà mai scelta perché il suo punteggio finale risulterà sempre minore di un'altra con meno livelli. La *Figura 1.2* ne mostra un esempio.

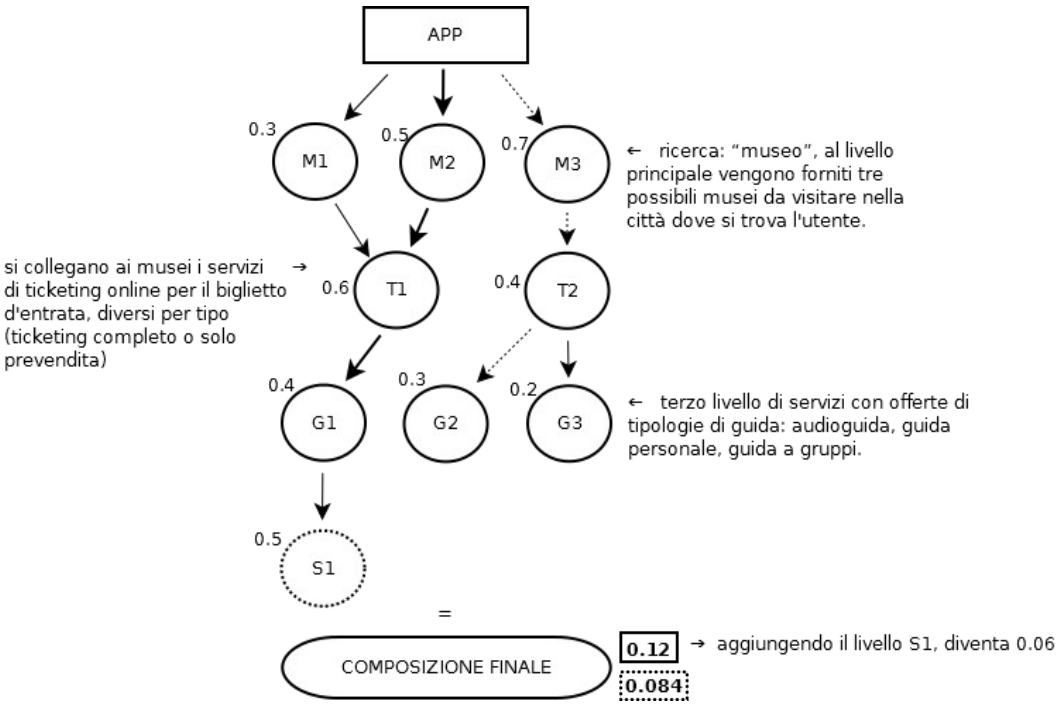


Figura 1.2 : composizione con un livello in più

Calcolando la composizione dei pesi, il percorso con punteggio maggiore risulta, nel caso in cui S1 non venga considerato, quello segnato con frecce in grassetto. Invece, se si considera il livello aggiuntivo, il percorso scelto è quello segnato con frecce tratteggiate.

2 Implementazione e funzionamento

2.1 Tecnologie e strumenti di sviluppo

L'applicazione è stata realizzata utilizzando il framework[4] Ionic, che è strettamente legato ad altre tecnologie e strumenti di sviluppo.

Ionic è un SDK[5] front-end di Apache Cordova.
Apache Cordova è un framework usato per sviluppare app mobile sfruttando le tecnologie web HTML, CSS e JS.

Ionic utilizza JavaScript e, di conseguenza, anche *Node.js*, un framework runtime per l'utilizzo di JavaScript lato server (back-end).
Node.js è stato creato in particolare per Javascript V8, l'interprete presente in Google Chrome (compila ed esegue codice JS), ma offre anche una funzionalità più generale. Esso integra un esteso database di moduli e librerie scaricabili per JavaScript, accessibili attraverso il package manager *NPM*.

Un altro strumento utilizzato da Ionic è *AngularJS*, un framework lato client (front-end) di cui si parlerà in seguito in questo capitolo.

Per le funzioni di analisi del testo necessarie nell'elaborazione della struttura del software è stata utilizzata la libreria *NLTK*, che sfrutta il database di *WordNet*. Inoltre sono stati sfruttati strumenti per l'elaborazione di testo come le librerie *Boilerpipe* e *Topia*. Queste librerie sono state utilizzate tramite codice Python.

4 *Framework*: architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitando il lavoro di sviluppo del programmatore. Esso è definito da un insieme di classi astratte e dalle relazioni tra esse. Alla sua base è sempre presente una serie di librerie di codice.

5 *SDK*: Software Development Kit, insieme di strumenti per lo sviluppo e la documentazione di software.

2.1.1 Ionic Framework

Ionic è un framework per lo sviluppo di app e in particolare per la creazione di interfacce grafiche di applicazioni mobile. Esso offre una libreria di componenti HTML, CSS e JavaScript ottimizzate per l'uso mobile e un insieme di tool per creare app interattive.

Nonostante non sia destinato all'utilizzo su web browser, Ionic è basato in buona parte su HTML5. Questo si spiega con il fatto che il risultato del lavoro del framework sono app ibride: simili a siti web ridotti, ma incapsulati in un ulteriore livello che permette loro di funzionare come fossero app native[6]. Ciò è reso possibile dall'utilizzo di Apache Cordova, che ha la funzione di wrapper.
Inoltre, alcuni tag usati nel codice HTML sono propri di Ionic e, arricchiti dalle direttive Angular, permettono di creare una app con un comportamento e una grafica uguali a quelli delle app native.
Il cuore del sistema è una pagina web ma tutto l'insieme viene eseguito all'interno di una struttura nativa. Quindi le applicazioni realizzate con Ionic devono essere considerate vere e proprie *app mobile* e non siti web ottimizzati per l'ambiente mobile.

Con Ionic le applicazioni possono essere multi-piattaforma, grazie all'utilizzo di tecnologie web standard che permettono l'esecuzione dello stesso codice in ambienti diversi (web browser, Android, iOS, Windows Phone).

Ionic utilizza semplici comandi da terminale per creare e testare l'applicazione.
Il comando fondamentale è *ionic start myApp,* che inizializza e crea la struttura dell'app in termini di cartelle e file.
La cartella principale del progetto è *www* e contiene altre cartelle e file per i codici CSS e JavaScript e per librerie e risorse grafiche. Un altro file importante è *index.html*, che contiene il markup della pagina principale in cui sarà visualizzata l'applicazione.

6 *App native*: applicazioni scritte e compilate per una specifica piattaforma, utilizzando linguaggi di programmazione e librerie supportati da quel sistema operativo. Hanno il vantaggio di poter sfruttare al meglio le funzionalità del sistema su cui sono eseguite.

Il comando *ionic platform add* serve per abilitare il supporto per la piattaforma mobile a cui puntiamo. Per l'effettiva generazione dell'app secondo la piattaforma desiderata si usa *ionic build*, seguito dalla chiamata all'emulatore *ionic emulate*.

Per una veloce visualizzazione della grafica dell'applicazione si utilizza *ionic serve*, che avvia il browser web predefinito, settandolo sulla pagina iniziale dell'app.
Un ultimo comando è *ionic run*, per eseguire l'applicazione direttamente su un dispositivo mobile collegato.

2.1.2 AngularJS

AngularJS è utilizzato da Ionic per sviluppare app robuste, sfruttando le specifiche direttive ed estensioni Angular che gli fornisce.

Esso è un framework lato client e si basa sul modulo *ng*, caricato di default quando l'applicazione inizia l'esecuzione. Inoltre sfrutta la tecnologia delle applicazioni single-page[7] e supporta l'architettura MVC.

Angular utilizza componenti organizzati in una struttura modulare, seguendo il principio della separazione delle competenze. I principali sono: *View*, componenti che riguardano l'interfaccia grafica; *Controller*, oggetti JavaScript che forniscono un ulteriore controllo su cosa visualizzare nell'interfaccia grafica; *Filtri*, che formattano il valore di un'espressione per visualizzarlo sulla view; *Direttive*, che estendono l'HTML; *Servizi*, che forniscono altre funzionalità.
Nel progetto sono stati utilizzati i controller, le direttive e i servizi.

Uno dei vantaggi di Angular è che esso utilizza il meccanismo di *two-way binding*, cioè fa in modo che ogni modifica al modello dei dati sia visibile automaticamente sulla view e, viceversa, che ogni modifica alla view venga riportata anche sul modello dei dati.

7 *Applicazione single-page*: applicazione web le cui risorse vengono create dinamicamente su richiesta, senza la necessità di ricaricare l'intera pagina web.

Angular si integra con HTML attraverso l'uso di specifiche direttive HTML del tipo "*ng-*...", inserite nei tag HTML. Grazie ad esse, Angular può definire al meglio i componenti dell'interfaccia grafica e il loro comportamento.
Le direttive sono marcatori HTML e svolgono la loro funzione senza dover essere accompagnate da ulteriore codice JavaScript, a differenza dei controller.
Ng-app è la direttiva che indica quale elemento della pagina prendere in considerazione e viene individuata in fase di inizializzazione, in quanto identifica la root dell'applicazione. Un'altra direttiva è *ng-model*, che definisce il modello dei dati e viene utilizzata principalmente per catturare i dati inseriti nei form. *Ng-controller* permette di associare controller e view, spesso in risposta ad un evento. Altre direttive utilizzate nel progetto sono *ng-show* e *ng-repeat*.

Il framework Angular viene inizializzato attraverso l'avvio del processo bootstrap (di inizializzazione e analisi) e la creazione dell'oggetto globale *angular*. Dopo aver individuato le direttive principali e aver creato le variabili del programma, il framework resta in attesa di un evento che coinvolga gli elementi indicati dalle direttive. In risposta all'evento parte in automatico una serie di elaborazioni.

2.1.3 WordNet

WordNet è un Thesaurus, cioè una raccolta di termini con le relative definizioni, associati ad altri termini secondo relazioni di significato. Esso presenta relazioni tra concetti e offre funzioni per elaborarle. Tra esse si distinguono relazioni di sinonimia, iperonimia, meronimia e tante altre. Nel progetto sono state studiate principalmente le prime due.

WordNet lavora su un hypernym tree, cioè una struttura gerarchica di parole legate da iperonimia e similarità. Per trovare la similarità tra parole sono stati studiati specifici algoritmi che sfruttano questa struttura. Infatti WordNet è un semplice database e deve essere integrato con funzioni che possano basarsi sulle sue proprietà. Questo è possibile tramite l'utilizzo di librerie e delle loro funzioni.

Nel progetto sono state necessarie elaborazioni del testo per:

- fare una *tokenization* di frasi, cioè separare ogni parola che le compone come entità indipendente;
- eliminare le *stopwords*, cioè le parole che non riguardano il significato, come congiunzioni, avverbi o articoli;
- fare uno *stemming*, cioè convertire le parole nella loro forma base.

Il database di WordNet è stato sfruttato per:

- trovare i *synsets* collegati ad una parola, cioè gruppi di parole accomunate dallo stesso senso;
- trovare la similarità tra parole, calcolando la loro distanza all'interno dell'*hypernym tree*.

Per sfruttare le funzionalità di analisi del testo e quelle offerte dal database di WordNet è stata utilizzata la libreria NLTK, scritta in linguaggio Python.

2.1.4 Altri strumenti

Nel progetto sono stati utilizzati anche *Boilerpipe*, *Topia* e *Flask*.

Boilerpipe è una libreria Java che, data una pagina web in input, permette di estrarre il testo contenuto in essa, eliminando i costrutti tipici del linguaggio HTML. La libreria fornisce varie tipologie di estrazione, per ottenere testi dettagliati o più riassuntivi.

Topia è una libreria Python che permette di determinare quali siano le parole più importanti contenute in un testo dato in input.

Flask è un microframework web scritto in Python. Esso permette di realizzare un'applicazione web che si comporti come un server, nel caso in cui sia target di una richiesta HTTP, e che generi una risposta.

2.2 Funzionamento dell'applicazione

All'avvio dell'applicazione viene presentata all'utente la schermata principale dell'interfaccia grafica. Per avviare una ricerca, l'utente inserisce la città in cui si trova e una o più parole chiave che identificano la sua query, cioè quello che desidera visitare nella città.

Vengono anche inserite nell'applicazione una o più parole chiave che identificano il profilo dell'utente, cioè che indicano le sue preferenze di ricerca, ad esempio se è interessato ad un servizio di acquisto biglietti online per il luogo che intende visitare oppure se è interessato ad avere a disposizione una guida. Queste caratteristiche non vengono inserite dall'utente, ma sono già presenti all'interno dell'applicazione al momento dell'interazione con l'utente. In una condizione reale e con funzionamento a regime, l'applicazione è installata sul dispositivo mobile dell'utente e si presume che abbia memorizzato al suo interno le caratteristiche dell'utilizzatore, fornite ad esempio durante la procedura di installazione. Una estensione dell'applicazione potrebbe essere di prevedere l'inserimento di tali preferenze insieme a query e luogo oppure di prevedere che l'applicazione stessa deduca le preferenze dell'utente dalle ricerche passate.

L'applicazione elabora le parole chiave inserite e calcola i pesi per ogni servizio disponibile. Queste operazioni sono eseguite su server, in comunicazione con l'applicazione.
Il luogo inserito dall'utente è determinante nella scelta dei servizi. Per fornire come risultato dei servizi che siano effettivamente presenti nella città in cui l'utente si trova, è fondamentale che, prima di ogni altro calcolo, l'applicazione filtri i servizi disponibili in base al luogo: se un servizio si trova esattamente nella città indicata viene preso, altrimenti il suo punteggio viene immediatamente posto a 0. Non vengono fatti match con luoghi vicini, ma solo con il luogo esatto.
Se questo passaggio non venisse eseguito, nella composizione finale potrebbero essere compresi anche servizi che si trovano in una città diversa da quella dell'utente e questo non deve accadere.

L'applicazione, o meglio il server, calcola quindi i pesi dei servizi filtrati, attraverso formule che esaminano il livello di similarità delle parole chiave che li compongono con quelle inserite dall'utente e quelle facenti parte del suo profilo. I pesi calcolati sono tutti normalizzati.

Una volta calcolati i pesi, vengono esaminate tutte le composizioni di servizi possibili, in modo da trovare quella che forma il punteggio più alto in base alle richieste dell'utente. Poi viene fornito come risultato la composizione trovata, passata dal server all'applicazione come risposta dell'elaborazione.

Ogni composizione è formata da tre livelli di servizi: un livello principale che indica il nome del servizio trovato e che si collega principalmente alla query inserita dall'utente; un livello “tickets”, che si collega al livello principale e indica quale servizio legato alla biglietteria è disponibile per il servizio principale scelto; un livello “guide”, sempre collegato al livello principale, che indica la disponibilità di varie tipologie di guida. Per ogni livello viene scelto un solo servizio.

I servizi dei livelli 2 e 3 non sono scelti solo perché collegati al livello principale, ma anche perché, come quest'ultimo, si collegano alle preferenze dell'utente. Essi vengono scelti perché i loro punteggi sono stati calcolati prendendo in considerazione anche il profilo dell'utente. Quindi, ad esempio, per un utente che desidera una audio-guida, un servizio che offre un tour guidato avrà peso inferiore di un altro che offre una audio-guida.

Dopo aver ottenuto la composizione di servizi, l'utente ha la possibilità di modificarla se non è soddisfatto o se desidera cercare delle alternative. La composizione proposta è quella che dà punteggio più alto, ma possono essere presenti anche alternative equivalenti che danno un punteggio uguale o simile. La modifica dell'utente potrebbe anche portare ad avere una composizione con punteggio più basso di quella precedente.

Esempio di funzionamento:

- *Città:* Florence → luogo inserito dall'utente
- *Profilo utente:* booking + tour → indica che l'utente è interessato ad un servizio di acquisto online di biglietti e alla possibilità di fare un tour guidato del luogo che vuole visitare
- *Query:* museum → ciò che l'utente vuole visitare (una o più parole chiave)

La città d'esempio è Firenze e i servizi disponibili sono:

- Accademia-florence-accademia-gallery-guide-museum-ticket-tour-work
- Bargello-florence-bronze-cellini-century-collection-courtyard-museum-sculpture-work
- Palazzo Vecchio-florence-booking-city-guide-museum-palace-ticket-time-tour-tours
- Uffizi-florence-booking-gallery-guide-museum-ticket-tour-uffizi

Ogni servizio è composto dal nome e da un elenco di parole chiave che lo caratterizzano e sulle quali vengono fatti i confronti per trovare ciò che si avvicina di più alla ricerca dell'utente.

In *Figura 2.1* viene mostrato il grafo che comprende i dati d'esempio presi in considerazione e la composizione scelta in base alle richieste inserite, con il punteggio finale di 0.49 (nodi in grassetto in *Figura 2.1*).
I servizi sono suddivisi per livelli: livello principale, livello "tickets" e livello "guide".

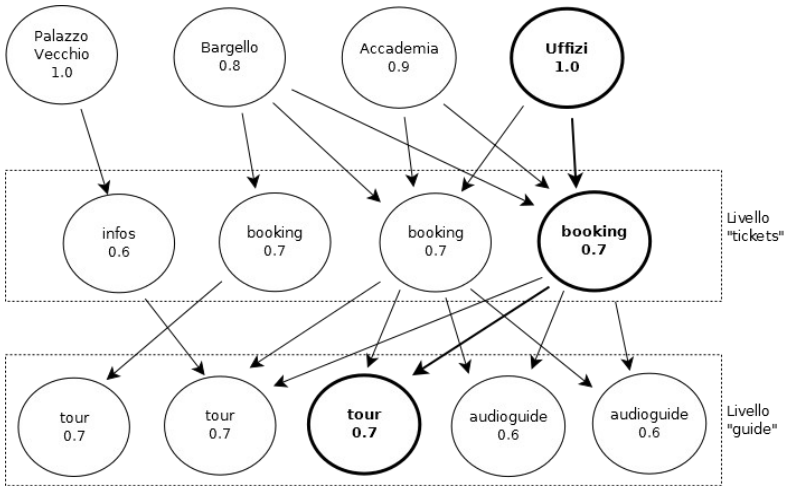


Figura 2.1 : grafo d'esempio

In *Figura 2.1*, i nodi con nome e peso uguale indicano servizi equivalenti. Nonostante abbiano caratteristiche molto simili, essi si distinguono per il percorso che porta a raggiungerli e per l'effettivo servizio offerto.
Per esempio, rispetto al livello “tickets”, due servizi *booking* con punteggio *0.7* possono indicare la possibilità di acquisto online di biglietti tramite due piattaforme diverse, come possono essere *Florence-tickets* o *Ticketone*.

2.3 Implementazione dell'applicazione

Il framework Ionic attraverso cui è stata sviluppata l'applicazione non si limita ad utilizzare le componenti del framework AngularJS, ma ne deriva anche il modello di presentazione: il *Model View Controller* (MVC).
Proprio perché segue questo modello, l'applicazione è gestita in base al principio della separazione delle competenze e i file che la compongono fanno parte di una struttura modulare.

Il modello MVC prevede la separazione di tre componenti: il *Model*, che gestisce il modello dei dati; la *View*, che si occupa della visualizzazione di tali dati attraverso l'interfaccia grafica; il *Controller*, che gestisce l'interazione tra i primi due componenti, registrando eventi che avvengono sulla View e modificando i dati del Model.

I file del progetto sono suddivisi in base a queste tre componenti: il file *services.js* contiene il codice che crea e inizializza i dati, cioè variabili e funzioni in codice JavaScript; i file **.html* e **.css* contengono il codice che crea e gestisce l'interfaccia grafica, definiscono in particolare templates e stili; il file *controllers.js* comprende i metodi controller propri di AngularJS.
Il file *app.js* può essere definito come una via di mezzo tra View e Controller, infatti gestisce la visualizzazione dei componenti grafici, ma lo fa attraverso metodi di configurazione scritti in codice JavaScript.

2.3.1 View

Il punto di partenza per lo sviluppo dell'applicazione è stata l'interfaccia grafica.
Per prima cosa è stato creato un nuovo progetto attraverso il comando *ionic start*, in modo da inizializzare la struttura di cartelle e file dell'applicazione (*Figura 2.2*).
Come modello iniziale è stato scelto il template *tabs* di Ionic, cioè una grafica a schede. Da qui sono state fatte modifiche e aggiunte per realizzare una grafica conforme al progetto.

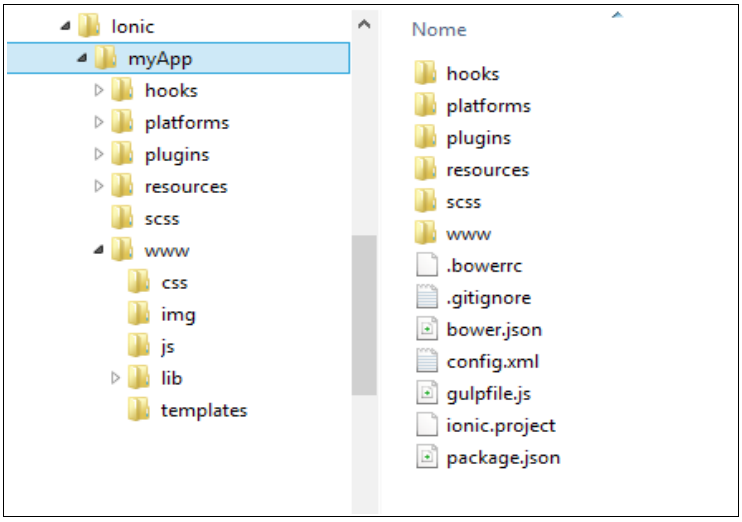


Figura 2.2 : struttura delle cartelle del progetto Ionic

Per questa parte di implementazione sono stati utilizzati tutti i file del progetto e non solamente quelli riguardanti la grafica. Infatti quest'ultima è stata inizialmente popolata con alcuni dati esemplificativi, creati attraverso una *factory*[8] di AngularJS in codice JavaScript, nel file *services.js*.

Completano la parte grafica i file *app.js* e *tabs*. Il primo comprende i comandi per far partire l'esecuzione e indicazioni riguardo quali siano i file a cui l'applicazione deve fare riferimento, cioè quelli contenenti dati e controller. In questo file sono stati configurati i template grafici, per ognuno indicando l'url, in modo da avere un riferimento ad essi, e i rispettivi controller (*Codice 2.1*).

8 *Factory*: metodo del modulo AngularJS che permette di definire variabili e funzioni e di utilizzarle all'interno di un controller, sfruttando il fatto che le funzioni vengono ritornate dal modulo tramite *return*.

Tali controller non devono essere necessariamente quelli utilizzati per gestire gli eventi di quella parte grafica e vengono di solito utilizzati per inizializzare i dati che verranno visualizzati.

```
angular.module('starter', ['ionic', 'starter.controllers',
'starter.services'])
....
.config(function($stateProvider, $urlRouterProvider) {

  $stateProvider

  // setup an abstract state for the tabs directive
    .state('tab', {
    url: '/tab',
    abstract: true,
    templateUrl: 'templates/tabs.html'
  })

  // Each tab has its own nav history stack:
  .state('tab.home', {
    url: '/home',
    views: {
      'tab-home': {
        templateUrl: 'templates/tab-home.html',
        controller: 'HomeCtrl'
      }
    }
  })
      //template figlio di home
      .state('tab.results', {
         url: '/home/results',
         views: {
           'tab-home': {
             templateUrl: 'templates/results.html',
             controller: 'ResCtrl'
             }
          }
      })
....
// if none of the above states are matched, use this as the
fallback
 $urlRouterProvider.otherwise('/tab/home');
});
```

Codice 2.1 : parte del file app.js

Il secondo file comprende codice HTML per la visualizzazione dei template che si riferiscono alle schede dell'applicazione (*Codice 2.2*).

```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">

 <!-- Home Tab -->
 <ion-tab title="Home" icon="ion-map" href="#/tab/home">
      <ion-nav-view name="tab-home"></ion-nav-view>
 </ion-tab>

 <!-- Edit Tab -->
 <ion-tab title="Edit" icon="ion-edit" href="#/tab/edit">
      <ion-nav-view name="tab-edit"></ion-nav-view>
 </ion-tab>

 <!-- History Tab -->
 <ion-tab title="History" icon="ion-earth" href="#/tab/history">
      <ion-nav-view name="tab-history"></ion-nav-view>
 </ion-tab>

</ion-tabs>
```

Codice 2.2 : file tabs.html

L'esecuzione dell'applicazione parte dalla pagina principale *index.html*, che inizializza il modello di grafica a schede. Grazie alla direttiva *ng-app=”starter”*, che fa riferimento alla configurazione dei template nel file *app.js*, dove è definito il modulo “starter”, la grafica viene puntata sulla prima scheda, che visualizza il template indicato come principale nel file, cioè *tab-home.html* (*Figure 2.3, 2.4*).

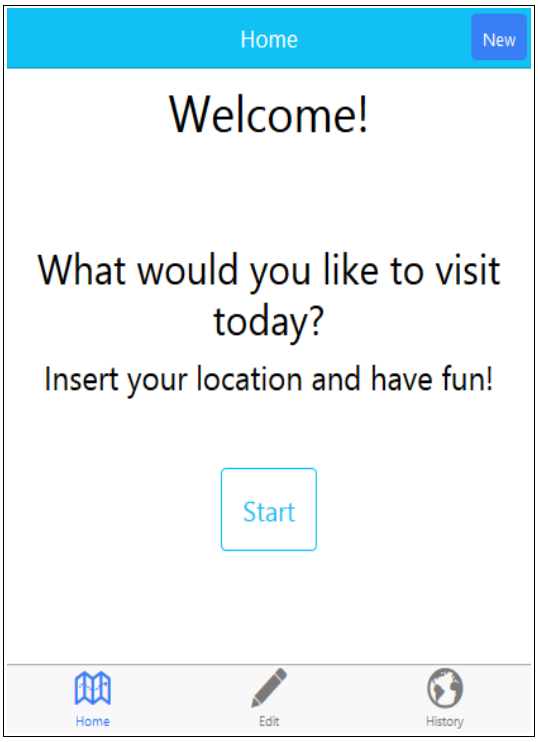


Figura 2.3 : schermata principale home, slide 1

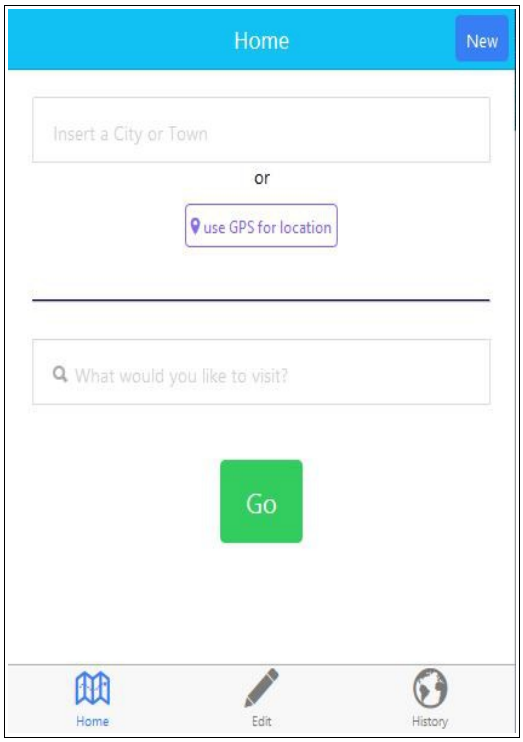


Figura 2.4 : schermata principale home, slide 2

Nel *Codice 2.3* viene mostrata l'inizializzazione del codice, all'interno del file index.html.

```
....
<!--modulo iniziale starter di AngularJS-->
  <body ng-app="starter" ng-controller="newSearch">
    <!-- The nav bar that will be updated as we navigate between
      views -->
    <ion-nav-bar class="bar-calm">
      <ion-nav-buttons side="right">
        <button class="button button-positive"
                  ng-click="newsearch()">New</button>
      </ion-nav-buttons>
    </ion-nav-bar>
    <!-- The views will be rendered in the <ion-nav-view>
      directive below. Templates are in the /templates folder. -->
    <ion-nav-view></ion-nav-view>
  </body>
....
```

Codice 2.3 : body HTML del file index.html, inizializzazione applicazione

Le schede della grafica vengono mostrate sfruttando la parte del body HTML *ion-nav-view* (*Codice 2.3*), dove vengono "inserite" le caratteristiche grafiche di ogni scheda, a seconda di quale viene visualizzata.

I file dei template delle schede contengono come primo tag proprio *ion-nav-view*, in modo da identificare la parte che verrà visualizzata in quella posizione del template principale.

Dalla scheda home si può partire a fare una ricerca, inserendo il luogo in cui ci si trova, a mano o cercandolo tramite GPS, e la richiesta di cosa si vorrebbe visitare. I risultati vengono mostrati tramite il template *results.html* (*Figure 2.5, 2.6*).

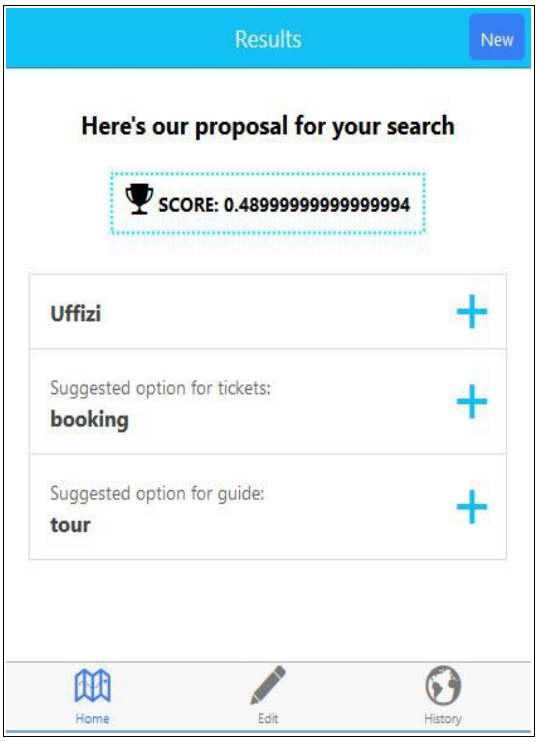


Figura 2.5 : schermata results, file results.html

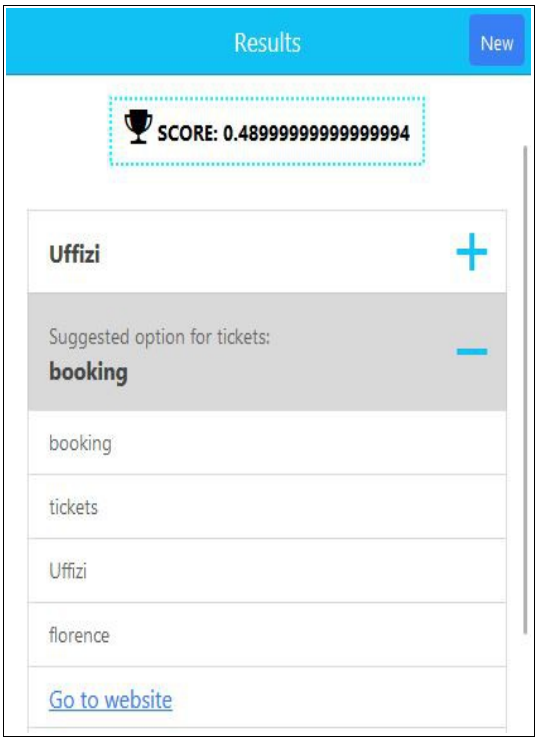


Figura 2.6 : schermata results, dettagli

La grafica a schede è composta, oltre che dalla scheda *tab-home*, anche dalle schede *tab-history*, per visualizzare le ricerche passate dell'utente, e *tab-edit*, per la modifica della composizione di servizi da parte dell'utente.

In *tab-history.html* (*Figura 2.7*), selezionando una ricerca, si possono visualizzare i risultati ottenuti quando era stata fatta la richiesta, con le rispettive informazioni. La visualizzazione dei dettagli è ad opera del template *view-past.html* (*Figura 2.8*).

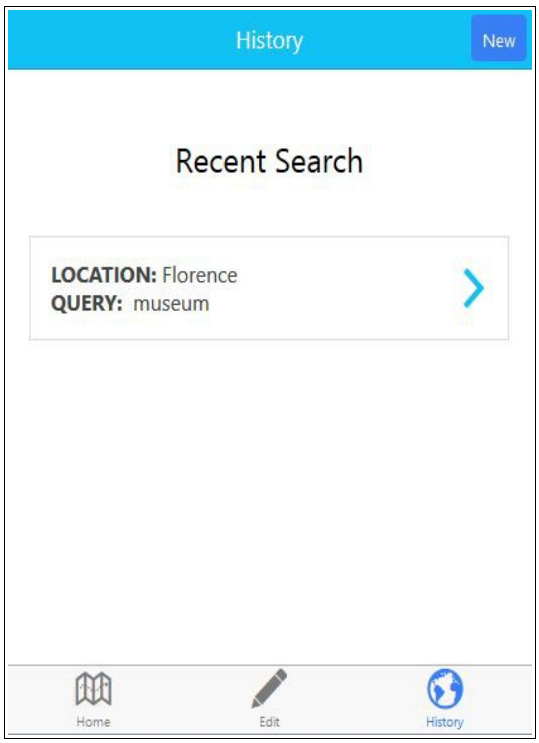


Figura 2.7 : schermata history, file tab-history.html

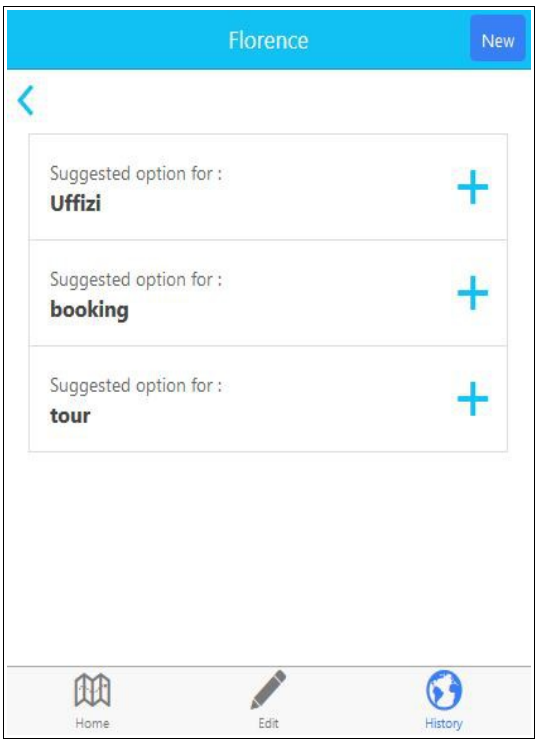


Figura 2.8: schermata history, dettagli

Tab-edit.html (*Figura 2.9*) permette di selezionare uno dei risultati forniti e di sostituirlo con un'alternativa di maggior gradimento all'utente. Per ogni alternativa vengono mostrati il punteggio e le informazioni, tramite il codice contenuto nel file *edit-res.html* (*Figura 2.10*).

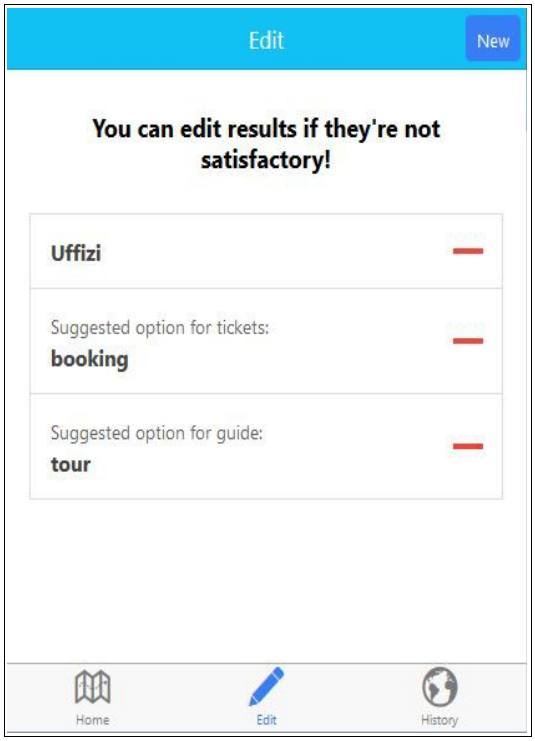


Figura 2.9 : schermata edit, file tab-edit.html

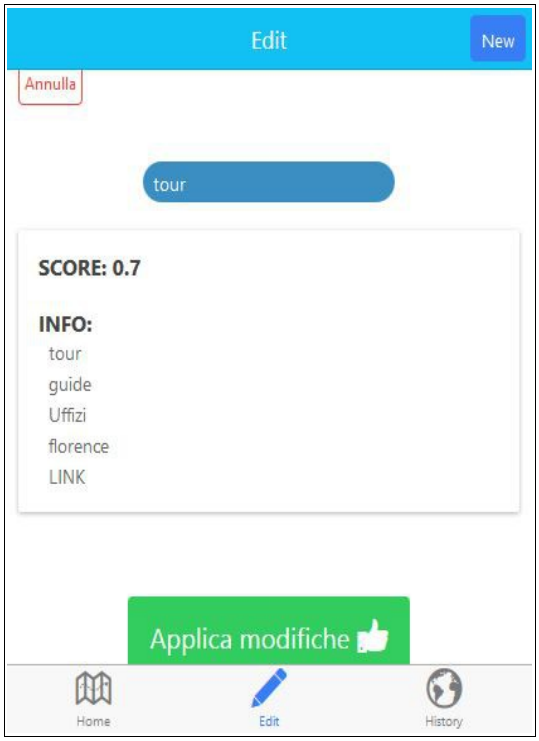


Figura 2.10: schermata edit, dettagli

2.3.2 Controller

Per far comunicare la grafica e i dati sono stati creati dei *controller*, componenti di AngularJS che gestiscono i cambiamenti nel modello dei dati e nella grafica al verificarsi di un evento. Il collegamento con i controller è realizzato all'interno del codice HTML, attraverso la direttiva *ng-controller* inserita all'interno dei tag delle sezioni. Il controller ha la funzione di "ascoltare" gli eventi che si verificano in quel determinato componente grafico e di registrarne i cambiamenti. Nel suo corpo viene gestito il cambiamento del modello dei dati attraverso codice JavaScript.

I controller sono contenuti nel file *controllers.js* (*Codice 2.4*). Essi prendono in input dei parametri e possono restituire un output. A loro volta possono comprendere definizioni di funzioni o chiamate a funzioni del modello dei dati. É in questo modo che riescono a gestire il modello dei dati.

I parametri di input sono il nome del controller, che corrisponde al nome usato nel codice HTML per richiamarlo, e una funzione. Quest'ultima ha sempre come parametro di input una variabile *$scope*, che fa riferimento all'insieme di variabili e funzioni utilizzate nel codice HTML. Altri input possono essere *$state*, che si riferisce al template grafico corrente, quello che si sta visualizzando, e *$stateParams*, variazione del precedente che indica i valori delle variabili del template corrente. Ultimo parametro che è stato molto usato nel progetto è il nome della *factory* utilizzata per creare i dati, in questo caso chiamata *Data*. Questo serve per riferirsi al modello dei dati e accedere a variabili e funzioni.

All'interno del corpo della funzione, le variabili e le funzioni riferite da *$scope, $stateParams* e *Data* vengono richiamate utilizzando una notazione puntata. Ad esempio si scriverà *$scope.results* per riferirsi alla variabile *results* comprendente la lista dei risultati visualizzata nella grafica.

```
....
.controller('EditRes', function($scope, $state, $stateParams,Data)
{

      $scope.edit = { selected: Data.get_res($stateParams.rId),
                      others: Data.get_others() };

      //serve per memorizzare valore, per usarlo dopo la conferma
      $scope.select = function() {
            $scope.newres = Data.set_sel($scope.edit.selected);
      };

      //per modificare dopo la conferma
      $scope.update = function() {
            Data.edit($scope.newres);
            $state.go('tab.edit');
      };
})

.controller('PastSearch', function($scope, $stateParams, Data) {
      //accesso alle informazioni di una specifica ricerca passata
      $scope.past = Data.get_past($stateParams.pId);
})

.controller('ResCtrl', function($scope, Data) {
      //verifico se sono stati trovati risultati
      if (Data.all_res().length > 0) {
            $scope.showres = true;
            $scope.results = Data.all_res();
            $scope.score = Data.get_score();
      }
      else {
            $scope.nores = true;
      }
})
....
```

Codice 2.4 : parte esemplificativa del file controllers.js

2.3.3 Model

Per poter far funzionare l'applicazione con dati reali, sono state prese in considerazione pagine web. Il funzionamento con questi dati è rimasto comunque esemplificativo.

Per lavorare con le informazioni contenute in tali pagine web è stata sfruttata la libreria Boilerpipe, utilizzata attraverso la corrispondente web API.
Come prima cosa sono state selezionate pagine web riguardanti servizi che potessero corrispondere a possibili query dell'utente. Da questi è stato poi estratto il testo con Boilerpipe, elaborato successivamente attraverso la libreria Topia.

Topia è stata gestita attraverso codice Python (*Codice* 2.5), nel quale è stato necessario aggiungere anche operazioni di tokenization, stemming e di eliminazione di caratteri che non fossero lettere, sfruttando la libreria NLTK.

Grazie al testo estratto con Boilerpipe e la selezione fatta da Topia, è stato possibile ottenere un elenco di caratteristiche riguardanti la pagina web di partenza. Tali caratteristiche sono state utilizzate come keyword dei servizi da analizzare per generare la risposta alla query dell'utente.

```
def elabora(f):
      fl = open(f, 'r')
      text = fl.read()
      fl.close()

      #elimino numeri e caratteri che non sono lettere
      text = "".join(ch for ch in text if ord(ch) in range(65,91)
                  or ord(ch) in range(97,123) or ord(ch) == 32)

      extractor = extract.TermExtractor()
      tupla = extractor(text)       #estraggo parole significative

      arr= list()
      for t in tupla:
            #devo restringere il range per questi file
            if (f == 'london tower-london.txt' or f == 'bargello-
                        florence.txt'
                        or f == 'egyptian museum-turin.txt'
                        or f == 'museo montagna-turin.txt' or f ==
                        'museum astronomy and space-turin.txt'):
                  if t[1] in range(4,10):
                        arr.append(t[0].lower())

            #per questo devo aumentare range
            elif f == 'cinema museum-turin.txt':
                  if t[1] in range(2,10):
                        arr.append(t[0].lower())
```

```
            else:
                  if t[1] in range(3,10):
                        arr.append(t[0].lower())

      txt = text.split()
      for i in txt:
            if (i.lower() in ['guide','guided','tickets',
                              'booking','tour','tours']):
                  arr.append(i.lower())

      s = set()   #rimuovo duplicati
      for a in arr:
            #elimino parole di solo una o due lettere
            if len(a) > 2:
                  #elimino stopwords
                  if (not a in stopwords.words('english') and
                        not a in stopwords.words('italian')):
                        if wn.morphy(a) != None:
                              #stemming
                              s.add(wn.morphy(a))
                        else:
                              s.add(a)

      arr = sorted(list(s))

      name = (os.path.splitext(basename(f)))[0]
      name = name.split('-')
      f_out.write(name[0].title().rstrip('\n')) #scrivo nome
      f_out.write('-'.rstrip('\n'))             #separo con -
      f_out.write(name[1].rstrip('\n'))         #scrivo nome città
      for k in arr:
            if k != name[1]:
                  f_out.write('-'.rstrip('\n')) #elimino il
                                          ritorno a capo alla fine
                  f_out.write(k.rstrip('\n'))
      f_out.write('\n')
```

Codice 2.5 : funzione elabora(), opera analisi lessicale, eliminazione di stopwords e stemming

I servizi sono stati creati e inseriti in una lista dopo aver ottenuto tutte le keyword relative. Anche questa operazione è stata svolta in codice Python. In questo modo è stato possibile popolare il modello dei dati in modo dinamico, in base agli input.

La parte di elaborazione dei dati, in cui vengono calcolati i pesi dei servizi e la composizione finale è stata completamente compresa nel codice Python.

Per questo motivo è stato necessario trovare un modo per far comunicare tale codice con quello principale dell'applicazione, scritto in JavaScript.
Il collegamento è stato fatto all'interno di una funzione richiamata da un controller (*Codice 2.6*). Quando l'utente inserisce la propria query e il luogo in cui si trova, il controller corrispondente ai campi inseriti raccoglie i dati e li invia ad una funzione che richiama il codice Python attraverso una richiesta jQuery, inviando i dati raccolti e aspettando in risposta il risultato dell'elaborazione.

```
....
.controller('loadRes', function($scope, $state, $ionicLoading,
Data) {
      ....
      $scope.loadResult = function(input) {
            //controllo se sono stati inseriti tutti i dati
            if ((input.city == null && $scope.nogeo) ||
                  input.query == null) {
                  alert("Please don't leave empty fields!");
            }
            else {
                  $ionicLoading.show();
                  //lista delle query inserite, per gestirne più
                  di una
                  var q = input.query.split(' ');
                  var c;
                  //se ho rilevato la posizione usando il GPS
                  if ($scope.showgeo) {
                        c =
                        document.getElementById("position").value;
                  }
                  else c = input.city;

                  var i = { city: c, query: q }
                  //elaborazione dati, richiesta verso server
                  //http://127.0.0.1:5000/process per server su
                  localhost
                  $.getJSON("http://192.168.43.129:5000/process",
                  {
                              city: i.city,
                              query: i.query,
                              user: Data.get_usr()
                        }).done(function(json) {
                              Data.set_serv(i,json.result[0]);
                              Data.set_score(json.result[1]);
                              Data.init();
                              $state.go('tab.results');
```

```
                        $ionicLoading.hide();
                    })
                    .fail(function(jqxhr,textStatus,error) {
                        var err=textStatus+', '+error;
                        alert(err);
                        //la pagina non avrà risultati
                        Data.init();
                        $state.go('tab.results');
                        $ionicLoading.hide();
                    });

                //azzero i campi
                this.input = {};
                document.getElementById("position").value =
                                                        null;
                $scope.nogeo = true;
                $scope.showgeo = false;
            }
        };
})
....
```

Codice 2.6 : funzione loadResult() per inviare al server ed elaborare la richiesta dell'utente

Per fare in modo che il codice Python potesse ricevere e inviare dati da/verso l'esterno è stato utilizzato il modulo Flask (*Codice 2.7*), che permette al codice di comportarsi come un server in ascolto delle richieste in arrivo.

```
from flask import Flask, request
#modulo per abilitare richieste cross-origin
from flask.ext.cors import CORS

app =  Flask(__name__)
CORS(app)

@app.route("/")
def hello():
      return "Welcome!"

@app.route("/process", methods=['GET','POST'])
....
if __name__ == "__main__":
      #app.run()  #run on localhost
      app.run(host= '192.168.43.129')
```

Codice 2.7 : importazione, inizializzazione ed avvio del modulo Flask

All'interno del codice Python è stata implementata quindi l'elaborazione dei pesi dei servizi in base agli input dati, cioè luogo, profilo e query dell'utente (*Codice 2.8*). A questo scopo sono state utilizzate le funzioni definite al momento della definizione della struttura base del progetto, che sfruttano il database WordNet (*Codice 2.9*).

```
def process():
      cityfromjs = request.args.get('city','')
      queryfromjs = request.args.getlist("query[]")
      userfromjs = request.args.getlist("user[]")
      ....
      for serv in services:         #assegnamento pesi
            if serv['level'] == 'main':         #primo livello
                  #il servizio deve essere nella città indicata
                  if cityfromjs.lower() in serv['infos']:
                        weight =
                        (0.5*dsim(queryfromjs,serv['infos']))
                              +(0.5*dsim(context,serv['infos']))
                  else: weight = 0.0
                  serv['score'] = weight
                  for s in services:
                        #collego gli altri livelli
                        if (s['level']!='main') and (serv['name']
                        in s['infos']):
                              #se è stato trovato match, collego
                              if weight > 0.0:
                                    s['score'] =
                              (0.5*dsim(queryfromjs,s['infos']))
                              +(0.5*dsim(context,s['infos']))
                              else:
                                    #altrimenti metto a 0 anche i
                                    collegamenti
                                    s['score'] = 0.0
....
```

Codice 2.8 : assegnamento pesi

```
def dsim(kw,serv_kw):
      out = 0
      for k in kw:
            out = out+(1*ksim(k,serv_kw))
      return out/len(kw)

def ksim(key1,key2):
      out = 0
      th = 0.6    #threshold per hsim
      for k in key2:
            h = hsim(key1,k)
            synk = []
            #isolo i nomi dei synset di k
            for s in wn.synsets(k, pos=wn.NOUN):
                  n = str(s.name())
                  n = n[:n.find('.')]
                  synk.append(n)

            if (key1 == k) or (key1 in synk):
                  if out <= 1:
                        #ogni volta aggiorno il valore di uscita a
                        quello della parola che fa più match
                        out = 1
            #se hsim maggiore della soglia, le kw sono in
            relazione
            elif h > th:
                  if out <= 0.6:
                        out = 0.6
            else:
                  if out <= 0:
                        out = 0
      return out

def hsim(key1,key2):
      sim = 0
      #scorro tutti i synsets per cercare quello più adatto
      for s in wn.synsets(key1, pos=wn.NOUN):
            for sy in wn.synsets(key2, pos=wn.NOUN):
                  lch = s.lch_similarity(sy)
                  if lch > sim:
                        #aggiorno il grado di similarità
                        sim = lch
      return sim
```

Codice 2.9: formule che utilizzano il database WordNet e le sue funzionalità

Una volta ottenuti i pesi di ciascun servizio, è stato implementato un algoritmo per la scelta della composizione che più corrispondesse agli input, cioè che producesse il punteggio maggiore combinando i servizi migliori (*Codice 2.10*).

In risposta all'applicazione è stata restituita una lista dei servizi elaborati con i rispettivi punteggi e l'indicazione di quali sono stati compresi nella composizione. Inoltre è stato fornito il punteggio finale ottenuto dalla composizione.

```
....
#divido i livelli
      main = []
      tickets = []
      guide = []
      for serv in services:
            if serv['level'] == 'main':
                  main.append(serv)

            elif serv['level'] == 'tickets':
                  tickets.append(serv)
            elif serv['level'] == 'guide':
                  guide.append(serv)

      #partendo dal livello principale provo tutte le combinazioni
      for m in main:
            serv_tick = []
            serv_guide = []
            name = m['name']
            #includo solo i servizi collegati con main
            for t in tickets:
                  if name in t['infos']:
                        serv_tick.append(t)
            for g in guide:
                  if name in g['infos']:
                        serv_guide.append(g)

            #per ogni livello cerco la combinazione migliore
            for st in serv_tick:
                  for sg in serv_guide:
                        pth = m['score']*st['score']*sg['score']

                        if pth > path:
                              mem = []
                              path = pth
                              mem.append(m)
                              mem.append(st)
                              mem.append(sg)
      for m in mem:
            for s in service:
                  if (s == m):
                        s['selected']=True

      return [service,path]
      ....
```

Codice 2.10 : algoritmo per scegliere la composizione con punteggio più alto

Ottenuta la risposta, si è fatto in modo che il codice JavaScript inizializzasse le variabili di servizi e punteggio in modo da poterli visualizzare nell'interfaccia grafica.

L'insieme delle variabili e delle funzioni che formano il modello dei dati è interamente compreso nel file *services.js* (*Codice 2.11*).

```
angular.module('starter.services', [])

.factory('Data', function() {
      //profilo utente
      var user = {
            id: 0,
            infos: ['booking','tour']
      };
      //query utente, composta da luogo e richiesta
      var input_query = {};

      //lista dei servizi da inizializzare attraverso elaborazione
      server
      var service = [];

      //risultato della composizione dei servizi scelti, settato
      con risposta server
      var score = 0.0;
      //lista d'appoggio per visualizzare i risultati nella view
      var results = [];
      //lista dei servizi alternativi a quelli proposti
      var others = [];

      //per gestire elemento selezionato per la modifica
      attraverso la view
      var sel = {};

      //lista delle ricerche recenti
      var p_search;
      //contatore per numero delle ricerche passate memorizzate
      var count;
```

Codice 2.11 : modello dei dati, variabili

2.3.4 Considerazioni

Durante l'implementazione sono state incontrate particolari difficoltà nella gestione della comunicazione tra applicazione e server.

Il modulo Flask di Python che ha permesso di creare il server funziona appoggiandosi ad un indirizzo locale su cui avviare il server. L'applicazione, per poter comunicare con esso, deve riferirsi a tale indirizzo.

Mentre il progetto era ancora in fase di realizzazione, per testare periodicamente la struttura dell'interfaccia grafica è stato utilizzato il comando *ionic serve*, che permette di visualizzare velocemente la grafica tramite un web browser.
Cercando di testare la comunicazione con il server tramite il browser è stato riscontrato un problema: il server risultava attivo e in ascolto, infatti riusciva a ricevere correttamente i dati inviati da parte dell'applicazione, ma la risposta elaborata non arrivava all'applicazione.
Dopo numerose ricerche è stata individuata l'origine del problema nel fatto che la comunicazione avvenisse tra due indirizzi di domini differenti. Infatti, per poter testare la grafica tramite browser, l'applicazione sfrutta anch'essa un indirizzo locale ed è proprio questa incompatibilità tra indirizzi locali che non permetteva una corretta comunicazione.

Il problema è stato risolto settando un parametro di funzionamento del server tramite Flask: è stato sufficiente abilitare la funzionalità *Cross-Origin Resource Sharing* (*CORS*) per permettere una comunicazione *cross-origin HTTP request*.

È importante notare che la restrizione alle richieste fatte da due domini differenti avviene soltanto ad opera dei browser, per bloccare richieste HTTP indesiderate e fare in modo di garantire un certo livello di sicurezza.
Questo aspetto non è stato preso in considerazione quando è stato utilizzato il comando *ionic run* per testare l'applicazione su dispositivo Android, una volta terminato lo sviluppo dell'applicazione. Il risultato è stato nuovamente l'impossibilità di far funzionare la comunicazione, in questo caso su dispositivo mobile.

La soluzione è stata trovata grazie ad un ragionamento fatto in termini di indirizzi: il server era sempre stato avviato su un indirizzo locale, ma naturalmente il dispositivo mobile non è in grado di riconoscere un indirizzo nella forma “localhost”. Perciò è stato sufficiente settare l'indirizzo del server in modo che questo fosse avviato sull'indirizzo pubblico del computer sul quale è stato sviluppato il software. In questo modo è stata possibile anche la comunicazione con il dispositivo mobile, dato che gli indirizzi erano a questo punto pubblici e visibili.

Un'ultima considerazione riguarda il tempo di risposta del server.
Una volta inviata la richiesta da parte dell'applicazione, è necessario attendere almeno 12 secondi prima di ottenere una risposta. Questo è dovuto al fatto che il server svolge tutte le operazioni di elaborazione: costruisce la lista dei servizi disponibili con le rispettive caratteristiche, analizza i dati passati dalla richiesta e li utilizza per assegnare i pesi ai servizi, prova tutte le combinazioni possibili dei servizi ed elabora la composizione finale.
Si nota però che, dalla seconda richiesta in poi, il server fornisce la risposta più velocemente. Infatti la costruzione della lista di servizi viene fatta soltanto durante l'elaborazione della prima richiesta e viene mantenuta tale per le successive. In questo modo si riduce il tempo di risposta perché il numero di operazioni da svolgere è minore.

3 Test dell'applicazione

I test che sono stati fatti sull'applicazione riguardano principalmente il corretto funzionamento delle formule di calcolo dei pesi e dell'algoritmo di scelta finale.

Sono stati raccolti dati riguardo alcuni servizi presenti in 4 città: Firenze, Torino, Londra e Schwangau. Sono stati scelti sia servizi molto simili tra loro sia servizi diversi.

Le istanze di test esaminate sono:

- per la città di Firenze, 55 test su 4 servizi principali (21 servizi totali)
- per la città di Torino, 74 test su 8 servizi principali (32 servizi totali)
- per la città di Londra, 53 test su 4 servizi principali (16 servizi totali)
- per la città di Schwangau, 25 test su 2 servizi principali (7 servizi totali).

In generale si possono trarre le seguenti conclusioni:

- se la città indicata dall'utente non presenta servizi, non viene fornito alcun risultato e il punteggio totale rimane a 0;

- i servizi proposti nella composizione sono al massimo uno per ogni livello, mentre i servizi alternativi vengono mostrati nella sezione di modifica;

- i servizi proposti vengono scelti in base alla similarità delle loro parole chiave con quelle inserite dall'utente e non solo in base ad un match perfetto con esse, quindi non è necessario per l'utente inserire una parola chiave esatta, ma è sufficiente che sia indicativa della tipologia di visita che desidera fare (ad esempio se l'utente inserisce “gallery”, la parola può indicare una galleria d'arte, un museo, una mostra e in generale un luogo che espone opere d'arte al pubblico);

- i servizi proposti sono scelti in buona parte per il loro match con la query, mentre, nel caso in cui la query abbia un match debole, essi sono scelti in base alla similarità con le caratteristiche del profilo utente;

- se la query dell'utente richiede un servizio non presente nella città indicata, l'applicazione cerca in ogni modo di fornire un risultato e la sua scelta viene direzionata verso il servizio che si avvicina di più alla ricerca; esso sarà scelto soprattutto in base alle preferenze indicate dal profilo utente e, dato che si collega debolmente alla query, avrà un punteggio totale basso; in ogni caso l'utente otterrà un servizio in risposta; questo vale anche se è il profilo utente che non trova match e in questo caso verrà scelto il servizio che si avvicina di più ad esso;

- alcuni punteggi finali ottenuti dai test risultano ricorrenti, cioè si ripresentano spesso in ricerche e composizioni diverse; questo è dovuto alle formule di calcolo dei pesi che, usando lo stesso metodo di calcolo per tutti i servizi, assegnano pesi "standard" quando esaminano servizi con lo stesso grado di similarità tra loro.

Di seguito sono riportate le osservazioni sui test fatti riguardo le singole città.
Per avere una visione completa dei dati ottenuti dai test, si vedano le appendici *Test Firenze, Test Torino, Test Londra* e *Test Schwangau*.

3.1 Firenze

Servizi principali disponibili a Firenze, con rispettive parole chiave:

- Accademia-florence-accademia-gallery-guide-museum-ticket-tour-work
- Bargello-florence-bronze-cellini-century-collection-courtyard-museum-sculpture-work
- Palazzo Vecchio-florence-booking-city-guide-museum-palace-ticket-time-tour-tours
- Uffizi-florence-booking-gallery-guide-museum-ticket-tour-uffizi

I servizi *Accademia* e *Uffizi* sono molto simili tra loro e nei test è stato scelto l'uno o l'altro in base a piccole caratteristiche che li distinguono.

Se la query chiede *museum* e nel profilo utente è presente la parola chiave *booking*, la scelta cade su *Uffizi*, mentre con la stessa query e *audioguide* nel profilo, la scelta cade su *Accademia*.

Nella query le parole *museum* e *gallery* identificano allo stesso modo sia Uffizi che Accademia, quindi è il profilo utente che indirizza la scelta verso uno dei due (*Tabella 3.1*).
In ogni caso essi sono quasi identici nelle parole chiave che li caratterizzano, quindi la scelta è guidata da sottigliezze.

QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
museum	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
museum gallery	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
gallery	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
museum	booking	0.48	Uffizi 1.0	Booking 0.8	Tour 0.6
gallery	booking	0.48	Uffizi 1.0	Booking 0.8	Tour 0.6
museum	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8
gallery	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8
museum gallery	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8

Tabella 3.1 : Firenze, query museum e gallery

Se il profilo utente comprende, invece, sia *booking* che *audioguide*, il servizio scelto è *Accademia* (*Tabella 3.2*).
In generale, la presenza di *booking* nel profilo utente risulta molto forte nel determinare la scelta, qualunque sia la query.

exhibition	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
palace	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
palace museum	booking	0.432	Uffizi 0.9	Booking 0.8	Tour 0.6

Tabella 3.2 : Firenze, query booking

Solo in due casi, nonostante la presenza di *booking*, è stato scelto il servizio *Accademia*. Nel primo la query richiede *accademia*, intesa come luogo di esposizione di opere d'arte, quindi la scelta è automatica, mentre nel secondo la query richiede *museum* e *collection*.

Se la query richiede un servizio non presente, l'app ne propone uno simile che, però, avrà inevitabilmente un punteggio basso. È il caso di *exhibition*. Nonostante la presenza di *audioguide*, il peso di *Accademia* viene diminuito perché le sue parole chiave non hanno un buon match con *exhibition*. Il punteggio totale risulta molto basso. Cercando di alzare i pesi degli altri livelli, la situazione viene però bilanciata (*Tabella 3.3*).

exhibition	audioguide	0.072	Accademia 0.3	Booking 0.3	Audioguide 0.8
exhibition	tickets	0.288	Accademia 0.6	Booking 0.8	Tour 0.6

Tabella 3.3 : Firenze, query exhibition

In generale si è visto come la presenza di *collection* ed *exhibition* nella query portino alla scelta di *Accademia.* Inserendo *booking* nel profilo utente si ritorna invece alla scelta di *Uffizi.* La presenza di questa parola chiave abbassa i pesi dei servizi di *Accademia* e alza quelli di *Uffizi* in modo determinante.

Se nella query è presente solo *collection*, il servizio scelto è *Bargello*, risultato che è possibile ottenere solo in questo modo perché inserendo altre parole chiave vengono inevitabilmente scelti altri servizi.

In ultima analisi si nota che la parola chiave *palace* nella query porta verso il servizio *Palazzo Vecchio*. Anche in questo caso è possibile ottenere questo risultato solo inserendo *palace.*

3.2 Torino

Servizi principali disponibili a Torino, con rispettive parole chiave:

- Automobile Museum-turin-automobile-biscaretti-car-collection
- Cinema Museum-turin-mole antonelliana-museum

- Egyptian Museum-turin-drovetti-egypt-egyptian-king-material-museo-museum-object-scholar
- Museo Arte Contemporanea-turin-art-artist-castello-castello di-centre-exhibition-museum-work
- Museo Montagna-turin-alpine-building-city-club-collection-exhibition-floor-italian-observation-year
- Museum Astronomy And Space-turin-force-installation-space-star
- Palazzo Madama-turin-art-collection-floor-museum-palazzo-room-tours-visit-work
- Venaria-turin-reggia-venaria

Per questa città i servizi sono tutti diversi, ognuno con le proprie caratteristiche.
La maggioranza sono musei, perciò i primi test fatti sono stati concentrati sulla query *museum*.
Su 17 test, i risultati trovati sono stati a parità i servizi *Cinema Museum* e *Egyptian Museum*, ognuno trovato attraverso match con profili utente diversi. In minor parte è stato proposto anche il servizio *Palazzo Madama*.

Dai test si nota che spesso i risultati ottenuti hanno gli stessi punteggi, ma formano composizioni leggermente diverse (*Tabella 3.4*).

QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
museum	tickets	0.384	Cinema Museum 0.8	Booking 0.8	Tour 0.6
museum	guide	0.384	Cinema Museum 0.8	Booking 0.6	Tour 0.8
museum	infos	0.384	Cinema Museum 0.8	Infos 0.8	Tour 0.6
collection gallery	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8

Tabella 3.4 : Torino, composizioni diverse con punteggi uguali

I test fatti per questa città mostrano anche in modo chiaro come i punteggi siano ricorrenti, riferiti a servizi diversi (*Tabella 3.5*).

collection gallery	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
collection	tickets	0.384	Automobile Museum 0.8	Infos 0.8	Virtual tour 0.6
collection	booking	0.384	Palazzo Madama 0.8	Booking 0.8	Tour 0.6

exhibition	tickets	0.384	MuseoArteContemp 0.8	Infos 0.8	Tour 0.6
exhibition	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
exhibition collection	guide	0.384	Museo Montagna 0.8	Infos 0.6	Tour 0.8

Tabella 3.5 : Torino, punteggi ricorrenti

Anche per i servizi di questa città le query *museum* e *gallery* propongono risultati equivalenti.

Osservando come cambia la composizione finale al cambiamento del profilo utente si nota come questa venga diretta verso:

- il servizio *Cinema Museum*, con profilo utente contenente *booking* o *tickets*;
- il servizio *Egyptian Museum*, con profilo utente contenente *audioguide*;
- il servizio *Palazzo Madama*, con profilo utente contenente *tour*.

Questo vale per qualsiasi query inserita dall'utente, perciò si può concludere che il profilo utente sia determinante nella scelta dei servizi.

La precedente affermazione non vale nel caso in cui la query rispecchi esattamente una caratteristica del servizio. Allora la query è più forte e prevale sul profilo utente.
Per esempio il servizio A*utomobile Museum* è caratterizzato, tra le altre, dalla parola chiave *collection* (*Tabella 3.6*), perciò inserendola nella query, la scelta risulta nella maggior parte dei casi indirizzata verso di esso. In questo modo il risultato cambia, anche se in precedenza il profilo utente *tickets* lo aveva indirizzato verso *Cinema Museum*.

museum	tickets	0.384	Cinema Museum 0.8	Booking 0.8	Tour 0.6
gallery	tickets	0.288	Cinema Museum 0.6	Booking 0.8	Tour 0.6
collection gallery	tickets	0.336	Automobile Museum 0.7	Infos 0.8	Virtual tour 0.6
collection	tickets	0.384	Automobile Museum 0.8	Infos 0.8	Virtual tour 0.6

Tabella 3.6 : Torino, query collection

Anche il servizio *Museo Montagna* è caratterizzato dalla parola chiave *collection*. Oltre ad essa, il servizio possiede anche la parola *exhibition*, quindi in caso di query che le richiede entrambe, la scelta è immediata. Il servizio si ottiene come risultato solo inserendo entrambe le query.
Se la query è composta dalla parola *exhibition*, invece, il risultato ottenuto la maggioranza delle volte è il servizio *Museo Arte Contemporanea*, indipendentemente dalle parole chiave che formano il profilo utente.

Anche in questo gruppo di test è stato notato che per una richiesta di servizio non presente nella città viene proposto il servizio che si avvicina di più al profilo dell'utente. A volte il risultato non ha nessun collegamento con la query, ma l'applicazione è stata studiata in modo che in ogni caso venga proposta una composizione. Essa rispecchia comunque le preferenze dell'utente.

In ultima analisi si può dire che i servizi rimanenti *Venaria* e *Museum Astronomy And Space* vengono compresi nella composizione solamente con query come *reggia* e *space stars*.

In conclusione, è stato dimostrato come nella maggior parte dei casi la composizione finale venga formata in base ad una parola chiave prevalente sulle altre, che sia parte della query o del profilo dell'utente. Quando in entrambi i gruppi di keywords sono presenti parole determinanti o che, al contrario, non rispecchiano alcuna caratteristica dei servizi disponibili, la scelta viene fatta attraverso il calcolo composto di tutte le keywords inserite.

3.3 Londra

Servizi principali disponibili a Londra, con rispettive parole chiave:

- History Museum-london-fee-history-information-museum-natural-transaction
- London Tower-london-armourie-display-guide-minsvisit-story-tours-tower-white
- National Gallery-london-gallery-tours

- Westminster Abbey-london-abbey-guide-tour-tours-westminster

Se la query richiede *museum* o *museum* e *gallery,* tra i risultati è predominante il servizio *History Museum*. Questo accade perché il servizio è l'unico che possiede *museum* tra le parole chiave che lo caratterizzano, perciò durante l'assegnazione dei pesi si verifica un match esatto. Gli altri servizi ottenuti con *museum* sono *National Gallery* e *Westminster Abbey,* presenti in minima parte tra i risultati.

La composizione finale proposta comprende spesso *History Museum* anche nel caso in cui il profilo utente richieda *audioguide*.

Con un profilo utente che indica *tour*, cioè la preferenza per un tour guidato, la composizione proposta è sempre diretta verso il servizio *National Gallery*, qualunque sia la query affiancata al profilo dell'utente. Nonostante 3 dei 4 servizi principali disponibili abbia *tour* tra le proprie caratteristiche, la scelta è sempre la stessa.

Il servizio *London Tower* è proposto, con diversi pesi, solo quando la ricerca dell'utente comprende *tower* oppure quando il profilo utente comprende *booking*.

In generale si può dire che, a parte alcuni casi in cui è presente un match quasi perfetto con le caratteristiche della query o del profilo utente, negli altri casi le composizioni proposte come risultato sono molto varie per questa città (*Tabella 3.7*).

QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
gallery	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
gallery	tickets	0.384	National Gallery 0.8	Infos 0.8	Tour 0.6
exhibition	tickets guide	0.343	London Tower 0.7	Booking 0.7	Audioguide 0.7
exhibition	infos	0.288	History Museum 0.6	Infos 0.8	Book 0.6
exhibition	booking tour	0.343	Westminster Abbey 0.7	Booking 0.7	Tour 0.7

Tabella 3.7 : Londra, composizioni

Per questa città i punteggi più bassi sono stati ottenuti in casi dove la query richiedeva servizi poco simili a quelli disponibili e la composizione è stata scelta guardando principalmente il profilo utente (*Tabella 3.8*).

paintings	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
exhibition	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
palace exhibition	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
palace collection	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8

Tabella 3.8 : Londra, punteggi bassi ottenuti

3.4 Schwangau

Servizi principali disponibili a Schwangau, con rispettive parole chiave:

- Hohenschwangau-schwangau-ludwig-swan
- Neuschwanstein-schwangau-castle-neuschwanstein

Questo luogo è stato compreso nei test per avere dei tipi di servizio diversi, che non fossero musei o simili, e per analizzare il comportamento dell'applicazione nel caso di due sole alternative di servizi disponibili.

I due servizi principali sono entrambi collegati alla parola chiave *castle*. Entrambi sono molto simili, anche per quanto riguarda i servizi di secondo e terzo livello ad essi collegati. Questi riguardano i servizi *infos*, cioè la possibilità di avere informazioni sui biglietti, e *tour*, cioè la presenza di tour guidati.

Proprio per questa somiglianza, i servizi sono proposti nelle composizioni finali in modo equivalente, con una piccola maggioranza di casi da parte di *Neuschwanstein.*
In generale si può dire che il servizio *Neuschwanstein* viene proposto nel caso di query che richiedono *castle* o *palace*. Invece il servizio *Hohenschwngau* viene proposto quando si richiede una mostra, un museo o un luogo in cui sia presente un'esposizione di opere d'arte (*Tabella 3.9*).

QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
castle	tickets	0.384	Neuschwanstein 0.8	Infos 0.8	Tour 0.6
castle	tour	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
palace castle	tour guide	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
palace castle	tour	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
museum	tickets	0.288	Hohenschwangau 0.6	Infos 0.8	Tour 0.6
museum	booking	0.252	Hohenschwangau 0.6	Infos 0.6	Tour 0.6
exhibition	guide	0.288	Hohenschwangau 0.6	Infos 0.6	Tour 0.8
statues	booking	0.027	Hohenschwangau 0.3	Infos 0.3	Tour 0.3

Tabella 3.9 : Schwangau, risultati ottenuti

In un test è stato ottenuto il punteggio più basso di tutti quelli analizzati per l'applicazione, pari a 0.027 (*Tabella 3.10*). Esso si riferisce ad una composizione che raggruppa servizi con pesi molto bassi, in quanto le loro caratteristiche hanno poco match sia con la query che con il profilo dell'utente. Questa è una ulteriore dimostrazione di come, nonostante la ricerca non possa produrre risultati adatti per mancanza del servizio richiesto, l'applicazione cerca in ogni caso di fornire un risultato, anche se con punteggio basso.

statues	booking	0.027	Hohenschwangau 0.3	Infos 0.3	Tour 0.3

Tabella 3.10 : Schwangau, punteggio basso

Conclusioni

Come conclusione del lavoro svolto, si può affermare che l'obiettivo del progetto è stato raggiunto.

Si è partiti dall'analisi e dalla progettazione di una app che fosse in grado di fornire informazioni e servizi all'utente che rispecchiassero nel miglior modo le sue richieste e necessità, adattandosi al suo profilo e al contesto.

Grazie allo studio di tecnologie di sviluppo e strumenti di elaborazione è stato possibile creare una applicazione con un'interfaccia grafica facile da comprendere e implementare funzionalità ad hoc per lo scopo prefisso. È stato testato che l'applicazione svolge in modo corretto i suoi compiti: essa analizza parole e termini in input ed è in grado di operare una valutazione su essi, producendo risultati soddisfacenti.

Nel corso dello sviluppo si sono incontrate difficoltà che sono state prontamente superate, rendendo l'applicazione robusta ed efficiente.

Sviluppi futuri

L'applicazione realizzata funziona al momento solo con alcuni dati d'esempio.
Alcuni sviluppi futuri per il progetto possono essere:

- uno studio per rendere l'applicazione effettivamente utilizzabile in un contesto reale e a regime;
- uno studio per aggiornare l'applicazione alla versione 2 di Ionic, rilasciata in Beta all'inizio di quest'anno e con nuove caratteristiche e funzionalità;
- l'estensione dell'applicazione alle piattaforme iOS e Windows Phone 10, quest'ultima aggiunta recentemente e utilizzabile solo con Ionic2;
- l'aggiunta di un metodo per acquisire dati sulle preferenze dell'utente;
- l'integrazione con ulteriori metodi di analisi semantica, per rendere l'offerta di servizi ancora più precisa e conforme alle caratteristiche dell'utente.

Appendici

Test Firenze

	QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
1	museum	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
2	museum gallery	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
3	gallery	booking tour	0.49	Uffizi 1.0	Booking 0.7	Tour 0.7
4	accademia	booking tour	0.144	Accademia 0.9	Booking 0.4	Tour 0.4
5	accademia museum	booking tour	0.27225	Accademia 0.9	Booking 0.55	Tour 0.55
6	museum	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8
7	gallery	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8
8	museum gallery	audioguide	0.12	Accademia 0.5	Booking 0.3	Audioguide 0.8
9	museum	booking audioguide	0.1365	Accademia 0.65	Booking 0.3	Audioguide 0.7
10	gallery	booking audioguide	0.1365	Accademia 0.65	Booking 0.3	Audioguide 0.7
11	museum gallery	booking audioguide	0.1365	Accademia 0.65	Booking 0.3	Audioguide 0.7
12	museum	booking	0.48	Uffizi 1.0	Booking 0.8	Tour 0.6
13	gallery	booking	0.48	Uffizi 1.0	Booking 0.8	Tour 0.6
14	museum	audioguide guide	0.18	Accademia 0.5	Booking 0.450	Audioguide 0.8
15	museum	audioguide tour	0.1575	Accademia 0.5	Booking 0.450	Tour 0.7
16	gallery	audioguide tour	0.1575	Accademia 0.5	Booking 0.450	Tour 0.7
17	museum gallery	audioguide tour	0.1575	Accademia 0.5	Booking 0.450	Tour 0.7
18	palace	tour	0.48	Palazzo vecchio 1.0	Infos 0.6	Tour 0.8
19	monument	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
20	palace	tickets	0.384	Palazzo vecchio 0.8	Infos 0.8	Tour 0.6
21	monument	tour	0.384	Accademia 0.8	Booking 0.6	Tour 0.8
22	exhibition	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
23	palace	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
24	palace museum	booking	0.432	Uffizi 0.9	Booking 0.8	Tour 0.6
25	statue	infos	0.288	Palazzo vecchio 0.6	Infos 0.8	Tour 0.6
26	palace museum	tour	0.48	Palazzo vecchio 1.0	Infos 0.6	Tour 0.8

27	gallery statue	tickets	0.336	Accademia 0.7	Booking 0.8	Tour 0.6
28	exhibition	audioguide	0.072	Accademia 0.3	Booking 0.3	Audioguide 0.8
29	palace gallery	tour guide	0.432	Accademia 0.9	Booking 0.6	Tour 0.8
30	palace gallery	tour	0.432	Accademia 0.9	Booking 0.6	Tour 0.8
31	paintings	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
32	paintings statue	tour guide	0.384	Accademia 0.8	Booking 0.6	Tour 0.8
33	museum	tickets tour	0.441	Accademia 0.9	Booking 0.7	Tour 0.7
34	museum gallery	tickets tour	0.441	Accademia 0.9	Booking 0.7	Tour 0.7
35	palace	tickets tour	0.441	Palazzo vecchio 0.9	Infos 0.7	Tour 0.7
36	palace	tickets	0.384	Palazzo vecchio 0.8	Infos 0.8	Tour 0.6
37	museum	tickets	0.384	Accademia 0.8	Booking 0.8	Tour 0.6
38	gallery	tickets	0.384	Accademia 0.8	Booking 0.8	Tour 0.6
39	museum gallery	tickets	0.384	Accademia 0.8	Booking 0.8	Tour 0.6
40	exhibition statue	tickets	0.288	Accademia 0.6	Booking 0.8	Tour 0.6
41	exhibition	tickets	0.288	Accademia 0.6	Booking 0.8	Tour 0.6
42	exhibition	booking guide	0.392	Uffizi 0.8	Booking 0.7	Tour 0.7
43	paintings	tickets guide	0.343	Accademia 0.7	Booking 0.7	Tour 0.7
44	statue	tickets	0.288	Accademia 0.6	Booking 0.8	Tour 0.6
45	palace statue	tickets	0.336	Palazzo vecchio 0.7	Infos 0.8	Tour 0.6
46	castle	tour	0.384	Accademia 0.8	Booking 0.6	Tour 0.8
47	museum collection	booking tour	0.441	Uffizi 0.9	Booking 0.7	Tour 0.7
48	museum collection	booking	0.432	Accademia 0.9	Booking 0.6	Tour 0.8
49	museum collection	guide	0.432	Accademia 0.9	Booking 0.6	Tour 0.8
50	collection	booking	0.384	Bargello 0.8	Booking 0.8	Tour 0.6
51	collection exhibition	booking	0.384	Uffizi 0.8	Booking 0.8	Tour 0.6
52	cathedral	tour	0.384	Accademia 0.8	Booking 0.6	Tour 0.8
53	collection	booking tour	0.392	Bargello 0.8	Booking 0.7	Tour 0.7
54	collection	tickets guide	0.392	Bargello 0.8	Booking 0.7	Tour 0.7
55	collection	infos tickets	0.336	Bargello 0.8	Booking 0.7	Tour 0.6

Test Torino

	QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
1	museum	booking	0.512	Cinema Museum 0.8	Booking 0.8	Booking 0.8
2	museum	booking tour	0.441	Palazzo Madama 0.9	Booking 0.7	Tour 0.7
3	museum	tickets	0.384	Cinema Museum 0.8	Booking 0.8	Tour 0.6
4	museum	tour	0.48	Palazzo Madama 1.0	Booking 0.6	Tour 0.8
5	museum	guide	0.384	Cinema Museum 0.8	Booking 0.6	Tour 0.8
6	museum	guide tickets	0.392	Cinema Museum 0.8	Booking 0.7	Tour 0.7
7	museum	audioguide	0.12	Egyptian Museum 0.5	Booking 0.3	Audioguide 0.8
8	museum	tour guide	0.432	Palazzo Madama 0.9	Booking 0.6	Tour 0.8
9	museum	infos	0.384	Cinema Museum 0.8	Infos 0.8	Tour 0.6
10	museum	guide booking	0.448	Cinema Museum 0.8	Booking 0.7	Booking 0.8
11	museum	audioguide tour	0.20475	Egyptian Museum 0.65	Booking 0.450	Audioguide 0.7
12	museum	audioguide booking	0.20475	Egyptian Museum 0.65	Booking 0.450	Audioguide 0.7
13	museum	infos tour	0.1365	Egyptian Museum 0.65	Booking 0.3	Audioguide 0.7
14	museum	tickets tour	0.441	Palazzo Madama 0.9	Booking 0.7	Tour 0.7
15	museum	audioguide tickets	0.1365	Egyptian Museum 0.65	Booking 0.3	Audioguide 0.7
16	museum	infos audioguide	0.20475	Egyptian Museum 0.65	Booking 0.450	Audioguide 0.7
17	museum	infos booking	0.392	Cinema Museum 0.8	Booking 0.7	Booking 0.7
18	museum gallery	booking	0.448	Cinema Museum 0.7	Booking 0.8	Booking 0.8
19	museum history	infos	0.336	Cinema Museum 0.7	Infos 0.8	Tour 0.6
20	collection	audioguide tour	0.14175	Egyptian Museum 0.450	Booking 0.450	Audioguide 0.7
21	museum collection	audioguide	0.096	Egyptian Museum 0.4	Booking 0.3	Audioguide 0.8
22	museum collection	booking	0.448	Cinema Museum 0.7	Booking 0.8	Booking 0.8
23	collection gallery	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
24	collection gallery	tickets	0.336	Automobile Museum 0.7	Infos 0.8	Virtual tour 0.6
25	gallery	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
26	collection	infos	0.384	Automobile Museum 0.8	Infos 0.8	Virtual tour 0.6
27	collection	booking audioguide	0.14175	Egyptian Museum 0.450	Infos 0.450	Audioguide 0.7
28	collection	tickets	0.384	Automobile Museum 0.8	Infos 0.8	Virtual tour 0.6

29	gallery	tickets	0.288	Cinema Museum 0.6	Booking 0.8	Tour 0.6
30	collection museum	audioguide infos	0.17325	Egyptian Museum 0.55	Booking 0.450	Audioguide 0.7
31	collection	booking	0.384	Palazzo Madama 0.8	Booking 0.8	Tour 0.6
32	collection	infos booking	0.336	Automobile Museum 0.8	Infos 0.7	Virtual tour 0.6
33	collection museum	guide infos	0.392	Palazzo Madama 0.8	Infos 0.7	Tour 0.7
34	museum	guide tickets	0.392	Cinema Museum 0.8	Booking 0.7	Tour 0.7
35	gallery	tickets audioguide	0.0945	Egyptian Museum 0.450	Booking 0.3	Audioguide 0.7
36	gallery	tickets tour	0.343	Palazzo Madama 0.7	Booking 0.7	Tour 0.7
37	gallery	booking tour	0.343	Palazzo Madama 0.7	Booking 0.7	Tour 0.7
38	exhibition	tickets	0.384	MuseoArteContemp 0.8	Infos 0.8	Tour 0.6
39	exhibition	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
40	exhibition collection	guide	0.384	Museo Montagna 0.8	Infos 0.6	Tour 0.8
41	exhibition	guide	0.384	MuseoArteContemp 0.8	Infos 0.6	Tour 0.8
42	exhibition	booking guide	0.336	Cinema Museum 0.6	Booking 0.7	Booking 0.8
43	exhibition	tickets guide	0.392	MuseoArteContemp 0.8	Infos 0.7	Tour 0.7
44	exhibition	infos guide	0.392	MuseoArteContemp 0.8	Infos 0.7	Tour 0.7
45	exhibition	infos	0.384	MuseoArteContemp 0.8	Infos 0.8	Tour 0.6
46	palace	tour	0.384	Palazzo Madama 0.8	Booking 0.6	Tour 0.8
47	palace	audioguide	0.072	Egyptian Museum 0.3	Booking 0.3	Audioguide 0.8
48	palace	infos tickets	0.288	Cinema Museum 0.6	Infos 0.8	Tour 0.6
49	palace	tickets	0.288	Cinema Museum 0.6	Booking 0.8	Tour 0.6
50	palace	guide	0.288	Cinema Museum 0.6	Booking 0.6	Tour 0.8
51	palace	infos guide	0.294	Cinema Museum 0.6	Infos 0.7	Tour 0.7
52	palace	booking tour	0.343	Palazzo Madama 0.7	Booking 0.7	Tour 0.7
53	palace	tickets tour	0.343	Palazzo Madama 0.7	Booking 0.7	Tour 0.7
54	palace	audioguide tour	0.14175	Egyptian Museum 0.450	Booking 0.450	Audioguide 0.7
55	palace	audioguide infos	0.14175	Egyptian Museum 0.450	Booking 0.450	Audioguide 0.7
56	palace	infos tour	0.343	Palazzo Madama 0.7	Infos 0.7	Tour 0.7
57	palace collection	infos	0.336	Automobile Museum 0.7	Infos 0.8	Virtual tour 0.6
58	palace exhibition	infos	0.336	MuseoArteContemp 0.7	Infos 0.8	Tour 0.6

59	castle	tour	0.384	Palazzo Madama 0.8	Booking 0.6	Tour 0.8
60	castle palace	infos	0.288	Automobile Museum 0.6	Infos 0.8	Virtual tour 0.6
61	castle	infos	0.288	Automobile Museum 0.6	Infos 0.8	Virtual tour 0.6
62	castle reggia	tour	0.20475	Venaria 0.7	Booking 0.450	Tour 0.65
63	reggia	infos	0.072	Venaria 0.8	Booking 0.3	Tour 0.3
64	reggia	booking infos	0.096	Venaria 0.8	Booking 0.4	Tour 0.3
65	reggia	booking tour	0.128	Venaria 0.8	Booking 0.4	Tour 0.4
66	museum space	booking	0.448	Cinema Museum 0.7	Booking 0.8	Booking 0.8
67	museum space	infos	0.336	Cinema Museum 0.7	Infos 0.8	Booking 0.6
68	museum space	tickets	0.336	Cinema Museum 0.7	Booking 0.8	Tour 0.6
69	exhibition space	infos	0.336	MuseoArteContemp 0.7	Infos 0.8	Tour 0.6
70	museum stars	booking	0.448	Cinema Museum 0.7	Booking 0.8	Booking 0.8
71	exhibition stars	infos	0.336	MuseoArteContemp 0.7	Infos 0.8	Tour 0.6
72	space stars	booking	0.384	Cinema Museum 0.6	Booking 0.8	Booking 0.8
73	space stars	infos	0.336	Museum Astronomy 0.7	Infos 0.8	Tour 0.6
74	space stars	tickets	0.336	Museum Astronomy 0.7	Infos 0.8	Tour 0.6

Test Londra

	QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
1	museum	booking tour	0.343	Westminster Abbey 0.7	Booking 0.7	Tour 0.7
2	museum	guide	0.384	History Museum 0.8	Infos 0.6	Book 0.8
3	museum	tickets	0.384	History Museum 0.8	Infos 0.6	Book 0.6
4	museum	tickets guide	0.392	History Museum 0.8	Infos 0.7	Book 0.7
5	museum	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
6	museum	tour tickets	0.343	National Gallery 0.7	Infos 0.7	Tour 0.7
7	museum	tour guide	0.384	Westminster Abbey 0.8	Booking 0.6	Tour 0.8
8	museum	infos	0.384	History Museum 0.8	Infos 0.8	Book 0.6
9	gallery	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
10	gallery	tickets	0.384	National Gallery 0.8	Infos 0.8	Tour 0.6
11	museum gallery	infos	0.336	History Museum 0.7	Infos 0.8	Book 0.6
12	museum gallery	tickets infos	0.336	History Museum 0.7	Infos 0.8	Book 0.6
13	museum gallery	tickets	0.336	History Museum 0.7	Infos 0.8	Book 0.6
14	museum gallery	booking	0.288	History Museum 0.7	Infos 0.8	Book 0.6
15	exhibition	tickets	0.288	History Museum 0.6	Infos 0.8	Book 0.6
16	exhibition	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
17	tower	tour tickets	0.378	London Tower 0.9	Booking 0.7	Audioguide 0.6
18	tower	guide tickets	0.441	London Tower 0.9	Booking 0.7	Audioguide 0.7
19	tower museum	booking	0.336	London Tower 0.7	Booking 0.8	Audioguide 0.6
20	jewellery museum	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
21	museum	booking	0.288	History Museum 0.8	Infos 0.6	Book 0.6
22	cathedral	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
23	cathedral	tickets guide	0.343	London Tower 0.7	Booking 0.7	Audioguide 0.7
24	abbey	booking	0.384	Westminster Abbey 0.8	Booking 0.8	Audioguide 0.6
25	cathedral abbey	tickets guide	0.392	Westminster Abbey 0.8	Booking 0.7	Audioguide 0.7
26	cathedral	tickets	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6

27	church	guide	0.384	London Tower 0.8	Booking 0.6	Audioguide 0.8
28	cathedral	tickets	0.288	History Museum 0.6	Infos 0.8	Book 0.6
29	paintings	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
30	paintings exhibition	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
31	paintings museum	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
32	paintings museum	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
33	paintings	tickets	0.288	History Museum 0.6	Infos 0.8	Book 0.6
34	paintings	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
35	paintings	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
36	exhibition	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
37	exhibition	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
38	exhibition	tickets guide	0.343	London Tower 0.7	Booking 0.7	Audioguide 0.7
39	exhibition	infos	0.288	History Museum 0.6	Infos 0.8	Book 0.6
40	exhibition	booking tour	0.343	Westminster Abbey 0.7	Booking 0.7	Tour 0.7
41	castle	guide	0.384	London Tower 0.8	Booking 0.6	Audioguide 0.8
42	collection	guide	0.384	London Tower 0.8	Booking 0.6	Audioguide 0.8
43	castle museum	guide	0.384	London Tower 0.8	Booking 0.6	Audioguide 0.8
44	castle museum	guide tickets	0.343	History Museum 0.7	Infos 0.7	Book 0.7
45	collection museum	guide booking	0.343	London Tower 0.7	Booking 0.7	Audioguide 0.7
46	palace	tour	0.384	National Gallery 0.8	Infos 0.6	Tour 0.8
47	palace	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
48	palace	infos	0.288	History Museum 0.6	Infos 0.8	Book 0.6
49	palace museum	booking	0.288	London Tower 0.6	Booking 0.8	Audioguide 0.6
50	palace exhibition	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
51	palace collection	audioguide	0.072	History Museum 0.3	Infos 0.3	Audioguide 0.8
52	palace collection	tickets	0.288	History Museum 0.6	Infos 0.8	Book 0.6
53	paintings museum	audioguide	0.096	History Museum 0.4	Infos 0.3	Audioguide 0.8

Test Schwangau

	QUERY	USER	SCORE	LIVELLO 1	LIVELLO 2 tickets	LIVELLO 3 guide
1	castle	tickets	0.384	Neuschwanstein 0.8	Infos 0.8	Tour 0.6
2	castle	tour	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
3	palace castle	tour guide	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
4	palace castle	tour	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
5	museum	tickets	0.288	Hohenschwangau 0.6	Infos 0.8	Tour 0.6
6	museum	booking	0.252	Hohenschwangau 0.6	Infos 0.6	Tour 0.6
7	exhibition	guide	0.288	Hohenschwangau 0.6	Infos 0.6	Tour 0.8
8	statues	booking	0.027	Hohenschwangau 0.3	Infos 0.3	Tour 0.3
9	palace	booking tour	0.336	Neuschwanstein 0.8	Infos 0.6	Tour 0.7
10	exhibition	booking tour	0.252	Hohenschwangau 0.6	Infos 0.6	Tour 0.7
11	palace	guide	0.384	Neuschwanstein 0.8	Infos 0.6	Tour 0.8
12	palace exhibition	tickets guide	0.343	Neuschwanstein 0.7	Infos 0.7	Tour 0.7
13	palace exhibition	guide	0.336	Neuschwanstein 0.7	Infos 0.6	Tour 0.8
14	palace exhibition	tour	0.336	Neuschwanstein 0.7	Infos 0.6	Tour 0.8
15	palace exhibition	infos	0.336	Neuschwanstein 0.7	Infos 0.8	Tour 0.6
16	museum palace	infos guide	0.343	Neuschwanstein 0.7	Infos 0.7	Tour 0.7
17	museum palace	infos	0.336	Neuschwanstein 0.7	Infos 0.7	Tour 0.7
18	castle exhibition	tour	0.336	Neuschwanstein 0.7	Infos 0.6	Tour 0.8
19	castle exhibition	infos	0.336	Neuschwanstein 0.7	Infos 0.8	Tour 0.6
20	statue	infos guide	0.294	Hohenschwangau 0.6	Infos 0.7	Tour 0.7
21	accademia	tickets	0.045	Hohenschwangau 0.3	Infos 0.5	Tour 0.3
22	gallery	booking	0.216	Hohenschwangau 0.6	Infos 0.6	Tour 0.6
23	accademia	booking	0.216	Hohenschwangau 0.6	Infos 0.6	Tour 0.6
24	accademia gallery	booking	0.091125	Hohenschwangau 0.450	Infos 0.450	Tour 0.450
25	collection	tickets	0.288	Hohenschwangau 0.6	Infos 0.8	Tour 0.6

Bibliografia

- Cabri G., Martoglia R., Zambonelli F., *Designing a Collaborative Middleware for Semantic and User-aware Service Composition*, Università di Modena e Reggio Emilia

- Marijn H., *Javascript. Guida completa per lo sviluppatore*, Milano, Ulrico Hoepli Editore (2016), pp. 438, titolo originale *Eloquent Javascript, 2nd edition*, traduzione italiana di Tadiello M.

- Schmitt C., Simpson K., *HTML5 Cookbook*, U.S.A., O'Reilly Media (2012), pp. 263

Sitografia

- AngularJS, https://angularjs.org/
  https://docs.angularjs.org/api
  http://www.html.it/guide/guida-angularjs/
  http://www.w3schools.com/angular/

- Cordova, https://cordova.apache.org/docs/en/latest/guide/overview/

- Flask, http://flask.pocoo.org/docs/0.10/api/#
  http://flask.pocoo.org/docs/0.10/patterns/jquery/
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
  https://flask-cors.readthedocs.io/en/latest/

- Ionic, http://ionicframework.com/
  http://ionicframework.com/docs/guide/
  http://ionicframework.com/docs/components/
  http://ionicframework.com/docs/api

http://www.html.it/guide/ionic-framework-la-guida/
http://www.codepen.io/ionic/pen/

- JavaScript, http://www.w3schools.com/jsref/
- jQuery, http://api.jquery.com/
- NodeJS, https://nodejs.org/en/

Altro materiale

- SRS, parte del materiale delle lezioni per il corso *Progetto del Software*, tenuto dal prof. G. Cabri
- WordNet, parte del materiale delle lezioni per il corso *Gestione Avanzata dell'Informazione*, tenuto dal prof. R. Martoglia e dalla prof.ssa F. Mandreoli

Ringraziamenti

Un sincero grazie ai miei relatori per la disponibilità e l'aiuto, senza i quali questo lavoro non sarebbe stato nemmeno pensabile. Grazie ai miei genitori, che mi hanno sempre sostenuto nelle scelte di percorso, le quali non potevano essere più diverse, e che mi hanno permesso di arrivare fino a questo punto dei miei studi dandomi sempre piena fiducia. Infine, un grazie ai miei amici più cari, per esserci sempre stati e per aver reso tutto più colorato.