Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Progettazione ed implementazione di una applicazione Android per la creazione di itinerari di viaggio

Relatore:
Dott.
LUCA BEDOGNI

Presentata da:
TOMMASO
CHERUBINI

III Sessione
Anno Accademico 2016/2017

A mia sorella, che mi accompagna da sempre.
A mia madre, la persona più forte che io conosca.
A mio padre, che mi ha insegnato il significato di sacrificio ed impegno.

Introduzione

Negli ultimi anni ha avuto luogo una grande rivoluzione nel campo della tecnologia con l'arrivo degli smartphone e di tutti i servizi che essi si portano dietro. L'esplosione di questi dispositivi ha portato alla loro diffusione rapida ed inesorabile in tutto il mondo, ormai la maggior parte della popolazione terrestre è in possesso di questi apparecchi e utilizza quotidianamente le loro applicazioni.
Allo stesso tempo, la globalizzazione e lo sviluppo di un mercato globale hanno dato luogo ad un aumento delle necessità che portano l'uomo, già di per se nomade, a viaggiare sempre più frequentemente, per motivazioni diverse tra loro, legate per lo più a curiosità e necessità. Il processo migratorio è profondamente influenzato dalle nuove tecnologie e dalle informazioni che i mass media diffondono, aumentando di fatto l'intensità dei contatti.
In questa Tesi di Laurea si è voluto unire questi due grandi punti fissi della civiltà attuale, sviluppando una applicazione Android che facilita l'organizzazione di una viaggio, occupandosi della creazione dell'itinerario migliore ottimizzando il tempo a disposizione e suggerendo le modalità di spostamento più consone.
Generalmente, le applicazioni mobili in uso nei principali market, sono volte ad automatizzare, attraverso una serie di algoritmi, delle azioni o dei servizi prima svolti senza l'aiuto della tecnologia. L'applicazione di cui questa tesi tratta è stata progettata e sviluppata per offrire alcuni dei servizi che solitamente fornisce un'agenzia turistica.
Si è fatto uso delle API offerte da Google come Google Maps Android API,

Google Maps Directions API, Google Places API for Android e Maps URLs per fornire la gamma più vasta possibile di Paesi, città e luoghi all'utente. I dati inseriti sono poi analizzati e computati per fornire finalmente un itinerario di viaggio diviso in giorni, contenente i luoghi da visitare in ordine di spostamento ottimizzato e offrendo i servizi di navigazione di Google.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Android

Android è un sistema operativo per dispositivi mobili basato su kernel Linux e sviluppato da Google. Progettato principalmente per smartphone e tablet, ha interfacce specializzate per televisori (Android TV), auto (Android Auto) e orologi da polso (Android Wear).
Il SO viene sviluppato attraverso l'Android Open Source Project ed il codice sorgente è rilasciato con licenza Open Source tranne le "Google Apps" ed i firmware non-liberi inclusi per i produttori di dispositivi.
Dal 2007, anno della prima presentazione ufficiale di Android, gli aggiornamenti per rimuovere problemi di sicurezza e migliorare le prestazioni sono stati moltissimi. Curioso è il modo di denominare ogni release del "robottino verde", ovvero attraverso nomi di dolci, in ordine alfabetico partendo dalla versione 1.5 e dalla lettera "D":

- Android 1.6 "Donut"
- Android 2.1 "Eclair"

- Android 2.2 "Froyo"
- Android 2.3 "Gingerbread"
- Android 3.0 "Honeycomb"
- Android 4.0 "Ice Cream Sandwich"
- Android 4.1 "Jelly Bean"
- Android 4.4 "KitKat"
- Android 5.0 "Lollipop"
- Android 6.0 "Marshmallow"
- Android 7.0 "Nougat"
- Android 8.0 "Oreo"

Per quanto riguarda le quote di mercato dei sistemi operativi per dispositivi mobili considerando esclusivamente gli smartphone, Android possiede oltre il 70% del mercato globale ed il suo unico competitor è iOS che ne possiede intorno al 20%. Il resto è diviso tra Windows ed altri sistemi operativi minori che per la maggior parte non superano l'1%.
Curiosa è la storia di Symbian OS, uno dei principali sistemi operativi mobile per architettura ARM tra la fine degli anni 90 e quasi tutto il decennio successivo. Sviluppato dalla Symbian Ltd., joint venture formata a giugno del 1998 da Ericson, Motorola, Nokia e Psion, tale OS prevaleva in quegli anni grazie al grande successo di Nokia. Dal 2010, in seguito all'acquisizione da parte di Nokia della totalità della società proprietaria del sistema operativo, è software libero. Dal 2011 la multinazionale ha scelto di adottare sul proprio hardware il nuovo sistema operativo di Microsoft Windows Phone, per poi esternalizzare lo sviluppo di Symbian ed infine terminare la produzione dei dispositivi Symbian nel 2013.

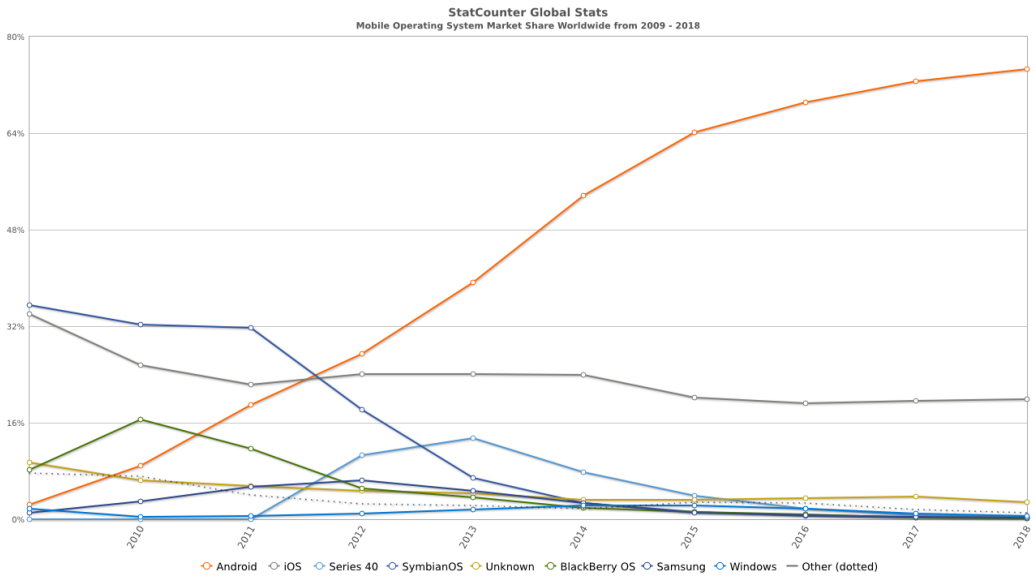

Figura 1.1: Grafico delle quote di mercato dei sistemi operativi per dispositivi mobili relativo al periodo 2009-2018 e riguardante solamente gli smartphone
Fonte: `http://gs.statcounter.com/`

Il market principale è Google Play, un negozio online di applicazioni, brani musicali, libri, riviste e pellicole cinematografiche sviluppato da Google Inc. per offrire servizi ai dispositivi mobili Android. Prima conosciuto come Android Market quando conteneva solamente applicazioni.
Esso contiene applicazioni sviluppate da terze parti ed una app che permette di scaricare ed aggiornare tali software mobile è generalmente preinstallata in ogni dispositivo Android.
Il numero di app nello store è in costante ascesa, dopo aver superato il milione nel Luglio 2013, ha recentemente raggiunto i 3.5 milioni a Dicembre 2017. La maggior parte di esse sono gratuite, secondo quanto riportato nel sito www.appbrain.com, infatti, attualmente sono più di 3 milioni quelle scaricabili senza spesa, mentre meno di 250000 quelle a pagamento. È però necessario considerare anche le applicazioni che offrono acquisti in-app, che attualmente il sito precedentemente citato afferma essere 280000.

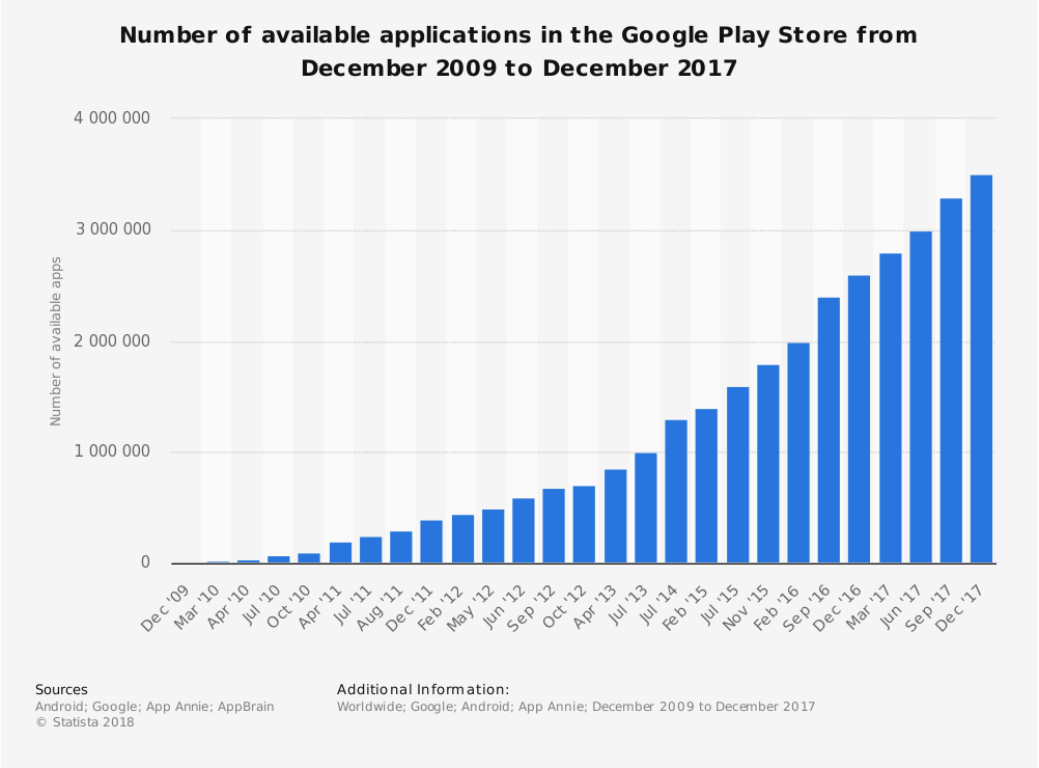

Figura 1.2: Grafico del numero di applicazioni disponibili nel Google Play Store tra Dicembre 2009 e Dicembre 2017. Fonte: `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`

1.2 Applicazioni mobili

Con il termine Applicazione o App, ci si riferisce ad un'applicazione software dedicata ai dispositivi di tipo mobile, quali smartphone o tablet. La differenza con le tradizionali applicazioni sta nel dispositivo nel quale è lanciata e nella semplificazione del contenuto, con l'obiettivo di ottenere leggerezza e velocità sia per le risorse hardware limitate del dispositivi mobili, sia per la concezione stessa di App, pensata per un uso più mirato possibile ad una determinata funzione.
Parlando specificatamente di Android, si parla di applicazioni Java-based che rappresentano il più alto livello nell'architettura del Sistema Operativo, i due principi fondanti alla base della maggior parte delle scelte progettuali sono:

- Risorse: attraverso un meccanismo di distruzione e ricostruzione di parti dell'applicazione impercettibile dall'utente, Android ha l'obiettivo di utilizzare la minor quantità possibile di risorse per permettersi di animare qualunque dispositivo in cui viene installato. Il tutto senza far perdere fluidità alla user-experience.

- Sicurezza: essendo Linux-based, Android ha nella ricerca di stabilità uno dei suoi tratti fondamentali.
  Ogni applicazione è isolata dalle altre, ogni processo provoca l'allocazione di una nuova istanza nella virtual machine, per evitare che problemi in una applicazione provochino instabilità nelle altre.
  Inoltre ogni applicazione ha un suo spazio di memoria da utilizzare per la computazione e il salvataggio dei dati ed è vietato invadere lo spazio di memoria di un'alta app.
  Questo non vuol dire che le applicazioni non possano dialogare tra loro, anzi ci sono meccanismi specifici per permettere di farlo in maniera sicura ed ottimizzata.

Ogni applicazione Android è formata dai seguenti blocchi costitutivi principali:

- Activity
- Service
- Content Provider
- Broadcast Receiver

Activity

L'Activity è il componente con cui l'utente è a contatto diretto, rappresenta l'interfaccia con la quale egli interagisce consultando dati ed inserendo input. L'applicazione è formata da diverse activity, che essendo collegate permettono all'utente di navigare tra l'una e l'altra, iniziando da quella denominata "Main", che si apre al lancio dell'applicazione.

Service

Il Service, al contrario dell'Activity, svolge il suo lavoro totalmente in background, in maniera continuativa e senza il bisogno dell'interazione dell'utente. Ha un ruolo fondamentale nella preparazione dei dati che verranno poi visualizzati dall'utente attraverso le activity.

Broadcast Receiver

Il Broadcast Receiver è un componente che reagisce all'avvenimento di un determinato evento, attraverso il Broadcast, con cui Android notifica tale avvenimento permettendo di sollecitare determinate azioni di gestione di circostanze speciali come l'arrivo di un SMS o di una chiamata.

Intent

Un Intent rappresenta l' "intenzione" di una componente di attivarne un'altra, è utilizzabile come una normale classe Java ed è la principale metodologia di comunicazione e scambio dati tra più Activity.

1.3 Applicazioni esistenti

Al giorno d'oggi nei vari market sono presenti applicazioni di qualunque tipo e per qualunque necessità, pertanto ci sono anche moltissime app riguardanti il viaggio, da quelle per la sua pianificazione a quelle utili mentre si è lontani da casa. Alcune di esse sono strumenti ormai fondamentali quando si va ad organizzare un viaggio o anche durante il viaggio stesso.
Si pensi, per esempio, ad i portali di recensioni di strutture ricettive, consultati ormai dalla maggior parte dei viaggiatori prima di decidere dove consumare un pasto, o ai famosi metamotori di ricerca di voli ed hotel, ormai indispensabili nella programmazione di un viaggio.
Oltre a questi colossi del mondo delle applicazioni per viaggiatori, gli sviluppatori nei diversi store mettono a disposizione anche una miriade di applicazioni molto meno conosciute per la semplificazione di diverse attività riguardanti il viaggio.
Partendo dalle semplici applicazioni per creare liste di attività da svolgere, fino ad arrivare a più complesse applicazioni che danno informazioni e suggerimenti sui luoghi da vedere, ormai sono innumerevoli i software disponibili ai viaggiatori.
Di seguito verranno elencate le applicazioni più utilizzate già esistenti in questo ambito divise in tre diverse categorie.

Prenotazione

Queste applicazioni offrono la possibilità di cercare, comparare ed acquistare voli, treni, bus, hotel e quant'altro. Spesso in questi portali si possono trovare sconti e promozioni esclusive.

- TripAdvisor: portale web di viaggi che pubblica le recensioni degli utenti riguardo hotel, attrazioni turistiche, appartamenti, Bed and Breakfast e ristoranti.

- Kayak: metamotore di ricerca dedicato ai viaggi, utilizzabile dagli utenti per trovare voli, hotel, autonoleggi e pacchetti vacanze.
- Airbnb: portale online che da la possibilità a persone che dispongono di un luogo da affittare di entrare in contatto con gli utenti in cerca di tale spazio.
- Booking: un fare aggregator e metamotore di ricerca per alloggi.
- Couchsurfing: un portale di scambio di ospitalità gratuito che ha creato una rete sociale di utenti che offrono e usufruiscono di servizi di alloggio, ma anche di scambio di opinioni e guide turistiche.

Preparazione

Di questa tipologia fanno parte le applicazioni ed i portali web dove vengono pubblicate recensioni da parte degli utenti riguardo ristoranti, attrazioni turistiche, hotel, appartamenti ed avvenimenti da non perdere.

- TripAdvisor: portale web di viaggi che pubblica le recensioni degli utenti riguardo hotel, attrazioni turistiche, appartamenti, Bed and Breakfast e ristoranti.
- Yelp: portale nel quale vengono pubblicate recensioni di attività locali e che permette inoltre di prenotare in tali luoghi.
- Trivago: metamotore di ricerca per il confronto dei prezzi di strutture ricettive.

Durante il viaggio

Tra i software per dispositivi mobili utili durante il viaggio spiccano quelli di indicazioni stradali, quelli per la creazione di un programma di viaggio con luoghi da visitare, avvenimenti da non perdere e ristoranti prenotati, quelli che forniscono servizi di trasporto alternativo ai mezzi tradizionali.

- Google Trips: organizzatore di eventi che utilizza le feautures di Google per offrire informazioni su città e luoghi da visitare e itinerari preimpostati.
- Google Maps: servizio che consente la ricerca e la visualizzazione di carte geografiche di buona parte della Terra, con servizi aggiuntivi come le indicazioni stradali e le informazioni sui luoghi.
- Uber: servizio di trasporto automobilistico privato che mette in collegamento diretto passeggeri e autisti.
- Packpoint: permette e facilita l'utente nel creare la sua "packing list", ovvero la lista delle cose necessarie per il viaggio.

1.3.1 Google Trips

Si vuole dare particolare importanza all'applicazione "Google Trips", causa scatenante della nascita di questo progetto di tesi in quanto in essa non è presente un generatore automatico di itinerari di viaggio personalizzati.
L'applicazione è stata presentata a Settembre 2016 da Google e risulta essere la più completa sul mercato, unendo tutte le informazioni dei database di Google su città e luoghi ad una grafica semplice ed intuitiva.
La app si presenta con una semplice domanda "Dove vuoi andare?" ed inserendo la risposta si entra in una schermata personalizzata per la destinazione divisa in sette sezioni:

- Prenotazioni
- Cose da fare
- Luoghi salvati
- Itinerari giornalieri
- Cibo e bevande

- Come spostarsi
- Informazioni utili

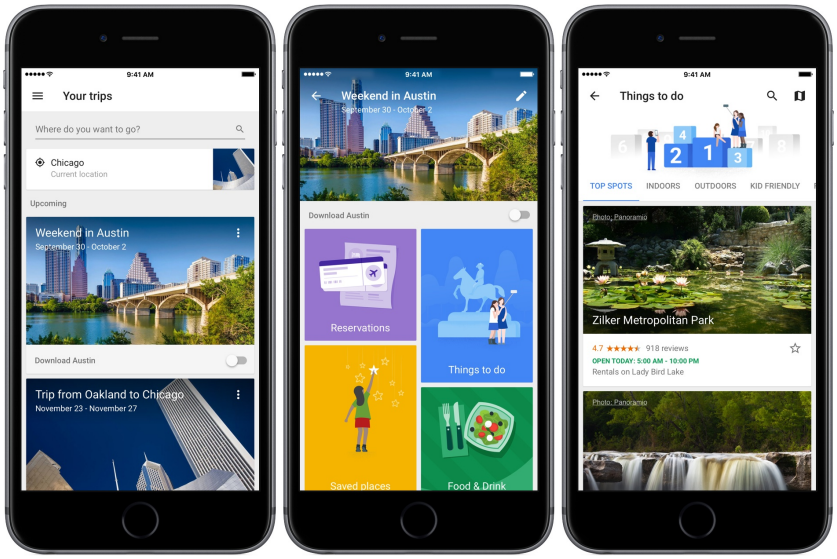

Figura 1.3: Esempi delle schermate di Google Trips

Prenotazioni

Questa prima sezione viene riempita in automatico con i dati ed i file delle diverse prenotazioni che vengono fatte dall'utente. Se gli vengono concessi i necessari permessi, infatti, la app individua da email, sms e app esterne le prenotazioni e le inserisce in questa sezione dove vengono visualizzate le informazioni più importanti.
Tipiche informazioni presentate sono i biglietti aerei o ferroviari, le prenotazioni di hotel o appartamenti e i ticket di musei.
Nel caso in cui non vengano concessi permessi particolari alla app, è comunque possibile inserire manualmente le prenotazioni attraverso un apposito form.

Cose da fare

In questa sezione vengono visualizzati i luoghi consigliati in base alle recensioni degli utenti in rete ed in base ai proprio gusti personali carpiti da Google grazie alle ricerche nel motore di ricerca.
Si potranno trovare musei, spiagge, statue e qualunque cosa sia stata suggerita dalla rete, il tutto seguito da recensioni, indicazioni stradali, sito web, foto e descrizioni.
Tali luoghi sono divisi in "Posti migliori", "Per te", "Luoghi al coperto", "Luoghi all'aperto", "Più lontani", "Luoghi A-Z", navigando tra queste categorie si potranno individuare i propri posti preferiti selezionarli per poi trovarli nella sezione corrispondente.

Luoghi salvati

In questa sezione è possibile visualizzare i luoghi marcati come speciali, leggere informazioni, orari, recensioni, accedere alle foto ed alla posizione.

Itinerari giornalieri

La sezione più interessante nella quale è possibile consultare itinerari preimpostati o creare i propri.
Se si decide di creare un itinerario personalizzato, viene presentata una mappa della città scelta come destinazione con tutti i luoghi di interesse contrassegnati con un cerchietto celeste, premendo su di essi vengono visualizzate le informazioni principali come rating ed orario ed è possibile aggiungerli al proprio itinerario giornaliero.
Tali itinerari, però, non sfruttano a pieno le possibili funzionalità di Google Maps, infatti aggiungendo ad uno ad uno i luoghi da visitare, l'ordine di visita viene scelto dall'utente e quindi si lascia a lui la pianificazione, non ottenendo probabilmente un risultato ottimale. Cosa che è possibile fare attraverso gli algoritmi di risoluzione del Traveling Salesman Problem messi a

disposizione dalle API di Google Maps ed utilizzati nel progetto di cui questa tesi tratta.

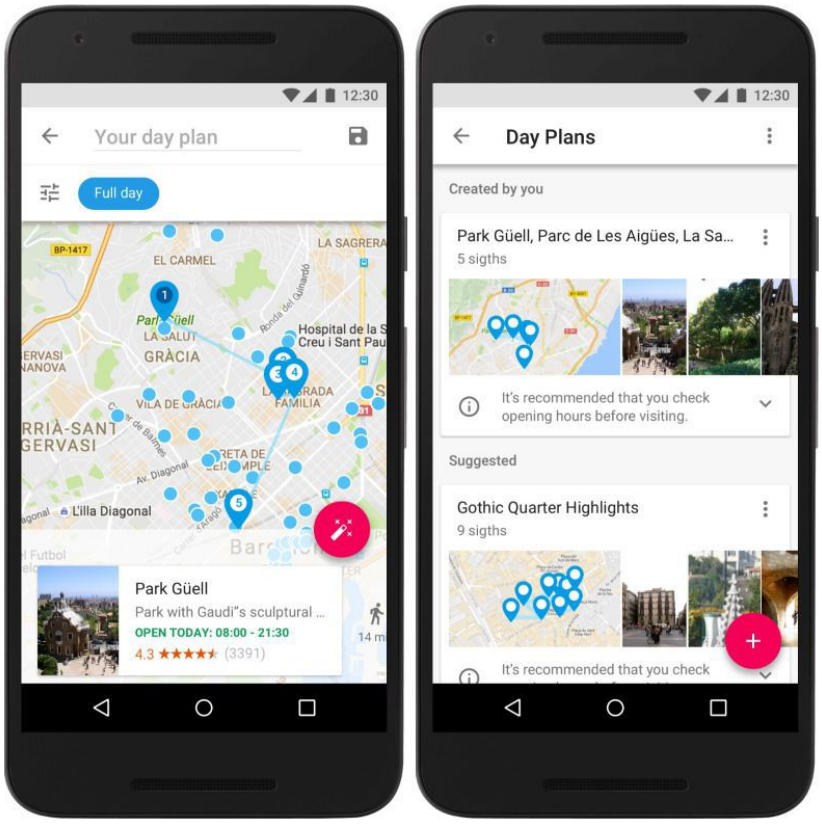

Figura 1.4: Esempio della schermata "Itinerari giornalieri" in Google Trips

Cibo e bevande

In questa sezione viene presentata una panoramica sulla ristorazione del posto, la sua storia, le influenze, le specialità locali ed infine i migliori posti dove mangiare, con le solite informazioni come recensioni, foto, indicazioni stradali ed orari.

Come spostarsi

In tale sezione è possibile consultare tutte le informazioni più importanti riguardanti il trasporto. Interessante è la parte in cui vi sono informazioni sui mezzi pubblici nella quale vengono dati orari, tariffe, linee e distanze di autobus, metropolitane, treni ed abbonamenti turistici.

Informazioni utili

Quest'ultima sezione presenta informazioni di ogni genere sulla città che si andrà a visitare, dagli orari della vita ai quartieri, dalle zone più popolate ad i centri commerciali ed i mercati.

Capitolo 2

Progettazione

In questo capitolo per prima cosa si presenta l'applicazione per poi elencarne le funzionalità, le tecnologie utilizzate e giustificare le scelte fatte.

2.1 Presentazione generale

L'applicazione mobile qui presentata è costituita da una serie di Activity di raccolta dati in cui viene chiesto all'utente di inserire le sue preferenze in quanto a destinazione, periodo, modalità di viaggio ed altre informazioni.
Completata questa fase viene attivato l'algoritmo che genera gli itinerari elaborando i dati raccolti.
Quando tale algoritmo termina, si hanno le Activity di presentazione dei risultati ottenuti, in cui si forniscono diversi servizi all'utente per facilitare i suoi movimenti nel momento effettivo del viaggio.

Raccolta informazioni

Le informazioni richieste all'utente nella prima fase sono le seguenti:

- Destinazione del viaggio, ovvero la città nella quale si troveranno i luoghi di interesse da visitare.

- Periodo del viaggio, data di partenza e di ritorno.
- Preferenze sull'algoritmo da utilizzare, parleremo di ciò in una sezione specifica.
- Ore al giorno che l'utente ha a disposizione da dedicare alle visite dei luoghi d'interesse.
- La posizione del luogo di pernottamento.
- La modalità di viaggio (in automobile,in bicicletta o a piedi).
- Luoghi che l'utente vuole visitare e tempo che ha intenzione di dedicare ad ognuno di essi.

Algoritmo di generazione degli itinerari

La generazione del programma migliore fa uso di un algoritmo che contiene al suo interno due metodi di Clustering per la creazione degli insiemi con una scelta ottimizzata dei luoghi che ne costituiscono il contenuto. In seguito vengono attuate delle HTTP Request attraverso le APIs di Google Maps per la generazione dei percorsi giornalieri ed infine, dopo diversi controlli per verificare che ogni itinerario sia consono in termini di tempo ed eventuali modifiche ad esso, vengono salvati i dati definitivi in un database per renderli fruibili in ogni momento.

Presentazione risultati

Gli itinerari generati nella fase descritta precedentemente sono fruibili in ogni momento, sarà sufficiente accedere all'applicazione e selezionare il viaggio interessato.
I risultati saranno divisi in percorsi giornalieri e per ogni giorno verrà presentata la lista di luoghi da visitare nell'ordine consigliato e ottimizzato.
Verrà inoltre data la possibilità di visualizzare la posizione dei luoghi da visitare nel giorno corrente in una mappa ed aprire automaticamente la App

di Google Maps con le indicazioni stradali a partire dalla posizione attuale e con destinazione il prossimo luogo da visitare.

2.2 Tecnologie utilizzate

In questa sezione vengono elencate e descritte le tecnologie utilizzate per lo sviluppo dell'applicazione mobile, dall'ambiente di sviluppo ai linguaggi di programmazione utilizzati, dal database ai tipi di dato di cui ci si è serviti.

2.2.1 Android Studio

Android studio è un ambiente di sviluppo integrato (IDE) per la piattaforma Android, disponibile sotto licenza gratuita Apache 2.0, annunciato da Google nel 2013 e rilasciato per la prima volta nel 2014.
Ogni progetto Android Studio contiene uno o più moduli con codici sorgente e file. Un modulo può essere di tre tipi: Android app modules, Library modules, Google App Engine modules.
L'IDE offre una vasta gamma di caratteristiche di agevolazione nella programmazione in Java, basato sul software IntelliJ IDEA, visualizzando i diversi file del progetto in una Project View organizzata per moduli. Ogni modulo della app contiene le seguenti cartelle:

- manifests: Contiene il file AndroidManifest.xml.
- java: Contiene i file sorgente in Java.
- res: Contiene tutte le risorse non in codice, come i layout xml, le stringhe UI e le immagini.

Ulteriori caratteristiche messe a disposizione da Android Studio sono:

- accesso a SDK Manager per la personalizzazione dell'SDK scaricato e AVD (Android Virtual Device) Manager per la gestione degli emulatori;

- editor per layout visuale utilizzabile in modalità drag and drop;
- supporto a ProGuard e preparazione del pacchetto di installazione;
- inline debugging, funzionalità che rende più immediata l'ispezione del codice durante il debug, affiancando alle righe di linguaggio Java i valori ed i riferimenti collegati agli oggetti;
- monitoraggio delle risorse di memoria e della CPU utilizzate dall'app, per seguire l'allocazione dinamica di oggetti ed effettuare dump della memoria heap.

2.2.2 Java

Java è il linguaggio di programmazione utilizzato per la logica dell'applicazione mobile, è un linguaggio di programmazione orientato agli oggetti, ad alto livello e a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.
Questo linguaggio risulta essere uno tra i più utilizzati al mondo, sviluppato originariamente da James Gosling ed altri ingegneri presso Sun Microsystems, acquisita nel 2010 da Oracle Corporation.
I principi fondamentali di Java sono la sua semplicità, l'orientamento agli oggetti e la familiarità; la robustezza e la sicurezza; la comprensione di strumenti e librerie per il networking; la progettazione per l'esecuzione di codice da remoto in modo sicuro; l'indipendenza dalla piattaforma.

2.2.3 XML

La presentazione dell'applicazione è sviluppata in XML(eXtensible Markup Language), un metalinguaggio per la definizione di linguaggi di markup, che consente la definizione ed il controllo degli elementi di un contenuto.
In Android Studio i layout in XML vengono progettati in una modalità che somiglia molto all'uso di HTML per le pagine web, permettendo di creare e visualizzare Layout personalizzati, Widget (Button, Label, EditText),

animazioni, forme geometriche, colori, stili, risorse e qualunque altra caratteristica presentazionale (e non solo) di un'applicazione che può essere ritenuta statica.
Il fondamentale file AndroidManifest.xml, di cui è fornita ogni applicazione, presenta le informazioni essenziali al sistema operativo del dispositivo. Tali informazioni riguardano componenti dell'applicazione, permessi, versione SDK (Software Development Kit) minima necessaria, nome del package.

2.3 Google Maps APIs

API

Con Application Programming Interface (API) si intende un insieme di procedure volte alla semplificazione nell'uso di un determinato programma. Si tratta di strumenti specifici disponibili al programmatore, funzioni o strutture dati predisposte per l'utilizzo di un dato software.

GOOGLE APIs

Le Google APIs sono un insieme di librerie messe a disposizione dal colosso di Mountain View che permettono la comunicazione con i "Google Services" e la loro integrazione con altri servizi. Esse vengono fornite attraverso il sito Google Developers, dove sono disponibili tutti gli strumenti per utilizzare i principali prodotti Google come Maps e Youtube.

GOOGLE MAPS APIs

In questa tesi è stato utilizzato un sottogruppo di tale grande insieme di librerie, le Google Maps APIs. Queste permettono la realizzazione di mappe personalizzate all'interno della propria applicazione, richiami alla app di Google Maps, esecuzione di HTTP Request e molti altri servizi.
Di questo insieme sono state utilizzate le seguenti APIs:

- Google Maps Android API
- Google Maps Directions API
- Google Places API for Android
- Maps URLs

2.3.1 Google Maps Android API

Con la Google Maps Android API è possibile aggiungere delle mappe GoogleMaps-based alla propria applicazione. La libreria gestisce automati-

camente l'accesso ai server di Google Maps, il download di dati, la visualizzazione delle mappe e i riscontri alle gesture nelle mappe.
È inoltre possibile usare le API-calls per aggiungere markers, polygons e overlays ad una mappa basica oltre a poter restringere la vista dell'utente ad un'area specifica nella mappa. Questi oggetti forniscono informazioni aggiuntive sulla localizzazione e permettono l'interazione dell'utente con la mappa.

2.3.2 Google Maps Directions API

La Google Maps Directions API è un servizio che calcola le indicazioni stradali utilizzando una HTTP Request, attraverso questa libreria è possibile:

- Specificare la modalità di trasporto tra auto, a piedi e in bicicletta.
- Integrare diverse tappe ,chiamate "waypoints", creando indicazioni stradali che forniscono un percorso a tappe.
- Specificare origine, destinazione e tappe sotto forma di testo, coordinate di latitudine e longitudine o identificatori di luoghi (place IDs)

La API restituisce i percorsi più efficienti ottimizzando tempo di viaggio, distanza, numero di svolte ecc. Il risultato viene restituito sotto forma di file json o xml, i quali vanno poi interpretati per estrapolare le informazioni necessitate dal programmatore.
È necessaria una considerazione importante sulla risoluzione del noto problema del commesso viaggiatore utilizzando tale API.

Traveling Salesman Problem con Google Maps Directions API

Il problema del commesso viaggiatore (TSP) è uno dei più famosi problemi nel campo dell'informatica teorica e della teoria della complessità. Esso consiste nel trovare il percorso di distanza minore necessario per visitare un dato insieme di città connesse tra loro tramite delle strade.
Google offre una modalità di risoluzione del Traveling Salesman Problem su

luoghi esistenti attraverso la sua libreria Google Maps Direction API, utilizzando una richiesta HTTP ed i sopracitati "waypoints". Di default il servizio calcola il percorso migliore passando per le varie tappe in ordine di comparsa nell'URL della richiesta HTTP, ma è sufficiente aggiungere a tale URL l'espressione "optimize:true" come primo parametro tra i waypoints inseriti per permettere al Direction Service di riorganizzare le tappe nell'ordine più efficiente. Questa ottimizzazione consiste in un'applicazione del problema del commesso viaggiatore.

2.3.3 Google Places API for Android

Con l'aiuto della Google Places API for Android è possibile creare applicazioni in grado di adeguarsi alla posizione del dispositivo e rintracciare i luoghi vicini.
Questo vuol dire che grazie a tale libreria è possibile arricchire la propria app con servizi di localizzazione di luoghi di interesse come negozi, ristoranti, musei e hotel. Tali luoghi sono rappresentati dall'interfaccia Place, che include informazioni come il nome, l'indirizzo, l'ID, l'URL del sito web ufficiale e molti altri.
I principali elementi per accedere a tale libreria sono il PlacePicker UI widget, l'Autocomplete UI widget, il GeoDataClient e il PlaceDetectionClient.

PlacePicker UI widget Consiste in una finestra di dialogo che permette all'utente di selezionare un Place attraverso una mappa interattiva. È possibile selezionare la posizione attuale o un luogo vicino.

Autocomplete UI widget La classe PlaceAutocomplete fornisce una Activity che permette all'utente di iniziare a scrivere il nome o l'indirizzo del Place a cui è interessato, per poi mostrare le previsioni, cioè i luoghi presenti tra i Places di Google che corrispondono alla ricerca.

GeoDataClient Tale classe rappresenta il principale metodo di accesso alla Google Places Geo Data API. Tale libreria permette l'accesso al data-

base dove Google immagazzina tutte le informazioni sulle classi Place. Attraverso l'ID del luogo è possibile procurarsi dati su di esso.

PlaceDetectionClient La classe PlaceDetectionClient fornisce il metodo di accesso principale per la Google Place Detection API, che da la possibilità di accedere velocemente al luogo in cui si trova attualmente il dispositivo.

2.3.4 Maps URLs

Grazie a Maps URLs è possibile creare un URL universale e multipiattaforma, per avviare la app di Google Maps o aprire Google Maps nel browser e svolgere ricerche, trovare indicazioni stradali o semplicemente visualizzare la mappa in una data posizione.

2.3.5 JSON

JSON (JavaScript Object Notation) è un formato particolarmente adatto all'interscambio di dati fra applicazioni client-server. JSON è un'alternativa molto comune al formato XML-XSLT, uno stream di tale formato è di solito restituito da una richiesta HTTP come risultato.
Nell'applicazione, infatti, le richieste HTTP effettuate attraverso le Google Maps APIs restituiscono uno stream JSON contenente i percorsi di viaggio in una struttura appositamente pensata. I dati vengono poi estratti dal file JSON attraverso un'apposita funzione ed analizzati a dovere per poi essere utilizzati.

2.4 Database

Dopo l'elaborazione dei dati e il raggiungimento dei risultati ottenuti, essi vengono salvati in un database per renderli fruibili all'utente ad ogni apertura dell'applicazione mobile.

La tecnologia utilizzata è SQLite, una libreria software che permette di gestire un database relazionale in un unico file. L'uso di tale libreria è nato dalla necessità da parte dei programmatori di utilizzare anche in Android i database relazionali e l'interazione tramite SQL. La soluzione a tali necessità era già presente nel mondo del software libero ed era appunto SQLite. La libreria è già presente nel sistema operativo Android e le API disponibili nel framework offrono tutto il supporto necessario.
Nella app del progetto di tesi questa libreria viene utilizzata per salvare i dati di ogni viaggio e permettere all'utente di aprire la app e visualizzare gli itinerari già calcolati in precedenza.

2.4.1 Struttura database

A questo punto viene descritta l'organizzazione del database nel progetto di tesi.

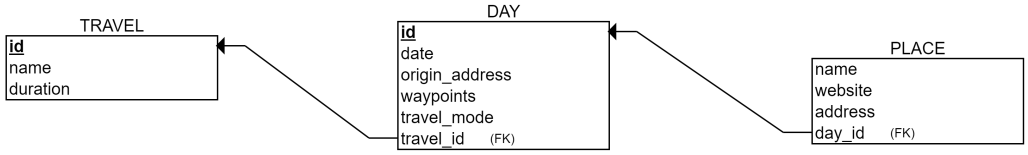

Figura 2.1: Schema relazionale del database prodotto attraverso `https://www.erdplus.com/`

Si utilizzano tre tabelle:

TABLE TRAVEL

Questa tabella contiene le informazioni generali sui viaggi, necessarie per il caricamento della lista degli itinerari già calcolati nella MainActivity all'apertura della app. Gli attributi sono:

- id: chiave primaria della tabella ed identificatore univoco di ogni viaggio.

- name: la destinazione del viaggio, ovvero ciò che compare nella RecyclerView all'apertura dell'applicazione.

- duration: la durata del viaggio in giorni, dato necessario a scopi implementativi.

TABLE DAY

Tale tabella contiene i dati necessari al caricamento di ogni itinerario giornaliero, insieme alla TABLE PLACE permette di visualizzare la schermata con la lista dei luoghi da visitare in un dato giorno e di ricavare i parametri per l'apertura di Google Maps attraverso Maps URLs. Gli attributi di questa tabella sono:

- id: chiave primaria della tabella ed identificatore univoco di ogni cluster giornaliero, equivale all'id del clusterhead dell'insieme dei Place di quel giorno.

- date: la data del giorno di viaggio.

- origin address: l'indirizzo del punto di partenza, ovvero del luogo di pernottamento.

- waypoints: la lista dei luoghi da visitare.

- travelmode: la modalità di spostamento scelta dall'utente.

- travel id: chiave esterna e identificatore viaggio a cui appartiene l'itinerario giornaliero descritto in questa tabella.

TABLE PLACE

Tabella che contiene i dati di un luogo, tali dati vengono utilizzati nell'activity di esposizione dell'itinerario giornaliero, per fornire i servizi sopra elencati insieme alla tabella precedente. Gli attributi di questa tabella sono:

- name: il nome del luogo, ciò che identifica l'elemento nella RecycerView.
- website: l'URL del sito web ufficiale del luogo, per permettere di aprirlo direttamente dall'applicazione.
- address: l'indirizzo del luogo, che permette di risalire alla sua posizione e dare le indicazioni stradali adatte.
- day id: chiave esterna che permette di sapere a quale itinerario giornaliero appartiene tale luogo.

Capitolo 3

Implementazione

Questo capitolo serve a descrivere nel dettaglio la fase di implementazione del progetto, spiegando le scelte fatte facendosi anche aiutare da pezzi di codice sorgente. Verranno presentati dettagliatamente i metodi implementati, il loro funzionamento e i risultati ottenuti attraverso il loro sviluppo.
L'ordine di esposizione è quello che si avrebbe avviando la app e procedendo con il suo utilizzo principale, ipotizzando di essere un utente che ne fa uso.

3.1 Primo impatto e raccolta dati

Per prima cosa, dopo aver parlato della MainActivity, si descrivono i meccanismi di raccolta dati attraverso una serie di schermate che accompagnano l'utente in un percorso guidato chiedendogli di inserire le informazioni necessitate dall'algoritmo affinché possa mettere in atto le sue funzioni.

3.1.1 MainActivity

La prima activity nella quale ci si imbatte aprendo l'applicazione è la MainActivity, se la app è già stata utilizzata sul dispositivo corrente allora saranno visibili gli itinerari di viaggio già calcolati precedentemente, mostrati attraverso una RecyclerView. In ogni caso, è inoltre visibile un Floating

Action Button (o FAB) che permette di iniziare il processo di inserimento dei dati necessari per la creazione di un nuovo itinerario di viaggio.

RecyclerView

Una RecyclerView è un modello fornito dalle librerie di supporto Android per permettere al programmatore di visualizzare in una schermata una sequenza di dati, magari di numero indefinito e prelevati dinamicamente. Questo elemento mostra una collezione di View disposte verticalmente e scrollabile, più flessibile ed ottimizzato rispetto alla ListView.
La RecyclerView si compone di un apposito widget, da utilizzare all'interno del layout dell'activity e di una serie di classi utilizzabili per la sua gestione. Per il funzionamento di una RecyclerView la classe fondamentale è l'adapter, il quale fornisce un legame tra l'insieme di dati da visualizzare e la RecyclerView stessa. All'interno di questo adapter vengono create le connessioni necessarie e impostati i canoni di visualizzazione attraverso la classe RecyclerView.ViewHolder ed un apposito file di layout.
In questo caso specifico, il layout viene implementato in modo da visualizzare una lista di viaggi etichettati per nome della città di destinazione. Nel caso in cui l'utente faccia click su uno degli elementi, l'adapter, si occupa di riconoscere quale degli elementi è stato premuto e apre l'activity TravelView con gli Extra necessari alla visualizzazione dell'itinerario corretto.

3.1.2 Scelta della destinazione

In questa activity, attraverso un Autocomplete Widget, si da la possibilità all'utente di selezionare la città di destinazione tra la vastissima gamma di oggetti Place contenuti nel database di Google.
Un Autocomplete Widget è un dialogo di ricerca con funzionalità di autocompletamento built-in, mentre l'utente inserisce i termini di ricerca il widget mostra una lista di predizioni dei luoghi tra i quali egli dovrà scegliere quello desiderato. Nel momento in cui l'utente compie la sua scelta, l'istanza Place

viene restituita per poi essere utilizzata per estrapolare informazioni sul luogo di destinazione.
Ci sono due modi per inserire tale widget in una app:

- Incorporazione di un PlaceAutocompleteFragment nell'activity.
- Utilizzo di un intent per avviare una autocomplete activity.

In questo caso si è fatto uso della prima opzione in quanto non vi era necessità di cambiare activity essendo questa una selezione unica, senza dover dare la possibilità di compierne altre successivamente.
Nel momento in cui l'utente compie la selezione, viene reso visibile il bottone "Next" che permette di passare all'activity successiva inserendo come Extra il Place selezionato come destinazione.

Extra

Un Extra è un valore che viene gestito negli intent con il metodo put, che permette di inserire un valore etichettato con una chiave e con il corrispondente metodo get che permette di prelevare il valore, richiedendolo mediante la chiave di riconoscimento. Da questa activity fino all'inizio dell'algoritmo nella ResumeActivity, i dati inseriti a mano a mano dall'utente verranno trasferiti da un'activity alla successiva attraverso il meccanismo degli Extra, fino ad arrivare all'ultima avendo tutte le informazioni necessarie per l'avvio dell'algoritmo principale.

3.1.3 Selezione del periodo di durata del viaggio

In questa activity, per dare la possibilità all'utente di selezionare la data di partenza e la data di ritorno, vengono utilizzati in modo combinato un elemento di tipo EditText ed un DatePickerDialog.
Un DatePickerDialog è una semplice finestra di dialogo in cui è presente un DatePicker, cioè un widget che permette di selezionare una data cliccando su di essa in un calendario.

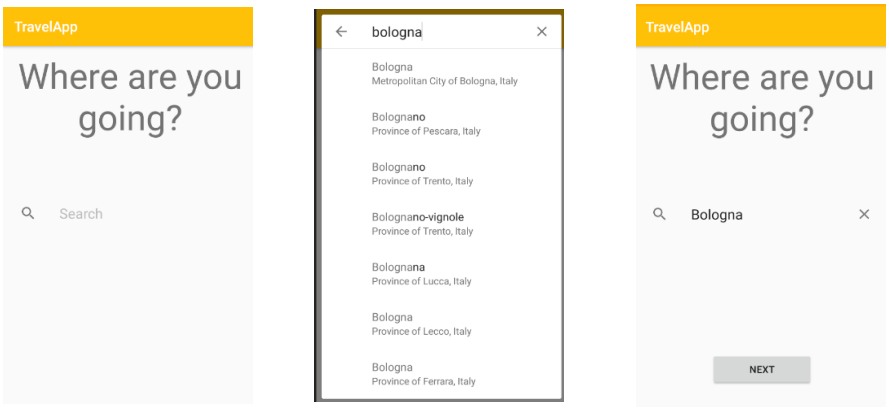

Figura 3.1: Screenshot della selezione della città di destinazione attraverso un AutocompleteFragment

Tale DatePickerDialog viene visualizzato quando l'utente seleziona l'EditText della data di partenza o arrivo e dopo che egli ha selezionato la data desiderata, essa viene visualizzata nell'EditText.
Naturalmente è presente un metodo di controllo delle date, che non permette di selezionare la data di ritorno anteriore a quella di partenza e le due date anteriori alla data odierna.
Quando l'utente ha scelto entrambe le date ed esse sono consone, appare il bottone "Next" per avviare l'activity seguente.

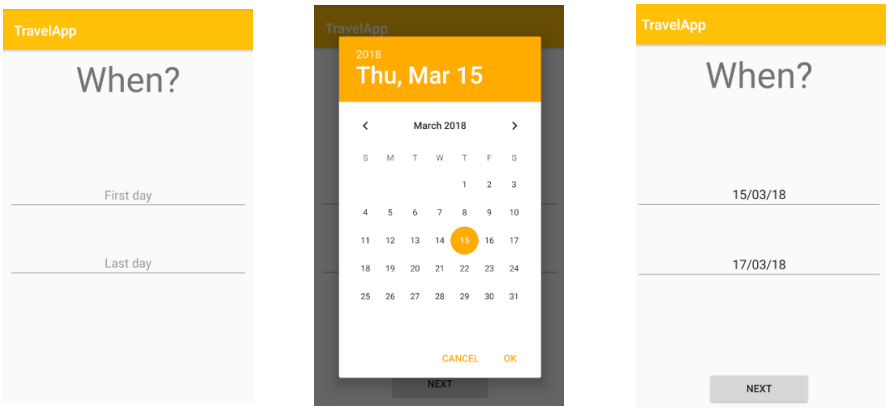

Figura 3.2: Screenshot della selezione delle date di partenza e ritorno attraverso EditText e DatePickerDialog

3.1.4 Scelta dell'algoritmo

In questa activity, attraverso dei radio button, si permette all'utente di scegliere tra 3 diverse metodologie di eliminazione/spostamento dei Place in caso di superamento del limite di ore giornaliero a disposizione.
Per prima cosa l'utente sceglie se considerare il tempo di spostamento dell'utente da un luogo ad un altro nel tempo giornaliero che mette a disposizione. Se decide di non considerarlo, si passa direttamente all'activity seguente in quanto si utilizzerà un algoritmo che nel caso in cui il totale giornaliero di ore dedicate alle visite dei luoghi di interesse superi le ore a disposizione, il Place più lontano dagli altri viene spostato nel cluster di un altro giorno, ovvero quello più vicino alla sua posizione non considerando quello dal quale viene eliminato. In seguito a tale operazione, vengono riordinati i due cluster interessati e viene riapplicato il controllo della durata.
Se invece la decisione cade sulla considerazione del tempo di spostamento, allora l'utente si trova davanti ad una seconda scelta, ovvero se vuole utilizzare l'algoritmo "Ranking" o quello "Optimized". Il primo, sfruttando l'ordine di inserimento dei luoghi da visitare, crea una scala di importanza che verrà poi utilizzata nel caso in cui si sfori il limite giornaliero di ore disponibili alle visite. In questo caso, si elimina dai luoghi da visitare quello con il ranking minore e si riavvia il procedimento di clustering dall'inizio, ricreando ogni cluster. Il secondo, invece, attua lo stesso algoritmo utilizzato in caso di non considerazione del tempo di viaggio, spostando i Place in eccesso da un cluster all'altro fino all'eliminazione in caso di rifiuto da parte di tutti i cluster. Si tratterà questo argomento in modo molto più approfondito in seguito.

3.1.5 Tempo a disposizione giornaliero

A questo punto la app chiede all'utente di inserire il numero di ore al giorno che intende dedicare alla visita. Questo inserimento avviene attraverso un elemento di tipo NumberPicker che ammette entrate da 1 a 12. Questo dato sarà fondamentale per la creazione del cluster, perché causerà lo spo-

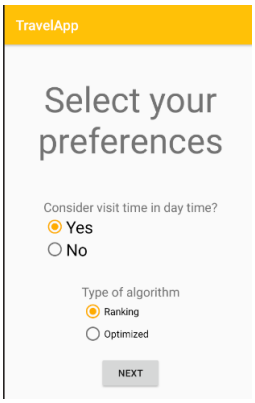

Figura 3.3: Screenshot della scelta dell'algoritmo attraverso dei RadioButton

stamento dei Place da un cluster all'altro nel caso in cui il tempo di visita totale eccederà il tempo a disposizione inserito in questa activity.

Figura 3.4: Screenshot della selezione del tempo a disposizione giornaliero attraverso un NumberPicker

3.1.6 Luogo di pernottamento

In questa fase viene inserito dall'utente l'indirizzo del luogo di pernottamento. Tale posizione verrà considerata come punto di partenza e di arrivo negli itinerari giornalieri. Anche in questo caso l'elemento utilizzato è un

AutocompleteFragment il quale permette di selezionare un Place tra quelli presenti nel database di Google.

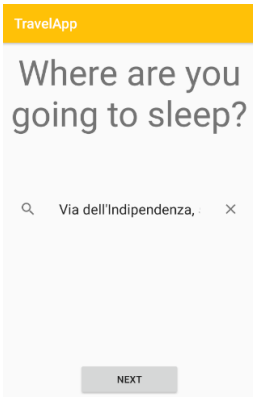

Figura 3.5: Screenshot della selezione del luogo di pernottamento attraverso un AutocompleteFragment

A tale elemento viene applicato un filtro che permette di selezionare luoghi nel raggio di 50 chilometri dal centro della città precedentemente scelta attraverso i metodi setBoundsBias() e controlPlace().
Il metodo setBoundsBias() prende in input un elemento di tipo LatLngBounds e permette di influenzare la scelta dell'utente suggerendo nell'AutocompleteFragment solamente i Place all'interno di un perimetro, fissato dal parametro in input. In questo caso viene passato in input un perimetro di 50 chilometri dal centro della città da visitare.

Listing 3.1: Applicazione del metodo setBoundBias sull'AutocompleteFragment

```
final PlaceAutocompleteFragment autocompleteFragment=
    (PlaceAutocompleteFragment)getFragmentManager()
.findFragmentById(R.id.place_autocomplete_fragment);

autocompleteFragment.setBoundsBias(new LatLngBounds(
```

```
new LatLng(cityLatLng.latitude - 0.5 , cityLatLng.longitude),
new LatLng(cityLatLng.latitude + 0.5 , cityLatLng.longitude)));
```

Il metodo controlPlace() invece prende in input un oggetto di tipo Place e controlla che esso sia effettivamente nel raggio di 50 chilometri dal centro città. Esso viene applicato dopo la selezione dell'utente, ricevendo come input l'oggetto Place restituito dall' AutocompleteFragment, se esso è accettato, verrà mostrato il bottone "Next" e verrà permesso di passare all'activity successiva inserendo tra gli Extra il luogo selezionato.

Listing 3.2: Metodo controlPlace

```
public Boolean controlPlace(Place place){

  if ( (((place.getLatLng().latitude) - (cityLatLng.latitude))
      < 0.5) && (((place.getLatLng().longitude) -
      (cityLatLng.longitude)) < 0.5)){

    return true;
  }
  else{

    return false;
  }
}
```

3.1.7 Modalità di spostamento

Attraverso tre Radio Button l'utente è chiamato a scegliere la modalità di spostamento da un punto di interesse ad un altro, le possibilità sono:

- In automobile
- In bicicletta

- A piedi

Questa scelta influenzerà la successiva creazione degli itinerari nel tempo di spostamento e nel percorso migliore.

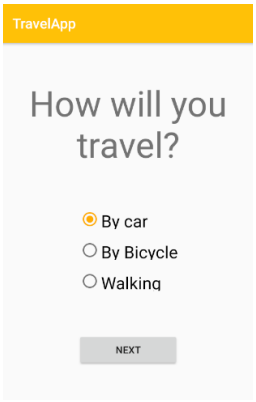

Figura 3.6: Screenshot della scelta della modalità di spostamento attraverso dei RadioButton

3.1.8 Luoghi da visitare

Questa è una delle activity più importanti, ovvero quella in cui l'utente seleziona, uno alla volta, i Place da inserire nella lista dei punti di interesse. Questi andranno a formare le varie tappe dell'itinerario.
Alla pressione del bottone "Add place" viene utilizzato un intent per avviare una autocomplete activity nella quale avverrà la selezione di ogni luogo da visitare, accettando, come nel caso del luogo di pernottamento, solo luoghi nel raggio di 50 chilometri dal centro città. Subito dopo la scelta, attraverso un AlertDialog, viene richiesto il tempo da dedicare alla visita del Place selezionato.
Il luogo ed il tempo di visita vengono aggiunti alla lista e visualizzati insieme agli altri in una RecyclerView appositamente creata. Quando l'utente è soddisfatto della sua lista di Place, premendo il pulsante "Next" accede alla activity successiva.

Per l'implementazione di questa parte del programma è stata creata una nuova classe, chiamata PlaceToVisit, per il salvataggio delle informazioni riguardanti ogni singolo luogo da visitare.
Ogni istanza della classe PlaceToVisit ha i seguenti attributi:

- name: il nome del luogo da visitare, che verrà visualizzato in caratteri grandi nella RecyclerView.
- address: l'indirizzo che verrà poi utilizzato per la creazione dei percorsi.
- id: l'identificativo corrispondente al place id di Google Places API.
- duration: la durata della visita del luogo inserita manualmente dall'utente.
- website: l'URL del sito web ufficiale.
- ranking: un intero, necessario per l'implementazione di uno dei 3 diversi algoritmi utilizzabili.
- phonenumber: il numero di telefono del luogo, fornito anch'esso dal database di Google.

Ogni volta che, seguendo il procedimento descritto in precedenza, l'utente seleziona un Place ed inserisce il tempo di visita, viene creata una nuova istanza della classe PlaceToVisit che viene subito aggiunta ad una lista listItems di tipo ArrayList di oggetti PlaceToVisit.
Questa lista, oltre ad essere inserita come Extra nell'intent per comunicare tutti i luoghi da visitare all'activity successiva, è quella da cui l'adapter della RecyclerView visualizzata nella schermata corrente acquisisce i dati necessari alla rappresentazione delle diverse card che lo compongono.
Ad ogni inserimento nella lista, infatti, l'adapter viene avvertito, la RecyclerView viene aggiornata insieme alla ProgressBar raffigurante l'incremento del tempo di visita dei luoghi selezionati rispetto al tempo totale a disposizione.

Durante il processo immissione dei luoghi da visitare vengono messi in atto dei controlli per far si che non venga superato il tempo di visita totale che si ha a disposizione durante il viaggio.
Se invece il tempo di visita di un PlaceToVisit sfora le ore giornaliere messe a disposizione dall'utente, tale oggetto viene diviso in più oggetti che verranno poi trattati come luoghi separati, questo permette all'utente di decidere di passare più di un giorno in un dato luogo.

Listing 3.3: Creazione nuova istanza PlaceToVisit. Aggiunta di tale istanza alla lista dei luoghi da visitare ed aggiornamento vista RecyclerView e ProgressBar.

```
PlaceToVisit newPlace = new PlaceToVisit(place.getName(),
    place.getAddress(), place.getId(), place.getPhoneNumber(),
    visitDuration, stringUri, countRanking);

listItems.add(newPlace);

adapter.notifyItemInserted(listItems.size() - 1);

adapter.notifyDataSetChanged();

Integer aumento=progressBar.getProgress() + visitDuration;
progressBar.setProgress(aumento);
```

3.1.9 Riepilogo dati inseriti

Questa è l'activity finale prima dell'attivazione dell'algoritmo di generazione dell'itinerario, permette di visualizzare tutti i dati inseriti prima di avviare il calcolo e raffigurare i risultati.

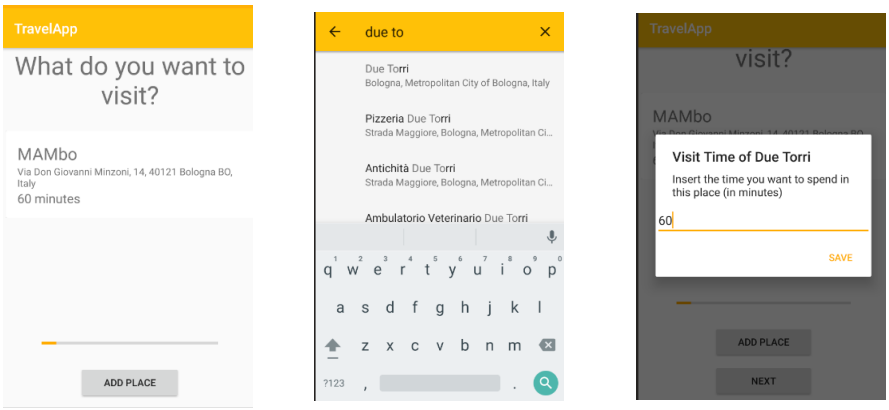

Figura 3.7: Screenshot dell'inserimento dei luoghi da visitare attraverso Autocomplete Activity e AlertDialog.

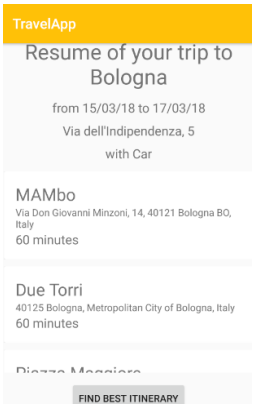

Figura 3.8: Screenshot del riepilogo dei dati inseriti

3.2 Algoritmo di Clustering

Nel momento in cui, dalla ResumeActivity, viene premuto il bottone "Find best itinerary" si avvia l'algoritmo principale dell'applicazione che porterà alla creazione dell'itinerario di viaggio.
Viene subito effettuato un processo di clustering, ovvero un insieme di tecniche per l'analisi, la selezione e il raggruppamento di elementi in un insieme di dati in base a determinati criteri.
Nel progetto viene adottata una tecnica di clustering divisa in due metodi

principali, il primo per la selezione dei Clusterhead, cioè gli elementi "capogruppo" dei sottoinsiemi da creare, il secondo per l'attribuzione di ogni PlaceToVisit al cluster adatto.

3.2.1 Metodo "Clustering1"

In generale, questo primo metodo si occupa della selezione dei luoghi più lontani tra loro così da renderli "capogruppo", o tecnicamente "clusterhead", di ogni insieme di luoghi.
Ci sarà un gruppo di luoghi da visitare per ogni giorno di viaggio, quindi è necessario lo stesso numero di clusterhead. L'idea è quella di scegliere, appunto, i luoghi più distanti dagli altri per poi utilizzarli come punti di riferimento per gli insiemi.
In termini più tecnici, questo metodo prende in input la lista di PlaceToVisit selezionati dall'utente nella precedente activity e restituisce in output una lista di oggetti PlaceToVisit contenente i clusterhead.
Per farlo fa uso di un ciclo for esterno con un numero di cicli uguale alla durata in giorni del viaggio ed un ciclo for interno che esplora ed analizza tutti i PlaceToVisit della lista ricevuta in input. Il primo for è giustificato dal fatto che servirà creare un cluster per ogni giorno di viaggio, in quanto ogni itinerario giornaliero necessita di un insieme di luoghi da visitare.
Il secondo, invece, servirà a trovare il PlaceToVisit più lontano da tutti gli altri per farne un clusterhead. Per conseguire il primo clusterhead si sceglie quello di posizione più lontana rispetto al luogo di pernottamento, per i seguenti, invece, iterando su ogni PlaceToVisit si sommano le distanze tra esso e i clusterhead già trovati, si trova la somma maggiore e si aggiunge il luogo corrispondente alla lista del clusterhead, chiamata nel codice "listaPoli". Quando la dimensione di tale lista è uguale alla durata del viaggio in giorni il metodo termina e la restituisce.

Listing 3.4: Selezione dei ClusterHead attraverso una comparazione delle distanze tra essi.

```
for (int x = 0; x <= travelDuration - 1; x++) {
  max = 0;
  maxIndex = 1000;
  for (int y = 0; y <= inputList.size() - 1; y++) {
    currentPlace = inputList.get(y);
    currentLatLng = getLocationFromAddress(context,
        currentPlace.address.toString());
    currentLocation = new Location("Current Point");
    currentLocation.setLatitude(currentLatLng.latitude);
    currentLocation.setLongitude(currentLatLng.longitude);
    currentDistance = currentLocation.distanceTo(sleepLocation);
    if (listaPoli.size() != 0) {
      for (int z = 0; z <= listaPoli.size() - 1; z++) {
        LatLng temporalLatLng = getLocationFromAddress(context,
            listaPoli.get(z).address.toString());
        Location temporalLocation = new Location("Temporal
            Point");
        temporalLocation.setLatitude(temporalLatLng.latitude);
        temporalLocation.setLongitude(temporalLatLng.longitude);
        currentDistance = currentDistance +
            currentLocation.distanceTo(temporalLocation);
      }
    }
    int resultCompare = Float.compare(currentDistance, max);
    if (resultCompare > 0) {
      max = currentDistance;
      maxPlace = new PlaceToVisit(currentPlace.name,
          currentPlace.address, currentPlace.id,
          currentPlace.phonenumber, currentPlace.duration,
          currentPlace.website, currentPlace.ranking);
      maxIndex = y;
    }
  }
```

```
    if (maxIndex != 1000) {
      listaPoli.add(maxPlace);
      inputList.remove(maxIndex);
    }
  }
```

3.2.2 Metodo "Clustering2"

Questo secondo metodo, invece, si occupa della creazione degli insiemi veri e propri, attribuendo ogni luogo all'insieme più consono. L'assegnazione avviene attraverso i luoghi "capogruppo" prima selezionati, in pratica per ogni luogo da visitare si rintraccia il clusterhead più vicino ad esso e gli si attribuisce il luogo corrente. In questo modo, al termine dell'analisi di ogni luogo da visitare si avranno gli insiemi completati.
Tenicamente, questo metodo si occupa della generazione di una HashMap contenente tutti i clusters creati ricevendo come input la lista dei clusterhead precedentemente generata.
Una HashMap è una implementazione contenuta nelle API Java dell'interfaccia java.util.Map. La mappa (Dictionary nel mondo .Net) è una collezione di oggetti che ha come obiettivo principale quello di rendere le operazioni di inserimento e ricerca veloci ed efficienti. In una mappa vengono memorizzate coppie (chiave, valore) attraverso due possibili implementazioni: HashMap e TreeMap.
In questo progetto la HashMap consiste in una collezione di coppie (id-clusterhead, cluster) definita in questo modo:

```
HashMap<String, ArrayList<PlaceToVisit>> dictClusters.
```

Per prima cosa viene creata nella HashMap una coppia per ogni clusterhead, inserendo come chiave il suo id e come valore una ArrayList di oggetti PlaceToVisit contenente temporaneamente un solo oggetto, ovvero tale clusterhead.
In seguito, iterando su ogni elemento della lista di PlaceToVisit selezionati

inizialmente, si attribuisce ognuno di essi al cluster che gli è più vicino calcolando la distanza tra esso ed ogni clusterhead.
Al termine di tale loop si avrà come risultato la HashMap dictClusters contenente tutti i cluster, raggiungibili attraverso la chiave id-clusterhead.

Listing 3.5: Procedura di selezione del cluster di appartenenza di ogni PlaceToVisit ed attribuzione ad esso.

```
for (int z = 0; z <= listaSenzaPoli.size() - 1; z++) {
  currentPlace = listaSenzaPoli.get(z);
  currentLocation = new Location("Current Point");
  currentLatLng = getLocationFromAddress(context,
      currentPlace.address.toString());
  currentLocation.setLatitude(currentLatLng.latitude);
  currentLocation.setLongitude(currentLatLng.longitude);
  for (int i = 0; i <= listaPoli.size() - 1; i++) {
    currentPoloPlace = listaPoli.get(i);
    currentPoloLocation = new Location("Current Polo");
    currentPoloLatLng = getLocationFromAddress(context,
        currentPoloPlace.address.toString());
    currentPoloLocation.setLatitude(currentPoloLatLng.latitude);
    currentPoloLocation.setLongitude(currentPoloLatLng.longitude);
    currentDistance =
        currentLocation.distanceTo(currentPoloLocation);
    if (i == 0) {
      minDistance = currentDistance;
      minId = currentPoloPlace.id;
      minPoloPlace = currentPoloPlace;
    } else {
      int resultCompare = Float.compare(currentDistance,
          minDistance);
      if (resultCompare < 0) {
        minDistance = currentDistance;
        minId = currentPoloPlace.id;
        minPoloPlace = currentPoloPlace;
```

```
      }
    }
  }
  if (dictClusters.get(minId) == null) {
    dictClusters.put(minId, new ArrayList<PlaceToVisit>());
  }
  dictClusters.get(minId).add(currentPlace);
}
```

3.3 Calcolo degli itinerari ed eventuale modifica dei cluster

In questa fase vengono effettuate le richieste HTTP Request attraverso la Google Maps Directions API, un servizio che calcola le indicazioni stradali. Nello specifico le HTTP Request utilizzate servono a risolvere il Traveling Salesman Problem trovando il percorso migliore per raggiungere tutti i luoghi da visitare giornalmente. Attraverso un ciclo for, quindi, si effettuano una quantità di HTTP Request pari al numero di giorni di viaggio, facendo uso della classe DirectionFinder.java.

3.3.1 DirectionFinder.java

Il costruttore di tale classe ha come argomenti il nome del luogo di pernottamento, che sarà origine e destinazione del percorso giornaliero, i "waypoints", ovvero la serie di luoghi da visitare, la modalità di spostamento e l'id del clusterhead.
Contemporaneamente all'istanziazione viene chiamato il metodo "execute()" che prima si occupa della creazione dell'URL necessario per effettuare la HTTP Request con la Google Maps Directions API attraverso il metodo crea-

teURL che codifica i dati necessari e restituisce un elemento di tipo String per poi creare un'istanza della classe privata DownloadRawData.

Listing 3.6: Metodo createUrl() all'interno della classe DirectionFinder

```
private String createUrl() throws UnsupportedEncodingException {
  String urlOrigin = URLEncoder.encode(origin, "utf-8");
  String urlDestination = URLEncoder.encode(origin, "utf-8");
  String urlWaypoints = URLEncoder.encode(waypoints, "utf-8");
  String urlCar="";

  if (travelMode=="Bicycle"){
    urlCar="&mode=bicycling";
  }
  else if (travelMode=="Walk"){
    urlCar="&mode=walking";
  }

  return DIRECTION_URL_API + "origin=place_id:" + urlOrigin +
      "&destination=place_id:" + urlDestination +
      "&waypoints=optimize:true" + urlWaypoints + urlCar +
      "&key=" + GOOGLE_API_KEY;
}
```

3.3.2 DownloadRawData

Tale classe estende la classe astratta AsyncTask, pensata per facilitare l'uso degi thread in Android. Permette di effettuare operazioni in background e pubblicare risultati senza manipolare i thread e gli handler.
In DownloadRawData ci sono due metodi principali che sovrascrivono quelli della superclasse: doInBackground e onPostExecute. Il primo è quello che viene eseguito su un thread secondario e al quale vanno attribuite tutte le operazioni più lente, mentre il secondo racchiude le azioni da svolgere al termine di doInBackground.

In pratica, nel metodo doInBackground viene effettuata l'HTTP Request, il cui risultato viene poi restituito in formato JSON ed analizzato all'interno del metodo onPostExecute attraverso l'ulteriore metodo parseJSon. Esso estrapola i dati dal documento JSON e crea una lista di istanze della classe Route con tutte le informazioni necessarie alla lettura dei percorsi trovati. Tale lista viene poi restituita all'activity principale.
A mano a mano che le liste vengono restituite, esse vengono inserite nella HashMap dictRoutes che contiene coppie con chiave clusterhead id e valore, appunto, lista di oggetti Route.
Quando il riempimento della HashMap termina, viene chiamato il metodo "FinalMethod()".

Listing 3.7: Classe DownloadRawData

```
private class DownloadRawData extends AsyncTask<String, Void,
    String> {

  @Override
  protected String doInBackground(String... params) {
    String link = params[0];
    try {
      URL url = new URL(link);
      InputStream is = url.openConnection().getInputStream();
      StringBuffer buffer = new StringBuffer();
      BufferedReader reader = new BufferedReader(new
          InputStreamReader(is));
      String line;
      while ((line = reader.readLine()) != null) {
        buffer.append(line + "\n");
      }
      return buffer.toString();
    } catch (MalformedURLException e) {
      e.printStackTrace();
    } catch (IOException e) {
```

```
            e.printStackTrace();
          }
          return null;
        }
        @Override
        protected void onPostExecute(String res) {
          try {
            parseJSon(res);
          } catch (JSONException e) {
            e.printStackTrace();
          }
        }
    }
```

3.3.3 Modifica cluster non consoni

Il metodo "FinalMethod()" amministra il controllo, la modifica ed il salvataggio nel database dei cluster attraverso altri metodi, in questa sottosezione ci occuperemo del controllo e della modifica di quelli non consoni.

Per prima cosa viene lanciato il metodo "controlAndModify()" con le due HashMap appena create come parametri, esso si assicura del tipo di preferenze scelto dall'utente in precedenza ed in base a questo effettua il controllo adeguato sui cluster.

Non considerazione del tempo di spostamento

In questo caso viene lanciato il metodo "controlDurationNotConsidering-TravelTime()" che attraverso un ciclo for su tutti i cluster, controlla che la durata delle visite programmate per ogni giornata non sia maggiore del tempo a disposizione quotidiano. Nel caso in cui si ecceda tale tempo, la funzione restituisce in output il clusterhead-id del cluster non accettabile. Tale output viene raccolto dal metodo "FinalMethod()" che lo da in input

all'ulteriore metodo "modifyClusters()". Quest'ultimo si occupa dell'eliminazione del PlaceToVisit più lontano dal clusterhead e della sua aggiunta al cluster più vicino.
Questo procedimento è svolto con l'aiuto di una nuova HashMap che per ogni PlaceToVisit racchiude tutti i cluster da cui è già stato eliminato, così da evitare che venga aggiunto ad un cluster che non può ospitarlo. Nel caso in cui il numero di cluster da cui è stato eliminato un luogo è uguale al numero di cluster totali, tale luogo viene eliminato definitivamente.
Al termine di ogni spostamento andato a buon fine vengono effettuate due nuove HTTP Request per rigenerare i nuovi percorsi dei due cluster appena modificati, quello da cui è stato eliminato un luogo e quello a cui è stato aggiunto. In seguito alle HTTP Request il metodo "FinalMethod" riavvierà i processi di controllo e modifica.

Considerazione del tempo di spostamento

In questo caso attraverso il metodo "controlDuration()" viene controllato se il tempo di visita totale dei luoghi da visitare in ogni giornata addizionato al tempo di spostamento necessario sono maggiori del tempo a disposizione giornaliero.
Nel caso in cui tale somma superi la quantità di ore a disposizione e l'algoritmo scelto sia quello che considera il ranking dei luoghi, va in esecuzione il metodo "deleteRanking()", il quale si occupa di trovare il PlaceToVisit con l'attributo ranking più alto per eliminarlo e riavviare il processo di creazione di cluster e itinerari dall'inizio richiamando in sequenza i metodi "clustering1()", "clustering2" e le HTTP Request.
Se invece l'algoritmo scelto dall'utente è quello "Optimized", il procedimento di modifica dei cluster è lo stesso del caso di non considerazione del tempo di spostamento e utilizza il metodo "modifyClusters()" per spostare ed eventualmente eliminare i PlaceToVisit in eccesso.

3.4 Salvataggio dati nel Database

Una volta terminato il processo di controllo e modifica dei cluster, è il momento del salvataggio dei dati utili nel database, così da renderli fruibili ed utilizzabili in qualunque momento per la presentazione all'utente degli itinerari originati.
Per la memorizzazione nel database si fa uso di una classe DBhelper che estende la classe SQLiteOpenHelper, che in Android consente di gestire la creazione di database, la gestione delle connessioni ed altri aspetti come il versioning dello stesso. In tale classe viene effettuato l'override dei metodi onCreate e onUpgrade al fine di poter creare ed aggiornare dinamicamente le tabelle del nostro DB. Vengono anche definiti metodi per l'inserimento e l'estrapolazione di dati dalle tabelle del database, poi utilizzati quando vi è la necessità di mostrare tali informazioni all'utente in una schermata.
Specificatamente vengono salvati nel database i dati riguardanti il viaggio in generale, i giorni di viaggio ed i luoghi da visitare.
Con il salvataggio si conclude l'algoritmo e si passa all'esposizione dei risultati ottenuti.

3.5 Esposizione risultati

I risultati ottenuti vengono visualizzati attraverso RecyclerView in due diverse activity.

3.5.1 Visualizzazione giorni

In questa activity è presente una RecyclerView con i giorni di viaggio, premendo su uno di essi, viene aperta l'activity contenente l'itinerario giornaliero.
La RecyclerView viene popolata passando al suo adapter una lista di oggetti DayView con attributi number, date e idClusterhead. Questi oggetti vengono

inizializzati estrapolando le informazioni necessarie dal database attraverso delle operazioni Get appositamente definite.

3.5.2 Visualizzazione itinerario giornaliero

Attraverso una RecyclerView viene messa in mostra la lista dei luoghi di interesse che fanno parte del programma giornaliero, nell'ordine di visita consigliato. Ci sono tre features aggiuntive in questa activity: la prima è il bottone "Show on Maps" che crea un intent aprendo una nuova Maps Activity e posizionando un Marker nella posizione dei Place facenti parte dell'itinerario del giorno; la seconda è un bottone di cui dispone ogni Place appartenente alla RecyclerView, che attraverso la libreria Maps URLs apre la app Google Maps offrendo indicazioni stradali dalla posizione attuale a tale Place; la terza è anch'essa un bottone attribuito ad ogni Place della RecyclerView munito di sito web ufficiale, premendo su di esso viene aperto nel browser tale sito. Gli ultimi due bottone descritti sono pensati per offrire un servizio all'utente nel momento in cui è in viaggio, dando la possibilità di avere indicazioni stradali e informazioni sul prossimo luogo da visitare in maniera veloce ed intuitiva.

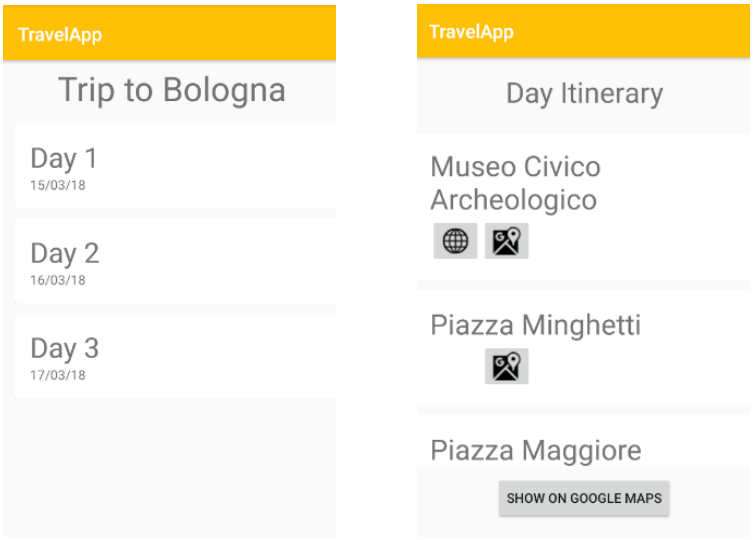

Figura 3.9: Screenshot della visualizzazione dei giorni di viaggio e dell'itinerario giornaliero

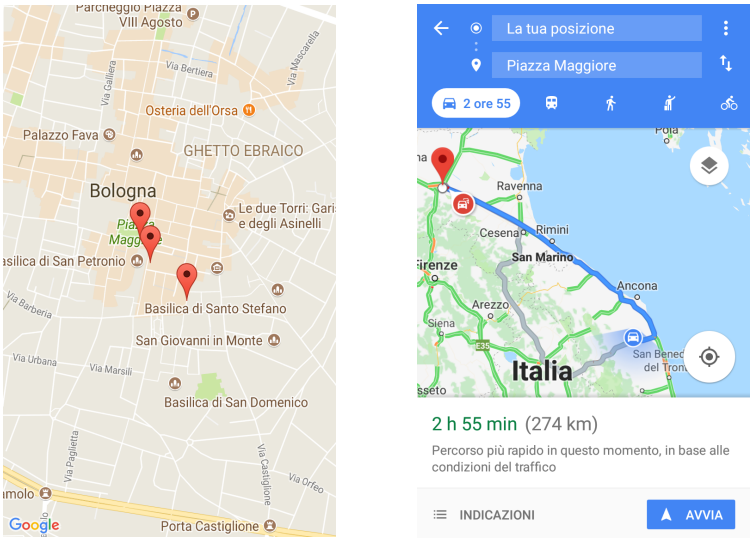

Figura 3.10: Screenshot della Maps Activity e delle indicazioni stradali attraverso la app di Google Maps

Capitolo 4

Esempi di esecuzione

In questo capitolo verranno eseguiti diversi esempi inserendo dati differenti ed analizzando i risultati ottenuti con le diverse tipologie di algoritmo descritte in precedenza.

4.1 Caso Londra: ripartizione visita luogo in più giorni

In questo esempio viene inserito un luogo da visitare con tempo di visita maggiore al tempo giornaliero a disposizione, si assume che l'utente abbia la necessità di visitare tale luogo in più giorni a causa della lunga durata della visita o del poco tempo a disposizione ogni giorno.
La città visitata è Londra, il viaggio ha la durata di 3 giorni con 2 ore a disposizione al giorno. Vengono inseriti 3 luoghi diversi da visitare: Piccadilly Circus con un tempo di visita di 60 minuti, il London Bridge con un tempo di visita di 60 minuti ed infine il London Eye con un tempo di visita di 140 minuti. Quest'ultimo è il fulcro dell'esempio, in quanto essendo 120 minuti il limite massimo giornaliero di tempo a disposizione, tale luogo verrà suddiviso in due diversi oggetti PlaceToVisit, uno da 120 minuti ed uno da 20 minuti, che andranno a far parte di due itinerari giornalieri diversi.
L'algoritmo per prima cosa crea tre cluster, con i tre luoghi inseriti come

clusterhead, ovviamente i due oggetti uguali del London Eye faranno parte del medesimo cluster in quanto la loro distanza l'uno dall'altro è nulla.
Riassumendo, la situazione in seguito all'esecuzione dei metodi Clustering1 e Clustering2 sarà quella con i seguenti cluster:

1. London Eye(120minuti), London Eye(20 minuti)
2. Piccadilly Circus(60 minuti)
3. London Bridge(60 minuti)

In seguito verrà applicato il controllo delle durate che porterà alla modifica dei cluster in quanto il primo cluster ha durata maggiore del massimo, quindi il secondo oggetto London Eye con tempo di visita 20 minuti verrà spostato nel cluster più vicino per poi effettuare di nuovo il controllo.
Il cluster di destinazione dell'oggetto da spostare è il numero 2 in quanto Piccadilly Circus è geograficamente più vicino al London Eye rispetto al London Bridge, quindi la situazione finale che verrà riportata all'utente sarà:

1. London Eye(120minuti)
2. Piccadilly Circus(60 minuti), London Eye(20 minuti)
3. London Bridge(60 minuti)

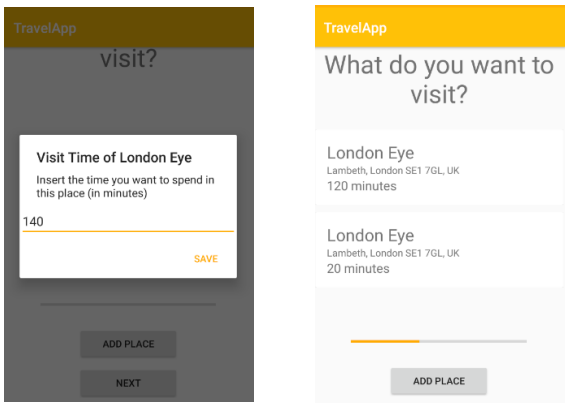

Figura 4.1: Caso Londra: inserimento dati

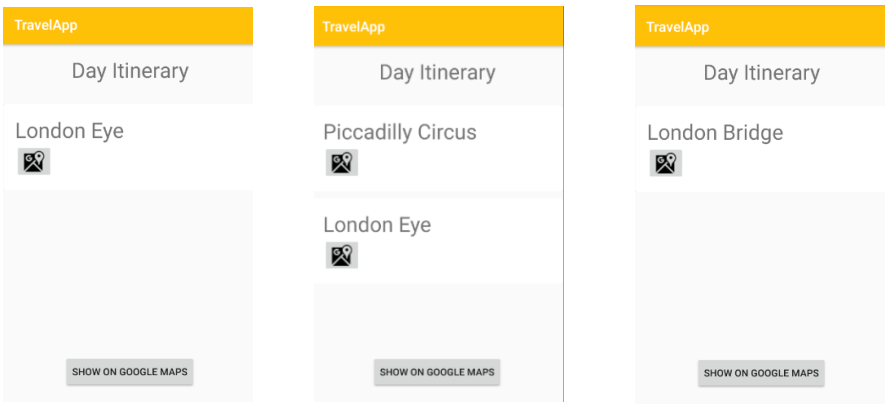

Figura 4.2: Caso Londra: visualizzazione risultati

4.2 Caso Praga: pochi luoghi da visitare

In questo esempio vengono inseriti un minor numero di luoghi da visitare rispetto alla durata del viaggio in giorni, il che provocherebbe la creazione di un itinerario di viaggio con dei giorni vuoti. Il programma, invece, all'inizio dell'algoritmo, esegue un controllo sulla durata del viaggio rispetto al numero di luoghi da visitare e nel caso in cui si sviluppi la situazione precedentemente descritta crea velocemente l'itinerario completo saltando tutta la parte di creazione di cluster e modifica di essi. Questo è reso possibile semplicemente generando un itinerario con un numero di giorni pari alla dimensione della lista di luoghi da visitare inseriti, dove ogni giorno conterrà un singolo luogo da visitare.

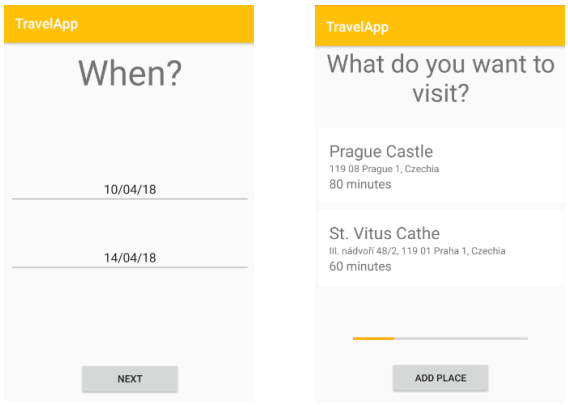

Figura 4.3: Caso Praga: inserimento dati

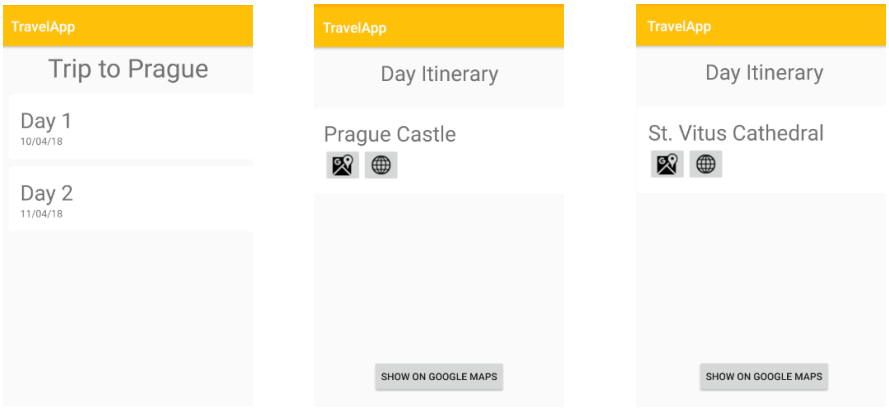

Figura 4.4: Caso Praga: visualizzazione risultati

4.3 Caso Roma: esecuzione dei 3 diversi algoritmi con gli stessi dati di input

In questo esempio vengono inseriti gli stessi dati sul viaggio che si vuole programmare, ma cambiando ogni volta il tipo di algoritmo da utilizzare, così da poter analizzare le differenze nei risultati ottenuti.
I dati di input inseriti sono volti a creare un itinerario di viaggio con destinazione Roma, di 3 giorni e con a disposizione 2 ore al giorno per le visite. Il luogo di pernottamento è impostato a Via Cavour ed i luoghi da visitare sono: Colosseo (80 minuti), Fontana di Trevi(40 minuti), Basilica di San

Pietro(60 minuti), Foro Romano(60 minuti), Circo Massimo(40 minuti).
I cluster che vengono creati dall'applicazione dei metodi Clustering1 e Clustering2 sono:

1. Fontana di Trevi, Foro Romano
2. Colosseo, Basilica di San Pietro
3. Circo Massimo

Adesso analizzeremo i diversi risultati ottenuti dall'applicazione degli algoritmi di modifica dei cluster.

4.3.1 Non considerazione del tempo di spostamento

Questo algoritmo non considera il tempo di spostamento necessario da un luogo d'interesse ad un altro, ma solo il tempo effettivo di visita inserito in input dall'utente.
In questo caso in seguito ad un semplice controllo sulla durata complessiva delle visite di ogni cluster si scopre che nel secondo essa è di 140 minuti, quindi maggiore del limite giornaliero di 120 minuti.
A questo punto si applica la modifica di tale cluster spostando il luogo più lontano al clusterhead in un altro cluster, tale luogo è la Basilica di San Pietro che viene quindi spostata nel cluster ad essa più vicino non considerando quello attuale. Dopo tale modifica la situazione è la seguente:

1. Fontana di Trevi, Foro Romano, Basilica di San Pietro
2. Colosseo
3. Circo Massimo

Ovviamente anche in questa situazione il controllo porterà alla luce un'eccedenza nel tempo di visita del primo cluster, causando uno spostamento della Basilica di San Pietro al terzo cluster, per arrivare alla situazione finale, accettata dal controllo, che avrà come cluster definitivi i seguenti:

1. Fontana di Trevi, Foro Romano
2. Colosseo
3. Circo Massimo, Basilica di San Pietro

Il tempo di visita del primo cluster è di 100 minuti, quello del secondo è di 80 minuti e quello del terzo è di 120 minuti.

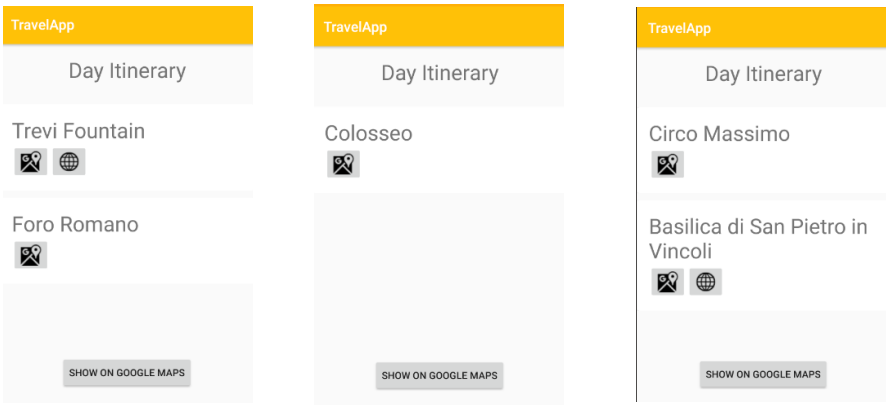

Figura 4.5: Caso Roma, algoritmo "Non considerazione del tempo di spostamento": visualizzazione risultati

4.3.2 Ranking

Come spiegato in precedenza, in questo algoritmo si ordinano per importanza i luoghi in base all'ordine di inserimento e nel momento in cui si ha un eccedenza di tempo in qualunque cluster, viene eliminato il luogo con ranking minore e si effettua nuovamente la creazione dei cluster. Importante ricordare che in questo caso nel controllo del tempo di visita di ogni cluster viene considerato anche il tempo di spostamento da un luogo ad un altro.
Il ranking dei luoghi in questo esempio è descritto dal seguente elenco, in ordine di importanza decrescente:

1. Colosseo
2. Fontana di Trevi

3. Basilica di San Pietro

4. Foro Romano

5. Circo Massimo

Naturalmente, anche in questo caso il secondo cluster ha una durata maggiore del limite e quindi si effettua subito l'eliminazione del luogo con ranking minore, cioè il Circo Massimo, per poi effettuare nuovamente il procedimento di clustering ed arrivare alla seguente situazione:

1. Fontana di Trevi

2. Colosseo, Basilica di San Pietro

3. Foro Romano

Nonostante l'eliminazione, il secondo cluster è rimasto immutato, quindi è necessaria l'ulteriore eliminazione di un'altro luogo e sempre considerando il ranking tale luogo è il Foro Romano.

1. Fontana di Trevi

2. Colosseo

3. Basilica di San Pietro

Questa la situazione finale, con un risultato peggiore rispetto al primo algoritmo in quanto è stato necessario eliminare ben due luoghi dalla lista inserita dall'utente.

4.3.3 Optimized

Questo algoritmo effettua un controllo considerando il tempo di spostamento ma, a differenza del precedente, in caso di eccedenza va a modificare solamente il cluster interessato, spostando uno dei suoi luoghi in un altro cluster. Solo nel caso in cui il luogo spostato non possa essere attribuito a

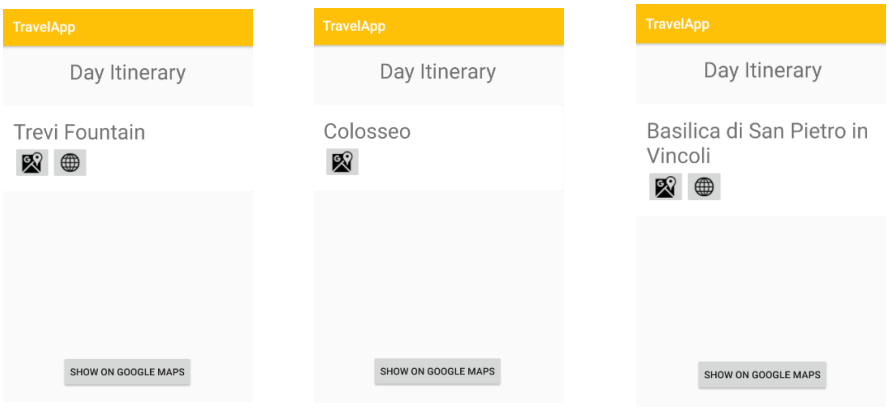

Figura 4.6: Caso Roma, algoritmo "Ranking": visualizzazione risultati

nessun cluster in quanto provoca un'eccedenza in ognuno, esso viene eliminato definitivamente.
Come negli altri casi, il secondo cluster non viene accettato dal metodo di controllo della durata delle visite e del tempo di spostamento quindi si attiva il processo di modifica di esso. Il luogo che viene spostato è la Basilica di San Pietro e il cluster di destinazione è il primo in quanto più vicino ad essa. La situazione in seguito a questa prima modifica è la seguente:

1. Fontana di Trevi, Foro Romano, Basilica di San Pietro
2. Colosseo
3. Circo Massimo

Il metodo di controllo viene effettuato nuovamente ed evidenzia un'eccedenza nel primo cluster che porta quindi la Basilica di San Pietro ad essere spostata nel terzo cluster.

1. Fontana di Trevi, Foro Romano
2. Colosseo
3. Circo Massimo, Basilica di San Pietro

Anche in questo caso, però, il terzo cluster non viene accettato dal metodo di controllo e quindi porta la Basilica di San Pietro alla cancellazione, in quanto è già stata rifiutata da tutti i cluster.
Successivamente alla cancellazione, la situazione dei cluster è la seguente e viene accettata come definitiva:

1. Fontana di Trevi, Foro Romano
2. Colosseo
3. Circo Massimo

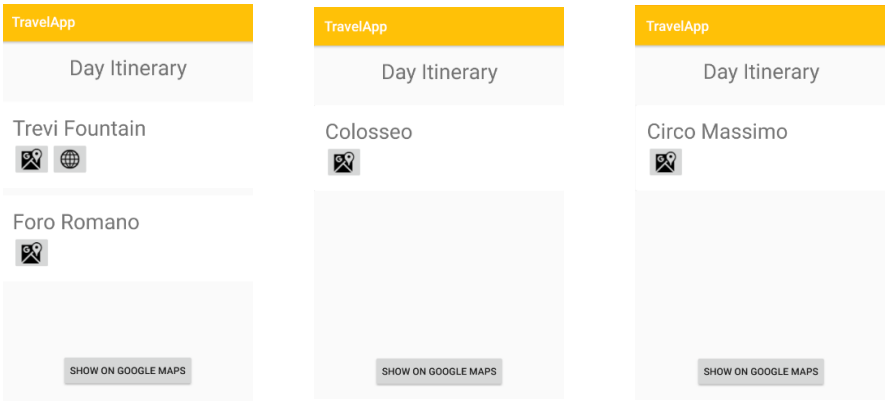

Figura 4.7: Caso Roma, algoritmo "Optimized": visualizzazione risultati

4.4 Caso Parigi: esempio con visualizzazione mappe e indicazioni stradali

In questo esempio verrà creato un nuovo viaggio inserendo più di 10 luoghi da visitare, per dimostrare che l'applicazione non funziona solo con un numero limitato di luoghi.
La città di destinazione è Parigi, il viaggio ha la durata di 5 giorni, con 6 ore al giorno di disponibilità alle visite.
I luoghi da visitare sono 12: Torre Eiffel(120 minuti), Museo del Louvre(180

minuti), Notre-Dame(60 minuti), Arco di Trionfo(45 minuti), Sacro Cuore(60 minuti), Museo d'Orsay(120 minuti), Place de la Concorde(30 minuti), Centro Pompidou(120 minuti), Champs-Élysées(60 minuti), Montmartre(120 minuti), Grand Palais(45 minuti), Quartiere Latino (120 minuti).
Viene scelto l'algoritmo di non considerazione del tempo di trasferimento da un luogo ad un altro e la macchina come metodo di spostamento.
I risultati ottenuti sono descritti nell'elenco numerato seguente che mostra la composizione degli itinerari giornalieri.

1. Sacro cuore, Montmartre.
2. Torre Eiffel, Champs-Élysées, Grand Palais, Museo d'Orsay.
3. Centro Pompidou
4. Quartiere latino, Notre-dame, Museo del Louvre
5. Arco di Trionfo, Place de la Concorde

Di seguito verranno mostrate le mappe che si aprono premendo il bottone "Show on Google Maps" all'interno della schermata di presentazione di ogni itinerario giornaliero. Tali mappe hanno la funzione di mostrare dove sono situati i luoghi proposti nell'itinerario. Ogni luogo è contrassegnato da un marker rosso e la mappa è impostata per mostrarli tutti contemporaneamente in una activity apposita, senza la necessità di uscire dall'applicazione ed aprire la app di Google Maps.

Successivamente, invece, l'intenzione è quella di mostrare il funzionamento della app nel momento effettivo delle visite. Infatti il pulsante posizionato sotto ogni luogo da visitare all'interno della schermata degli itinerari giornalieri funge da aiuto nello spostamento pratico da un posto all'altro.
Premendo su di esso viene aperta la app di Google Maps o il browser e vengono mostrate le indicazioni stradali dalla posizione attuale del dispositivo al

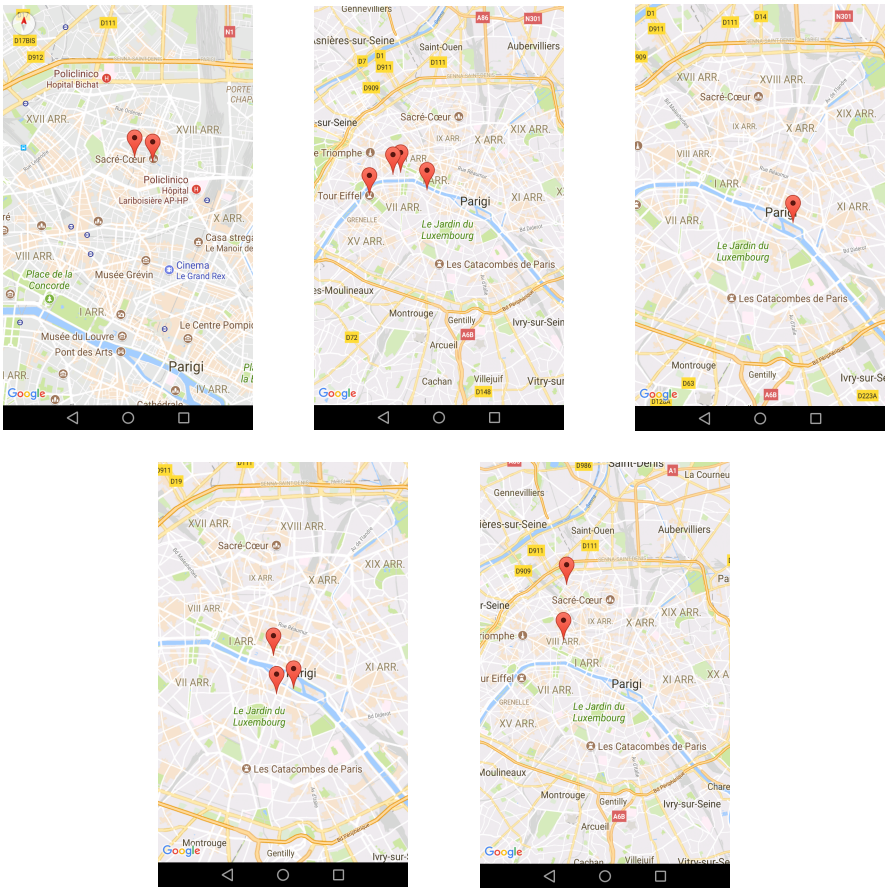

Figura 4.9: Caso Parigi: visualizzazione itinerari giornalieri su mappa

luogo da visitare in questione. Questo permette di avere indicazioni stradali passo a passo, terminata la visita di un luogo, basta premere il bottone del luogo successivo e si avrà il percorso migliore per raggiungerlo.
Immaginando di partire dal luogo di pernottamento, cioè Avenue Victoria, e di voler seguire l'itinerario del secondo giorno, si mostrano gli screenshot delle indicazioni stradali.

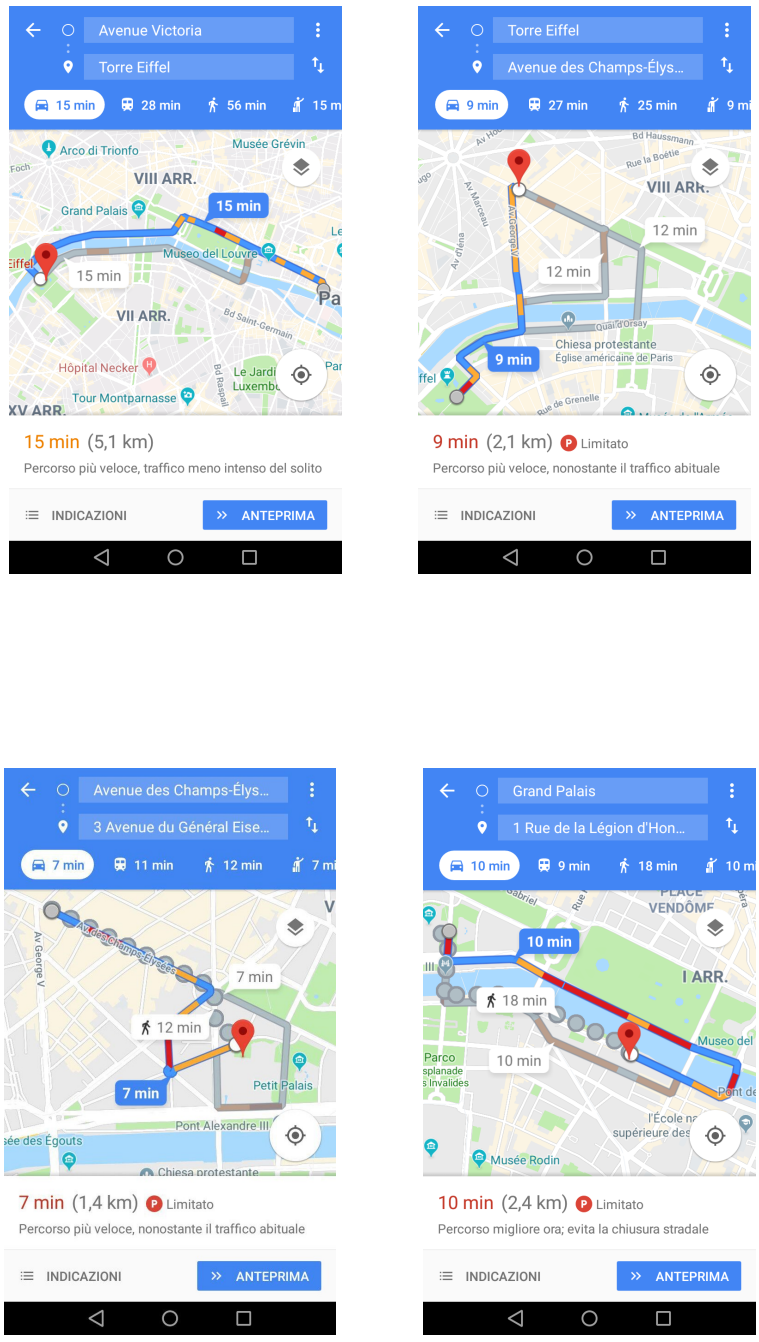

Figura 4.11: Caso Parigi: visualizzazione indicazioni stradali

Conclusioni

È stato provato come l'utilizzo degli smartphone e delle applicazioni mobili sia in costante ascesa, gli utenti sono sempre più abituati a compiere attraverso queste nuove tecnologie le azioni che prima erano abituati a realizzare manualmente. Questo mercato è uno dei più proficui ed è ormai possibile eseguire un'infinità di operazioni attraverso tali device.
I software disponibili nei diversi market permettono di risparmiare in termini sia monetari che di tempo e facendo uso di algoritmi e metodi automatizzati spesso sono anche più sicuri ed efficienti.
Nel campo della programmazione e gestione dei viaggi sono disponibili molteplici app mobili ma queste, spesso, non si servono delle capacità di computazione ed analisi disponibili, limitandosi a fornire servizi di somministrazione di informazioni e salvataggio di mete ed itinerari personali.
Riassumendo, il progetto si diversifica dal resto dei software presenti sul mercato grazie alle proprietà di analisi e computazione dei dati che portano poi alla generazione di un itinerario personalizzato ed ottimizzato, a differenza di quelli "preconfezionati" disponibili online.
L'obiettivo del presente progetto è stato quello di sviluppare un software che fornisca un servizio utile ed efficiente, da poter poi migliorare e sviluppare ulteriormente per arrivare ad un prodotto completo, che oltre alla creazione del progetto offra anche servizi aggiuntivi ormai indispensabili in un'applicazione mobile che si rispetti.

Sviluppi futuri

In futuro verrà seguito un processo di sviluppo e miglioramento del progetto di cui le migliorie finora pensate sono:

- Miglioramento di UI e UX per perfezionare l'applicazione in quanto a design e esperienza di utilizzo.
- Permettere all'utente di selezionare luogo, orario e tempistica dei pasti, per includerli nell'itinerario.
- Permettere all'utente di scegliere tra diversi itinerari offerti invece di presentare quello definitivo, cosicché possa esprimere le sue preferenze in quanto all'ordine di visita del luoghi di interesse.
- Aggiungere un meccanismo di login e di salvataggio dati su Cloud per permettere all'utente di accedere ai suoi dati da qualunque dispositivo.
- Dare la possibilità di scaricare gli itinerari sotto forma di PDF o di altro tipo di file, così da poterli condividere tramite email e applicazioni di messaggistica.
- Migliorare ulteriormente l'algoritmo di clustering e quello di gestione delle richieste HTTP per fornire un servizio più veloce ed efficiente possibile.

Bibliografia

[1] Google Maps Android API
https://developers.google.com/maps/documentation/android-api/

[2] Google Maps Directions API
https://developers.google.com/maps/documentation/directions/

[3] Google Places API for Android
https://developers.google.com/places/android-api/

[4] Maps URLs
https://developers.google.com/maps/documentation/urls/guide

[5] Struttura applicazione Android https://developer.android.com/design/patterns/app-structure.html

[6] Android Studio https://developer.android.com/studio/intro/index.html

[7] Java https://www.java.com/en/download/faq/whatis_java.xml

[8] XML https://techterms.com/definition/xml

[9] JSON https://www.json.org/

[10] Android Manifest https://developer.android.com/guide/topics/manifest/manifest-intro.html

[11] Place Autocomplete https://developers.google.com/places/android-api/autocomplete

[12] RecyclerView https://developer.android.com/guide/topics/ui/layout/recyclerview.html

[13] FloatingActionButton https://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html

[14] DatePicker https://developer.android.com/reference/android/widget/DatePicker.html

[15] HashMap https://developer.android.com/reference/java/util/HashMap.html

[16] ArrayList https://developer.android.com/reference/java/util/ArrayList.html

[17] AsyncTask https://developer.android.com/reference/android/os/AsyncTask.html

[18] SQLite https://developer.android.com/training/data-storage/sqlite.html

[19] Problema del commesso viaggiatore(TSP) con Google Directions API https://developers.google.com/optimization/routing/tsp/tsp

[20] statcounter GlobalStats http://gs.statcounter.com/os-market-share/mobile/worldwide

[21] Symbian OS http://www.storiainformatica.it/symbian-os

[22] Statista https://www.statista.com

Ringraziamenti

Ringrazio il relatore per la sua grande disponibilità.
Ringrazio la mia famiglia per l'appoggio incondizionato.
Ringrazio i miei amici per supportarmi e sopportarmi sempre.