Giacomo Cacco 593022

iTurista: localizzazione di punti di interesse su piattaforma Android

Sommario

Questa relazione di tirocinio illustra come è stata realizzata l'applicazione per dispositivi Android chiamata “iTurista”, utilizzata per il recupero di informazioni riguardanti i luoghi di interesse della regione Campania.

Viene descritta la struttura dell'azienda all'interno della quale si è lavorato, il lavoro svolto durante il periodo di stage, i requisiti che l'applicazione doveva soddisfare, le scelte tecniche adoperate per rispettarli, includendo anche la descrizione dettagliata del codice scritto durante lo sviluppo dell'applicazione, le problematiche riscontrate durante la realizzazione di essa e la strada scelta per la loro risoluzione.

Infine, viene mostrato il funzionamento dell'applicazione tramite l'utilizzo di immagini che la raffigurano durante il suo utilizzo.

Indice

Capitolo 1: Descrizione dell'azienda e finalità del lavoro di stage

1.1 L'azienda

L'azienda presso la quale si è svolto il tirocinio è Egea Tecnologie Informatiche s.r.l., situata a Quarto d'Altino. Si tratta di un'impresa inserita nel campo tecnologico da ormai 10 anni con occupazione principale quella di creazione e sviluppo di applicazioni web come siti internet, portali, programmi gestionali, ecc. anche se, negli ultimi anni, si è passati alla progettazione di applicazioni per dispositivi mobili come smartphone, tablet e blackberry. L'azienda è racchiusa all'interno di un locale di circa 100 mq nel quale sono presenti 6 persone; tra queste vi sono un presidente ed un amministratore delegato che svolgono il compito di amministrare l'azienda e di tenere i rapporti commerciali con i vari clienti; mentre le restanti 4 persone sono gli sviluppatori, cioè coloro che creano i programmi e le applicazioni richieste dai clienti. Ognuno di questi sviluppatori ha competenze specifiche di diverse tecnologie, in modo tale che, se necessario, si riesce a lavorare su di un progetto anche in mancanza della persona con più confidenza sulla tecnologia da utilizzare per il progetto. Oltre a questi dipendenti l'azienda utilizza collaborazioni con persone e aziende esterne per gli aspetti riguardanti la grafica e l'hosting per poter soddisfare al meglio le richieste del cliente riguardante la realizzazione dell'applicazione voluta. Sul mercato Egea Tecnologie Informatiche si propone come azienda che offre un “global service”, cioè non solo sviluppa l'applicazione web o per dispositivo mobile, ma fornisce anche aiuto su ogni aspetto legato al corretto funzionamento del prodotto realizzato, come supporto per il software, per l'hardware e la risoluzione dei problemi legati a questi due aspetti. L'obiettivo principale dell'azienda è quello di cercare e scegliere sempre nuove tecnologie da proporre ai partner ed ai clienti per innovare i prodotti aziendali e proporne sempre di innovativi. Grazie a ciò è stato possibile ottenere importanti progetti da grossi clienti italiani ma anche provenienti dall'estero che riguardano lo sviluppo di applicazioni web e per dispositivi mobile all'avanguardia.

1.2 Prodotti aziendali

Nel corso degli anni di attività, l'azienda Egea Tecnologie Informatiche ha creato molte applicazioni web per le più varie necessità, ma comunque raggruppate in alcune categorie. Di questi prodotti ne verranno citati alcuni fra i più importanti per ogni categoria. La prima di esse è la categoria di prodotti gestionali, cioè le applicazioni sviluppate per facilitare la gestione di risorse, persone e progetti da parte dei committenti dell'applicazione. I prodotti gestionali più importanti realizzati

sono stati quelli per:

- COMUNE DI MARCON - RESTYLING ED IMPLEMENTAZIONE WEBSITE: il progetto (Premio Poloest 2012) ha riguardato l'adattamento del sito del Comune di Marcon agli ultimi criteri di accessibilità, il restyling grafico, il caricamento dei dati esistenti e le attività di assistenza e formazione del personale all'utilizzo del CMS;
- PARI OPPORTUNITÀ - PROVINCIA DI VENEZIA: sviluppo del website delle Pari Opportunità della Provincia di Venezia in tecnologia MS asp/Sql Server;
- GESTIONE DELLE NOMINE - CONSIGLIO REGIONALE DEL VENETO: realizzazione di un applicativo web per la gestione delle procedure di nomina delle cariche istituzionali di competenza del Consiglio Regionale del Veneto;
- AVEPA - AGENZIA VENETA PER I PAGAMENTI IN AGRICOLTURA: realizzazione del Portale di A.VE.P.A. – Agenzia Veneta per i Pagamenti in Agricoltura, sviluppato in tecnologia ASP;
- BANCA DATI ELETTORALE - CONSIGLIO REGIONALE DEL VENETO: intranet sviluppata da Egea negli anni 2006-2011 in tecnologia Java-Oracle e sviluppo del backoffice per l'importazione automatica da file excel/csv strutturati.

Un'altra categoria è quella dei prodotti rivolti al mondo della finanza, dove fra i lavori più importanti abbiamo:

- INTERBANCA: sito Istituzionale di comunicazione bancaria, gestione di informazioni e news finanziarie in realtime (quotazioni di borsa e notizie finanziarie);
- RAS ASSICURAZIONI: Integrazione di un'area dinamica (Press Office) all'interno di un sito preesistente e gestione della comunicazione istituzionale (Investor Relations).

La categoria di prodotti successiva è quella che riguarda lo sviluppo di applicazioni per i clienti più famosi e quotati, e fra i prodotti più importanti realizzati vi troviamo:

- PORTALE ASSOCAMPING: realizzazione di uno strumento di C.M.S. (Content Management System) per lo sviluppo del sito istituzionale dell'Associazione Campeggiatori del Cavallino (VE);
- SEGNI DEL 900 - CONFERENZA EPISCOPALE ITALIANA: Realizzazione di uno strumento di C.M.S. (Content Management System) per la gestione della Mostra itinerante "Architettura ed Arti per la liturgia in Italia", a cura della Conferenza Episcopale Italiana (CEI). Il percorso guidato realizzato permette la navigazione virtuale della mostra;

- BOSCOLO GROUP: sviluppo del sito istituzionale del Gruppo Boscolo, società leader nel campo della ricettività e dei servizi turistici.

Infine vi sono i tutti gli altri prodotti realizzati dall'azienda che non rientrano in nessuna delle categorie precedenti, in quanto sono prodotti realizzati sulla tipologia di portali web, utilizzati soprattutto nel fornire informazioni su specifici argomenti agli utenti che li consultano. Fra i più importanti citiamo:

- CASSA PADANA BCC /ISTITUTO ALCIDE CERVI - Portale “Memorie in cammino”: il progetto è lo sviluppo del portale www.memorieincammino.it e del corrispondente DMS. Obiettivo del progetto è la costruzione di un percorso su internet che raccoglierà le memorie (foto, video, interviste, documenti,…) di uno dei periodi più duri della storia d’Italia dal fascismo, alla seconda guerra mondiale, fino alla Resistenza e alla Liberazione.
- VEGAL: portale web in tecnologia php/PostgreSql e MS Virtual Earth per effettuare la georeferenziazione delle informazioni delle aree geografiche del Veneto Orientale visualizzate tramite mappa;
- ULSS 12 VENEZIA - Gestione delle pratiche amministrative dei medici: la procedura consente l’invio e la gestione digitale delle pratiche amministrative mensili con firma digitale dei moduli PIP (Prestazioni di particolare Impegno Professionale) – ADI (Assistenza Domiciliare Integrata) – ADP(Assistenza Domiciliare Programmata)) che i medici di medicina generale convenzionati con l’ULSS 12 di Venezia devono produrre per attestare la loro attività;
- LEGA DEL FILO D'ORO: Egea ha progettato e sviluppato il sistema di integrazione (web services) tra l'e-commerce/CMS del portale della Lega del Filo d'oro ed i sistemi informativi interni.

Il progetto sul quale si è lavorato durante il tirocinio, però, riguarda la realizzazione di un'applicazione per dispositivi mobili Android, attività della quale l'azienda si è occupata solo nell'ultimo periodo della propria esistenza. Per tale motivo non è ancora collocabile in una delle categorie di lavori sopra citate.

1.3 Il progetto

L'applicazione oggetto di questa relazione è stata pensata per i turisti in vacanza nella regione Campania che, trovandosi in un territorio di notevole estensione, volessero sapere cosa ci sia nei dintorni del luogo in cui si trovano ed anche come eventualmente raggiungere i posti interessanti segnalati. E' stato richiesto inoltre che nell'applicazione, da sviluppare esclusivamente per dispositivi mobili Android, fosse disponibile la scelta della lingua di utilizzo fra italiano e inglese per renderla utilizzabile anche da turisti stranieri.

Per soddisfare tali richieste è stata realizzata l'applicazione per dispositivi Android chiamata iTurista, la quale reperisce da internet l'ubicazione di tutti i luoghi utili e di interesse (come stazioni di polizia, vigili del fuoco, ospedali, farmacie, parchi e luoghi turistici) situati nella regione Campania. Inoltre, per i luoghi trovati, fornisce un elenco delle informazioni principali e, se disponibili, mostra anche alcune immagini del luogo. Infine, una volta selezionato un luogo di proprio interesse, l'applicazione consente di ottenere le indicazioni stradali utili per raggiungerlo partendo dalla posizione in cui si trova il dispositivo al momento della richiesta dell'operazione.

Per svolgere tutte queste funzioni l'applicazione utilizza una connessione ad internet stabilita dal dispositivo tramite l'aggancio ad una rete Wi-Fi presente nelle vicinanze oppure alla rete 3G presente ormai ovunque. Inoltre è richiesta l'abilitazione del dispositivo alla possibilità di essere rintracciato nello spazio tramite GPS oppure grazie al metodo alternativo chiamato cellID.

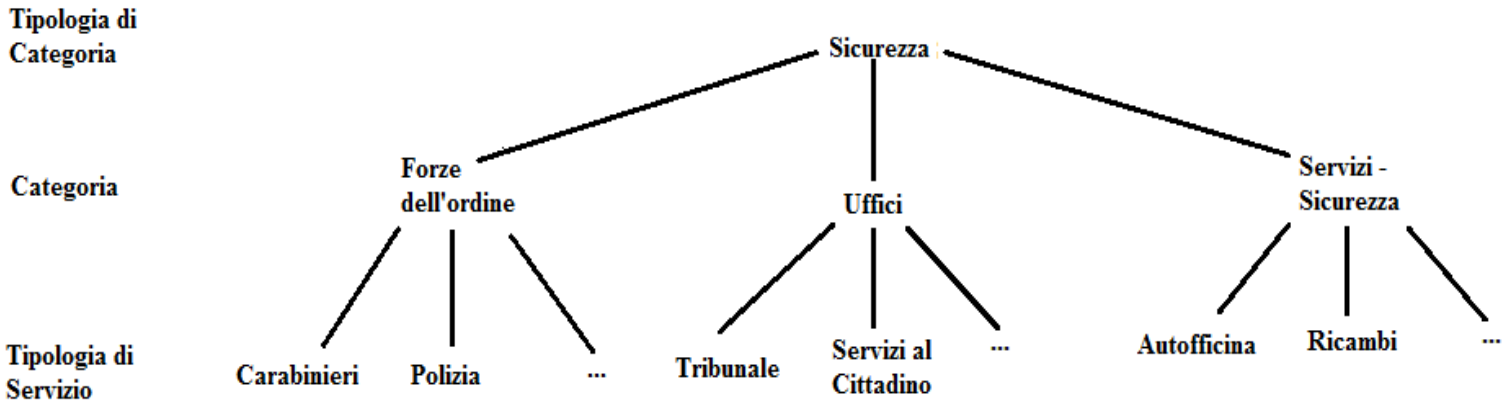
CellID è un sistema di ricerca della posizione del dispositivo mobile tramite l'utilizzo degli identificativi delle antenne che appartengono agli operatori mobili. La posizione è rintracciabile grazie all'appoggio su alcuni database, i quali sono stati popolati di dati in tutto il mondo, con un'accuratezza elevata.

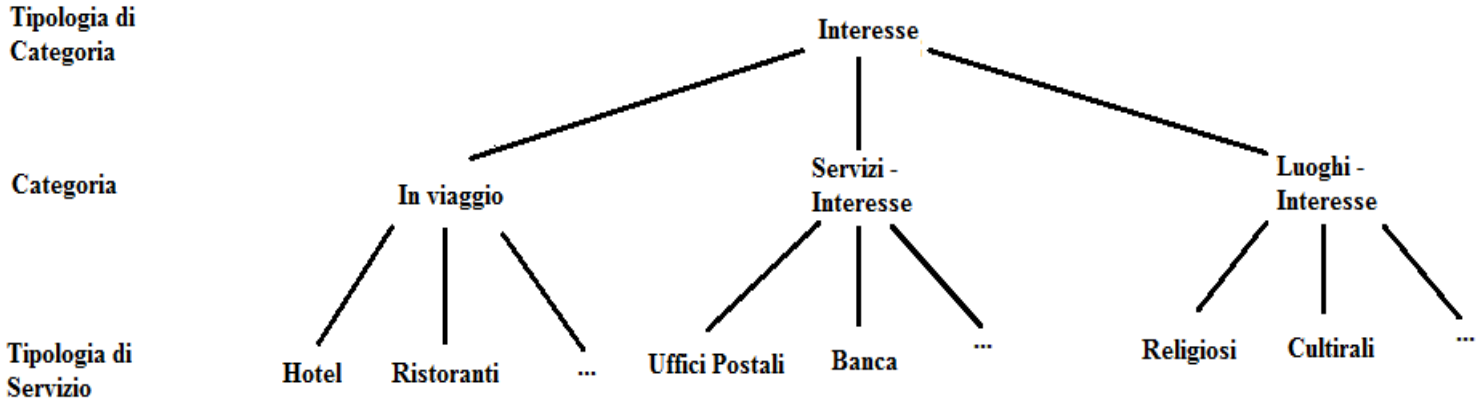
L'applicazione si basa sulle mappe di Google, caricando sullo schermo la parte di mappa riguardante l'Italia per poi posizionarsi automaticamente sulla posizione attuale del dispositivo una volta rilevata. Successivamente si può passare all'utilizzo della funzione di ricerca dei luoghi. La ricerca permette di specificare alcuni parametri per facilitare il reperimento di luoghi filtrando i risultati ottenuti da una eventuale ricerca di tutti i luoghi presenti nel database del server che fornisce tali informazioni. I parametri disponibili per la ricerca sono: la definizione di tipologia di luoghi ricercati, scelta tra i valori di “Sicurezza”, “Sanità” od “Interesse”; a seconda della tipologia di categoria selezionata si può impostare anche il parametro di Categoria, scelto da un elenco di valori possibili, quali “Medico” o “Luoghi – Sanità” se la tipologia scelta è quella di “Sanità”; “Forze dell'ordine”, “Uffici” o “Servizi – Sicurezza” se la categoria scelta è quella di “Sicurezza” oppure tra “In viaggio”, “Servizi – Interesse” o “Luoghi – Interesse” se la categoria scelta è quella

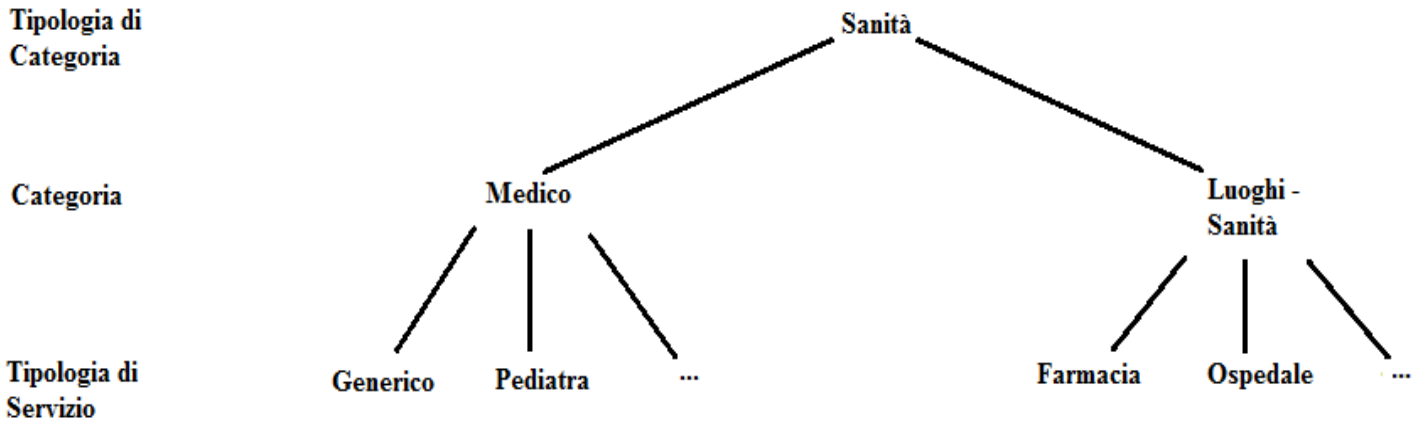
di “Interesse”. Se si imposta anche il parametro di Categoria è possibile specificare anche quello di Tipologia di servizio, scelto fra una nutrita lista di valori che varia a seconda della categoria scelta. Questi tre parametri sono legati tra loro in cascata: infatti anche se il campo di ricerca è comunque visibile nella schermata, non posso specificare una Categoria di luoghi se prima non scelgo una Tipologia di Categoria ed analogamente non posso definire una Tipologia di Servizio se prima non scelgo una Categoria. Tale impedimento è segnalato all'utente dall'applicazione tramite un popup di allerta quando si tenta di specificare uno di questi parametri senza aver prima scelto gli altri necessari. Inoltre su questi tre parametri è possibile eseguire una scelta multipla, cioè si può specificare più di un singolo valore per il parametro. Di conseguenza la lista di valori disponibile per i parametri in cascata si allunga includendo tutti quelli definiti dalla scelta dei parametri precedenti, ad esempio se come Tipologia di Categoria imposto “Sicurezza” e “Sanità” fra le possibili scelte del parametro Categoria mi compariranno tutte le opzioni possibili legate a “Sicurezza” e “Sanità” insieme.

Disegnati di seguito troviamo gli schemi che legano tra di loro i tre parametri appena spiegati e le scelte disponibili a seconda di selezioni singole o multiple dei valori offerti dai parametri di ricerca.

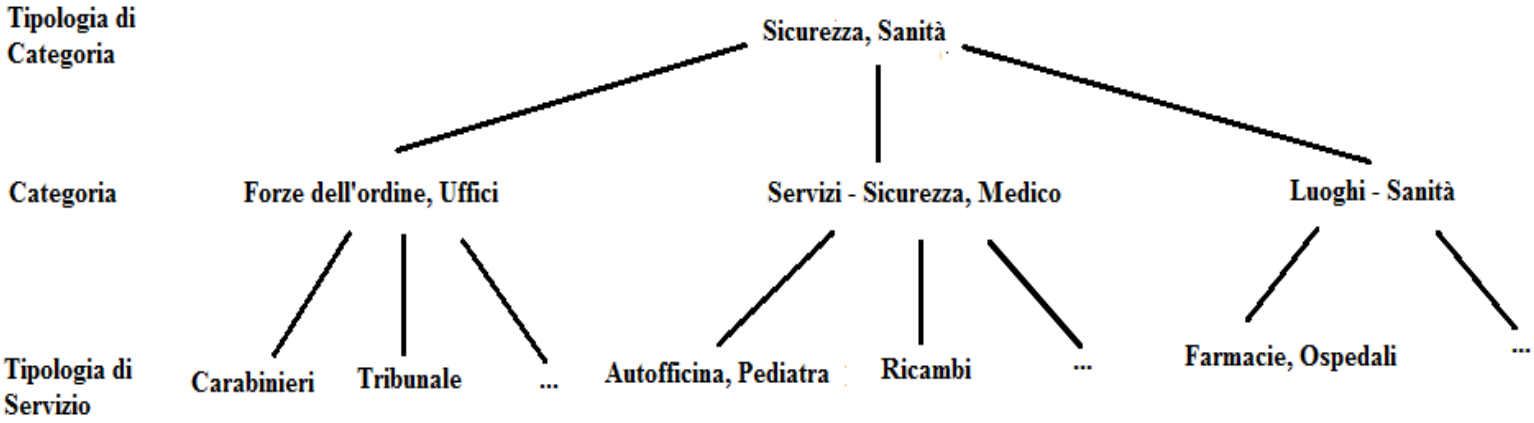
Esempio di scelta di singoli valori







Esempio di scelta di valori multipli



Gli altri parametri utili per raffinare la ricerca dei luoghi sono quelli di Nome POI, che consente di specificare il nome o parte del nome dei luoghi da ricercare, utile se si conosce soltanto il nome del luogo ma non la sua ubicazione; Comune, il quale consente di restringere la ricerca solo ai luoghi

situati nell'elenco di comuni scelti nel parametro (infatti anche qui è possibile operare la scelta multipla); Indirizzo, che consente di specificare una via od un indirizzo dei luoghi ricercati; Servizi, che consente di specificare la funzione svolta nei luoghi da ricercare; Giorni di Apertura, che consente di filtrare i luoghi cercati a seconda dei giorni nei quali sono aperti al pubblico, utile soprattutto nella ricerca di uffici o studi medici. Anche per i Giorni di Apertura è possibile effettuare una scelta multipla tra i valori proposti dall'applicazione. I campi di ricerca che consentono di specificare i parametri della ricerca sono di due tipi: menù con tutti i possibili valori per i parametri sui quali è possibile effettuare la scelta multipla o campo di testo per tutti gli altri parametri.

Le informazioni sui luoghi ottenibili dalla ricerca sono memorizzate in un server online e reperite tramite richieste WMS ad esso indirizzate. WMS (Web Map Service) è una specifica tecnica utilizzata per contattare i server web geografici ed ottenere da loro dati spazialmente riferiti da informazioni geografiche. Questa tecnica utilizza una mappa come rappresentazione delle informazioni geografiche mentre fa restituire dal server un'immagine digitale idonea ad essere visualizzata. Nel nostro caso, come risultato di una richiesta WMS il server restituisce una specie di lucido (layer) che viene sovrapposto dall'applicazione alla mappa di Google caricata all'avvio in modo tale da indicare con delle icone colorate dove sono situati i luoghi ottenuti dalla ricerca. Per far ciò bisogna specificare nell'URL di richiesta al server alcuni parametri quali la dimensione dell'area interessata (a seconda del livello di zoom applicato), il formato dell'immagine lucido ottenuta come risposta dal server e la query da eseguire come filtro ottenuta assemblando i parametri di ricerca selezionati. Le icone di identificazione dei luoghi si suddividono per colore, infatti per un luogo legato alla tipologia “Sanità” viene utilizzata un'icona di colore rosso, per un luogo legato alla tipologia “Sicurezza” viene utilizzata un'icona di colore blu mentre per un luogo di tipologia “Interesse” viene utilizzata un'icona di colore verde.

Una volta visualizzati i luoghi ottenuti dalla ricerca, possiamo andare a leggere le informazioni riguardanti il luogo toccando sopra l'icona che lo identifica, aprendo così la pagina di visualizzazione delle informazioni. Tali informazioni sono sempre ottenute dal server tramite una richiesta WMS con parametri diversi da quelli impostati per ottenere il lucido. Infatti, in questo caso il server restituisce all'applicazione non un'immagine, ma una pagina web che contiene l'elenco di tutte le informazioni generali disponibili per il luogo selezionato. Sarà poi compito dell'applicazione estrapolare i dati e visualizzarli secondo le direttive di layout della pagina di descrizione. Le informazioni principali reperite dal server sono quelle indicanti l'indirizzo del luogo, l'eventuale numero di telefono, i servizi forniti, la tipologia di servizi alla quale il luogo appartiene e i giorni di apertura con i relativi orari. Inoltre, se presenti, si possono visualizzare due immagini del posto con visuale dalla strada, le quali vengono ingrandite se si tocca sopra una di esse.

Dalla pagina di descrizione del luogo si può, inoltre, attivare la funzionalità di indicazioni stradali, che serve a reperire la strada da percorrere per raggiungere il luogo selezionato partendo da dove si trova attualmente il dispositivo. È per garantire questa funzionalità che l'applicazione richiede all'avvio l'abilitazione di ricerca della posizione del dispositivo tramite GPS o cellID, in quanto serve a determinare il punto di partenza per le indicazioni stradali. Si può inoltre scegliere in quale modo ottenere le indicazioni stradali: collegandosi ad internet e sfruttando la funzione offerta da Google per ottenere le indicazioni stradali, oppure utilizzando il navigatore presente all'interno del dispositivo Android.
Nelle pagine di ricerca e descrizione è possibile scegliere la lingua di visualizzazione dei testi delle etichette che identificano i dati, scelta ristretta a italiano e inglese, mentre i risultati ottenuti dal server sono sempre in lingua italiana perché presenti solo in tale lingua nel server. Inoltre la restrizione delle ricerche alla sola regione Campania è dovuta al fatto che nei database del server sono presenti le informazioni solo dei luoghi di tale regione. Se in futuro verranno aggiunte le informazioni dei luoghi anche di altre regioni, grazie a questa applicazione sarà possibile ottenere come risultati luoghi presenti in tutte le regioni aggiunte.

Capitolo 2: Requisiti dell'applicazione e scelte tecniche operate per realizzarle

2.1 Strumenti utilizzati

Per creare l'applicazione descritta in “Progetto” sono stati impiegati sia strumenti hardware che software specifici. Come supporto software per la scrittura di codice Java è stata utilizzata la piattaforma di sviluppo Eclipse opportunamente integrata con i plug-in dedicati alla scrittura e compilazione di codice Java specifico per dispositivi Android; in particolare il progetto è stato sviluppato per la versione di Android 2.2, mentre la fase di test è stata eseguita su di un dispositivo Android invece che tramite un simulatore in quanto l'utilizzo di un dispositivo esterno rendeva molto più pratico e veloce il lavoro di test dell'applicazione. Inoltre, dato che l'applicazione utilizza la connessione alla rete internet via Wi-Fi o 3G era necessario disporre di un dispositivo con installato Android per verificare il corretto andamento di tali funzionalità. Durante la parte di sviluppo come dispositivo Android è stato utilizzato uno smartphone della Samsung, il Galaxy Next, mentre l'applicazione finita è stata testata anche su altri tipi di dispositivi Android, tranne tablet, in quanto l'applicazione non è stata progettata per funzionare su di essi.
Durante lo sviluppo dell'applicazione si è potuto contare sull'aiuto del dipendente dell'azienda esperto in programmazione per dispositivi Android soprattutto per la parte di gestione delle

chiamate al server WMS e per la parte riguardante l'utilizzo del GPS o cellID per la localizzazione iniziale della posizione del dispositivo sul quale viene lanciata l'applicazione realizzata.

2.2 Scelte tecniche effettuate

Per poter ottenere i punti di interesse e le informazioni relative ad essi della regione Campania, come richiesto dai committenti dell'applicazione, si è deciso di appoggiarsi ad un server web gratuito, già creato da altre aziende, che fornisse tali informazioni a seguito di adeguate richieste passate con parametri di ricerca specifici. L'alternativa sarebbe stata quella di creare un proprio server ad hoc contenente le informazioni necessarie al funzionamento dell'applicazione, ma tale operazione sarebbe risultata troppo onerosa in termini economici e di tempo.
Come mappa di base dove vengono indicate le posizioni dei luoghi di interesse viene utilizzata la mappa fornita da Google, in quanto risulta essere fra le più utilizzate e quindi familiari all'utente ed inoltre è già supportata dalla maggior parte dei dispositivi Android attualmente in commercio.
Altra importante questione è risultata essere la frequenza di aggiornamento della schermata del dispositivo quando vi sono già visualizzati sopra mappa e luoghi di interesse; infatti se tale operazione avvenisse ogni qualvolta l'utente modifica la zona di visualizzazione della mappa, anche per piccoli spostamenti di essa sullo schermo verrebbe richiamata la procedura per ottenere le posizioni dei luoghi dal server web caricando eccessivamente di lavoro l'applicazione e riducendone sensibilmente prontezza e fluidità di funzionamento. Per evitare tali inconvenienti si è deciso di effettuare l'aggiornamento della mappa e dei punti di interesse individuati sullo schermo del dispositivo solo se ci si sposta di una distanza significativa dalla zona visualizzata nella mappa.
Infine si è presentata una scelta su come realizzare la modalità di richiesta delle indicazioni stradali per raggiungere un punto di interesse selezionato, in quanto il sistema Android consente di reperire tali informazioni dal servizio in internet fornito da Google oppure di richiamare le funzioni di un navigatore eventualmente installato nel dispositivo. Si poteva obbligare l'utente ad avvalersi del servizio fornito da Google, invece alla fine si è optato per inserire un menù di scelta quando l'utente richiede le indicazioni stradali. Grazie a questo menù l'utente può liberamente scegliere tra quali dei due metodi citati sopra avvalersi per ottenere le indicazioni che gli occorrono per poter raggiungere il luogo selezionato.

2.3 Problematiche riscontrate

Durante lo sviluppo dell'applicazione sono sorte alcune problematiche ad intralciare i lavori, tra cui quella di come riuscire a piazzare sopra la mappa di Google i punti di interesse forniti dal server web, in quanto sono disegnati su di un'immagine in formato png. La soluzione a tale problema è risultata essere quella di costruire sopra la porzione di mappa visualizzata sullo schermo un canvas, sul quale possiamo disegnare sopra l'immagine raffigurante i luoghi trovati ed in questo modo riuscire a sovrapporli alla mappa per darne l'ubicazione geografica. Sorge però un altro inconveniente, ovvero che su di un canvas di Java non si riesce a piazzare un'immagine in formato png come quella restituita dal server. Si è reso necessario quindi convertire l'immagine in formato bitmap per poter eseguire questa operazione ed ottenere così la mappa con indicate sopra le posizioni dei punti di interesse trovati.

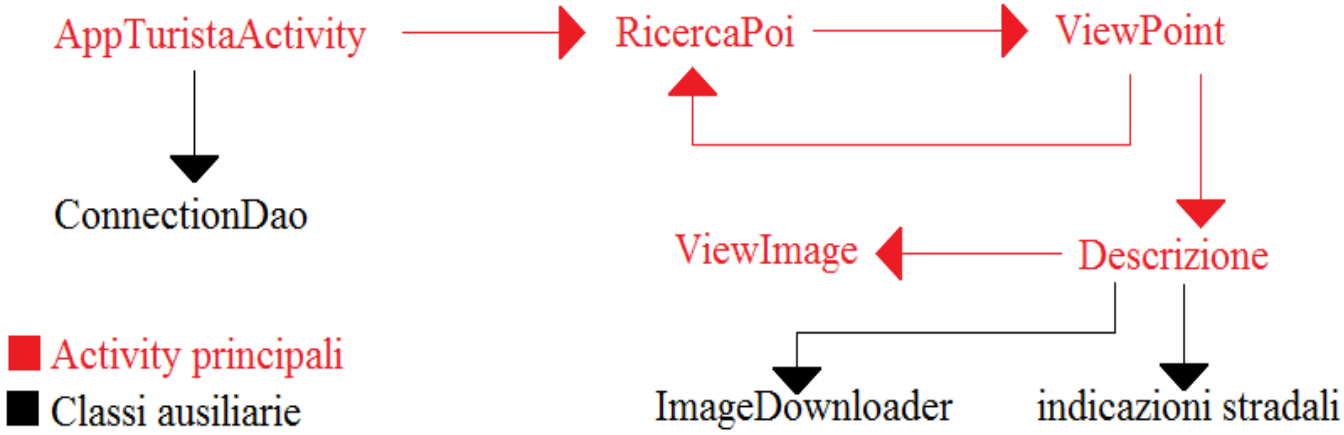
Sempre a proposito dei dati forniti in risposta dal server web si è riscontrata una certa difficoltà nel separare le singole informazioni di dettaglio riguardanti un luogo selezionato sullo schermo. Infatti tali informazioni sono restituite dal server all'interno di una pagina web testuale. Per riuscire ad estrarle si è reso quindi necessario scaricare e salvare la pagina web e, successivamente, grazie all'utilizzo di un parser creato appositamente, è stato possibile suddividere le informazioni del luogo selezionato e farle visualizzare correttamente dall'applicazione. Per quanto riguarda invece le immagini del punto di interesse selezionato, nella pagina web viene riportato solo l'indirizzo internet dal quale è possibile scaricarle. Ci si è trovati di fronte al problema di come poter fare in modo che l'applicazione riuscisse ad ottenere le immagini avendo a disposizione l'indirizzo web dal quale reperirle e, fortunatamente, è stata trovata in rete una classe Java che svolga tale compito, riuscendo a scaricare l'immagine come flusso di byte per poi salvarla già sotto forma di immagine senza altri passaggi intermedi.

L'ultima problematica rilevante riguarda la qualità dei dati ottenuti dal server web, infatti l'applicazione deve avere la possibilità di scelta della lingua di utilizzo fra le due disponibili, italiano e inglese, mentre le informazioni di dettaglio dei punti di interesse sono disponibili solo in lingua italiana. Per limitare tale problematica si è fatto in modo di tradurre in inglese tutto ciò che si poteva, per cui le etichette di identificazione dei campi di ricerca ed il testo dei pulsanti dell'applicazione sono disponibili in entrambe le lingue, mentre le informazioni di dettaglio di un luogo vengono visualizzate in lingua italiana anche se si è selezionata la lingua inglese.

2.4 Descrizione della struttura dell'applicazione e delle classi

La prima Activity lanciata dall'applicazione ha il nome di AppTuristaActivity e serve a caricare la mappa di Google usata come base dell'applicazione e di controllare che il dispositivo sia abilitato alla connessione 3G o Wi-Fi grazie all'utilizzo dei metodi descritti dalla classe ConnectionDao. Tramite un pulsante di ricerca si passa all'Activity chiamata RicercaPoi, nella quale si possono specificare i parametri di ricerca per ottenere i punti di interesse desiderati. La parte di richiesta dati al server viene eseguita dall'Activity ViewPoint, alla quale si accede dalla schermata di ricerca tramite il pulsante “cerca”. Oltre a richiedere dati al server, questa Activity si preoccupa di visualizzare correttamente sulla mappa le posizioni dei luoghi ottenuti dal server. A questo punto è possibile avviare una nuova ricerca di luoghi tornando all'Activity RicercaPoi, oppure, premendo sopra una delle icone che rappresentano i punti di interesse trovati, accedere alla schermata di visualizzazione delle informazioni riguardanti il luogo selezionato. La corretta visualizzazione di queste informazioni è lasciata come compito all'Activity Descrizione, la quale prende i dati passategli dall'Activity precedente e li dispone sotto forma di elenco ordinato. Inoltre, grazie all'aiuto della classe ImageDownloader, riesce a reperire dal server web le immagini del luogo se esse sono disponibili. Dalla schermata di elenco delle informazioni di dettaglio si può passare alla funzione di visualizzazione delle indicazioni stradali verso il luogo selezionato, disponibili all'utente tramite l'utilizzo di un navigatore proprio installato nel dispositivo oppure grazie al servizio internet di Google. Infine, se viene premuta una delle due immagini del luogo si passa all'Activity ViewImage che ha il compito di visualizzare a schermo intero l'immagine selezionata per poterla guardare meglio.

Riportiamo uno schema riassuntivo delle Activity e delle classi che formano l'applicaizone realizzata.



Capitolo 3: Dettagli delle classi realizzate

3.1 Descrizione della classe AppTuristaActivity

Questa classe contiene l'Activity lanciata al momento dell'apertura dell'applicazione sul dispositivo. Essendo la prima Activity ha il compito di caricare tutte le variabili che verranno utilizzate nel corso del funzionamento dell'applicazione e di verificare lo stato delle connessioni del dispositivo.
Per prima cosa, in questa classe sono dichiarate le variabili che verranno utilizzate per poter utilizzare le funzionalità dell'applicazione che subito dopo viene lanciata dal metodo `onCreate`. In questo metodo sono scritte le istruzioni per caricare la visualizzazione dell'applicazione descritta nel file di layout main, poi si passa alla verifica della connessione del dispositivo ad una rete Wi-Fi o alla rete 3G grazie all'utilizzo dei metodi della classe `ConnectionDao` e, dato che se nessuna delle due connessioni è attiva l'applicazione non può funzionare al pieno delle proprie funzionalità, viene segnalato all'utente tale limitazione tramite un popup di alert a video.

```
setContentView(R.layout.main);
//connessione telefono
ConnectivityManager cm = (ConnectivityManager) this
            .getSystemService(Context.CONNECTIVITY_SERVICE);
//verifica abilitazione 3G o Wi-Fi tramite classe ConnectionDao
if (!ConnectionDao.isConnected3G(cm)
            && !ConnectionDao.isConnectedWifi(cm)) {
      //creazione popup di alert
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setTitle("Attenzione")
                  .setIcon(android.R.drawable.ic_dialog_info)
                  .setMessage("Per utilizzare l'applicazione occore
                              essere connessi alla rete")
                  .setCancelable(false)
                  .setPositiveButton("OK",
                  new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                                                int id) {
                              finish();
                        }
                  });
      AlertDialog alert = builder.create();
```

```
                alert.show();
            }
```

Una volta stabilito che il dispositivo è correttamente collegato alla rete internet tramite Wi-Fi oppure 3G il metodo passa al controllo della ricerca di posizione del dispositivo tramite GPS oppure cellID per poter garantire la funzionalità dell'applicazione che consente di ottenere le indicazioni stradali dal punto in cui ci si trova fino al punto di interesse scelto come destinazione.
La verifica della possibilità di stabilire la posizione del dispositivo viene eseguita dal metodo `isLocationServiceAvaiable`, il quale va a ricercare fra le impostazioni del dispositivo se è abilitata quella di rintracciabilità della posizione.

```
//funzione per vedere se e' abilitato posizionamento
  public boolean isLocationServiceAvaiable(Context context) {
      String provider = Settings.Secure.getString(context.getContentResolver(),
                                    Settings.Secure.LOCATION_PROVIDERS_ALLOWED);
      if(provider != null){
         if(provider.equals("")){
                 return false;
         }
         else {
                 //OK
                 return true;
         }
      } else {
         return false;
  }
```

Se la verifica della ricerca di posizione tramite GPS o cellID risulta negativa, l'applicazione provvede ad avvertire l'utente di abilitare uno dei due metodi con la comparsa di un popup informativo nel quale è spiegato che per poter funzionare, l'applicazione necessita dell'attivazione del GPS o del cellID.

```
//verifica abilitazione GPS o cellID con metodo ad hoc
      else if (!isLocationServiceAvaiable(getApplicationContext())) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Info").setIcon(android.R.drawable.ic_dialog_info)
                  .setMessage("Per utilizzare l'applicazione occore attivare il
                        GPS o la ricezione della posizione tramite cellID")
```

```
            .setCancelable(false)
            .setPositiveButton("OK",new DialogInterface.OnClickListener(){
                        public void onClick(DialogInterface dialog,int id)
                              {Intent myIntent = new Intent(
                              Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                              startActivity(myIntent);
                              }
                        });
            AlertDialog alert = builder.create();
            alert.show();
      }
```

Successivamente si passa all'inizializzazione della mappa secondo le impostazioni desiderate e secondo quelle specificate nel layout di main richiamato all'inizio del metodo `onCreate`. Inoltre, dopo aver caricato la mappa di Google sullo schermo, la visualizzazione della mappa si centra automaticamente sulla posizione in cui si trova il dispositivo, rintracciata tramite GPS o cellID.

```
//inizializza mappa
mapView = (MapView) findViewById(R.id.mapview);
mapView.setBuiltInZoomControls(true);
mapView.getController().setZoom(6);
mapView.getController().setCenter(punto);
//sposta schermo su mia posizione
myLocationOverlay = new MyLocationOverlay(this, mapView);
myLocationOverlay.runOnFirstFix(new Runnable() {
      public void run() {
            mapView.getController()
                  .animateTo(myLocationOverlay.getMyLocation());
            mapView.getController().setZoom(15);
      }
});
mapView.getOverlays().add(myLocationOverlay);
```

Infine vengono caricati i pulsanti di ricerca dei luoghi di interesse e di zoom della mappa, con la definizione di cosa accade quando vengono cliccati.

Se si clicca sul pulsante di ricerca, l'applicazione fa partire l'Activity RicercaPoi che consente di impostare un filtro o meno sui luoghi di interesse che verranno rintracciati dall'Activity di ricerca e poi caricati sulla mappa visualizzata dal dispositivo; mentre se si clicca sui pulsanti di zoom viene

ingrandita o rimpicciolita la visualizzazione della mappa.

```
//tasto di ricerca
cerca = (ImageButton) findViewById(R.id.cerca);
cerca.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View arg0) {
          Intent i = new Intent(AppTuristaActivity.this, RicercaPoi.class);
          i.putExtra("zoom", mapView.getZoomLevel());
          i.putExtra("long", mapView.getMapCenter().getLongitudeE6());
          i.putExtra("lat", mapView.getMapCenter().getLatitudeE6());
          startActivity(i);
    }
});
//tasti di  zoom
ZoomControls zoomControls = (ZoomControls) findViewById(R.id.zoomcontrols);
zoomControls.setOnZoomInClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
          mapView.getController().zoomIn();
    }
});
zoomControls.setOnZoomOutClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
          mapView.getController().zoomOut();
    }
});
```

Gli ultimi tre metodi della classe servono per definire come si comporta l'Activity quando l'applicazione non è più usata dall'utente, tramite il metodo `onStop` nel quale si impone all'applicazione di disabilitare l'aggiornamento continuo della posizione del dispositivo tramite il comando `myLocationOverlay.disableMyLocation();` quando viene richiamata dalla messa in background, tramite il metodo `onResume` che riabilita l'aggiornamento continuo della posizione del dispositivo; mentre l'ultimo metodo dice all'applicazione di non fare nulla di particolare nel caso il display venga ruotato.

3.2 Descrizione della classe ConnectionDao

La funzione di questa classe consiste nel verificare se il dispositivo Android è connesso o può connettersi alla rete internet tramite Wi-Fi o 3G. Per far ciò sono presenti due metodi all'interno della classe chiamati `isConnectedWifi` e `isConnected3G` che mi ritornano un valore booleano positivo o negativo a seconda della disponibilità di connessione ad internet del dispositivo. A questi metodi viene passato un oggetto `ConnectivityManager` contente tutte le informazioni riguardanti le proprietà di connessione del dispositivo. In particolare grazie al metodo `getNetworkInfo`, al quale passiamo come parametro la tipologia di connessione richiesta da verificare, riusciamo ad ottenere le informazioni cercate, perché tale metodo tenta di iniziare una connessione alla rete definita dal parametro passato.

```
public static boolean isConnectedWifi(ConnectivityManager cm) {
            if (IS_EMULATOR) return false;
            NetworkInfo ni= cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
            return ni.isConnected();

      }

public static boolean isConnected3G(ConnectivityManager cm) {
            if (IS_EMULATOR) return true;
            NetworkInfo ni= cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
            long timeoutMillis=CONNECTION_TIMEOUT;
            long sleep=100L;
            while (timeoutMillis>=0 && !ni.isConnected()) {
                  try {
                        Thread.sleep(sleep);
                        timeoutMillis-=sleep;
                  }
                  catch (InterruptedException e) {
                  }
            }
            return ni.isConnected();
      }
```

All'inizio di ciascuno dei due metodi è presente un'istruzione di verifica per capire se l'applicazione sta girando su di un dispositivo reale oppure su un emulatore. Ciò consente di utilizzare questi

metodi anche in caso di test tramite emulatore in quanto è possibile simulare la connessione 3G ma non quella Wi-Fi.
La connessione tramite 3G può subire rallentamenti a seconda della ricezione del dispositivo. Per evitare che la connessione 3G non sia mancata solo perché non risulta esserci abbastanza campo, nel metodo `isConnected3G` viene applicato un timeout prima di dichiarare fallita la connessione tramite 3G.

3.3 Descrizione della classe Descrizione

Questa Activity viene richiamata quando si preme l'icona di un luogo trovato dopo la ricerca. Vengono richiamate e visualizzate tutte le informazioni reperite sul luogo tramite internet grazie alla chiamata WMS al server ed inoltre viene predisposto il pulsante per accedere alla funzionalità di reperimento delle indicazioni stradali per raggiungere il luogo.
Per prima cosa la classe controlla se, nella schermata di ricerca precedentemente interpellata per ottenere i luoghi, la lingua fosse settata su italiano od inglese per caricare il rispettivo layout con le etichette dei campi scritte nella giusta lingua.

```
String value = getIntent().getStringExtra("value");
//se descrizione inglese o italiano
if (value == null || value.equals("it"))
      setContentView(R.layout.descrizione);
else if (value.equals("en"))
      setContentView(R.layout.descrizione_en);
```

Successivamente viene richiamato il metodo `initialize` che procede al reperimento dei dati dalla pagina web restituita dalla chiamata WMS al server ed alla loro visualizzazione secondo le impostazioni nel file di layout di descrizione. Vengono per primi creati i due bottoni di scelta della lingua nella pagina di descrizione, facendo in modo che alla loro eventuale pressione venga caricato il corretto layout in italiano od inglese.

```
//imposto pulsanti per scelta italiano o inglese
it = (RadioButton) findViewById(R.id.btnIt);
it.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            setContentView(R.layout.descrizione);
```

```
                initialize();
            }
        });
        en = (RadioButton) findViewById(R.id.btnEn);
        en.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                setContentView(R.layout.descrizione_en);
                initialize();
            }
        });
```

Subito dopo vengono richiamati i dati ricevuti dal server e pronti ad essere estrapolati e vengono impostati i campi che conterranno tali dati, dando loro come stile di caratteri quelli della famiglia Verdana

```
//salvo i dati provenienti dal WMS
ris = getIntent().getStringExtra("dati");
//imposto gli elementi della pagina
indicazioni = (ImageButton) findViewById(R.id.inidicazioni);
nome = (TextView) findViewById(R.id.nome);
indirizzo = (TextView) findViewById(R.id.indirizzo);
indirizzo.setTypeface(Typeface.createFromAsset(getAssets(),
                                                "Verdana.ttf"));
tel            =            (TextView)            findViewById(R.id.tel);
tel.setTypeface(Typeface.createFromAsset(getAssets(),"Verdana.ttf"));
```

Si passa poi alla fase di estrapolazione dei dati, salvando in una stringa tutto il contenuto della pagina web con le informazioni ottenute da internet e dividendola in modo da avere le singole informazioni per poterle scrivere al posto giusto nella pagina di descrizione.

```
//ottengo info dal file arrivato via WMS
String[] split1 = ris.split(System.getProperty("line.separator"));
for (int i = 0; i < split1.length; i++) {
            String[] split2 = split1[i].split("=");
```

Il ciclo for consente serve per poter leggere tutto il contenuto della stringa dove abbiamo salvato i

dati da visualizzare e, a seconda di ciò che rappresentano, tali dati verranno estrapolati e visualizzati a dovere. Il secondo split si rende necessario in quanto i dati sono forniti uno per riga secondo la notazione “NomeCampo” = “Valore”. Come prima cosa controllo se i dati indicano il nome del luogo selezionato,

```
if (split2[0].equals("poi_name ")) {
      nome.setText(split2[1]);
```

poi controllo se indicano i giorni di apertura con i rispettivi orari se presenti,

```
else if (split2[0].equals("orari_poi ")) {
      pos = 1;
      if (split2[1].equals(" "))
            apertura.setText("Non disponibili");
      else {
            //info giorni apertura
            String[] finalSplit = split2[1].split(" ");
            for (int k = 0; k < finalSplit.length; k++) {
                  if (finalSplit[k].equals("Luned“")) {
                        apertura.setText(finalSplit[k]
                                    .replaceFirst(" ", "") + " ");
                  } else if (finalSplit[k].equals("Marted“")||
                              finalSplit[k].equals("Mercoled“")|
                              finalSplit[k].equals("Gioved“")||
                              finalSplit[k].equals("Venerd“")||
                              finalSplit[k].equals("Sabato")||
                              finalSplit[k].equals("Domenica")  &&
                              k != 0) {
                                    apertura.append("\n\n" +
                                          finalSplit[k] + " ");

                        } else if (finalSplit[k].equals("dalle")) {
                              //info orari apertura
                              apertura.append("\n          " +
                              finalSplit[k] + " ");
                        } else if (finalSplit[k].equals("dall")) {
                              apertura.append("\n          " +
                              finalSplit[k] + "e ");
                        } else
                              apertura.append(finalSplit[k] + " ");
```

segue un controllo che verifica se i dati letti nel ciclo for rappresentano le immagini relative al luogo e, in caso affermativo, procedono al loro download grazie alle funzioni di una classe ausiliaria chiamata `ImageDownloader` reperita da internet il cui scopo è di consentire il download delle immagini da internet per poi poterle utilizzare nell'applicazione per Android. Infatti di tale classe ausiliaria vengono utilizzati solo il metodo `download`, che consente di scaricare l'immagine da internet, e `clearCache`, che serve per pulire lo spazio di memoria occupato dalla cache durante il download dell'immagine. Inoltre per segnalare che l'applicazione sta scaricando l'immagine mentre continua lo stesso a funzionare attiviamo un messaggio di informazione grazie alla classe Toast definita per i dispositivi Android. I Toast sono wiget di Android che consentono di visualizzare brevi messaggi sotto forma di piccole finestre in fondo allo schermo mentre l'Activity principale continua ad essere svolta.

```
else if (split2[0].equals("image_street ")) {
      pos = 2;
      String x = split2[3].replace('\'', ' ');
      String[] lastSplit = x.split(" ");
      url1 = lastSplit[1];
      ImageDownloader imageDownloader = new ImageDownloader();
      Toast.makeText(this, "Caricamento immagini...",
                        1).show();
      imageDownloader.download(url1, img1);
      imageDownloader.clearCache();
```

Successivamente si passa a verificare se i dati letti contengono informazioni sul numero di telefono del luogo o su quali servizi vengono svolti al suo interno con un semplice controllo come quello utilizzato per la verifica del nome del luogo,

```
else if (split2[0].equals("telephone ")) { //info telefono
      pos = 4;
      if (split2[1].equals(" "))
            tel.setText("Non disponibile");
      else {
            tel.setText(split2[1].replaceFirst(" ", ""));
      }
} else if (split2[0].equals("services ")) { //info servizio
      pos = 5;
      if (split2[1].equals(" "))
```

```
                        servizi.setText("Non disponibili");
                  else {
                        servizi.setText(split2[1].replaceFirst(" ", ""));
                  }
            }
```

poi si passa alla verifica della presenza di informazioni riguardanti l'indicazione della tipologia di servizi offerti nel luogo selezionato e di note di accompagnamento per la descrizione di tali servizi, se presenti

```
      else if (split2[0].equals("note ")||
                  split2[0].equals("tipology_of_category ")) {
                  pos = 6;
                  if (split2[0].equals("note ")) {
                        note = split2[1].replaceFirst(" ", "");
                  }
                  else if (split2[0].equals("tipology_of_category ")) {
            //info tipologia di categoria, accodo le note se presenti
                        if (!note.equals(""))
                              tipServizi.append(split2[1].replaceFirst("
                              ", "")+ " - " + note);
                        else if (split2[1].equals(" ") || note.equals(""))
                              tipServizi.append(split2[1].replaceFirst("
                              ", "")+note);
                        else if (split2[1].equals(" ") || note.equals(""))
                              tipServizi.append(split2[1].replaceFirst("
                              ", ""));
                        else
                              tipServizi.setText("Non disponibili");
                  }
            }
```

Come ultimo controllo delle informazioni rimane la verifica se la riga di dati letta rappresenta l'indirizzo o il comune dove è situato il luogo.

```
//info indirizzo
else if (split2[0].equals("indirizzo ") || split2[0].equals("comune ")) {
      pos = 7;
      if (split2[1].equals(" "))
            indirizzo.setText("Non disponibile");
```

```
        else {
              if (split2[0].equals("indirizzo "))
                    indirizzo.append(split2[1].replaceFirst(" ", ""));
              else if (split2[0].equals("comune "))
                          indirizzo.append(" - "+ split2[1]
                                                .replaceFirst("", ""));
              }
        }
```

Una volta letti ed impostati tutti i campi contenenti le informazioni dei luoghi, l'Activity di descrizione imposta il funzionamento del pulsante per ottenere le indicazioni stradali. Premendo tale pulsante l'applicazione fa si che venga lanciata un'Activity generale del telefono che richiama l'utente alla scelta di come ottenere le indicazioni stradali, se tramite internet oppure utilizzando uno dei navigatori installati nel dispositivo, passando poi all'applicazione scelta i punti di partenza, cioè il posto dove viene localizzato il dispositivo, e di arrivo, cioè il luogo di cui abbiamo richiesto la pagina di descrizione.

```
indicazioni.setOnClickListener(new OnClickListener() {
//indicazioni stradali da dove sono al punto selezionato
            @Override
            public void onClick(View arg0) {
                  GeoPoint Loc = new GeoPoint(getIntent().getIntExtra(
                              "pointLatitude", 0),getIntent().getIntExtra(
                              "pointLongitude", 0));
                  GeoPoint myLoc = new GeoPoint(getIntent().getIntExtra(
                              "myLatitude", 0), getIntent().getIntExtra(
                              "myLongitude", 0));
                  startActivity(new Intent(Intent.ACTION_VIEW, Uri
                  .parse("http://maps.google.com/maps?saddr="
                        + myLoc.getLatitudeE6() * COSTANTE + ","
                        + myLoc.getLongitudeE6() * COSTANTE + "&daddr="
                        + Loc.getLatitudeE6() * COSTANTE + ","
                        + Loc.getLongitudeE6() * COSTANTE)));
            }
});
```

Come ultima operazione la classe Descrizione imposta il comportamento dell'applicazione quando l'utente preme una delle immagini presenti nella descrizione del luogo, se caricate nel database del

server. Quando viene premuta una di tali immagini si passa alla pagina gestita dall'Activity ViewImage che consente di vedere a schermo pieno l'immagine selezionata.

```
img1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
                  Intent i = new Intent(Descrizione.this,ViewImage.class);
                  i.putExtra("img", url1);
                  startActivity(i);
            }
      });
```

3.4 Descrizione della classe RicercaPoi

L'Activity definita in questa classe ha il compito di impostare correttamente i campi utilizzati per la ricerca di luoghi, in particolare settando la lingua delle etichette che identificano i vari campi e caricando i dati corretti tra cui scegliere per i parametri con valori a scelta multipla; inoltre esegue controlli sui campi di Categoria e Tipologia di servizi, in quanto ricordiamo che non si può specificare una Categoria di luoghi se prima non viene scelta una Tipologia di categoria e non si può scegliere una Tipologia di servizi presenti nel luogo se prima non si è scelto una Categoria di luoghi. Oltretutto, a seconda della Tipologia di categoria scelta vengono rese disponibili solo alcune Categorie di luoghi e lo stesso dicasi per le Tipologie di servizi con le Categorie.
Per prima cosa in questa classe vengono definite tutte le variabili utili allo svolgimento dell'Activity, in particolare vengono definiti degli Arraylist nei quali andremo a caricare tutti i dati selezionati per i parametri a scelta multipla, mentre gli altri parametri saranno rappresentati da stringhe contenenti il valore scritto nel relativo campo della ricerca.

```
String tipS,catS,tipcatS,nameS,comS,indS,servS,orariS = "";
CharSequence[] days = { "Luned“", "Marted“", "Mercoled“", "Gioved“",
            "Venerd“", "Sabato", "Domenica" };
ArrayList<CharSequence> selectedDay = new ArrayList<CharSequence>();
CharSequence[] coms;
ArrayList<CharSequence> selectedComs = new ArrayList<CharSequence>();
CharSequence[] tipo = { "Sanità", "Sicurezza", "Interesse" };
ArrayList<CharSequence> selectedTipo = new ArrayList<CharSequence>();
CharSequence[] Cat = new CharSequence[10];
ArrayList<CharSequence> selectedCat = new ArrayList<CharSequence>();
```

```
CharSequence[] TipCat;
ArrayList<CharSequence> selectedTipCat = new ArrayList<CharSequence>();
```

Subito dopo parte la creazione dell'Activity: viene controllata la lingua selezionata e,di conseguenza, caricate le etichette corrette dei campi di ricerca.

```
//lingua della ricerca
if (value == null || value.equals("it"))
      setContentView(R.layout.ricerca);
else if (value.equals("en"))
      setContentView(R.layout.ricerca_en);
initialize();
```

Nel metodo `initialize` chiamato vengono eseguiti tutti i settaggi dei campi di ricerca, a partire dai bottoni per il cambio di lingua con le istruzioni da svolgere in caso uno dei due sia premuto.

```
it = (RadioButton) findViewById(R.id.btnIt);
it.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            tipS = tip.getText().toString();
            catS = cat.getText().toString();
            tipcatS = tip_servizi.getText().toString();
            nameS = nome.getText().toString();
            comS = comune.getText().toString();
            indS = ind.getText().toString();
            servS = servizi.getText().toString();
            orariS = giorni.getText().toString();
            setContentView(R.layout.ricerca);
            initialize();
      }
});
en = (RadioButton) findViewById(R.id.btnEn);
en.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            tipS = tip.getText().toString();
            catS = cat.getText().toString();
            tipcatS = tip_servizi.getText().toString();
```

```
                nameS = nome.getText().toString();
                comS = comune.getText().toString();
                indS = ind.getText().toString();
                servS = servizi.getText().toString();
                orariS = giorni.getText().toString();
                setContentView(R.layout.ricerca_en);
                initialize();
        }
});
```

Poi si passa alla definizione dei campi di ricerca a scelta multipla, per ognuno dei quali viene definita la funzione da eseguire quando l'utente apre il menu di scelta multipla. I campi di ricerca in questione sono quelli di Tipologia di Categoria, Categoria, Tipologia di Servizi, Comuni e Giorni di Apertura.

```
//lista tipologie
tip = (Button) findViewById(R.id.buttonTip);
tip.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                showSelectTipDialog();
        }
});
//lista categorie
cat = (Button) findViewById(R.id.buttonCat);
cat.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                showSelectCatDialog();
        }
});
//lista tipologie di servizi
tip_servizi = (Button) findViewById(R.id.buttonTipoServizi);
tip_servizi.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                showSelectTipCatDialog();
        }
});
```

```
//lista comuni
comune = (Button) findViewById(R.id.buttonComune);
comune.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            showSelectComuneDialog();
      }
});
//lista giorni di apertura
giorni = (Button) findViewById(R.id.buttonGiorni);
giorni.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            showSelectDaysDialog();
      }
});
```

Successivamente vengono impostati i campi di ricerca testuali e vengono assegnati tutti i valori delle etichette dei campi di ricerca reperiti in precedenza dal layout corretto dopo la verifica eseguita all'inizio della classe.

```
nome = (EditText) findViewById(R.id.nomePoi);
servizi = (EditText) findViewById(R.id.servizi);
ind = (EditText) findViewById(R.id.indirizzo);
tip.setText(tipS);
cat.setText(catS);
tip_servizi.setText(tipcatS);
comune.setText(comS);
nome.setText(nameS);
servizi.setText(servS);
ind.setText(indS);
giorni.setText(orariS);
```

Infine viene impostato il comportamento del pulsante di ricerca, il quale, se premuto, legge tutti i dati selezionati nei campi di ricerca e passa il tutto all'Activity ViewPoint che procederà a contattare il server per reperire l'immagine di lucido da sovrapporre alla mappa e provvederà inoltre a disegnarcela sopra.

```
search = (ImageButton) findViewById(R.id.startSearch);
search.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            Intent i = new Intent(RicercaPoi.this, ViewPoint.class);
            i.putExtra("tip", tip.getText().toString());
            i.putExtra("cat", cat.getText().toString());
            i.putExtra("nome", nome.getText().toString());
            i.putExtra("serv", servizi.getText().toString());
            i.putExtra("tip_serv",tip_servizi.getText().toString());
            i.putExtra("comuni", comune.getText().toString());
            i.putExtra("ind", ind.getText().toString());
            i.putExtra("gg", giorni.getText().toString());
            i.putExtra("zoom", getIntent().getIntExtra("zoom", 0));
            i.putExtra("lat", getIntent().getIntExtra("lat", 0));
            i.putExtra("long", getIntent().getIntExtra("long", 0));
            if (it.isChecked())
                  i.putExtra("value", "it");
            else if (en.isChecked())
                  i.putExtra("value", "en");
            startActivity(i);
            finish();
```

Si passa poi alla definizione dei metodi necessari per il corretto funzionamento dei menu a scelta multipla. I primi metodi descritti riguardano il funzionamento del menu relativo alla scelta dei giorni. Per primo viene definito il metodo chiamato quando si apre il menu a scelta multipla. In questo metodo, chiamato `showSelectDaysDialog`, vengono caricati tutti i giorni della settimana nel menu, in modo tale che per ognuno di essi venga posto anche il pulsante di scelta multipla. Per prima cosa il metodo controlla che non ci siano già dei giorni caricati da una precedente apertura del menu.

```
boolean[] checkedDays = new boolean[days.length];
int count = days.length;

for (int i = 0; i < count; i++)
      checkedDays[i] = selectedDay.contains(days[i]);
```

Successivamente si passa alla definizione del menu a scelta multipla con le relative istruzioni da

eseguire quando un elemento del menu viene selezionato.

```
DialogInterface.OnMultiChoiceClickListener daysDialogListener = new
                        DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which,
                        boolean isChecked) {
                  if (isChecked)
                        selectedDay.add(days[which]);
                  else
                        selectedDay.remove(days[which]);
                  onChangeSelectedDays();
            }
      };
```

Viene poi impostato il titolo che compare sopra il menu a scelta multipla.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
if (it.isChecked())
      builder.setTitle("Giorni di Apertura");
else if (en.isChecked())
      builder.setTitle("Days Open");
```

Infine viene creato il menu passandogli i giorni che deve contenere, quelli da eventualmente evidenziare perché già selezionati e il titolo del menu.

```
builder.setMultiChoiceItems(days, checkedDays, daysDialogListener);
AlertDialog dialog = builder.create();
dialog.show();
```

Subito dopo viene descritto il metodo lanciato ogni volta che si seleziona un giorno di apertura dal menu. Tale metodo controlla quali sono i giorni selezionati nel menu a scelta multipla e provvede a scriverli dentro il campo di ricerca presente nella pagina di ricerca sottostante, infatti i menu a scelta multipla vengono proposti come dei popup che si sovrappongono alla schermata visualizzata.

```
public void onChangeSelectedDays() {
StringBuilder stringBuilder = new StringBuilder();
```

```
            //controllo quanti e quali giorni sono selezionati
            for (int x = 0; x < selectedDay.size(); x++) {
                  if (selectedDay.size() == 1 || (selectedDay.size() - x == 1))
                        stringBuilder.append(selectedDay.get(x));
                  else
                        stringBuilder.append(selectedDay.get(x) + ",");
            }
            giorni.setText(stringBuilder.toString());
      }
```

Successivamente sono definiti altri due metodi, del tutto simili ai precedenti per impostare il funzionamento del menu per quanto riguarda la scelta dei comuni all'interno dei quali ricercare i luoghi. Questi due metodi sono denominati `showSelectComuneDialog` e `onChangeSelectedComs`.
Definiamo poi i metodi necessari al funzionamento dei menu di scelta per quanto riguarda i parametri di Tipologia di Categoria, Categoria e Tipologia di servizi. In questi metodi bisogna impostare, oltre al funzionamento dei menu come nei casi precedenti per Giorni di Apertura e Comuni, anche i controlli per impedire all'utente di scegliere una Categoria finché non ha scelto una Tipologia di categoria o di scegliere una Tipologia di Servizi finchè non ha scelto una Categoria.
Il primo metodo di questo blocco, chiamato `showSelectTipDialog` serve per costruire il menu di scelta delle Tipologie di Categorie nello stesso modo fatto per i menu precedenti.

```
public void showSelectTipDialog() {
            boolean[] checkedTips = new boolean[tipo.length];
            int count = tipo.length;
            for (int i = 0; i < count; i++)
                  checkedTips[i] = selectedTipo.contains(tipo[i]);
            DialogInterface.OnMultiChoiceClickListener daysDialogListener = new
                  DialogInterface.OnMultiChoiceClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which,
                              boolean isChecked) {
                        if (isChecked)
                              selectedTipo.add(tipo[which]);
                        else
                              selectedTipo.remove(tipo[which]);
                        onChangeSelectedTips();
                  }
            };
```

```
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        if (it.isChecked())
              builder.setTitle("Tipologia");
        else if (en.isChecked())
              builder.setTitle("Tipology");
        builder.setMultiChoiceItems(tipo, checkedTips, daysDialogListener);
        AlertDialog dialog = builder.create();
        dialog.show();
  }
```

Il metodo `onChangeSelectedTips`, invocato ogni qualvolta venga selezionata una Tipologia di categoria dal menu, serve per scrivere nel campo di ricerca i valori di tali scelte ed inoltre ha il compito di caricare le opzioni di scelta del menu delle Categorie a seconda di quali Tipologie vengano selezionate.

```
        if (selectedTipo.size() == 1 || (selectedTipo.size() - x == 1)) {
              stringBuilder.append(selectedTipo.get(x));
              if (selectedTipo.get(x).equals("Sanitˆ"))
                    san = 1;
              else if (selectedTipo.get(x).equals("Sicurezza"))
                    sic = 1;
              else if (selectedTipo.get(x).equals("Interesse"))
                    in = 1;
        } else {
              //scelte più tipologie
              //se più di una separate da virgola
              stringBuilder.append(selectedTipo.get(x) + ",");
              if (selectedTipo.get(x).equals("Sanitˆ"))
                    san = 1;
              else if (selectedTipo.get(x).equals("Sicurezza"))
                    sic = 1;
              else if (selectedTipo.get(x).equals("Interesse"))
                    in = 1;
        }
        //carico i dati per le categorie delle tipologie selezionate
              if (san == 1 && sic == 1 && in == 1) {
                    CharSequence[] ris = { "Medico", "Luoghi - Sanità",
                                "Forze dell'ordine", "Uffici", "Servizi -
                                Sicurezza","In viaggio", "Servizi -
```

```
                        Interesse","Luoghi - Interesse" };
                Cat = ris;
            } else if (san == 1 && sic == 1 && in == 0) {
                CharSequence[] ris = { "Medico", "Luoghi - Sanità",
                        "Forze dell'ordine", "Uffici", "Servizi -
                        Sicurezza" };
                Cat = ris;
            } else if (san == 0 && sic == 1 && in == 0) {
                CharSequence[] ris = { "Forze dell'ordine", "Uffici",
                        "Servizi - Sicurezza" };
                Cat = ris;
            } else if (san == 0 && sic == 0 && in == 1) {
                CharSequence[] ris = { "In viaggio", "Servizi -
                            Interesse","Luoghi - Interesse" };
                Cat = ris;
            }
            …
```

Viene definito poi il metodo `showSelectTipDialog` utilizzato per la creazione del menu di scelta della Categoria. Qui, in più rispetto ai metodi di creazione dei menu descritti finora, abbiamo inserito anche il controllo se è già stata eseguita una scelta fra le Tipologie di Categoria, altrimenti il menu non può essere caricato e l'applicazione lo segnala all'utente tramite un messaggio di allerta. Per eseguire questo controllo è stato sufficiente scrivere la riga di codice

```
if (selectedTipo.size() > 0) {
```

che certifica la scelta di almeno una Tipologia di Categoria tra quelle proposte dal menu precedente. Se la scelta è stata eseguita viene creato il menu a scelta multipla come già fatto per tutti quelli visti finora, altrimenti viene creata la finestra con il messaggio di allerta per l'utente che gli ricorda di eseguire tale scelta prima di poter aprire il menu delle categorie.

```
else {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    //in italiano
    if (it.isChecked()) {
        builder.setTitle("Attenzione")
            .setMessage("Prima di selezionare una
            Categoria devi aver scelto almeno una Tipologia")
```

```
                .setPositiveButton("OK",
                        new DialogInterface.OnClickListener()
                                public void onClick (DialogInterface dialog,
                                                int id)
                                {
                                        dialog.cancel();
                                }
                });
        } else if (en.isChecked()) {
                //o in inglese
                builder.setTitle("Attention")
                        .setMessage("Before you select a category you have
                                selected at least one Type")
                        .setPositiveButton("OK",
                                new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                                int id)
                                {
                                        dialog.cancel();
                                }
                        });
        }
```

Per definire il comportamento dell'applicazione quando si seleziona una Categoria si utilizza il metodo `onChangeSelectedCats`, il quale oltre a scrivere sul campo di ricerca delle Categorie quelle selezionate dal menu apposito ha anche il compito di caricare le Tipologie di servizio disponibili a seconda delle Categorie selezionate dal menu a scelta multipla.

```
        // per ogni categoria selezioanta
        for (int x = 0; x < selectedCat.size(); x++) {
                if (selectedCat.size() == 1 || (selectedCat.size() - x == 1))
                {
                        stringBuilder.append(selectedCat.get(x));
                //carico i dati per le tipologie di categoria selezionate
                        if (selectedCat.get(x).equals("Medico")) {
                                ris1.add("Generico");
                                ris1.add("Pediatra");
                                ris1.add("Odontoiatra");
                                ris1.add("Specialista");
                                ris1.add("Paramedico");
```

```
                }
                else if (selectedCat.get(x).equals("Luoghi - Sanitˆ"))
                {
                        ris1.add("Farmacia");
                        ris1.add("Ospedale");
                        ris1.add("Ambulatorio");
                        ris1.add("Guardia Medica");
                        ris1.add("Misericordia");
                        ris1.add("Assistenza");
                }
                ...
                //utilizzato poi per il menu di tipologie di servizi
                TipCat = new CharSequence[ris1.size()];
                for (int i = 0; i < ris1.size(); i++) {
                        TipCat[i] = ris1.get(i);

                }
```

Infine abbiamo i due metodi `showSelectTipCatDialog` e `onChangeSelectedTipCat` utilizzati per la creazione ed il controllo del menu di scelta delle Tipologie di Servizi. Il primo metodo serve per creare tale menu, controllando però che sia già avvenuta una scelta tra le Categorie disponibili altrimenti le Tipologie di Servizio non sarebbero disponibili.

```
        if (selectedCat.size() > 0) {
                boolean[] checkedTipCat = new boolean[TipCat.length];
                int count = TipCat.length;
                for (int i = 0; i < count; i++) {
                        checkedTipCat[i] = selectedTipCat.contains(TipCat[i]);
                }
                DialogInterface.OnMultiChoiceClickListener
    tipCatDialogListener = new DialogInterface.OnMultiChoiceClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which,
                                        boolean isChecked) {
                                if (isChecked)
                                        selectedTipCat.add(TipCat[which]);
                                else
                                        selectedTipCat.remove(TipCat[which]);
                                onChangeSelectedTipCat();
```

```
                }
            };
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            if (it.isChecked())
                builder.setTitle("Tipologia di Servizi");
            else if (en.isChecked())
                builder.setTitle("Type of Services");
            builder.setMultiChoiceItems(TipCat, checkedTipCat,
                    tipCatDialogListener);
            AlertDialog dialog = builder.create();
            dialog.show();
```

Nel caso non sia stata scelta alcuna Categoria l'applicazione avvisa l'utente di effettuare la scelta tramite una finestra di allerta

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
      if (it.isChecked()) {
            builder.setTitle("Attenzione")
                  .setMessage("Prima di selezionare una Tipologia di
                  Servizio devi aver scelto almeno una Categoria")
                  .setPositiveButton("OK",
                        new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog,
                                                      int id)
                              {
                                    dialog.cancel();
                              }
                  });
      }
```

Il metodo `onChangeSelectedTipCat` scritto in fondo alla classe serve a popolare il campo di ricerca delle Tipologie di Servizi con i valori scelti dal menu a scelta multipla del parametro definito dal metodo illustrato sopra.

```
//tipologie di servizio selezionate nel campo di ricerca
      public void onChangeSelectedTipCat() {
            StringBuilder stringBuilder = new StringBuilder();
            for (int x = 0; x < selectedTipCat.size(); x++) {
                  if (selectedTipCat.size() == 1 || (selectedTipCat.size() - x
```

```
                    == 1))
                    stringBuilder.append(selectedTipCat.get(x));
              else
                    //se più di una separate da virgola
                    stringBuilder.append(selectedTipCat.get(x) + ",");
        }
        tip_servizi.setText(stringBuilder.toString());
  }
```

3.5 Descrizione della classe ViewImage

Questa classe viene richiamata quando si vuole visualizzare un'immagine della descrizione del luogo scelto ingrandita rispetto a quelle visualizzate sulla descrizione.
In questa Activity viene innanzitutto richiamato il layout imageview dove sono definite le dimensioni per avere le immagini ingrandite in modo tale che occupino la maggior parte dello schermo del dispositivo senza venire sgranate troppo, poi viene richiamato il metodo `download` della classe `ImageDownloader` definita nel file contenente anche l'Activity `Descrizione` che consente di scaricare da internet l'immagine richiesta e visualizzarla secondo le direttive del layout caricato in precedenza. Inoltre viene impostato un messaggio di attesa visualizzato finché il download dell'immagine non è completo. Tale messaggio viene visualizzato grazie all'utilizzo di un Toast come quelli già utilizzati nelle classi Descrizione e ViewPoint.

```
// richiamo il layout
setContentView(R.layout.imageview);
img = (ImageView) findViewById(R.id.image);
ImageDownloader i = new ImageDownloader();
//messaggio durante il caricamento
Toast.makeText(this, "Caricamento immagine...", 1).show();
i.download(getIntent().getStringExtra("img"), img);
```

3.6 Descrizione della classe ViewPoint

L'Activity definita in questa classe ha il compito di richiedere i luoghi al server online e di disegnare sopra la mappa di Google il lucido contenente i risultati della richiesta. Inoltre gestisce anche le operazioni da eseguire in caso venga premuta l'icona di un luogo aprendo così la pagina di

descrizione.

Innanzi tutto vengono dichiarate tutte le variabili che serviranno al corretto funzionamento della classe, con particolare attenzione per quelle che serviranno a disegnare sulla mappa i luoghi trovati.

```
MapView mapView;
MyLocationOverlay myLocationOverlay;
WMSOverlay wMSOverlay;
GeoPoint punto = new GeoPoint(41885223, 12399086);
GeoPoint puntoScelto = punto;
Rect canvasRect;
```

Poi si passa al metodo `onCreate` lanciato alla creazione dell'Activity il quale legge tutti i dati passatigli dalla pagina di ricerca, utilizzati poi per creare il filtro per la richiesta dei luoghi da ricercare, carica la mappa di sfondo e disegna sopra di essa il lucido con i luoghi trovati.

```
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.main);
	//set variabili
	nome = getIntent().getStringExtra("nome");
	serv = getIntent().getStringExtra("serv");
	tip_serv = getIntent().getStringExtra("tip_serv");
	tip = getIntent().getStringExtra("tip");
	cat = getIntent().getStringExtra("cat");
	gg = getIntent().getStringExtra("gg");
	comuni = getIntent().getStringExtra("comuni");
	//inizializzazione mappa
	mapView = (MapView) findViewById(R.id.mapview);
```

Inoltre vengono anche impostati i funzionamenti dei pulsanti di zoom, infatti il layout utilizzato è lo stesso della schermata principale dell'applicazione. Quando viene premuto uno di tali pulsanti bisogna ridisegnare i lucido in quanto la mappa sottostante è cambiata, infatti risulta ingrandita o rimpicciolita a seconda di che pulsante di zoom si è premuto

```
//zoom della mappa
ZoomControls zoomControls = (ZoomControls)findViewById(R.id.zoomcontrols);
zoomControls.setOnZoomInClickListener(new View.OnClickListener() {
	@Override
```

```
        public void onClick(View v) {
              mapView.getController().zoomIn();
              //ridisegna i luoghi
              wMSOverlay.draw(wMSOverlay.c, mapView, false);
              //messaggio di caricamento
              Toast.makeText(ViewPoint.this, "Caricamento POI...",
                                    Toast.LENGTH_SHORT).show();
        }
  });
  zoomControls.setOnZoomOutClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
              mapView.getController().zoomOut();
              //ridisegna i luoghi
              wMSOverlay.draw(wMSOverlay.c, mapView, false);
              //messaggio di caricamento
              Toast.makeText(ViewPoint.this, "Caricamento POI...",
                          Toast.LENGTH_SHORT).show();
        }
  });
```

Seguono poi le istruzioni per settare la corretta visualizzazione della mappa e per far si che si posizioni sul luogo individuato dal sistema di localizzazione del dispositivo.

```
        mapView.getController().setZoom(6);
        mapView.setBuiltInZoomControls(true);
        mapView.getController().setCenter(punto);
        myLocationOverlay = new MyLocationOverlay(this, mapView);
        GeoPoint g = new GeoPoint(getIntent().getIntExtra("lat", 0),
                                        getIntent().getIntExtra("long", 0));
        mapView.getController().setZoom(getIntent().getIntExtra("zoom", 6));
        mapView.getController().animateTo(g);
        mapView.getOverlays().add(myLocationOverlay);
```

Si passa poi a disegnare il lucido con i luoghi sopra la mappa

```
        wMSOverlay = new WMSOverlay();
        mapView.getOverlays().add(wMSOverlay);
```

ed ad impostare il funzionamento del pulsante di ricerca, il quale, se premuto, deve passare all'Activity e quindi alla pagina di ricerca dei luoghi definita nella classe RicercaPOI con tutti i valori necessari al suo funzionamento.

```
cerca = (ImageButton) findViewById(R.id.cerca);
cerca.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View arg0) {
		Intent i = new Intent(ViewPoint.this, RicercaPoi.class);
		i.putExtra("zoom", mapView.getZoomLevel());
		i.putExtra("long",mapView.getMapCenter().getLongitudeE6());
		i.putExtra("lat", mapView.getMapCenter().getLatitudeE6());
		i.putExtra("value", getIntent().getStringExtra("value"));
		startActivity(i);
		finish();
	}
});
```

Seguono poi le definizioni dei metodi lanciati nel caso in cui l'applicazione venga messa in pausa o riprenda il suo funzionamento, abilitando o disabilitando la localizzazione del dispositivo.

```
@Override
protected void onResume() {
	super.onResume();
	myLocationOverlay.enableMyLocation();
}
@Override
protected void onStop() {
	super.onStop();
	myLocationOverlay.disableMyLocation();
}
```

Inoltre definiamo poi anche i metodi che si occupano di salvare e ripristinare le variabili dell'applicazione in caso venga interrotta bruscamente

```
//salvo variabili per ripristino applicazione
public void onSaveInstanceState(Bundle outState) {
	outState.putString("nome", nome);
	outState.putString("servizi", serv);
```

```
        outState.putString("tipo_servizi", tip_serv);
        outState.putString("tip", tip);
        outState.putString("cat", cat);
        outState.putString("gg", gg);
        outState.putString("comuni", comuni);
        super.onSaveInstanceState(outState);
    }

    @Override
    //ripristino variabili dell'applicazione
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        nome = savedInstanceState.getString("nome");
        tip = savedInstanceState.getString("tip");
        serv = savedInstanceState.getString("servizi");
        tip_serv = savedInstanceState.getString("tipo_servizi");
        cat = savedInstanceState.getString("cat");
        gg = savedInstanceState.getString("gg");
        comuni = savedInstanceState.getString("comuni");
    }
```

Si passa poi a descrivere la classe utilizzata per richiedere i dati al server e disegnare il lucido sopra la mappa visualizzata dall'applicazione, chiamata `WMSOverlay`. Per prima cosa dichiariamo le variabili utili che ci serviranno.

```
        Canvas c;
        GeoPoint altoSinistra = new GeoPoint(0, 0);
        GeoPoint bassoDestra = new GeoPoint(0, 0);
        Paint paint = new Paint();
        Bitmap overlay;
        boolean scaricaMappa = false;
        String cql;
```

Il costruttore di tale classe definisce poi la tipologia di colori da utilizzare e lo stile di disegno adottato dall'applicazione

```
public WMSOverlay() {
      paint.setAlpha(128);
      paint.setStyle(Paint.Style.STROKE);
}
```

Definiamo successivamente un metodo utilizzato per disegnare un rettangolo che delimita la zona dentro la quale vengono caricate, e quindi disegnate, le icone che rappresentano i luoghi ritrovati dalla ricerca. Per far ciò bisogna passare al metodo le coordinate dei vertici in alto a sinistra ed in basso a destra del rettangolo.

```
//per disegnare rettangolo su zona di interesse da caricare
private Rect geoRect(GeoPoint altoSinistra, GeoPoint bassoDestra) {
      Point p1 = mapView.getProjection().toPixels(altoSinistra,null);
      Point p2 = mapView.getProjection().toPixels(bassoDestra, null);
      Rect rect = new Rect(p1.x, p1.y, p2.x, p2.y);
      return rect;
}
```

Subito dopo viene definito il metodo che consente all'applicazione di disegnare sopra la mappa il lucido contenente tutti i luoghi ottenuti dalla ricerca. Per prima cosa tale metodo verifica che l'applicazione sia lanciata come Activity principale, altrimenti non procede nemmeno al reperimento e disegno dei luoghi trovati.

```
public void draw(Canvas c1, MapView mapView, boolean shadow) {
            this.c = c1;
            if (shadow)
                  return;
```

Si passa poi a creare un rettangolo che comprenda tutte le coordinate della mappa visualizzata, infatti se la porzione di mappa visualizzata dallo schermo non comprende le coordinate del lucido sul quale sono indicate le posizioni dei luoghi trovati, l'applicazione non le disegna perchè non ce ne sarebbero nelle vicinanze. Inoltre viene utilizzato un parametro per indicare se la mappa col lucido è già stata scaricata dal server per non doverla riscaricare ancora inutilmente.

```
canvasRect = new Rect(0, 0, c.getWidth(), c.getHeight());
            Rect rect = geoRect(altoSinistra, bassoDestra);
            if ((!rect.contains((canvasRect.centerX() + 100),
                        (canvasRect.centerY() + 100)) || !rect.contains(
                        (canvasRect.centerX() - 100),
                        (canvasRect.centerY() - 100)))
                        && !scaricaMappa)
            {
```

Si passa poi alla creazione della stringa della query che indicherà il filtro per i dati da richiedere al server. Dato che si accederà ai dati del server tramite una richiesta WMS in modalità GET, tutti i valori dei campi di ricerca vengono riscritti secondo la sintassi utilizzata per gli indirizzi url dei siti web, ad esempio il carattere “spazio” viene codificato con %20, il carattere “à” viene codificato con %E0 e così via per tutti i caratteri speciali per ogni campo di ricerca possibile fra quelli proposti nella schermata di ricerca.

```
      //filtro per info richieste
      cql = "&cql_Filter=";
      //filtro nome
      if (!getIntent().getStringExtra("nome").equals("")) {
            if (!cql.equals("&cql_Filter="))
                  cql += "%20AND%20";
            String [] sp =getIntent().getStringExtra("nome").split(" ");
            String ris = "";
            for (int i = 0; i < sp.length-1; i++) {
                  ris += sp[i]+"%20";
            }
            ris += sp[sp.length-1];
            cql += "poi_name%20like%20'%25"+ris+"%25'";
      }
      //filtro tipologie
      if (!getIntent().getStringExtra("tip").equals("")) {
            if (!cql.equals("&cql_Filter="))
                  cql += "%20AND%20";
            String [] sp = getIntent().getStringExtra("tip").split(",");
            for (int i = 0; i < sp.length-1; i++) {
                  if (sp[i].equals("Sanità"))
                        cql += "typology='Sanit%E0'%20OR%20";
                  else
```

```
                cql += "typology='"+sp[i]+"'%20OR%20";
            }
            if (sp[sp.length-1].equals("Sanità"))
                cql += "typology='Sanit%E0'";
            else
                cql += "typology='"+sp[sp.length-1]+"'";
            cql.replace(" ", "%20");
        }
```

Infine viene scaricata la mappa lucido grazie ad un altro metodo definito poi sempre nella classe `WMSOverlay` e viene visualizzato inoltre un messaggio di attesa finché tale operazione di download non viene ultimata.

```
downloadMappa(mapView.getProjection(), canvasRect,cql);
Toast.makeText(ViewPoint.this, "Caricamento POI...",
                Toast.LENGTH_SHORT).show();
```

Come ultima cosa viene controllato che il lucido, contenuto nella variabile `overlay` e ricavato dal metodo `downloadMappa` contenga dei dati da disegnare e, in caso affermativo, tali dati vengono piazzati sopra la mappa grazie al rettangolo che delimita la zona contenente dati.

```
        if (overlay != null)
            c.drawBitmap(overlay, canvasRect, rect, paint);
        c.drawRect(rect, paint);
```

Vengono definiti poi altri quattro metodi per convertire il valore selezionato dei parametri di ricerca dei Giorni di apertura, Categorie, Comuni e Tipologie di Servizi in valori sintatticamente corretti per poter essere accettati nella url di richiesta al server. Tali metodi sono `checkDay`, `checkCat`, `checkComuni` e `checkTipCat`. Ne vediamo un paio qui di seguito.

```
//conversione nome dei giorni
public String checkDay (String day) {
    String ris= "";
    if (day.equals("Luned“"))
        ris = "Luned%EC";
    else if (day.equals("Marted“"))
        ris = "Marted%EC";
```

```
        else if (day.equals("Mercoled“"))
              ris = "Mercoled%EC";
        else if (day.equals("Gioved“"))
              ris = "Gioved%EC";
        else if (day.equals("Venerd“"))
              ris = "Venerd%EC";
        else if (day.equals("Sabato"))
              ris = "Sabato";
        else if (day.equals("Domenica"))
              ris = "Domenica";
        return ris;
  }
  //conversione nome delle categorie
  public String checkCat (String cat) {
        String ris= "";
        if (cat.equals("Forze dell'ordine"))
              ris = "Forze%20dell%25";
        else if (cat.equals("In viaggio"))
              ris = "In%20viaggio";
        else if (cat.equals("Punti di Interesse"))
              ris = "Punti%20di%20Interesse";
        else if (cat.equals("Servizi - Interesse") ||
                    cat.equals("Servizi - Sicurezza"))
              ris = "Servizi";
        else if (cat.equals("Luoghi - Sanità") ||
                    cat.equals("Servizi - Interesse"))
              ris = "Luoghi";
        else
              ris = cat;
        return ris;
  }
```

Dopo ciò abbiamo la definizione del metodo che gestisce le azioni dell'applicazione quando un'icona di un luogo viene toccata. Quando accade ciò, l'applicazione provvede a richiamare il metodo `downloadDesc` che ha il compito di scaricare dal server tutte le informazioni di dettaglio riguardanti il luogo selezionato. A tale funzione vengono anche passate le coordinate del punto toccato affinché la pagina di dettaglio si apra anche se premo in un intorno della posizione effettiva del luogo.

```
//metodo richiamo info se premo icona
public boolean onTap(GeoPoint pkt, MapView mv) {
      puntoScelto = pkt;
      Projection projection = mapView.getProjection();
      Point point = new Point();
      projection.toPixels(pkt, point);
      downloadDesc(mapView.getProjection(),canvasRect,point,pkt,cql);
```

Come ultimi metodi della classe vengono definiti `downloadMappa` e `downloadDesc` che hanno il compito di collegarsi e reperire le informazioni dal server web. Il primo dei due è `downloadMappa` che, per prima cosa, si preoccupa di inserire tutte le istruzioni da eseguire al suo interno in un `AsyncTask`, cioè in un task asincrono. Facendo ciò si permette all'applicazione Android di svolgere le sue funzionalità anche durante il periodo di tempo in cui il dispositivo sta scaricando la mappa del lucido dal server, senza bloccare il funzionamento di tutta l'applicazione per aspettare i risultati ottenuti dal server.

```
      new AsyncTask() {
                  GeoPoint geo1 = projection.fromPixels(0, 0);
                  GeoPoint geo2 =projection.fromPixels(canvasRect.width(),
                              canvasRect.height());
                  @SuppressWarnings("unused")
                  Rect newRect = geoRect(geo1, geo2);
                  String CQL = cql;
```

Le prime istruzioni inoltre servono per confinare l'area attualmente visualizzata dalla schermata del dispositivo con un rettangolo dentro al quale verranno disegnate le icone dei luoghi se presenti.
Subito dopo viene definito il metodo `doInBackground` che consente alle istruzioni scritte al suo interno di essere svolte in background mentre l'applicazione continua ad essere eseguita regolarmente . In questo blocco di istruzioni viene, per prima cosa, costruito l'url per poter effettuare la richiesta al server. Tale url è formato inizialmente dall'indirizzo del server, seguito poi da alcuni parametri che specificano il tipo di chiamata WMS eseguita via GET, seguiti poi dalle coordinate dei punti in alto a sinistra e in basso a destra dello schermo che identificano la porzione di mappa sulla quale andremo a disegnare il lucido. Tali coordinate sono poi seguite dalla larghezza e dalla lunghezza del rettangolo dove andiamo a disegnare le icone contenute nel lucido, che ricopre peraltro tutta l'estensione dello schermo del dispositivo. Viene indicato poi il tipo di immagine che il server restituisce, nel nostro caso il lucido con segnati i luoghi viene restituito a noi sotto forma di

immagine png ed infine viene accodata la query utilizzata come filtro per i dati da reperire dal server.

```
protected Object doInBackground(Object... arg0) {
      try {
            String url ="http://imaa.geosdi.org/geoserver/wms?
                  service=WMS&version=1.1.0&request=GetMap&
                  layers=base:geoplatform_poi&styles=&bbox="
                  + (COSTANTE * geo1.getLongitudeE6())+ ","
                  + (COSTANTE * geo2.getLatitudeE6())+ ","
                  + (COSTANTE * geo2.getLongitudeE6())+ ","
                  + (COSTANTE * geo1.getLatitudeE6()+"&width="
                  + canvasRect.width()+ "&height="
                  + canvasRect.height()+
                  "&srs=EPSG:4326&format=image/png&TRANSPARENT=true"+CQL;
```

Vengono poi scritte le istruzioni per scaricare la mappa richiamando l'url appena creato

```
InputStream input = new URL(url).openStream();
```

e viene codificata l'immagine png ottenuta in risposta dal server per poter essere effetivamente disegnata nel canvas applicato poi sopra la mappa visualizzata dall'applicazione

```
overlay = BitmapFactory.decodeStream(input);
```

Dato che tutte queste istruzioni sono contenute all'interno di una classe `AsyncTask` creata al volo,bisogna impostare il comando `.execute()` di tale task affinché le istruzioni scritte all'interno vengano realmente eseguite.
Il metodo `downloadDesc` presenta le stesse funzionalità e struttura del metodo `downloadMappa` appena descritto, anche se differisce da esso per alcune caratteristiche. Infatti tale metodo costruisce l'url di richiesta al server in modo tale che dia come risposta una pagina web contente tutte le informazioni riguardanti i dettagli su un luogo selezionato e non un'immagine da disegnare poi sopra la mappa. Tali informazioni devono poi essere passate all'Activity Descrizione che farà in modo di leggerle e visualizzarle per l'utente che le ha richieste premendo sopra l'icona di un luogo trovato. Questo metodo predispone quindi, dentro un altro `AsyncTask`, le variabili utili alla ricezione di dati tramite una pagina web come `HttpClient`, dove verrà salvata la pagina web

restituita dal codice, e `HttpGet` che consente di eseguire la richiesta al server tramite l'url appositamente creato.

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet();
request.setURI(new   URI("http://imaa.geosdi.org/geoserver/wms?
            bbox="+ (COSTANTE * geo1.getLongitudeE6())+ ","
            + (COSTANTE * geo2.getLatitudeE6())+ ","
            + (COSTANTE * geo2.getLongitudeE6())+ ","
            + (COSTANTE * geo1.getLatitudeE6())+
            "&styles=&format=jpeg&info_format=text/plain&
            request=GetFeatureInfo&layers=base:geoplatform_poi
            &query_layers=base:geoplatform_poi&width="
            + canvasRect.width() + "&height=
            + canvasRect.height() + "&x="+(punto.x)
            +"&y="+(punto.y)+CQL));
```

Anche questo url di richiesta è composto innanzi tutto dalla prima parte che indica il percorso del server web da contattare, seguito poi dalle coordinate in alto a sinistra e in basso a destra della mappa visualizzata attualmente sullo schermo del dispositivo. Dopo queste però sono impostati i parametri della richiesta che consentono di avere come risposta una pagina web di testo e non un'immagine di tipo png con disegnati i luoghi ottenuti, inoltre la parte `request=GetFeatureInfo` indica al server di restituire le informazioni di un luogo anche se non vengono passate via url le corrette coordinate di tale punto ma anche alcune appartenenti ad un suo intorno. Dopo questi parametri vengono passate le informazioni di larghezza e lunghezza del canvas dove si può disegnare sopra la mappa visualizzata dallo schermo anche se in questo caso poi non verrà disegnato nulla sopra di essa, infine vengono passate le coordinate premute sullo schermo che dovrebbero rappresentare quelle esatte di dove si trova il punto, ma grazie allo specifico parametro definito prima vengono accettate anche coordinate nelle vicinanze del luogo scelto e come ultimo parametro viene passata la query utilizzata per trovare i luoghi fra i quali si è operata la scelta e richiesta di dettaglio per facilitare l'operazione di ricerca delle informazioni al server. Infatti potrebbe capitare che, premendo su di un punto della mappa vicino ad un luogo selezionato, vi sia situato esattamente un luogo le cui informazioni sono presenti nel server, ma non soddisfa i criteri di ricerca impostati dall'utente. L'aggiunta della query di filtro alla fine della url di richiesta consente di escludere a priori tale punto in quanto non compare come risultato dopo aver applicato tale filtro ai dati del database presenti nel server.

Dopo aver creato l'url per la richiesta delle informazioni al server bisogna farla eseguire dall'applicazione e salvare il risultato in una variabile di tipo `HttpResponse` grazie alla quale possiamo poi leggere ciò che contiene la pagina web restituita tramite un `BufferedReader` e suddividere quindi i dati contenuti in tale pagina conoscendo il tipo di separatore che divide i dati tra di loro.

```
HttpResponse response = client.execute(request);
in = new BufferedReader(newInputStreamReader(
                        response.getEntity().getContent()), 8 * 1024);
StringBuffer sb = new StringBuffer("");
String line = "";
String NL = System.getProperty("line.separator");
while ((line = in.readLine()) != null) {
      sb.append(line + NL);
}
in.close();
page = sb.toString();
```

Infine, se la pagina web contiene delle informazioni di dettaglio, bisogna passarle all'Activity di Descrizione per far si che vengano visualizzate.

```
if (!page.equals("no features were found" + NL)) {
      Intent i = new Intent(ViewPoint.this,
      Descrizione.class);
      i.putExtra("dati", page);
      i.putExtra("value", getIntent().getStringExtra("value"));
      i.putExtra("pointLatitude", geopoint.getLatitudeE6());
      i.putExtra("pointLongitude", geopoint.getLongitudeE6());
      i.putExtra("myLatitude",
                  myLocationOverlay.getMyLocation().getLatitudeE6());
      i.putExtra("myLongitude",
            myLocationOverlay.getMyLocation().getLongitudeE6());
      startActivity(i);
}
```

Anche per questo metodo, dato che tutte le istruzioni sono definite all'interno di una classe `AsyncTask` creata al volo, per essere eseguite c'è bisogno di lanciare il comando `.execute()` di tale classe.

3.7 Layout

Possiamo vedere qui di seguito alcune immagini di come appare l'applicazione durante la sua funzione. Si parte innanzi tutto dalla schermata di main che contiene solo la mappa di Google, i pulsanti di zoom ed il tasto per attivare la ricerca dei luoghi.

Premendo sul pulsante Cerca si passa alla visualizzazione della pagina di ricerca luoghi dell'applicazione gestita dalla classe RicercaPOI. Tale pagina è disponibile in due lingue, italiano

oppure in inglese

mentre i menu a scelta multipla richiamati nei campi Tipologia di Categoria, Categoria, Tipologia di Servizio e Comuni risultano avere un aspetto simile

Una volta premuto sul tasto di ricerca, poi, si passa alla visualizzazione della mappa con disegnato sopra il lucido contenente le icone dei luoghi trovati grazie al server web

e se viene premuta una di tale icone, si passa alla visualizzazione della pagina di Descrizione di tale luogo. Anche questa pagina dell'applicazione è disponibile in due lingue, italiano

oppure inglese.

Infine, se premiamo sul pulsante di indicazioni, viene mostrata una scelta su come ottenere le indicazioni stradali, se tramite internet oppure grazie ad un navigatore installato nel dispositivo Android.

Capitolo 4: Conclusioni

4.1 Conclusioni

L'applicazione per dispositivi Android realizzata è risultata molto soddisfacente, in quanto riesce ad adempire a tutte le esigenze segnalate ed individuate dal committente all'inizio dei lavori. Particolare apprezzamento ha riscosso l'opzione di recupero delle indicazioni stradali a partire dalla posizione del dispositivo fino al punto di interesse selezionato, grazie soprattutto alla possibilità di scegliere se ottenere tali indicazioni sfruttando le funzionalità delle mappe di Google direttamente da internet oppure utilizzando un navigatore già installato all'interno del dispositivo. Rimane il rammarico che, per una mancanza tecnica nel server web di appoggio, la funzione di traduzione in lingua inglese non risulti completa, lasciando scritte in italiano le informazioni riguardanti un luogo selezionato anche se è stata impostata la visualizzazione in lingua inglese. Tale mancanza però non compromette il corretto funzionamento dell'applicazione né la soddisfazione generale per il prodotto finale ottenuto.