UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria Enzo Ferrari

Corso di Laurea Triennale in

Ingegneria Informatica

**VIOLA: un'applicazione ANDROID che rivoluziona lo shopping**

Relatore: Tesi di Laurea di:

Prof.ssa Sonia Bergamaschi Marco Orlando

Anno Accademico 2013-2014

**Indice**

[Introduzione 4](#_Toc392486985)

[Capitolo 1 – Panoramica generale applicazione Android Viola 6](#_Toc392486986)

[Capitolo 2 – Tecnologie utilizzate 8](#_Toc392486987)

[2.1 Ambiente di sviluppo 8](#_Toc392486988)

[2.2 Requisiti di sistema 9](#_Toc392486989)

[2.3 Linguaggi e formati 10](#_Toc392486990)

[Capitolo 3 – Progetto e implementazione 11](#_Toc392486991)

[3.1 Progettazione schema Entità/Relazioni 11](#_Toc392486992)

[3.2 Progettazione logica relazionale 12](#_Toc392486993)

[3.3 Class Diagram 14](#_Toc392486994)

[3.4 Implementazione 15](#_Toc392486995)

[Capitolo 4 – Descrizione generale 17](#_Toc392486996)

[4.1 Login 17](#_Toc392486997)

[4.2 Ricerca sul database 19](#_Toc392486998)

[4.3 Il menù 22](#_Toc392486999)

[4.4 Mostra mappe 24](#_Toc392487000)

[4.5 Mostra vetrine 26](#_Toc392487001)

[4.6 Mostra offerte della città 28](#_Toc392487002)

[4.7 Mostra preferiti 29](#_Toc392487003)

[4.8 Codice 29](#_Toc392487004)

[4.9 Note conclusive sulle scelte implementative 30](#_Toc392487005)

[Conclusione 32](#_Toc392487006)

[Appendice [1] - Consigli per i principianti su Android 33](#_Toc392487007)

[Appendice [2] - Funzionalità delle classi 33](#_Toc392487009)

[Appendice [3] - Risposte del server per login 36](#_Toc392487015)

[Appendice [4] - Funzione crittografia password 37](#_Toc392487017)

[Bibliografia 38](#_Toc392487019)

[Sitografia 38](#_Toc392487020)

[Ringraziamenti 40](#_Toc392487021)

Introduzione

L’obiettivo di questo elaborato è la creazione di un’applicazione dedicata allo shopping, chiamata Viola come il colore dell’unione e realizzata su piattaforma ANDROID.

L’applicazione Viola ha come scopo principale quello di dare un supporto ai negozi di quartiere che spesso non sono presenti né *online* né tantomeno sul *mobile*. Creando delle reti sociali tra negozi e clienti, si offre ai consumatori il massimo del *comfort* in relazione a tutte le fasi d’acquisto. La miglior promozione dei negozi con marchi “*made in italy*” potrà inoltre produrre effetti positivi sul turismo, con gli stessi negozi che diventano delle piccole attrazioni turistiche.

Viola combina i vantaggi dell’*e-commerce* e le comodità del *traditional commerce*. Per illustrarne brevemente il funzionamento, può essere opportuno paragonarla ad una piattaforma *web* dove cercare prodotti e servizi, che verranno acquistati direttamente nei negozi vicino casa.

I consumatori avranno:

* un’ampia scelta di prodotti e servizi, come quando effettuano acquisti su internet;
* la possibilità di evitare gli inconvenienti che spesso caratterizzano gli acquisti dal *web*, principalmente: lunghi tempi di attesa, elevati costi di spedizione, descrizioni dei prodotti ingannevoli.

I venditori usufruiranno:

* di una vetrina sul *web* dei loro prodotti e servizi;
* della possibilità di instaurare con i clienti relazioni personali, al fine di ottenere una maggior fedeltà ed evitare di fare affidamento esclusivamente su un prezzo basso.

Viola può essere dunque paragonata ad Amazon e Ebay, ma sono proprio le differenze rispetto a questi *competitor* ad evidenziarne punti di forza ed elementi distintivi. Per la capacità di aumentare la visibilità dei negozi (specialmente quelli più piccoli) sul *mobile* e creare reti sociali per lo shopping, fornisce un servizio simile a Facebook.

Con questa tesi, si mostra il progetto in base al quale è stata sviluppata l’applicazione e se ne mostra il funzionamento. Si sono analizzate le tecnologie utilizzate e le motivazioni che hanno portato alle loro scelte. In particolare, si è descritto lo sviluppo del *database* e le implementazioni più importanti.

Da un punto di vista tecnico lo sviluppo dell’applicazione ha richiesto lo studio di programmazione lato *server* e lato *client*.

L’elaborato è organizzato come segue. Nel primo capitolo si evidenzia l’idea alla base dell’applicazione. Il secondo capitolo introduce le specifiche tecniche su cui è basato il progetto di tesi. Nel terzo capitolo vengono presentati i dettagli del progetto e dell’implementazione. Il quarto capitolo illustra le funzionalità principali dell’applicazione.

Capitolo 1 – Panoramica generale applicazione Android Viola

Per illustrare più approfonditamente le funzioni di Viola, può risultare utile, innanzitutto, distinguere gli utilizzatori dell’applicazione in due macrocategorie:

* gli utenti che usufruiscono dei contenuti ovvero gli acquirenti di prodotti e servizi (clienti);
* gli utenti che forniscono contenuti ossia le imprese con la propria vetrina *online*.

I clienti potranno:

* risparmiare tempo (trovando i prodotti nei negozi più vicini, accedendo alle informazioni relative all’apertura/chiusura dei negozi in tempo reale);
* curiosare, in quanto i negozi reali pubblicheranno: foto del locale, di eventi, del personale (l’applicazione ha quindi questo aspetto simile ai *social networ*k, che può contribuirne alla pubblicità e diffusione);
* cercare informazioni sui prodotti (i venditori potranno inserire un *link* diretto al proprio sito o a quello del produttore con la descrizione del prodotto o servizio offerto);
* essere aggiornati tempestivamente su promozioni, nuovi arrivi e qualunque comunicazione da parte del venditore.

Non potranno invece confrontare i prezzi: questi non saranno presenti, in quanto lo scopo dell’applicazione non è una competizione su prezzi al ribasso.

Le imprese disporranno:

* della presenza del loro negozio su dispositivi *mobile* (il *business* è sempre più *mobile* e Viola offre l’opportunità anche ai piccoli esercenti di usufruire di tale modalità di accesso al mercato);
* di una lista clienti: grazie alla quale si avranno i riferimenti per poter contattare i propri *followers* aumentando così la fidelizzazione del cliente (ad esempio cartoline di auguri per festività e compleanni);
* della possibilità di gestire le proprie informazioni in modo diretto e in *real time*: tramite una bacheca virtuale si potranno modificare orari di apertura, informare di aperture/chiusure straordinarie, avvisare i clienti dell’arrivo dei nuovi prodotti, annunciare nuove promozioni.

I venditori non possono invece:

* vendere: lo scopo dell’applicazione è quello di generare potenziali clienti e fidelizzare maggiormente i clienti acquisiti. L’obiettivo non è quindi raggiungere clienti in tutto il mondo (ci sono modalità migliori per questo) ma focalizzarsi sui clienti vicini al negozio;
* usufruire di una chat privata dell’applicazione: le offerte saranno pubblicate solo nella propria bacheca e pubblicizzate con i tradizionali canali di comunicazione (telefono, email, corrispondenza).

Capitolo 2 – Tecnologie utilizzate

2.1 Ambiente di sviluppo

L’applicazione è nativa Android[[1]](#footnote-1), per dispositivi con almeno Android 3.1 Honeycomb (API level 12)[[2]](#footnote-2), sia *smartphone* che *tablet* (grafica orientata alle dimensioni variabili dello schermo). Testata e funzionante su Android 4.1 Jelly Bean (API level 16).

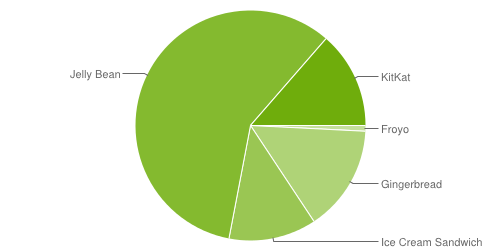
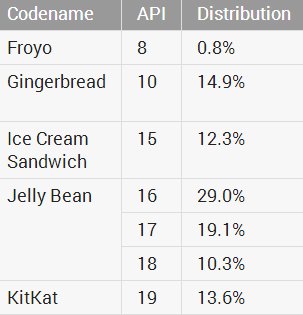


Figura 1: diffusione versioni Android

Fonte: https://developer.android.com/about/dashboards/index.html

Sebbene i contenuti sono gli stessi per *smartphone* e per *tablet*, nella realizzazione dell’app si è tenuto conto delle differenze fra le due tipologie di dispositivi ricercando un compromesso tra:

* un’esperienza dell’utente veloce, generalmente in piccole pause (*smartphone*);
* un’esperienza caratterizzata da un tempo maggiore, con l’utente disposto a visualizzare contenuti più ricchi (*tablet*).

Si è cercato quindi di creare *layout* semplici, intuitivi e facili da esplorare, eliminando contenuti superflui.

*Il design non è solo aspetto e apparenza esteriore.*

*Il design è anche il modo di funzionare.*

*Steve Jobs*

2.2 Requisiti di sistema

Per sviluppare l’applicazione è stato utilizzato un sistema Windows, con i seguenti *software*:

* **Java Development Kit (JDK)[[3]](#footnote-3)**
  + Una serie di strumenti (compilatore, interprete) con cui si possono creare applicazioni Java;
* **Eclipse[[4]](#footnote-4)** 
  + È l’IDE consigliato per lo sviluppo Android;
* **Android SDK[[5]](#footnote-5)**
  + Una serie di strumenti necessari per sviluppare, testare ed eseguire il *debug* delle applicazioni Android;
* **Android Development Tools (ADT)[[6]](#footnote-6)** Plugin per Eclipse
  + Il quale consente di utilizzare gli strumenti dell’SDK nell’IDE Eclipse;
* **Photoshop[[7]](#footnote-7)**
  + Programma grafico per creare immagini, icone, pulsanti, loghi;
* **Komodo[[8]](#footnote-8)**
  + Editor di testo che riconosce i linguaggi di programmazione;
* **Wamp[[9]](#footnote-9)**
  + piattaforma di sviluppo per applicazioni web, basata su: Windows Apache MySQL Php (come da acronimo).

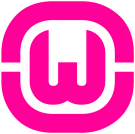


Figura 2: Icone *software* utilizzati

2.3 Linguaggi e formati

* **Java [[10]](#footnote-10)**
  + linguaggio utilizzato per creare l’applicazione;
* **Php [[11]](#footnote-11)**
  + linguaggio utilizzato per elaborare le risposte del *web serve*r;
* **Json[[12]](#footnote-12)**
  + formato di testo utilizzato per lo scambio di dati;
* **Sql[[13]](#footnote-13)**
  + linguaggio utilizzato per interrogare e gestire basi di dati;
* **Xml[[14]](#footnote-14)**
  + linguaggio di *markup* utilizzato per la creazione della grafica.

Capitolo 3 – Progetto e implementazione

3.1 Progettazione schema Entità/Relazioni

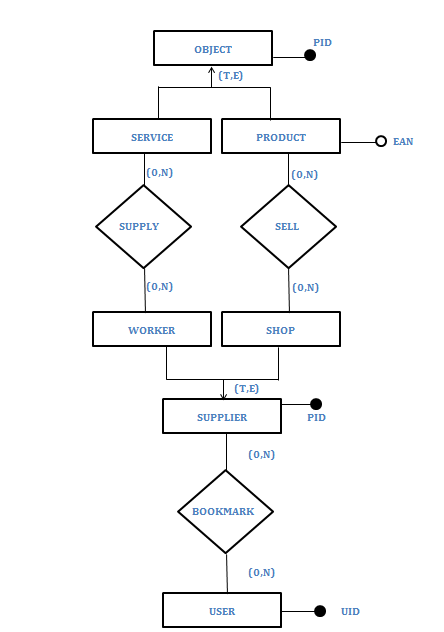


Figura 6:Schema E-R del database principale sul *web server* Apache

Per rappresentare come le informazioni, vengono gestite dell’app Viola, si utilizza lo schema E/R rappresentato nella figura 6; le principali entità saranno quindi: utenti, negozi, lavoratori, prodotti e servizi.

A ciascun utente sono associati: un nome, una mail e delle informazioni per la gestione del *login*.

Ogni negozio è descritto da informazioni principalmente di contatto, fra cui la latitudine e la longitudine necessarie per la geolocalizzazione, gli orari di apertura e chiusura, una bacheca per le informazioni in tempo reale.

Un lavoratore ha gli stessi dati di un negozio.

L’entità prodotto contiene informazioni varie fra cui un nome, una descrizione ed il codice a barre EAN.

Un servizio è descritto come un prodotto ma non possiede un codice a barre.

Inoltre, si evidenziano le seguenti relazioni:

* gli utenti iscritti che possono avere negozi o lavoratori fra i loro preferiti;
* ogni negozio può vendere uno o più oggetti;
* ogni lavoratore può offrire uno o più servizi;
* ogni prodotto può essere venduto da uno o più negozi;
* ogni servizio può essere offerto da uno o più lavoratori.

3.2 Progettazione logica relazionale

Dopo aver progettato lo schema E/R secondo le specifiche desiderate, si è proceduto con la realizzazione dello schema logico relazionale, considerando le operazioni che andranno ad essere eseguite su di esso.

La semplificazione dello schema E/R ha richiesto principalmente:

* l’eliminazione delle gerarchie presenti con copertura totale ed esclusiva. La scelta è ricaduta sulla metodologia del collasso verso il basso di tutte le gerarchie, eliminando l’entità padre e trasferendo gli attributi o associazioni a tutte le entità figlie. Tale strategia è ottimale in quanto essendo una copertura esclusiva, non introduce nessuna ridondanza, favorendo tutte le operazioni che in questo caso, accedono ai dati separatamente;
* il partizionamento orizzontale dell’associazione Bookmark, il quale permette di dividere i preferiti in base alla loro appartenenza a negozi o lavoratori (scelta che permette di migliorare l’efficienza delle operazioni svolte su di essi) [[15]](#footnote-15).

Essendo presenti solo associazioni binarie del tipo (0,N) : (0,N) la traduzione di entità e associazioni è stata effettuata mediante traduzione *standard*, in questo caso unica opzione possibile.

Si riporta lo schema logico relazionale:

USER (UID, UNIQUE\_ID,NAME, EMAIL, ENCRYPTED\_PASSWORD, SALT, CREATED\_AT, UPDATED\_AT)

**AK**: UNIQUE\_ID

**AK**: EMAIL

SHOP ( SHID, NAME, TEL, TIME, DESKTOP, DESCRIPTION, LAT, LNG, CREATED\_AT, UPDATED\_AT)

**AK**: TEL

WORKER( WID, NAME, TEL, TIME, DESKTOP, DESCRIPTION, LAT, LNG, CREATED\_AT, UPDATED\_AT)

**AK**: TEL

BOOKMARKSHOP ( UID, SHID)

**FK**: UID **REFERENCES** USER not null

**FK**: SHID **REFERENCES** SHOP not null

BOOKMARKWORKER ( UID, WID)

**FK**: UID **REFERENCES** USER not null

**FK**: WID **REFERENCES** WORKER not null

PRODUCT ( PID, EAN, NAME, DESCRIPTION, CREATED\_AT, UPDATED\_AT)

EAN UNIQUE

SERVICE ( SEID, NAME, DESCRIPTION, CREATED\_AT, UPDATED\_AT)

SELL ( SHID, PID)

**FK**: SHID **REFERENCES** SHOP not null

**FK**: PID **REFERENCES** PRODUCT not null

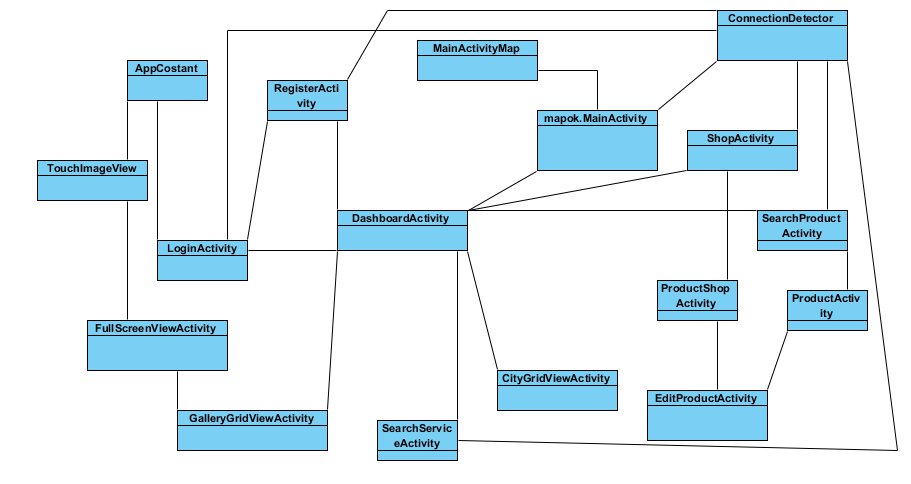
SUPPLY ( WID, SEID)

**FK**: WID **REFERENCES** WORKER not null

**FK**: SEID **REFERENCES** SERVICE not null

3.3 Class Diagram

Il Class Diagram illustra le principali classi utilizzate dal *software* e i loro legami. Si è volontariamente scelto di omettere alcune classi, alcuni legami, attributi e metodi in quanto tale elenco avrebbe comportato il venirsi a formare di uno schema incomprensibile.



Le classi più importanti sono[[16]](#footnote-16):

* ConnectionDetector.java
  + per effettuare un controllo di connessione ad internet (fondamentale per permettere all’app di funzionare sia in modalità *online* che *offline*);
* GPSTracker.java
  + per rilevare la posizione, controllare l’attivazione del GPS o del wi-fi, avvertire l’utente in caso entrambi i dispositivi sono disattivati;
* DashboardActivity.java
  + è la schermata principale da cui si può accedere alle varie funzioni;
* JSONParser.java
  + classe importantissima per effettuare le richieste http sia con il metodo GET che POST ed ottenere risposte in formato JSON;
* DatabaseHandler.java
  + per gestire il database interno su SQlite.

3.4 Implementazione

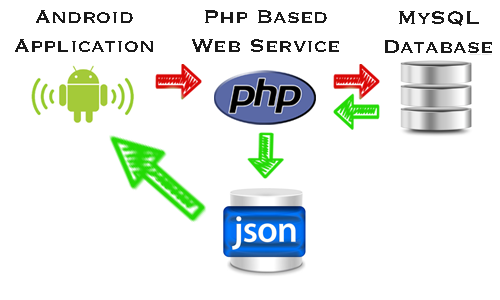


Figura 3: Architettura *client/server*

L’applicazione sul *device* Android funziona da *client*, inviando richieste http ai *web server*; questi elaborano le richieste e forniscono risposte in formato Json all’applicazione.

Come formato di rappresentazione dei dati scambiati fra le API adoperate e l’applicazione, è stato scelto JSON (JavaScript Object Notation). Questa scelta, alternativa all’utilizzo del XML, si basa sulla semplicità di utilizzo e sulla efficacia nella descrizione di ogni tipologia di risorsa.

Le caratteristiche principali di questa architettura sono:

* l’impossibilità per i *client* di interagire direttamente con il *data storage*, che rimane interno ad ogni *server* (in questo modo si migliora la portabilità del codice del *client*);
* l’inabilità del *server* di interagire con l’interfaccia dell’utente o con lo stato dell’utente, consentendo di avere un *server* più semplice e scalabile. Inoltre *server* e *client* possono essere rilocati e sviluppati indipendentemente, senza alterare l’interfaccia tra loro;
* un sistema *stateless*. Non è necessario quindi implementare funzionalità sul lato server atte a mantenere le sessioni utenti. Tale caratteristica implica che ogni nuova richiesta è indipendente da quelle precedenti e può essere gestita da un’istanza qualsiasi del *server*. Pertanto qualsiasi richiesta deve contenere tutte le informazioni necessarie per consentire a tutte le istanze del server di gestirla;
* l’impossibilità di un *client* di stabilire se è connesso direttamente al *server* oppure ad uno stato intermedio. I server intermedi sono in grado di ottimizzare la scalabilità del sistema rendendo il carico bilanciato e provvedendo alla condivisione della *cache*. Questa caratteristica inoltre, è utile ai fini delle politiche di sicurezza.

Le richieste http possono essere effettuate verso:

* *Web server* di Google
  + per scaricare le mappe
  + per ricercare i riferimenti geolocalizzati
  + per ricercare ulteriori contenuti
* *Web server* principale dell’applicazione
  + per effettuare il *login* e la registrazione
  + per cercare prodotti e servizi
  + per cercare i negozi aggiunti ai preferiti



Figura 4: Suite Wamp: Apache + Php + Mysql

Costruendo una API in Php, si possono svolgere operazioni sui database senza preoccuparsi del DBMS implementato (Google sconsiglia pienamente di interrogare i database direttamente dalle applicazioni).

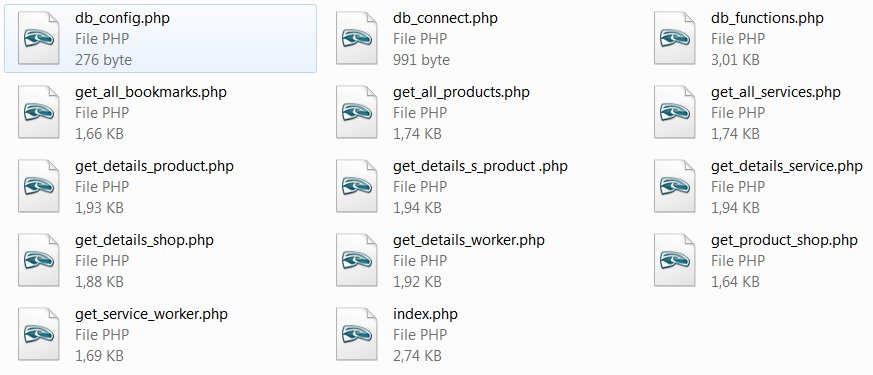


Figura 5: files Php

Capitolo 4 – Descrizione generale

4.1 Login

Per condurre analisi mirate sui singoli clienti, nonché per ottenerne l’indirizzo email, è stato necessario introdurre la registrazione degli utenti. Tuttavia si è cercato di minimizzare le informazioni richieste per la procedura di accesso al sistema informatico e in futuro verrà implementato l’accesso con account Google Plus e Facebook.

Graficamente si è cercato di facilitare l’utilizzo e renderlo quanto più possibile intuitivo, impiegando pulsanti, icone e strutture di menu standard già note agli utenti.

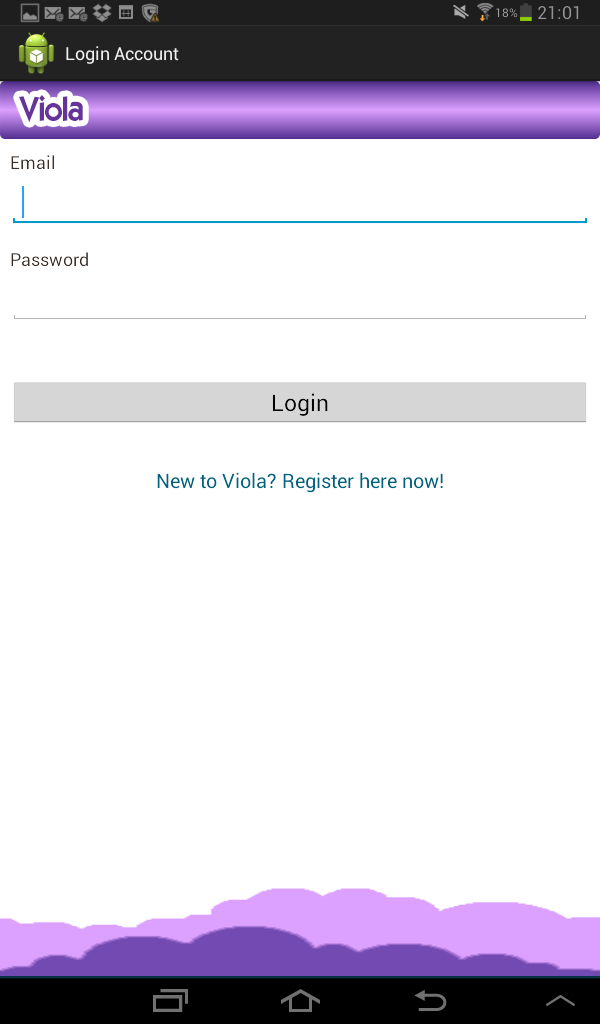
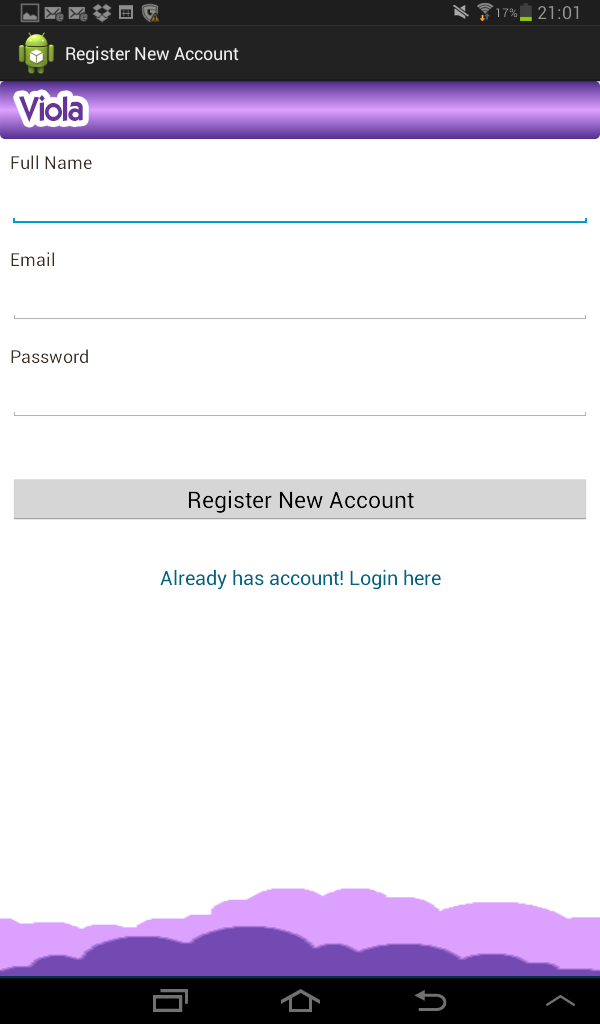


Figura 7: schermata registrazione e login

Mediante la classe JSONParser.java l’applicazione è in grado di effettuare richieste http[[17]](#footnote-17) e analizzare le risposte restituite dal *web server* in formato json. In relazione a queste schermate si avranno, in particolare, cinque possibili risposte[[18]](#footnote-18):

La tabella USERS contiene le informazioni dei nostri utenti, di cui evidenziamo:

* “unique\_id” ovvero il codice che identifica in modo univoco ogni *user*, generato con **uniqid(’’, true);**
* “encrypted\_password” e “salt” ossia i dati necessari per memorizzare le password con il metodo base64\_encode[[19]](#footnote-19).

Figura 8: prima riga tabella USERS

Vengono quindi riportate le istruzioni utilizzate per la creazione della tabella USERS e le medesime istruzioni generate automaticamente durante l’esportazione del database. Il codice allegato sarà in quest’ultima forma.

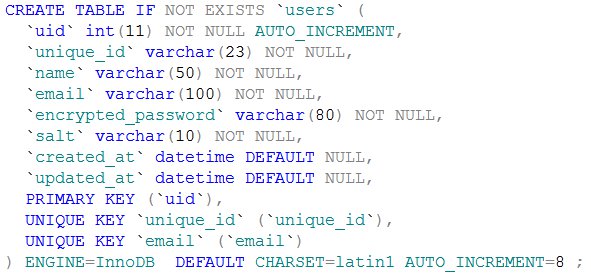
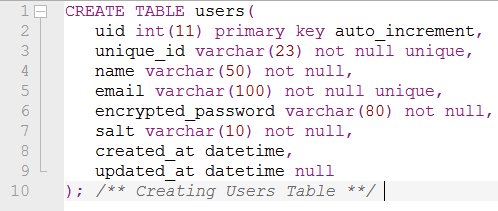


Figura 9: istruzioni tabella USERS originali e di esportazione

Per memorizzare le informazioni dell’utente nell’applicazione è stato utilizzato un database SQLite: ciò permette all’utente di poter riaccedere all’applicazione senza ripetere il login volta per volta.

4.2 Ricerca sul database

L’utente può ricercare un prodotto digitando il testo nell’apposito riquadro o mediante la scansione del codice a barre EAN (funzione implementata in parte poiché accessibile solo da dispositivi dotati di auto focus per la lettura dei codici).

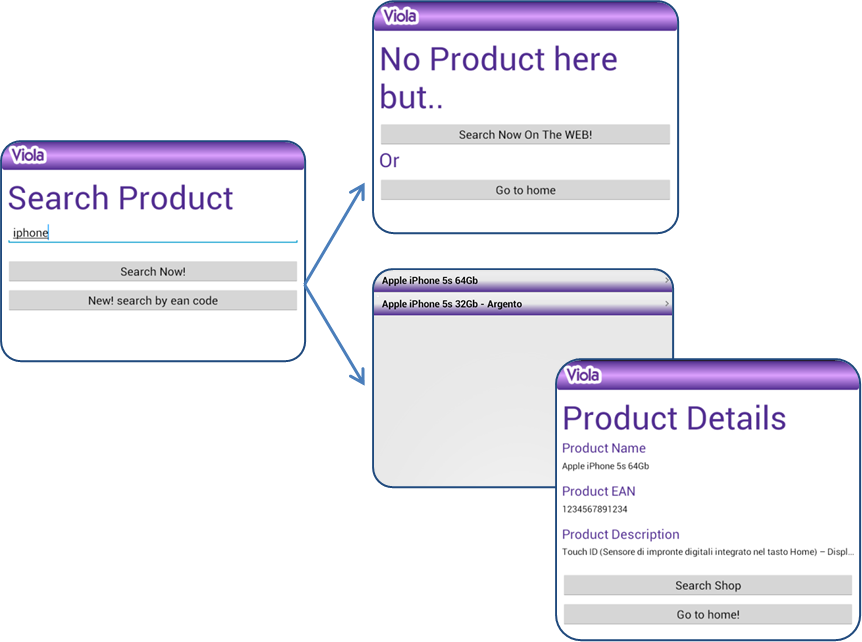


Figura 10: output ricerca prodotti

A seconda della presenza o meno dei prodotti ricercati sul database principale, l’utilizzatore sarà indirizzato alternativamente:

* alla lista dei risultati, in questo caso si ha la disponibilità del prodotto e si possono visionarne le caratteristiche, nonché accedere successivamente ai riferimenti dei venditori;
* alla schermata di atterraggio per richieste con esito negativo: si permette in questo caso di visionare i risultati presenti su Google tramite *browser*. Questa implementazione è utile particolarmente all’inizio del periodo di diffusione dell’applicazione, caratterizzato inevitabilmente da un database di dimensione ridotte. Si potranno inoltre suggerire le ricerche effettuate dai potenziali clienti ai venditori che già dispongono della propria vetrina virtuale su Viola.



Figura 11: esempio riga ListView activity prodotti

L’inserimento di un *input* nella ricerca prodotti rende l’applicazione oggetto di potenziali attacchi (come mysql injection). Sebbene tale aspetto non è stato particolarmente approfondito e non è possibile assicurare che il codice sia totalmente sicuro, sono stati inseriti dei controlli *input* e successivamente testati alcuni semplici attacchi ai quali l’app non risulta vulnerabile.

Viene riportata la *query* per la ricerca dei prodotti, in grado di fornire risultati corretti anche per parole in disordine o separate erroneamente (ad esempio verrà fornito lo stesso risultato sia digitando “iphone 5s” sia “5 s ipHoNe”).

C:\Users\MediaWorld\Desktop\immagini tesi\Fotografia_dello_schermo_062714_032524_PM.jpg

Figura 12: *Query* per la ricerca prodotti

Il database per ogni prodotto contiene una sola istanza[[20]](#footnote-20) (anche se venduto da soggetti diversi), questa caratteristica comporta dei vantaggi considerevoli per la piattaforma Viola:

* possibilità dei negozi di inserire i prodotti, presenti nel database, all’interno della propria vetrina *online* immediatamente;
* controllo e possibili migliorie da parte del produttore di mini descrizione, titolo e foto;
* si evita la ridondanza dei dati e i problemi ad essa legati.

La ricerca dei servizi è speculare alla ricerca prodotti, con differenze esclusivamente nelle tabelle di ricerca.

Da queste schermate, come in tutte, è possibile ritornare alla pagina precedente o alla home.

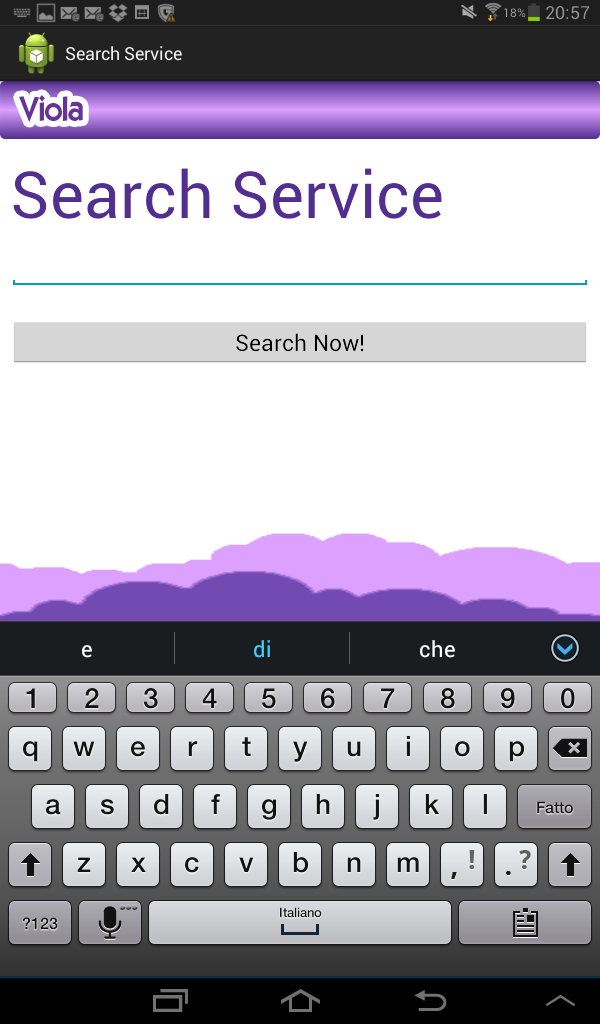
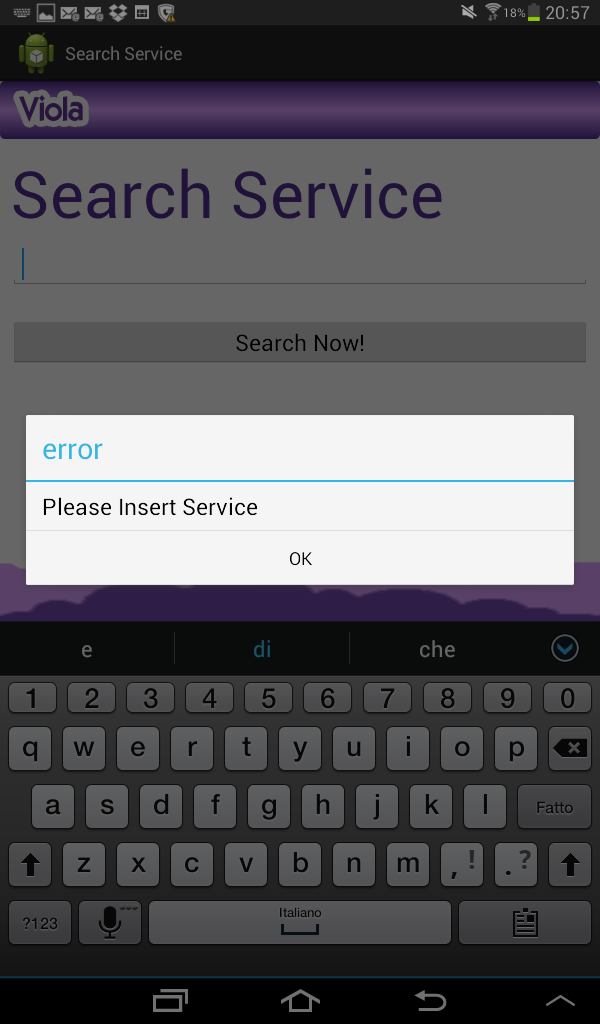


Figura 13: schermate ricerca servizi

4.3 Il menù



Figura 14: Smartphone (3.8”) vs Tablet(7.0”)

Il menù permette di accedere alle sei funzioni principali dell’applicazione, rappresentate da icone visivamente attraenti e di facile comprensione per semplificare la navigazione dell’utente.

Nella figura 14 è possibile notare uno dei problemi principali affrontato dai *designer* in relazione ai dispositivi Android: gli schermi hanno diverse dimensioni, differenti risoluzioni e per di più possono essere rotati dall’utente.

Tale problematica è stata affrontrata con un *design* sviluppato in un’ottica relativa e mediante l’utilizzo di codice messo a disposizione da Google per le *dashboard*[[21]](#footnote-21); la soluzione implementata ottimizza il numero di righe e colonne, sulle quali verranno disposte le icone, in funzione delle dimensioni dello schermo.

In Android per avere immagini di qualità, ovvero con giuste dimensioni e corretta densità di *pixel*, è necessario crearle in base alle specifiche caratteristiche delle figure ed inserirle nelle apposite *directory*. Inoltre per produrre l’effetto *feedback* nella selezione di icone da parte dell’utente, vengono create immagini correlate come raffigurato nella figura seguente.



Figura 15: esempio effetto selezione di icona

La *dashboard* realizzata, utilizza per semplicità esclusivamente immagini in “hdpi”, che come mostrano le statistiche ufficiali sono ancora le più diffuse: in base a questa scelta si è dovuto creare soltanto diciotto immagini (ovvero tre figure, per l’effetto selezione, per ognuna delle sei icone) ottenendo un risultato soddisfacente.

Per un risultato grafico quasi perfetto sarebbe stato necessario realizzare novanta immagini[[22]](#footnote-22) (ossia le diciotto figure precedenti per ognuna delle cinque tipologie di risoluzione più utilizzate[[23]](#footnote-23)); questa scelta avrebbe tuttavia determinato un maggior consumo di spazio in memoria.

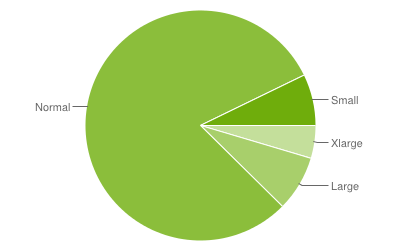
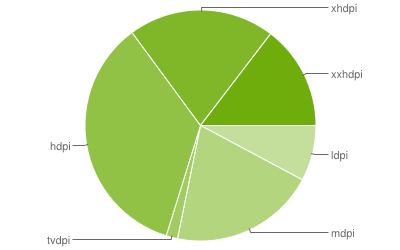


Figura 16: diffusione dimensioni e densità schermi su dispositivi Android

Fonte: https://developer.android.com/about/dashboards/index.html

4.4 Mostra mappe

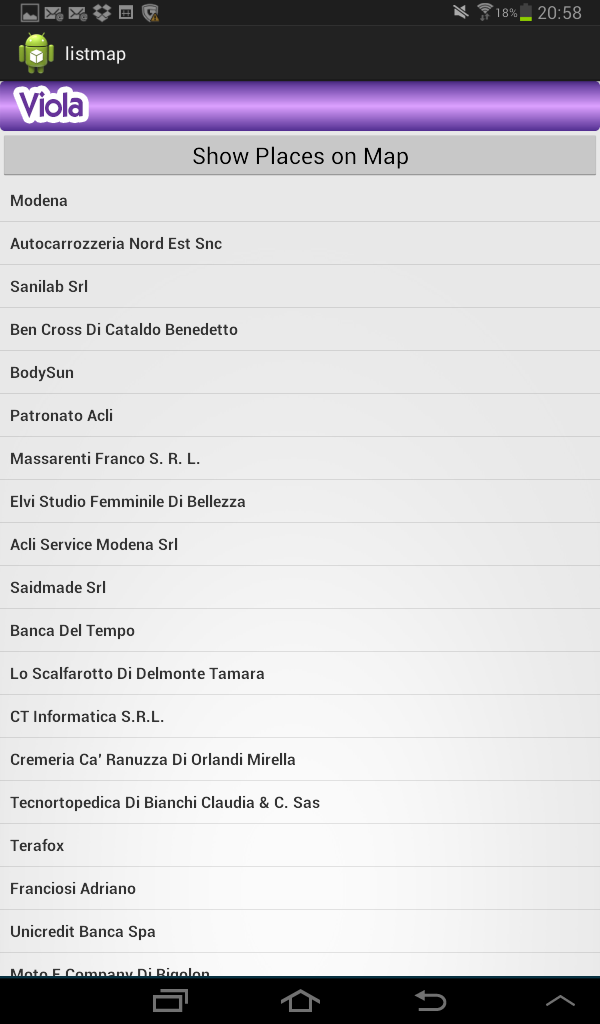


Figura 17: layout activity mappe

## Le due activity, riportate nella figura 16, mostrano i punti di riferimento su lista e su mappa, presenti nel raggio di 500 metri rispetto alla posizione dell’utente.

## Per usufruire dei servizi maps è stato necessario scaricare le librerie opportune[[24]](#footnote-24) e generare una “Google Maps API key” specifica per questo progetto.

## Per quanto riguarda la geolocalizzazione si è preferito utilizzare la connessione internet che, seppur meno precisa, ha migliori prestazioni in termini di consumo della carica del dispositivo rispetto al sistema GPS (quest’ultimo sarà utilizzato solo se già attivo).

Si sottolinea che anche Google, come il sistema di comunicazione dal *server* dell’applicazione, per fornire i dettagli su Places ha preferito il formato Json[[25]](#footnote-25).

La ricerca sulla mappa viene effettuata per qualsiasi categoria di punti d’interesse e permette all’utente di accederne ai dettagli se riportati. È dunque fondamentale, per la visibilità *online* di un’attività, la presenza su Google Places[[26]](#footnote-26) e l’indicazione delle proprie informazioni (quest’ultima non essendo obbligatoria, viene spesso rimandata, tralasciando dati importanti come l’URL del sito web o della pagina Facebook).

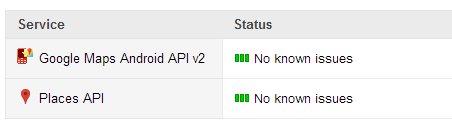
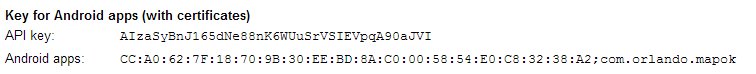


Figura 18: API Key specifica di Viola e relativi servizi attivati

Da un punto di vista estetico, si è preferito utilizzare la visualizzazione normale con un certo grado di zoom di default, in quanto è risultata la raffigurazione più chiara tra le alternative testate.

L’utente può scorrere ed ingrandire la mappa a suo piacimento, nonché ritornare alla visualizzazione di partenza tramite la “bussola”, posizionata in alto a destra dello schermo.

Selezionando un elemento della lista se ne ottengono i dettagli. Quando è presente il sito internet, mediante un *listener* ed un *intent*, è possibile aprirlo in automatico con il *browser*.

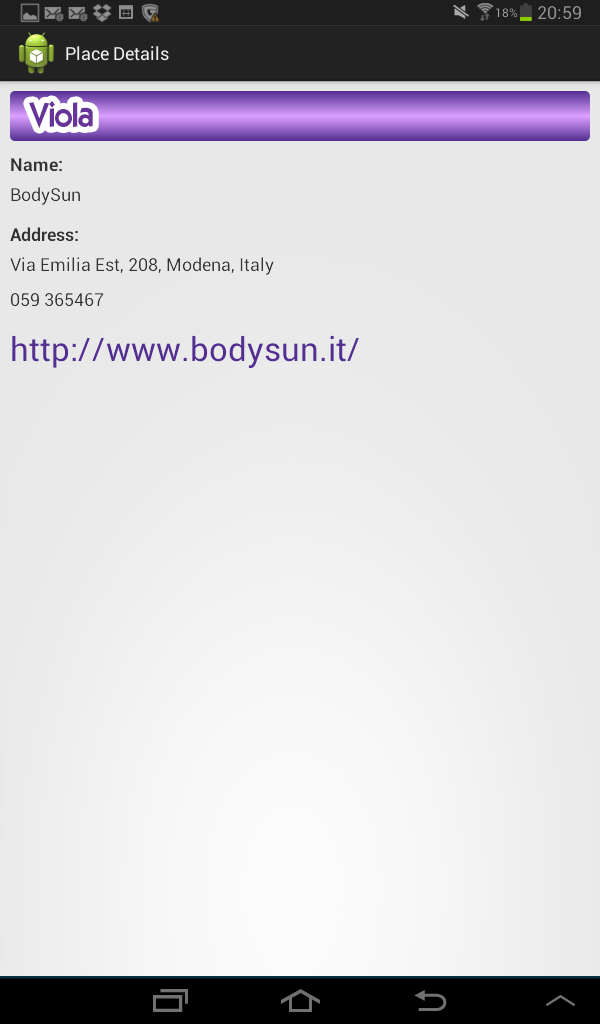


Figura 19: esempio schermata dettagli negozio

4.5 Mostra vetrine

Questa sezione mostra le foto delle vetrine dei negozi registrati su Viola: lo scopo principale di questa funzione è lo stesso di quelle reali, incuriosire le persone per farle entrare in negozio. L’idea di una piattaforma che raccoglie le varie vetrine sembra ancora non essere presente sul web[[27]](#footnote-27).

Mediante la memorizzazione delle foto all’interno dell’app, si permette all’utente anche la consultazione *offline*.

La sezione “Mostra Vetrine” è caratterizzata da:

* la disposizione su griglia delle figure;
* la possibilità di ingrandire le immagini a tutto schermo e la funzionalità di *swipe* per sfogliarle;
* la possibilità di effettuare lo zoom con il movimento delle dita;
* la possibilità di accedere al *link* ufficiale del negozio, mediante la “*call to action*” intitolata “Go now!”.



Figura 20: esempio vetrine negozi Modena

4.6 Mostra offerte della città



Figura 21: esempio offerte in città

Per promuovere le attività turistiche, un servizio molto interessante è fornito da Minicards Mobile[[28]](#footnote-28) che sia con delle tessere cartacee, sia con un’applicazione, pubblicizza le attrazioni di molte città nel mondo.

Considerando che Minicards Mobile non è attualmente disponibile in Italia e sembrano non essere presenti altre piattaforme simili, Viola cerca di implementarne le funzioni di base, offrendo il medesimo servizio ai turisti del “Belpaese”.

4.7 Mostra preferiti



Figura 22: esempio negozio in lista preferiti

L’ultima funzionalità dell’app permette di creare una lista personalizzata di negozi o professionisti. L’utente può quindi visualizzare le informazioni preferite in modo rapido e senza digitare nessun carattere.

4.8 Codice

L’intero codice formato da progetto Eclipse, librerie, database, pagine Php sarà disponile, su richiesta, sul sito http://www.dbgroup.unimo.it.

Il codice è ampiamente commentato per cui si è evitato di descriverlo nuovamente all’interno di questa tesi.

4.9 Note conclusive sulle scelte implementative

Per realizzare l’idea alla base di Viola, sono state effettuate una serie di scelte:

* Sviluppare app native oppure un sito *mobile-friendly*?

Si è deciso di implementare un’app nativa per offrire all’utente un’esperienza più personalizzata e non una versione minimalista di un sito *web* (seppure viene considerata un’alternativa molto valida)

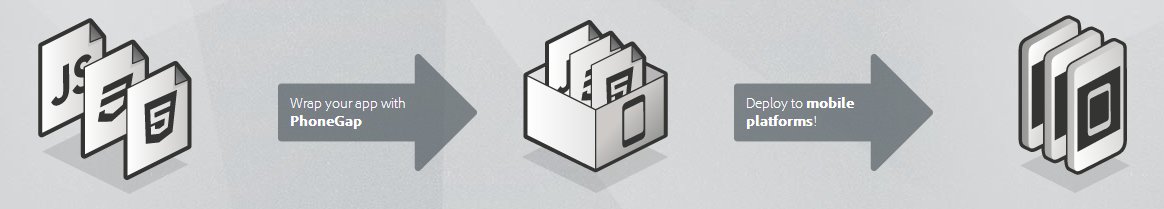
* Per quale piattaforma?

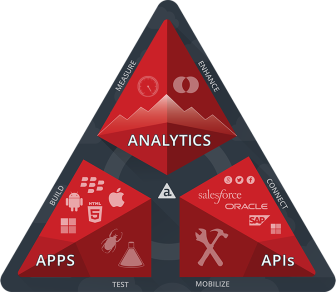
Android: data la sua enorme diffusione e per sfruttare le potenzialità di servizi *open-source*, principalmente gratuiti.

* Con quale linguaggio?

Java: il linguaggio madre di Android, per offrire maggiori prestazioni in termini di efficienza.

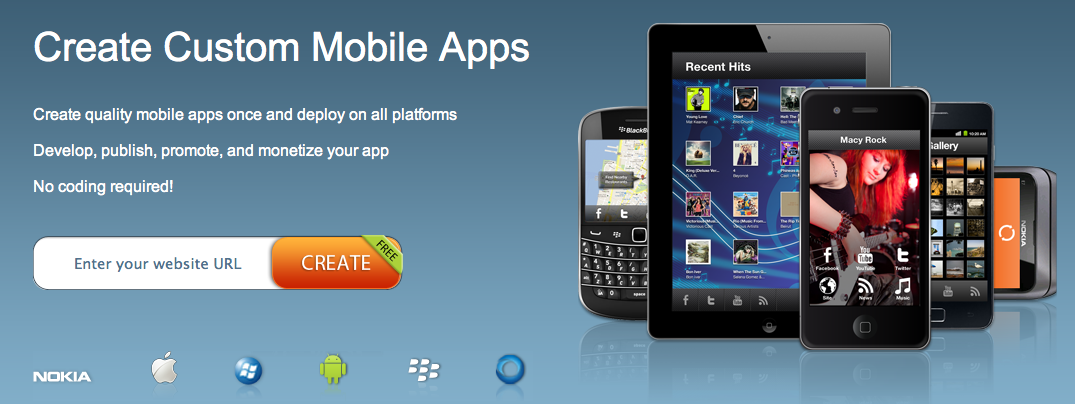
Le soluzioni con cui si sono dovute confrontare queste decisioni sono davvero tante. Oggi, infatti, non si è ancora giunti a definire in modo univoco l’utilizzo di tecnologie *standard* e di un linguaggio comune, ma esistono strumenti *open source* che agevolano la portabilità di un’applicazione per i diversi tipi di dispositivi e su tutte le piattaforme (caso ideale). Fra questi i più popolari sono PhoneGap e Appcelerator[[29]](#footnote-29).



*“PhoneGap is a free and open source framework that allows you to create mobile apps using standardized web APIs for the platforms you care about[[30]](#footnote-30).”*

*“Create amazing, cloud-connected native apps  
using JavaScript[[31]](#footnote-31).”*

Una scelta alternativa riguarda siti gratuiti ed incredibilmente utili, come Conduit.com: una rete di sviluppatori *web* e *mobile* che permettono di creare app personalizzate di buona qualità per telefoni e *tablet* senza problemi di sistemi operativi o *device*, e di implementarle su tutte le piattaforme. Conduit.com inoltre, consente di sviluppare, pubblicare, promuovere, monetizzare, scambiare e distribuire la propria app senza scrivere una riga di codice (particolarmente utile se si vuole verificare la reazione del mercato, prima di fare investimenti notevoli nello sviluppo di un’app).



Simile a Conduit.com vi è AppsBuilder, che consente a chiunque, attraverso pochi click guidati e senza alcuna conoscenza di programmazione, di creare la propria applicazione mobile e pubblicarla sui vari market o generare istantaneamente la versione mobile di qualsiasi sito web.

Se invece non si possiedono il *know-how* tecnico e/o le risorse finanziarie necessarie a creare un’app, ma si possiede un’idea favolosa, si potrebbe sottoporla a un App Incubator (MedlMobile.com), disposto a sviluppare grandi idee per applicazioni offrendo in cambio un accordo di *revenue sharing* (non si avranno costi iniziali, ma si otterranno solo il 25% dei profitti).

L’analisi di queste alternative evidenzia che la scelta effettuata per realizzare l’applicazione Viola, sebbene efficiente, non è stata ottimale in termini di portabilità e velocità di sviluppo.

Conclusione

Si legge in un post di un blog iReaderReview.com:

“*In qualsiasi nicchia, le imprese eccezionali in cima alla piramide (l’1%) hanno un’enorme successo. Il successivo 4-9% è composto da imprese che vanno bene. Tutte le altre vanno malissimo. La stampa si concentra su quell’1% e questo può darvi l’impressione che anche vostro fratello quattordicenne possa guadagnare un milione al mese. Basta che progetti un’applicazione*”.

Infatti non basta creare un’applicazione, ma bisogna avviare una start-up, cercare finanziamenti, promuovere il progetto...... in altre parole occorrerà avvalersi di tutto il proprio impegno, della massima determinazione e augurarsi di avere un po’ di fortuna.

L’applicazione sviluppata con questo progetto di tesi, non è quindi un punto di arrivo, ma un punto di partenza per una potenziale futura attività lavorativa nonché scelta di vita.

Appendice [1]

Consigli per i principianti su Android

Per Android consiglio questo manuale: http://www.sauronsoftware.it/teaching/uniroma2/2013/android/

Seppur non aggiornato, fornisce informazioni di base in modo chiaro.

Fare riferimento il più possibile al sito ufficiale:

http://developer.android.com/index.html/

Blog fondamentale per *tutorial* interessanti su concetti avanzati di Android:

http://www.androidhive.info/

Ho studiato molto sul libro “Sviluppare App per Android” di Paul Deitel et al. (2012),

tuttavia lo sconsiglio (ormai obsoleto, utilizza un approccio basato su esempi di codice di applicazioni non più funzionanti, rimane a concetti di base).

Appendice [2]

Funzionalità delle classi

* FullScreenViewActivity.java
  + Permette di visualizzare le immagini delle offerte a schermo intero
* GalleryFullScreenViewActivity.java
  + Permette di visualizzare le immagini delle vetrine a schermo intero
* GalleryGridViewActivity.java
  + Permette di visualizzare le immagini delle vetrine disposte su griglia
* GridViewActivity.java
  + Permette di visualizzare le immagini delle offerte disposte su griglia
* FullScreenImageAdapter.java
  + per associare l’immagine delle offerte al *layout* *fullscreen* e permettere il *link* diretto all’offerta
* GalleryFullScreenImageAdapter.java
  + per associare l’immagine delle vetrine al *layout fullscreen* e permettere il *link* diretto al sito del negozio in vetrina.
* GalleryImageAdapter.java
  + per associare l’immagine delle vetrine al *layout* griglia
* ImageAdapter.java
  + per associare l’immagine delle offerte al *layout* griglia
* TouchImageView.java
  + Per effettuare lo zoom sia sulle vetrine che sulle offerte

Appendice [2] (cont.)

Funzionalità delle classi

* AlertDialogManager.java
  + Per mostrare una semplice Alert Dialog (finestra di avviso)
* ConnectionDetector.java
  + Per effettuare un controllo di connessione ad internet (fondamentale per permettere all’app di funzionare sia in modalità *online* che *offline*)
* GooglePlaces.java
  + Per interrogare il *server* di google relativo ai luoghi ed ottener i loro dettagli.
* GPSTracker.java
  + Per rilevare la posizione, controllare l’attivazione del GPS o del wi-fi, avvertire l’utente in caso entrambi i dispositivi sono disattivati
* MainActivity.java
  + Per mostrare la lista dei punti di riferimento relativi ai luoghi se presenti
* MainActivityMap.java
  + Per mostrare i punti di riferimento relativi ai luoghi sulla mappa
* Place.java
  + Per permettere la comunicazione di un luogo con tutti i suoi dettagli fra le varie classi
* PlaceDetails.java
  + Per permettere di ricavare i dettagli relativi ad un punto di riferimento (contiene un’informazione di stato in più rispetto alla classe Place)
* PlacesList.java
  + Per permettere la comunicazione dei luoghi disposti su lista con la activity relativa alla mappa
* SinglePlaceActivity.java
  + Per mostrare i dettagli di un singolo punto di riferimento
* DashboardActivity.java
  + È la schermata principale da cui si può accedere alle varie funzioni
* DashboardLayout.java
  + Per gestire il *layout* della *home* (variabile in base alle dimensioni dello schermo)
* LoginActivity.java
  + Per effettuare il login
* RegisterActivity.java
  + Per effettuare la registrazione
* AppConstant.java
  + Un contenitore per costanti fruibili dall’app, usato principalmente per inserire l’ip dinamico durante le prove in locale
* DatabaseHandler.java
  + Per gestire il database interno su SQlite
* JSONParser.java
  + Classe importantissima per effettuare le richieste http sia con il metodo GET che POST ed ottenere risposte in formato JSON

Appendice [2] (cont.)

Funzionalità delle classi

* UserFunctions.java
  + Varie funzioni utili: eseguire il logout, controllare se un utente è già loggato, restituire le informazioni sull’utente
* EditProductActivity.java
  + Per inserire i dettagli del prodotto nella relativa schermata
* EditServiceActivity.java
  + Per inserire i dettagli del servizio nella relativa schermata
* EditShopActivity.java
  + Per inserire i dettagli del negozio nella relativa schermata
* EditWorkerActivity.java
  + Per inserire i dettagli del lavoratore nella relativa schermata
* NoProductActivity.java
  + Nel caso di prodotto non trovato, permette la ricerca sul web
* NoServiceActivity.java
  + Nel caso di servizio non trovato, permette la ricerca sul web
* NoShopActivity.java
  + Nel caso di negozio non trovato, permette la ricerca sul web
* NoWorkerActivity.java
  + Nel caso di lavoratore non trovato, permette la ricerca sul web
* ProductActivity.java
  + Per disporre i prodotti cercati su di una lista
* ProductShopActivity.java
  + Per disporre i negozi trovati rispetto al medesimo prodotto su una lista
* SearchProductActivity.java
  + Per effettuare la ricerca prodotto
* SearchServiceActivity.java
  + Per effettuare la ricerca servizio
* ServiceActivity.java
  + Per disporre i servizi cercati su di una lista
* ServiceWorkerActivity.java
  + Per disporre i lavoratori trovati rispetto al medesimo servizio su una lista
* ShopActivity.java
  + Per disporre i negozi aggiunti ai preferiti su di una lista

Appendice [3]

Risposte del server per login

**Registration Success Response – Success Code = 1 (User Successfully Stored)**

{

    "tag": "register",

    "success": 1,

    "error": 0,

    "uid": "4f074ca1e3df49.06340261",

    "user": {

        "name": "Francesco",

        "email": "a@a.it",

        "created\_at": "2014-01-07 01:03:53",

        "updated\_at": null

    }

}

**Registration Error Response – Error Code = 1 (Error in storing)**

{

    "tag": "register",

    "success": 0,

    "error": 1,

    "error\_msg": "Error occured in Registartion"

}

**Registration Error Response – Error Code = 2 (User Already Existed)**

{

    "tag": "register",

    "success": 0,

    "error": 2,

    "error\_msg": "User already existed"

}

**Login Success Response – Success Code = 1 (User Logged in)**

{

    "tag": "login",

    "success": 1,

    "error": 0,

    "uid": "4f075eca631fb8.88015924",

    "user": {

        "name": "Francesco",

        "email": "a@a.com",

        "created\_at": "2014-01-07 01:03:53",

        "updated\_at": null

    }

}

**Login Error Response – Error Code = 1 (Login Error – Incorrect username/password)**

{

    "tag": "login",

    "success": 0,

    "error": 1,

    "error\_msg": "Incorrect email or password!"}

Appendice [4]

Funzione crittografia password



Bibliografia

Beneventano, Domenico, Sonia Bergamaschi, e Francesco Guerra. *Progetto Di Basi Di Dati Relazionali. Lezioni Ed Esercizi*. Bologna: Pitagora Editrice, 2007.

Cialdini, Robert B. *Le armi della persuasione. Come e perché si finisce col dire di sì*. Traduzione di G. Noferi. Giunti Editore, 2013.

Deitel, Paul, Harvey Deitel, Abbey Deitel, e Michael Morgano. *Sviluppare App per Android*. Torino: Pearson, 2012.

Kim, W. Chan, e Renée Mauborgne. *Strategia oceano blu. Vincere senza competere*. Rizzoli Etas, 2011.

Norman, Donald A. *La caffettiera del masochista. Psicopatologia degli oggetti quotidiani*. Firenze; Milano: Giunti Editore, 2009.

Rickman, Cheryl. *La start-up digitale. Guida pratica step by step. Dall’idea al mercato per il successo: dall’idea all’exit*. Milano: Franco Angeli, 2013.

Ries, Al, e Jack Trout. *Le ventidue immutabili leggi del marketing. Se le ignorate, è a vostro rischio e pericolo!* Traduzione di F. Giardini. Anteprima Edizioni, 2009.

Sitografia

http://www.oracle.com/technetwork/java/javase/downloads/index.html

http://www.eclipse.org/

http://developer.android.com/sdk/index.html

http://developer.android.com/tools/sdk/eclipse-adt.html

http://www.adobe.com/it/products/photoshop.html

http://komodoide.com/komodo-edit/

http://www.wampserver.com/en/

http://docs.oracle.com/javase/8/

http://www.php.net/

http://www.json.org/

http://www.sql.org/

http://www.w3.org/TR/REC-xml/

http://developer.android.com/about/dashboards/index.html, http://android.devapp.it/20-000-risoluzioni-sotto-android-adattiamo-la-grafica-delle-nostre-applicazioni-a-tutti-i-display-esistenti

http://www.e-xtrategy.net/2013/04/22/generazione-di-asset-multirisoluzione-per-titanium-appcelerator/

https://developers.google.com/maps/documentation/android/start#getting\_the\_google\_maps\_android\_api\_v2

http://www.androidhive.info/2013/08/android-working-with-google-maps-v2/

https://developers.google.com/places/documentation/details

http://minicards.com/

http://www.sauronsoftware.it/teaching/uniroma2/2013/android/

http://developer.android.com/index.html/

http://www.androidhive.info/

http://blog.universalmind.com/mobile-html5-phonegap-vs-appcelerator-titanium/

http://www.phonegap.com/

http://www.appcelerator.com/

http://www.conduit.com/

Ringraziamenti

Desidero innanzitutto ringraziare di cuore la Prof.ssa Sonia Bergamaschi, relatore di questa tesi, per avermi sostenuto nelle mie idee, sorprendendomi con i suoi insegnamenti didattici e di vita.

Ringrazio inoltre tutti i Professori del Corso di Laurea Triennale in Ingegneria informatica per aver contribuito egregiamente alla mia formazione accademica.

Desidero infine ringraziare chi sarà contento di questo traguardo e naturalmente la mia famiglia, a cui è dedicato.

1. Si veda appendice [1]. [↑](#footnote-ref-1)
2. Scelta dovuta per le funzionalità necessarie delle API. [↑](#footnote-ref-2)
3. Si veda http://www.oracle.com/technetwork/java/javase/downloads/index.html [↑](#footnote-ref-3)
4. Si veda http://www.eclipse.org/ [↑](#footnote-ref-4)
5. Si veda http://developer.android.com/sdk/index.html [↑](#footnote-ref-5)
6. Si veda http://developer.android.com/tools/sdk/eclipse-adt.html [↑](#footnote-ref-6)
7. Si veda http://www.adobe.com/it/products/photoshop.html [↑](#footnote-ref-7)
8. Si veda http://komodoide.com/komodo-edit/ [↑](#footnote-ref-8)
9. Si veda http://www.wampserver.com/en/ [↑](#footnote-ref-9)
10. Si veda http://docs.oracle.com/javase/8/ [↑](#footnote-ref-10)
11. Si veda http://www.php.net/ [↑](#footnote-ref-11)
12. Si veda http://www.json.org/ [↑](#footnote-ref-12)
13. Si veda http://www.sql.org/ [↑](#footnote-ref-13)
14. Si veda http://www.w3.org/TR/REC-xml/ [↑](#footnote-ref-14)
15. Per una spiegazione dettagliata si veda Beneventano, Domenico, Sonia Bergamaschi, e Francesco Guerra. *Progetto Di Basi Di Dati Relazionali. Lezioni Ed Esercizi*. Bologna: Pitagora Editrice, 2007. [↑](#footnote-ref-15)
16. La descrizione completa delle classi è riportata nell’appendice [2]. [↑](#footnote-ref-16)
17. Nel caso specifico del login mediante il metodo *post.* [↑](#footnote-ref-17)
18. Si veda appendice [3]. [↑](#footnote-ref-18)
19. Si veda appendice [4]. [↑](#footnote-ref-19)
20. Questo approccio è stato utilizzato negli ultimi anni da Amazon ed è totalmente opposto a quello caratteristico di Ebay). [↑](#footnote-ref-20)
21. Il copyright sarà presente nei commenti del codice. [↑](#footnote-ref-21)
22. Esistono script che automatizzano questo processo, si vedano http://developer.android.com/about/dashboards/index.html, http://android.devapp.it/20-000-risoluzioni-sotto-android-adattiamo-la-grafica-delle-nostre-applicazioni-a-tutti-i-display-esistenti e

    http://www.e-xtrategy.net/2013/04/22/generazione-di-asset-multirisoluzione-per-titanium-appcelerator/ [↑](#footnote-ref-22)
23. Rappresentate nel grafico a destra nella figura 16. [↑](#footnote-ref-23)
24. Si vedano https://developers.google.com/maps/documentation/android/start#getting\_the\_google\_maps\_android\_api\_v2

    http://www.androidhive.info/2013/08/android-working-with-google-maps-v2/ [↑](#footnote-ref-24)
25. È tuttavia disponibile anche il formato Xml, si veda https://developers.google.com/places/documentation/details [↑](#footnote-ref-25)
26. Tale visibilità è garantita dall’accesso da tutte le applicazioni che utilizzano queste API di Google. [↑](#footnote-ref-26)
27. La maggior parte delle foto delle vetrine vengono postate su Instagram, Facebook, nelle pagine personali dei negozi o spesso in siti comunali; l’implementazione di questa funzione potrebbe essere quindi la caratteristica vincente di Viola che diventerebbe la prima piattaforma in Italia specializzata in questo ambito. [↑](#footnote-ref-27)
28. Si veda http://minicards.com/ [↑](#footnote-ref-28)
29. Per un confronto si veda http://blog.universalmind.com/mobile-html5-phonegap-vs-appcelerator-titanium/ [↑](#footnote-ref-29)
30. Si veda http://www.phonegap.com/ [↑](#footnote-ref-30)
31. Si veda http://www.appcelerator.com/ [↑](#footnote-ref-31)