ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN'APPLICAZIONE ANDROID NELL'AMBITO DEI PRODOTTI DEL KM '0'

Relazione finale in

PROGRAMMAZIONE DI SISTEMI MOBILE

Relatore
Dott. Mirko Ravaioli

Presentata da
Roberto Ricciardi

Seconda Sessione di Laurea
Anno Accademico 2015 – 2016

Alla mia famiglia e a tutti coloro
che mi hanno sostenuto in questo percorso

Indice

Introduzione

1.1 Il piccolo grande mondo del Km '0'.

Scandali di certificazioni di prodotti biologici, contraffazione di alimenti, materie prime (farine e altro) provenienti dall'estero vendute come di origine italiane; è diventato frequente ascoltare notizie di questo tipo nei principali canali di comunicazione mediatica.
Quello dell'alimentazione può essere considerato un 'mondo' con tutte le sue varietà e difficoltà; il consumatore sta mostrando crescente interesse verso le tematiche di buona alimentazione.

Si può parlare di una vera e propria sensibilizzazione all'acquisto, sempre più persone sono attente a cosa comprano, cercando risposta a domande come: Questo prodotto da dove arriva realmente? Quali ingredienti sono presenti? La risposta a queste e altre domande diventa ancora più difficile quando i prodotti provengono da grandi filiere produttive. Esaminiamo nel dettaglio cosa si intende per chilometro zero e cos'è la filiera corta.

1.1.1 Chilometro zero e la filiera corta

"Il *Chilometro Zero* (anche *chilometro utile*, *km zero* o *km 0*) in economia è un tipo di commercio nel quale i prodotti vengono commercializzati e venduti nella stessa zona di produzione". [1]

La locuzione a chilometri zero identifica una politica economica che predilige l'alimento locale garantito dal produttore nella sua genuinità, in contrapposizione all'alimento globale spesso di origine non adeguatamente certificata. [1]

I consumatori hanno una crescente sensibilità verso le tematiche ambientali; il

commercio a chilometri zero concilia questi fattori offrendo la possibilità di conoscere i produttori; a volte è possibile sapere anche come vengono prodotti e seguendo quali principi.

Si parla di filiera corta quando la filiera produttiva è formata da un numero limitato di passaggi produttivi, e in particolare di intermediazioni commerciali, che possono portare anche al contatto diretto fra il produttore e il consumatore.[1]

1.1.2 Uno sguardo alle statistiche

Le statistiche confermano quanto detto sopra? Ecco alcune citazioni e riferimenti per quanto riguarda l'aumento dell'attenzione dei consumatori e l'utilizzo di e-commerce per l'acquisto online.

Nel solo 2015 il consumo online enogastronomico è cresciuto del 27% rispetto al 2014 e ha superato i 460 milioni di euro, ossia il 3% dell'e-commerce in Italia, secondo i dati dell'Osservatorio e-commerce B2c Netcomm-Politecnico di Milano. [2]

La maggiore confidenza degli utenti verso gli strumenti informatici e la facilità di acquisto mediante carte di credito, prepagate e altri mezzi di pagamento hanno contribuito a questo fenomeno.

Per quanto riguarda la crescente attenzione da parte dei consumatori La Coldiretti, che è la principale Organizzazione degli imprenditori a livello nazionale ed europeo, ha pubblicato la seguente analisi qui citata:

Dall' aumento del 50% degli acquisti di alimenti senza glutine all'incremento del 20% di quelli biologici senza l'uso della chimica fino al boom dei consumatori che cercano la garanzia "Ogm free" sono cresciuti a due cifre in Italia nel 2015 i consumi di alimenti "senza".

Un exploit da ricondurre – sottolinea la Coldiretti – all' attenzione per il benessere, la forma fisica e la salute, oltre che la crescente diffusione di intolleranze alimentari. Una tendenza in forte ascesa nonostante il sovrapprezzo

da pagare poiché – precisa la Coldiretti - il 70% degli italiani è disposto a pagare di più' un alimento del tutto naturale, il 65% per uno che garantisce l'assenza di OGM, il 62% per un prodotto bio e il 60% per uno senza coloranti, secondo l'ultimo rapporto Coop. Se gli acquisti di prodotti biologici confezionati fanno registrare un incremento record del 20 per cento con più' di un italiano su 3 che dichiara di acquistare cibi bio o naturali, sono quindici milioni le persone che - sottolinea la Coldiretti - mettono nel carrello prodotti locali a chilometri zero, mentre ad acquistare regolarmente prodotti tipici legati sono ben 2 italiani su tre secondo l'indagine Doxa per Coop. [3]
Come ha risposto a questo bisogno l'informatica?

1.2 L'informatica e il Km '0'

Grazie alla diffusione di smartphone e tablet, oggigiorno la maggioranza delle persone ha a disposizione uno strumento per accedere a internet; la diffusione dei social network ha portato gli utilizzatori a familiarizzare sempre di più con il concetto di condivisione di informazioni, siano esse foto, video, un pensiero scritto o altro; questo si è riflesso anche nell'alimentazione. Associazioni, cooperative e produttori utilizzano sempre di più internet come strumento di vendita o per far pubblicità alle loro produzioni. Ecco alcuni esempi e una breve analisi di cosa offrono alcuni siti:

- www.tuttogreen.it: è un *sito informativo* di green economy con una mappa di hotel, ristoranti, negozi e agriturismi bio con recensioni correlate; l'utilizzatore può quindi informarsi riguardo a deteminate strutture, e valutarle mediante recensioni.
- www.naturex.it: è un *e-commerce basato* sul territorio locale (Per ora zona Palermo-Sicilia); gli utenti possono effettuare prenotazioni online, e in determinate date fissate dai produttori viene fatto un vero e proprio mercato dove i consumatori possono ritirare le loro prenotazioni.
- universobio.com: è un *e-commerce biologico,* con servizio di spedizione

integrato; gli utenti ordinano i prodotti e questi vengono spediti a costi contenuti in tutto il territorio italiano ad eccezione delle isole e delle zone disagiate.

- amaterra.coop: sito che mette a disposizione degli utilizzatori una vetrina in cui possono visualizzare i prodotti del giorno; mediate telefono o email si prenotano i prodotti che vengono consegnati a domicilio in determinati giorni della settimana; i prodotti possono anche essere ritirati in una delle strutture della cooperativa AmaAquilone. Questo strumento è disponibile nel comune di Ascoli Piceno nella regione Marche.

Le soluzioni sopra riportate sono una rappresentanza di quello che è un vero è proprio movimento di digitalizzazione della vendita di prodotti alimentari, nonché della consegna. Ecco invece alcune soluzioni esistenti e in crescita per quanto riguarda i dispositivi mobili.

- 'L'Alveare che dice Sì – Ordina online prodotti a km0 '. Ecco come viene descritta l'applicazione: Un Alveare è una comunità che acquista direttamente dagli agricoltori ed artigiani della propria regione. Iscriviti a un Alveare vicino a te, scegli ed ordina i tuoi prodotti online; recati alla distribuzione con la tua lista della spesa direttamente sul tuo smartphone. Questa applicazione è attualmente disponibile solo per dispositivi con sistema operativo iOS. [4]
- 'Cortilia, la campagna a casa tua'. Cortilia offre un e-commerce di prodotti locali con un servizio di consegna a domicilio. È un'applicazione disponibile per dispositivi mobili con sistema operativo Android oppure iOS.

1.2.1 I limiti dell'informatica del KM '0'

La filosofia del km 0, della filiera corta e di coloro che acquistano questo tipo di prodotti, richiede il limitare al massimo la figura dell'intermediario; inoltre vi è da

parte del consumatore finale un desiderio a conoscere come il prodotto viene coltivato, seguendo quali principi e in che condizioni. Si costruisce un vero e proprio rapporto di fiducia tra produttore e consumatore; questo può essere un limite per le soluzioni informatiche, in quanto si rischia di far diventare lo strumento informatico, sia esso un sito, un'applicazione o altro, l'intermediario stesso che in realtà il consumatore vuole evitare. Questo costituisce una soglia nella scelta e nella definizione di una soluzione adatta a questo specifico dominio applicativo.

1.3 BlaBlaFood: come nasce l'idea

La passione per l'alimentazione e l'esigenza dei consumatori nel reperire un prodotto di qualità e locale, come descritto nei paragrafi precedenti, sono stati fattori determinanti nella scelta di questo dominio. Nel definire cosa doveva essere BlaBlaFood, è stato fondamentale capire quale esigenza dei consumatori si volesse soddisfare. Inizialmente si pensava a un sistema in cui produttori locali e consumatori potevano iscriversi e acquistare i prodotti, che eventualmente potevano essere ritirati in punti precisi o eventualmente consegnati. Questa soluzione, oltre a essere ambiziosa in quanto richiedeva la messa a punto di una rete di consegna, andava a collidere con il problema dell'intermediario esposto nel paragrafo precedente come principale limite dell'informatica del km 0.

Un'altra idea era quella di creare un sistema che permettesse ai produttori di condividere prodotti in offerta, con focus sempre nei prodotti bio e del km 0; i limiti principali di quest'altra idea combaciano in parte a quelli della soluzione precedente.

Solo facendo un'analisi di chi è in realtà il consumatore del prodotto bio e del prodotto locale, si è riusciti a definire in maniera precisa cosa BlaBlaFood dovesse offrire ai futuri utilizzatori.

L'idea finale è un sistema nel quale ogni iscritto può condividere le proprie

produzioni agricole e visualizzare quelle degli altri iscritti. BlaBlaFood non si pone come un intermediario per l'acquisto, ma come uno strumento per far conoscere ai potenziali consumatori interessati cosa altri utenti hanno da offrire per loro dal piccolo orto di casa; non è rivolta quindi ai grandi produttori, ma all'utente finale stesso; l'utente che coltiva un terreno per la propria famiglia può condividere i propri prodotti agricoli attraverso quella che è di base una sua vetrina visualizzabile dagli iscritti al sistema.

Questa è la linea seguita da altri sistemi di successo come Subito.it [5] , letgo [6] e altre società, che hanno fatto dell'individuo finale il principale attore del sistema.

Analisi

In questo capitolo si analizzano i requisiti che il sistema dovrà soddisfare e si evidenziano gli aspetti da tenere in considerazione per il dominio applicativo specifico.

2.1 Requisiti

Il software BlaBlaFood deve soddisfare la seguente specifica:

“ Il sistema deve essere in grado di permettere agli **utenti** di iscriversi per poter usufruire delle varie funzionalità disponibili; l'utente può iscriversi mediante social network (Facebook, Google+) e in seguito accedere al sistema. L'utente una volta autenticato può pubblicare i propri **prodotti**, ovvero ciò che vuole mettere in vendita agli altri utenti del servizio. Un **annuncio** è associato a un utente, ed è riferito a uno specifico prodotto; deve essere indicata anche la posizione di vendita così da permettere a chi visualizza l'annuncio di recarsi dall'offerente. Durante la compilazione dell'annuncio l'utente può inserire anche immagini, e una descrizione. Ogni utente può visualizzare gli annunci degli altri iscritti, ordinati in base alla posizione dalla quale li sta visualizzando. ”
Dalla specifica, si evidenziano le proprietà statiche del sistema; esse sono meno variabili nel tempo, sono su queste entità che il sistema si basa:

- Utente
- Prodotto
- Annuncio

2.2 Diagrammi dei casi d'uso

Segue una descrizione formale attraverso diagrammi dei casi d'uso.

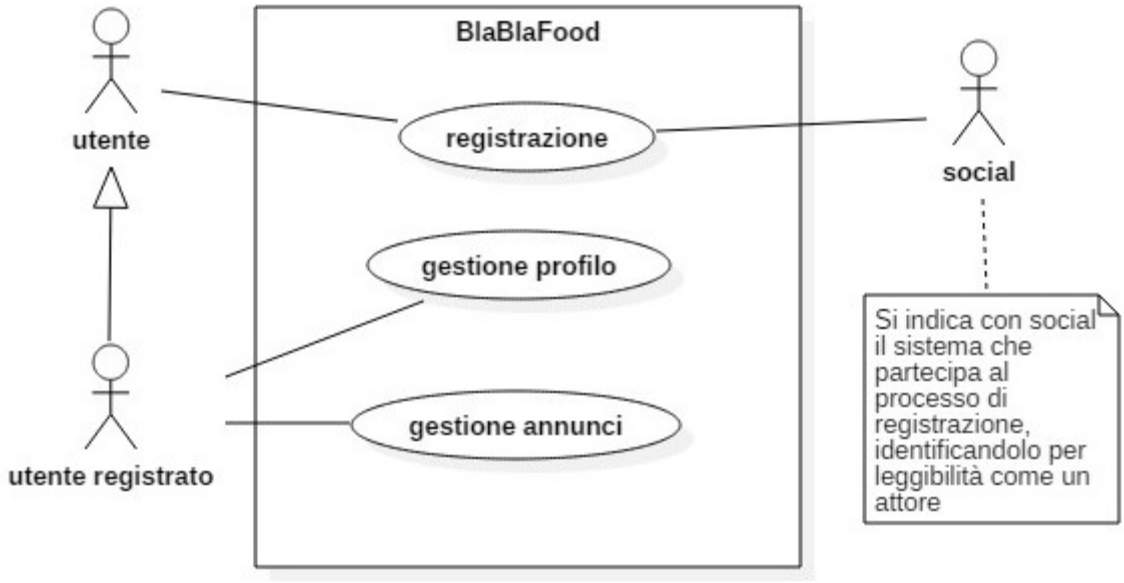

Figura 2.1 'Diagramma dei casi d'uso generale'

Come descritto sopra l'utente per usufruire delle funzionalità del sistema deve registrarsi; la registrazione avviene mediante social. L'utente registrato nel sistema BlaBlaFood può interagire con esso nella gestione del suo profilo e degli annunci. Nel diagramma seguente si pone maggiore dettaglio alle funzionalità offerte all'utente.

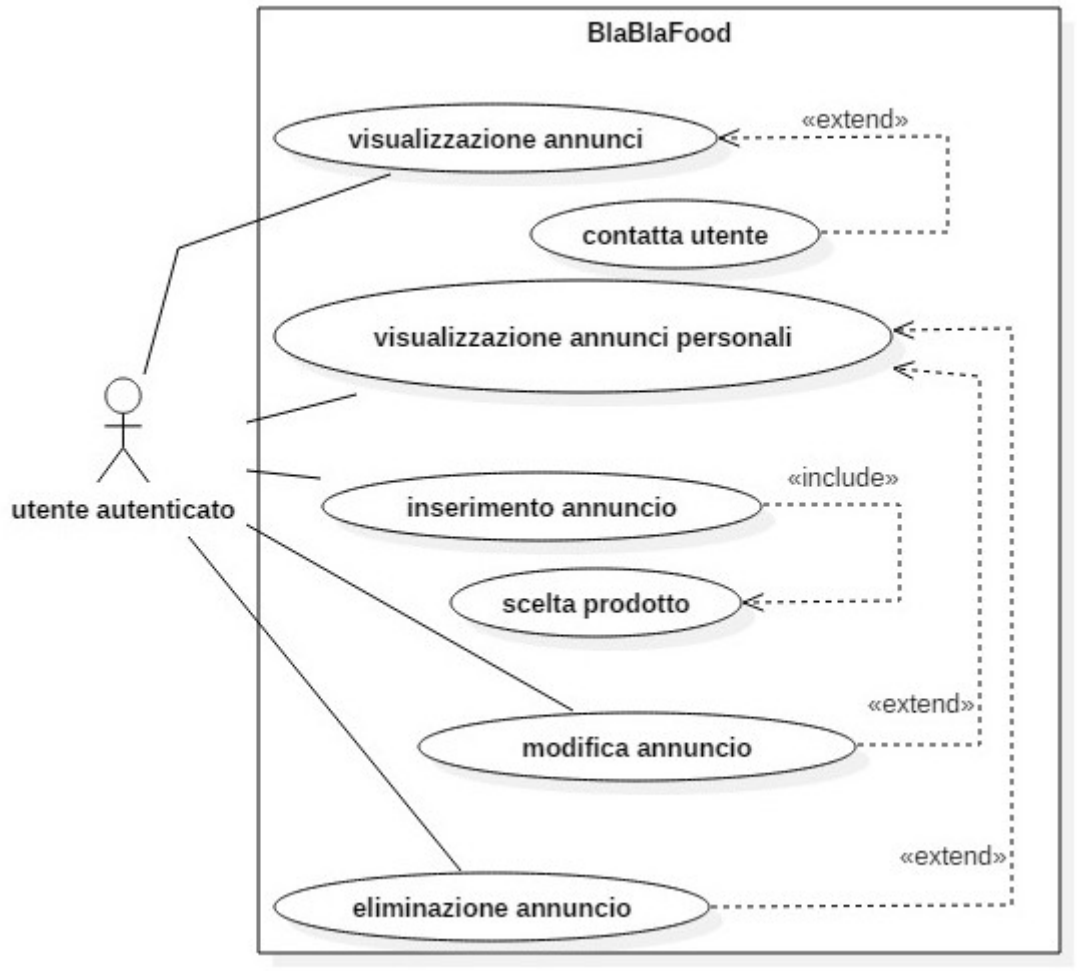

Figura 2.2 'Diagramma dei casi d'uso specifico

Dal seguente diagramma, si possono estrapolare ulteriori aspetti per poter completare correttamente l'analisi del dominio in questione. Un utente è autenticato quando è riconosciuto dal sistema, è precondizione che esso sia registrato per poter operare con esso. E' permessa la visualizzazione degli annunci degli altri utenti, sono concessi permessi CRUD (create, read, update, delete) solo sulle pubblicazioni da lui inserite.

2.3 Problematiche da gestire

Al fine di offrire un servizio di qualità occorre garantire una buona esperienza

d'uso; l'utente deve poter accedere alle funzioni principali dell'applicazione in maniera semplice e diretta. Ulteriori aspetti importanti da curare saranno la compilazione dell'annuncio, la gestione dei prodotti e delle immagini associate alla pubblicazione.

E' inoltre importante considerando lo scenario odierno fare un uso parsimonioso delle risorse, con particolare attenzione sul consumo della rete dati.

Progettazione

In funzione dei requisiti evidenziati in fase di analisi, si procede con la definizione della struttura del prodotto software da realizzare; si delinea l'architettura generale del sistema, per procedere in seguito, con maggior dettaglio, nel design dei moduli specifici.

3.1 Architettura generale

Nel seguente diagramma dei componenti si mostra l'architettura ad alto livello del sistema , allo scopo di evidenziare i macro moduli che verranno progettati.

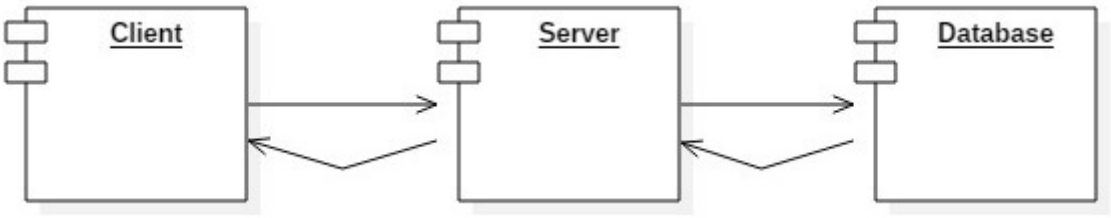

Figura 3.1 'Diagramma dei componenti: Architettura generale'

L'architettura è di tipo client-server; il client, che nella realizzazione specifica è un'applicazione Android, effettua richieste al server che le soddisferà ritornando i dati nel formato deciso per la comunicazione tra i due componenti, ovvero il JSON (la scelta è argomentata nel *capitolo 4 Implementazione)*. Per operare correttamente il sistema deve garantire la persistenza dei dati, a tal proposito il server è il responsabile della comunicazione con il Database, effettuando opportune interrogazioni sui dati. In questo scenario di alto livello il server ricopre un ruolo da intermediario, come è classicamente predisposto nelle architetture di tipo client-server.

3.1.1 Client e pattern architetturale

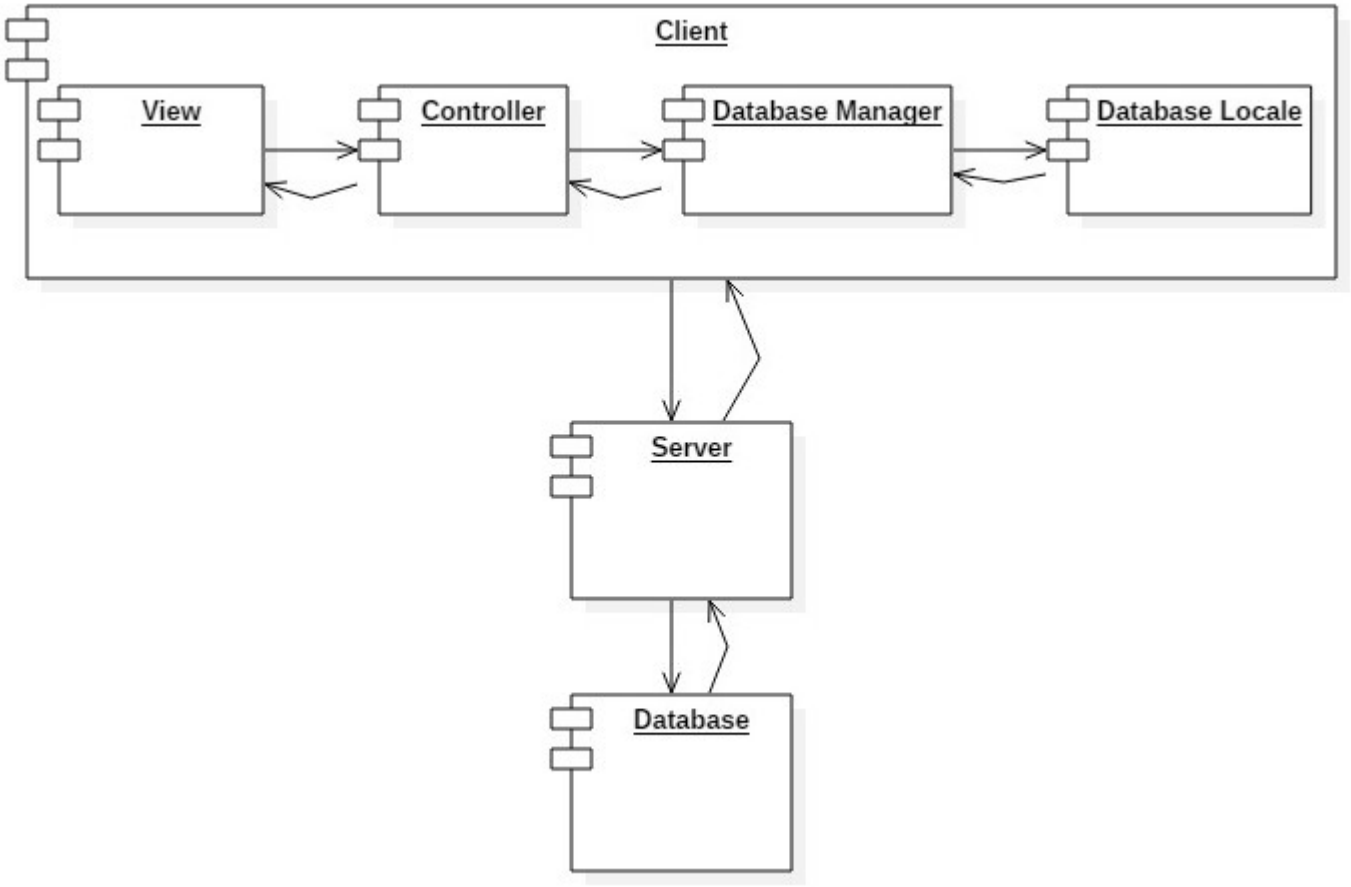

Figura 3.2 'Diagramma dei componenti: Client e pattern architetturale'

Rispetto alla figura 3.1 il diagramma sopra evidenzia l'architettura del client; si individuano i moduli della View, del Controller, del Database Manager e del Database Locale; si adescisce al pattern MVC, model-view-controller, grazie al quale si limitano le dipendenze tra model e view. Essendo il client un'applicazione Android, si precisa però che seguire rigidamente il pattern architetturale MVC non è sempre possibile. Nell'architettura di Android abbiamo già dei componenti con delle responsabilità ben stabilite; activity e fragment possono essere visti essi stessi come controllers. L'entry point naturale di Android non è il Controller, inteso come entità separata dalle activity (o fragment) che opera come

intermediario tra View e Model, bensì l'activity. Si sceglie quindi di aderire al pattern architetturale MVC tenendo in considerazione però il sistema Android.

3.1.2 Modello e entità

La parte statica del sistema è quella che muta meno nel tempo perché fortemente legata al dominio applicativo specifico; è su di essa che si basa l'intero sistema. Nel seguente diagramma si prosegue con la progettazione strutturale mettendo in risalto le entità del modello e le relazioni tra esse.

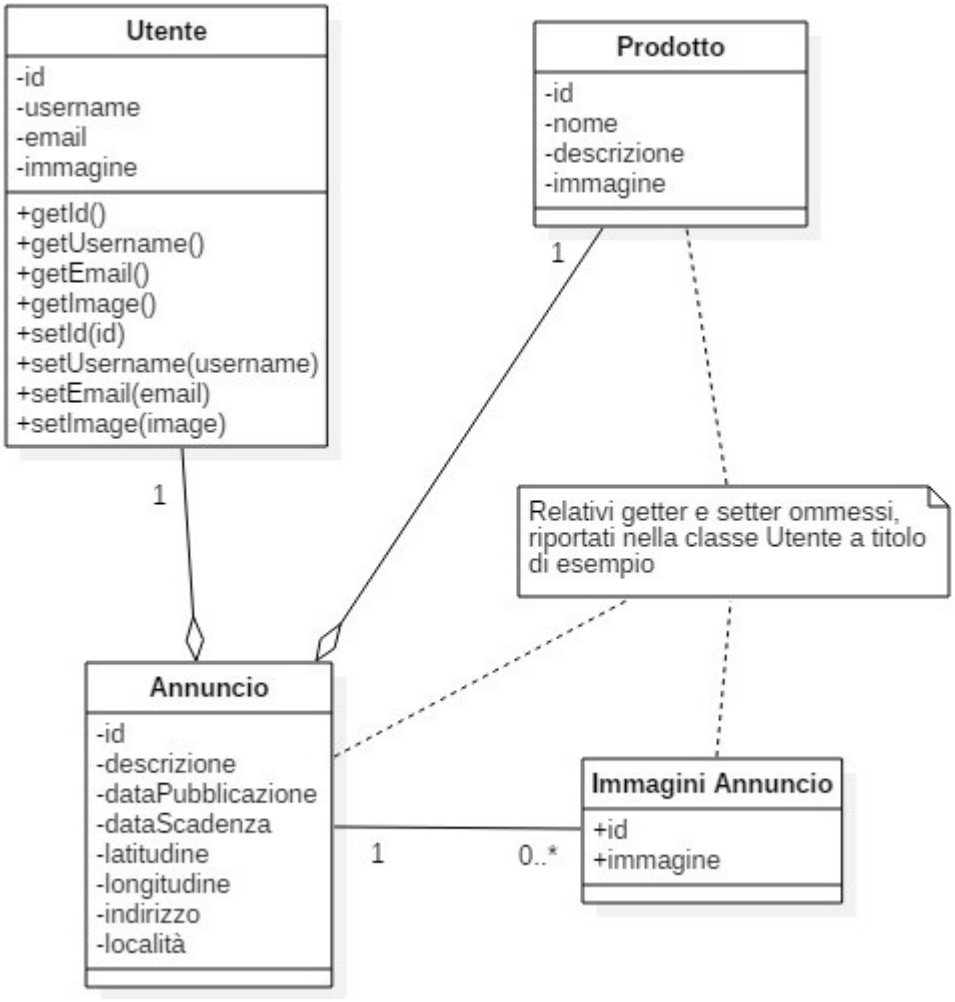

Figura 3.4 'Diagramma delle classi: Modello'

L'entità principale del modello è l'annuncio, a esso è associato un utente e un prodotto (nel corso della trattazione si farà riferimento a questa entità chiamandola

anche Pubblicazione o prodottoPubblicato); il campo descrizione, indica la descrizione aggiuntiva che l'utente può inserire rispetto a quella già presente nel prodotto. Vi è la possibilità di aggiungere delle immagini oltre a quella già fornita con il prodotto come evidenziato dall'associazione tra Annuncio e Immagini annuncio; dataPubblicazione e dataScadenza permettono di dare un limite temporale indicando la validità dell'annuncio. Per posizionare geograficamente l'annuncio, occorrono anche gli attributi latitudine e longitudine; in merito all'inserimento dei campi indirizzo e località, essi potrebbero essere evitati in quanto sono dati derivabili dalle coordinate geografiche; è stata una scelta di design inserirli visto che il processo di codifica e decodifica delle coordinate geografiche per ottenere l'indirizzo e la località richiede l'utilizzo di servizi di Geodecoding [7] e revers geodecoding come quelli offerti da Google. 'Geodecoding' è il processo di conversione di un indirizzo in coordinate geografiche, 'revers geodecoding' è, al contrario, la conversione da coordinate a indirizzo. Entrambi i servizi richiedono la connessione dati per poter operare, si può fare cache dei risultati ma secondo la policy dell'offerente del servizio. Considerando i requisiti emersi in analisi si è rivelato opportuno quindi inserire gli attributi indirizzo e località, si riesce così a garantire una buona esperienza d'uso anche in condizioni offline; in questo modo si diminuiscono le richieste ai servizi di geocoding e reverse geocoding migliorando l'uso delle risorse.

3.2 Servizi web

Considerando i requisiti si definiscono i servizi che il server deve offrire; si fa notare che nella seguente tabella si fa riferimento al parametro 'data', esso viene usato per ottenere un meccanismo di versionamento dei dati, utile per la sincronizzazione dei dati tra database del client e database lato server; questo aspetto viene approfondito dovutamente al *paragrafo 3.3.1 'Tecniche di sincronizzazione client-server'*. Si riportano inoltre delle note numerate consultabili a fine tabella.

Servizio	*Metodo*	*Parametri*	*Risposta*[*]	*Significato*
/login	POST	userId, social	SessionId, username, imagePath, email	Registra l'utente se non è presente già nel database, altrimenti esegue il login
/productType	POST	nome, immagine, descrizione	idProduct	Inserisce un nuovo prodotto con i parametri forniti, ritorna l'id del prodotto aggiunto
	GET	data[3]	Lista dei prodotti[4], nuova data[3]	Ritorna i prodotti che sono stati aggiunti o modificati dopo la data inviata
	DELETE	rowId[1]	Risultato operazione	Elimina il prodotto identificato da rowId
/publication	GET	userId, latitudine, longitudine, offset, limit, data[3]	Lista degli annunci[5], nuova data[3]	Ritorna gli annunci, degli altri utenti, che sono stati aggiunti o modificati dopo la data inviata, ordinati per posizione geografica
/publicationUser	POST	userId, Parametri publication[2]	idPublication	Se tra i parametri passati vi è l'id dell'annuncio allora modifica l'annuncio stesso con i parametri passati, altrimenti viene inserito un nuovo annuncio
	GET	userId, data[3]	Lista degli annunci[5] dell' utente, nuova data[3]	Ritorna gli annunci, dell'utente identificato con lo userId passato, che sono stati modificati o aggiunti dopo la data inviata

Servizio	*Metodo*	*Parametri*	*Risposta*[*]	*Significato*
	DELETE	rowId[1], userId	Risultato operazione	Elimina l'annuncio identificato da rowId
/imgPublication	POST	publicationId, userId, image	idImage	Inserisce un'immagine per la publicationId passata, userId necessario per la corretta identificazione dell'utente e del relativo privilegio CREATE sulla pubblicazione
	DELETE	idImage, publicationId, userId	Risultato Operazione	Elimina l'immagine identificata da idImage per la publication id passata, userId necessario per la corretta identificazione dell'utente e del relativo privilegio DELETE sulla pubblicazione

Tabella 1 'Servizi'

[*] In questa tabella si indica la risposta del servizio nello scenario di successo, per tutti gli altri casi di insuccesso si considera una risposta negativa che permette all'utilizzatore del servizio di capire quale parametro della richiesta è errato o quale errore lato server è stato generato a fronte dell'input dato.

Note 'Tabella 1':

[1] rowId: indica l'id della riga nel database, così da poter identificare correttamente la risorsa e procedere opportunamente con l'operazione

[2] Parametri publication: si distinguono in necessari→ idUtente, idProdotto,

latitudine, longitudine, città, dataPubblicazione, dataScadenza, e opzionali → indirizzo, descrizione (descrizione aggiuntiva rispetto a quella del prodotto data dall'utente), idServer. L'idServer identifica in maniera univoca la risorsa, questo parametro è opzionale e determina l'azione eseguita dal servizio in quanto se il parametro è presente si provvede a verificare che l'annuncio esista e se esiste si procede con la modifica; viceversa il servizio prosegue con l'inserimento di un nuovo annuncio.

[3] data, nuova data: utile per il versionamento, questo aspetto viene approfondito dovutamente al *paragrafo 3.3.1 'Tecniche di sincronizzazione server-client'*.

[4] lista dei prodotti: la lista dei prodotti contentente le informazioni di ogni prodotto; nello specifico per ogni prodotto si riporta → id, nome, immagine e descrizione.

[5] lista degli annunci: la lista degli annunci contenente le informazioni di ognuno; per ogni pubblicazione si ha→ id, latitudine, longitudine, città, indirizzo, descrizione, dataPubblicazione, dataScadenza, idUtente, idProdotto.

Ulteriori aspetti

Una scelta a livello di design come si può evincere dalla '*tabella 1'* è quella di offrire un meccanismo di paginazione per il servizio 'publication'; questo servizio risponde con la lista degli annunci di tutti gli altri utenti ordinando i risultati geograficamente; è opportuno quindi offrire una paginazione vista la quantità di dati chiamati in causa. Questo non si è fatto per il servizio 'publicationUser' che riguarda gli annunci del singolo utente, in quanto considerando il dominio applicativo un utente non avrà una quantità di annunci pubblicati tali da richiedere un meccanismo di paginazione.

3.3 Persistenza dei dati

Garantire persistenza dei dati è un requisito fondamentale e progettare un sistema

che offra dei buoni meccasismi di persistenza richiede attenzione. Per BlaBlaFood si richiede un database lato server ma un ulteriore requisito è l'uso parsimonioso delle risorse; questo porta a definire un metodo di memorizzazione lato client così da non vincolare l'utilizzo dell'applicazione alla presenza di rete dati; cosi facendo si hanno benefici anche nell'uso delle risorse lato server, visto che le richieste saranno inferiori grazie alla cache. Essendo il client un'applicazione Android, si hanno a disposizione tre principali metodi di memorizzazione: 1) Shared Preferences, 2) File, 3) database SQLite; per avere un accesso strutturato ai dati e un'organizzazione migliore degli stessi si decide di utilizzare un database SQLite per salvare le entità principali come Prodotti, Utenti e Annunci; le Shared Preferences saranno invece utilizzate per salvare dati utili al funzionamento corretto dell'applicazione, questo verrà approfondito nel *capitolo 4 Implementazione*.

I dati presenti nel database locale devono essere sincronizzati con quelli presenti nel database del server; si esaminano, al fine di progettare la sincronizzazione tra i due database, alcune tecniche di sincronizzazione.

3.3.1 Tecniche di sincronizzazione client-server

Si individuano tre principali tipi di sincronizzazione dati tra database locale e database online:

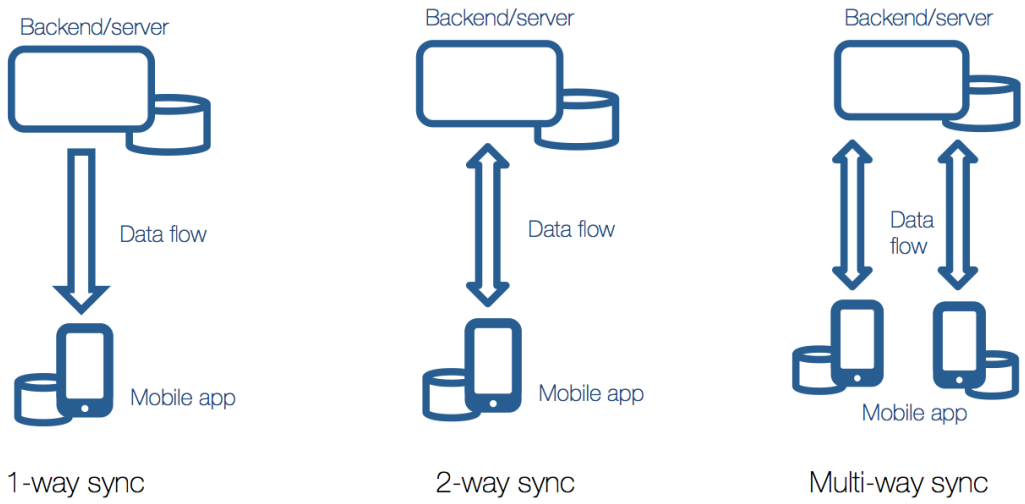

Figura 3.5 'Tipi di sincronizzazione fra database'

Sincronizzazione a una via: i dati vengono sincronizzati dal server verso i client (es. un'applicazione di news dove i contenuti sono sincronizzati dal backend verso i client) oppure dall'applicazione al server (es. logging, analytics).
Sincronizzazione a due vie: i dati vengono sincronizzati sia dall'app verso il backend che viceversa; un esempio può essere quello di un utente che è autenticato in un sito web e a un'applicazione e gestisce i suoi dati da entrambe; in questo tipo di sincronizzione si assume che l'utente non possa essere autenticato nello stesso momento dai due client.
Sincronizzazione multi-via: i dati sono sincronizzati da più client verso il server/backend e viceversa; l'utente può gestire i suoi dati da più device.

Generalmente oggigiorno un utente possiede più di un device, dare la possibilità di usufruire del servizio da ognuno di essi può attribuire maggiore qualità al sistema; è una scelta di design permettere agli utilizzatori di BlaBlaFood di usufruire del servizio da più device associati allo stesso account; nel caso specifico quindi si progetta un meccanismo di sincronizzazione multi-via.
Occorre precisare che, considerando il dominio applicativo, l'utente ha privilegi CRUD solo sui dati di sua proprietà, mentre, per tutti gli altri, ha solo privilegi di visualizzazione (R→READ); la progettazione della sincronizzazione è quindi

adattata al dominio, in quanto non è richiesta la stessa attenzione di un sistema multi-via (es. applicazioni collaborative) dove gli utenti possono avere privilegi CRUD anche su dati condivisi.

Si prosegue analizzando due approcci di aggiormento adottabili:

1) "aggiormento totale"

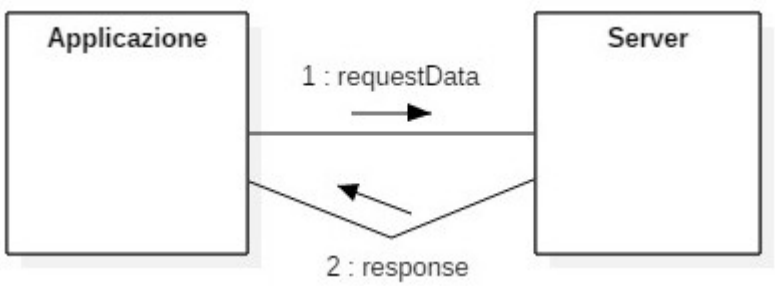

Figura 3.6 'Sincronizzazione: aggiornamento totale'

Ad ogni richiesta di aggiornamento, il server ritorna tutti i dati richiesti dall'applicazione salvati nel backend senza tener conto dei dati che il client potrebbe già possedere; si comprende che un metodo di questo tipo non è adatto per database di grandi dimensioni, risulterebbe svantaggioso in termini di consumo di rete dati con un conseguente uso delle risorse client e server poco adeguato. Il vantaggio nell' adottare meccanismi di questo tipo può risiedere nella semplicità di sviluppo del sistema, e può essere vantaggioso per sistemi di piccole dimensioni con database limitati in grandezza.

Per quanto riguarda il sistema BlaBlaFood è ragionevole non adottare questo metodo in quanto rientra nei requisiti un uso ottimizzato delle risorse e una buona esperienza utente, non soddisfabili seguendo un approccio di questo tipo.

2) "aggiornamento selettivo"

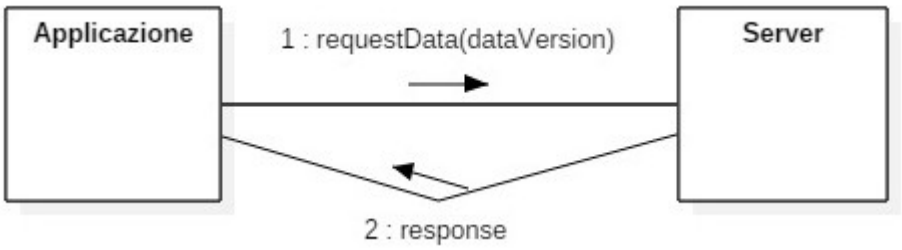

Figura 3.7 'Sincronizzazione: aggiornamento selettivo'

Un approccio selettivo permette di aggiornare solo i dati che hanno subito una variazione; occorre progettare un meccanismo di versionamento dei dati stessi, decidendo il criterio che permette di identificare un dato come aggiornato o da aggiornare.

A tal fine si associa ai dati salvati nel backend una marca temporale (*timestamp*), essa viene rinnovata al valore attuale a ogni cambiamento del dato stesso. Il client memorizza una data di versionamento che inserirà nelle richieste inviate al server; tale data sarebbe nulla se fosse la prima richiesta; il server seleziona i dati che hanno una marca temporale maggiore alla data presente come parametro nella richiesta; inoltre nella risposta del server è indicata anche la nuova data di versionamento che corrisponde al massimo timestamp tra i dati ritornati. Il client aggiornerà la cache e salverà la nuova data di versionamento inviata dal server, che sarà utile per la prossima richiesta. In questo modo l'applicazione si aggiorna correttamente e in maniera selettiva.

Seguendo questo approccio è il client che chiede al server l'aggiornamento, mentre è il server che valuta quali dati inviare; in questo modo si riesce a distribuire in maniera efficacie e efficiente le responsabilità tra i componenti. Si procede con ulteriori aspetti che richiedono attenzione.

Inserimento dei dati in condizioni offline

In assenza di rete dati si può scegliere di permettere o non permettere l'inserimento o modifica dei dati; non permettere l'inserimento sarebbe più facile ma considerando i requisiti emersi in analisi occorre per il sistema in questione

progettare opportunamente anche questo aspetto così da garantire una buona esperienza d'uso anche in assenza di rete dati.
I dati vengono prima inseriti o modificati localmente, in seguito si propaga l'operazione lato server quando le condizioni lo permettono; per fare questo si potrebbe inserire una tabella nel database locale specifica per questo, dove inseriamo ogni dato modificato offline; quando le condizioni lo permettono si sincronizzerebbe ogni dato presente nella tabella con il server. Si è preferito però adottare un approccio che semplifica la gestione: si aggiunge a ogni dato lato client un campo 'uploaded' che varrà '1' per i dati correttamente sincronizzati, '0' per quelli in attesa di sincronizzazione.

Cancellazione dei dati

Cancellare direttamente un dato dal backend non permetterebbe ai client di sincronizzarsi dovutamente e questo porterebbe a tutta una serie di problemi correlati da gestire.
Per evitare scenari di questo tipo, al fine di eseguire una corretta sincronizzazione, occorre effettuare inizialmente una delete logica del dato; la delete logica del dato si ottiene aggiungendo un colonna 'deleted' alle tabelle del database lato server, questa conterrà di default '0' che ha significato 'non deleted'; nel momento in cui un client richiede la cancellazione del dato (sul quale ha privilegio DELETE) questo campo viene settato a '1', a indicare che quel dato è stato cancellato; nel settaggio del valore deleted a '1' si aggiorna, come per le altre operazioni di modifica, il timestamp così da permettere ai client di individuare correttamente il cambiamento; gli altri client riceveranno il dato con flag deleted a '1' e provvederanno a cancellare la copia locale.
Questo approccio nel corso del tempo porterebbe ad avere molti dati eliminati logicamente, con conseguente aumento della sparsità degli stessi, in quanto i dati non verrebbero mai concretamente eliminati; per evitare questo, occorre inserire una 'delete fisica' dei dati.
Si procede impostando una regola di invalidità della cache lato client, si noti il

seguente diagramma:

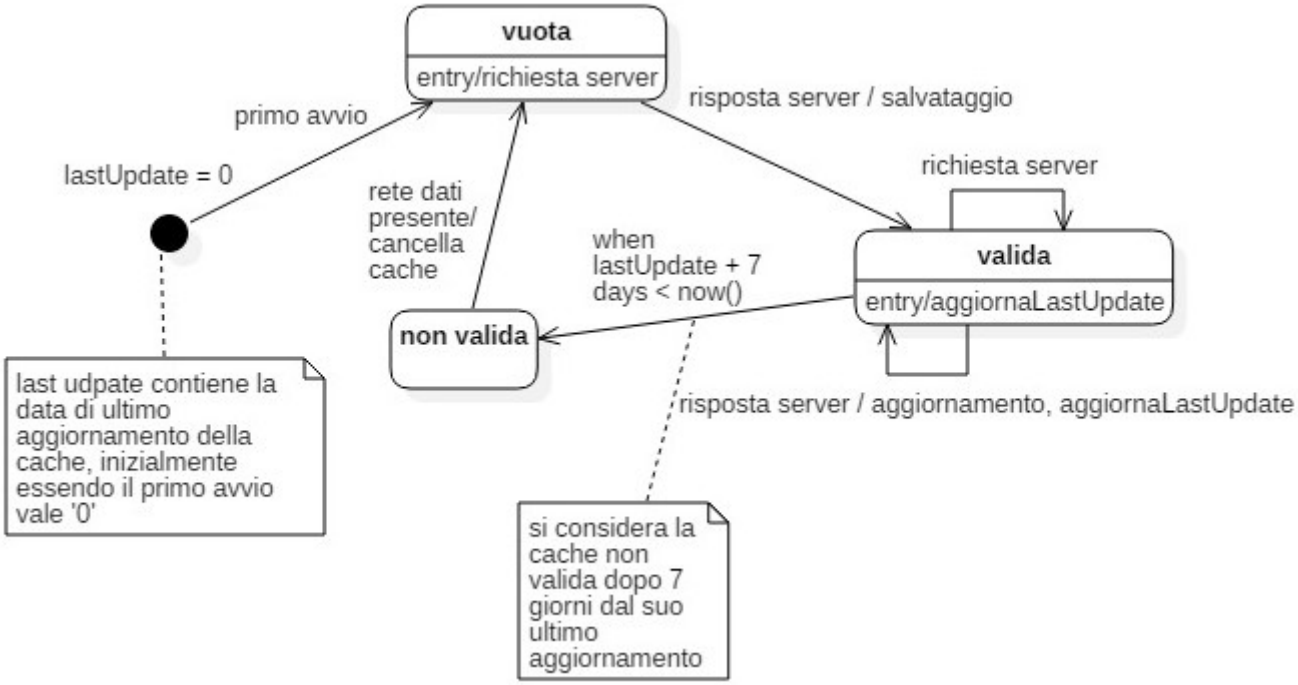

Figura 3.8 'Diagramma di stato: cache locale'

Come si può notare dal diagramma, si identificano tre stati principali per la cache: vuota, valida, non valida. Si mantiene un indicatore lastUpdate che contiene la data di ultimo aggiornamento, quando questa indica che l'ultimo aggiornamento è stato eseguito più di 7 giorni fa rispetto alla data attuale, la cache viene considerata non valida; se la rete dati è presente si procede con la cancellazione e la richiesta al server dei dati.

Grazie all' impostazione di questa regola di validità della cache, il server può effettuare delle delete fisiche dei dati, senza compromettere la sincronizzazione dei client; per farlo correttamente avendo la garanzia di una corretta sincronizzazione del sistema, occorre eliminare solo i dati con timestamp più vecchio di 7 giorni; in questo modo si riesce a limitare la sparsità dei dati, si hanno benefici anche per quanto riguarda l'occupazione di memoria lato server, e il consumo di rete.

Seguendo le osservazioni fatte, si procede con la progettazione del database server

e client.

3.3.2 Database del server

Il database è composto da 4 tabelle:

Utenti: è la tabella contenente gli utenti registrati al servizio; un utente si registra mediante social, durante il processo di registrazione il social ritorna un id univoco che identifica l'utente; il server genera un token univoco di registrazione interno usato dal sistema per identificare l'utente (per maggiori dettagli si riporta al *paragrafo '3.4 Login e registrazione degli utenti'*). Si procede con la spiegazione di alcuni campi della tabella:

- socialId, nomeSocial: rispettivamente indicano id univoco ritornato dal social network durante la registrazione e nome del social network utilizzato; la coppia socialId e nomeSocial permette di identificare univocamente gli utenti così da evitare, per esempio, doppie registrazioni.
- tokenIdClient: identificatore, anch'esso univoco, generato dal server
- telefono: essendo un dato personale, si è valutato con attenzione se usarlo come informazione di contatto; considerando il dominio e la tipologia di utente si pensa che sia una buona scelta utilizzare il numero di telefono. Altri sistemi come Subito.it utilizzano anch'essi il telefono o la mail per mettere in contatto gli utenti del sistema. Allo stesso modo si è deciso di fare per BlaBlaFood.
- dataVers: utile per il versionamento dei dati come spiegato al paragrafo precedente.
- deleted: utile per la cancellazione nel rispetto della corretta sincronizzazione come spiegato al paragrafo precedente.

Prodotti: è la tabella contenente tutti i prodotti che l'utente può selezionare e inserire in un annuncio.

- ImagePath: è l'url univoco che identifica la risorsa

- Descrizione: descrizione del prodotto

ProdottoPubblicato: contiente tutti gli annunci degli utenti

- descrizione: è la descrizione aggiuntiva inseribile dall'utente associata all'annuncio
- idProdotto: indica l'id del prodotto associato a questo annuncio
- idUtente: indica l'utente che ha pubblicato l'annuncio

ImmaginiPubblicazione:

- imagePath: url univoco che indirizza alla risorsa
- idProdottoPubblicato: è il riferimento all'annuncio, indica a quale annuncio è associata l'immagine

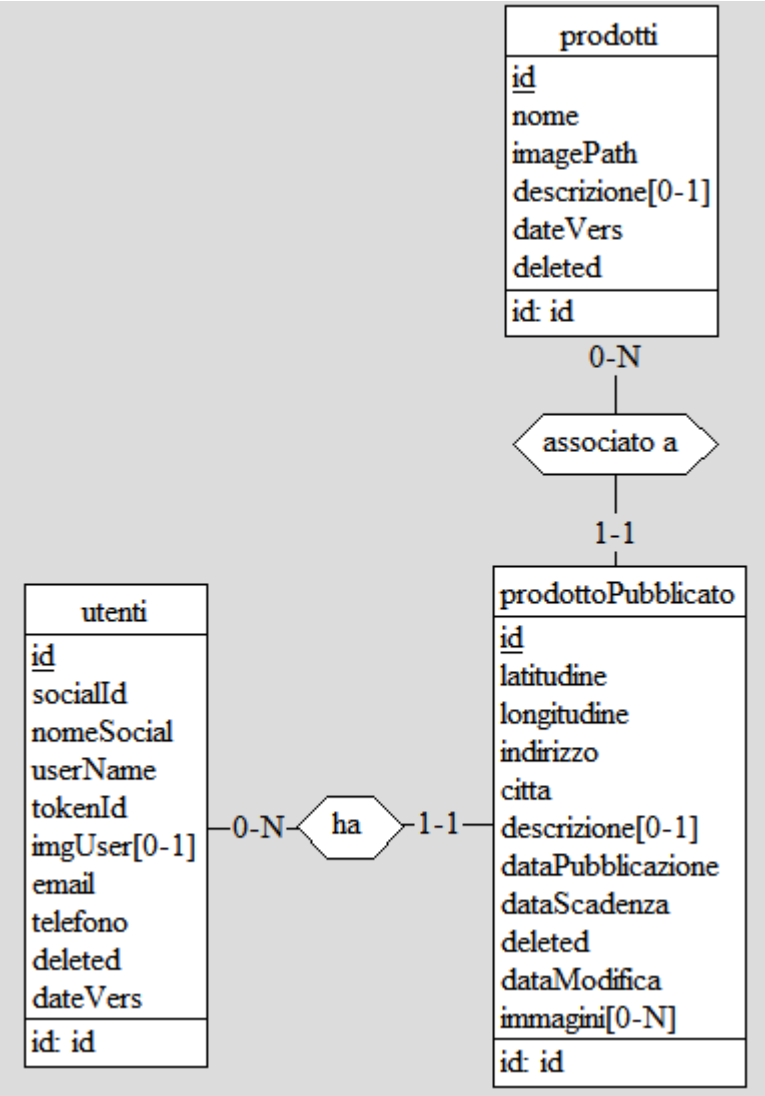

Figura 3.8 'database server: Schema E/R'

Lo schema sopra riporta la progettazione concettuale; a un prodotto pubblicato (annuncio o pubblicazione) è associato un utente e un prodotto; opzionalmente possono essere presenti delle immagini inserite dall'utente identificate nello schema dall'attributo 'immagini[0-N]' che nella progettazione logica verrà opportunamente gestito.

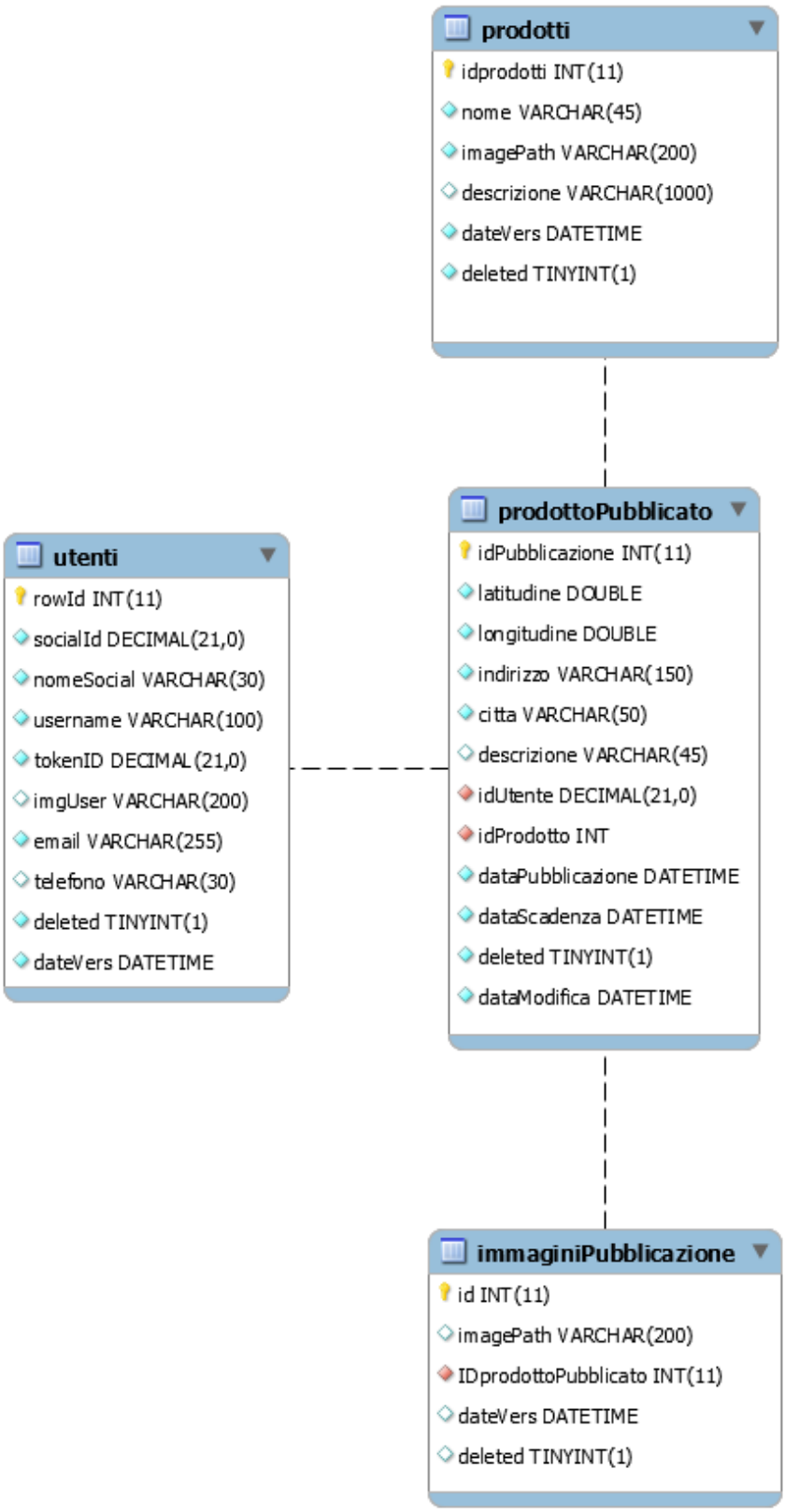
prodotti
idprodotti INT(11)
nome VARCHAR(45)
imagePath VARCHAR(200)
descrizione VARCHAR(1000)
dateVers DATETIME
deleted TINYINT(1)
utenti
rowId INT(11)
socialId DECIMAL(21,0)
nomeSocial VARCHAR(30)
username VARCHAR(100)
tokenID DECIMAL(21,0)
imgUser VARCHAR(200)
email VARCHAR(255)
telefono VARCHAR(30)
deleted TINYINT(1)
dateVers DATETIME
prodottoPubblicato
idPubblicazione INT(11)
latitudine DOUBLE
longitudine DOUBLE
indirizzo VARCHAR(150)
citta VARCHAR(50)
descrizione VARCHAR(45)
idUtente DECIMAL(21,0)
idProdotto INT
dataPubblicazione DATETIME
dataScadenza DATETIME
deleted TINYINT(1)
dataModifica DATETIME
immaginiPubblicazione
id INT(11)
imagePath VARCHAR(200)
IDprodottoPubblicato INT(11)
dateVers DATETIME
deleted TINYINT(1)

Figura 3.9 'database server: modello relazionale'

Nel modello relazionale sopra riportato sono state tradotte le associazioni; la tabella prodottoPubblicato ha ora due attributi idUtente, idProdotto foreign key rispettivamente della tabella Utenti e Prodotti. L'attributo 'immagini[0-N]' della tabella prodottoPubblicato è stato tradotto con la tabella immaginiPubblicazione che ora ha l'attributo idProdottoPubblicato, foreign key della tabella prodottoPubblicato.

3.3.3 Database del client

Per quanto riguarda il database lato client, la struttura è la stessa del database del server. Si hanno delle differenze nei tipi di dato in quanto SQLite supporta text, numeric, integer, real, blob. Si sostituisce opportunamente varchar(n) con text, double con real, tinyint(1) con integer.

3.4 Login e registrazione degli utenti

Per poter usufruire delle funzionalità dell'applicazione gli utenti devono registrarsi; possono farlo mediante social network: Facebook o Google+; generalmente gli utenti sono restii a fornire dei dati al primo avvio di un'applicazione che non conoscono, la registrazione mediante social network permette di effettuare la registrazione in maniera semplice e veloce, evitando l'inserimento di dati in una prima fase delicata come questa.
Essendo la registrazione obbligatoria, al fine di dare una buona esperienza d'uso, si decide di inserire al primo avvio un tutorial consultabile dall'utente, terminata la sua visualizzazione viene mostrata all'utente la vista di registrazione/login. Questo è l'approccio seguito da molti fornitori di servizi che pongono attenzione a questi aspetti di UX. Si descrive attraverso il seguente schema il flusso di navigazione in questione:

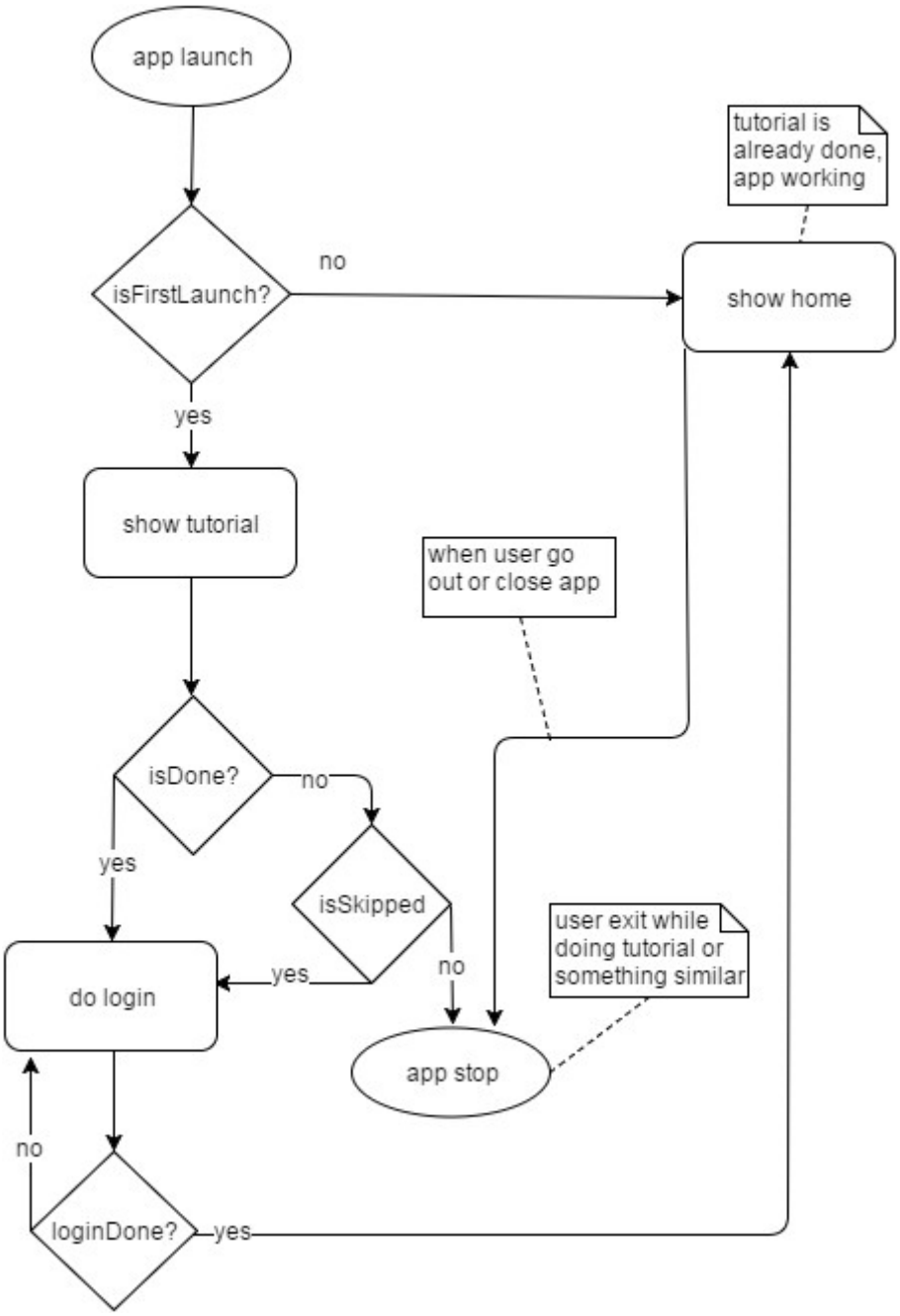

Figura 3.10 'Flusso di navigazione: Avvio'

Come si nota nella figura l'utente visualizza il tutorial al primo avvio, esso può essere 'saltato' così da visualizzare direttamente la view di Login, altrimenti si arriverà ad essa quando il tutorial termina.

Si illustra il processo di registrazione evidenziano la comunicazione tra client e server attraverso il seguente diagramma di sequenza:

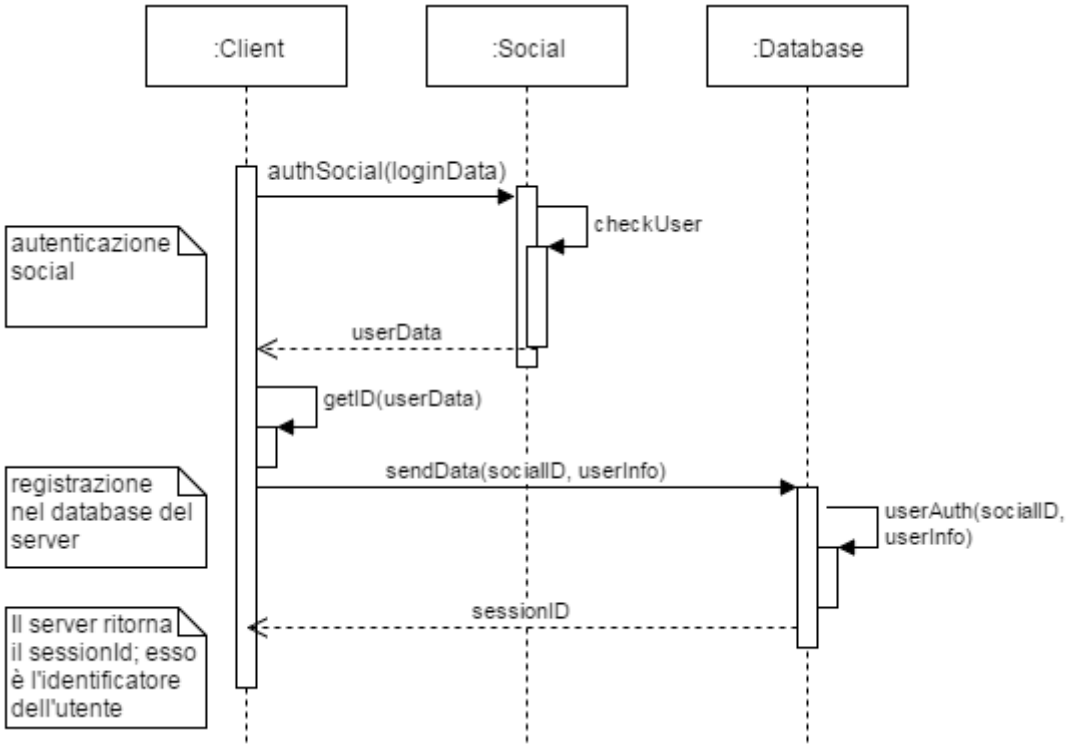

Figura 3.10 'Diagramma di sequenza: registrazione utente'

Nel diagramma si può notare la comunicazione tra utente, social network e backend; l'utente esegue il login mediante social e al termine di questa operazione procede con la comunicazione verso il backend attraverso una richiesta HTTP; usando il metodo GET i dati verrebbero inseriti nella forma chiave → valore nell'URL della richiesta; essendo informazioni di registrazione si è preferito utilizzare il metodo POST, mediante il quale si inseriscono i dati nel corpo del messaggio. Il servizio registra l'utente nel backend, genera un session id, in maniera opportuna (*vedi Capitolo 4 Implementazione paragrafo 4.1.2* per verificare come avviene la generazione dell'id seguendo accorgimenti alla sicurezza informatica), che viene ritornato in risposta all'applicazione in formato JSON; si fa notare che non viene mantenuta una sessione lato server, ma il client includerà in ogni richiesta il sessionID attraverso il quale si identifica l'utente.

Si mostra ora il wireframe relativo al login allo scopo di evidenziare la struttura della vista:

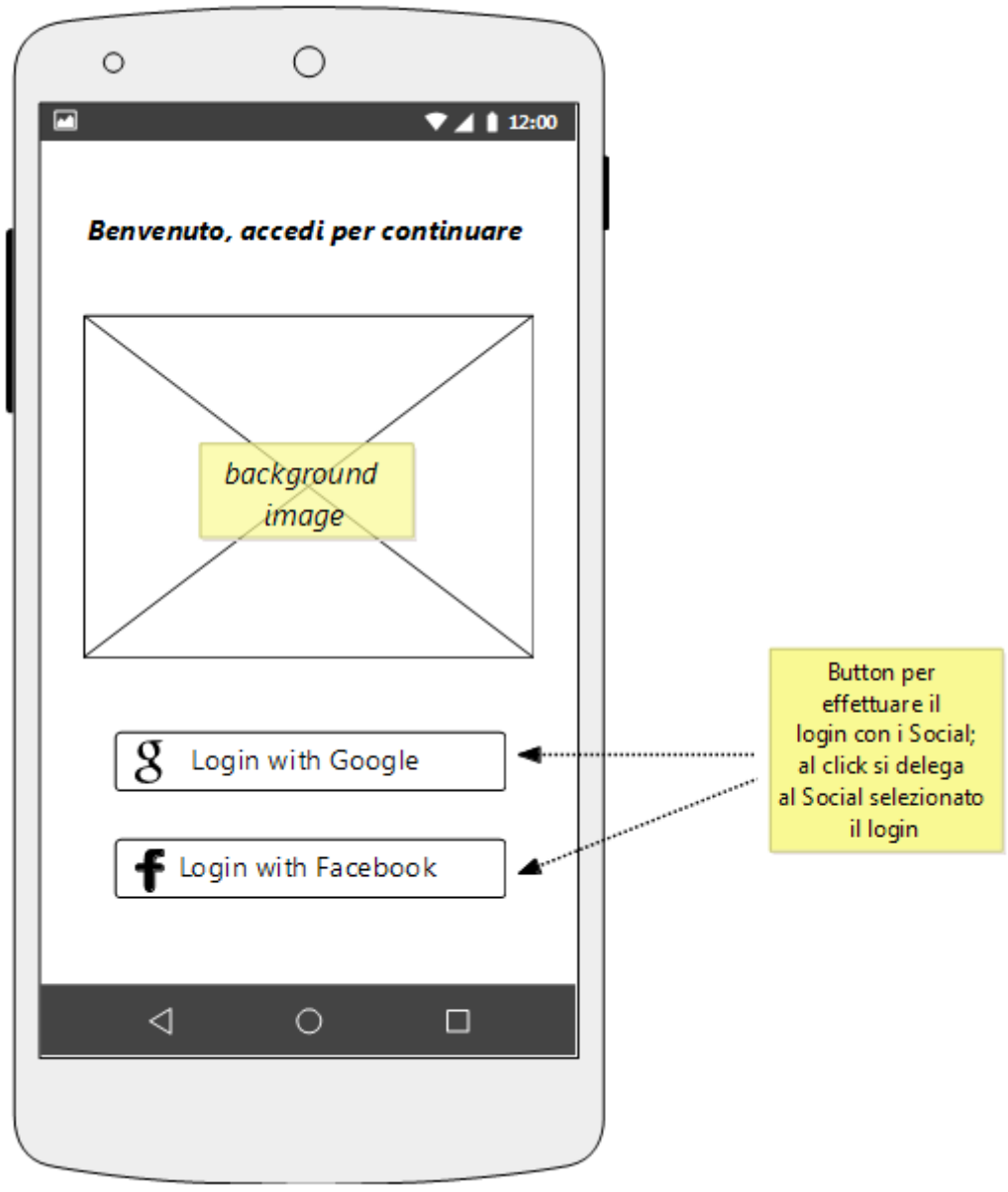

Figura 3.11 'Wireframe Login'

3.5 Annunci

Gli annunci sono l'elemento principale, occorre quindi permettere all'utente di

visualizzarli, inserirli, modificarli e cancellarli in modo intuitivo e veloce. Si mostra primariamente la *home* dell'applicazione:

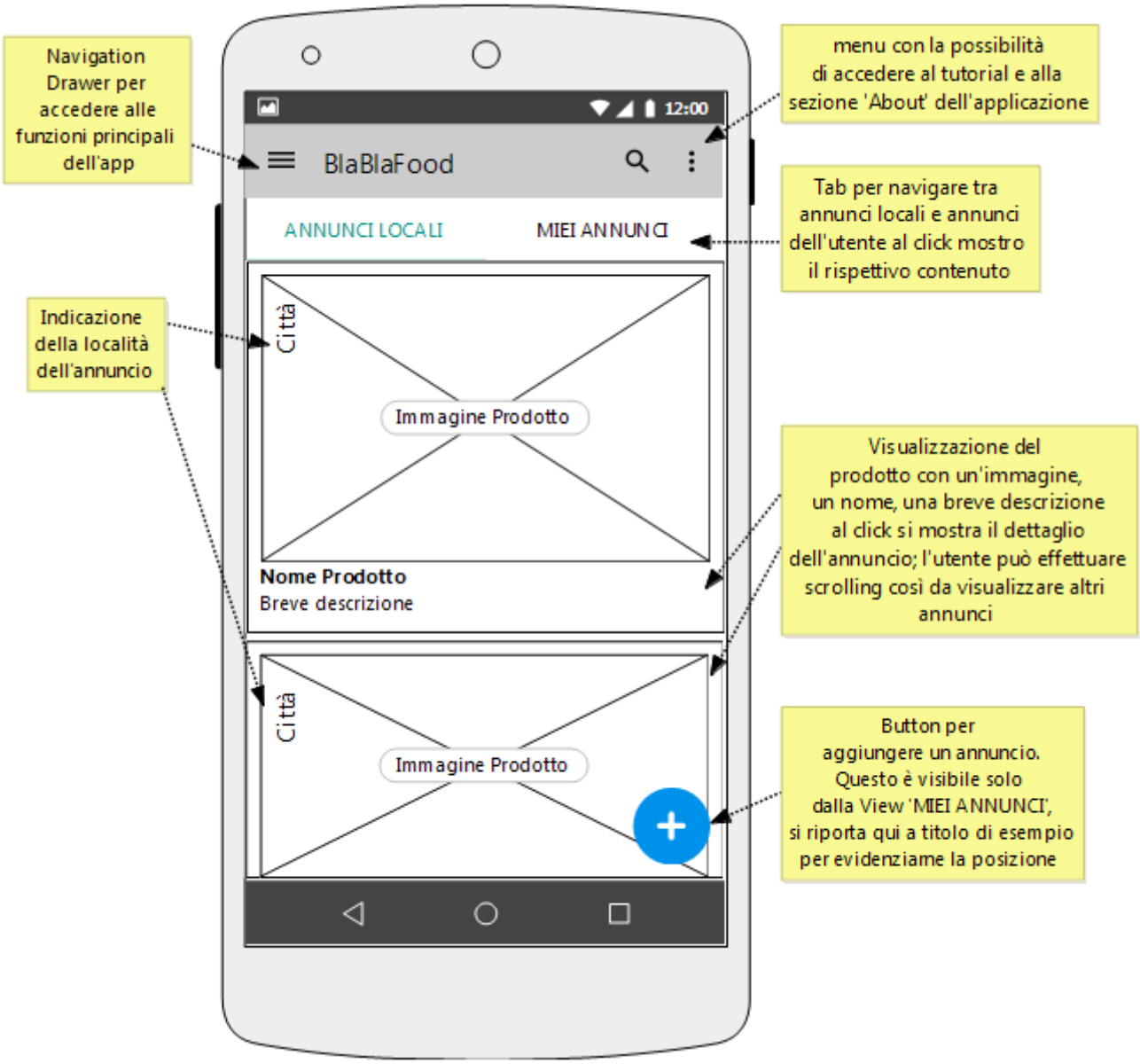

Figura 3.12 'Wireframe Home'

3.5.1 Visualizzazione ordinata geograficamente

In aggiunta alle note dela figura 3.12, si evidenzia che la visualizzazione degli annunci nella View 'Annunci Locali' è ordinata geograficamente rispetto alla

posizione dell'utente. Questo è fondamentale in quanto rientra in uno degli scopi principali di BlaBlaFood quello di permettere agli utilizzatori di conoscere quali sono i prodotti offerti localmente. L'ordinamento degli annunci, come è stato già trattato nel *paragrafo 3.1.2 Modello e entità*, è ottenuto confrontando gli attributi latitudine e longitudine, dell'annuncio stesso con gli altri annunci. Si descrive formalmente attraverso il seguente schema il processo di ordinamento:

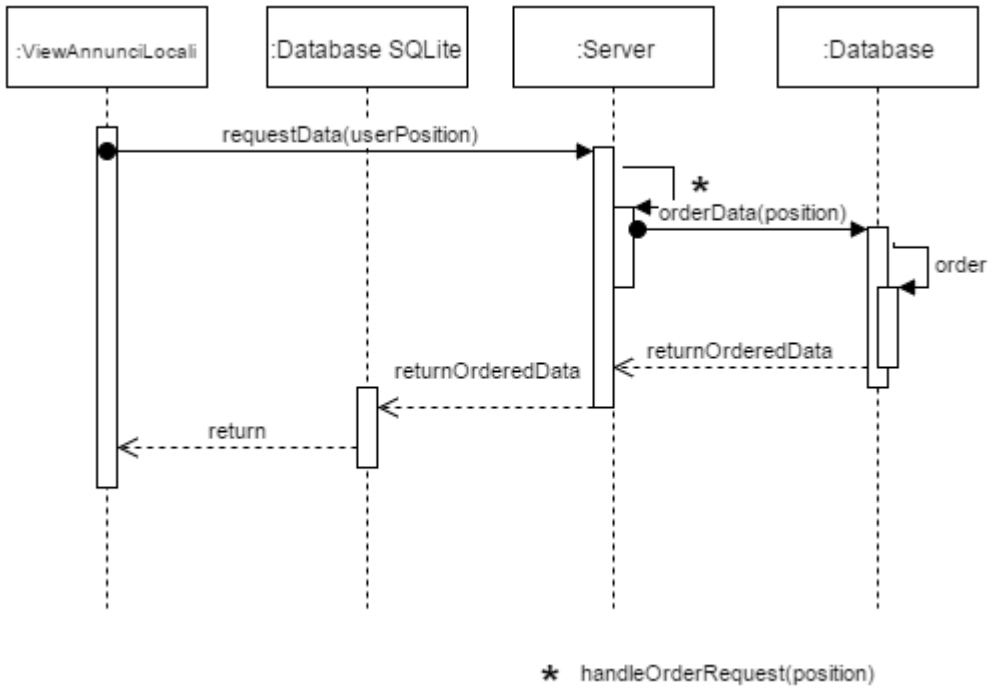

Figura 3.13 'Diagramma di sequenza: ordinamento annunci'

Come si può notare dal diagramma di sequenza sopra riportato, l'ordinamento è effettuato dal database lato server e non dal client; il client richiede al servizio web specifico (in questo caso */publication, vedi tabella 1 Servizi al paragrafo 3.2 Servizi Web per ulteriori dettagli sul servizio*), gli annunci ordinati, inviando la posizione dell'utente; il servizio gestisce la richiesta effettuando un'opportuna interrogazione al database. I dati ritornati al client vengono salvati nella cache locale e visualizzati ordinati all'utente.

Al fine di offrire una buona esperienza d'uso, il caricamento degli annunci avviene

in maniera incrementale; l'utente effettua scroll per visualizzare gli annunci, una volta che viene raggiunto il fondo della vista, altri contenuti vengono caricati; questo è comunemente identificato come endless scroller o infinite scroll.
Per quanto riguarda gli annunci dell'utente visualizzabili dalla View '*Miei Annunci*', per questi non è necessario un ordinamento; inoltre essendo la quantità di questi annunci minore, non si ha un meccanismo di endless scrolling.

3.5.2 Inserimento, modifica, cancellazione

Come già enfatizzato in precedenza, l'inserimento, la modifica e la cancellazione degli annunci sono azioni effettuabili dal proprietario dello stesso; queste operazioni sono eseguibili anche in condizioni offline.
Si mostra con il seguente diagramma di sequenza l'interazione tra i componenti coinvolti nel processo di inserimento, lo stesso schema è applicabile anche alla modifica e alla cancellazione.

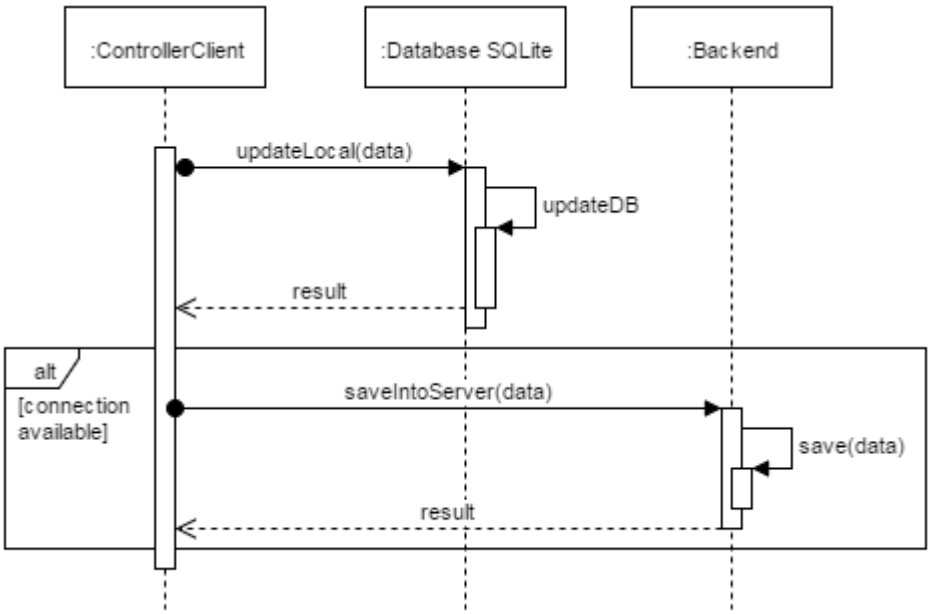

Figura 3.14 'Diagramma di sequenza: Create, Update, Delete'

Come si può notare dal diagramma sopra, il controller del client è responsabile del coordinamento delle operazioni di salvataggio locali e sul server. Per un corretto funzionamento si effettua un salvataggio nel database locale; successivamente se la connessione è disponibile si procede con il salvataggio nel database del server.

La comunicazione tra Client e Server avviene sempre attraverso richieste HTTP e le risposte del Server sono in formato JSON. Il servizio utilizzato è /publicationUser per ulteriori approfondimenti sui parametri della richiesta, e dati inseriti nella risposta, si faccia riferimento alla *tabella 1 Servizi al paragrafo 3.2 Servizi Web.*

Nell'inserimento dell'annuncio l'utente può aggiungere delle immagini, rispetto a quella fornita automaticamente con il prodotto selezionato; l'upload delle immagini e della pubblicazione è un'operazione di rete non breve; per questo occorre progettare attentamente l'upload e quali componenti di Android utilizzare per effettuarlo; questo aspetto viene discusso nel *capitolo di Implementazione.*

Si mostra ora il wireframe relativo all'inserimento dell'annuncio:

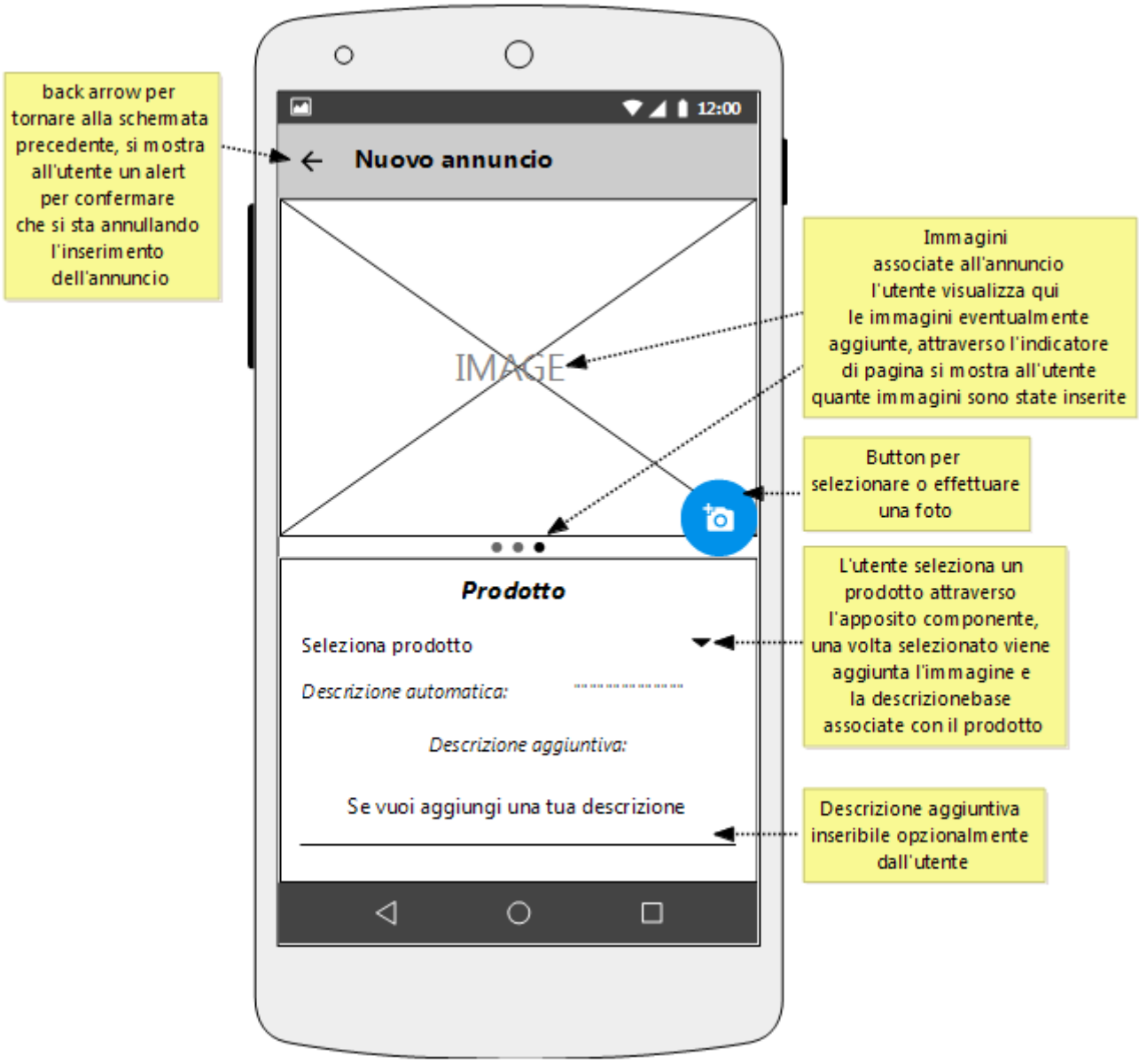

Figura 3.15 'Wireframe Aggiunta annuncio parte 1/2'

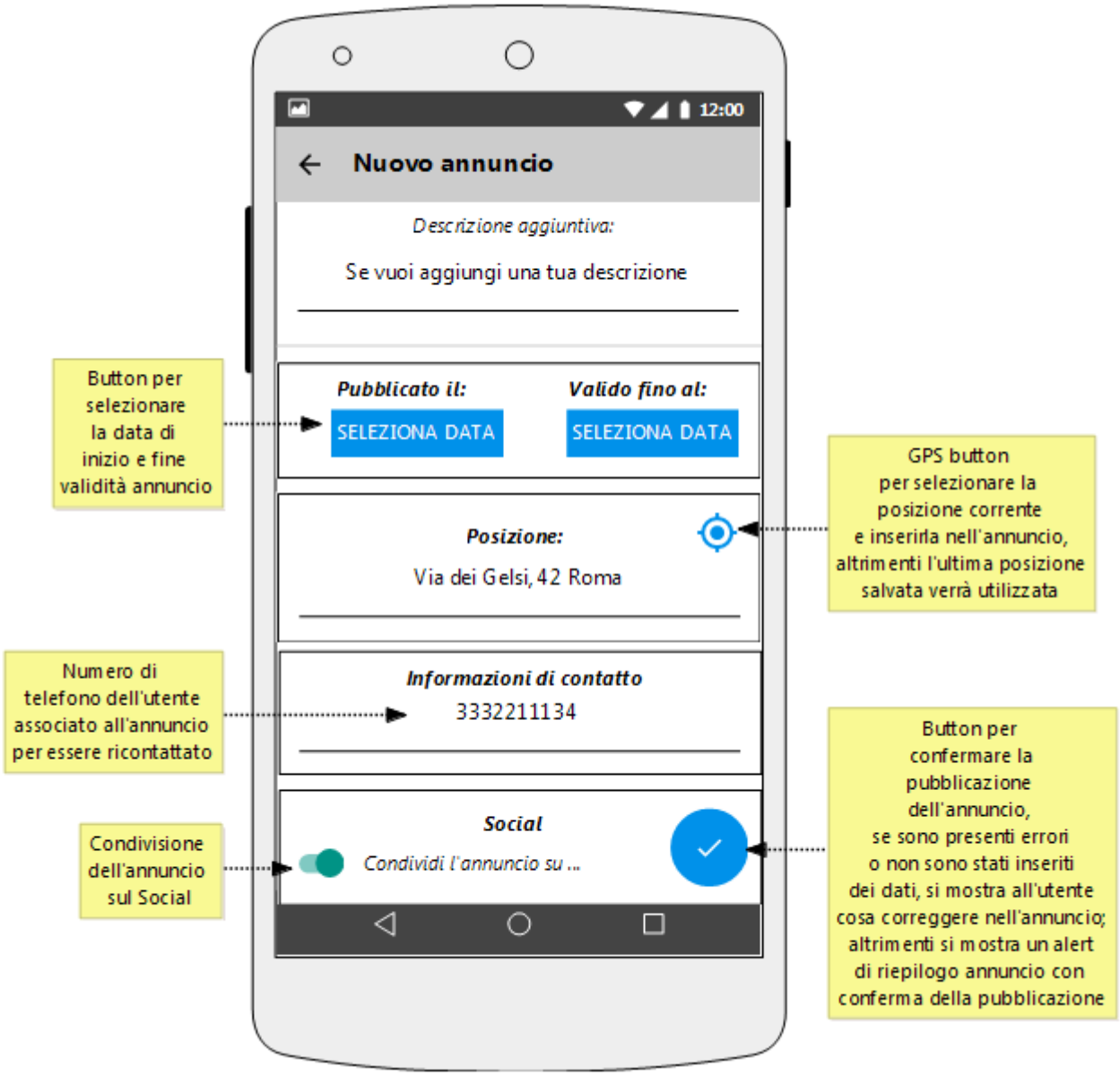

Figura 3.16 'Wireframe aggiunta annuncio parte 2/2'

Il prodotto associato all'annuncio, viene selezionato dall'utente attraverso l'apposito componente; il catalogo prodotti non è gestito dagli utenti, questo permette di dare uniformità agli annunci e offrire un'immagine e una descrizione automaticamente una volta selezionato il prodotto. La compilazione è così velocizzata, infatti l'utente deve selezionare un prodotto, inserire l'intervallo di

validità dell'annuncio, una posizione se non è già presente o deve essere cambiata, un numero di telefono se non è già presente o deve essere cambiato, e indicare se vuole postare sul Social l'avvenuta pubblicazione; opzionalmente possono essere inserite ulteriori immagini e una descrizione aggiuntiva.

3.5.3 Dettaglio

Si mostra il wireframe di dettaglio dell'annuncio:

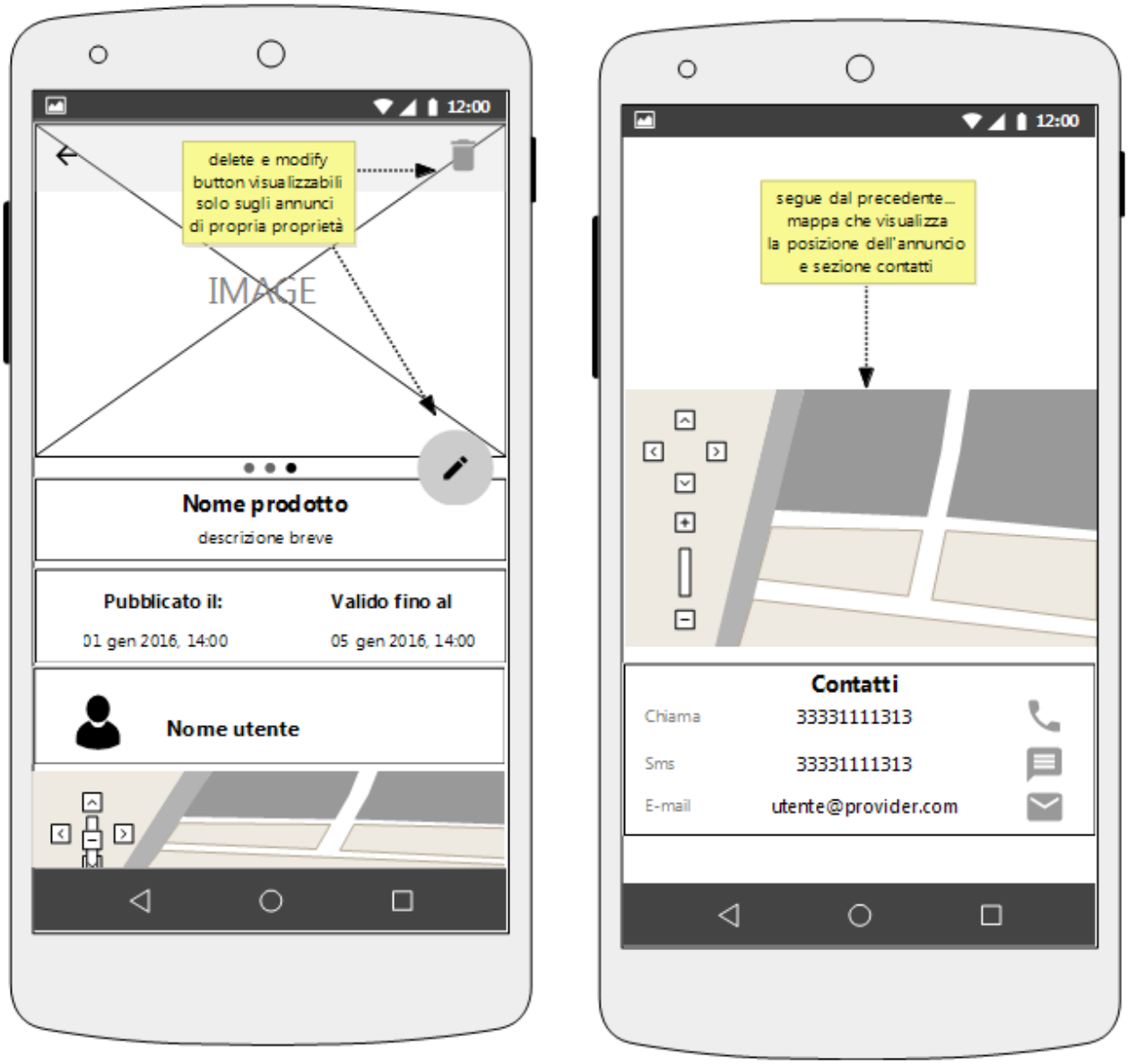

Figura 3.17 'Wireframe dettaglio annuncio'

Come si può notare dal wireframe sopra riportato, la visualizzazione è costituita dalle seguenti parti:

- *Cosa?* : ovvero l'oggetto principale dell'annuncio, il prodotto che l'utente offre.
- *Quando?* : sezione che indica l'intervallo di validità dell'annuncio
- *Chi?* : chi offre il prodotto ovvero l'utente che lo ha pubblicato; al click si naviga verso il profilo dell'utente, con la possibilità di visualizzare gli altri annunci da lui pubblicati.
- *Dove?* : la posizione dell'offerente, visualizzabile attraverso l'apposita mappa.
- *Come?* : le modalità di contatto disponibili, telefono, sms o e-mail.

3.6 Flussi di navigazione

Si riepilogano, infine, i flussi di navigazione dell'applicazione, citati nei paragrafi precedenti, attraverso il seguente schema:

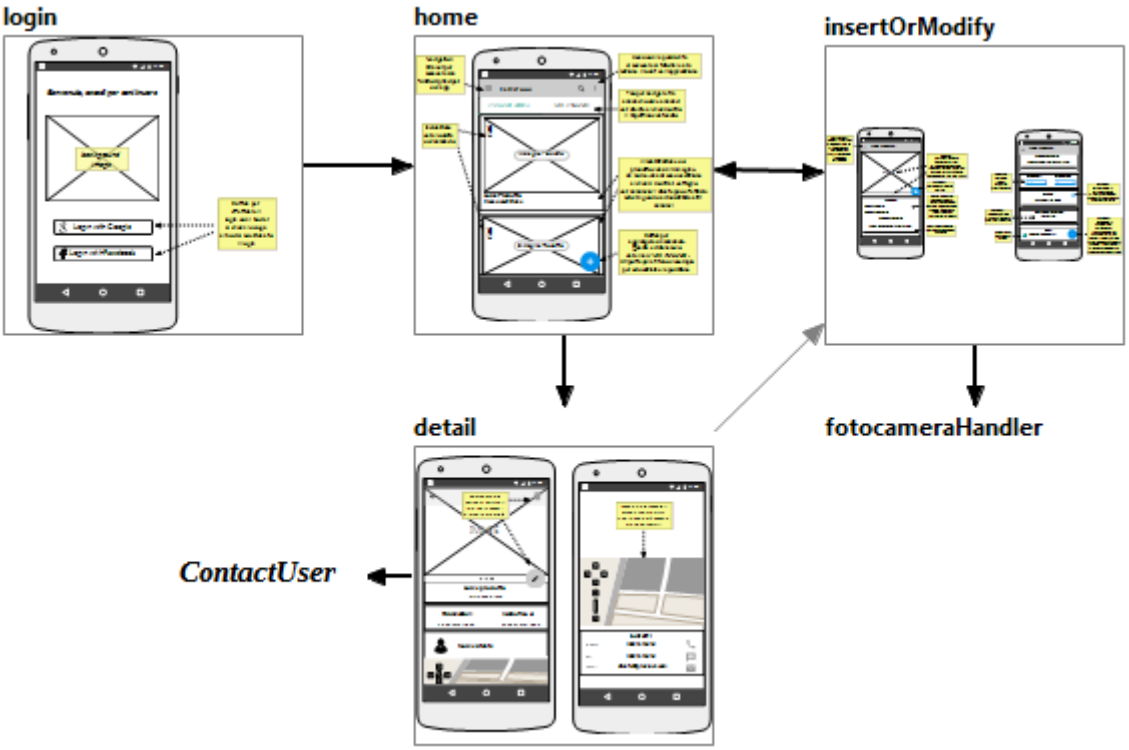

Figura 3.18 'Flussi di navigazione'

3.7 Integrazione con i Social

BlaBlaFood permette agli utenti di effettuare il login mediante Social Network, nello specifico Google o Facebook come si è argomentato in precedenza; al fine di offrire una maggiore integrazione con le reti sociali, si offre anche la possibilità di condividere l'avvenuta pubblicazione di un annuncio.

3.7.1 Facebook SDK

Facebook offre la possibilità di condividere attraverso dei modelli prestabiliti: *Link, Foto, Video, Contenuti Multimediali, Notizie Open Graph.* Occorre modellare i contenuti da contividere e successivamente aggiungere un'interfaccia di condivisione attraverso la quale l'utente può effettuare lo sharing.

3.7.2 Google API

Una feature offerta dai servizi Google è la condivisione su social Google+; come

per Facebook, si ha la possibilità di condividere attraverso un'interfaccia di condivisione dei contenuti come *testo, foto, link e posizioni*; inoltre è facilmente inseribile anche del *testo precompilato*.

4 Implementazione

Si procede con la realizzazione del sistema, questo comprende sia la parte server che client. Si evidenzieranno i linguaggi, le tecnologie e le scelte implementative fatte, considerando il dominio specifico e motivandole opportunamente.

4.1 Server

Come si è trattato nel *capitolo 3 Progettazione,* l'architettura dell'applicativo è di tipo client-server; si realizza primariamente la base del sistema per offrire persistenza dei dati, ovvero il database; si procede poi con l'implementazione dei servizi.

4.1.1 Implementazione database server

Si è scelto un DBMS di tipo relazione, nello specifico MySQL; la decisione che ha portato a selezionare questo DBMS tra quelli disponibili nel mercato è basata principalmente sui seguenti aspetti:

- MySQL è un prodotto Open Source rilasciato con doppia licenza GPL e commerciale, questo significa che può essere utilizzato senza costi nella sua versione Community Edition, che non limita il servizio offerto; liberamente si può passare in un secondo momento alle versioni commerciali se le circostanze lo richiedono. Scegliere questo DBMS permette di evitare determinati vincoli, per esempio di licenze o sistemi operativi specifici. Questo permette al sistema BlaBlaFood di avere una buona base, che può soddisfare le richieste di un bacino di utenti eventualmente crescente (*si veda il capitolo 6 Conclusioni, sviluppi futuri e ringraziamenti* per comprendere lo stato attuale del sistema e eventuali prossimi sviluppi). [15]
- Per l'implementazione dei servizi web si è scelto PHP; questo ha motivato

ulteriormente l'utilizzo di MySQL, in quanto la coppia PHP-MySQL attualmente è tra le più diffuse nel settore, entrambi sono cross-platform è ben integrati, infatti PHP offre diverse funzioni per interagire con i database MySQL.

La creazione del database è stata eseguita basandosi sul modello relazionale evidenziato in progettazione; inoltre i dati vengono gestiti dai servizi nel rispetto delle politiche di sincronizzazione. Si riporta la query che ha richiesto più attenzione, al fine di evidenziarne degli aspetti rilevanti.

Selezioni degli annunci ordinati rispetto alla posizione geografica :

```
SELECT *, 6371 * 2 * ASIN(SQRT( POWER(SIN(('lat' -
latitudine)*pi()/180/2),2)+COS('lat'*pi()/180 )
*COS(`latitudine`*pi()/180)*POWER(SIN(('long'-
longitudine)*pi()/180/2),2))) as distance
FROM prodottopubblicato
WHERE longitudine
between ('long'-100/cos(radians('lat'))*111.044736)
          and ('long'+100/cos(radians('lat'))*111.044736)
          and latitudine
between ('lat'-(1/111.044736))
          and ('lat'+('rangeKm'/111.044736))
          and idUtente <> 'userId'
          and  dateVers > "dateVers"
having distance < 'rangeKm'
ORDER BY distance limit 'limit' offset 'offset'
```

Per ottenere l'ordinamento degli annunci rispetto alla posizione dell'utente si è utilizzata una formula trigonometrica; si descrive brevemente l'utilizzo, non approfondendo nel dettaglio aspetti geometrici o correlati.

La formula utilizzata è denominata, 'Formula dell'emisenoverso', o in inglese, Haversine formula; questa è una formula in trigonometria sferica utile alla navigazione; determina la distanza tra due coordinate, in questo modo si riesce a confrontare le coordinate dell'annuncio rispetto alla posizione dell'utente, in modo da ottenere l'ordinamento. Si è scelto di usare questa formula dopo aver fatto ricerche specifiche sull'argomento; alcuni fattori positivi relativi alla formula sono l'efficacia e l'efficienza della stessa come si può leggere da questo *paper* [8]; l'unica negatività si potrebbe trovare nel fatto che la formula approssima in

maniera errata vicino ai poli, ma considerando il dominio applicativo, è un aspetto largamente accettabile.

4.1.2 Implementazione dei servizi web

La scelta di PHP come linguaggio di scripting può essere riassunta in aggiunta a quanto detto in precedenza nei seguenti punti principali:

- Ottima integrazione con MySQL
- Disponibilità di API per la conversione di oggetti in formato JSON
- L'esecuzione dei servizi non necessita di un web server, come sarebbe stato necessario per esempio se si avesse scelto Java; infatti i servizi, sono dei file in PHP hostati sul server, che in questo caso risiede nella piattaforma di Altervista.

I servizi sono stati realizzati seguendo la *Tabella 1 'Servizi'* esposta nel *capitolo 3 Progettazione*. Si evidenziano alcune parti ritenute significative con relativa spiegazione:

Conversione dei dati e preparazione per l'invio al Client:

```
<?php
function deliver_response($success, $status_message, $data,
$dateVers){
    $response['success']= $success;
    $response['status_message']= $status_message;
    $response['data']= $data;
    //data per il "versionamento dei dati"
    $response['dateVers']=$dateVers;

$json_response=json_encode($response, JSON_UNESCAPED_SLASHES);
echo $json_response;
}
?>
```

La risposta al Client viene preparata utilizzando la function json_encode alla quale si passano i dati da convertire in formato JSON; tra i valori della risposta si invia:

- Risultato richiesta, ovvero il successo o insuccesso

- Un messaggio di stato, ai fini di una maggiore comprensione della risposta
- I dati richiesti, che corrispondo al risultato dell'operazione effettuata dal servizio sul database
- La data di versionamento, utile per la sincronizzazione tra server e client come discusso in precedenza

Generazione del sessionId

Ulteriore attenzione richiede la realizzazione del servizio di registrazione degli utenti; come descritto in precedenza, al termine del login mediante Social si ha un id univoco che identifica l'utente; questo viene salvato nel backend così da poter identificare correttamente l'account BlaBlaFood associato all'utente; occorre precisare che utilizzare questo identificativo per successive comunicazioni tra client e server non è sicuro. A questo proposito si genera un sessionId lato server che il client utilizzerà per identificarsi come si è evidenziato in progettazione. La generazione di questo id deve essere fatta considerando i seguenti aspetti:

- univocità dell'identificatore, per evitare problemi di identificazione tra gli utenti; avere un id non univoco significherebbe che due utenti diversi potrebbero essere considerati identici per il sistema.
- sicurezza nella generazione, essendo l'identificatore dell'utente

Dopo aver effettuato ricerche in merito, si è giunti alla conclusione di non utilizzare la funzione built-in di PHP uniqid(); questo perché la funzione offerta da PHP, come si legge dalla documentazione ufficiale, non garantisce univocità e può presentare problemi a livello di sicurezza. Anche se la probabilità di collisione di due valori generati con uniqid() è molto bassa si è preferito avere garanzia al riguardo così da mostrare piena attenzione ai punti sopra elencati.

Le ricerche effettuate hanno condotto alla scelta della seguente funzione che si riporta brevemente. Con la seguente si riesce a generare un id univoco con un livello di sicurezza accettabile per il dominio in questione

```php
<?php
//generazione di un id univoco
//si specifica che la funzione non è scritta dal sottoscritto
//si faccia riferimento per la fonte della funzione a
//stackoverflow.com e alla documentazione ufficiale di php
function random_text( $type = 'alnum', $length = 8 )
{
        switch ( $type ) {
                case 'alnum':
                        $pool =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
                        break;
                case 'alpha':
                        $pool =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
                        break;
                case 'hexdec':
                        $pool = '0123456789abcdef';
                        break;
                case 'numeric':
                        $pool = '0123456789';
                        break;
                case 'nozero':
                        $pool = '123456789';
                        break;
                case 'distinct':
                        $pool = '2345679ACDEFHJKLMNPRSTUVWXYZ';
                        break;
                default:
                        $pool = (string) $type;
                        break;
        }


        $crypto_rand_secure = function ( $min, $max ) {
                $range = $max - $min;
                // not so random...
                if ( $range < 0 ) return $min;
                $log    = log( $range, 2 );
                // length in bytes
                $bytes  = (int) ( $log / 8 ) + 1;
                $bits   = (int) $log + 1; // length in bits
                // set all lower bits to 1
                $filter = (int) ( 1 << $bits ) - 1;
                do {
                        $rnd = hexdec(
bin2hex( openssl_random_pseudo_bytes( $bytes ) ) );
                        // discard irrelevant bits
                        $rnd = $rnd & $filter;

                } while ( $rnd >= $range );
                return $min + $rnd;
        };
```

```
        $token = "";
        $max   = strlen( $pool );
        for ( $i = 0; $i < $length; $i++ ) {
                $token .= $pool[$crypto_rand_secure( 0, $max )];
        }
        return $token;
}?>
```

Un esempio di consumo della funzione:

```
$sessionId = random_text(numeric, 16);  //generation of unique and secure id
```

In questo modo si ha un id numerico univoco di 16 cifre, questo è il session id per lo specifico utente.

Selezione della data di versionamento

Per i servizi che ritornano un insieme di valori (es. servizi con metodo GET) si devono effettuare le seguenti macro operazioni:

- selezione dei dati richiesti
- selezione della data di versionamento

Ai fini di una corretta sincronizzazione tra i client e il server occorre prestare attenzione al selezionamento della data di versionamento. Il servizio esegue prima la ricerca della data massima, e successivamente seleziona i dati dal database; seguendo questo ordine si evitano scenari di errata sincronizzazione; infatti se si selezionassero prima i dati e poi la data di versionamento, ogni dato inserito nel database da altri client tra la prima richiesta (riferita ai dati) e la seconda (riferita alla data di versionamento) andrebbe perso.

Gestione degli annunci

Ulteriore attenzione richiede la gestione degli annunci e delle immagini a essi associate; si sono rispettati i seguenti aspetti:

- Ogni modifica sull'annuncio comporta l'aggiornamento della data di

versionamento del dato stesso.

- Ogni cambiamento nella tabella *immaginipubblicazione (*le immagini dell'annuncio), comporta l'aggiornamento della marcatura temporale dell'immagine e dell'annuncio a cui essa fa riferimento.
- L'eventuale cancellazione del dato non può essere effettuata direttamente, come argomentato in progettazione, ma occorre in prima fase effettuare una cancellazione 'logica' dell'annuncio. Successivamente verrà effettuata l'eliminazione effettiva nel rispetto delle politiche di sincronizzazione del sistema. Per una corretta propagazione della cancellazione, occorre eliminare anche le immagini associate all'annuncio; si effettuano quindi due cancellazioni 'logiche': la prima riferita all'annuncio, la seconda sulle eventuali immagini aggiuntive associate ad esso; in entrambe si aggiorna la 'dateVers' così da indicare ai client l'avvenuto cambiamento.

4.2 Client

La progettazione relativa al Client delineata nel capitolo tre, è nella quasi totalità indipendente dalla specifica implementazione in Android; la scelta di Android come piattaforma di sviluppo è sicuramente stata influenzata dal fatto di non possedere un computer con sistema operativo Mac OS; questo non elimina la possibilità di sviluppare in seconda fase anche un'applicazione per dispositivi con sistema operativo iOS come si parlerà nel capitolo *6 'Conclusioni, sviluppi futuri'*.
Per lo sviluppo si è usato l'IDE ufficiale *Android Studio* e come linguaggio *Java*.

4.2.1 Organizzazione progetto

L'applicazione sviluppata in Android è organizzata nei seguenti package principali:

- ***Model,*** contenente le classi wrapper delle entità principali *Product, User, Publication, Image.*
- ***Controller,*** interfaccia *IController* e sua implementazione *Controller.*

- ***Database,*** contenente la classe *PublicationDbOpenHelper* e *PublicationDBManager.*
- ***Service,*** servizi utili per il funzionamento dell'applicazione, classe *Service* per operazioni di delete, e aggiornamento della cache locale, classe *ServiceCUD* per operazioni di modifica, upload di annunci e immagini correlate.
- ***Receiver,*** contenente i gestori dei risultati dei servizi.
- **Login,** contenente LoginActivity e LoginFragment.
- ***Publication,*** contenente le activity e fragment riferiti all'entità annuncio, ovvero *AddOrEditPublicationActivity*, *PublicationDetailsActivity* *WorldPublicationFragment* (fragment degli annunci locali), *MyBoardFragment* (fragment degli annunci dell'utente).
- ***Exception,*** eccezioni lanciabili dall'applicazione come *SaveLocalException, o SaveServerException.*

Nel package base, oltre a contenere ciò che è sopra elencato, si hanno le activity *HomeActivity, TutorialActivity, ProfileActivity* e la classe *Utilities*.

Nel corso dei prossimi paragrafi si tratteranno più dettagliatamente ogni activity, fragment e servizi.

Dopo aver descritto brevemente l'organizzazione, è doveroso procedere evidenziando degli aspetti riguardanti l'architettura.

4.2.2 Enfasi sull'architettura

Per ottenere un software di qualità occorre prestare attenzione a molti aspetti, tra questi l'impostazione di una buona architettura, che determina la struttura dell'applicativo, è un fattore da curare nel dettaglio; si noti il seguente diagramma dove si riportano i componenti principali e le loro relazioni:

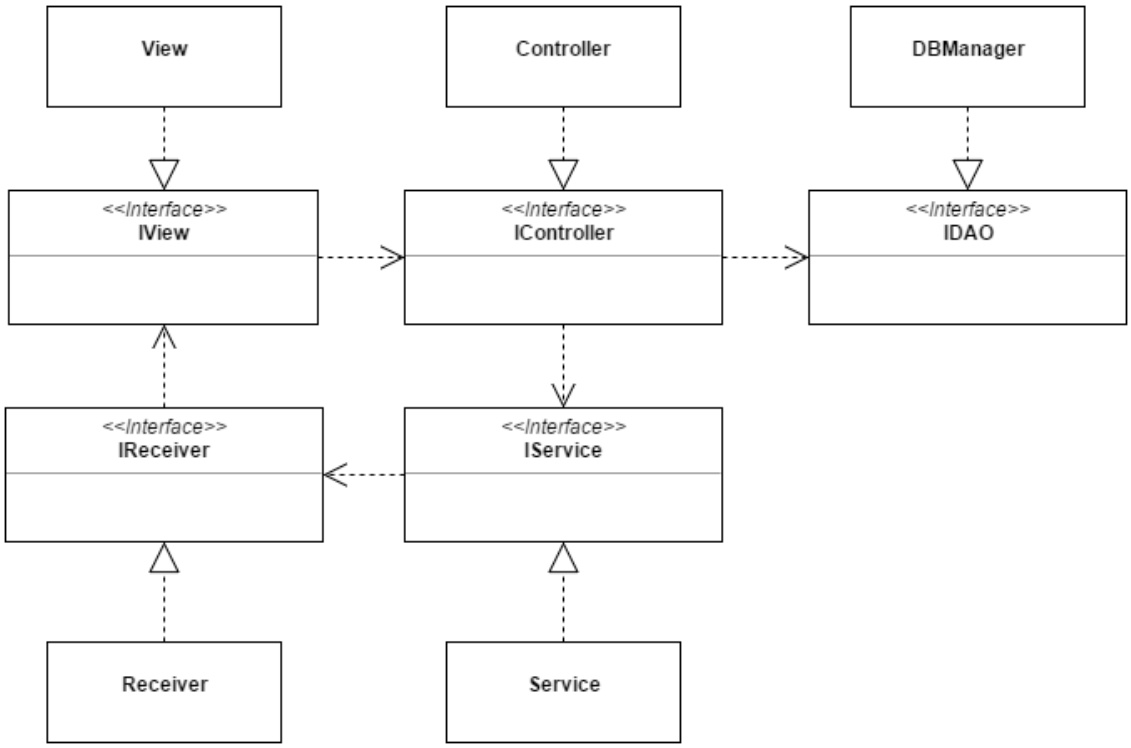

Figura 4.1 'Diagramma delle classi: architettura'

Aderendo al pattern architetturale MVC, come si può notare dal diagramma, i componenti principali sono i seguenti:

- *View*, con View si intende l'insieme dei componenti, come si è soliti individuare in Android, per l'interfaccia grafica. Si noti che le activity stesse possono essere viste come controller. Le viste interagiscono con l'utente e comunicano con il Controller, effettuando richieste opportune a fronte di eventi.
- *Controller*, è l'intermediario, principale responsabile della separazione tra dati e presentazione; il controller gestisce i dati attraverso l'interfaccia IDAO; grazie a questo non ci sono dipendenze tra controller e modalità di salvataggio dei dati, in quanto il controller fa riferimento al contratto di IDAO non curandosi dell'implementazione specifica che è delegata a chi realizza l'interfaccia IDAO, ovvero il DBManager. Ulteriore responsabilità del controllore è la delegazione ai servizi delle operazioni di rete.
- *Manager del database,* realizza l'interfaccia IDAO, gestisce il database

locale; questo evita l'inserimento dell'interazione con il database in ogni cliente, per esempio nelle Activity o Fragment.

- *Service,* componente Android progettato per eseguire lunghe operazioni, senza provvedere un'interfaccia utente (UI); ha il compito di gestire le richieste del controllore, rispondendo ai rispettivi receiver; si precisa che Service è una classe astratta, per la scelta della classe concreta e altri aspetti correlati si veda il *paragrafo 4.2.11 Background processing.*
- *Receiver,* gestiscono i risultati dai rispettivi Service e notificano ai componenti della View l'operazione da eseguire; nello specifico si utilizzano ResultReceiver, si faccia riferimento al *paragrafo 4.2.11 Background processing* per ulteriori dettagli.

Si descrive attraverso il seguente diagramma la comunicazione tra i componenti per migliorare la comprensione e la leggibilità dell'architettura:

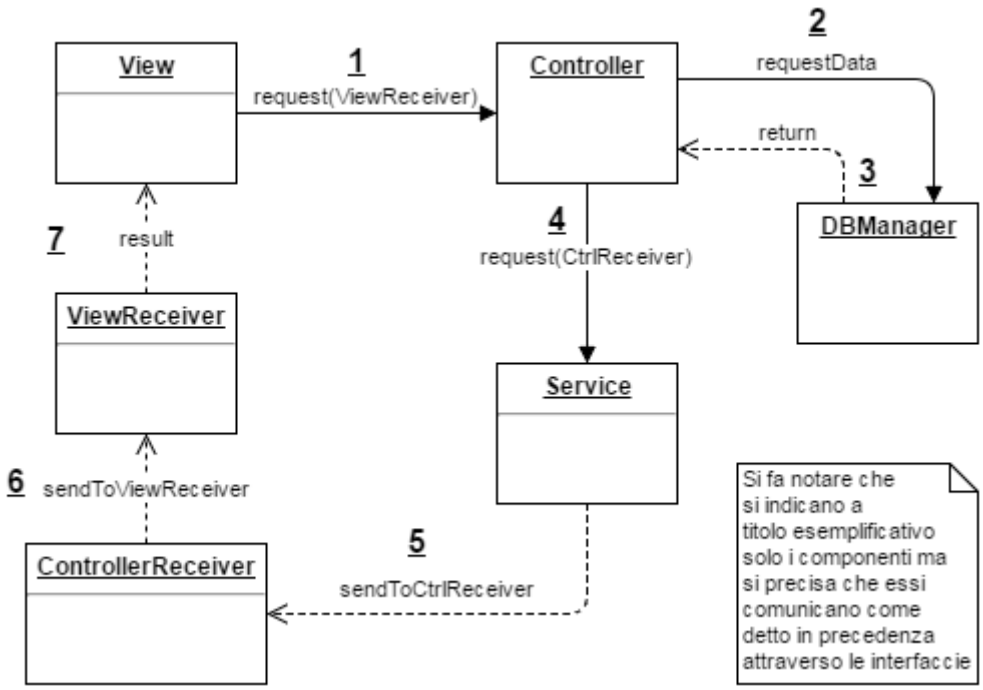

Figura 4.2 'Scenario di comunicazione tra i componenti'

Si descrive utilizzando la numerazione esposta in figura un possibile scenario generico di comunicazione tra i componenti. La *View* effettua richieste al

Controller passando il riferimento del *ViewReceiver* (1)*;* il Controller gestisce la richiesta, si possono distinguere questi casi:

- Cache vuota, o invalida: questo implica un'operazione di rete; il *Controller* effettua la richiesta al servizio passando il riferimento del *ControllerReceiver* (4); il *Service* gestisce la richiesta opportunamente comunicando verso il backend via HTTP; in seguito interpreta la risposta ricevuta dal Server e invia al *ControllerReceiver* il risultato dell'operazione (5); il *ControllerReceiver* segnala al *ViewReceiver* (6), che a sua volta ritornerà il risultato al chiamante iniziale ovvero la *View* (7).
- Richiesta di aggiornamento: come al punto precedente occorre proseguire delegando al servizio l'operazione (4) e di seguito il flusso sarà come descritto sopra.
- Cache presente e valida: il *Controller* comunica con il *DBManager* richiedendo i dati salvati localmente (2); il *DBManager* gestisce la richiesta e risponde (3); il *Controller* ritornerà al chiamante (in questo caso la *View),* la risposta alla sua richiesta effettuata inizialmente.

Nello scenario descritto, il *Controller* verifica opportunamente che la rete dati sia presente, se questa è necessaria per effettuare l'operazione; il controllo avviene mediante il *ConnectivityManager* presente nelle API di Android.

4.2.2.1 Pattern utilizzati

L'utilizzo di pattern di programmazione contribuisce al miglioramento della qualità del codice, aumentandone la leggibilità, la munutenibilità e l'estendibilità.
Oltre a quelli già presenti in Android si descrivono i principali pattern alla quale si è aderito:

Singleton

Intento di questo pattern è garantire che una classe abbia un'unica istanza, provvedendo un unico punto di accesso globale. Si è ritenuto utilizzare questo pattern per la classe *Controller* e *DbOpenHelper,* la motivazione risiede

nell'intento stesso del pattern.

Facade

La *HomeActivity* costituisce il controllore della View principale dell'applicazione. Su di essa è stato utilizzato il pattern strutturale su oggetti Facade, in questo modo si riesce a dare un accesso facilitato alle funzionalità delle altre View che la *HomeActivity* contiene. A beneficiarne è anche l'esperienza d'uso in quanto si ha un facile e diretto accesso alle funzionalità principali offerte dall'applicazione. Nel seguente diagramma si evidenzia la struttura costituita:

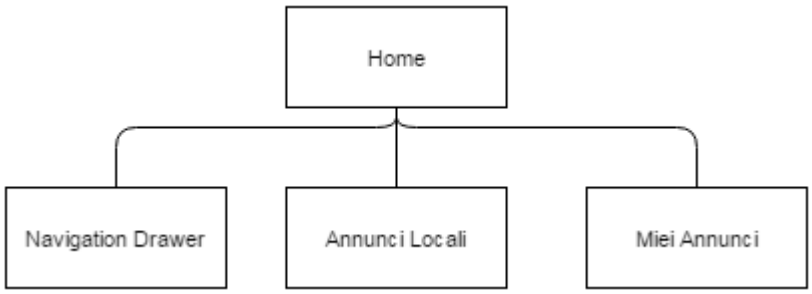

Figura 4.3 'Struttura View'

Come si può notare la Home contiene la view degli 'Annunci Locali', la view 'Miei Annunci' e il Navigation Drawer; 'Annunci Locali' e 'Miei Annunci sono stati realizzati mediante *Fragment* così da aumentare ulteriormente la riusabilità.

Template Method

Template Method è un pattern comportamentale su classi, permette di definire lo scheletro di un algoritmo delegando l'implementazione specifica alle sottoclassi. A tal proposito si è utilizzato, per esempio, nella classe *AbstractService*, si noti il seguente diagramma:

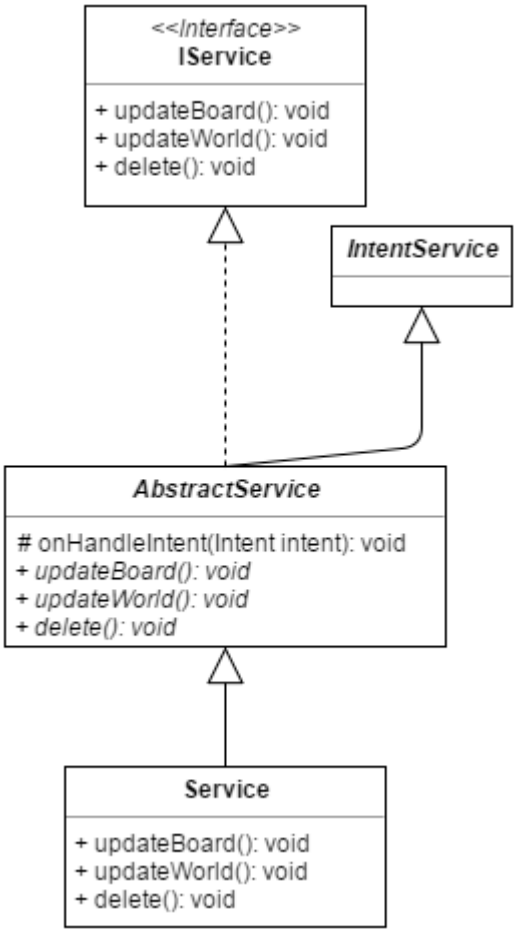

Figura 4.4 'Template Method '

AbstractService estende IntentService e realizza l'interfaccia IService; il metodo onHandleIntent è un template method in quanto utilizza i metodi astratti *updateBoard(), updateWorld(), delete()*. Service estende AbstractService è implementa i metodi usati nel template method.

4.2.3 Database Locale e gestione dati

La realizzazione del database in SQLite è costituita da due classi:

- *PublicationDbOpenHelper* classe che estende SQLiteOpenHelper, contiene i nomi delle tabelle e delle relative colonne; queste stringhe sono pubbliche e statiche in questo modo gli utilizzatori del database avranno un'unico punto di riferimento per conoscere questi valori ovvero la classe

stessa *PublicationDbOpenHelper;* metodi principali presenti in questa classe sono quelli riguardandi la creazione delle tabelle, la cancellazione e l'upgrade del db.

- *PublicationDBManager:* a questa classe è affidata, come argomentanto in precedenza, la gestione locale dei dati. La classe realizza il contratto di *IDAO* è ha un riferimento privato al PublicationDbOpenHelper; si indicano alcuni metodi principali dell'interfaccia IDAO che la classe realizza:
    - *insertOrUpdate(Publication publication),* modifica, se già presente nel db, l'annuncio passato altrimenti lo inserisce opportunamente; la modifica avviene delegando al metodo *updatePublication*, invece l'aggiunta mediante il metodo *addPublication*; questi metodi gestiscono anche l'inserimento di eventuali immagini aggiuntive associate all'annuncio passato chiamando il metodo *insertOrUpdateImage.*
    - *insertOrUpdate(User user),* inserimento o modifica di un utente.
    - *insertOrUpdate(Product product),* inserimento o modifica di un prodotto.
    - *logicalDeletePublication(int idPublication),* eliminazione logica di un annuncio dalla cache locale; l'eliminazione logica lato client è necessaria per offrire all'utente la possibilità di cancellare gli annunci di sua proprietà in condizioni di assenza di rete; inoltre il metodo può essere usato per offrire meccanismi di undo-redo sulla cancellazione, questo al fine di migliorare l'esperienza d'uso.
    - *DeletePublication (Publication publication),* effettua la cancellazione del dato dal db locale in modo permanente.
    - *getAllProductType*, seleziona dal database tutti i prodotti presenti e lì ritorna al chiamante.
    - *getAllUser,* analogamente al metodo precedente ma per l'entità utente.

- *getPublicationsWorld(int limit, int offset),* seleziona e ritorna gli annunci salvati localmente riferiti agli altri utenti; limit e offset sono utili per la paginazione.
- *getPublicationUser(int idUser),* seleziona e ritorna gli annunci salvati localmente riferiti a uno specifico utente.

Alcuni dati utili al funzionamento dell'applicazione sono salvati nelle *SharedPreferences,* come ad esempio, il sessionId, il controllo se il tutorial è stato eseguito, oppure la data di ultimo aggiornamento e versionamento della cache per il controllo di validità della stessa.

4.2.4 Realizzazione dell'interfaccia IController

La classe Controller realizza il contratto esposto in IController; il ruolo del controller è stato già argomentato precedentemente, si procede descrivendo i principali metodi implementati dalla classe Controller:

- *List<Publication> getPublicationUser(double id, ResultReceiver receiver, boolean watchCacheDirectly):* attraverso questo metodo gli utilizzatori della classe Controller ottengono gli annunci di uno specifico utente identificato dal parametro id; se il parametro watchCacheDirectly è true allora il Controller provvede a effettuare una richiesta verso il backend e il flusso di comunicazione tra i componenti avviene come argomentato nel *paragrafo 4.2.2 Enfasi sull'architettura.*
- *List<Publication> getPublicationWorld(ResultReceiver receiver, int offset, int limit, boolean watchCacheDirectly):* analogamente al precedente ma per gli annunci degli altri utenti; mediante i parametri offset e limit si offre il meccanismo di paginazione.
- *deletePublication(ResultReceiver receiver, Publication publication):* effettua la cancellazione di un annuncio gestendo l'aggiornamento verso il backend e della cache locale.
- *addPublication(Publication publication):* aggiunge un annuncio, gestendo

opportunamente l'aggiunta nel database locale e lato server, delegando al servizio *ServiceCUD*.

- *modifyPubblication(Publication publication, List<String> imagesDeleted, List<String imageAdded):* permette di modicare un annuncio; attraverso i parametri imagesDeleted si indicano le immagini eventualmente cancellate durante la modica e con imageAdded quelle eventualmente aggiunte; se l'annuncio è già stato inviato al server, occorre aggiornare attraverso il ServiceCUD il database online; se l'annuncio ha il campo upload con valore false vuol dire che l'annuncio non è ancora online, perciò si procede modificando solamente il database locale e ritentando l'invio al server se la rete è disponibile.
- *startApplication():* questo metodo è utilizzato all'avvio dell'applicazione; effettua il controllo di validità sulla cache, provvedendo al rinnovamento totale della stessa, se fosse necessario e se la rete dati è presente; ulteriore controllo effettuato dal metodo è sulla validità degli annunci dell'utente, se un annuncio è scaduto viene eliminato automaticamente avvisando opportunamente prima l'utente.
- Altri metodi presenti in IController e implementati dalla classe Controller sono quellli relativi all'accesso diretto alla cache locale per le entità *utente, prodotti, immaginiAnnunci.*

4.2.5 Implementazione meccanismo di sincronizzazione client-server

Per quanto riguarda la sincronizzazione lato client, occorre gestire il salvataggio delle date di versionamento ritornate dal server, da includere nelle richieste successive ai servizi. Nello specifico vengono memorizzate la *dataVers dei prodotti*, indicante l'ultimo aggiornamento locale effettuato su di essi, *dataVers degli annunci* utente e degli *annunci locali* di proprietà degli altri utenti. Per dettagli su come queste date vengono usate nel meccanismo di sincronizzazione si riveda il *paragrafo 3.3.1 'Tecniche di sincronizzazione Client-Server'*.

4.2.6 Gestione permessi

Dalla versione Android 6.0 (livello API 23), gli utenti concedono i permessi all'applicazione a tempo di esecuzione, e non all'installazione come avviene per le versioni precedenti. L'obiettivo di questo cambiamento è dare all'utente un maggior controllo delle funzionalità dell'applicazione individuando lo specifico permesso nel contesto in cui viene chiesto, cercando di offrire una maggiore consapevolezza e padronanza sul concederlo o meno. I permessi di sistema sono divisi in due categorie principali:

- *Normali,* un permesso appartiene a questa categoria se non è a richio la privacy dell'utente; per questa tipologia di permessi basta inserire nel manifest la dicitura relativa al pemesso e l'applicazione concede il permesso automaticamente all'installazione.
- "*Pericolosi*", sono divisi in 9 gruppi; potendo concedere l'accesso a dati personali dell'utente, questi permessi devono essere esplicitamente richiesti a run-time.

Compatibilità tra versioni

Entrambe le tipologie di permessi vanno inserite nel manifest, se il sistema sulla quale si installa l'applicazione ha una versione di Android precedente la 5.1 o la 5.1 stessa, questi verrano richiesti all'installazione; se il device ha una versione successiva i permessi vanno richiesti e gestiti a run-time come descritto sopra.

Permission life cycle e Pattern

Per non avere un impatto negativo sull'esperienza d'uso, Google stessa dall'I/O del 2015, raccomanda di seguire i seguenti 4 pattern:

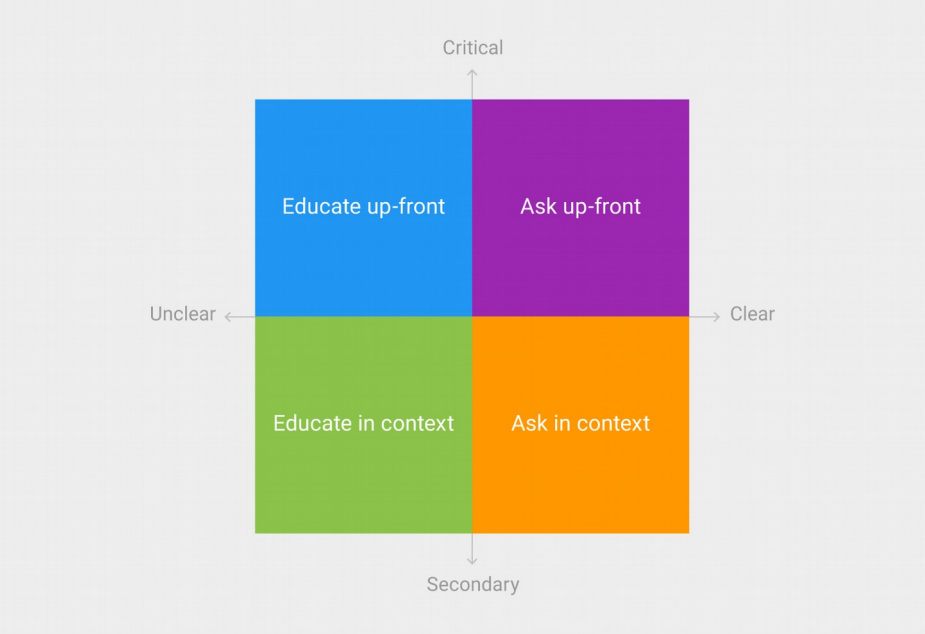

Figura 4.5 'Permission Pattern'

Importanza e chiarezza influenzano il modo in cui il permesso deve essere richiesto; permessi critici e ovvi dovrebbero essere richiesti in modalità up-front (primo avvio) per esempio un'applicazione di messaggistica richiederebbe in questa modalità il permesso *SMS;* permessi meno importanti o meno ovvi andrebbero richiesti nel contesto in cui si devono usare. Quando l'utente potrebbe non comprendere il permesso richiesto, si dovrebbe mostrare un messaggio che spieghi all'utente il permesso richiesto e il motivo della richiesta; questa pratica vale sia per i permessi richiesti nel contesto d'uso che in modalità up-front.

Per quanto riguarda il “ciclo di vita” e la gestione di un permesso si noti il seguente schema:

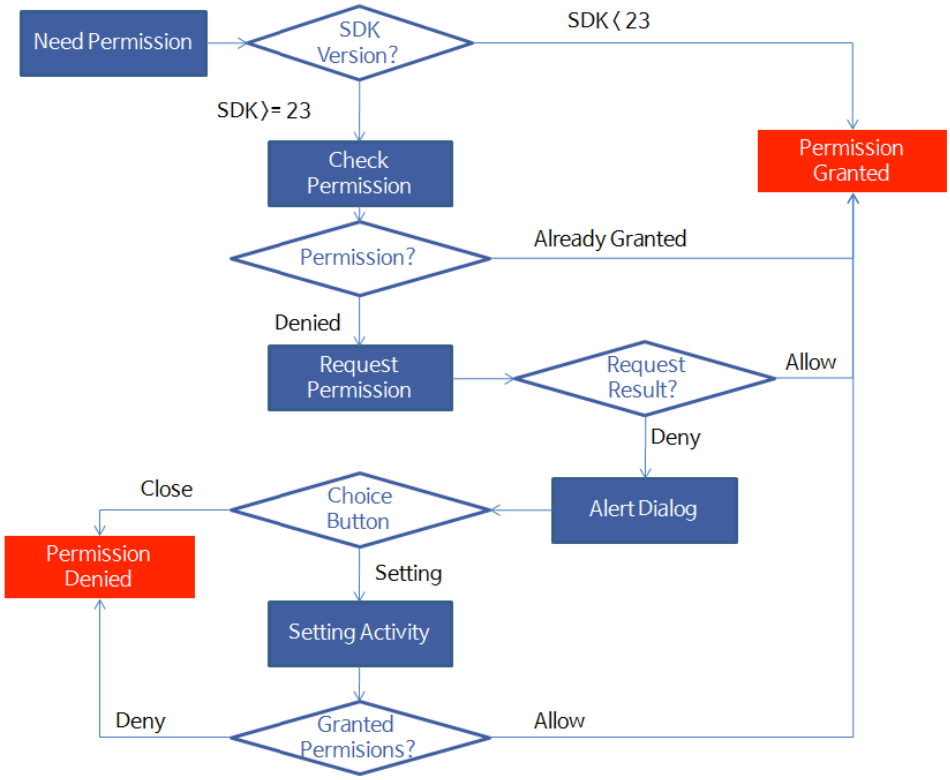

Figura 4.6 'Permission life cycle'

Rispetto a quanto detto, si aggiunge che il permesso può essere negato dall'utente in ogni momento dalle impostazioni, quindi occorre opportunamente ricontrollare se il permesso è stato revocato.

Nell'applicazione si sono usati i seguenti permessi classificati *dangerous: Location, Storage, Camera;* si procede descrivendo ed enfatizzando la modalità e il contesto in cui vengono richiesti.

4.2.6.1 Gestione della posizione

Per poter ordinare gli annunci locali in base alla posizione dell'utente è necessario accedere alla posizione del device; il permesso relativo appartiene al gruppo *Location,* nel manifest occorre dichiarare:

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Come si è descritto in progettazione dopo aver effettuato l'accesso, viene mostrata

la *Home* view. Questa contiene tra gli altri il fragment *Annunci locali* che mostra gli annunci ordinati; in questo contesto si richiede all'utente il permesso relativo, in quanto la posizione dell'utente è necessaria per effettuare la richiesta lato server di ordinamento:

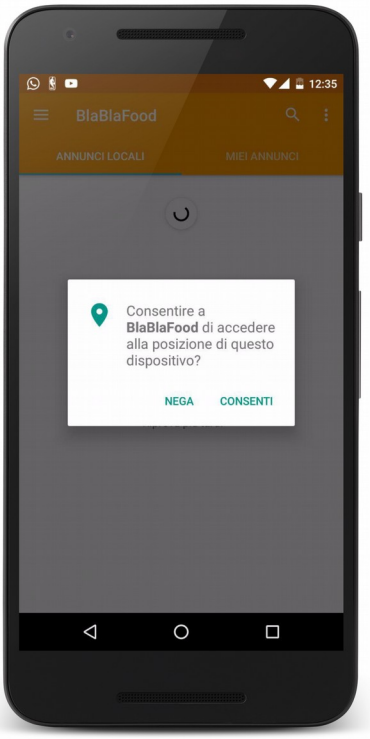

Figura 4.7 'Location Permission request'

Se l'utente non comprende la richiesta e nega il permesso, seguendo le linee guida dei patter sopra citati, si mostra un *razionale* che descrive la motivazione della richiesta del permesso:

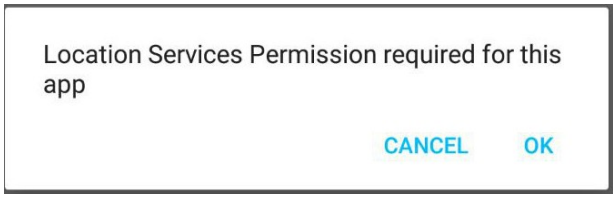

Figura 4.8 'Mostra razionale'

Se l'utente non comprende o non vuole concedere il permesso all'applicazione, può "negare per sempre" (*'Never ask again')* il permesso; in questo caso si mostra un messaggio all'utente in cui si comunica di andare nelle impostazioni del device,

per condere il permesso, se volesse farlo in un futuro. Per garantire una buona esperienza d'uso, con il permesso *Location* negato si è scelto di procedere comunque mostrando all'utente gli annunci ordinati secondo una città scelta di *default;* questo approccio si rivela utile ed è utilizzato da altre applicazioni che richiedono analogamente il permesso *Location.*

4.2.6.2 Gestione immagini e permessi correlati

Un'altro permesso ritenuto 'pericoloso' è quello denominato '*Camera'* per accedere alla fotocamera del dispositivo; questo permesso è necessario per permettere all'utente di scattare foto che si possono aggiungere opzionalmente all'annuncio. Altri permessi correlati alla gestione delle immagini sono: *WRITE_EXTERNAL_STORAGE* e *READ_EXTERNAL_STORAGE* appartenenti al gruppo *Storage.* La richiesta dei permessi in questione viene mostrata nel contesto in cui occorre utilizzarli, ovvero quando l'utente, dall'activity di inserimento annuncio (*AddOrEditActivity),* decide di scattare foto e aggiungerle all'annuncio stesso.

Gestione Immagini dell'applicazione

Con immagini dell'applicazione si intendono quelle dei prodotti e le immagini aggiuntive associate agli annunci inserite dagli utenti; tutte queste vengono gestite attraverso una libreria apposita denominata *Picasso;* si è scelto di utilizzare questa libreria perché offre degli ottimi meccanismi di caricamento e caching; si migliora anche all'esperienza d'uso in quanto la libreria stessa offre la possibilità di mostrare all'utente un download placeholder fino a quando l'immagine non è caricata; se si verificano problemi viene mostrato un error placeholder. Si mostra l'utilizzo descritto sopra attraverso il seguente codice:

```
Picasso.with(context)
   .load(url)
   .placeholder(R.drawable.user_placeholder)
```

```
    .error(R.drawable.user_placeholder_error)
    .into(imageView);
```

Picasso offre una buona gestione delle immagini anche contesti di riciclo come negli Adapter; un ulteriore meccanismo offerto è la possibilità di adattare la dimensioni delle immagini. Queste ultime due feature citate hanno contribuito ulteriormente alla scelta della libreria.

4.2.7 Gestione contatti

Offrire la possibilità di inserire e visualizzare annunci sarebbe stata una funzione fine a stessa senza la possibilità di contattare chi ha pubblicato il prodotto; si è mostrata particolare attenzione nella scelta di quali strumenti e come essi devono essere offerti per mettere in contatto gli utilizzatori dell'applicazione; come descritto nel capitolo di progettazione, l'utente pubblicatore può essere contattato attraverso due canali principali: *Telefono, E-Mail.*

L'activity PublicationDetails gestisce questa parte, presentando all'utente che visualizza l'annuncio, chi lo ha pubblicato e in che modo può contattarlo; nella view si distingue la possibilità di effettuare una *chiamata*, mandare un *sms* o un' *email*; nel caso di invio via *sms* o *email* si fornisce un messaggio precompilato esaustivo, in modo da permettere all'utente di contattare velocemente e in modo chiaro il pubblicatore.

4.2.8 Gestione delle date

Ogni annuncio è valido in un intervallo di date stabilite dall'utente al momento della creazione; il formato di tale date viene gestito mediante l'utilizzo della classe *Utilities* nella quale sono implementati dei metodi statici; si descrivono i principali:

- *String getDateAndTime(long datetime):* permette di ottenere la data e orario in formato String; la data sarà formattata secondo questo pattern *dd*

MMM yyyy, HH:mm.

- *String getClientFormatDate(String dateToConvert):* il server ritorna la data degli annunci nel formato stabilito lato server ovvero yyyy-MM-dd HH:mm:ss; questo metodo permette di ottenere la data nel formato deciso lato client che corrisponde al pattern indicato al punto sopra.
- *boolean isExpired(String date):* permette di controllare se un determinato annuncio è scaduto; il Controller utilizza questo metodo per controllare gli annunci dell'utente, se sono scaduti procede con l'eliminazione; per migliorare l'esperienza d'uso l'annuncio che è in scadenza viene opportunamente evidenziato lato View così da permettere all'utente di prorogarlo o eliminarlo.

4.2.9 Login mediante Social Network

All'avvio dell'applicazione l'activity *LoginActivity* verifica se è già salvato un sessionId nelle *SharedPreferences;* se è presente significa che l'utente ha già effettuato l'accesso e si procede mostrando la *HomeActivity,* altrimenti si mostra la View di login attraverso il *LoginFragment* che principalmente contiene due button per effettuare il login mediante social; si gestisce l'autenticazione mediante i seguenti metodi:

- *LoginGoogle(),* invocato al click da parte dell'utente sul relativo button "Accedi con Google"; attraverso le API di Google si identifica l'utente e si procede con la registrazione; per fare questo occorre procedere come si può notare dal seguente codice commentato:

```
//Configure GoogleSignInOption, we request profile info and email
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestEmail()
                .requestProfile()
                .build();

//Build a GoogleApiClient with access to the Google Sign-In API
//we use options specified by gso (GoogleSignInOption).
mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
   .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
   .build();
```

```
//start sign in flow
Intent signInIntent =
Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
   startActivityForResult(signInIntent, RC_SIGN_IN);
```

Riepilogando occorre: 1) configurare le opzioni di login, 2) creare un oggetto GoogleApiClient con le opzioni configurate, 3) delegare all'activity mediante il signInIntent il login, 4) gestire il risultato dall'activity.

- *LoginFB(),* invocato dal button "Accedi con Facebook", si delega il login all'activity gestita da Facebook e precedentemente configurata dovutamente nel manifest (si veda la documentazione ufficiale per la configurazione); occorre registrare un callback di questo tipo:

```
btnLoginFb.registerCallback(callbackManager, new
FacebookCallback<LoginResult>() {

     @Override
     public void onSuccess(LoginResult loginResult) {}

     @Override
     public void onCancel() {}

     @Override
     public void onError(FacebookException exception) {}
});
```

Nel caso di successo si procede effettuando una richiesta GraphRequest, per ottenere le informazioni dell'utente; nel caso di errore o cancellazione si comunica all'utente il problema.

Dopo aver effettuato il login tramite social e ottenuto le informazioni dell'utente, si procede con la registrazione nel backend mediante un AsyncTask che effettua in background una richiesta HTTP con methodo POST al servizio /login (per ulteriori informazioni al riguardo si rimanda al *paragrafo 4.2.11 Background Processing)* . La risposta positiva dal server implica la fine della registazione e il salvataggio lato client del SessionId ritornato.

4.2.10 Realizzazione front-end

Si descrivono le Activity e Fragment utilizzati per realizzare il front-end,

enfatizzando alcuni aspetti importanti.

4.2.10.1 Colori e Material Design, scelta del colore

"We challenged ourselves to create a visual language for our users that synthesizes the classic principles of good design with the innovation and possibility of technology and science. This is material design."

Questa è la frase che introduce al sito ufficiale "https://material.google.com/" sul material design introdotto da Google nella conferenza I/O del 2014. Tra le varie guidelines e pattern enfatizzati da Google stessa, questo linguaggio di design offre le denominate *material palette,* ovvero delle "tavolozze" di colori comprendenti colori primari e variazioni di essi; la variazione è indicata da un numero specifico: 50, 100, 200, 300, 400, 500, 600, 700, 800, 900 dove 50 indica il più chiaro/lucente e 900 il più scuro; le linee guida incoraggiano a usare il colore 500 come primary color, e il 700 come primary dark utile per esempio per la barra di stato.

Per la maggioranza dei colori, nella *material palette,* sono inoltre presenti quelli che in inglese si definiscono "accent color" indicati con: A100, A200, A400, A700; seguendo le linee guida di Google questi colori vanno usati per enfatizzare determinati componenti grafici come per esempio i Floating Action Button.

La scelta del colore è strettamente correlata e condizionata dal servizio o prodotto che si offre; BlaBlaFood ha come tema principale i prodotti agricoli del km zero; considerando questo si è ritenuto opportuno scegliere il colore arancione. Il rispettivo material color seguendo le linee guida è "Orange 500"; "Orange 700" è utilizzato per la status bar; come accent color si è scelto il "Light Blue Accent 400".

4.2.10.2 TutorialActivity

Mostra il tutorial dell'applicazione all'utente al primo avvio; successivamente si può accedere a questa activity dal NavigationDrawer. La struttura della vista è

semplice e presenta nella parte superiore un titolo , al centro un'immagine, e sottostante all'immagine una descrizione della funzionalità o della sezione dell'applicazione che si sta spiegando. Si è utilizzata la libreria *AppIntro* che utilizza un ViewPager, ovvero un layout che permette di navigare effettuando degli swipe a destra o sinistra attraverso le “pagine” o “sezioni”, in questo caso Fragment (il componente e la sua gestione viene descritta in seguito); attraverso un indicatore di pagina si mostra all'utente di quante pagine è composto il tutorial e quale pagina sta visualizzando.

4.2.10.3 HomeActivity

Come si è argomentato in progettazione, questa è l'activity principale; viene 'avviata' dalla LoginActivity se l'utente ha già effettuato l'accesso al sistema. Essa è composta principalmente dal NavigationDrawer e da un ViewPager contenente i fragment 'Annunci Locali' (WorldPublicationFragment) e 'I Miei Annunci' (MyBoardFragment) gestiti attraverso un FragmentPagerAdapter. La comunicazione tra activity e fragment avviene mediante le interfaccie esposte dagli stessi, della quale si espongono i metodi principali:

- onPublicationClicked(Publication publication): al click su un annuncio la home activity delega alla PublicationDetailActivity la visualizzazione del dettaglio dell'annuncio.
- onLongPublicationClick(Publication publication): al click prolungato su un annuncio, se esso appartiene all'utente si mostra una dialog che chiede se si vuole proseguire con la cancellazione.
- onUpdateOver(boolean result): al termine dell'aggiornamento degli annunci dell'utente, si notifica alla HomeActivity il risultato; essa provvederà a visualizzare all'utente mediante Snackbar un messaggio indicante l'esito dell'aggiornamento.

Le prime operazioni che la HomeActivity esegue sono le seguenti, per la spiegazione dei metodi del controller citati si faccia riferimento al paragrafo

relativo (numero 4.2.4) :

- *Richiesta al Controller degli annunci personali:* mediante il metodo getPublicationUser
- *Richiesta al Controller degli annunci degli altri utenti:* mediante il metodo getPublicationWorld.

Al termine delle operazioni sopra la View si aggiorna mostrando all'utente i dati, il risultato della realizzazione del wireframe riportato in progettazione è il seguente:

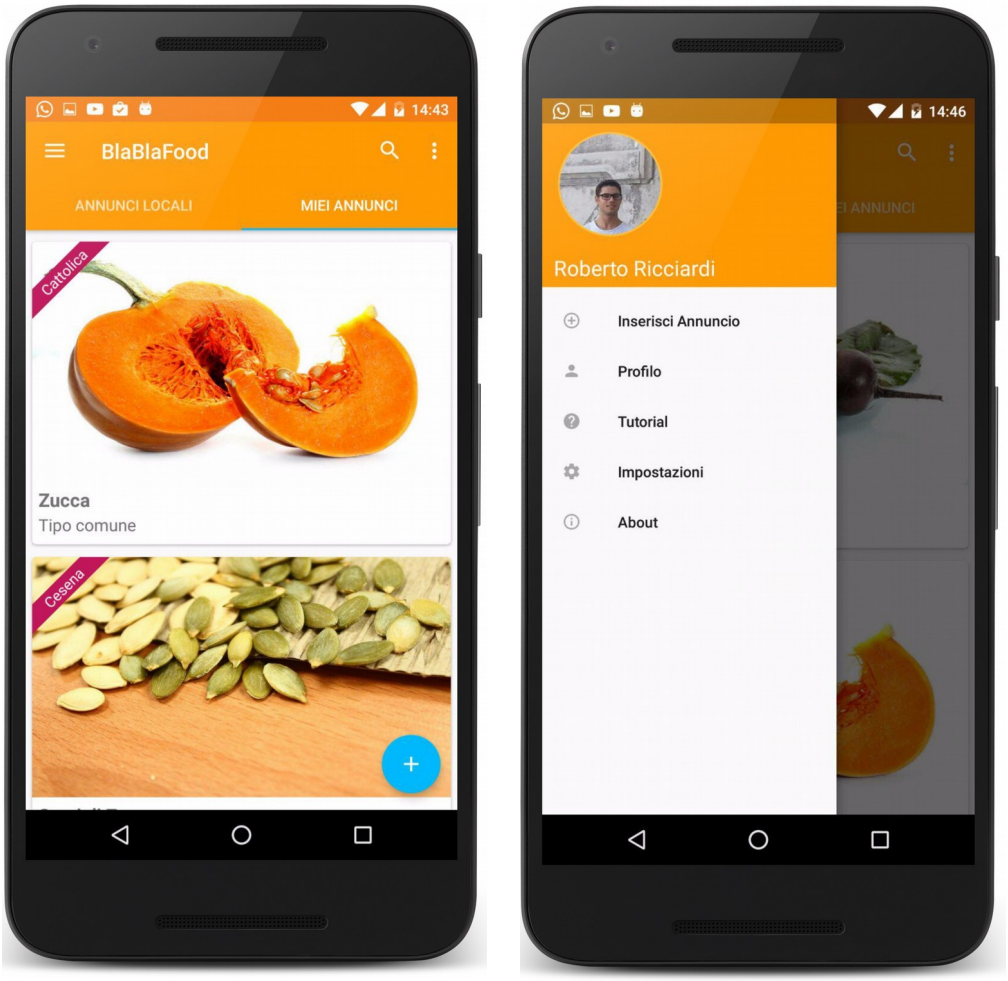

Figura 4.9 "Home"

Come si può notare nell'immagine, attraverso il tab del ViewPager o mediante swipe si può navigare dal Fragment 'Annunci Locali' a 'Miei Annunci' e viceversa. Dal Floating Action Button si inserisce un annuncio. Scorrendo laterlamente dal

lato sinistro verso destra si apre il Navigation Drawer. Il risultato è stato ottenuto usando i seguenti layout: DrawerLayout, attraverso questo layout si gestisce il NavigationDrawer; CoordinatorLayout contenente l' AppBarLayout, e il ViewPager.

Per ottenere una buona esperienza d'uso e un buon risultato visivo, senza tralasciare aspetti come un uso parsimonioso delle risorse, si è deciso di utilizzare il componente grafico *RecyclerView* e il widget *CardView.*

RecyclerView è una versione più avanzata e flessibile del componente ListView. E' progettato per contenere grandi quantità di dati, fa buon uso della memoria mantendo un numero limitato di View attive. Nella HomeActivity viene utilizzato insieme alle CardView, un widget per organizzare il layout degli item all'interno della RecyclerView. Si mostrano gli elementi principali attraverso la seguente immagine:

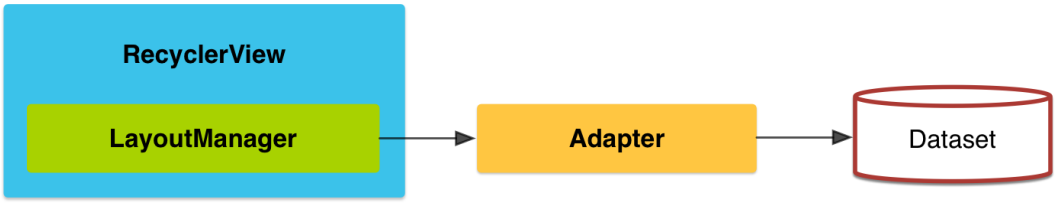

Figura 4.10 "RecyclerView"

Distinguiamo:

Layout manager: il compito principale del layout manager è posizionare gli elementi nella RecyclerView e determinare quando riusare le view che non sono visibili all'utente sostituendone il contenuto; per fare questo il manager interagisce con l'adapter; RecyclerView fornisce tre built-in layout manager: LinearLayoutManager, GridLayoutManager, StaggeredGridLayoutManager; nello specifico per l'applicazione si è deciso di utilizzare il LinearLayoutManager.

***Adapter*:** permette di popolare la RecyclerView con gli elementi presi da un Dataset; l'adapter realizzato è *PublicationAdapter* presente nel package pubblication che estende la classe RecyclerView.Adapter. Nell'implementazione si si aderisce al "*View Holder" design Pattern,* grazie al quale si evita l'uso ripetuto

del findViewById() necessario per recuperare gli elementi grafici; per fare questo occorre implementare una PublicationViewHolder che estende RecyclerView.ViewHolder fornendo un costruttore nel quale per ogni dato complesso (in questo caso immagine, titolo e descrizione del prodotto) si referenzia le view atte a contenere le informazioni da visualizzare.

I metodi principali, invocati dal layout manager e realizzati sono i seguenti:

- *onCreateViewHolder(...):* crea nuove View utilizzando il layout specificato e la PublicationViewHolder.
- *onBindViewHolder(ViewHolder holder, int position):* sostituisce il contenuto di una CardView con il dato presente nel dataset in posizione 'position'.
- *getItemCount():* ritorna la dimensione del dataset.

***Dataset**:*è la sorgente dei dati, nello specifico una List<Pubblication> contenente gli annunci.

Animazioni e aggiornamento View

RecyclerView offre la possibilità di utilizzare delle animazioni, per esempio nell'aggiunta o rimozione di un elemento nella lista. Per migliorare l'esperienza d'uso si è deciso di utilizzarle al meglio; si distinguono i seguenti scenari e come sono state ottenute le relative animazioni:

- *aggiunta di un annuncio*: aggiunta nel dataset del dato, notifica del cambiamento attraverso il metodo notifyItemInserted(position).
- *rimozione di un annuncio*: rimozione dal dataset del dato, notifica del cambiamento attraverso il metodo notifyItemRemoved(position).
- *modifica di un annuncio*: modifica del dato nel dataset, notifica del cambiamento attraverso il metodo notifyItemChanged(position).
- *animazione durante lo scrolling*: mentre l'utente scorre la lista per la prima volta si effettua un'animazione degli elementi, facendoli comparire da sinistra verso destra; per ottenere questo effetto si utilizza il metodo startAnimation(Animation animation) sulla CardView con l'animazione

android.R.anim.slide_in_left.

Gestione

Per poter utilizzare il componente dopo aver implementato quanto è stato scritto occorre: 1) recuperare il widget mediante findViewById, 2) instanziare un layout manager, nel caso specifico un linear layout, 3) mediante il metodo setLayoutManager si associa alla Recycler il manager del layout, 4) si instanzia il PublicationAdapter e mediante il metodo setAdapter si associa alla Recycler l'Adapter.

Si è prestata attenzione anche a dettagli come la gestione di una lista vuota, questo può accadere le prime volte che l'utente usa l'applicazione o quando non ha annunci attivi; per migliorare l'esperienza d'uso si è evitato di lasciare completamente vuota la schermata, mostrando una View che indichi all'utente che non ci sono annunci nella zona; mentre se è nel fragment 'Miei Annunci' si mostra all'utente che non ci sono annunci attivi e si danno brevi istruzioni su come aggiungere un nuovo annuncio. A livello implementativo per ottenere questo risultato occorre estendere la classe RecyclerView, nel caso specifico si denomina *EmptyRecyclerView;* occorre implementare un AdapterDataObserver, e fare override del metodo setAdapter, all'interno del quale si registra l'observer mediante il metodo registerAdapterDataObserver. Si noti il seguente codice:

```
final private AdapterDataObserver observer = new
AdapterDataObserver() {
        @Override
        public void onChanged() {
            checkIfEmpty();
        }

        @Override
        public void onItemRangeInserted(int positionStart, int
itemCount) {
            checkIfEmpty();
        }

        @Override
        public void onItemRangeRemoved(int positionStart, int
itemCount) {
            checkIfEmpty();
        }
    };
```

Come si può notare occorre al cambiamento della sorgente dati controllare se la lista è vuota; attraverso il metodo checkIfEmpty, invocando il metodo getItemCount sull'adapter, si verifica se sono presenti degli item; se la lista è vuota si mostra la info View relativa all'utente come citato precedentemente.

Aggiornamento

Mediante l'utilizzo dello SwipeRefreshLayout si permette all'utente mediante l'apposita gesture verticale di effettuare l'aggiormento dei dati. L'aggiornamento è gestito invocando sul Controller il metodo relativo e al termine della procedura si aggiorna la vista.

Utilizzo

RecyclerView e CardView fanno parte della libreria *v7 Support,* per poterli utilizzare occorre inserire nel progetto queste Gradle dependencies al modulo app:

```
compile 'com.android.support:cardview-v7:21.0.+'
compile 'com.android.support:recyclerview-v7:21.0.+'
```

4.2.10.4 DetailActivity

Al click su un annuncio si mostra il dettaglio attraverso la PublicationDetailActivity; per fare questo il fragment esegue il seguente codice:

```
Intent intent = new Intent(getActivity(),
PublicationDetailsActivity.class);

intent.putExtra(PublicationDetailsActivity.PROFILE, false);
intent.putExtra(PUBLICATION, publication);
intent.putExtra(POSITION, currentPosition);
getActivity().startActivityForResult(intent,
PUBLICATION_DETAILS_ACTIVITY);
```

Come si può notare mediante il metodo putExtra(…) si passa alla PublicationDetailActivity l'annuncio da mostrare, la posizione nell'Adapter e un flag riguardante il profilo utente; annuncio e posizione sono valori ritornati dall'Adapter stesso al fragment mediante callback al click sull'item; il flag riguardo al profilo è utile per indicare alla PublicationDetailActivity, se al click sul nome o immagine nella sezione che mostra chi ha pubblicato l'annuncio, si

debba mostrare il relativo profilo; questo per evitare fastidiosi cicli di navigazione per l'utente del tipo: fragment→ dettaglio→ profilo→ fragment→ dettaglio→ profilo.

Si riporta la realizzazione del wireframe:

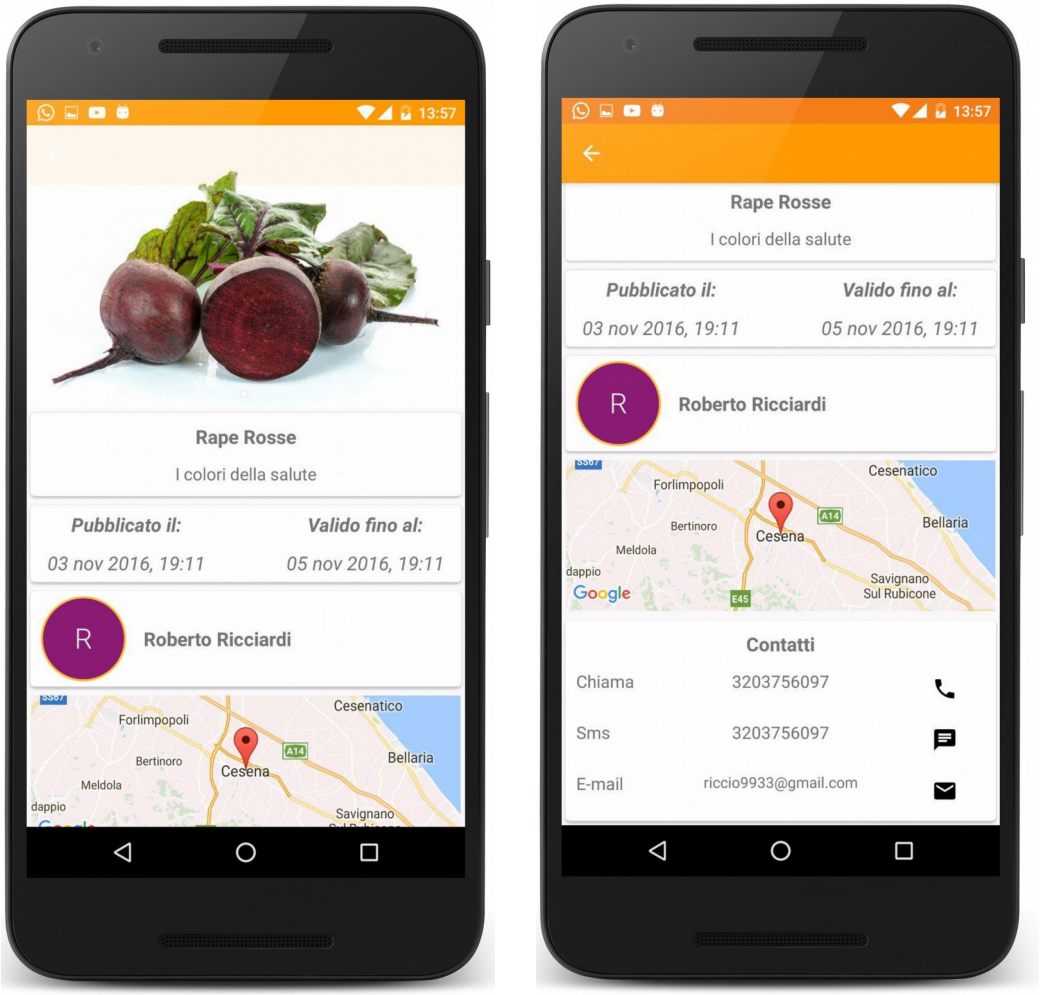

Figura 4.11 "Dettaglio annuncio"

Se l'annuncio è di proprietà dell'utente, si mostra un Floating Action Button per permettere eventualmente di modificarlo attraverso la AddOrEditActivity che verrà descritta in seguito; inoltre si aggiunge un'icona a forma di cestino nella action bar per permettere di eliminare l'annuncio.

La composizione e organizzazione della view e stata già argomentata in progettazione, qui si descrive come si sono realizzati determinati componenti.

Immagini

Nella parte superiore si mostrano le immagini associate all'annuncio, per fare questo si è utilizzato un ViewPager; per gestire il componente si è implementato un ImagePagerAdapter estendendo la classe astratta PagerAdapter, i metodi principali sono i seguenti:

- getCount(): ritorna il numero di View disponibili.
- instantiateItem(…): crea la pagina per la posizione corrente, attraverso la libreria Picasso si effettua il caricamento dell'immagine.
- destroyItem(…): rimuove una pagina alla posizione specificata, l'adapter è il responsabile della rimozione.

Attraverso il metodo setOffscreenPageLimit(int limit) si specifica al ViewPager quante pagine devono essere mantenute per ogni lato (destro e sinitro) in stato inattivo; il valore di default è 1, come è consigliato dalla documentazione se si ha un numero contenuto di pagine (3-4) si possono mantenere tutte quante per migliorare le performance e evitare a seguito di swipe laterali veloci dei fastidiosi rallentamenti; nello specifico considerando che un annuncio può ragionevolmente avere associate in media 2/3 immagini si è deciso di mantenerle in memoria.

Informazioni Utente

Le altre sezioni sono realizzate utilizzando il widget CardView. La parte che mostra l'immagine e il nome dell'utente che ha pubblicato l'annuncio permette di mostrare il profilo al click su uno di questi componenti; per fare questo si associa il listener mediante il metodo setOnClickListener sui due componenti, nel quale si effettua lo startActivity della ProfileActivity.

Mappa

Si mostra la posizione dell'utente che ha pubblicato l'annuncio mediante le Google API relative; si utilizza una MapView, ovvero una vista che visualizza una mappa; i dati sono ottenuti mediante il servizio Google Maps. Per fare questo occorre:

- Configurare il proprio progetto attraverso la delevoper console, e ottenere una API key necessaria per usufruire del servizio.

- Aggiungere nel manifest attraverso il tag <meta-data> la versione del Google Play Services usata, e l'API key ottenuto.
- Aggiungere il componente grafico nel file layout xml nel quale si specifica il valore della chiave ottenuta:

```
<com.google.android.gms.maps.MapView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:apiKey="@string/api_key_maps"
    android:clickable="true"
    android:id="@+id/mapView" />
```

- Implementare onMapReady callback nel quale si inizializza la mappa e si aggiunge il marker nel seguente modo:

```
@Override
public void onMapReady(GoogleMap googleMap) {
  map = googleMap;
  map.getUiSettings().setMyLocationButtonEnabled(false);

  MapsInitializer.initialize(PublicationDetailsActivity.this);

  LatLng latLng = new LatLng(publication.getLatitudine(),
  publication.getLongitudine());

  map.addMarker(new
  MarkerOptions().position(latLng).title(TITLE));

  // Updates the location and zoom of the MapView
  CameraUpdate cameraUpdate =
  CameraUpdateFactory.newLatLngZoom(latLng, 10);
        map.animateCamera(cameraUpdate);
}
```

Contatti e scelta dei button

La sezione che mostra i contatti è stata realizzata utilizzando un RelativeLayout; i componenti utilizzati sono delle TextView per visualizzare il contatto, mentre l'azione effettuabile su di essi si mostra mediante ImageButton di tipo *flat;* per ottenere il risultato estetico occorre applicare il *Borderless* style come mostrato:

```
<ImageButton
  android:id="@+id/Sms"
  android:layout_alignParentRight="true"
  android:layout_alignParentEnd="true"
  android:src="@drawable/ic_chat_black_24dp"
  style="@style/Widget.AppCompat.Button.Borderless"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

La scelta di utilizzare un button di tipo *flat* è stata presa seguendo le linee guida di Google sul material design; l'elemento principale della scelta è ragionare sull'importanza del button nel contesto in cui è inserito; per evitare di distrarre l'utente nella visualizzazione, occorre scegliere con attenzione lo stile del bottone. Considerando il contesto della PublicationDetailActivity si è deciso quindi di utilizzare un button di tipo *flat* piuttosto che un *raised* button. Si riportano per aiutare il lettore e aumentare la leggibilità la "gerarchia" dei button dalle linee guida:

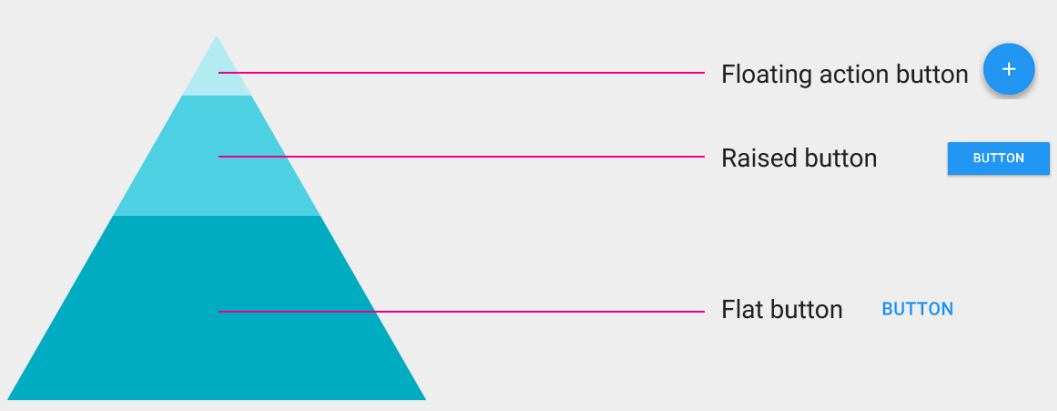

Figura 4.12 'Gerachia Button'

4.2.10.5 AddOrEditActivity

La AddOrEditActivity ha richiesto molto impegno a livello implementativo; obiettivo principale è stato offrire un inserimento di un annuncio (o modifica) veloce e intuitivo; seguendo le linee definite in progettazione la realizzazione si divide in questi punti principali:

- Aggiunta Immagini
- Selezione del Prodotto
- Selezione dell'intervallo di validità

- Selezione della posizione
- Condivisione su Social

Si mostra la realizzazione del wireframe per poi procedere con la descrizione di ogni punto.

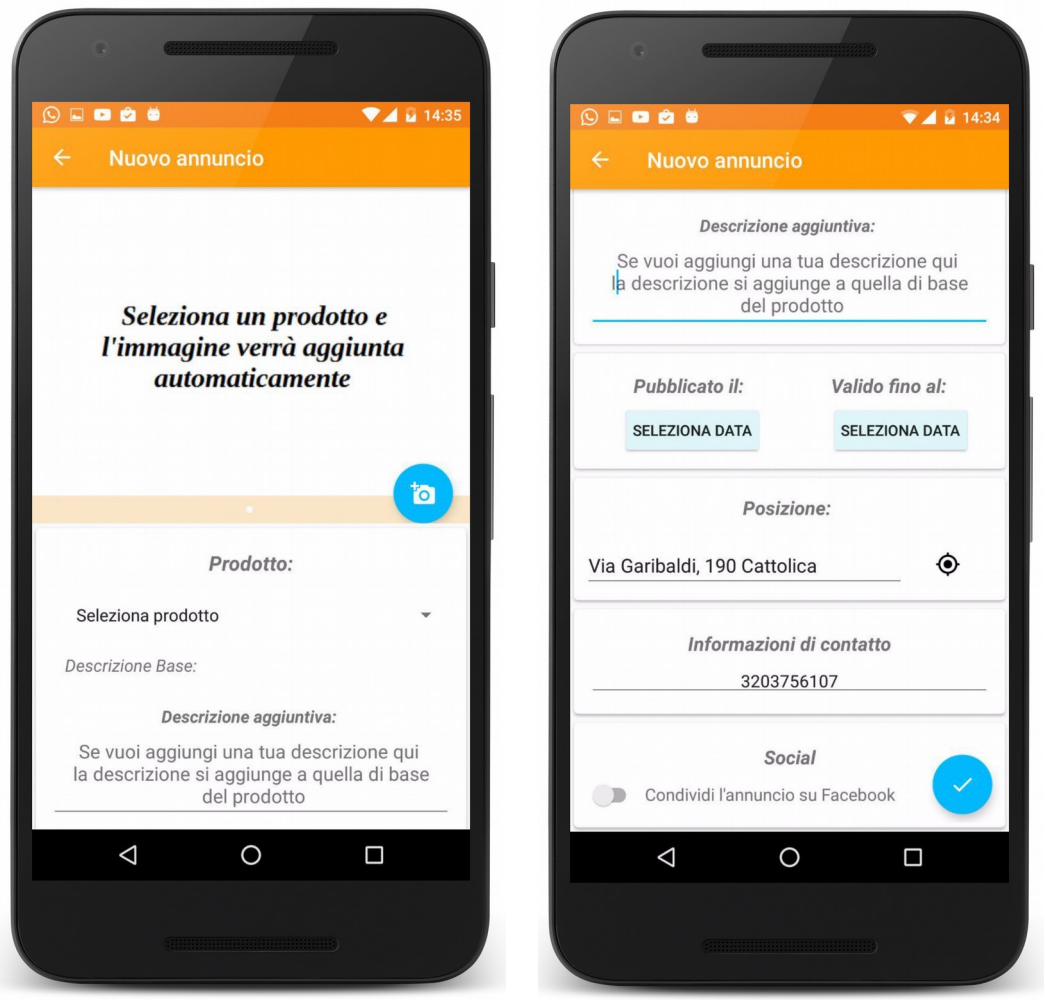

Figura 4.13 'Aggiunta annuncio'

Aggiunta Immagini

Selezionando un prodotto, viene automaticamente inserita l'immagine di base associata visualizzandola nel ViewPager posto nella parte superiore dell'Activity; per permettere all'utente di aggiungere ulteriori immagini; al click sul relativo

Floating Action Button, si delega all'Activity che gestisce la galleria immagini del telefono; inoltre mediante essa si può procedere scattando nuove immagini. Per fare questo, a seguito di attente ricerche, si è scelto di utilizzare la libreria "Gallery Module"; essa permette in un'unica schermata di visualizzare la galleria del telefono e scattare nuove immagini; si è deciso di utilizzarla vista la praticità, inoltre l'approccio che adotta è quello seguito da applicazioni come WhatsApp e Telegram.

Per poter utilizzare la fotocamera o la galleria immagini occorre preventivamente controllare se i permessi Camera e Storage sono stati concessi, per vedere come questo è stato fatto si rimanda al paragrafo in cui si è già spiegato questo aspetto (*paragrafo 4.2.6.2);* quando i permessi vengono concessi si esegue il seguente codice:

```
startActivityForResult(new
GalleryHelper().setMultiselection(true)
                    .setMaxSelectedItems(3)
                    .setShowVideos(false)
                    .getCallingIntent(this), 500);
```

Nel far partire l'activity, attraverso il metodo fornito dalla libreria, setMultiSelection si comunica se si vuole dare la possibilità di selezionare più immagini e mediante setMaxSelectedItems si comunica quante devono essere queste immagini.

Nell'onActivityResult si gestisce il risultato, l'activity "GalleryHelper" ritorna un insieme di Uri che permettono di identificare le immagini aggiunte/selezionate.

Al termine della procedura si aggiorna il ViewPager mostrando le immagini che sono attualmente associate all'annuncio che si sta creando/modificando.

Selezione del prodotto

Per permettere una facile selezione del prodotto da associare all'annuncio, si è scelto di utilizzare uno Spinner, componente che permette di selezionare un elemento da una lista visualizzabile a video. Considerando che i prodotti possono essere tanti, si è deciso di aggiungere la funzionalità di ricerca dentro lo spinner, così da velocizzare la selezione; per fare questo prima di procedere con

l'implementazione si è cercato in rete, per controllare se fosse già presente una libreria adatta a questo. La ricerca a condotto a questo componente visualizzato in figura che si è rivelato ottimo:

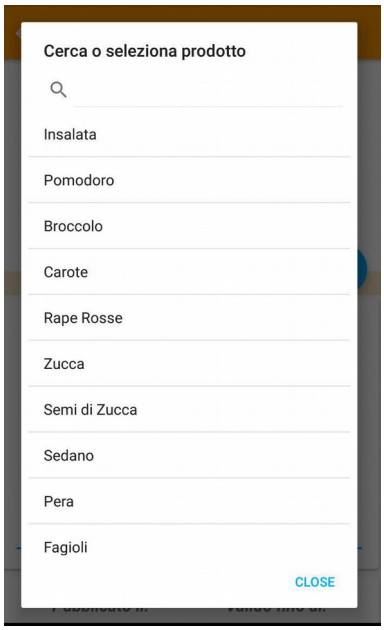

Figura 4.14 'Seleziona prodotto'

Selezione dell'intervallo di validità

Per permettere di inserire data iniziale e finale velocemente, attraverso i raised button si mostra un DatePicker, componente predisposto per selezionare una data. Nello specifico si è utilizzata la libreria 'RetainedDateTimePickers' che permette di gestire facilmente la visualizzazione del selettore anche a fronte di cambi di configurazione. Quando l'utente seleziona la data iniziale, nel secondo selettore si mostra la possibilità di selezionare una data successiva a quella; disattivando la selezione di date inferiori si evitano errori nell'inserimento dell'intervallo di validità da parte dell'utente.

Selezione della posizione

L'ultima posizione utilizzata viene visualizzata nell'EditText relativo; si ha anche un flat button con l'icona del gps per permettere di inserire la posizione corrente.

L'ottenimento della posizione e la gestione dei permessi relativi avviene in modo analogo a come si è fatto per la HomeActivity, spiegata in precedenza.

Condivisione su Social

Nell'inserimento si offre all'utente la possibilità di condividere l'annuncio creato sul Social; attraverso la checkbox relativa l'utente esprime la sua intenzione a condividerlo; se l'utente ha effettuato l'accesso al sistema mediante Facebook si procede con la condivisione mediante la ShareDialog presente nell'SDK del social; viceversa se l'utente ha effettuato l'accesso con Google si procede alla condivisione su Google+ utilizzando le relative API.

4.2.10.5.1 Condivisione su Google+

Attraverso il seguente codice si mostra l'interfaccia di condivisione:

```
Intent shareIntent = new PlusShare.Builder(this)
        .setType("text/plain")
        .setText("Ho appena pubblicato questo annuncio su
BlaBlaFood!")

.setContentUrl(Uri.parse(publication.getProduct().getImmagine()))
        .getIntent();
startActivityForResult(shareIntent, SHARE_GOOGLE_ACTIVITY);
```

Come si può notare dal codice attraverso il metodo setType(…) si specifica il formato dei dati che si sta condividendo, con setText(…) si precompila il messaggio di condivisione, con setContentUrl(…) si associa l'immagine del prodotto da condividere. Viene avviata l'activity passanto l'Intent, dall'interfaccia di condivisione l'utente può selezionare a quali cerchie o singoli utenti vuole condividere il post. Il risultato è il seguente:

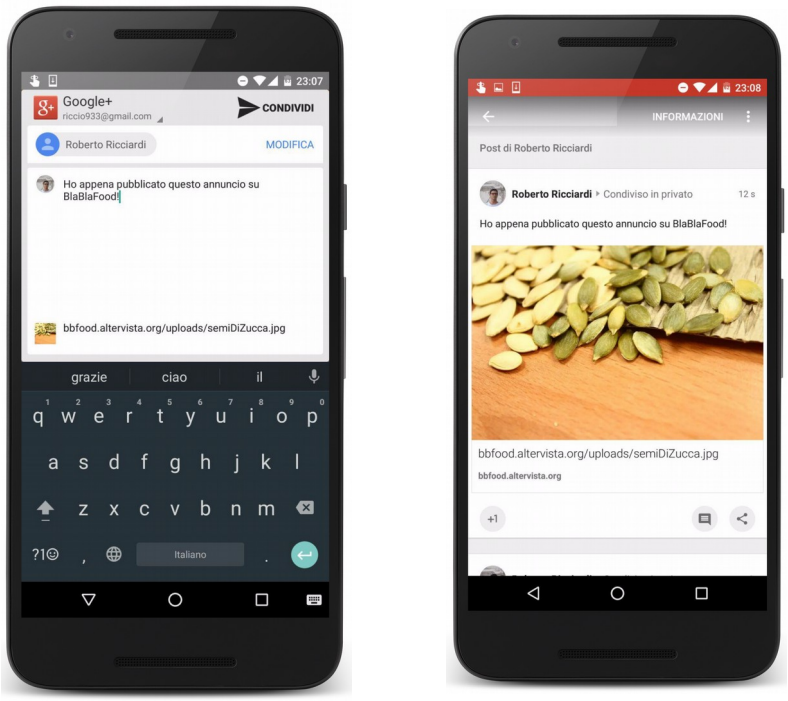

Figura 4.15 'Interfaccia share Google+ e risultato ottenuto'

4.2.10.5.2 Condivisione su Facebook

Analogamente utilizzando l'SDK di Facebook attraverso una *ShareDialog* si mostra all'utente l'interfaccia di condivisione, si noti il codice seguente:

```
if (ShareDialog.canShow(ShareLinkContent.class)) {
    ShareLinkContent linkContent = new ShareLinkContent.Builder()
            .setContentTitle("BlaBlaFood")
            .setContentDescription(
                    "Ho appena aggiunto un annuncio su
BlaBlaFood")

.setContentUrl(Uri.parse(publication.getProduct().getImmagine()))
            .build();
    shareDialog.show(linkContent);
}
```

Attraverso il metodo setContentDescription si setta il titolo del link condiviso, con setContentDescription si aggiunge la descrizione, e mediante setContentUrl si assoicia l'Url dell'immagine del prodotto che si è appena pubblicato. Con

shareDialog.show si mostra all'utente la dialog per effettuare la condivisione. Il risultato è il seguente.

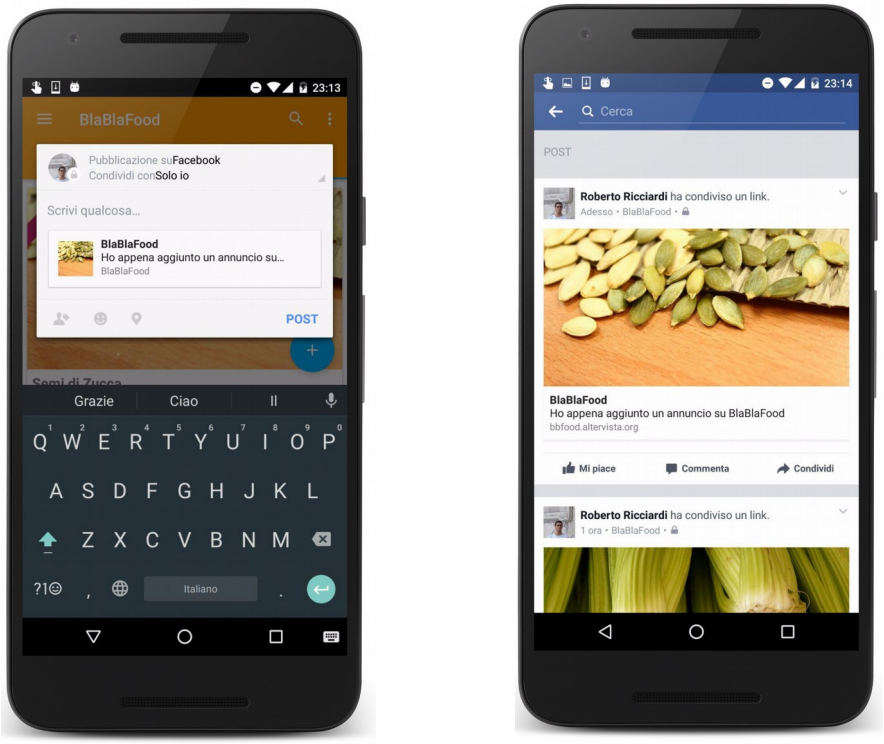

Figura 4.16 'Dialog Share Facebook e risultato ottenuto'

4.2.10.6 ProfileActivity

Grazie all'utilizzo dei fragment la ProfileActivity è ottenuta riusando il fragment già implementato 'I miei Annunci'. Si riporta il risultato della realizzazione:

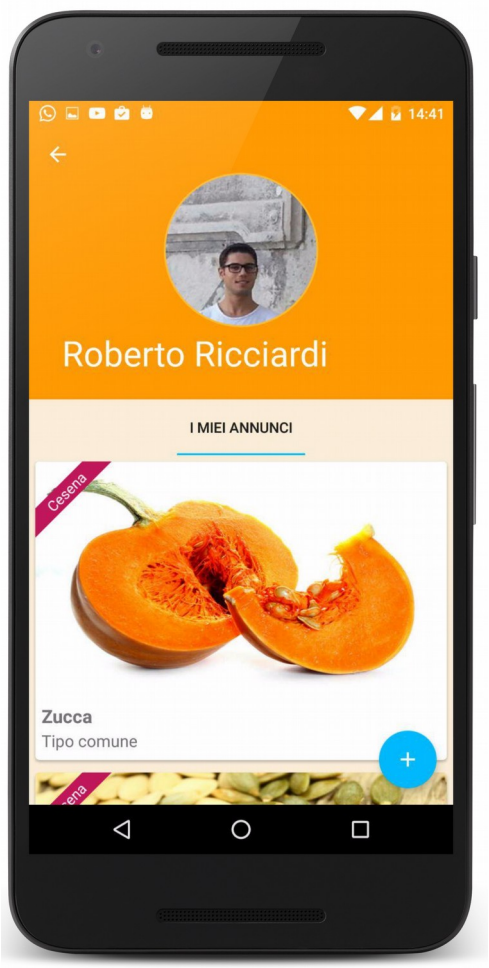

Figura 4.17 'Profilo utente'

Come si può notare si mostra l'immagine e il nome del profilo nella parte superiore attraverso il *CollapsingToolbarLayout;* questo è un wrapper della toolbar e fornisce un meccanismo di adattamento della stessa durante lo scrolling da parte dell'utente. Riguardo al fragment riusato si rimanda alla HomeActivity nel quale si è già descritto.

4.2.10.7 Gestione dei cambi di configurazione

Le activity (o fragment) vengono "distrutte" e rimosse dalla memoria e successivamente ricreate nuovamente da zero a fronte di cambi di configurazione;

un esempio è quando si verifica una rotazione del dispositivo. In queste situazioni una best practice a cui si è aderito è salvare e ripristinare lo stato dell'activity.
Il sistema quando effettua la chiusura dell'activity, chiama il metodo *onSaveInstanceState(Bundle savedInstanceState)*; in questo metodo si possono salvare le informazioni da mantenere con una collezione di coppie chiave → valore.
Successivamente il sistema chiama *onRestoreInstanceState(Bundle savedInstanceState)*, in questo metodo si possono recuperare i valori salvati precedentemente nell' onSaveInstanceState attraverso il Bundle.
Per i fragment la gestione è analoga, l'unica differenza è nel restore in quanto lo stato viene ripristinato nell'onCreateView.

4.2.11 Background processing

Dopo aver descritto le activity e i fragment utilizzati è doveroso completare l'argomentazione descrivendo cosa si è deciso utilizzare, e perché, per quanto riguarda le operazioni da svolgere in backgroud.
È necessario evitare di svolgere operazioni pesanti nell' UI(User interface) Thread per evitare che le View si blocchino o non rispondino; in questo modo si evitano le ANR (Application Not Responding) dialog.

4.2.11.1 Asynctask

Un *Asynctask* è un componente progettato per facilitare la gestione dell'UI thread, in quanto permette di eseguire operazioni in paralello e inviare risultati all' UI thread senza dover manipolare Thread o Handler.
Gli Asynctask andrebbero usati per gestire task di breve durata, per operazioni più lunghe occorre ricorrere ai *Service,* componente di Android progettato per questi compiti.
Si è deciso di utilizzare gli Asynctask per il *login* dell'utente, e per il *caricamento dei dati* dalla cache locale. Si riporta a titolo esemplificativo il seguente codice:

```
@Override
public void onRefresh() {
   new AsyncTask<Void, Void, Void>() {
      @Override protected void onPreExecute() {
           mSrl.setRefreshing(true);
      }

      @Override protected Void doInBackground(Void... params) {

       Controller.getInstance().getPublicationsUser(user.getId(),
       receiver, false);

       return null;
      }

      @Override
      protected void onPostExecute(Void aVoid) {
         mSrl.setRefreshing(false);
      }
   }.execute();
}
```

Come si nota dal codice, i metodi principali sono onPreExecute(), doInBackground(...), onPostExecute(); essi vengono eseguiti nell'ordine in cui sono scritti; onPreExecute e onPostExecute sono svolti nell' UI thread e permettono di interagire con le view; mentre doInBackground è eseguito in parallelo. Nell'esempio sopra si fa partire la progress view che mostra che l'aggiornamento è iniziato, nel doInBackground si esegue la richiesta al controller e nell'onPostExecute si ferma la progress view.

4.2.11.2 Service e IntentService

Un Service è un componente concepito per eseguire lunghe operazioni senza coinvolgere direttamente l'utente. Ogni classe Service utilizzata nell'applicazione deve essere dichiarata nel manifest attraverso il relativo tag <service>. La comunicazione tra activity e servizi avviene via Intent, e il servizio può essere avviato mediante il metodo Context.startService() oppure Context.bindService(); nel secondo si instaura una connessione/collegamento tra activity e servizio stesso. Service è una classe astratta, per poter utilizzare un servizio occorre estendere tale classe; IntentService è un estensione base di Service.

Si evidenziano le differenze principali tra Service e IntentService:

- Service viene eseguito in background ma sempre nel Main Thread dell'applicazione, mentre IntentService lavora in un worker thread separato.
- IntentService esegue un messaggio/intent ricevuto per volta, mettendo gli altri in coda; in questo modo aderisce al *work queue processor pattern.*
- Se si decidesse di implementare un Service, occorre chiamare opportunamente il metodo stopSelf() o stopService() per fermare il servizio al termine del lavoro; nel caso dell'IntentService invece non è necessario fare questo in quanto il servizio viene fermato appena tutte le richieste sono state gestite, per questo non si deve chiamare stopSelf().

4.2.11.2.1 CRUD degli annunci

Alla luce delle informazioni sopra esposte e delle ricerche fatte, analizzando il dominio applicativo, si è deciso di utilizzare IntentService e ResultReceiver per gestire le operazioni di rete come aggiornamento della cache o aggiunta, rimozione, e aggiornamento degli annunci.
Per utilizzare gli IntentService occorre estendere tale classe e implementare il metodo onHandleIntent(Intent), nel quale si gestisce la richiesta inviata mediante l'Intent. La classe *AbstractService* estende IntentService e fornisce, aderendo al pattern template method, uno 'scheletro' per gestire i messaggi ricevuti, si noti il seguente codice:

```
@Override
protected void onHandleIntent(Intent intent) {
    receiver = intent.getParcelableExtra(RECEIVER);
    op = (Operation) intent.getSerializableExtra(OPERATION);
    this.intent = intent;
    switch (op){
        case CHECK_UPDATE_BOARD:
            doUpdateBoard();
            break;
        case CHECK_UPDATE_WORLD:
```

```
            doUpdateWorld();
            break;
        case DELETE:
            doDelete();
    }
}
```

Si utilizza un *enum Operation* per identificare il tipo di operazione richiesta così da invocare il metodo relativo per gestirla. I 3 metodi sopra sono implementanti dalla classe *Service* (da non confondere con la classe astratta di Android descritta precedentemente) che estende AbstractService. Si descrivono i 3 metodi:

- doUpdateBoard(): verifica e aggiorna gli annunci di uno specifico utente effettuando una richiesta HTTP con metodo GET al servizio lato server /publicationUser.
- doUpdateWorld(): verifica e aggiorna gli annunci degli altri utenti effettuando una richiesta HTTP con metodo GET al servizio /publication.
- doDelete(): esegue l'eliminazione di un annuncio passato nell'Intent, attraverso una richiesta HTTP con metodo DELETE al servizio /publicationUser.

Le altre operazioni sono gestite analogamente consumando i servizi web appositi.
Le richieste in rete vengono eseguite attraverso la classe HttpURLConnection, la documentazione della classe descrive bene il suo utilizzo, si riportano i punti chiave in ordine, per enfatizzare come si è adoperata:

1. *Ottenimento di una connessione* mediante il metodo openConnection() su un URL.
2. *Preparazione della richiesta*, utilizzando i metodi forniti dalla classe.
3. *Upload* (se richiesto) di un messaggio nel body della richiesta
4. *Lettura della risposta* mediante lo stream di dati ritornato dal metodo getInputStream().
5. *Disconnessione* e rilascio delle risorse.

Al termine dell'operazione di rete il servizio procede con l'aggiornamento della cache locale, e notifica al ResultReceiver passato nell'Intent l'avvenuta operazione; esso valuterà in base al contesto come agire. (Per una spiegazione approfondita si veda il *paragrafo 4.2.2 Enfasi sull'architettura).*

Per l'aggiunta di un annuncio, essendo l'operazione più onerosa in termini tempo, si è deciso di utilizzare un *foreground IntentService;* quando è attivo si mostra una notifica nella status bar che non può essere tolta finché il servizio non termina.

In questo modo si riesce a terminare l'operazione anche se l'utente esce dall'applicazione durante l'upload dell'annuncio. Si noti il seguente codice per mostrare/avviare la notifica:

```
Intent notificationIntent = new Intent(this, HomeActivity.class);
notificationIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
PendingIntent pendingIntent=PendingIntent.getActivity(this, 0,
        notificationIntent, 0);
Notification notification=new NotificationCompat.Builder(this)
        .setSmallIcon(RESOURCE_ICON)
        .setContentTitle("BlaBlaFood")
        .setContentText("Pubblicazione dell'annuncio in
corso...")
        .setOngoing(true)
        .setContentIntent(pendingIntent)
        .build();

startForeground(NOTIFICATION_ID, notification);
```

Mediante il metodo startForeground a cui si passa la notifica configurata si mette il service in foreground; la notifica verrà tolta automaticamente al termine dell'operazione.

Sviluppi futuri

Nel prossimo futuro si provvederà al testing in maniera più concreta rispetto a quanto fatto durande lo sviluppo; il sistema è una prima versione completa nelle sue funzionalità ma comunque migliorabile; per esempio un miglioramento potrà essere quello riguardante la gestione della posizione e degli indizzi in situazioni offline.

Il sistema è stato realizzato in modo da poterlo estendere in maniera piuttosto facile. Per quanto riguarda l'applicazione Android si possono aggiungere delle funzionalità che permettano agli utenti di migliorare la loro conoscenza sulle tematiche alimentari, mediante delle news o dei wiki riferiti ai prodotti.

L'integrazione con i Social Network è già presente ma può essere sicuramente aumentata per esempio aggiungendo funzionalità come “Mi piace” per Facebook o “+1” di Google+; si potrà valutare anche l'integrazione con altri Social Network in quanto si ritiene che le reti sociali influenzino molto la probabilità di successo del sistema.

Nei prossimi mesi il completamento e raffinamento dell'applicazione porterebbe alla pubblicazione della stessa.

Per completare il servizio sarebbe utile programmare lo sviluppo di un Client web e un'applicazione iOS. Prima di investire ulteriori risorse, è necessario procedere prima con ulteriori test e interviste per valutare se il servizio soddisfa una reale esigenza.

Queste sono solo alcune osservazioni in merito allo stato attuale e futuro del sistema BlaBlaFood, in corso d'opera si valuterà e si rettificherà il planning di sviluppo e testing, al fine di migliore l'affidabilità e il valore del sistema .

Conclusioni

Al termine di questa esperienza si può dire che sono complessivamente soddisfatto del lavoro svolto; è stato bello poter applicare concretamente la conoscenza acquisita durante gli studi in un progetto completo come questo. Di vitale importanza nella riuscita del progetto è stato seguire le buone pratiche di suddivisione del lavoro tipiche del ciclo di sviluppo del software. Una buona e attenta progettazione ha richiesto molto tempo, ma grazie ad essa si è potuto valutare nel dettaglio molti aspetti: dalla struttura del sistema a considerazioni sulle performance come l'uso parsimonioso delle risorse.

Porre attenzione ad aspetti di user experience è stato bello ed intrigante, in quanto è una tematica che piace particolarmente al sottoscritto; la cura dei flussi di navigazione e l'interazione con l'utente è stata progettata nei minimi particolari considerando anche un utilizzatore non molto esperto. Sicuramente si può fare ancora meglio ma il risultato ottenuto è piacevole alla vista e funzionale.

Sarebbe bello continuare a sviluppare il sistema continuando a imparare e migliorare nel completarlo. La pubblicazione dell'applicazione nello store sarebbe un bel traguardo ma occorre valutare bene e testare l'intero sistema a dovere come si è detto negli *Sviluppi futuri.*

Bibliografia

(1) *https://en.wikipedia.org*, *https://it.wikipedia.org*

(2) La Redazione, *E-commerce alimentare in aumento le vendite*
8 Gennaio 2016
http://www.fastweb.it/web-e-digital/e-commerce-alimentare-in-aumento-le-vendite/

(3) Coldiretti, *Statistiche*
3 Aprile 3016
http://www.coldiretti.it/News/Pagine/251-%E2%80%93-3-Aprile-2016.aspx

(4) L'alveare, app iOS
https://itunes.apple.com/it/app/lalveare-che-dice-si-acquista/id1052198033?mt=8

(5) http://www.subito.it

(6) https://it.letgo.com/it

(7) https://developers.google.com/maps/documentation/geocoding/intro

(8) https://www.scribd.com/presentation/2569355/Geo-Distance-Search-with-MySQL

(9) https://material.google.com

(10) https://developers.google.com

(11) https://developer.android.com

(12) http://stackoverflow.com

(13) https://developers.facebook.com

(14) http://square.github.io/picasso

(15) https://www.gardainformatica.it/blog/sviluppo-software/perche-microsoft-sql-server-non-e-la-scelta-migliore-come-relational-database-management-system-rdbms

Figure

Le figure inserite nel testo sono del sottoscritto ad eccezione di quelle riportate in questo elenco dove viene indicata la fonte:

-*Figura3.5*
http://confluence.tapcrowd.com/pages/viewpage.action;jsessionid=21062D4A7BE40E3B7E2D7F33FA95C7D4?pageId=2262404

-*Figura 4.6*
https://material.google.com/patterns/permissions.html#permissions-request-patterns

-*Figura4.7*
https://raw.githubusercontent.com/ParkSangGwon/TedPermission/master/Screenshot_cases.png

-*Figura 4.11*
https://material.google.com/components/buttons.html#buttons-button-types

Ringraziamenti

Al termine di questo percorso universitario ringrazio tutti coloro che mi hanno sostenuto nel portarlo a termine al meglio.
Un primo e sentito grazie al dott. Mirko Ravaioli che mi ha dato la possibilità di collaborare con lui; è stata un'esperienza che mi ha fatto crescere molto grazie ai suoi consigli, alla sua esperienza e alla sua disponibilità.
Grazie alla mia famiglia che mi ha dato l'aiuto nei momenti più particolari, anche quando problemi di salute e indecisioni stavano interrompendo questo percorso.
Un grazie anche a tutti i compagni di corso e amici che mi hanno incoraggiato durante gli esami e nella preparazione di questa tesi.