Università degli studi di bologna
Corso di laurea in Informatica

Relazione finale progetto
Laboratorio Applicazioni Mobili 2015/2016

Applicazione Android: Walli

16 luglio 2016

Davide Quadrelli
matricola 0000694103

Indice

1 Introduzione

Walli è un'applicazione per smartphone Android che nasce per agevolare chiunque abbia l'esigenza di effettuare spese condivise con altre persone: che si tratti di fare un regalo di gruppo ad un amico, gestire le bollette con i propri coinquilini o di fare una vacanza con i propri amici, non c'è bisogno di fare i conti di quanti soldi ognuno debba agli altri, perché ci pensa l'applicazione.
Walli tiene traccia di tutti i soldi che ogni persona deve agli altri, sia all'interno di un singolo gruppo che fra tutti, gestendo anche valute di denaro differenti: se si dovessero fare in un gruppo alcune spese in Italia e in un secondo gruppo altre spese in America, ed alcuni utenti dovessero essere in comune fra i due gruppi, si potrebbero saldare tutti i debiti insieme nella valuta preferita dall'utente.

2 Funzionalità

L'applicazione, previa registrazione, permette di creare nuovi gruppi. Per registrarsi sono richiesti un nome utente, una mail e alcuni dati personali (nome, cognome, cellulare). La creazione di un gruppo richiede un nome, una foto (facoltativa), la scelta di quale valuta utilizzare per le spese che si inseriranno e almeno un'altra persona con cui condividerle: il creatore del gruppo ne sarà automaticamente l'amministratore. Sia i dati dell'utente che quelli del gruppo sono personalizzabili: è possibile cambiare email, dati personali e immagine per l'utente, nome e valuta per il gruppo. Non è possibile inserire un'immagine profilo in fase di registrazione di un nuovo account, ma solo successivamente e la modifica/eliminazione del gruppo è consentita solo all'amministratore dello stesso.

2.1 Gruppi e spese

La creazione di un gruppo richiede di inserire un nome e di scegliere la valuta da utilizzare per le spese all'interno di esso: le valute disponibili sono tutte quelle supportate dal sistema operativo del dispositivo che si utilizza, ed è possibile cambiare in qualsiasi momento quella con cui eseguire le conversioni nelle impostazioni di Walli.
Una volta generato un gruppo, si possono cominciare ad inserire le spese effettuate oppure le "notifiche di spesa", ovvero un promemoria per una spesa che non è stata ancora fatta, ma che è necessaria; ad esempio, nel gruppo condiviso con i propri coinquilini, si può ricordare il fatto che serva l'olio in casa e, il primo componente che potrà, lo andrà a comprare, inserendo la cifra che ha speso. Tale funzionalità può anche essere utile per poter stilare un elenco di ciò che serve prima di partire per una vacanza o per poter stilare una lista dei preparativi per una festa, rimuovendo i singoli elementi, una volta comprati, inserendone il prezzo. In ogni gruppo è possibile controllare la situazione finanziaria, controllando quali spese sono saldate e quali no, l'elenco delle spese ancora da fare ed, infine, chi sono tutti gli utenti partecipanti.
L'amministratore si occupa di gestire gli utenti nel gruppo: può modificare l'elenco dei partecipanti in qualsiasi momento, rimuovendo anche chi non ha ancora pagato tutti i suoi debiti (essi saranno comunque saldabili e visibili dalla schermata dei crediti/debiti globali); inoltre, è l'unico, oltre ai creatori delle spese e

delle notifiche spese, a poterle rimuovere (solo se nessuno ha ancora pagato la propria quota).

2.2 Crediti e debiti

Quando cominciano ad esserci spese in un gruppo, si comincia ad essere in credito o debito verso gli altri individui: nel caso di credito, è possibile segnare come pagato il debito di un'altra persona o ricordargli che deve ancora pagare (al massimo una volta ogni 24 ore); nel caso in cui si vogliano saldare tutti i debiti verso una persona, indipendentemente dalle cifre singole all'interno di ogni gruppo (e quindi dalla valuta), Walli mostra, nella schermata dedicata, il credito/debito complessivo convertito nella valuta preferita dall'utente.
Per agevolare gli utenti nella gestione delle proprie spese, sono disponibili anche due grafici: il primo grafico, a torta, mostra l'attuale situazione di debiti/crediti personale, mentre il secondo, a linee, mostra il rapporto fra spese ed entrate nel tempo: nel secondo grafico è possibile personalizzare il lasso temporale nel quale mostrare i dati (che sono convertiti, come di consueto, nella valuta scelta dall'utente).

2.3 Notifiche push

Se si rimane collegati col proprio account nell'applicazione, vengono ricevute dal dispositivo notifiche push per avvisare l'utente dell'avvenimento di operazioni importati, come essere inseriti in un nuovo gruppo, l'aggiunta di una nuova spesa o una nuova notifica di spesa, la segnalazione da parte di un altro utente che un nostro debito è stato saldato o la richiesta che ci venga inviato un reminder per pagare un debito; quest'ultimo servizio è limitato ad una notifica al giorno per evitare spam, probabile causa di immediata disinstallazione di Walli dal dispositivo.

3 Caratteristiche e requisiti

Walli è sviluppata per dispositivi Android con versione del sistema operativo Marshmallow (API 23), con compatibilità con le versioni precedenti fino a JellyBean (API 16): si è fatta tale scelta per consentire l'utilizzo dell'applicazione sulla maggioranza dei dispositivi Android esistenti (il 96% circa) senza esagerare a supportare versioni troppo datate. Per l'esecuzione dell'applicazione è consigliato un dispositivo con almeno 512MB di RAM e un "VM heap size" (dimensione dello spazio allocabile dalla Virtual Machine) di almeno 32MB, altrimenti Walli non riesce ad essere eseguito correttamente.
L'app è stata sviluppata in ambiente Windows, utilizzando l'ultima versione di Android Studio attualmente disponibile (2.1.2) e l'ultima versione di Gradle (2.1.2), mentre il testing è stato eseguito nei seguenti ambienti:

1. Xperia Z2
    - Schermo: 5.2" 1080x1920 (424 PPI)
    - Android: Marshmallow 6.0.1 in beta (API 23)
2. Samsung Galaxy S4
    - Schermo: 5" 1080x1920 (441 PPI)
    - Android: custom rom basata su Lollipop 5.0.1 (API 21)
3. Samsung Galaxy S2
    - Schermo: 4.3" 480x800 (217 PPI)
    - Android: Jelly Bean 4.1 (API 16)
4. Mediacom Smartpad S4
    - Schermo: 10.1" 800x1280 (148 PPI)
    - Android: Jelly Bean 4.2.2 (API 17)
5. Emulatore Nexus 6
    - Schermo: 5.96" 1440x2560 (560 PPI)
    - Android: Lollipop 5.1 (API 22)
6. Emulatore Nexus 7
    - Schermo: 7.02" 1200x1920 (323 PPI)
    - Android: Kitkat 4.4 (API 19)
7. Emulatore device generico
    - Schermo: 2.7" 240x320 (148 PPI)
    - Android: Kit Kat 4.4 (API 19)

3.1 Design

L'interfaccia utente cerca di rispettare il più possibile il Material Design: questa scelta permette semplicità e chiarezza nell'interfaccia grafica, oltre che uniformità con la maggior parte delle applicazioni che un utente medio è abituato a usare ogni giorno. Tutti i colori all'interno dell'applicazione e le icone rispettano tale design, tranne l'icona dell'applicazione, perchè non è stata presa dalle icone Material della Google ma creata dallo sviluppatore stesso.
L'interfaccia è progettata principalmente per schermi di dimensioni piccole-medie (4-5"), anche se perfettamente utilizzabile sia su schermi di dimensioni maggiori sia di dimensioni minori, con varie densità di pixel.

4 Progettazione e scelte implementative

L'applicazione si divide in client e server e la progettazione dell'applicazione ha dovuto tenere conto di tale divisione: entrambi seguono il modello MVC (Model View Controller), per quanto le differenze fra i due siano sostanziali.

4.1 Permessi

Per il corretto funzionamento dell'applicazione, sono necessari i permessi di accesso a internet e i permessi di lettura e scrittura sulla memoria esterna: la necessità dell'accesso alla rete nasce dal fatto che il database è su un server remoto centrale, mentre la scrittura sul file system serve per poter fare caching delle immagini dei profili degli altri utenti e dei gruppi, in maniera da ottimizzare il consumo della rete dati.

4.2 Framework esterni

Per facilitare lo sviluppo dell'applicazione, si sono scelti alcuni framework per gestire svariate operazioni: AppCompat per la compatibilità di tutte le funzioni grafiche del Material Design con le versioni di android precedenti a Lollipop, CircleImageView per avere un widget simile a ImageView ma con una cornice dell'immagine rotonda, MPAndroidChart per la creazione e la gestione dei grafici e, infine, Firebase per implementare le notifiche push. Durante lo sviluppo, si è reso necessario utilizzare il meccanismo di multidexing per permettere a Gradle di poter compilare l'applicazione (si era raggiunto il limite massimo di funzioni per un unico file dex) e l'aggiornamento di AppCompat per aggiungere la conversioni di immagini vettoriali in bitmap, poiché le immagini vettoriali sono supportate nativamente in Android solo dalle API 20 in poi.

4.3 Activity

Il client si suddivide in una activity per il login, una per la registrazione di un nuovo account, l'activity principale (dalla quale si può accedere a tutte le funzionalità principali dell'applicazione) e le activity che gestiscono l'interfaccia per i singoli task da eseguire (creazione, modifica e visualizzazione dei dati riguardanti un gruppo, aggiunta e modifica di una spesa e di un reminder per una spesa). L'activity principale contiene più schermate al suo interno: si è lasciato tutto su un'unica activity invece che usarne di più per poter implementare un

menu laterale di navigazione. Relativamente alla gestione di tali schermate, si è utilizzato il widget ViewFlipper: lo si è preferito rispetto ad un fragment per ogni schermata perché presentava un'implementazione più semplice e immediata. L'utilizzo dei fragment avrebbe ottimizzato leggermente l'applicazione, ma essendo il numero di schermate statiche e decisamente limitato (e lo rimarrebbero molto probabilmente anche con l'aggiunta di future nuove feature), non ci sarebbe stato un miglioramento notevole.
Ogni activity deve poter aver accesso ai dati salvati nel database centrale dell'applicazione: ogni connessione viene eseguita da un thread secondario per lasciare a quello principale solo il compito di gestire la logica locale dell'applicativo e la grafica. Alla fine di ogni richiesta al server (sia di invio che di recupero dati), il thread principale viene notificato del risultato dell'operazione, gestendone il risultato.

4.4 Caching

La gestione degli utenti e dei gruppi ha generato un problema: avendo, in linea teorica, ogni utente e gruppo un immagine profilo, è necessario un meccanismo di caching delle immagini. Android offre alcuni meccanismi utili per questo scopo, ma si è preferito crearne uno ex novo, in modo da salvare le immagini in una cartella dello storage disponibile all'utente, e quindi lasciandogliene libero accesso. Si è, quindi, creata una cartella nominata come l'applicazione, con all'interno due cartelle e un file: le prime contengono rispettivamente le immagini degli utenti e quelle dei gruppi, mentre il file contiene un codice univoco per ogni immagine.
Questo codice, che si tratta di un timestamp (indice temporale univoco), permette di aggiornare un immagine solo quando la versione remota è più recente di quella locale: interrogando il server, si controlla se il codice ricevuto è lo stesso in memoria centrale, ottenendo una risposta affermativa o negativa che avvisano il client se dover aggiornare l'immagine o meno. Tutto questo meccanismo genera un quantitativo abbastanza elevato di richieste, ma tutte di dimensioni contenute e inferiori rispetto a quella di un immagine, in modo da dover inviare e ricevere meno dati possibili.

4.5 Elenchi

Ogni elenco presente nell'applicazione è generato utilizzando un RecyclerView con un custom Adapter per dare ad ogni elemento delle liste un particolare layout: si è optato per questo widget rispetto al classico ListView perchè la RecyclerView tiene in memoria un numero limitato di elementi, rigenerando gli altri quando necessario riciclando gli elementi non visibili al momento (ovvero durante lo scrolling della lista): questa ottimizzazione è fondamentale e la scelta della RecyclerView è risultata obbligatoria.

4.6 Server

Il server è un Ubuntu Server 14.04 LTS, con applicativo scritto in NodeJS, configurato per comunicazioni HTTP sicure e database MySQL. Il server non si deve occupare solo di eseguire query sul database e inviare i dati di risposta al client, ma anche di gestire le conversioni di valuta e le immagini di profilo e dei gruppi.

Per la gestione delle immagini si è scelto semplicemente di creare sul filesystem due cartelle (una per utenti e una per gruppi) dove memorizzare le immagini (convertite tutte in formato png e con nome uguale ad un identificativo univoco dell'entità a cui sono legate) e di utilizzare un dizionario per indicizzarle: questo perchè si tratta di una soluzione estremamente semplice, efficace e di rapida implementazione.
La conversione delle valute avviene con l'utilizzo del modulo di nodeJS money, che esegue i calcoli veri e propri, e le API di Open Exchange Rates, che forniscono via HTTP i valori di ogni singola valuta in tempo reale. Da quest'ultimo servizio si ottiene un JSON contenente le sigle delle valute e il loro corrispondente valore, con il quale money esegue i calcoli.
Il limite di richieste delle valute è di 1000 al mese, ma si aggiornano comunque solo una volta ogni 24 ore: questo perché non si è interessati alle minime oscillazioni che avvengono minuto per minuto, ma solo ad avere conversioni corrette e accettabili rispetto ai valori attuali delle monete mondiali.

4.7 Messaggi asincroni e notifiche push

Oltre alle classiche comunicazioni sincrone client-server, tramite le API Google di Firebase Messaging, è possibile ricevere messaggi asincroni sul client, indipendentemente se l'applicazione sia in foreground, background o non in esecuzione: questo è possibile grazie a un servizio indipendente dalla vita dell'applicazione che riceve i messaggi e li gestisce. Per generare dalla ricezione di un messaggio di Firebase una notifica push, è stata implementata una funzione all'interno di Firebase per leggere i dati del messaggio e costruire una semplice notifica Android in base ad esso. Questa personalizzazione del codice è stata necessaria per poter utilizzare un custom layout delle notifiche: se non fossero stati necessari questi cambiamenti, Android avrebbe potuto gestire in automatico le notifiche, senza richiedere alcun intervento di codice client di Walli.
Le operazioni che possono generare notifiche sono:

- Creazione di un nuovo gruppo
- Inserimento di una spesa o di un reminder di spesa da fare
- Un utente segna come pagato un nostro debito
- Un utente manda un reminder per il pagamento di un nostro debito

Il server deve, quindi, comunicare a Firebase quali dispositivi contattare e quali informazioni inviare; per farlo ha al suo interno un dizionario di utenti-token che gli permette di conoscere quali siano i dispositivi interessati all'evento.

4.8 Sicurezza

Tutte le comunicazioni in ingresso ed in uscita, sia dal client che dal server, sono criptate con SSL, in modo da proteggere i dati sensibili trasmessi. Per un ulteriore livello di sicurezza, il database è configurato per avere un solo utente a cui è permesso l'accesso (con password) solo da "localhost": ciò consente soltanto connessioni locali al databse, impedendo l'esposizione dei dati in maniera diretta alla rete.
L'accesso alla macchina che esegue il processo(server Amazon con un 1 core da

1Ghz, 1GB di RAM, 30GB di disco SSD) è possibile solo tramite SSH con autenticazione tramite chiave privata: se non si è in possesso della chiave, generata da Amazon Web Services in fase di creazione del server, ogni collegamento verrà rifiutato. I soli servizi esposti alla rete sono il processo di NodeJS (HTTPS) e il server mail(POP3, IMAP e SMTP) per mandare la conferma dell'iscrizione e gestire il caso in cui un utente dimentichi la sua password: tutte le porte rimanenti sono chiuse sia dal firewall della macchina sia dal firewall di Amazon (che filtra tutte le connessioni in entrata).
Come ultimo meccanismo di protezione dati,la password viene memorizzata nel database con un meccanismo di doppio hashing: il client esegue un primo hash con algoritmo MD5 della password prima di inviare la richiesta di login al server; il server ripete l'operazione e utilizza il digest ottenuto come dato da memorizzare; così facendo, la password non viene mai trasmessa in maniera diretta.
Ad ogni login eseguito correttamente, inoltre, viene mandato al client una chiave di accesso univoca per dispositivo: senza di essa non è possibile eseguire alcuna operazione e, nel caso di utilizzo di due dispositivi differenti con lo stesso account, ogni dispositivo avrà la propria chiave. Ogni richiesta successiva al login, che sia per modificare dati o per ottenerne, richiederà l'identificativo utente e la chiave univoca: se i due campi non dovessero combaciare fra di loro, l'operazione verrà negata.

5 Estensioni future e ottimizzazioni

L'applicazione offre già tutte le funzionalità base per l'obiettivo che si prepone, però ci sono svariate possibili migliorie ed estensioni che potrebbero essere inserite. Ecco un breve elenco di alcune di esse.

- Migliorare la cache
  - Si può sfruttare lo stesso meccanismo usato per le notifiche push (cioè le API di Firebase) per notificare al dispositivo che un immagine di un gruppo o di un profilo è stata cambiata. Si riceve tale informazione in background e la si memorizza: la prima volta che si aprirà l'applicazione, Walli saprà già quali immagini dovrà scaricare, senza interrogare il server, riducendo sensibilmente l'utilizzo della rete.
- Pagamento con Paypal
  - Si potrebbe inserire la possibilità di pagare i propri debiti verso gli altri utenti tramite il proprio account Paypal, visto che per fare un pagamento basta conoscere la mail del destinatario (già data in fase di iscrizione). Sfruttando le API di Paypal, si possono gestire pagamenti tra utenti e pagamenti multipli, ma questa funzionalità, denominata Payout, è limitata solo all'interno degli Stati Uniti.
- Selezione multipla
  - Attualmente si può selezionare solo un elemento alla volta negli elenchi mostrati all'utente (elenco crediti/debiti, lista spese e lista spese da fare): consentire una selezione multipla potrebbe migliorare sensibilmente l'utilizzo dell'applicazione, rendendola molto più "user friendly".
- Assegnamento delle spese da fare
  - Si potrebbe permettere di assegnare ad un utente o di potersi prendere in carico una o più spese ancora da fare, in modo che gli altri utenti sappiano di non doversene occupare; ciò può prevenire possibili spese doppie, migliorando l'affidabilità dell'applicazione.
- Login tramite social network
  - Per velocizzare la fase di iscrizione, che può esssere noiosa per un utente medio, è possibile integrare il login via Facebook, Twitter e Google+, in modo da consentire all'utente di utilizzare account che già ha e che conosce bene per utilizzare Walli, rendendo minore la probabilità di dimenticarsi le credenziali di accesso.
- Rendere asincrone la maggior parte delle richieste client
  - Ogni richiesta client avviene in maniera sincrona, ovvero si richiede al server di ricevere i dati aggiornati sui gruppi, sulle spese, ecc. Se il server, quando riceve una richiesta di modifica dati, inviasse un messaggio asincrono (tramite Firebase) ai dispositivi interessati a quell'azione, il client potrebbe implementare un database locale, sincronizzato con quello remoto, contenente solo le informazioni utili per un utente, senza dover eseguire alcuna richiesta sincrona.

6 Conclusione

L'applicazione ha svariati margini di miglioramento e di integrazione di nuove funzionalità che possono migliorarne decisamente l'utilità e che ottimizzerebbero i consumi di rete dati, ma gli scopi base che Walli si pone di svolgere vengono già correttamente gestiti, senza richiedere all'utente nulla che non sia strettamente necessario. L'interfaccia è abbastanza user-friendly e non sono presenti funzioni (o bug) che potrebbero portare un utente a decidere di disinstallare l'app, se non per un effettivo inutilizzo.

7 Referenze

- Icone Material Design - https://design.google.com/icons/
- CircleImageView - https://github.com/hdodenhof/CircleImageView
- MPAndroidChart - https://github.com/PhilJay/MPAndroidChart
- Firebase - https://firebase.google.com/docs/android/setup
- Money NodeJS - https://www.npmjs.com/package/money
- API Openexchanges - https://openexchangerates.org/
- Tutorial Firebase Messaging da server remoto - https://www.youtube.com/watch?v=LiKCEa5_Cs8

Altri riferimenti sono disponibili all'interno del codice sorgente nelle funzionalità che sono state rese possibili grazie ai suggerimenti, all'aiuto e agli esempi di codice trovato su repository GitHub (sia di Google che di terze parti) e su StackOverflow.