Scuola di scienze - Corso di laurea in Informatica

Relazione laboratorio di applicazioni mobili

Walli - Applicazione gestionale per spese tra utenti

Daniele Piergigli
0000691460

14 luglio 2016

Indice

1 Scopo dell'applicazione

Walli è una applicazione di gestione delle spese tra gruppi di utenti, che permette di tenere sempre aggiurnati i crediti e i debiti che si hanno verso altre persone. Lo scopo principale è quello di sapere sempre quanti soldi si devono ad altre persone in generale o solo in uno specifico gruppo di spesa.
Walli è nato dalla necessità di tenere traccia delle spese tra coinquilini dovuta al fatto che facendo piccoli acquisti per la casa è difficile tenere traccia delle piccole spese che si hanno verso le altre persone della casa e si finisce per non avere più idea di quanti soldi si deve a un altro. L'idea è stata quindi poi espansa al concetto di gruppo che può essere anche solo di due persone e che può riguardare qualsiasi ambito della gestione delle spese.
Da tenere in considerazione è anche la sicurezza che deriva dal mantenere salvata una spesa, questo evita che vi siano delle richieste da parte di altri utenti di spese che sono già state pagate andando ad ampliare l'utilizzo di questa applicazione.

2 Funzionalità previste

- Effettuare login e logout dall'applicazione;
- Recuperare la password dimenticata;
- Effettuare una registrazione per ricevere un account;
- Aggiungere e rimuovere un gruppo;
- Aggiungere o modificare gli utenti di un gruppo;
- Modificare e gestire le immagini;
- Creare un grafico delle spese effettuate dall'utente;
- Modificare il proprio profilo;
- Pagare le spese verso un utente;
- Aggiungere una spesa e una notifica;
- Modificare una spesa e una notifica;
- Eliminare una spesa e una notifica;
- Cambiare la valuta preimpostata;
- Inviare una notifica se una propria spesa non è ancora stata pagata da un altro utente a quell'utente.

3 Caratteristiche e requisiti

3.1 Caratteristiche

L'applicazione è in grado di tenere sempre aggiornate le informazione grazie a richieste continue al server, in questo modo però le richieste sprecano continuamente risorse di rete. Per questo motivo è stato aggiunto un database interno

che aggiurna costantemente i dati all'interno del telefono e che blocca richieste non necessarie andando a limitare l'uso della rete e permettendo una gestione dei dati molto più economa e rapida anche nella visualizzazione.
Il landscape è stato bloccato in modo da visualizzare le informazioni in modo verticale sia su IPhone che su Ipad, una prova è stata svolta a riguardo per la visualizzazione orizzontale ma i grafici non si adattavano bene alla struttura con cui era stata pensata Walli per cui non è stato preso in considerazione.
Le notifiche push sono state utilizzate in quanto le persone dovrebbero essere sempre aggiornate di ogni spesa inserita e ogni notifica inviata in modo da avere in ogni momento la possibilità di rispondere pagando o rimuovendo un qualcosa.

3.2 Requisiti

Il progetto è stato sviluppato seguendo le linee guida apple per sviluppatori utilizzando l'ultima versione in commercio di IOS, cioè la 9.3 per IPhone e IPad. Xcode è aggiornato all'ultima versione, cioè la 7.3.1, con una licenza da sviluppatore attiva. Swift 2.2 è stato il linguaggio adottato per la programmazione di tutta la parte client del progetto, per quanto riguarda la parte server, NodeJS e MySql sono stati utilizzati per le chiamate e la gestione dei dati.
Le librerie scaricate sono state implementate utilizzando due programmi di gestione dei framework.
Carthage, per tutte le librerie utilizzate e spiegate successivamente, in quanto, utilizzare un framework occupa molto spazio, e con questo programma viene lasciata la libertà di implementare un framework per una piattaforma specifica senza andare a appesantire un progetto con tutte le versioni che questo implementa.
Cocoapods d'altro canto è stato utilizzato solo per Firebase in quanto non è stato fornito un framework per questa libreria a Carthage negandone l'implementazione.

4 Progettazione

4.1 Librerie

Sono state utilizzate diverse librerie per soperire ad alcuni aspetti complicati della programmazione in Swift.

- **Alamofire**: Utilizzato per le richieste al server di dati, è stato fondamentale per permettere una gestione chiara e sisntetica dei completionHandler;
- **SwiftyJSON**: Per il parsing dei JSON di risposta alle chiamate;
- **Realm**: Per una gestione ottimale dei dati salvati all'interno del cellulare è stato di fondamentale importanza, ha permesso la creazione di un database locale senza andare a sprecare risorse di rete inutilmente. Inoltre è molto più veloce e leggero di CoreData direttamente implementato in IOS;
- **IOS-Charts**: Utilizzato per la creazione dei grafici;
- **CryptoSwift**: Utilile per la creazione di password sicure da inviare al server in doppio MD5;

- **Firebase**: Il nuovo sistema di APNs di Google è molto veloce nel inviare notifiche push e permette la gestione anche mediante server prioritario senza per forza dover trasferire i dati completamente nel loro server, dando la possibilità a gli sviluppatori di progettare le chiamate per le notifiche push secondo le loro preferenze.

4.2 Scelte implementative

L'utilizzo delle **UITableView** è stato cruciale per permettere la corretta visualizzazione dei contenuti. Sfruttate le celle ha permesso di ripetere un layout, di modo da mostrare dati simili nella struttura, ma differenti nei contenuti senza appesantire il codice.
Il tutto viene gestito dall' UITabBarController che si muove tra le quattro categorie principali dell'applicazione e permette di accedere rapidamente a gli utilizzi principali dell'app in modo intuitivo.
Per quanto riguarda la schermata deglu utenti di un gruppo una scelta adeguata è sembrata quella del UICollectionView, con una visualizzazione più grande delle immagini e delle azioni possibili da compiere su un singolo utente del gruppo.
L'utilizzo delle storyboard per il 90% dei passaggi da una view alla successiva, è stato utilizzato per mantenere attivo il sistema del navigation controller che si trovava in difficoltà a gestire più segue uno dopo l'altro.

5 Difficoltà

5.1 Ricevere dei dati nel chiamante quando la view chiamata viene abbandonata

I protocolli sono stati molto utili per risolvere il problema, nella seconda view è stato definito il protocollo con la chiamata della funzione dichiarata nella prima view, mentre nella prima view è stato aggiunto il delegate alla seconda view.

5.2 Utilizzare i popover non ancorati a un UITabBarButton

Il popover è ancorato al Bottom Layer della view corrente, così quando la chiamata al popover viene eseguita, il popover emerge dal basso e permette di essere cliccato con il pollice più facilmente.

5.3 Gestione delle chiamate al server in modo sincrono

Il competionHandler è stato utilizzato per rimuovere questo problema, aspettando come return della chiamata un dato, che doveva prima essere restituito dal metodo che comunicava con il server.

5.4 Gestione delle chimate al server in modo sincrono all'interno di un ciclo for

Risolto utilizzando la chiamata tramite completionHandler all'interno del metodo di popolamento delle celle di una tableview, in modo da caricare singolarmente ogni dato richiesto al server.

5.5 Notifiche push in background

Per risolvere il problema, è stato implementato il sistema di Payload delle notifiche in background nel server, in modo da restituire i dati nella forma letta dal client APNs installato nel cellulare.

6 Aspetti rilevanti

Organizzare l'applicazione come fosse una chat non è stato fatto per moda del momento delle app di messaggistica ma bensi con un criterio ben ponderato. L'interfaccia permette di passare da una categoria all'altra in maniera rapida, senza dover per forza aprire menu laterali, e in modo piuttosto intuitivo. Dare a le persone un metodo di gestione dell'applicazione simile a quello già utilizzato nella quotidianità, permette di approcciarsi con l'app sin da subito velocemente. Questo permette all'utente un approccio all'applicazione più gratificante e meno difficile rispetto a un applicazione che non si adatta a gli standard canonici dall'utente ordinario.

7 Casi d'uso

7.1 Notifica a un utente

Poniamo per esempio di esserci scordati di comprare il pane e di voler notificare a qualcuno che è ancora in giro di andarlo a comprare.

- Apro l'applicazione e scelgo il gruppo dei coinquilini in casa a Bologna ad esempio;
- Accedo ora alla lista delle spese tramite l'apposito bottone e premo sul pulsante di invio notifica;
- Compilo il campo con la descrizione adeguata ('Compra pane per cena' ad esempio) e invio;
- Una notifica push arriverà a tutti i componenti del gruppo per avvertirli della notifica così da permettergli di andare a acquistare il pane.

7.2 Trasformazione di una notifica in una spesa

Seguendo il ragionamento precedente ora sono al supermercato e arrivatami la notifica voglio notificare l'acquisto del pane da parte mia.

- Apro l'applicazione e scelgo il gruppo contenente la notifica;
- Accedo ora alla lista delle spese tramite l'apposito bottone e premo sul pulsante di 'Buy' che si trova affianco della notifica appropriata;
- La notifica si trasforma in una spesa e mi permette di aggiungere un costo;
- Una notifica push arriverà a tutti i componenti del gruppo per avvertirli della spesa e questa viene ripartita tra i membri del gruppo automaticamente senza stare a fare conti successivi.

7.3 Saldare un pagamento e inviare un avviso

Arrivatami la notifica di generazione di una spesa, voglio controllare quanto devo a la persona che ha comprato il pane in totale.

- Apro l'applicazione e scelgo la sezione 'Pagamenti';
- Controllo e in realtà sei tu a dovermi dei soldi, in quanto il bottone non contiene la scritta 'Pay' ma un aeroplanino di carta che sta a significare la possibilità di inviare un avviso di pagamento;
- Premo su l'aeroplanino e ti invio un avvertiento per ricordarti di pagarmi.

8 Estensioni future

Il lavoro all'interno dell'applicazione non si è ancora concluso e quindi è importante avere a mente una serie di future implementazione per l'applicazione:

- Pagamento tramite PayPal e ApplePay dei debiti;
- Accesso all'applicazione mediante login con Facebok e Google+;
- Controllo su chi ha pagato o no una determinata spesa;
- Gestione migliore delle notifiche push tramite lockscreen;
- Scanner di uno scontrino mediante fotocamera;
- Gestione dell'immagine da salvare come foto profilo o del gruppo;
- Traduzione dell'applicazione i più lingue (italiano, francese, spagnolo, tedesco, ...).

9 Conclusioni

Walli è una applicazione che ogni persona dovrebbe avere nel proprio telefono, ognuno di noi ha sempre qualche debito o credito e tenerne traccia, non fa che rendere il tutto più semplice da ricordare e organizzare.
Per questo nuove funzionalità devono essere implementate per rendere allettante l'acquisto dell'applicazione che anche se gratis, permette di guadagnare utilizzando le informazioni condivise dai soggetti che la utilizzano.
Gli utilizzi e gli eventuali impieghi sono vasti, sia dalla parte di chi utilizza l'applicazione, sia da parte nostra, come sviluppatori, che possiamo generare guadagni vendendo a supermercati e agenzie di viaggi informazioni su mete di viaggi e beni acquistati tramite la descrizione delle spese degli utenti.

10 Fonti

Si ringraziano la comunità di StackOverflow per i metodi su il ridimiensionamento di un immagine e il controllo di attivazione della rete internet.
Gli utenti di youTube, per le dimostrazioni sull'utilizzo delle notifiche push e la richiesta di un certificato da apple per le notifiche in background.