



SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione e sviluppo di un'applicazione di Home Banking in ambiente iOS

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea Magistrale in Informatica

Candidato

Mario Greco

Matricola 984280

Relatore

Prof. Emanuele Panizzi

Correlatore

Dott. Raffaele Gitto

Anno Accademico 2013/2014

Tesi non ancora discussa

Progettazione e sviluppo di un'applicazione di Home Banking in ambiente iOS
Tesi di Laurea Magistrale. Sapienza – Università di Roma

© 2014 Mario Greco. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Versione: 26 febbraio 2014

Email dell'autore: mrgreco3@gmail.com

Dedicato a

Sommario

.

Ringraziamenti

Indice

1	Introduzione al progetto	1
1.1	Descrizione dell'applicazione	1
1.2	Primi requisiti funzionali	1
1.3	Primi mockup, wireframe e prototipi	2
1.3.1	Mockup	2
1.3.2	Wireframe	5
1.3.3	Prototipi	8
2	Metodologie usate e raffinamenti successivi	9
2.1	Metodologia Agile	9
2.2	Requisiti funzionali finali	9
2.3	User Experience e usabilità	14
2.3.1	iOS Human Interface Guidelines	14
2.3.2	Esempi di GUI e raffinamenti successivi	15
3	Casi d'uso	23
3.1	Diagrammi dei casi d'uso	23
3.1.1	Caso d'uso: Ricarica telefonica	23
3.1.2	Caso d'uso: Ricarica telefonica	23
3.2	Specifica dei casi d'uso	25
3.2.1	Caso d'uso: Ricarica telefonica	25
3.2.2	Caso d'uso: Visualizza lista conti e carte e Visualizza lista movimenti	29
4	Architettura software	33
4.1	Architettura generale	33
4.2	Architettura applicazione	33
4.3	Diagramma delle classi	35
4.3.1	Connessione ai servizi	35
4.3.2	Caching dei dati	35

4.3.3	Autenticazione e sicurezza dei dati	36
4.3.4	Controller principali	38
5	Dettagli implementativi	41
5.1	Custom container controller	41
5.1.1	MiaSituazioneContainerController	42
5.1.2	MovimentiContainerController	42
5.1.3	Comunicazione tra container e child controller	45
5.2	Posizionamento di oggetti in uno spazio cartesiano	46
5.2.1	Realizzazione	46
6	Sviluppi futuri	55

Capitolo 1

Introduzione al progetto

1.1 Descrizione dell'applicazione

1.2 Primi requisiti funzionali

Di seguito sono riportati i primi requisiti funzionali ricavati dal processo di analisi delle richieste del committente, ottenute durante le interviste iniziali del progetto. Tali requisiti definiscono le funzionalità e i servizi offerti dal sistema da realizzare.

Tabella 1.1. Primi requisiti funzionali

Requisito funzionale	Descrizione
Login ai servizi	Permettere all'utente l'accesso ai servizi bancari mediante credenziali
Visualizzazione riepilogo conti e carte	Fornire opportune viste di riepilogo dei prodotti posseduti da un utente (esempio: conti correnti, carte di credito, ecc...)
Visualizzazione storico saldi	Rappresentare tramite grafici dello storico saldi di un prodotto
Riepilogo lista movimenti	Recuperare e visualizzare la lista dei movimenti di un determinato prodotto
Filtraggio lista movimenti	Permettere il recupero dei movimenti in base un certo arco temporale
Disporre operazioni bancarie	Permettere l'esecuzione di dispositive bancarie (esempio: bonifici, giroconti, ecc...)
Interazione con i social network	Consentire la visualizzazione dei contenuti social messi a disposizione del cliente
Messaggistica	Permettere la lettura delle comunicazioni ricevute

1.3 Primi mockup, wireframe e prototipi

1.3.1 Mockup

La realizzazione di mockup grafici ha permesso di offrire al cliente una prima rappresentazione visiva dei requisiti funzionali iniziali. Di seguito sono riportate alcuni di questi mockup relativi allo stadio iniziale del progetto.

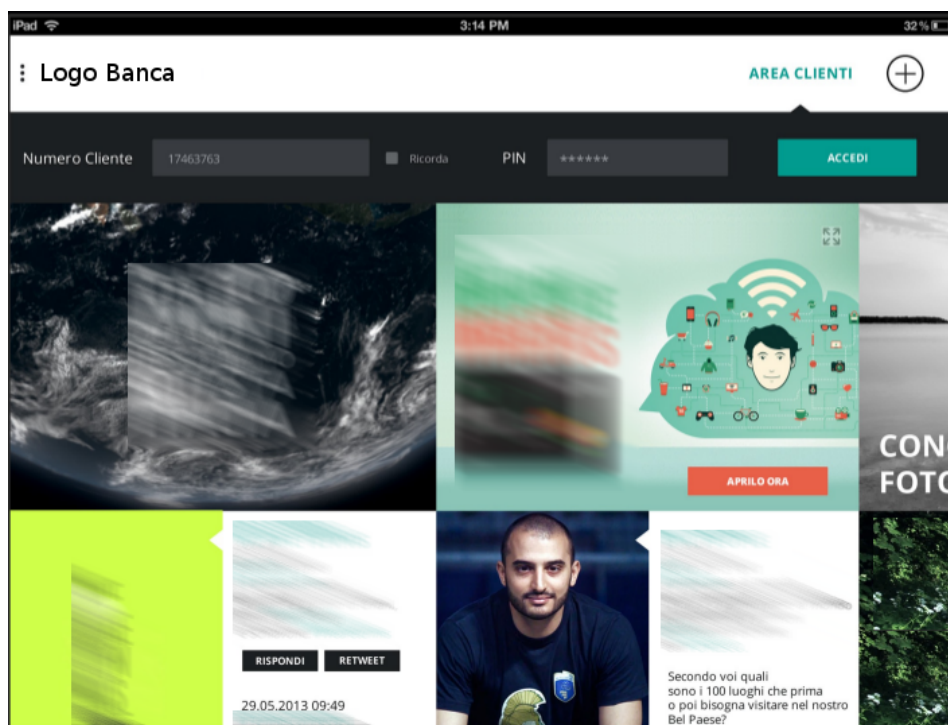


Figura 1.1. Home contenuti social e pannello login

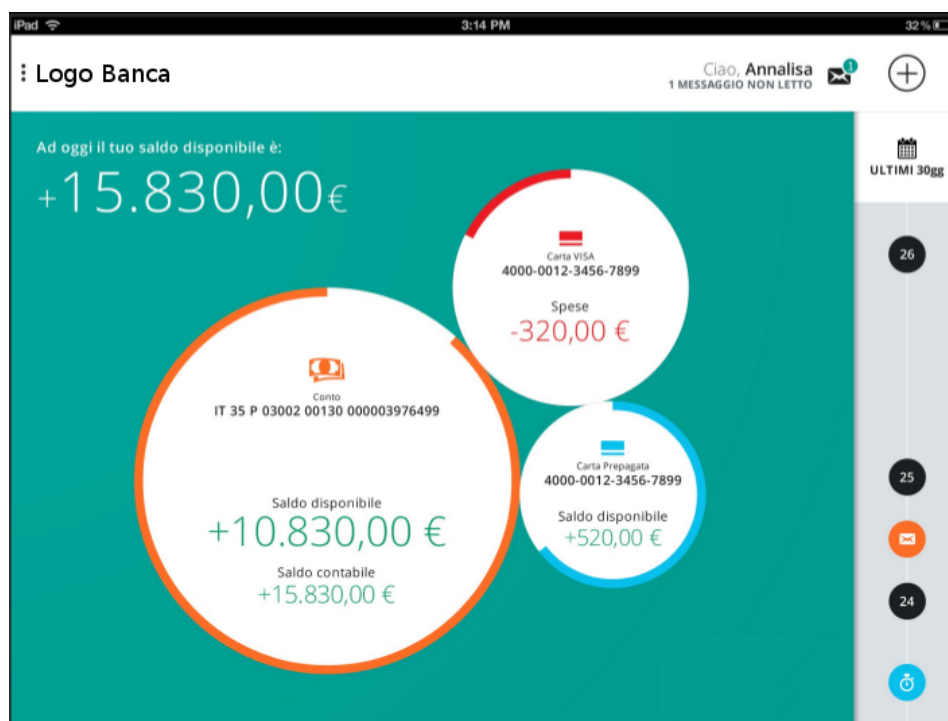


Figura 1.2. Riepilogo conti e carte

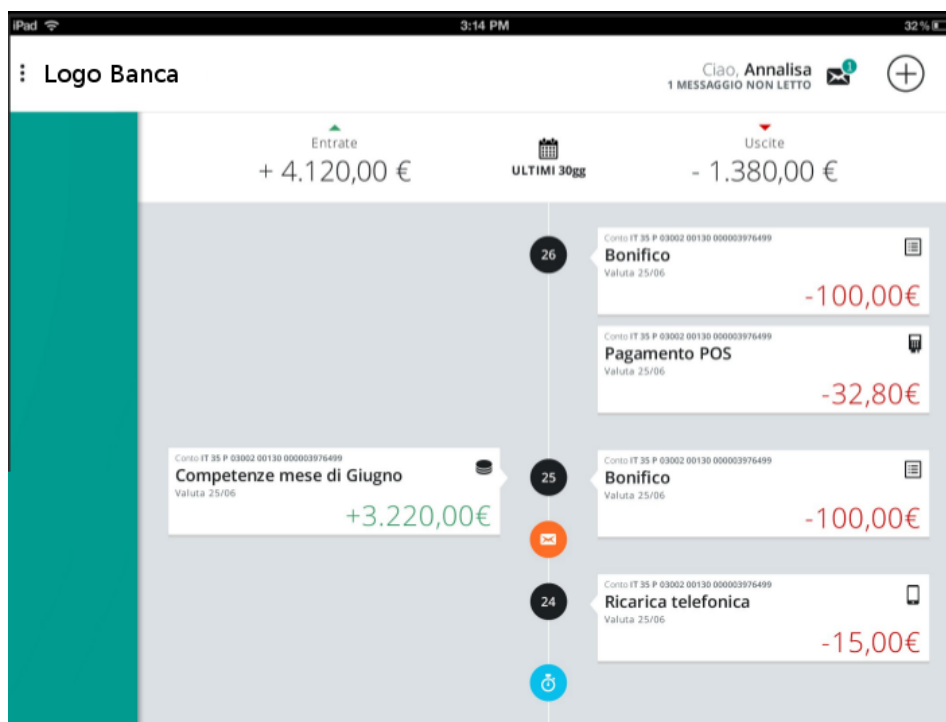


Figura 1.3. Timeline movimenti

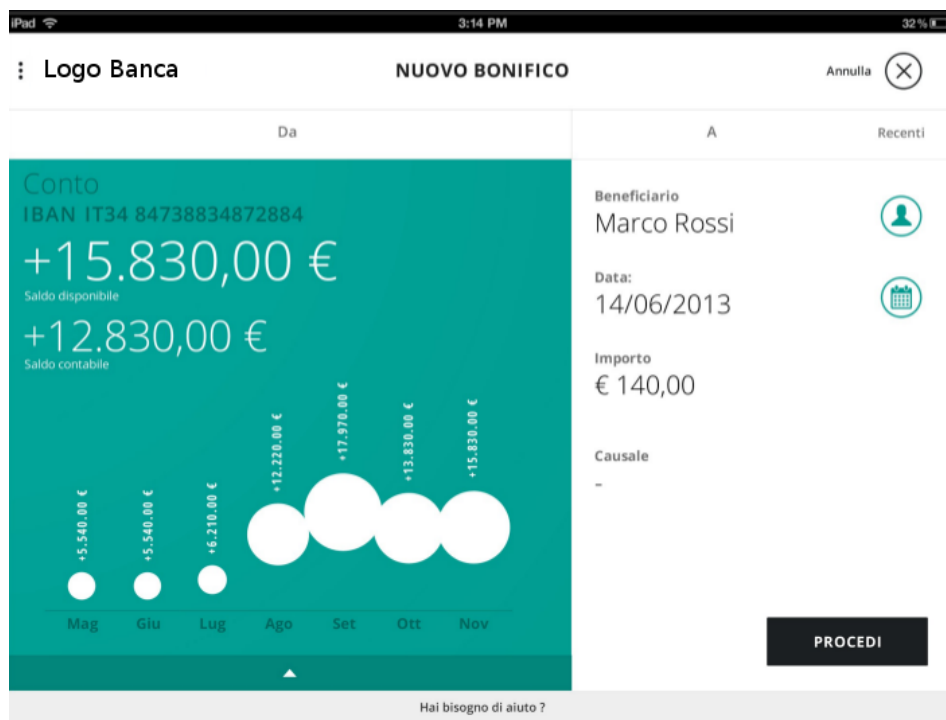


Figura 1.4. Operazione bonifico

1.3.2 Wireframe

Parallelamente ai mockup e durante tutto il ciclo di vita del software sono stati realizzati e raffinati i wireframe.

I wireframe forniscono una rappresentazione strutturale di un'applicazione software e permettono di individuare le dinamiche del progetto in termini di usabilità ed utilizzo pratico, i punti critici e quelli che richiedono uno sviluppo più accurato o miglioramenti.

Di seguito sono mostrate alcune delle immagini di uno dei primi wireframe realizzati:



Figura 1.5. Riepilogo conti e carte

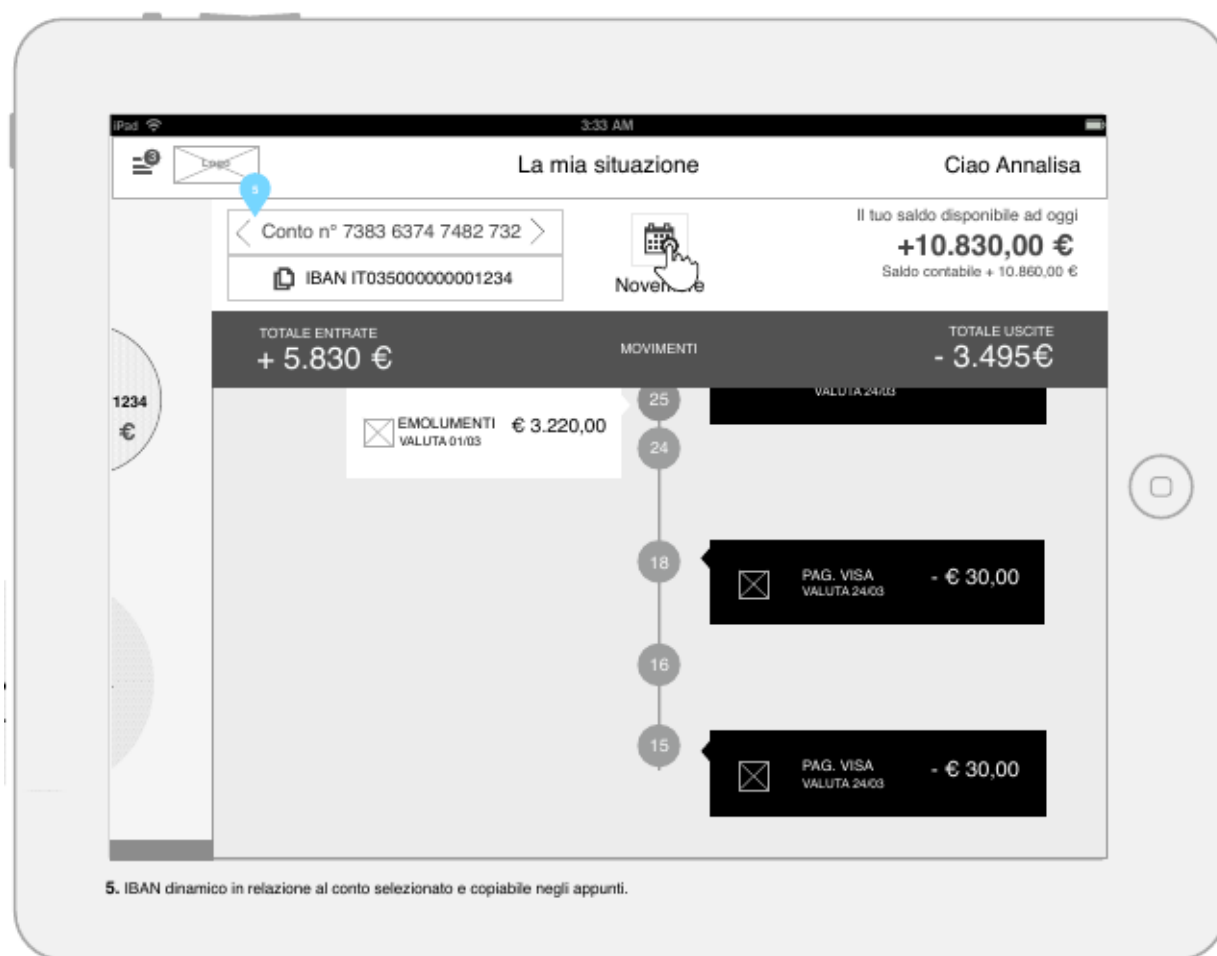


Figura 1.6. Timeline movimenti

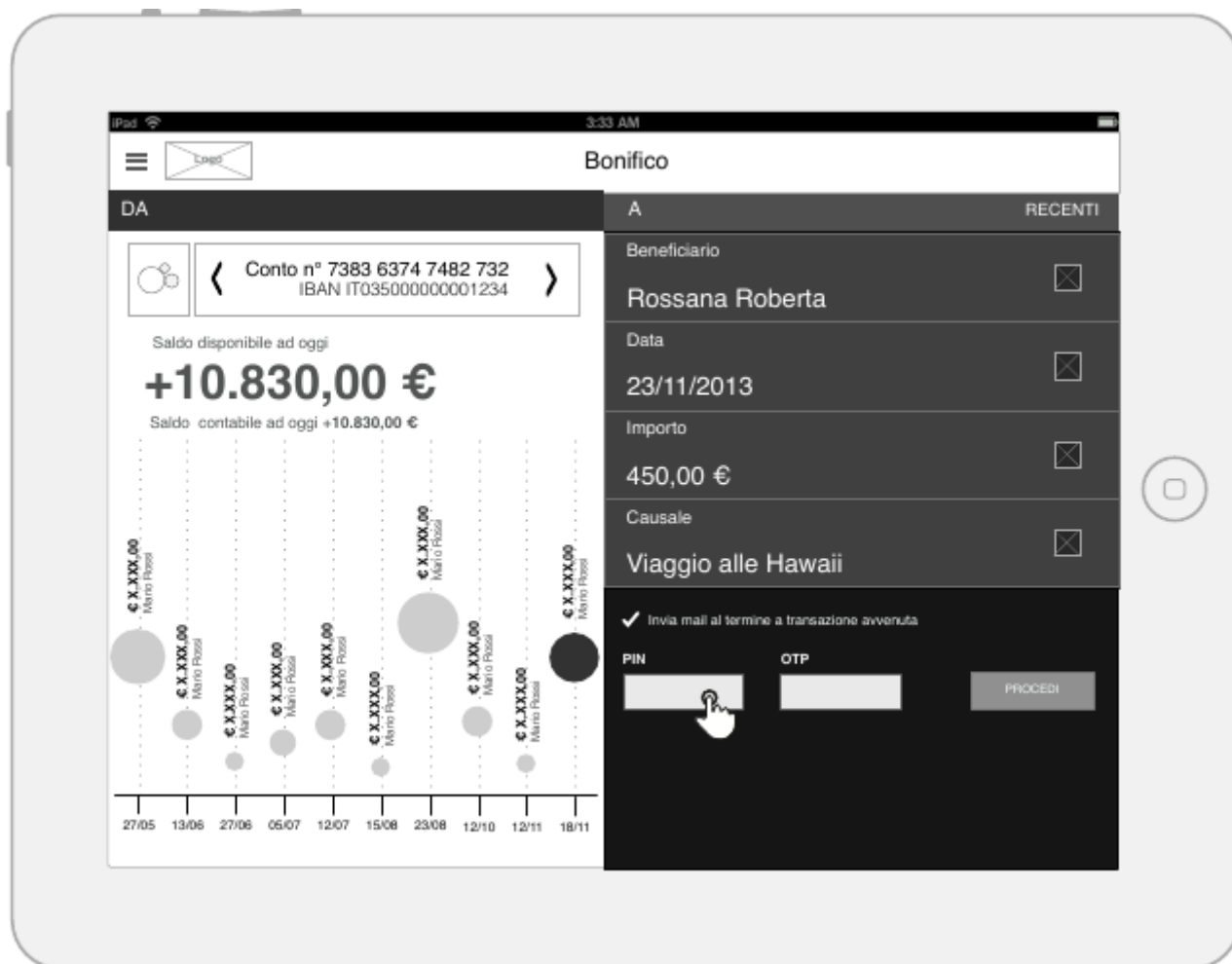


Figura 1.7. Dettaglio bonifico

1.3.3 Prototipi

Un'altro passo fondamentale durante la fase iniziale del progetto è stato la realizzazione di prototipi.

Un prototipo è un modello approssimato del sistema che si sta realizzando e che simula o esegue solo una parte delle funzioni del sistema finale. La prototipizzazione permette di:

- tenere il design centrato sull'utente
- sperimentare design alternativi
- ottenere feedback rapidi sul progetto
- trascurare dettagli secondari (come qualità del codice, efficienza, ecc...)
- valutare l'usabilità
- ridurre i rischi di un progetto permettendoci di mettere prima a fuoco alcune caratteristiche del sistema e capire se sono adeguate o meno

Durante le prime settimane del progetto sono stati quindi realizzati dei prototipi contenenti funzionalità *stub*¹ descritte dalla tabella 1.1. Tali prototipi sono stati successivamente messi a disposizione del cliente e testati su device Apple iPad, portando alla raccolta e valutazione dei primi feedback sull'utilizzo del software.

¹Funzionalità che simulano il comportamento del sistema restituendo valori accettabili in un ipotetico scenario reale.

Capitolo 2

Metodologie usate e raffinamenti successivi

2.1 Metodologia Agile

L'intero ciclo di vita del software è stato gestito adottando una metodologia *Agile*.

I metodi Agile sono tali da coinvolgere il più possibile il committente, dando quindi vita a un processo di tipo adattativo: cioè che si adatta alle esigenze del cliente, che possono cambiare durante lo sviluppo. L'Agile è un processo costituito da finestre di tempo limitate (2-4 settimane) chiamate iterazioni, le quali sono a loro volta scomposte nelle fasi di progettazione, di sviluppo e di test.

Il progetto è quindi suddiviso in singoli componenti indipendenti dalle funzionalità così da poterne analizzare e valutare i costi e i tempi. Ogni iterazione conterrà quindi tutto ciò che è indispensabile per rilasciare un piccolo incremento nelle funzionalità del software e sono tali da essere soggette a modifiche al fine di migliorare l'architettura finale del software.

2.2 Requisiti funzionali finali

Una progettazione iterativa e focalizzata sull'utente ha permesso quindi di ampliare e raffinare i requisiti fino ad ottenere un risultato sempre più vicino alle richieste e alle necessità del committente.

Per brevità di seguito sono riportati solo alcuni dei requisiti finali del progetto (che vanno ad aggiungersi o a integrare quelli della tabella 1.1), per comodità raccolti in categorie.

Tabella 2.1. Requisiti funzionali finali

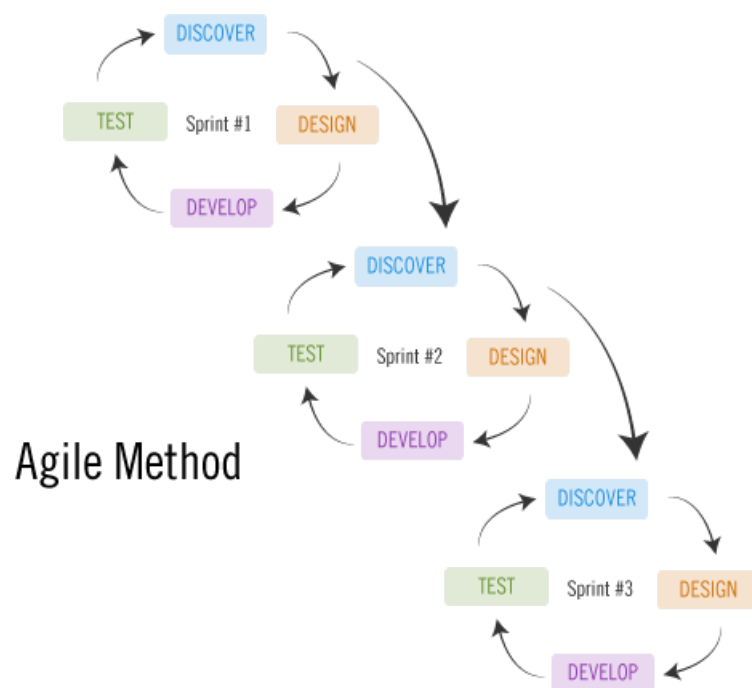


Figura 2.1. Iterazioni e fasi della metodologia Agile

Requisito funzionale	Descrizione
Accesso ai servizi	<ul style="list-style-type: none">• Permettere l'accesso ai servizi attraverso credenziali• Permette l'accesso veloce a un sottoinsieme di funzioni salvando le credenziali• Permettere recupero Codice Cliente

Riepilogo movimenti	<ul style="list-style-type: none">• Rappresentare i movimenti di un conto attraverso una timeline• Mostrare i movimenti di un conto in una lista• Visualizzare una scheda di dettaglio per un movimento selezionato
Filtro movimenti	Offrire la possibilità di filtrare i movimenti per data e per tipo (entrate, uscite, tutti)
Riepilogo conti e carte	<ul style="list-style-type: none">• Visualizzare i prodotti di un utente attraverso bubble contenenti informazioni di riepilogo• Visualizzare i prodotti di un utente attraverso una lista• Permettere la selezione di un determinato prodotto
Filtro conti e carte	Permettere la visualizzazione o meno di certi prodotti mediante un filtro
Storico saldi	Visualizzare in un grafico a bolle lo storico dei saldi di un prodotto per mesi o settimane
Disporre operazioni bancarie	<ul style="list-style-type: none">• Fornire un form per la compilazione di dispositive• Permettere la riproposizione di dispositive già effettuate

Riepilogo dispositive effettuate

- Visualizzare le ultime n dispositive in una lista ordinata
- Visualizzare le ultime n dispositive in un grafico a bolle
- Permettere il salvataggio delle dispositive effettuate in una lista di preferiti

Preferiti

Permettere la visualizzazione e la modifica di una lista di dispositive scelte come preferite

Rubrica

Permettere l'accesso in lettura e scrittura alla rubrica del dispositivo per poter salvare o recuperare numeri telefonici

Help e gestore

- Offrire una sezione interna al software in cui l'utente può richiedere appuntamenti con un consulente
- Mettere a disposizione una sezione di help e numeri utili

Integrazione social network

- Visualizzare i contenuti dei canali social offerti dal committente
- Offrire la possibilità di condividere i contenuti sui diversi social network
- Integrare un player video per la visualizzazione di filmati relativi alle attività del committente

Browsing interno	Permettere la navigazione tra contenuti web attraverso un browser interno all'applicazione
Notifiche Push	Permettere la ricezione di notifiche push
Messaggistica	<ul style="list-style-type: none">• Visualizzare i messaggi ricevuti dai servizi• Permettere la gestione dei messaggi (esempio: cancella, segna come letto, ecc...)

Sono inoltre elencati alcuni dei requisiti non funzionali:

Tabella 2.3. Requisiti non funzionali

Requisito non funzionale	Descrizione
Comunicazione sicura	<ul style="list-style-type: none">• Garantire la comunicazione sicura tra client e server• Garantire che i dati salvati all'interno del device siano protetti
Ottimizzazione flusso dati	Limitare il più possibile il consumo dei dati
Efficienza	Garantire uso efficiente delle risorse offerte dal device (cpu, batteria, ecc. . .)

2.3 User Experience e usabilità

Oltre al consolidamento dei requisiti, le fasi di analisi e test (racchiuse in ogni iterazione della metodologia Agile) hanno permesso una graduale evoluzione del progetto, partito da uno stadio prototipale, anche sotto il punto di vista dell'usabilità e della user experience.

I test e i collaudi sono stati eseguiti presentando il progetto su dispositivi Apple iPad a personale selezionato (per policy interne i collaudi sono avvenuti con personale interno all'azienda cliente) e sono stati discussi, raccolti e valutati i feedback ottenuti durante l'utilizzo del software. Questo approccio ha consentito l'immediata valutazione della qualità del software prodotto e di concentrarsi immediatamente sull'iterazione successiva per migliorarlo.

2.3.1 iOS Human Interface Guidelines

Le *GUI* (graphic user interface) sono state realizzate considerando le linee guida fornite da Apple nelle *Human Interface Guidelines*. Di seguito sono elencate alcune delle linee chiave adottate:

- Integrità estetica: rappresenta quanto l'apparenza di un'interfaccia è coerente con le azioni e le funzionalità che offre
- Consistenza: permettere all'utente di trasferire le proprie abilità e conoscenze da un'interfaccia di un'applicazione all'altra

- Feedback: fornire un feedback all'utente permette di avere una risposta immediata di un'azione intrapresa sull'interfaccia

Oltre ai componenti standard offerti dall'*SDK* iOS sono stati realizzati dei componenti grafici custom che hanno migliorato l'esperienza d'uso rispettando al tempo stesso le linee guida e il know how medio posseduto dalle persone che utilizzano dispositivi mobili.

2.3.2 Esempi di GUI e raffinamenti successivi

Colori

Inizialmente si era scelto di utilizzare un tema di colori focalizzato sullo stesso colore del brand del committente e degli altri prodotti precedentemente realizzati (esempio: sito web, app smartphone, ecc. . .).

I test con gli utenti hanno però evidenziato che temi di colori troppo scuri possono risultare pesanti alla vista e percepiti difficili da leggere. Si è scelto quindi di utilizzare un tema di colori chiaro, con gli elementi più importanti in contrasto di colore in modo tale da indurre un focus immediato e guidare l'utente durante le operazioni.

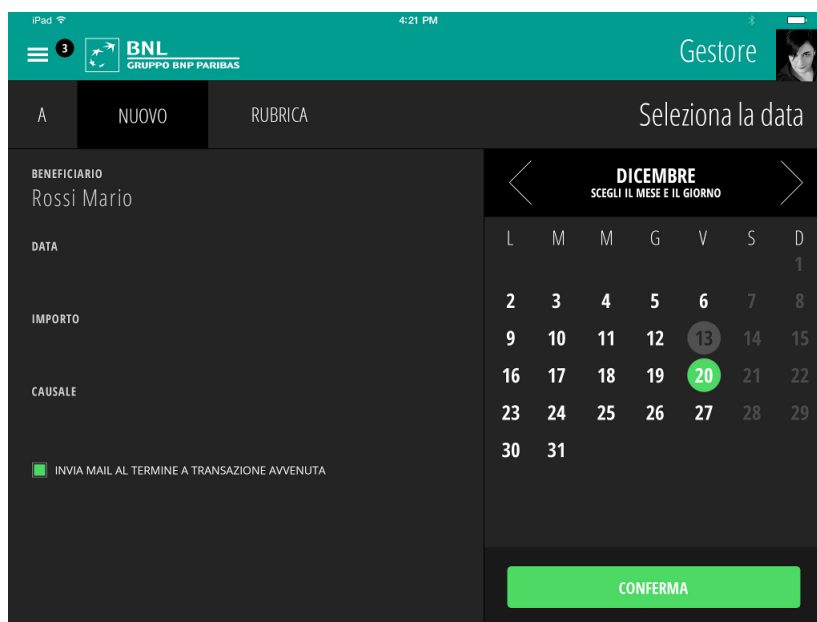


Figura 2.2. Tema iniziale nero per la sezione bonifico

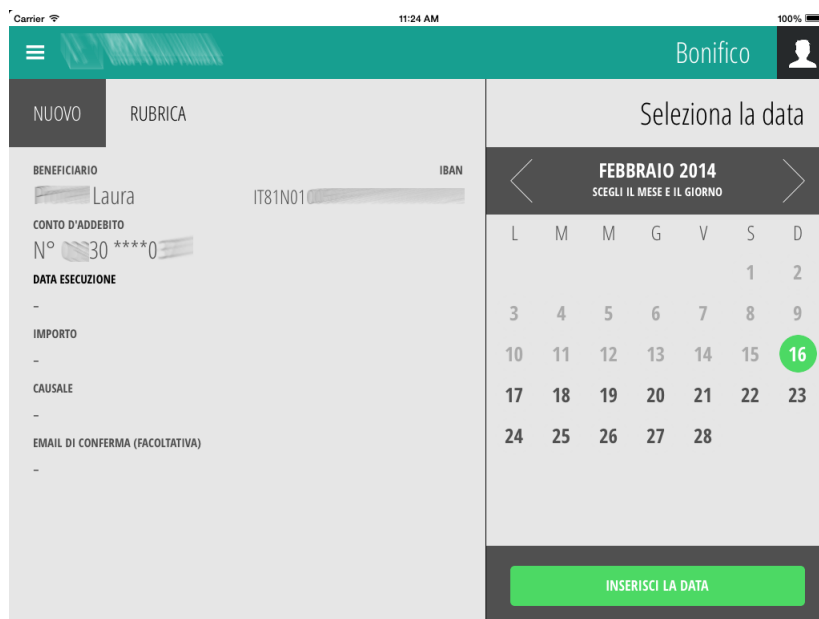


Figura 2.3. Tema chiaro per il bonifico nella sua versione finale, si noti il pulsante *conferma* in risalto rispetto altri elementi della GUI

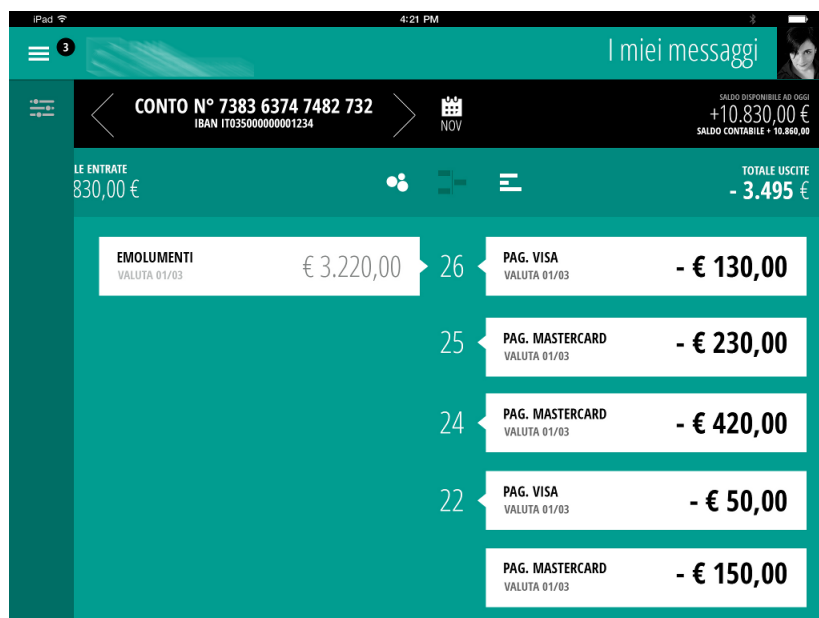


Figura 2.4. Tema verde per il background della timeline

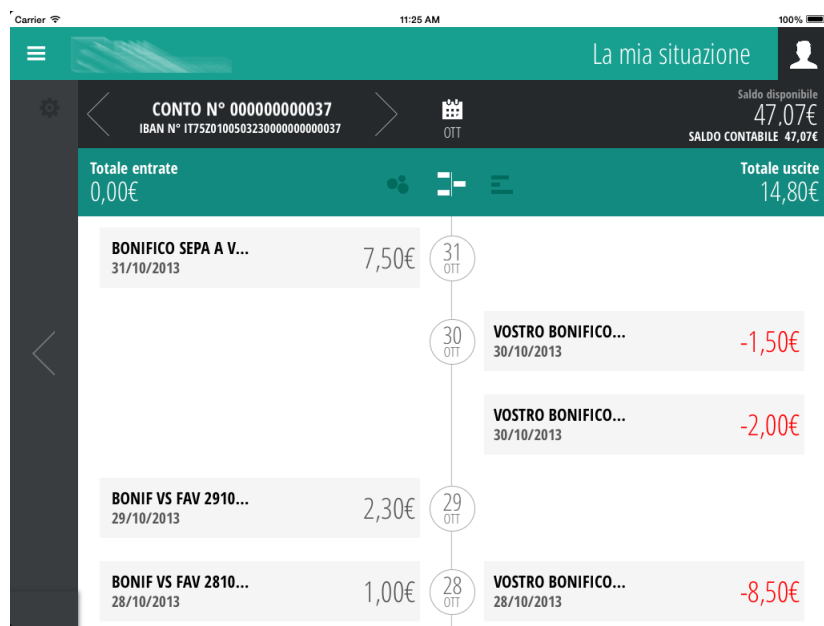


Figura 2.5. Tema chiaro per il background della timeline e utilizzo del colore rosso e grigio per dare risalto ai saldi negativi e positivi dei movimenti

Suggerimenti

Dai test effettuati si è notata la necessita di aiutare l'utente attraverso suggerimenti (*hint*) visivi posti in particolari punti delle interfacce. L'utilizzo moderato di suggerimenti ha aumentato e migliorato la *discoverability*.

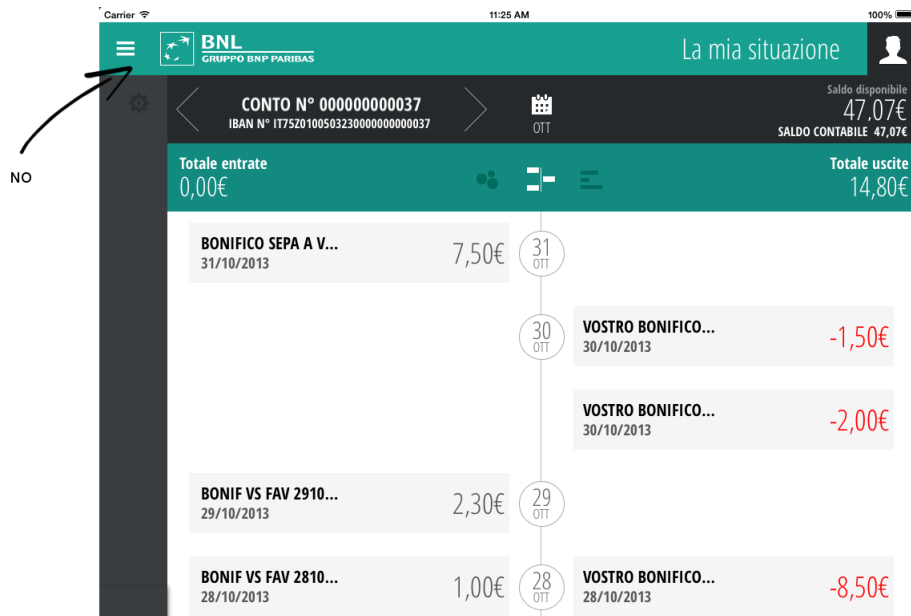


Figura 2.6. L'utente premeva il pulsante *menu* per cercare di tornare alla schemata dei conti

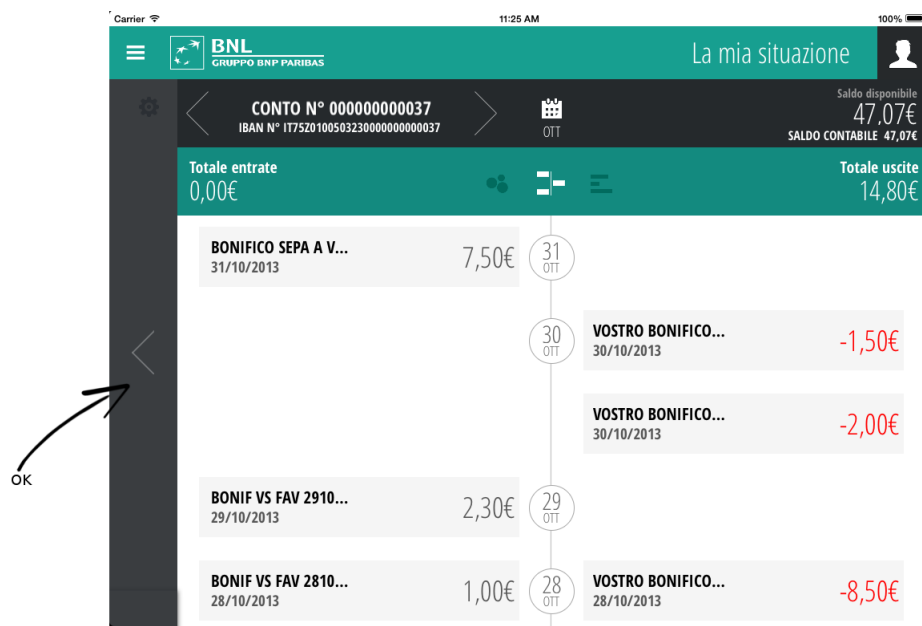


Figura 2.7. Con una freccia che rappresentasse il significato di *indietro* si è messo in risalto la relativa azione

Feedback

Per indicare che, in risposta ad una azione dell'utente, è avvenuto un evento nel sistema si è scelto di utilizzare feedback visivi e diversificati in base al tipo di azione intrapresa. Di seguito sono riportati alcuni dei feedback utilizzati all'interno dell'applicazione:

- componenti che indicano il caricamento durante task di lunga durata (esempio durante le chiamate http ai servizi, figura 2.8)
- l'inserimento di animazioni per passare da un'interfaccia ad un'altra dopo l'azione eseguita dall'utente (figura 2.9)
- elementi grafici che cambiano il loro stato in risposta a eventi particolari (figura 2.10)

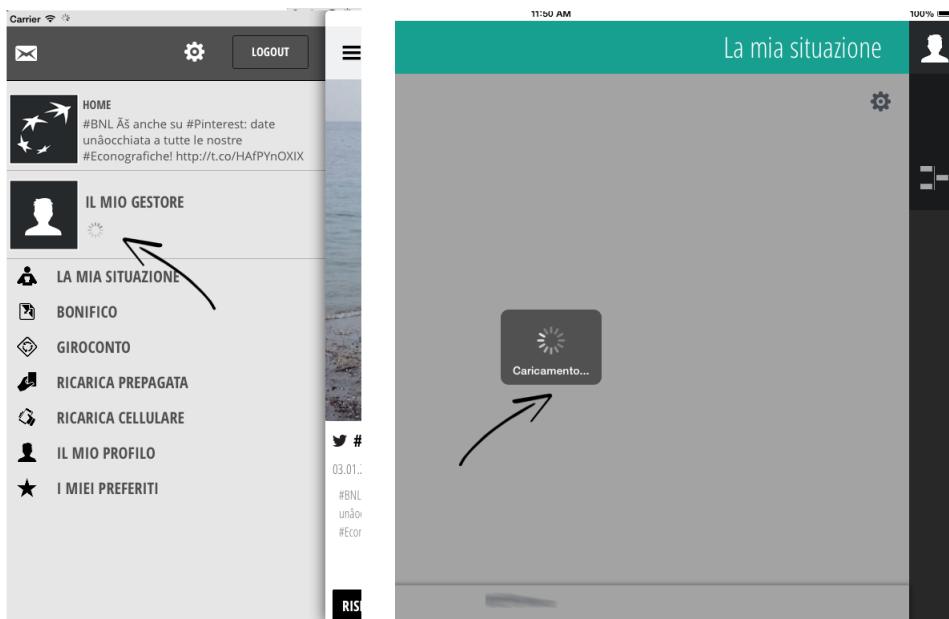


Figura 2.8. Esempi di loading view durante la chiamata ai servizi

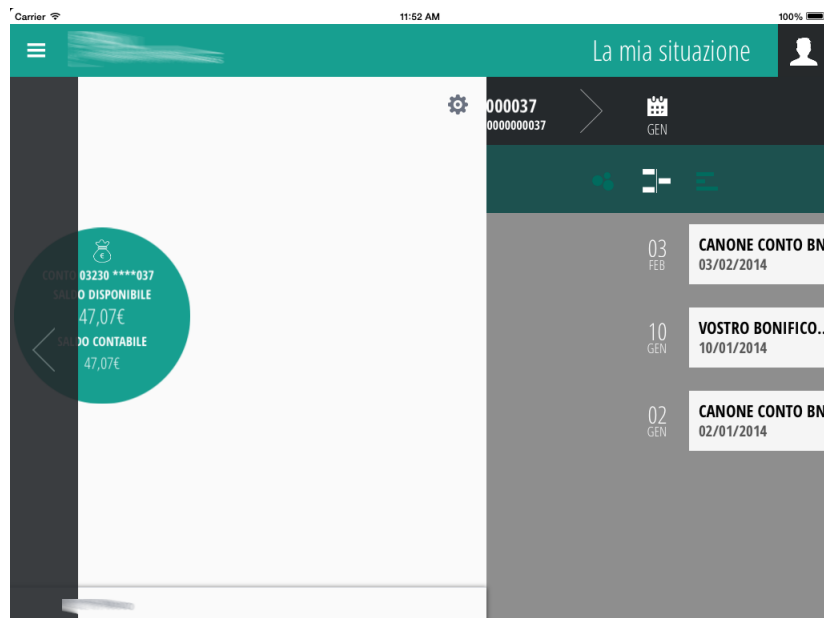


Figura 2.9. Fotogramma preso durante un'animazione rappresentante il cambio di interfaccia e del relativo contesto (colori, immagini, contenuti)

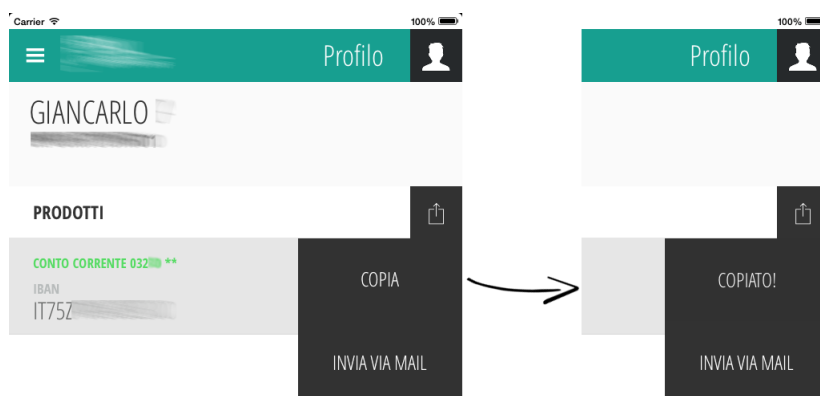


Figura 2.10. Dopo l'azione di copia il relativo pulsante cambia per qualche frazione di secondo il suo contenuto in *copiato!*

Componenti personalizzati

L'utilizzo di componenti grafici personalizzati ha permesso di ottenere interfacce grafiche più ricche di funzioni e al tempo stesso intuitive e facili da usare. La figura 5.1 rappresenta un componente grafico che ha la funzione di poter selezionare attraverso un tap sui pulsanti o attraverso la gesture swipe un elemento in una lista di oggetti (in questo caso di conti); tale componente è stato realizzato come libreria esterna, e quindi riutilizzabile in diversi contesti della stessa applicazione.

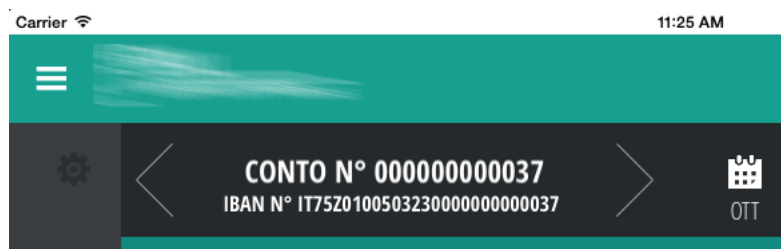


Figura 2.11. Il componente custom *selettore conti*

Un'altro esempio di componente personalizzato sono i tasti di azione rapida accessibili effettuando la gesture *swipe* sulle righe della *Message box* interna all'applicazione. In questo caso si è ricreato lo stesso design e si è mantenuta la stessa interazione che presenta l'applicazione *Mail* nativa di iOS 7, permettendo così all'utente di riutilizzare le stesse abilità acquisite nell'utilizzo di un software originale Apple.

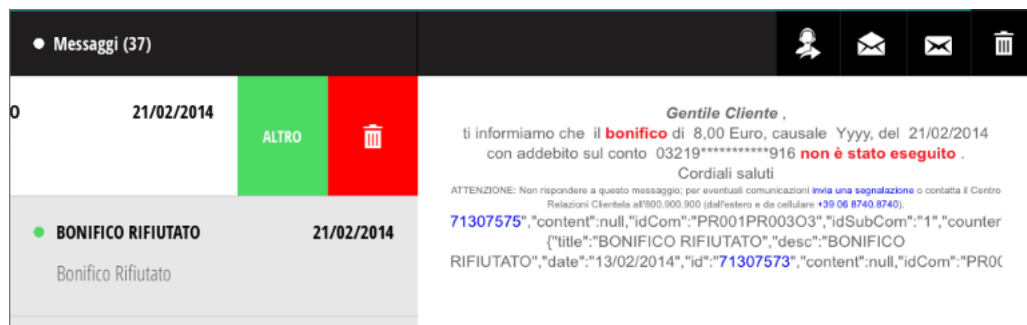


Figura 2.12. Dettaglio delle funzionalità della *Message box*

Help

Per permettere un facile e rapido utilizzo dell'applicazione a tutti i possibili utenti, che spaziano dall'utilizzatore abituale di device e applicazioni mobili a quelli che si avvicinano per la prima volta alle nuove tecnologie, si è deciso di utilizzare al primo avvio dell'app delle schermate di aiuto (figura 2.13) che descrivessero le principali e le meno intuitive funzionalità del software.

Le fasi di test hanno evidenziato l'effettiva necessità di tale scelta, che hanno portato al miglioramento della discoverability soprattutto nei primi utilizzi dell'app da parte di un utente.

Prestazioni

Per rendere più gradevole la user experience si è deciso di eseguire il *caching*¹ di alcuni dei dati scaricati dalla rete (come ad esempio la lista dei prodotti, o dati relativi al gestore, ecc...). Questa scelta ha quindi permesso di ridurre i tempi di accesso ad alcuni servizi migliorando le prestazioni del software in generale e l'interazione dell'utente con lo stesso.

¹Eseguire il salvataggio in ram o in memoria secondaria dei dati di frequente utilizzo, in modo tale da renderli disponibili per accessi futuri.

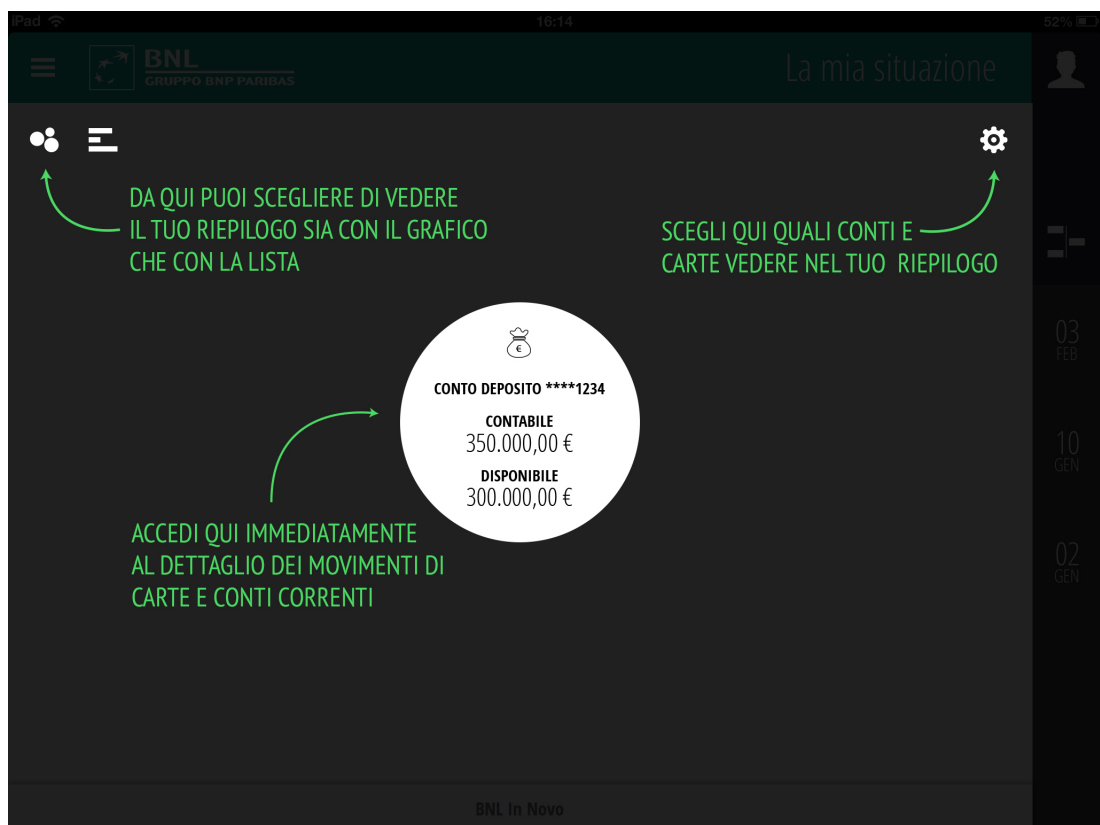


Figura 2.13. Schermata di help per descrivere le funzionalità dell'interfaccia di riepilogo di conti e carte

Capitolo 3

Casi d'uso

I casi d'uso permettono di modellare il comportamento del sistema e descrivere i requisiti funzionali in linguaggio naturale. Grazie a questi modelli è quindi possibile rappresentare il comportamento esterno del sistema, visto come una black-box, dal punto di vista di un attore.

Oltre a una descrizione testuale (gli *Scenari*) i casi d'uso possono esser rappresentati mediante diagrammi. In tali diagrammi troviamo le seguenti entità:

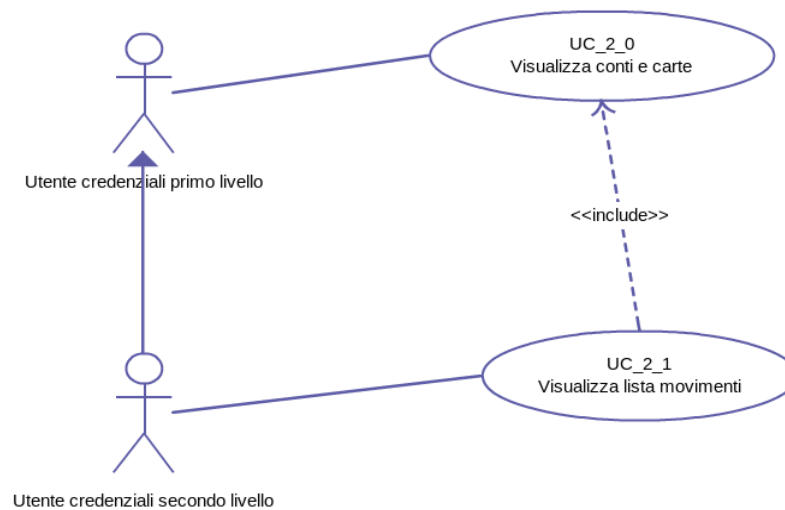
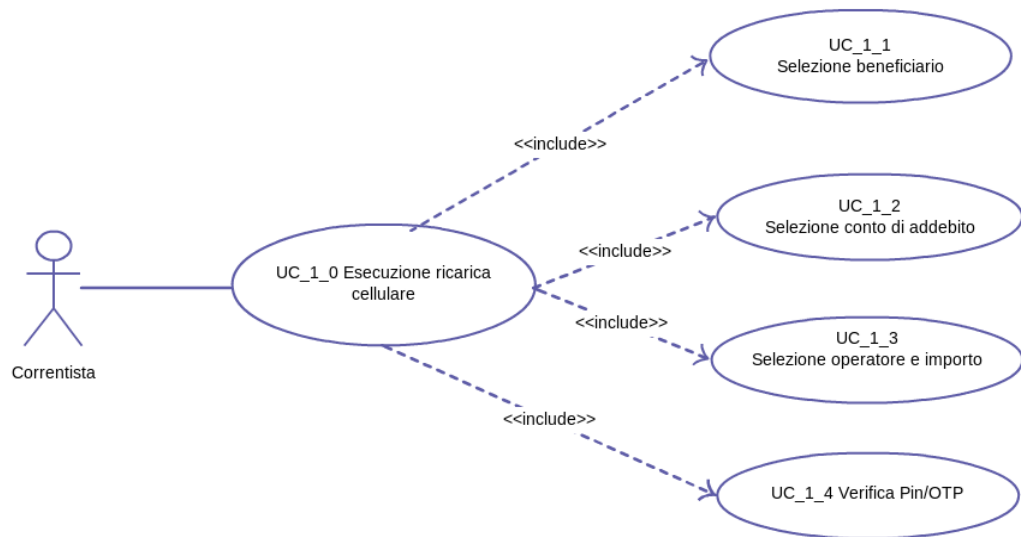
- gli attori: essi rappresentano un soggetto o un'entità esterna che interagisce col sistema
- i casi d'uso: rappresentano un particolare scenario di interazione tra attore e sistema. Sono rappresentati graficamente da ellissi
- le associazioni: rappresentate da una linea mettono in comunicazione un attore con i caso d'uso

Nei prossimi paragrafi saranno mostrati alcuni dei casi d'uso modellati durante la fase di analisi.

3.1 Diagrammi dei casi d'uso

3.1.1 Caso d'uso: Ricarica telefonica

3.1.2 Caso d'uso: Ricarica telefonica



3.2 Specifica dei casi d'uso

3.2.1 Caso d'uso: Ricarica telefonica

Le tabelle..... mostrano i casi d'uso relativi all'operazione di ricarica telefonica, nel caso in cui l'utente decida di ricaricare un numero di telefono associato ad un contatto presente nella rubrica del dispositivo.

Caso d'uso UC_1_0	Ricarica telefonica di un numero salvato in rubrica
Attore	Correntista bancario
Precondizioni	L'utente è loggato al sistema e ha iniziato una sessione inserendo un codice OTP
Postcondizioni di successo	Il sistema ha registrato l'operazione
Postcondizioni di fallimento	Il sistema non ha registrato l'operazione
Scenario	
	<ol style="list-style-type: none"> 1. L'utente seleziona l'operazione di ricarica telefonica 2. Include UC_1_1 <i>Visualizza lista beneficiari</i> 3. Include UC_1_2 <i>Visualizza lista conti di addebito</i> 4. Include UC_1_3 <i>Visualizza lista operatori</i> 5. Include UC_1_4 <i>Verifica pin e OTP</i> 6. Il sistema mostra lo scontrino dell'operazione
Scenari alternativi	
	<ul style="list-style-type: none"> • 1 L'utente annulla operazione • Il caso d'uso termina

Caso d'uso UC_1_1	Selezione beneficiario
Attore	Correntista bancario
Precondizioni	L'utente ha iniziato una nuova ricarica telefonica
Postcondizioni di successo	Il sistema ha convalidato la scelta del beneficiario
Postcondizioni di fallimento	Il sistema non ha cambiato stato
Scenario	
	<ol style="list-style-type: none"> 1. Il sistema visualizza la lista dei beneficiari 2. L'utente seleziona un beneficiario 3. Il sistema aggiorna l'interfaccia con la selezione effettuata
Scenari alternativi	
	<ol style="list-style-type: none"> 2. L'utente seleziona la voce <i>Beneficiario</i> 3. Il caso d'uso riprende dal punto 1
	<ul style="list-style-type: none"> • L'utente annulla l'operazione • Il caso d'uso termina
Scenari di errore	
	<ol style="list-style-type: none"> 3. Il sistema non valida le credenziali 4. Il sistema comunica il tipo di errore 5. Il caso d'uso riprende dal punto 1

Caso d'uso UC_1_2	Selezione conto di addebito
Attore	Correntista bancario
Precondizioni	L'utente ha iniziato una nuova ricarica telefonica
Postcondizioni di successo	Il sistema ha convalidato la scelta del conto

Postcondizioni di fallimento	Il sistema non ha cambiato stato
Scenario	
	<ol style="list-style-type: none"> 1. Il sistema visualizza la lista dei conti disponibili 2. L'utente seleziona un conto 3. Il sistema aggiorna l'interfaccia con la selezione effettuata
Scenari alternativi	
	<ol style="list-style-type: none"> 2. L'utente seleziona la voce <i>Conto di addebito</i> 3. Il caso d'uso riprende dal punto 1
	<ul style="list-style-type: none"> • L'utente annulla l'operazione • Il caso d'uso termina

Caso d'uso UC_1_3	Selezione operatore
Attore	Correntista bancario
Precondizioni	L'utente ha iniziato una nuova ricarica telefonica
Postcondizioni di successo	Il sistema ha convalidato la scelta dell'operatore
Postcondizioni di fallimento	Il sistema non ha cambiato stato
Scenario	

	<ol style="list-style-type: none"> 1. Il sistema visualizza la lista delle compagnie telefoniche 2. L'utente seleziona una compagnia 3. Il sistema mostra i tagli di ricarica disponibili 4. L'utente seleziona un taglio di ricarica 5. Il sistema aggiorna l'interfaccia con la selezione effettuata
Scenari alternativi	
	<ol style="list-style-type: none"> 2. L'utente seleziona la voce <i>Importo</i> 3. Il caso d'uso riprende dal punto 3
	<ul style="list-style-type: none"> • L'utente annulla l'operazione • Il caso d'uso termina
Scenari di errore	
	<ol style="list-style-type: none"> 2. L'utente seleziona una compagnia telefonica errata 3. Il sistema mostra messaggio di errore 4. Il caso d'uso riprende dal punto 1

Caso d'uso UC_1_4	Verifica Pin e OTP
Attore	Correntista bancario
Precondizioni	L'utente ha iniziato una nuova ricarica telefonica e ha immesso tutti i dati richiesti

Postcondizioni di successo	Il sistema ha convalidato l'operazione e ha aggiornato il suo stato
Postcondizioni di fallimento	Il sistema non ha convalidato l'operazione e ha mostrato un messaggio di errore
Scenario	
	<ol style="list-style-type: none"> 1. Il sistema mostra il riepilogo dell'operazione e richiede conferma 2. L'utente conferma i dati 3. Il sistema richiede Pin e OTP 4. L'utente immette i dati richiesti e conferma 5. Il sistema verifica le credenziali 6. Il sistema mostra lo scontrino dell'operazione effettuata
Scenari alternativi	
	<ul style="list-style-type: none"> • L'utente annulla l'operazione • Il caso d'uso termina
	<ol style="list-style-type: none"> 5. Il sistema non valida le credenziali 6. Il sistema visualizza tipo di errore 7. Il caso d'uso riprende dal punto 3

3.2.2 Caso d'uso: Visualizza lista conti e carte e Visualizza lista movimenti

Caso d'uso	UC_2_0 Visualizza lista conti e carte
Attore	Correntista bancario

Precondizioni	L'utente ha effettuato un login di primo o secondo livello
Postcondizioni di successo	Il sistema rimane nel suo stato o aggiorna la lista dei conti in memoria
Postcondizioni di fallimento	Il sistema rimane nel suo stato
Scenario	
	<ol style="list-style-type: none"> 1. L'utente seleziona l'operazione di visualizzazione lista conti e carte 2. Il sistema verifica che l'elenco dei prodotti non è vuoto 3. Il sistema mostra la lista dei prodotti
Scenario alternativo	
	<ol style="list-style-type: none"> 2. Il sistema ha verificato che la lista dei prodotti è vuota 3. Il sistema notifica l'evento con un opportuno messaggio 4. Il caso d'uso termina
Scenario di errore	
	<ol style="list-style-type: none"> 2. Il sistema riceve un errore 3. Il sistema notifica l'evento con un messaggio di errore e la possibilità di riprovare l'azione 4. L'utente seleziona <i>riprova</i> 5. Il caso d'uso riprende dal punto 2

Caso d'uso	UC_2_1 Visualizza lista movimenti
-------------------	-----------------------------------

Attore	Correntista bancario
Precondizioni	L'utente ha effettuato un login di secondo livello
Postcondizioni di successo	Il sistema rimane nel suo stato
Postcondizioni di fallimento	Il sistema rimane nel suo stato
Scenario	
	<ol style="list-style-type: none"> 1. Include UC_2_0 <i>Visualizza lista conti e carte</i> 2. L'utente seleziona un conto o una carta 3. Il sistema verifica che la lista dei movimenti non è vuoto 4. Il sistema mostra la lista dei prodotti
Scenario alternativo	
	<ol style="list-style-type: none"> 2. Il sistema ha verificato che la lista dei movimenti è vuota 3. Il sistema notifica l'evento con un opportuno messaggio 4. Extend <i>Filtra movimenti per data</i>
Scenario di errore	
	<ol style="list-style-type: none"> 2. Il sistema riceve un errore 3. Il sistema notifica l'evento con un messaggio di errore e la possibilità di riprovare l'azione 4. L'utente seleziona <i>riprova</i> 5. Il caso d'uso riprende dal punto 3

Capitolo 4

Architettura software

4.1 Architettura generale

Il progetto è stato realizzato seguendo un'architettura software client-server (figura 4.1). In questa architettura i componenti principali sono: il client rappresentato dall'applicazione iPad oggetto di questo documento, e il lato server scomposto in *middleware* e i sistemi messi a disposizione dal cliente (l'istituto bancario).

Il middleware opera da intermediario tra i servizi bancari e l'applicazione. Tale componente ha il compito di ricevere le richieste dal client, di sottomettere a sua volta queste richieste ai servizi lato banca e di ritornare i dati al client opportunamente formattati.

Le richieste verso il middleware (ospitante un webserver *Apache*) vengono eseguite seguendo un'architettura *RESTFUL* che prevede richieste *HTTP* (POST, GET, PUT, DELETE). Le risposte fornite dal middleware verso il client sono costruite in formato *JSON*.

4.2 Architettura applicazione

L'applicazione è stata progettata usando il design pattern *MVC* (Model-View-Controller, figura 4.2). In tale pattern ogni oggetto dell'applicazione assolve uno dei seguenti ruoli:

- Model: gli oggetti di tipo model hanno il compito di incapsulare i dati specifici a un'applicazione e di definire le logiche per modificare e processare tali dati
- View: sono gli oggetti dell'applicazione visibili all'utente. Tali oggetti hanno il compito di visualizzare i dati dell'applicazione e di permetterne l'interazione (esempio la modifica)

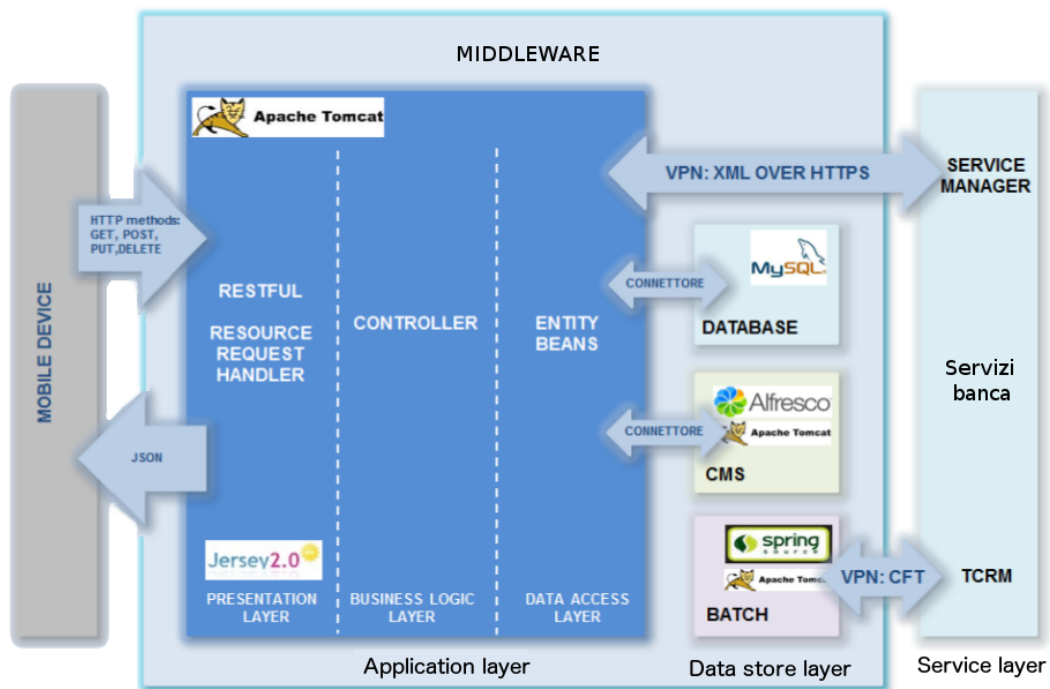


Figura 4.1. Architettura client-server

- **Controller**: è un oggetto che opera da intermediario tra la view e uno o più model dell'applicazione. Tali oggetti hanno quindi la funzione di comunicare i cambiamenti tra le view e i model, e viceversa.

L'utilizzo del pattern MVC permette una migliore estensione del codice, di costruire componenti riusabili e migliorare la definizione delle interfacce.

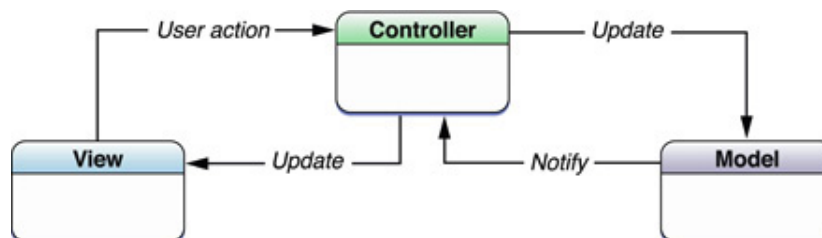


Figura 4.2. Schema del pattern MVC

Nei prossimi paragrafi verranno descritte le classi principali del progetto.

4.3 Diagramma delle classi

4.3.1 Connessione ai servizi

La parte relativa alla gestione delle chiamate ai servizi è stata realizzata implementando delle classi *singleton*. Tali classi estendono la libreria di terze parti **AFNetworking** che opera da wrapper sulle tecnologie di comunicazione messe a disposizione dall'SDK iOS (Foundation URL Loading System).

Utilizzare una libreria come AFNetworking ha permesso di ridurre i tempi di sviluppo e di sfruttare le potenzialità offerte dalle sue API come: la serializzazione e deserializzazione dei dati, la possibilità di effettuare richieste HTTP asincrone, una gestione efficiente delle code, la gestione di sessioni, ecc. . . .

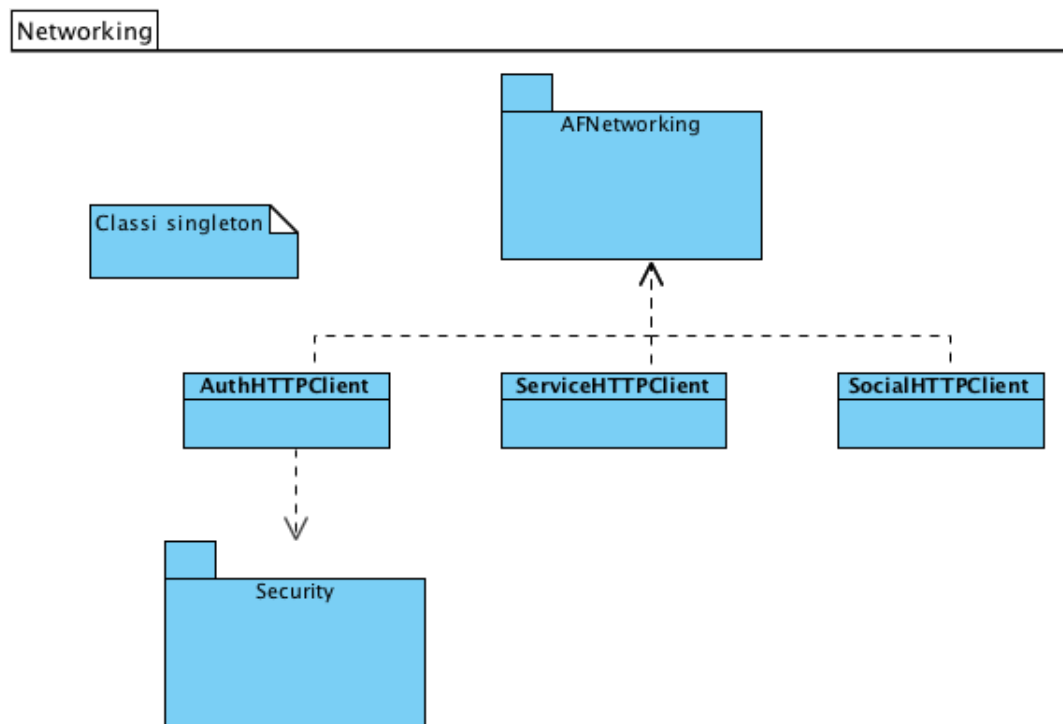


Figura 4.3. Diagramma delle classi per le connessioni di rete

4.3.2 Caching dei dati

Per migliorare le prestazioni e limitare il traffico di rete generato durante l'utilizzo dell'applicazione è stato implementato un meccanismo di caching dei dati che consente di salvare in memoria primaria i dati necessari all'applicazione durante una sessione di utilizzo.

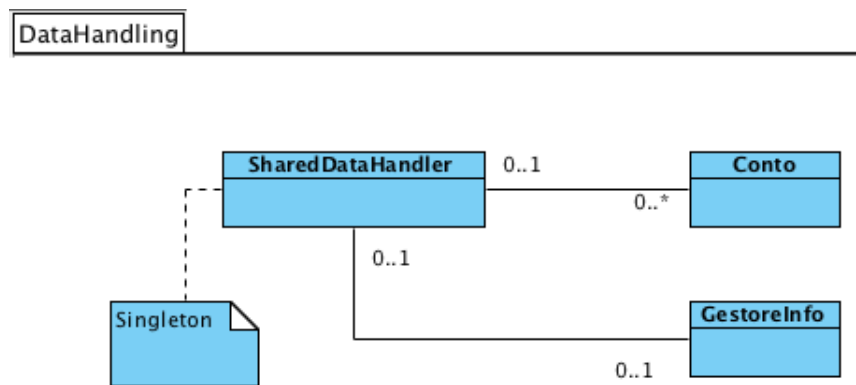


Figura 4.4. Diagramma delle classi per il caching dei dati

4.3.3 Autenticazione e sicurezza dei dati

L'applicazione prevede un sistema di autenticazione ai servizi basata su livelli. In particolare sono previsti tre livelli di autenticazione:

- autenticazione di livello zero: è il livello associato a un utente che non ha ancora effettuato il login
- autenticazione di primo livello: è il livello associato ad un utente che ha precedentemente effettuato il login, ha scelto di ricordare le credenziali e che ha momentaneamente sospeso l'utilizzo dell'applicazione. Questo livello permette all'utente di accedere a un gruppo ristretto di funzionalità nel momento in cui riprenderà l'utilizzo dell'applicazione
- autenticazione di secondo livello: è il livello associato ad un utente che ha precedentemente effettuato il login e che permette l'accesso a tutte le funzionalità previste dall'applicazione. Questo livello permane fino a quando l'utente non richiede esplicitamente il logout o sospende l'applicazione

L'autenticazione ai servizi è a carico delle classi descritte nel paragrafo 4.3.1. Questa operazione restituisce in output i dati confidenziali dell'utente che saranno gestiti dalle classi adibite alla sicurezza.

A tal fine è stato implementato un oggetto singleton, il **SecurityManager**, che ha la responsabilità di stabilire il livello di autenticazione in risposta agli eventi che occorrono all'interno dell'applicazione e di gestire il salvataggio e il recupero dei dati confidenziali dell'utente (come le informazioni sul token, sul numero utente, ecc...). Tale classe utilizza la libreria di terze parti **JNKeychain** che estende le funzioni messe a disposizione dell'SDK iOS, in particolare quelle relative al *Keychain*.

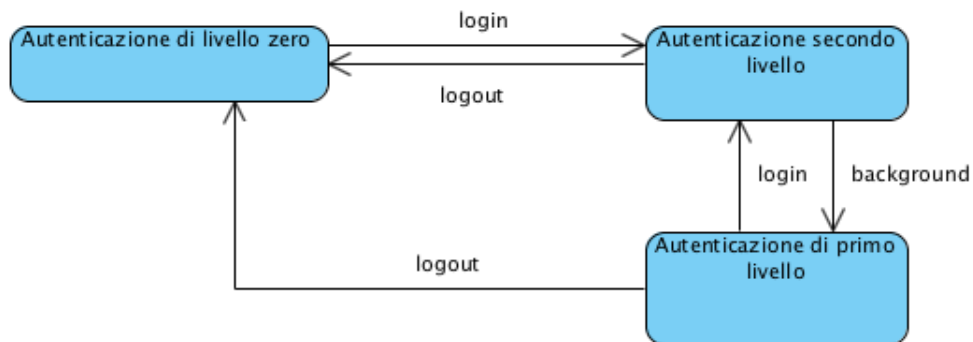


Figura 4.5. Diagramma degli stati di autenticazione

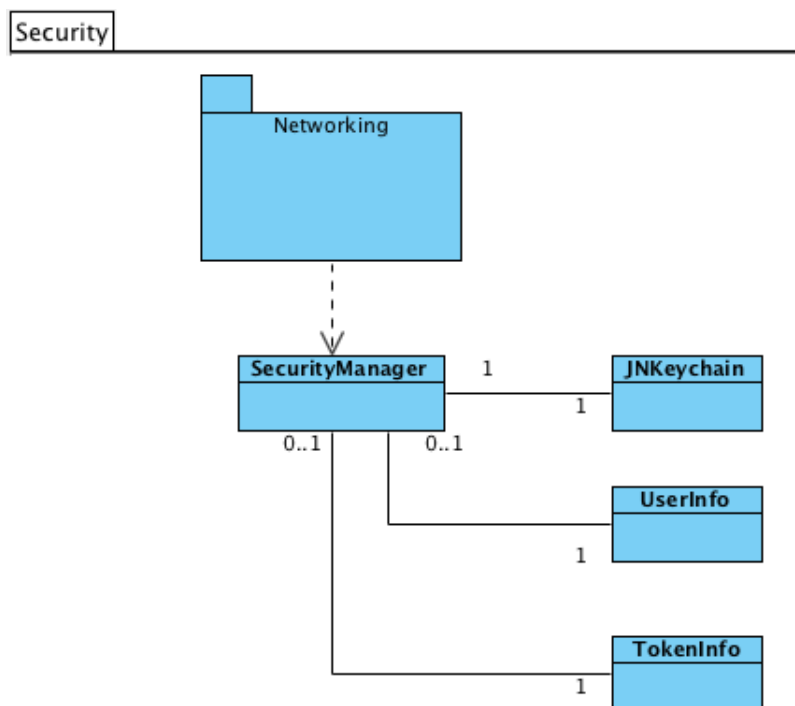


Figura 4.6. Diagramma delle classi relative alla sicurezza

4.3.4 Controller principali

Di seguito è riportato il diagramma delle classi rappresentanti i controller principali creati durante la progettazione e lo sviluppo. Per semplicità di esposizione sono stati rappresentati solo i nomi dei controller e le relazioni tra essi.

A titolo esemplificativo nel diagramma si può notare che il `BaseViewController` è il controller principale dal quale gli altri controller ereditano le funzionalità e proprietà generiche. Tale controller si occupa della creazione e visualizzazione degli elementi grafici comuni a tutte le interfacce (ad esempio la navigation bar personalizzata) e delle logiche necessarie ad ogni controller che lo specializza.

In particolare uno dei compiti principali di questo controller è quello di creare il controller che gestisce l'interfaccia e la logica per il login ai servizi (`LoginViewController`) e di implementare il protocollo `LoginViewControllerProtocol` dichiarato dallo stesso `LoginViewController`. Sarà quindi compito del `LoginViewController` lanciare il processo di login e comunicare al `BaseViewController` l'esito dell'operazione attraverso il protocollo descritto precedentemente; così facendo il `BaseViewController` sarà in grado di effettuare l'aggiornamento dello stato dell'interfaccia.

Controllers

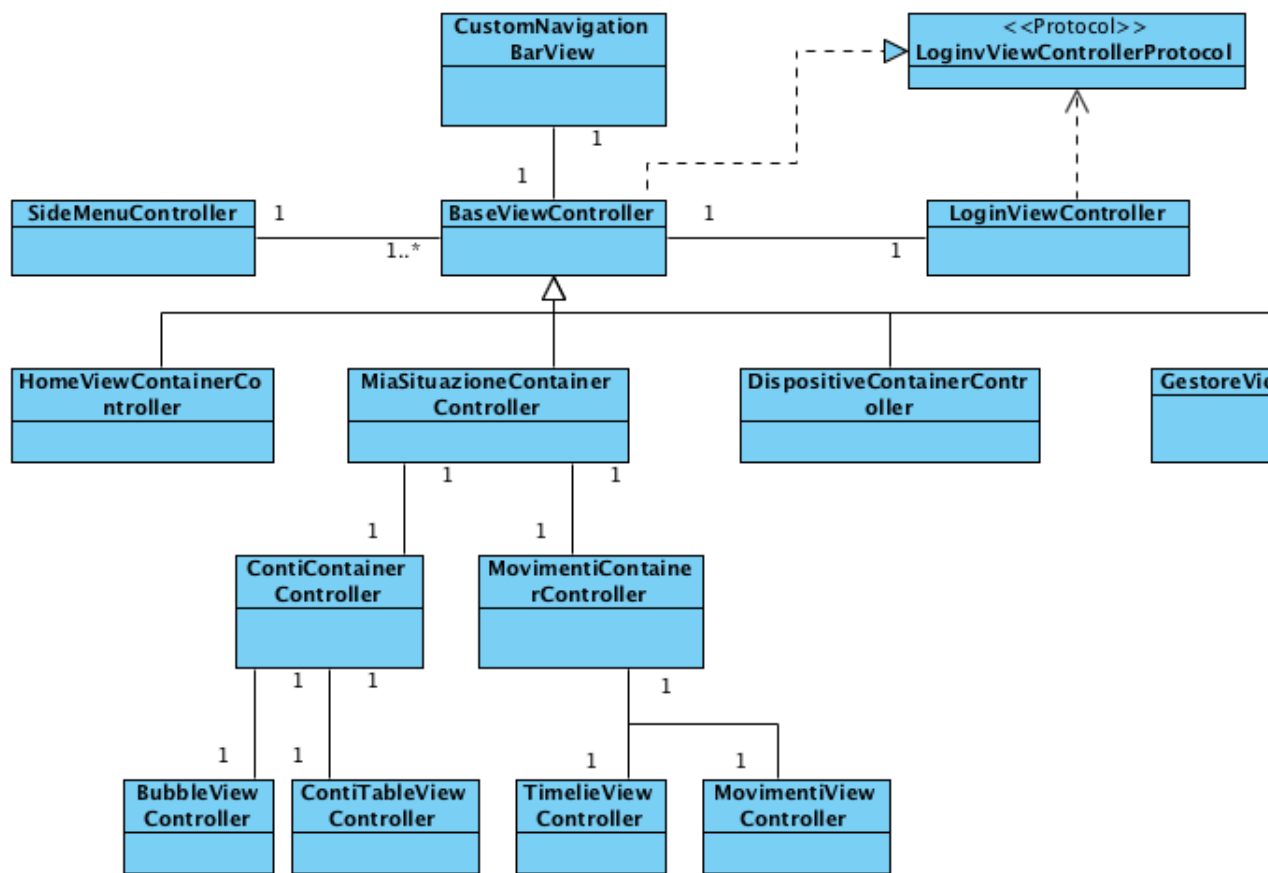


Figura 4.7. Diagramma delle classi controller

Capitolo 5

Dettagli implementativi

In questo capitolo verranno descritti i pattern principali usati durante la progettazione e lo sviluppo del software e le motivazioni che hanno portato a tali scelte.

5.1 Custom container controller

Il *Container View Controller* è un pattern di progettazione iOS che permette di scomporre le applicazioni in parti più piccole e semplici, ognuna delle quali è dedicata ad assolvere un determinato compito. I container controller sono usati quindi come intermediari tra queste diverse parti (*child controller*) allo scopo di fornire un'interfaccia che sia coerente con il contesto in cui i controller figli operano.

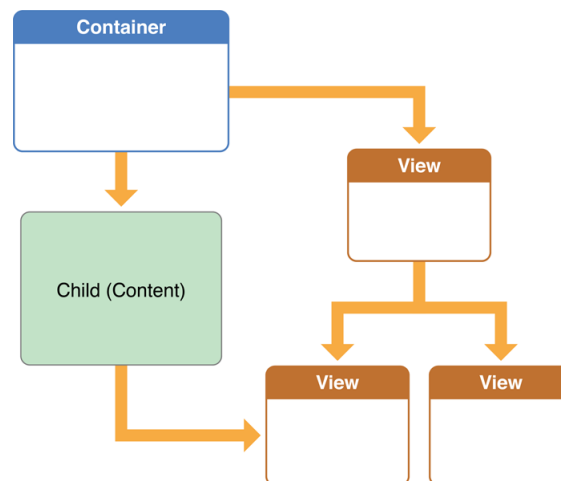


Figura 5.1. Schema del *container controller design*

L'utilizzo di tale pattern ha permesso durante la progettazione dell'applicazione di ottenere controller più puliti sotto il punto di vista implementativo ed efficienti sotto il punto di vista delle prestazioni.

5.1.1 MiaSituazioneContainerController

Uno dei requisiti funzionali principali dell'applicazione prevede la visualizzazione dei conti e carte di un utente e l'eventuale visualizzazione dei movimenti ad essi relativi.

La parte dell'architettura realizzata per realizzare tali funzionalità è composta dalla seguente gerarchia di container e child controller:

- *MiaSituazioneContainerController* (figura 5.2): è il container controller che ha il compito di creare l'interfaccia di base e di creare un array di child controller:
 - *ContiContainerController*: è il container che gestisce la creazione e la comunicazione tra i controller adibiti alla visualizzazione dei prodotti bancari come vista a bolle o vista a tabella (nello specifico *BubbleViewController* e *ContiTableViewController*)
 - *MovimentiContainerController*: è il container che gestisce la creazione e la comunicazione tra i controller adibiti alla visualizzazione dei movimenti di un determinato conto (nello specifico *TimelineViewController* e *MovimentiTableViewController*)
 - *ContiFilterTableViewController*: il controller che si occupa della visualizzazione e della logica di filtraggio dei conti che l'utente può visualizzare a schermo

Quando *MiaSituazioneContainerController* viene istanziato è eseguita una chiamata ai servizi per ottenere la lista dei conti dell'utente, tale lista verrà quindi utilizzata come model comune a tutta la gerarchia dei controller descritti sopra. È compito quindi del *MiaSituazioneContainerController* gestire gli eventi dell'interfaccia principale e passare il model a uno dei controller figli quando l'utente sceglierà una delle modalità di visualizzazione descritte precedentemente.

5.1.2 MovimentiContainerController

Come brevemente descritto nel paragrafo 5.1.1 tale container ha il compito di gestire i child controller adibiti alla visualizzazione dei movimenti di un conto. Nello specifico tale container si occupa della gestione ad alto livello delle funzioni e degli eventi relativi ai componenti dell'interfaccia da esso gestiti (figura 5.6). A titolo illustrativo esso gestisce gli eventi scatenati dal selettore dei conti e dal filtro calendario; in particolare ad ogni cambio di conto o di data del filtro il *MovimentiContainerController* effettua una chiamata ai servizi per ottenere una lista di movimenti che verrà utilizzata come model comune a tutti i figli di tale container.

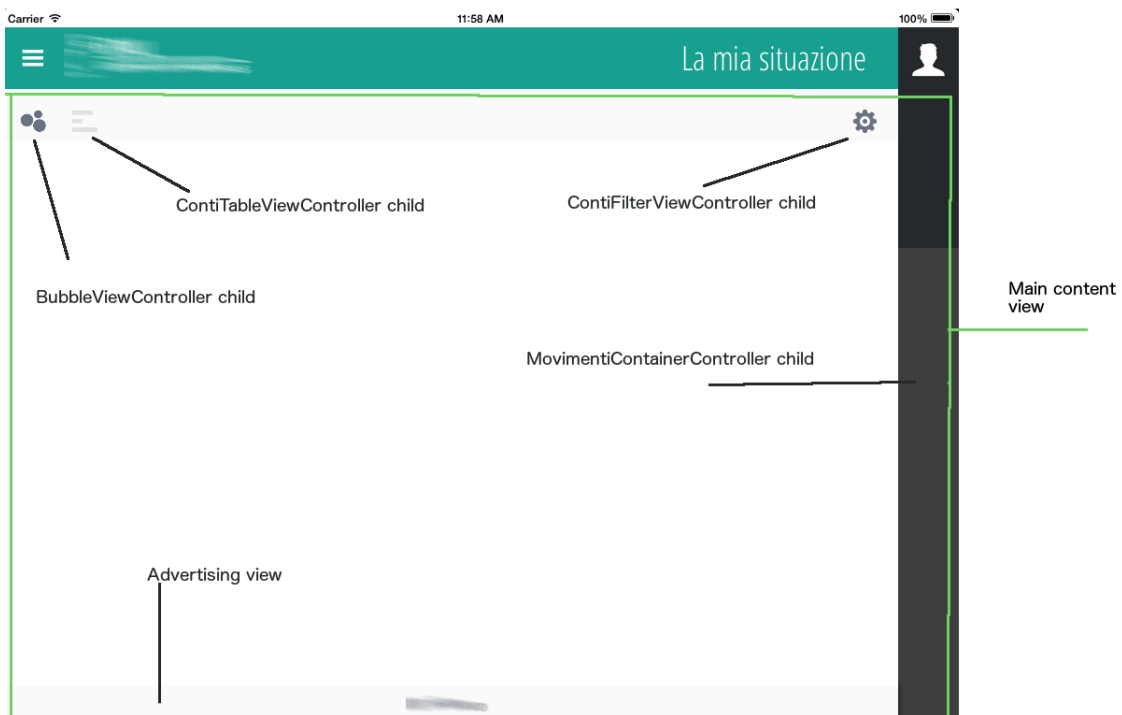


Figura 5.2. Dettaglio componenti gestiti da MiaSituazioneContainerController

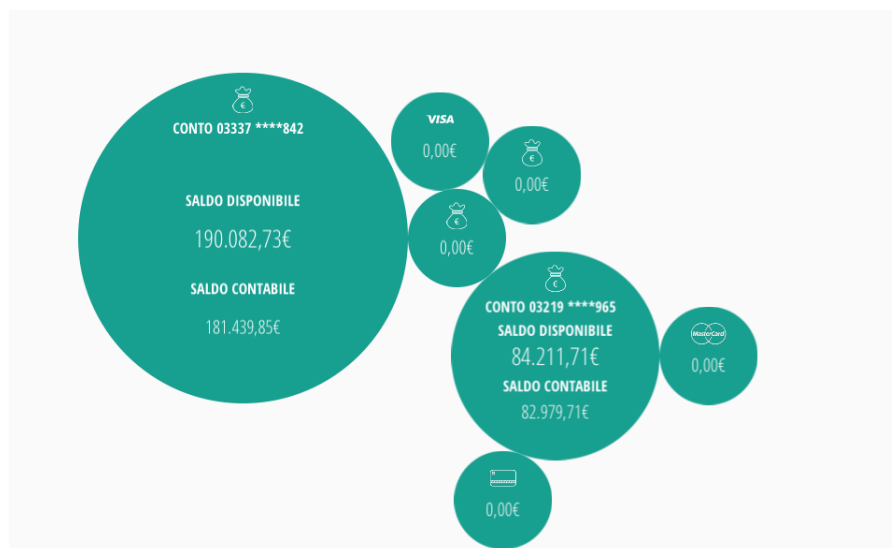


Figura 5.3. BubbleViewController







CONTO CORRENTE 03219 ****916		
SALDO DISPONIBILE	SALDO CONTABILE	IBAN
0,00€	0,00€	IT69L0100503219000000014916 
CONTO CORRENTE 03219 ****965		
SALDO DISPONIBILE	SALDO CONTABILE	IBAN
84.211,71€	82.979,71€	IT12F0100503219000000016965 
CONTO CORRENTE 03337 ****842		
SALDO DISPONIBILE	SALDO CONTABILE	IBAN
190.082,73€	181.439,85€	IT89U0100503337000000004842 
CONTO CORRENTE 03337 ****144		
SALDO DISPONIBILE	SALDO CONTABILE	IBAN
0,00€	0,00€	IT79T0100503337000000005144 
CARTA DI CREDITO ****5047		DATA DI SCADENZA
0,00€		01.01.0001 
CARTA DI CREDITO ****5591		DATA DI SCADENZA
0,00€		01.01.0001 

Figura 5.4. ContiTableViewController

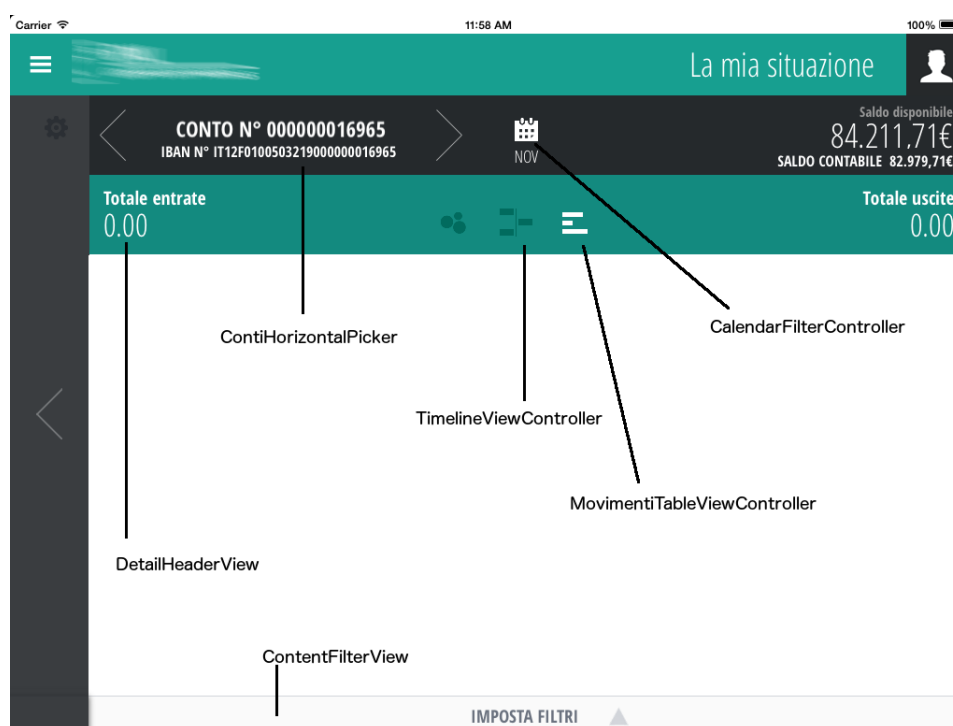


Figura 5.5. Dettaglio dei componenti gestiti dal controller MovimentiContainerController

5.1.3 Comunicazione tra container e child controller

I container controller sono gli unici componenti consapevoli dell'esistenza di altri controller nella gerarchia, in particolare dei child controller da loro direttamente creati. I child controller sono realizzati quindi come componenti indipendenti che non hanno percezione del controller padre o dei loro fratelli.

La comunicazione tra i container e i suoi figli è stata realizzata in modo tale da garantire sempre un'elevata modularità dell'architettura mediante l'utilizzo dei pattern messi a disposizione dall'SDK iOS:

- message passing: realizza la comunicazione in avanti tra un container e un child controller mediante opportune interfacce messe a disposizione dai child controller
- pattern delegate: realizza la comunicazione all'indietro da un child ad un container controller mediante la dichiarazione di opportuni protocolli (*protocol* nella tecnologia iOS) da parte dei child controller. I container controller vengono quindi dichiarati conformi a tali protocolli e posti come *delegate* (oggetti che operano come aiutanti adibiti all'esecuzione di compiti particolari)

L'adozione di tali pattern durante la progettazione ha permesso di ottenere un'architettura modulare composta da componenti specializzati facilmente estendibili e riutilizzabili.

5.2 Posizionamento di oggetti in uno spazio cartesiano

All'interno dell'applicazione è stato realizzato un algoritmo per il posizionamento di oggetti grafici in uno spazio cartesiano. Tali oggetti sono visualizzati come circonferenze all'interno dell'interfaccia grafica e sono utilizzati per riassumere i dati di un conto corrente o di una carta di credito (come importo, tipo di conto, ecc...).

La visualizzazione delle circonferenze ha dovuto rispettare i seguenti vincoli:

- devono esser visualizzate il più possibile una vicino all'altra
- non si devono sovrapporre
- non devono uscire da nessun lato dello schermo
- devono entrare esser tutte visibili

5.2.1 Realizzazione

Per non bloccare l'interfaccia grafica durante il calcolo, l'esecuzione dell'algoritmo di posizionamento avviene in un thread asincrono così da non occupare il *main thread* e lasciare quindi la libertà all'utente di continuare ad utilizzare l'applicazione.

Nei prossimi paragrafi saranno illustrati gli oggetti e le logiche utilizzate per realizzare l'algoritmo e rispettare i vincoli descritti precedentemente.

Bubble

La documentazione Apple suggerisce di non effettuare il disegno e l'allocazione di oggetti grafici (oggetti che estendono la classe `UIView`) al di fuori del main thread, in quanto può portare a problemi di thread safety. Per questo motivo è stata realizzata la classe `Bubble` che ha il compito di astrarre un oggetto view e di incapsulare solo le sue proprietà geometriche (come la dimensione, l'origine, il centro, ecc...). L'algoritmo sfrutterà quindi gli oggetti `Bubble` per effettuare il calcolo del posizionamento nello spazio.

Per decidere in quale punto dello spazio posizionare una circonferenza e per decidere in quale punto su di essa posizionare un vicino, ogni oggetto `Bubble` calcola le seguenti proprietà:

- `frame`: rappresenta l'origine e la dimensione di una bubble
- `center`: le coordinate del centro di una bubble
- `radius`: il raggio
- `anchors`: un array di oggetti `AnchorPoint`

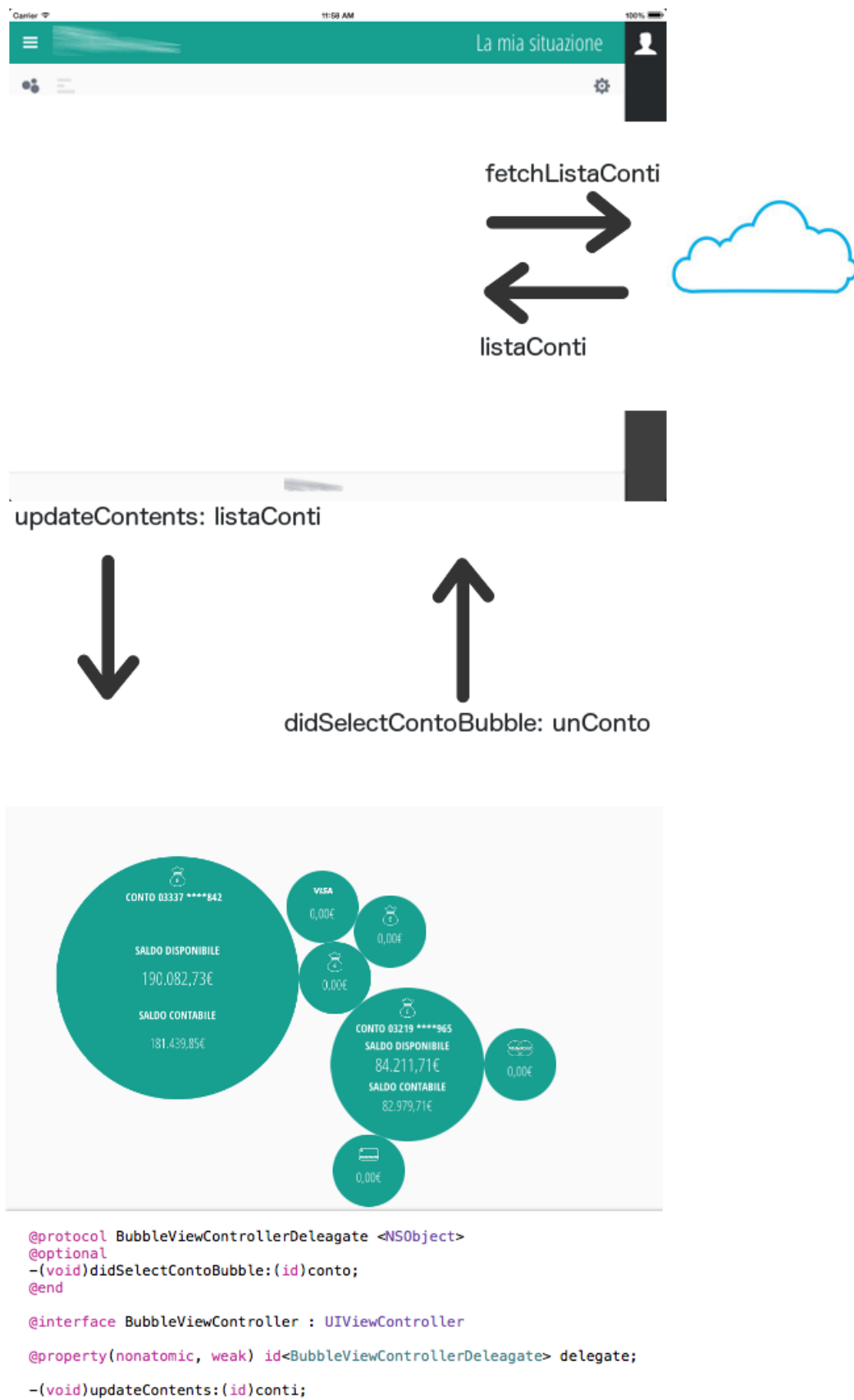


Figura 5.6. Esempio di comunicazione da un container padre a un child controller

AnchorPoint

La classe `AnchorPoint` rappresenta le coordinate (x, y) di un punto su una circonferenza. Tale oggetto è utilizzato per decidere in quale posizione su di una bubble aggiungere un'altra bubble (figura 5.7).

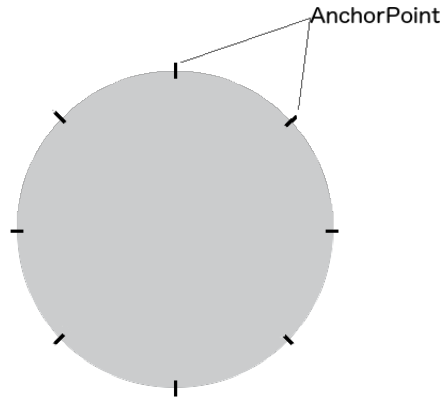


Figura 5.7. Esempio di `AnchorPoint` su una circonferenza

Algoritmo

Di seguito saranno descritte in pseudo-codice le procedure che compongono l'algoritmo di posizionamento.

La procedura 5.1 calcola la dimensione di ogni bubble in base al saldo di un conto e controlla se nell'area delimitata dall'interfaccia tali bubble entrano, altrimenti esegue un ridimensionamento e riesegue ricorsivamente il calcolo.

Listing 5.1. `startBubblesAlgorithm`

```
startBubblesAlgorithm(float resizePercent){  
  
    for each(conto in contiList){  
        bubbleSize = calculateBubbleSizeForConto(conto)  
        add bubble to sizesArray  
    }  
  
    bool isGood = checkArea(sizesArray)
```

```
    if (isGood){
        createBubbles
        placeBubblesInSpace
    }
    else {
        startBubblesAlgorithm (0.1)
    }
}
```

La procedura 5.2 esegue il calcolo delle posizioni degli oggetti Bubble all'interno dell'area delimitata dall'interfaccia. Questo calcolo viene ripetuto fino a quando non si trova una disposizione ideale delle circonferenze. Al termine della computazione verranno creati gli oggetti **BubbleView** che saranno successivamente disegnati all'interno dell'interfaccia grafica.

Listing 5.2. placeBubblesInSpace

```
placeBubblesInSpace{

    bool finished = false

    while(finished == false){

        //calcola la posizione per ogni Bubble
        calculateOriginForBubble

        if(recalculate){

            //pulisce le strutture dati
            //create dai calcoli precedenti
            cleanOldData
            //ricrea gli oggetti Bubble
            //con ridimensionamento
            resizeValue += 0.1
            createBubbles
        }
        else{
            finished = true;
        }
    }
}
```

```

        }
    }

    //disegna sull'interfaccia le BubbleView
                                //ottenute dalle Bubble
    drawBubbleViewInSpace
}

```

La procedura `calculateOriginForBubble` si occupa di posizionare la prima Bubble al centro dello schermo. Nei passi successivi per ogni Bubble già posizionata considera i suoi `AnchorPoint` e controlla se ne esiste uno utilizzabile: cioè un punto su una Bubble tale che affiancandogli un'altra Bubble quest'ultima non esce dai lati dallo schermo o si sovrapponga alle altre già posizionate. Se tale punto esiste la Bubble viene considerata come *posizionata*, altrimenti vengono controllati iterativamente gli `AnchorPoint` di tutte le altre Bubble già posizionate.

Questo meccanismo permette di sfruttare tutta l'area a disposizione ricontrollando iterativamente gli spazi che avanzano dopo il posizionamento di una Bubble, in modo tale da ottenere una distribuzione uniforme delle circonferenze.

Listing 5.3. `calculateOriginForBubble`

```

calculateOriginForBubble{

    for each(newBubble in bubbles){

        BOOL isFound = NO;

        //se prima bolla
        if(newBubble isFirst){
            add newBubble to placedBubbles
            continue;
        }

        BOOL intersect = NO;

        //cerco posizione per newBubble tra gli anchor point
        delle bubble gi posizionate
        for(bubble in placedBubbles){

            //controllo anchorPoint di bubble

```

```
for each (anchorPoint in bubble){

    //un punto di ancoraggio
    anchorPoint = bubble.getRandomAnchorPoint

    if(anchorPoint.isBusy){
        //se occupato salto ciclo
        continue;
    }

    intersect = NO;

    //provo con questo punto
    //calcolo il centro dove posizionare la
        newBubble
    //rispetto l'anchorPoint selezionato e il
        centro ad esso relativo
    aCenter = calculateCenter:bubble.center ,
        point.angle ,newBubble.radius + bubble.
        radius)

    newBubble.center = aCenter;

    //salto se con questa posizione si esce
        sotto , sopra o a sinistra dello schermo
    if(exit from screen){
        continue;
    }

    //termino se esce a destra dello schermo
    if(aCenter.x + newBubble.radius >
        screenFframe.size.width)
        recalculate = YES;
        return;
    }

    //controllo se newBubble si interseca con le
        altre view gi posizionate
```

```
for(bubble2 in placedBubbles){

    if (bubble != bubble2){

        distance = distanceFrom:bubble2.
            center ,aCenter);
        extendedRadius = newBubble.radius +
            bubble2.radius;

        //se distanza tra due centri
        minore della somma dei raggi, si
        intersecano
        if( distance < extendedRadius){
            //si intersecano
            intersect = YES;
            break;
        }
    }
}

//se finito ciclo allora newBubble non si
interseca con nessun'altra bubble
if(intersect == false){
    //segno come occupato l'anchor point in
    esame
    point.isBusy = true;
    isFound = true;
    //interrompo controllo degli altri
    anchor point
    break;
}

if(isFound ){
    //interrompe ricerca sulla prossima bolla
    tra quelle inserite
    break;
}
```

```
    }

    if (!isFound){
        //Arria qui se ha finito tutti gli anchor point
        //di tutte le bolle gi inserite.
        recalculate = YES;
        return;
    }

    //se arrivato qui aggiungo newBubble alle gi
    //posizionate
    add newBubble to placedBubbles
}
}
```


Capitolo 6

Sviluppi futuri

