TOD

Today To date To do

Martina Cavallucci, Claudia Severi, Alfredo Tonelli

21 agosto 2017

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software mira alla realizzazione di un sistema che fornisca all'utente la possibilità di organizzare e gestire al meglio il proprio tempo. TOD (Today, To date, To do) permette di gestire informazioni personali, eventi, appuntamenti e compleanni fornendo in aggiunta servizi di utilità atti ad aiutare l'utente con le attività giornaliere come una rubrica personale e una lista di compiti da fare.

Requisiti funzionali

- TOD dovrà occuparsi della gestione di eventi, compleanni e appuntamenti mettendo a disposizione dell'utente funzionalità personalizzabili a seconda delle proprie esigenze.
- TOD fornirà la possibilità all'utente di interagire con il sistema attraverso un'interfaccia user-friendly, permettendo di creare, modificare ed eliminare informazioni in maniera semplice.
- L'applicazione dovrà fornire una rubrica personale in cui inserire tutti i contatti e collegarli ai rispettivi appuntamenti e compleanni.
- Il sistema permetterà di creare più calendari nel caso l'applicazione venga usata da più utenti, ognuno potrà accedere al suo dopo aver effettuato il log-in.
- TOD dovrà occuparsi di notificare promemoria riguardanti gli eventi inseriti dagli utenti, nel caso i promemoria si verificassero a sistema spento TOD dovrà avvisare l'utente di questi subito dopo il login.
- Il sistema dovrà memorizzare calendario, rubrica e lista di to do in modo da ritrovare tutte le informazioni inserite dall'utente alla riapertura dell'applicazione.

Requisiti non funzionali

- Il sistema dovrà criptare le password degli utenti per garantire sicurezza e riservatezza dei dati.
- TOD deve gestire automaticamente i compleanni collegati ai contatti presenti in rubrica.

1.2 Analisi e modello del dominio

TOD dovrà gestire più utenti, ognuno con il proprio calendario, rubrica e lista di to do.
Ogni utente può visualizzare una rubrica con all'interno tutti i contatti personali completi delle informazioni aggiunte in fase di inserimento. Un utente può consultare una lista di to do ovvero un elenco di compiti che deve svolgere. Infine l'utente ha a disposizione un calendario nel quale può aggiungere degli eventi sotto forma di impegno o appuntamento con uno o più contatti della rubrica.
Gli elementi costitutivi il problema sono sintetizzati in Figura 1.1

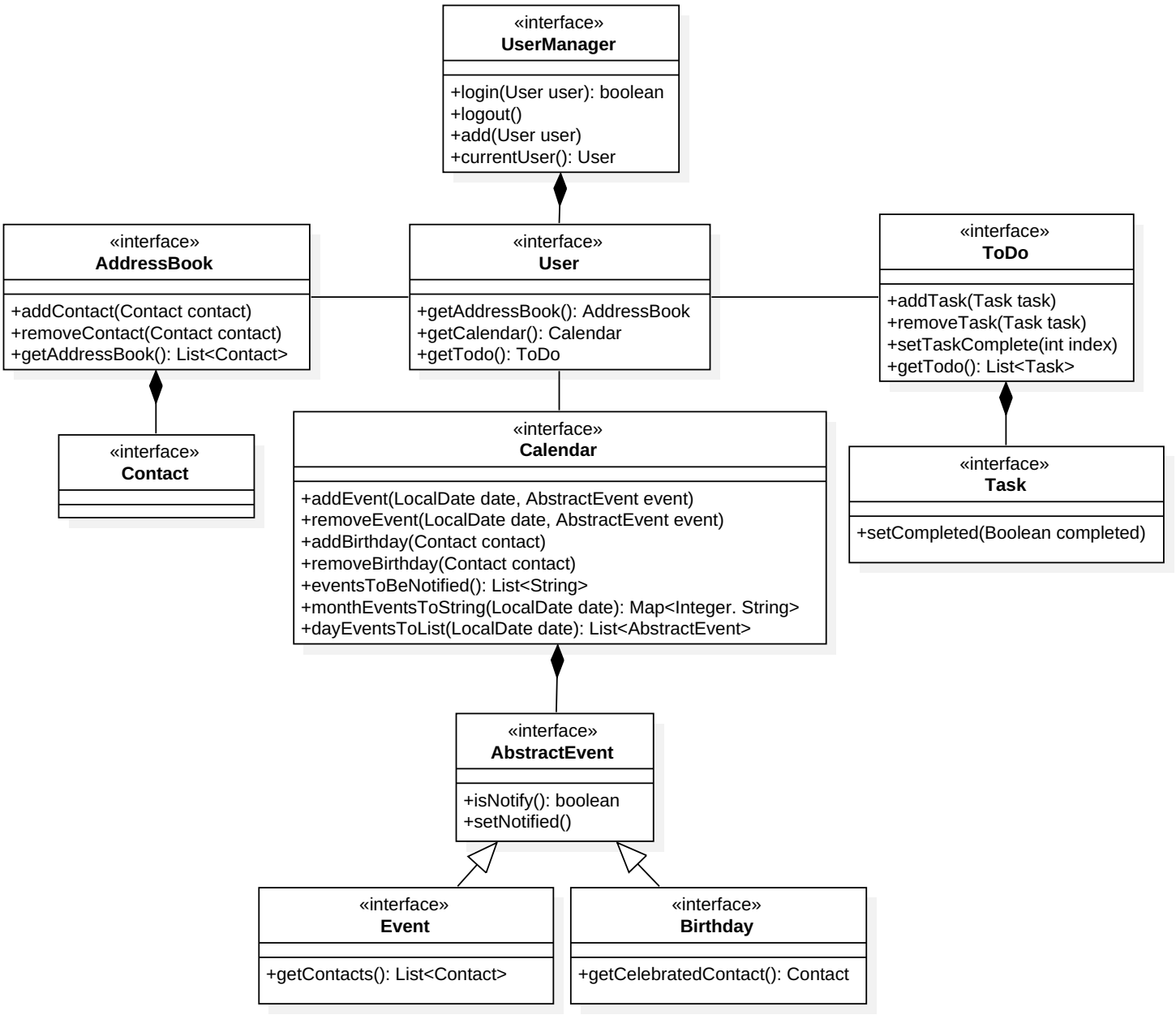

Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

Il progetto è stato sviluppato seguendo il pattern architetturale MVC, in maniera tale da rendere indipendenti le tre parti che compiono compiti distinti.
Il Controller si occupa di coordinare il Model e la View, interagendo con l'utente tramite un'interfaccia intuitiva, e gestendo le sue scelte tramite chiamate successive su metodi del Model. Abbiamo seguito questo pattern poiché, oltre a semplificare la divisione dei compiti tra i vari componenti del gruppo, offre un'elevata estensibilità al sistema.
In particolare ogni GUI è coordinata dal rispettivo Controller, il quale viene chiamato dalla View associata all'atto di particolari eventi che comportano modifiche dei dati.
La View notifica i cambiamenti al Controller che di conseguenza interroga il modello, il quale produce risultati, successivamente il Controller setta i dati prodotti nella rispettiva interfaccia grafica inoltre esso accede al modello sostanzialmente in un entry-point: MainController, ContactController, ToDoController e CalendarController accedono tramite UserManager, infatti ogni utente ha associata la propria rubrica, lista di cose da fare ed eventi.
Abbiamo cercato di rispettare le politiche dell'MVC dove, la View non esegue calcoli (se non strettamente legati alla realizzazione di componenti grafici come la creazione di un calendario) ma delega tutto al Controller.
Il Model non conosce l'esistenza dell'interfaccia grafica, il Controller si avvale semplicemente dei metodi pubblici del Model per estrapolare (o settare) dati del modello in modo che, se cambiando o modificando l'interfaccia grafica le modifiche da fare saranno solo nel Controller.

2.2 Design dettagliato

Cavallucci Martina(View)
Le interfacce grafiche principali sono LoginView, CalendarView, ContactView e toDoView ognuna di esse implementa l'interfaccia "View", la quale definsce di default alcune operazioni di base che ogni view deve mettere a disposizione al Controller,

ad esempio mostrare messaggi di errore dentro ad un JPanel oppure il metodo di inizializzazione e settaggio della GUI. Oltre ad estendere da View ogni interfaccia grafica viene gestita da una classe chiamata "MainViewTabbed" che ha il compito di gestire ogni tab che comparirà sulla sinistra della schermata principale e sarà accessibile per tutto il tempo di attività all'utente. Essa risulta quindi di utilità per gestire ogni servizio che TOD mette a disposizione all'utente.
Come si evidenzia dall'UML in figura i contatti e i to do registrati necessitano di essere visualizzati dall'utente quindi verranno inseriti in una tabella che verrà aggiornata per poter manipolare gli oggetti creati. L'interfaccia di login invece ha un metodo che permette di pulire il form all'occorrenza. L'interfaccia principale MainViewTabbed contiene i metodi principali per gestire le relative tab, quindi per poterle aggiungere, switchare e andare in una determinata tab selezionata.

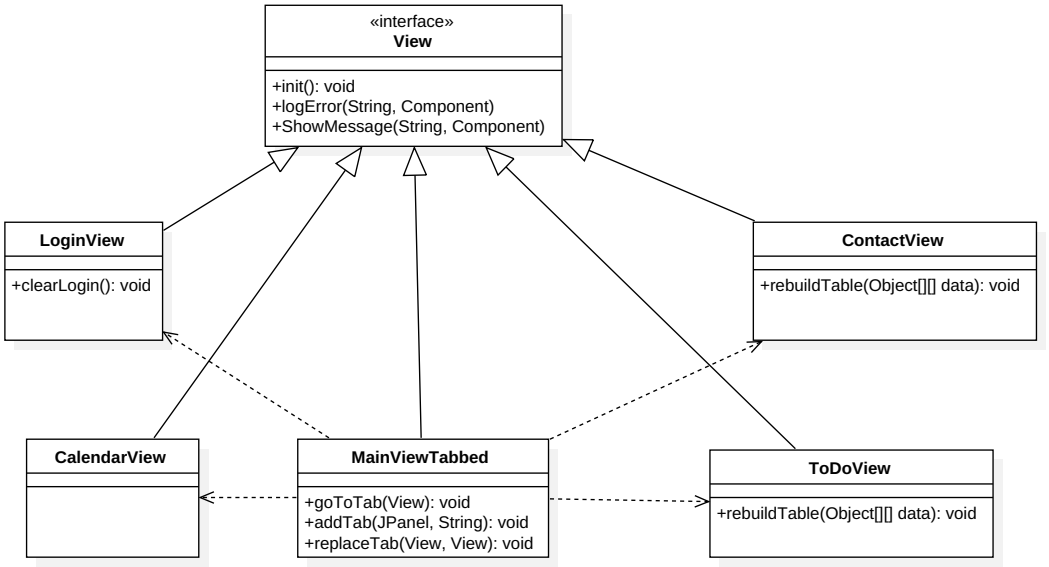

Figura 2.1: Schema UML rappresentante la relazione tra le varie interfacce dei tab

La classe LoginViewImpl costruisce la form di login e in base alle scelte dell'utente il Controller potrebbe far apparire ulteriori frame come quello per cambiare la password (ChangePasswordViewImpl) oppure il frame per la registrazione di un utente(AddUserViewImpl). Se l'utente inserisce le giuste credenziali automaticamente il MainController controlla la correttezza e fa visualizzare le tab principali con i dati salvati in precedenza invece se ciò non avviene verrà visualizzato un opportuno messaggio di errore.
L'utente potrà fare logout dal suo profilo e ciò comporterà la chiusura dell'applicazione e il salvataggio dello stato di tutte i dati dell'utente. Questa vista sarà visibile all'utente per tutta la durata dell'applicazione in modo che in ogni momento potrà gestire il suo profilo.

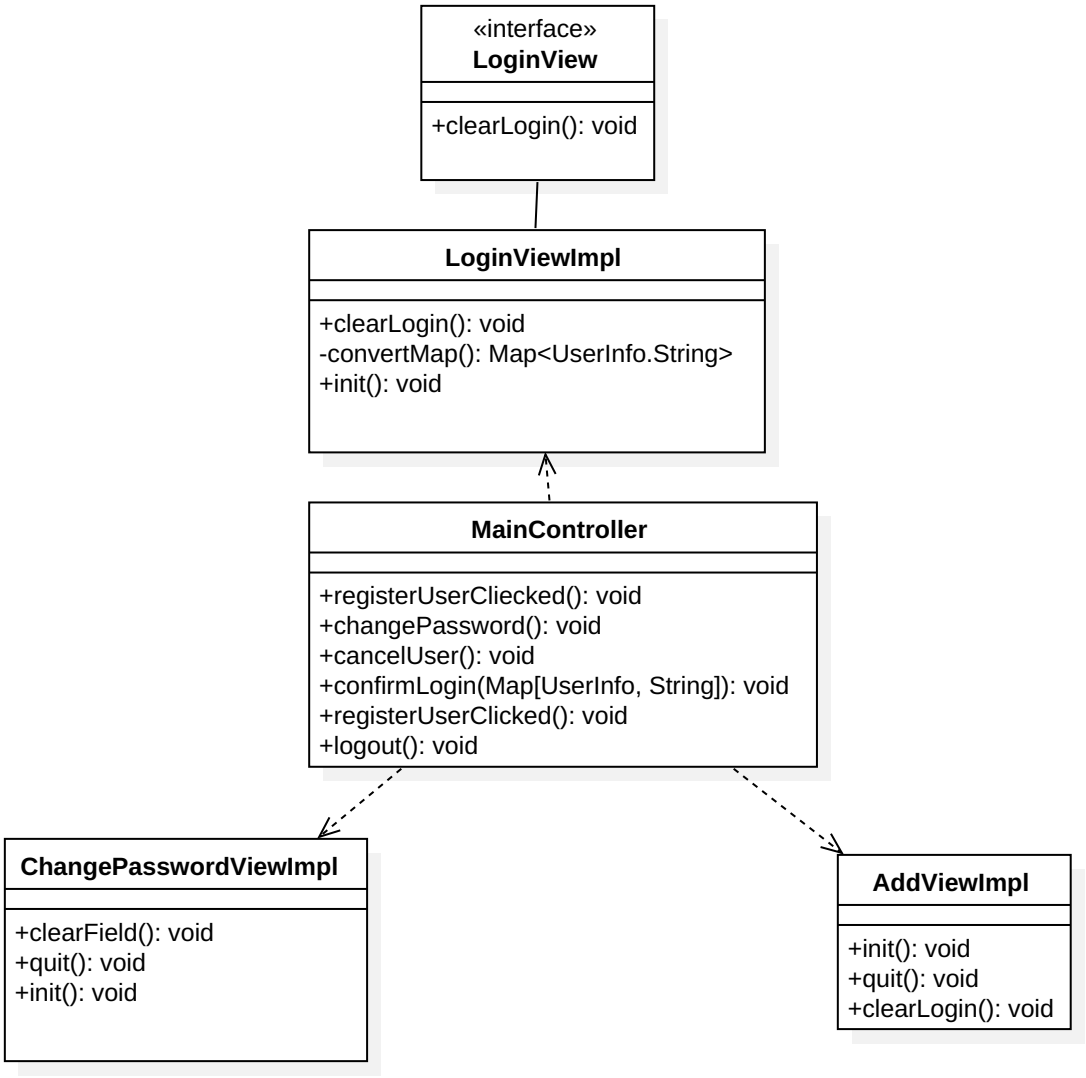

Figura 2.2: Schema UML rappresentante la relazione tra le classi e le interfacce che gestiscono il login di un utente

La classe CalendarViewImpl costruisce la tabella che costituirà il calendario, tale tabella conterrà i giorni del mese e dell'anno selezionato i quali saranno sensati, ciò è permesso da un algoritmo che posiziona i giorni giusti e controlla se il mese è bisestile in un certo anno.
Con il pulsante "Show" l'utente avrà la possibilità di vedere tali giorni e di caricare eventuali eventi salvati, invece con il pulsante "Refresh" si potrà aggiornare il calendario una volta aggiunto un evento.
Per l'aggiunta di un evento occorre cliccare due volte sulla cella del giorno selezionato e si aprirà un frame (EventSummaryViewImpl) che permette la creazione di un appuntamento associato o meno con i contatti registrati dall'utente.

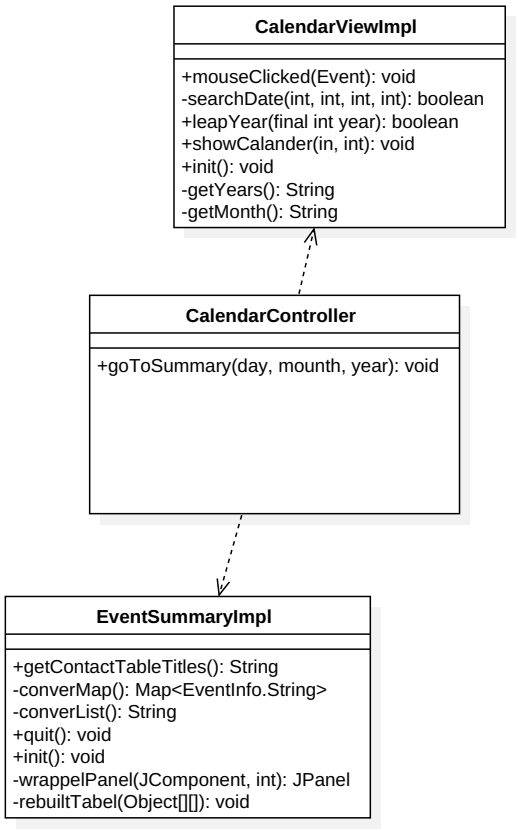

Figura 2.3: Schema UML rappresentante la relazione tra le classi che gestiscono il calendario e la GUI di creazione eventi

La classe ContactViewImpl costruisce un pannello per l'inserimento e la visualizzazione di contatti, è possibile inserire informazioni di una certa persona alcune obbligatorie altre invece opzionali.
I contatti salvati compariranno in un'apposita tabella creata che si aggiorna con gli inserimenti, tali informazioni potranno venire modificate o eliminate.
Importante funzionalità di TOD consiste nel ricordare e quindi di salvare come evento il compleanno di un utente salvato così ogni anno si riceverà la notifica.
Vincolo della nostra applicazione è quello che solo se in questa sezione si salvano contatti verranno poi visualizzati nel riepilogo dell'evento e sarà possibile associarli.

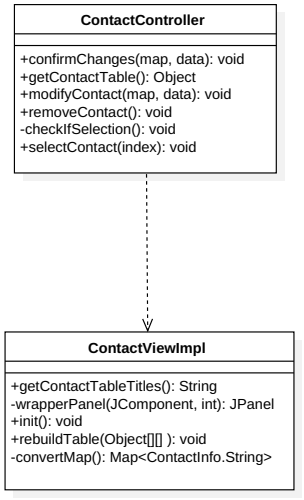

Figura 2.4: Schema UML rappresentante la relazione tra le due classi che gestiscono i contatti e il loro inserimento

La classe ToDoViewImpl costruisce un pannello che permette di creare e visualizzare i to do salvati dall'utente.
Di tale impegno potrà essere cambiato il suo stato e potrà essere eliminato tramite gli appositi pulsanti.

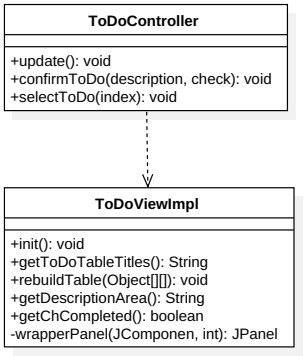

Figura 2.5: Schema UML rappresentante la relazione tra le due classi che gestiscono i todo e il loro inserimento

Severi Claudia (Controller)
Il controller è la parte del progetto con il compito di mettere in relazione View e Model, ricevendo dalla prima le informazioni per modificare lo stato degli oggetti principali del programma. Tutta la sua logica è contenuta nell'omonimo package. Il Controller si occupa anche del salvataggio e della rilettura di dati rilevanti, come gli eventi salvati, i contatti, i to-do e gli username e password degli utenti.

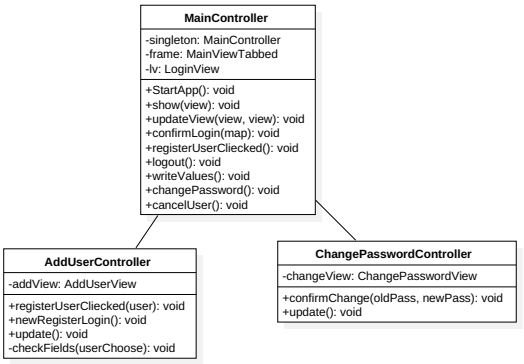

Figura 2.6: Schema UML rappresentante la relazione tra il main controller e le classi che controllano uno user

Come si può notare dagli UML la classe principale è data dal MainController al quale verranno collegate le varie schermate e i vari Tab, infatti al lancio dell'app sarà dapprima eseguito il MainController che darà la possibilità di loggarsi con il proprio utente o di crearne uno nuovo. Una volta loggato l'utente potrà tramite la classe changePassword cambiare la propria password, tramite una apposita interfaccia lanciata dal MainController.

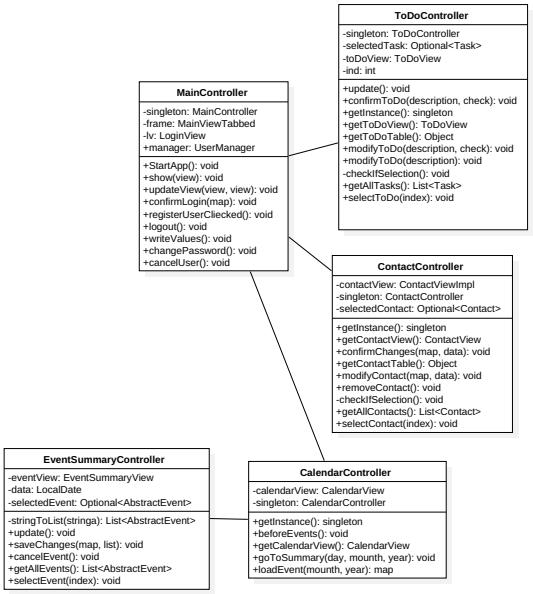

Figura 2.7: Schema UML rappresentante la relazione tra il main controller e le classi che controllano le informazioni che lo user intende manipolare

Una volta fatto il login il MainController tramite il metodo userLogged provvederà a lanciare i tab contenenti le interfacce delle varie classi quindi: calendario, contatti e to-do, che sono quindi strettamente collegati al MainController. Le principali classi, come si può notare sono caratterizzate dall'utilizzo del pattern Singleton. Inoltre il MainController si occupa della serializzazione dei dati e della scrittura su file, prendendo i dati dal fine 'user storage.dat' creato al momento dell'avvio se non esistente e caricandoli al momento del logout dello user. Attraverso l'attributo manager nel MainController sarà possibile personalizzare l'esperienza dell'utente.
Tramite la classe ContactController sarà possibile aggiungere, eliminare e modificare un contatto e il sistema provvederà ad aggiungere il compleanno relativo al contatto nella schermata del calendario. Nella classe To-Do sarà possibile aggiungere ed eliminare task, permettendo di creare una lista di cose da fare e modificarne lo stato da "Complete" a "Incomplete" e viceversa. Tramite la classe CalendarController sarà possibile visualizzare un riepilogo degli eventi mese per mese e sarà possibile aggiungere un nuovo evento, o eliminarne uno facendo doppio click sul giorno scelto. Verrà a questo punto eseguita la classe EventSummaryController che darà la possibilià all'utente di eseguire le azioni precedentemente elencate.

Tonelli Alfredo (Model)
All'interno del progetto mi sono occupato della realizzazione delle classi e in generale di ciò che riguarda la componente Model del pattern architetturale MVC. La classe principale sulla quale si basa l'applicazione è UserManagerImpl che contiene al suo interno tutti gli utenti e mette a disposizione metodi gestire login, logout, aggiunta, eliminazione e crea una scorciatoia tramite la quale il controller può richiamare un metodo che gli restituisce l'utente attualmente loggato.
Per la suddetta classe è stato utilzzato il design patter Singleton in modo che venga istanziato un solo manager con il quale effettuare tutte le operazioni, l'implementazione thread-safe comprende una classe statica ManagerHolder che verrà inizializzata al primo utlizzo.
Risulta fondamentale questo aspetto in quanto, come già detto, il manager contiene al suo interno gli utenti dell'applicazione e istanziando o lavorando su più di un manager si avrebbero informazioni diverse per ognuno.
All'interno della classe UserImpl, che rappresenta un utente dell'applicazione, si trovano i tre principali raccoglitori di informazioni riguardanti l'utente: la classe CalendarImpl contiene il calendario, classe AddressBookImpl contiene la rubrica e la classe ToDoImpl contiene i compiti che l'utente inserisce in una lista. Tramite gli appositi getter è possibile accedere direttamente alle tre classi per effettuare le operazioni messe a disposizione da ciascuna classe.

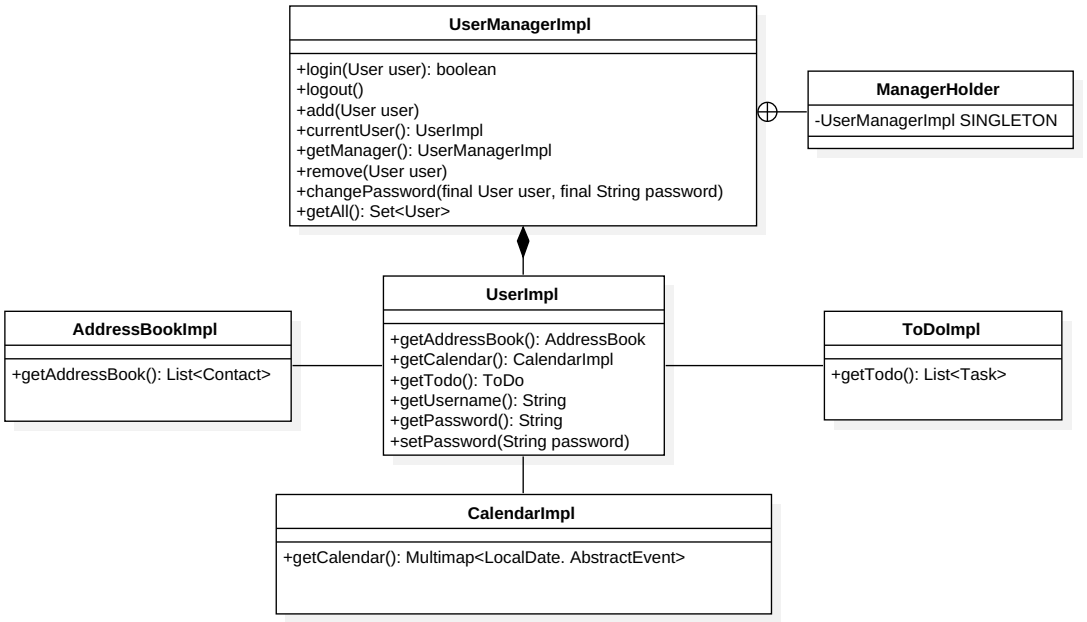

Figura 2.8: Schema UML rappresentante la relazione fra UserManagerImpl, UserImpl, AddressBookImpl, CalendarImpl, ToDoImpl

All'interno della classe UserImpl si trova la classe AddressBookImpl che contiene la rubrica personale di ogni utente. AddressBookImpl consiste a sua volta in un contenitore di ContactImpl e mette a disposizione le operazioni principali di aggiun-

ta, modifica ed eliminazione dei contatti, nonché quella di accesso alla collezione. Per inserire un contatto non è necessario compilare tutti i campi disponibili, per questo ho deciso di utilizzare il patter creazionale Builder implementato tramite una classe innestata.

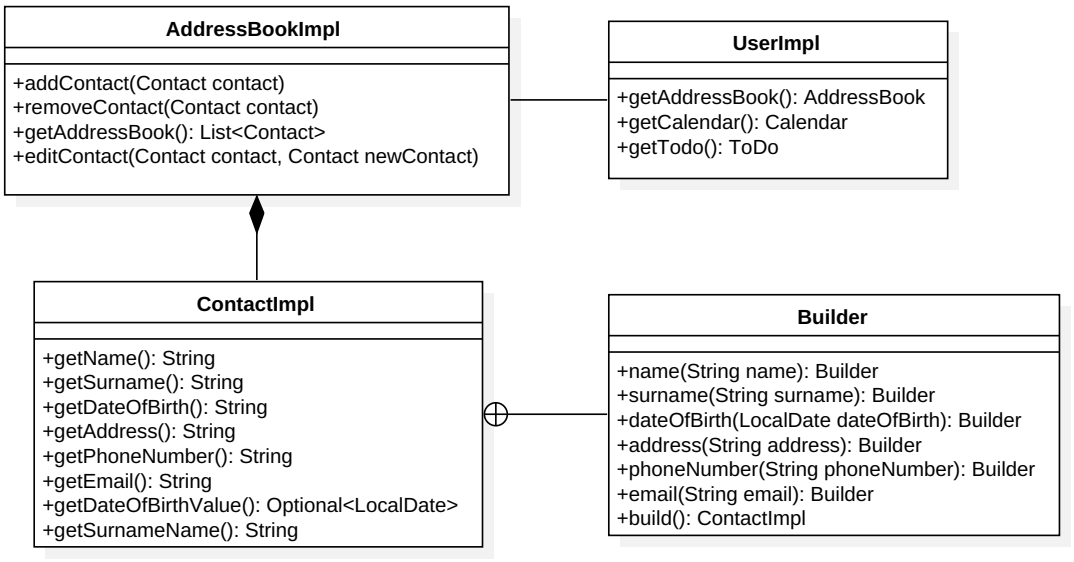

Figura 2.9: Schema UML rappresentante la relazione fra UserImpl, AddressBookImpl e ContactImpl

La classe UserImpl contiene al suo interno ToDoImpl che consiste in un contenitore di TaskImpl, ovvero i compiti che l'utente vuole segnarsi di fare. ToDoImpl fornisce i metodi principali per aggiungere, rimuovere, modificare ed accedere alla collezione di TaskImpl, sono inoltre messi a disposizione due metodi per modificare lo stato di completamento di un Task.

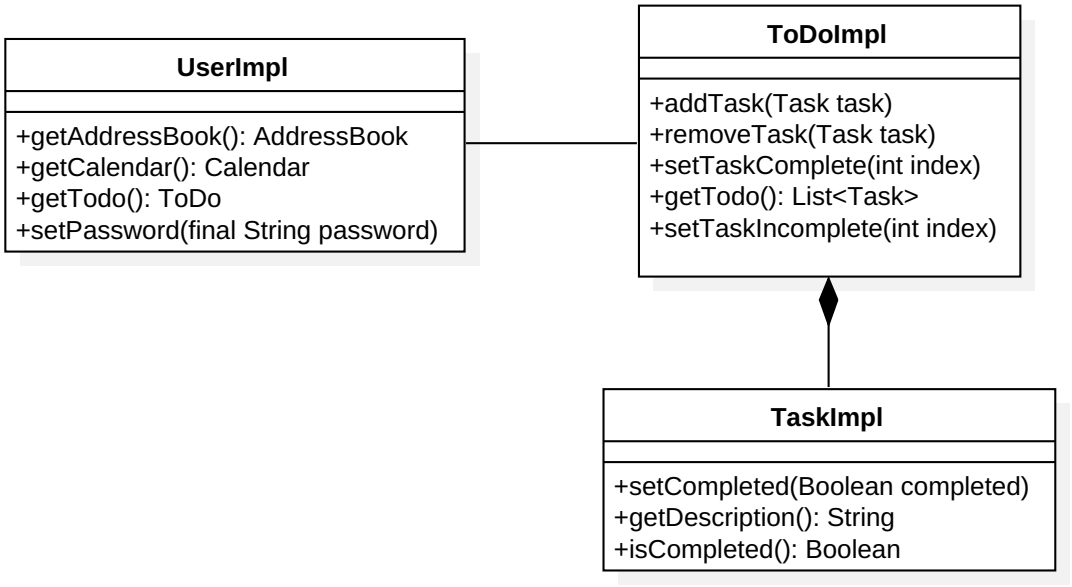

Figura 2.10: Schema UML rappresentante la relazione fra UserImpl, ToDoImpl e TaskImpl

CalendarImpl, contenuta all'interno di UserImpl, è la classe che rappresenta il calendario personale dell'utente dell'applicazione.
CalendarImpl è un contenitore di AbstractEventImpl, ho deciso di utilizzare il pattern creazionale Template Method perché EventImpl e BirthdayImpl hanno campi e metodi in comune che ho riunito in AbstractEventImpl. Giustamente non può essere istanziato un oggetto del tipo di quest'ultimo.
CalendarImpl mette a disposizione metodi per l'aggiunta, eliminazione di eventi e compleanni, oltre a vari metodi per effettuare diverse tipologie di ricerche sul calendario.
Per la creazione di EventImpl ho deciso di utilizzare il pattern creazionale Builder implementato tramite una classe innestata. Questa scelta nasce dalla necessità di non dover inserire tutti i campi disponibili per un evento.

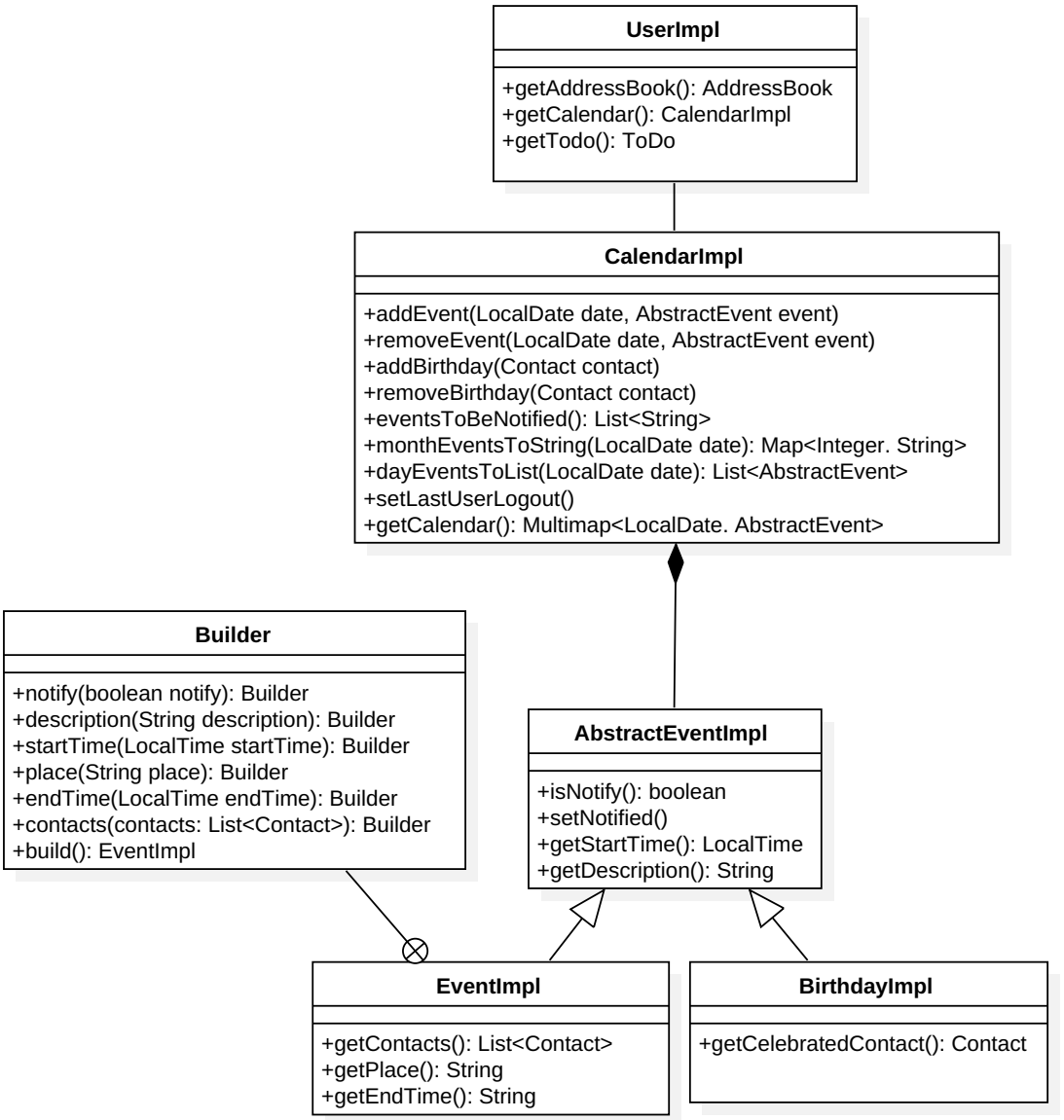

Figura 2.11: Schema UML rappresentante la relazione fra UserImpl, CalendarImpl, AbstractEventImpl, EventImpl, BirthdayImpl

La classe statica di utility PasswordUtility mette a disposizione un metodo per la cifratura della password tramite l'utilizzo di un algoritmo ci cifratura che genera il fingerprint della password inserita dall'utente.
Questa classe permette di mettere al sicuro la password di accesso dell'utente dato che il risultato dell'algoritmo di cifratura non permette di risalire alla password originale.

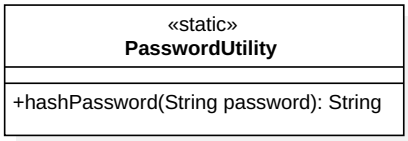

Figura 2.12: Schema UML rappresentante la classe statica PasswordUtility

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per il test automatizzato dell'applicazione abbiamo utilizzato Junit, cercando di sottoporre a test più metodi e funzionalità possibili del package Model, cercando di garantirne la consistenza delle operazioni del Model e avere così il minor numero di bug da correggere in un eventuale futuro. Mentre per la View ci siamo soffermati a svolgere numerosissimi test manuali atti a verificare il corretto funzionamento dell'interfaccia grafica, la corretta posizione degli elementi interattivi, nonché sui meccanismi di eccezioni, collegamento tra le varie View, passaggio dei corretti dati e tutti quei meccanismi introdotti dalla View per un corretto funzionamento dell'applicazione. Per quanto riguarda il Controller, i test sulla View hanno permesso di verificare il corretto funzionamento insieme ad alcuni test sul Controller stesso per verificare la corretta gestione di eccezzioni, salvataggi e collegamento tra MainController e vari tab.

3.2 Metodologia di lavoro

Come già in parte rispecchiato della sezione di Design di dettaglio il team si è diviso i compiti nel seguente modo:

Cavallucci Martina (View): Mi sono occupata di comunicare al Controller gli eventi che accadono all'interno della View, inoltre ho gestito interamente la fase di set del vari componenti come i tab che vanno inseriti tramite il controller nella MainClass.

Severi Claudia (Controller): mi sono occupata di far interagire Model e View per qualsiasi funzione che prevedesse un certo intervento da parte del Model stesso. Mi sono occupata della gestione di parti prettamente del Controller, e del salvataggio dei dati su file, in modo da rendere i dati persistenti anche dopo la chiusura del programma. Inoltre mi sono occupata del caricamento dei dati sull'applicazione, passandoli alla View quando necessario.

Tonelli Alfredo (Model): mi sono occupato del Model, ovvero delle classi e delle interfacce presenti all'interno del package Model per la gestione di tutta la parte riguardate le classi in cui memorizzare e gestire le informazioni dell'utente.
Mi sono inoltre occupato dello sviluppo delle classi di test JUnit per testare le principali funzioni delle classi sulle quasi si basa il funzionamento generale dell'applicazione.

3.3 Note di sviluppo

Il nostro lavoro ha avuto inizio con lo stabilire insieme l'architettura del sistema, per poi procedere a sviluppare indipendentemente le nostre parti individuali. Ci siamo però accorti, a volte, di aver dimenticato qualcosa in fase di design architetturale, il quale ci ha costretti a tornare indietro modificando e aggiungendo funzionalità al nostro progetto. Per quanto riguarda lo sviluppo, ci siamo posti dei goal intermedi, in modo da completare step-by-step (ad esempio completando dapprima il login, successivamente To-do e Contacts e in seguito sviluppando il calendario) l'applicazione in modo da avere un riscontro passo-passo del funzionamento dell'applicazione. Per quanto riguarda il DVCS, abbiamo sempre sviluppato su branch diversi, per poi, a codice semi-funzionante, mergiare sul develop il quale fungeva da entry-point condiviso. Il develop è stato aperto subito dopo la creazione del repository e la configurazione di Eclipse e plugin annessi (FindBugs, PMD, CheckStyle, ecc... nonostante alcuni problemi successivamente risolti con PMD). Abbiamo infine compiuto merge sul branch master solo ed esclusivamente al completamento della nostra applicazione.

Cavallucci Martina (View): per la stesura di questa porzione di codice si è scelto di utilizzare la libreria grafica Swing anche se ho riscontrato alcuni limiti di tale libreria ad esempio sulla propensione ad adattarsi.
Ad ogni tab ho dedicato uno studio particolare del Layout e dei componenti che rendessero le informazioni più leggibili possibili.
Ho pensato di creare una classe di enumerazione apposta chiamata ViewColor per poter utilizzare gli stessi colori in ogni tab .
Per poter far selezionare all'utente una data da un calendario quando inserisce un evento ho inserito due librerie esterne chiamate swingx e JXDatePicker invece per selezionare un orario ho importato la libreria JTimeChoose. Esse mettono a disposizone componenti grafici validi e semplici da utilizzare grazie ai metodi forniti.
Ho deciso di creare una classe per ogni macro-oggetto grafico in modo da coordinare bene il rapporto con la parte del Controller; ogni classe all'interno del costruttore contiene il relativo controller al quale all'avvenimento di un evento richiama uno dei suoi metodi.
Per la realizzazione della classe dualListBox ho attinto al codice del seguente link: http://www.java2s.com/Code/Java/Swing-JFC/DualJListwithbuttonsinbetween.htm il quale fornisce metodi per la creazione e la gestione di due box dinamici.

Severi Claudia (Controller): ho fatto uso del pattern singleton per garantire

che una classe abbia un'unica istanza, accessibile globalmente e facilmente, senza dovermi preoccupare di fornirne il riferimento a chi lo richiede. Inoltre faccio uso di Enum allo scopo di controllare l'inserimento dei campi nelle classi Calendar, User e Contact e per coordinare il passaggio di parametri tra Controller e View.

Tonelli Alfredo (Model):
Ho utilizzato numerose volte le Lambda expression insieme agli Stream, principalmente nella classe CalendarImpl, per effettuare diversi tipi di richieste sulle collezioni (principalmente ricavare dati di interesse in relazione ai filtri inseriti) e operazioni su ciascun elemento come settare notificati tutti gli eventi inviati al controller per generare i promemoria.
Dato che uno dei requisiti dell'applicazione è la possibilità di poter salvare i dati inseriti dall'utente e ricaricarli al login, ho utilizzato la libreria Google Guava per servirmi di Optional che possano essere serializzati.
Ho utilizzato anche la classe Joda Time per avere accesso a funzioni avanzate per effettuare operazioni sulle date come il confronto per capire quale è maggiore di un'altra.
Per garantire la protezione dei dati ho deciso di creare una classe per la cifrare la password di accesso dell'utente tramite l'utilizzo di un algoritmo di hashing che restituisce il fingerprint della password che verrà salvato al posto di quest'ultima.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Cavallucci Martina: Sono molto soddisfatta del risultato finale in quanto utilizzando Swing, che fornisce funzionalità limitate, sono comunque riuscita a portare a termine un'interfaccia completa. Sicuramente, essendo la prima esperienza di progettazione a questi livelli, penso che i risultati potessero essere migliori, ma è un'esperienza che mi è molto servita per riuscire a familiarizzare con un futuro ambiente lavorativo. Non sono mancati problemi, in particolare con la coordinazione nel passaggio di parametri, ma dopo un'apposita fase di debug siamo giunti ad una soluzione. Sono complessivamente soddisfatta del rapporto con il team e del coordinamento, nonostante le modifiche richieste in corso d'opera. Sono molto soddisfatta anche dell'approccio step-by-step adottato che ci ha permesso di vedere l'applicazione crescere e prendere forma. Mi piacerebbe riprendere in mano questo progetto in un futuro, per rendere più reattiva l'applicazione, ed eventualmente gestire e ricevere informazioni anche da remoto.

Severi Claudia: A parer mio il progetto poteva essere strutturato meglio, sia a livello di intuitività delle azioni per l'utente, sia dal punto di vista dell'analisi, avendo avuto la necessità di modificare il codice in corso d'opera. Essendo il mio primo progetto ho riscontrato numerose difficoltà nella gestione del tempo, soprattutto nella strutturazione iniziale del programma, cercando di recuperare al meglio delle mie possibilità in corso d'opera. Lato positivo è che questo tipo di errori mi aiuteranno con lo sviluppo di progetti futuri. Non sono mancati i problemi tipici del Controller con il debug e il collegamento tra Model e View che spesso lavoravano con parametri molto diversi. Lato positivo è stata la coordinazione con gli altri membri del gruppo, entrambi molto disponibili e pazienti. Ho apprezzato l'ambiente lavorativo che si è creato, in termini di coordinazione e aiuto reciproco. Per quanto riguarda il progetto in esame, sarei interessata a continuarlo, rendendolo più dinamico.

Tonelli Alfredo: sono tutto sommato soddisfatto di come è venuta l'applicazione, certo è migliorabile, ma secondo me rappresenta comunque il frutto di un ottimo lavoro svolto da tutti i componenti del gruppo. Per quanto mi riguarda ho raggiun-

to i risultati che mi ero prefissato, come quello di fare il più possibile uso di lambda e stream per avere un codice più leggibile ed efficace.
Se ci fosse la possibilità sarei lieto di continuare lo sviluppo di questo progetto, secondo me rappresenta una buona base per questo tipo di applicazioni per la gestione degli impegni giornalieri degli utenti.
Sono soddisfatto delle conoscenze acquisite durante questa esperienza, ho fatto numerose ricerche nel materiale didattico del corso e su internet per poter chiarire ogni mio dubbio e per poter creare codice di buona qualità.

Appendice A

Guida utente

All'avvio dell'applicazione il programma richiede Username e Password, e mette a disposizione la possibilità di creare un utente (purché non siano campi vuoti e la password abbia almeno 6 caratteri). Una volta loggato l'utente potrà navigare nei vari tab. Nel calendario avrà la possibilità di visualizzare i vari mesi selezionando nelle apposite caselle di selezione l'anno e il mese desiderato. Potrà, facendo doppio click sul giorno desiderato, inserire un nuovo evento, o selezionare l'evento da eliminare.
Nella sezione dei contatti l'utente potrà inserire i dati del contatto e cliccare su "Insert" (facendo attenzione a inserire campi validi, avendo l'obbligo di inserire il nome, mentre per quanto riguarda il campo telefono deve contenere solo numeri e per la e-mail deve ci sarà il controllo se è presente il campo "@"). Per eliminare il contatto sarà sufficiente selezionarlo e cliccare su "Delete", mentre per la modifica sarà necessario selezionarlo e nelle box di inserimento inserire i dati corretti (se necessario riscrivendo quelli precedentemente corretti) e cliccare su "Modify". Per quanto riguarda la sezione to-do, si potranno inserire promemoria nell'apposita text-box e cliccare su "Insert", dopo di che per modificare lo stato ("Completed" o "Incompleted") si potrà selezionare, cliccare sulla checkbox e poi il bottone "Modify". Per l'eliminazione si potrà procedere in modo analogo ai contatti. Se mentre l'applicazione non era attiva si verificano degli eventi l'applicazione provvederà ad aggiornare l'utente con apposite finestre di dialogo. Inoltre l'utente avrà la possibilità con l'apposito bottone di cambiare la password ed eliminare il profilo, perdendo permanente i dati salvati. Sarà necessario per salvare permanentemente le modifiche cliccare su logout, tasto che provvederà anche a sloggare l'utente e uscire dall'applicazione, mentre cliccando sulla 'X' chiuderà senza salvare, per questa ragione mostrerà un messaggio per chiedere all'utente se è sicuro.