

RELAZIONE PER IL PROGETTO “Info-Mng”

Alessandro Montalti

Mattia Berretti

Juan Goytia

31 maggio 2016

Indice

1 Analisi	
1.1Requisiti.....	
1.2Analisi e modello del dominio.....	
2 Design	
2.1 Architettura.....	
2.2 Design dettagliato.....	
3 Sviluppo	
3.1 Testing automatizzato.....	
3.2 Metodologia di lavoro.....	
4 Commenti finali	
4.1 Autovalutazione e lavori futuri.....	
A Guida utente	

Capitolo 1

Analisi

1.1 Requisiti

Il software mira a gestire la parte economica di un negozio di informatica, permettendo dunque di rintracciare ogni tipo di informazione riguardante le operazioni di acquisto o di vendita della merce

Requisiti concordati:

- Gestione clienti/fornitori
- Gestione fatture acquisto
- Gestione fatture vendita
- Gestione scontrini fiscali
- Gestione magazzino
- Preventivi vendita
- Registro IVA acquisti
- Registro IVA corrispettivi

1.2 Analisi e modello del dominio

Il programma deve essere in grado di gestire i dati inseriti e adattarli alle varie situazioni senza ulteriore sforzo da parte dell'utente. Per fare ciò si introducono le entità Fatture, Movimenti, Clienti, Fornitori e Scontrini.

Le relazioni tra le varie entità individuate per la risoluzione del problema sono rappresentate in Figura 1.1.

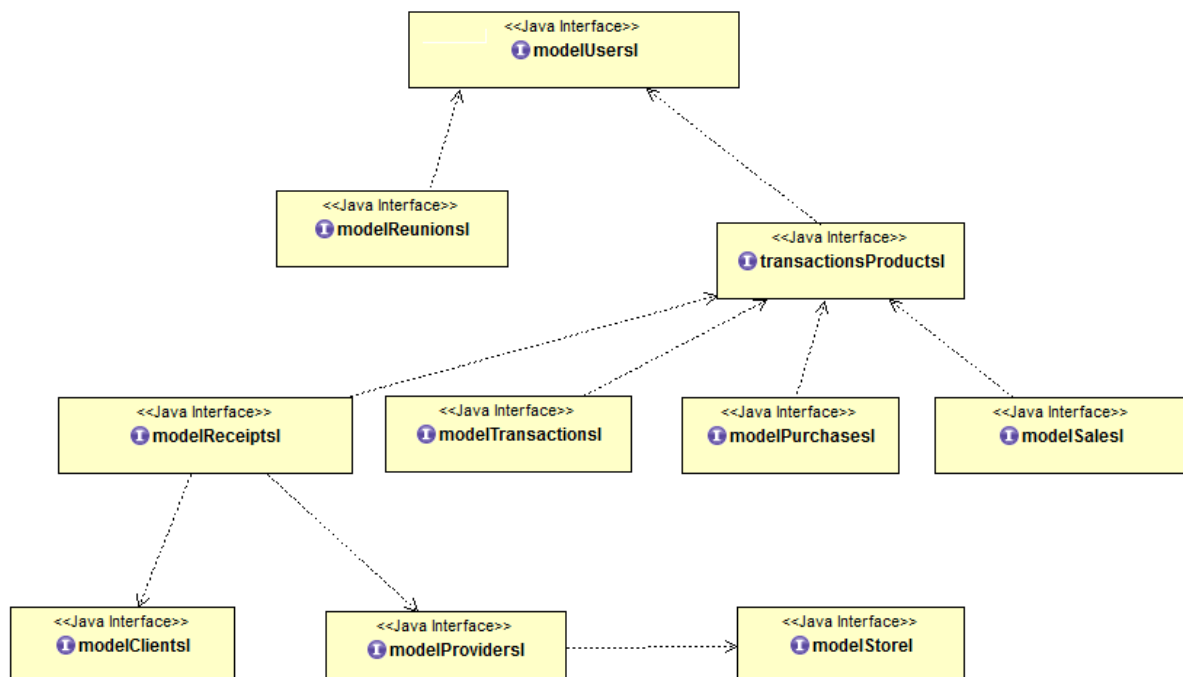


Figura 1.1: Schema UML che rappresenta le entità individuate per la risoluzione del problema

Capitolo 2

Design

2.1 Architettura

Per questo progetto si è scelto di utilizzare il pattern architetturale Model-View-Controller così suddiviso:

- Model: Implementa il dominio del modello studiato per risolvere il problema, e quindi si occupa di implementare tutte le entità precedentemente individuate e la comunicazione tra esse.
- View: si occupa di mostrare graficamente all'utente le operazioni che possono essere eseguite per la gestione della parte economica di un negozio e di farlo interagire con esse alla creazione di un database per poi interagire con esso attraverso le varie view implementate.
- Controller: si occupa di gestire e stabilire la corretta connessione tra model e view oltre ad occuparsi del caricamento e del salvataggio del database.

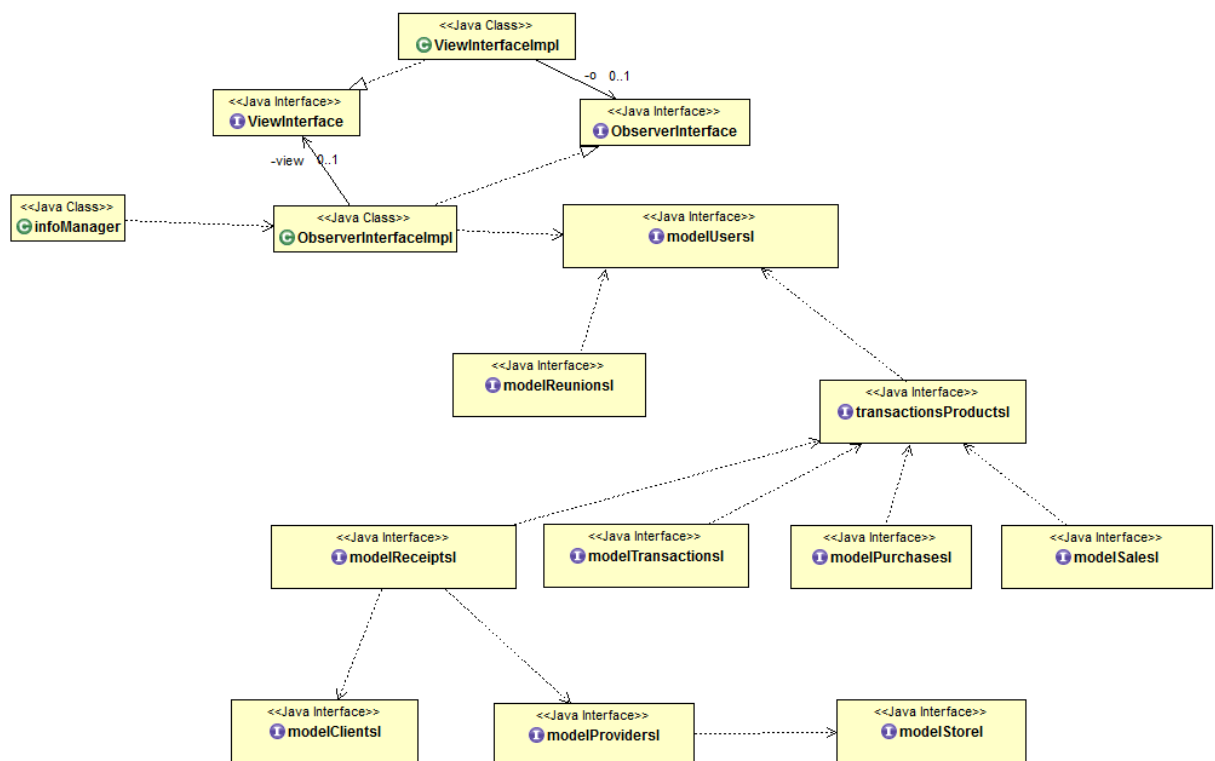


Figura schema UML del modello MVC.

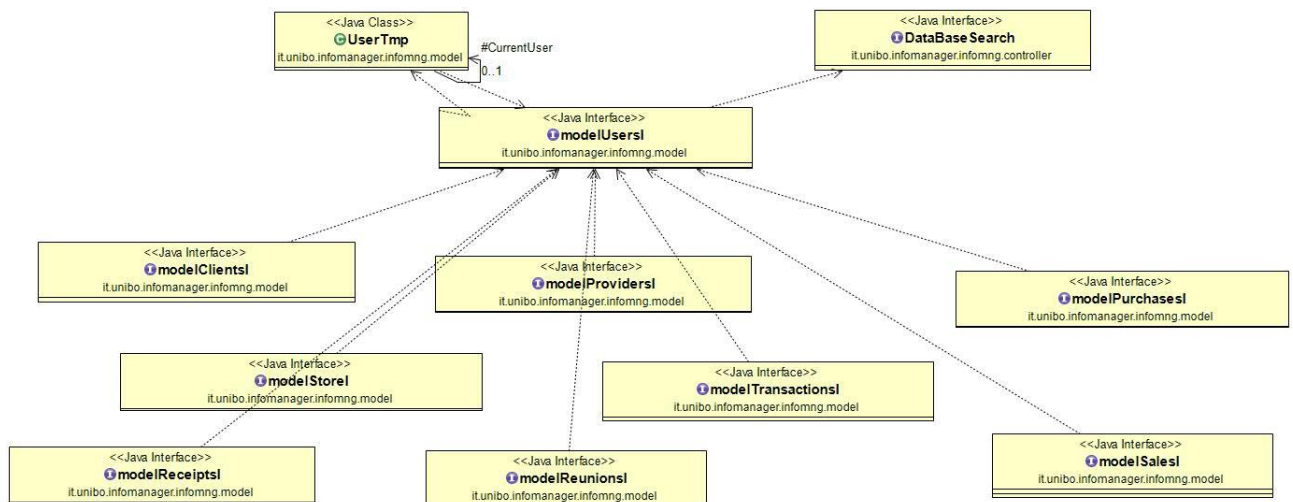
2.2 Design dettagliato

Il progetto è stato suddiviso in package per suddividere logicamente le varie parti dello stesso. La radice comune che è stata data è `it.unibo.infomanager.info-mng.`, da qui si suddividono in `infoMng` nel quale risiede la classe con il main che si occupa di far partire l'applicazione, `model.DB` dove risiedono le classi e le interfacce del model, `controller` e `controller.delegate` dove risiedono le classi e le interfacce del model, `view.classes`, `view.dialog`, `view.tabelle` e `view.toolbar` dove risiedono le classi e le interfacce della view e `view.icons` dove risiedono le icone personalizzate del menu.

2.2.1 Model

In questa parte di progetto sono implementate le entità citate prima in fase di analisi che servono a risolvere il problema, facendo attenzione al fatto che più classi devono avere la possibilità di poter tracciare ogni entità collegata ad essa. La scrittura e lettura, così come le eventuali modifiche e salvataggio dei dati e soprattutto l'accesso degli utenti, perché sono loro a dover realizzare tutti i movimenti descritti prima, dovevano essere tutti modellati con delle entità prima di essere salvati nel data base.

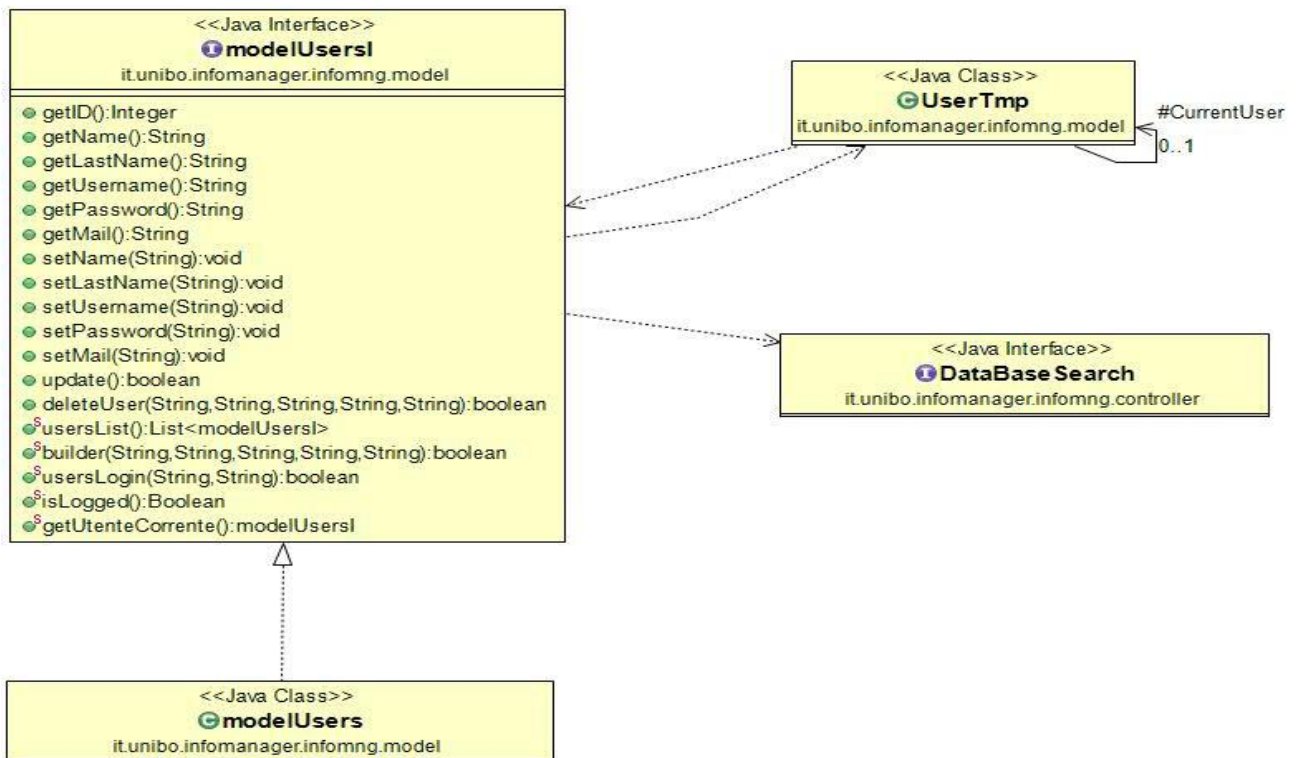
Un esempio molto semplificato potrebbe essere l'accesso degli utenti al nostro gestionale e tutte le entità collegate a quest'ultimo tramite le interfacce:



Si può notare infatti che tutte le entità, per essere maneggiate, dipendono dal utente perché è lui il responsabile di una qualunque azione nel data base e tramite il metodo `isLogged` si controlla, ad ogni modifica o creazione, se l'utente si è loggato prima di effettuare alcuna azione. Infine anche gli utenti prima di poter accedere devono confermare le sue credenziali (forniti dalla view tramite il controller) nel data base.

Andando nel dettaglio, dopo essere stato ideato l'entità per gli utenti e a conoscenza di tutte le richieste forniti dal controller per il salvataggio dei dati, procederò ad elencare ogni entità creata:

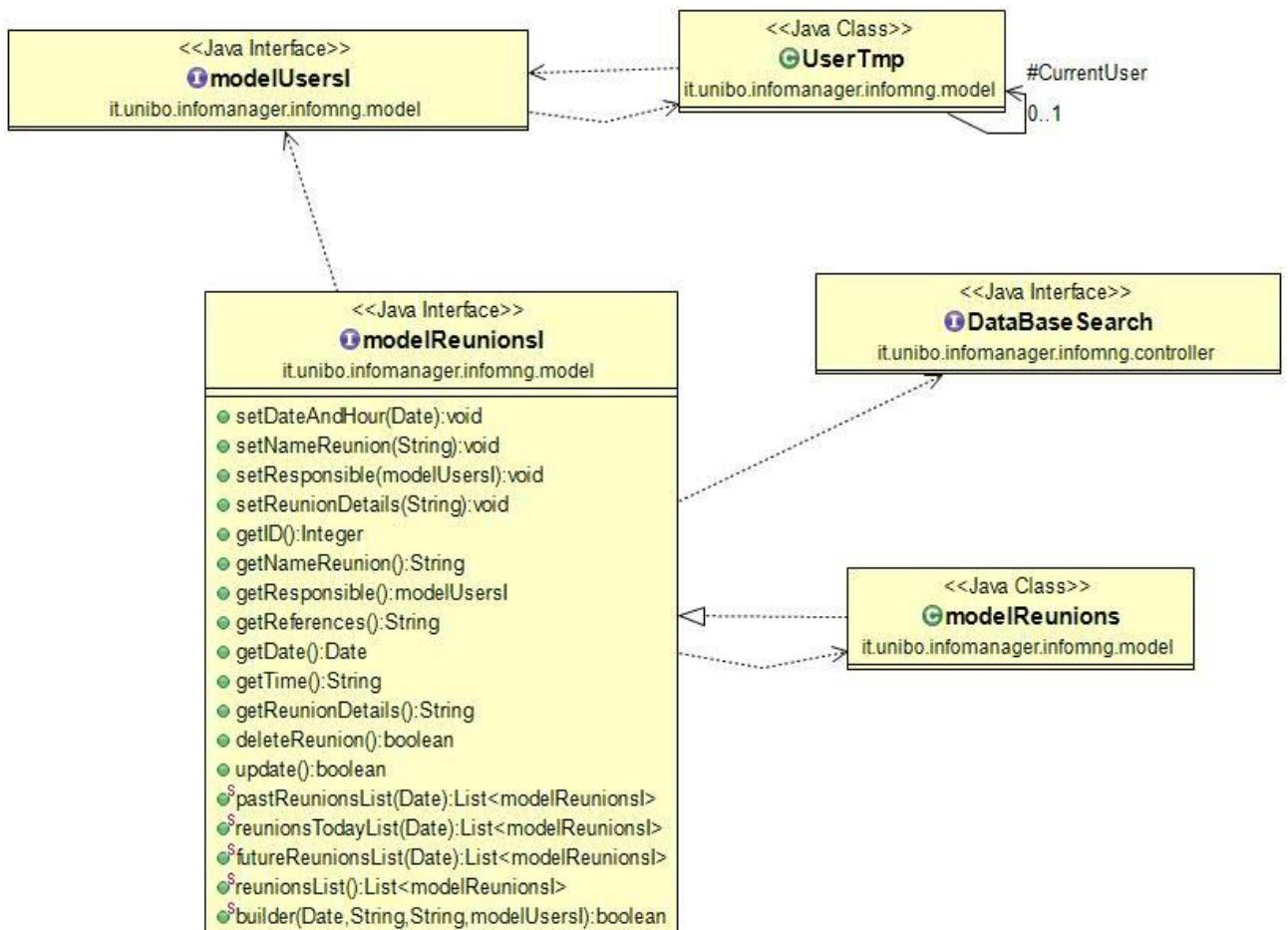
Per gli utenti:



Per gli utenti è stato creato la classe `modelUsers` che gestisce/crea l'entità "Utenti" e la sua rispettiva interfaccia ha dipendenze con la classe `UserTmp` che tiene traccia dell'utente loggato finché non si chiude l'applicazione o si effettua il logout. Si può apprezzare che ha anche una dipendenza con l'interfaccia `DataBaseSearch` creata dal controller tramite la quale si realizza il salvaggio/modifica/eliminazione/ dei dati direttamente nel DataBase. Vale a dire che quest'ultima dipendenza sarà presente in tutte le interfacce create da me.

Per le riunioni:

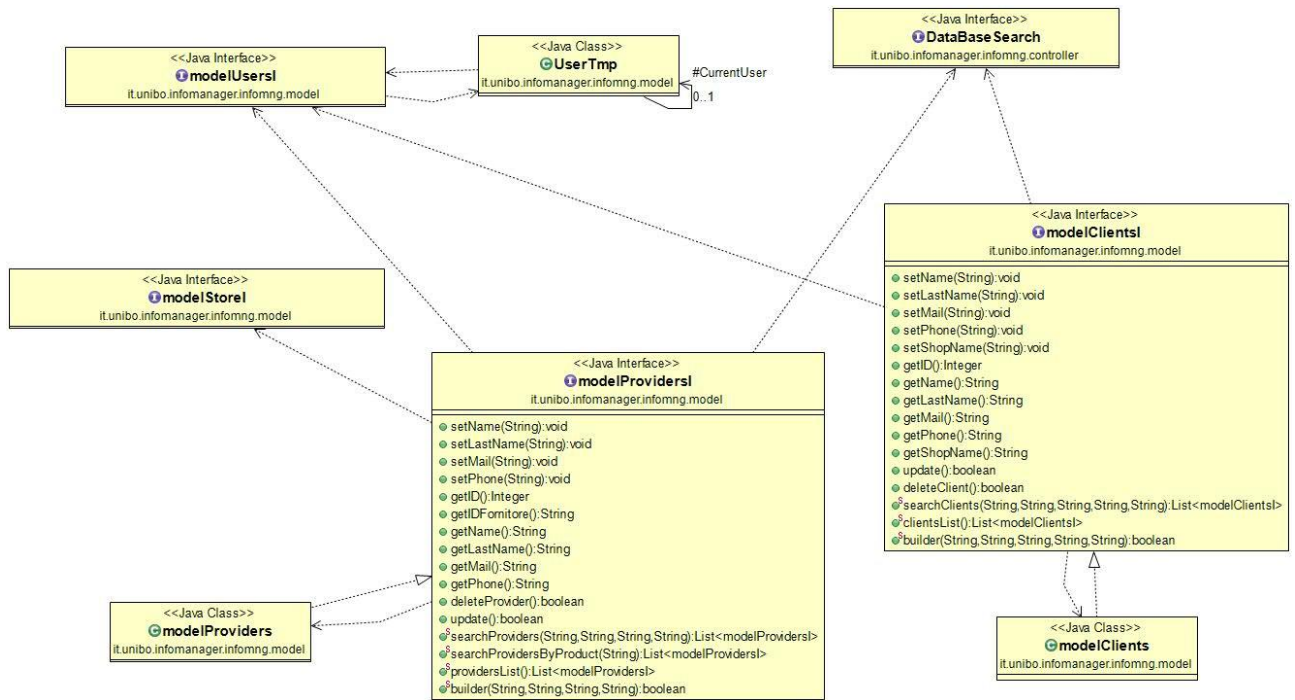
È stato creato la classe `modelRiunions` che al suo interno gestisce/crea l'entità "Riunioni" e la sua rispettiva interfaccia gode delle dipendenze basi spiegate nel paragrafo precedente (per gli utenti).



Per i clienti e fornitori:

Per i clienti è stata creata la classe `modelClients` che al suo interno gestisce/crea l'entità "Clienti" con le possibilità di realizzare modifiche/eliminazione/salvataggio di un determinato cliente. Per i fornitori invece è stato creato la classe `modelProviders` che conta con gli stessi metodi creati nella classe per i clienti soltanto che si lavora in un'altra entità chiamata "Fornitori".

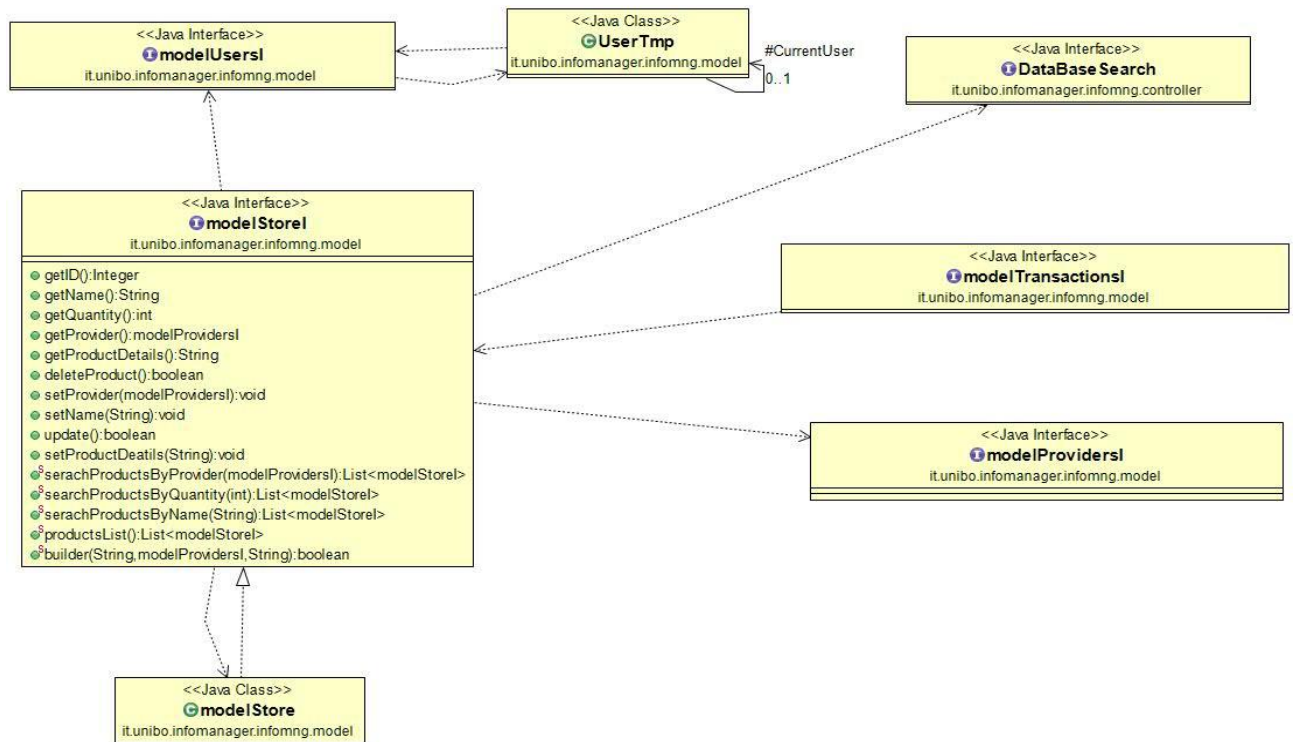
Per quanto riguarda le dipendenze si può apprezzare che l'interfaccia `modelProvidersI` a differenza del `modelClientsI` ha una dipendenza con il magazzino (`modelStoreI`), ideato per una eventuale ricerca di fornitori tramite il nome di un prodotto (`searchProvidersByProduct`).



Per il magazzino:

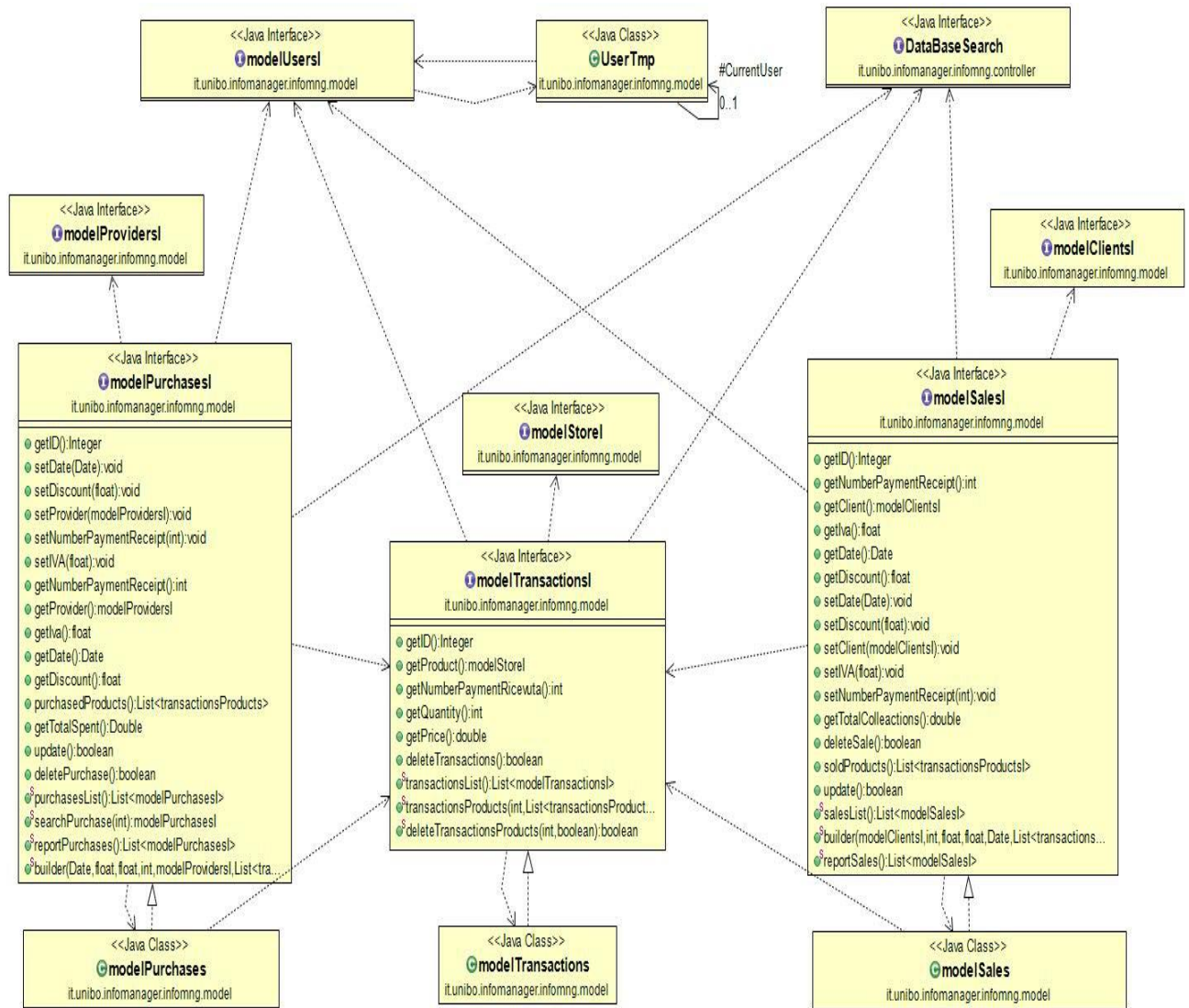
Per il magazzino è stato creato l'entità Magazzino dentro la classe "modelStore" e come si può notare nel diagramma, la sua interfaccia ha dipendenze sia con le solite modelUsersI e DataBaseSearch che con i fornitori(modelProvidersI) e le operazioni con i prodotti(modelTransactionsI).

Con i fornitori il motivo è nato dal bisogno di implementare una ricerca di prodotti tramite un determinato fornitori. Mentre che la relazione con l'interfaccia "modelTransactionsProducts" è nata con la necessità di aggiornare ad ogni chiamata del metodo getQuantity() di ogni prodotto la disponibilità di quest'ultimo.



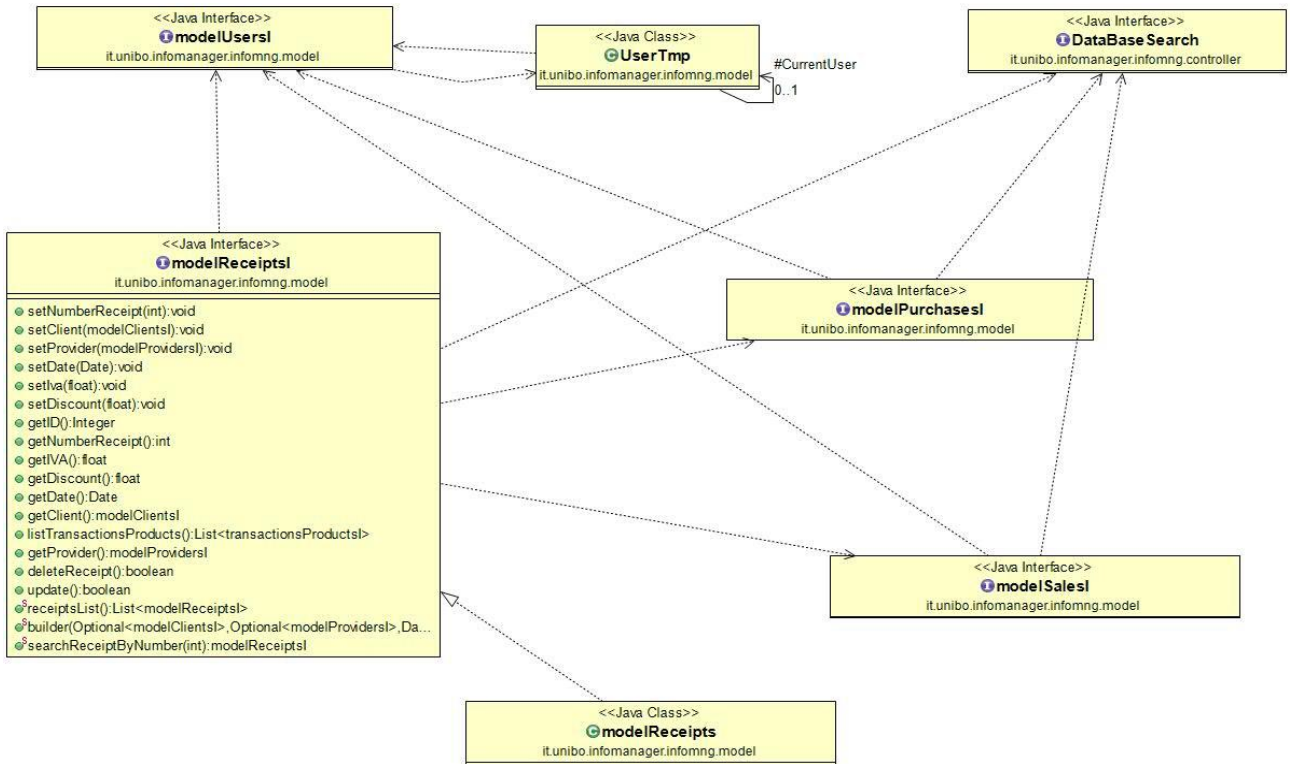
Per le vendite e acquisti:

Per poter modellare i dati degli acquisti esiste la classe modelPurchases(che al suo interno gestisce/crea l'entità Acquisti). Per le vendite invece esiste la classe modelSales(che al suo interno gestisce/crea l'entità Vendite). In fine ho dovute creare oltre alle classi precedentemente menzionate anche la classe modelTransactionsProducts, quest'ultima incaricata di salvare ogni singolo prodotto e la sua rispettiva quantità, venduta o acquistata, in una entità del data base chiamata "Movimenti". A differenza di modelTransactionsProducts, le entità per le vendite(Vendite) e gli acquisti(Acquisti) non tengono traccia dei prodotti venduti, e per poter arrivare ai singoli prodotti bisogna passare dall'entità Movimenti tramite appositi metodi creati dentro modelSales e modelPurchases.

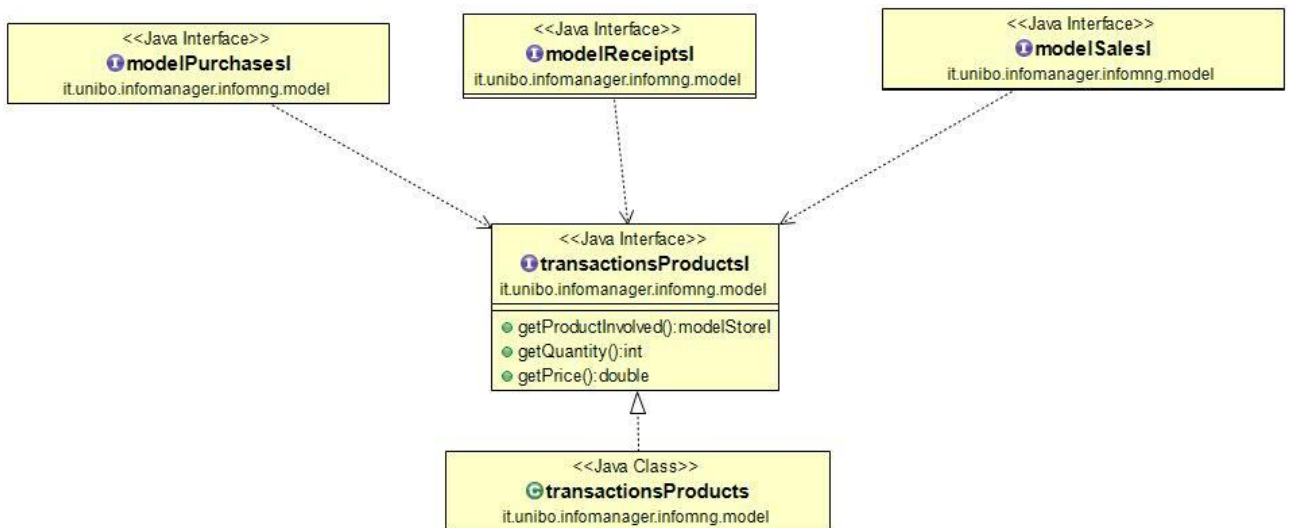


Per gli scontrini:

E stata creata la classe modelReceipts che al suo interno gestisce/crea l'entità "Scontrini". Gli scontrini che saranno creati sia per le vendite che per gli acquisti, non avranno necessita di tenere traccia dei prodotti venduti o acquistati, questo perché ci si potrà arrivare a loro tramite le interfacce modelPurchasesI e modelSalesI.



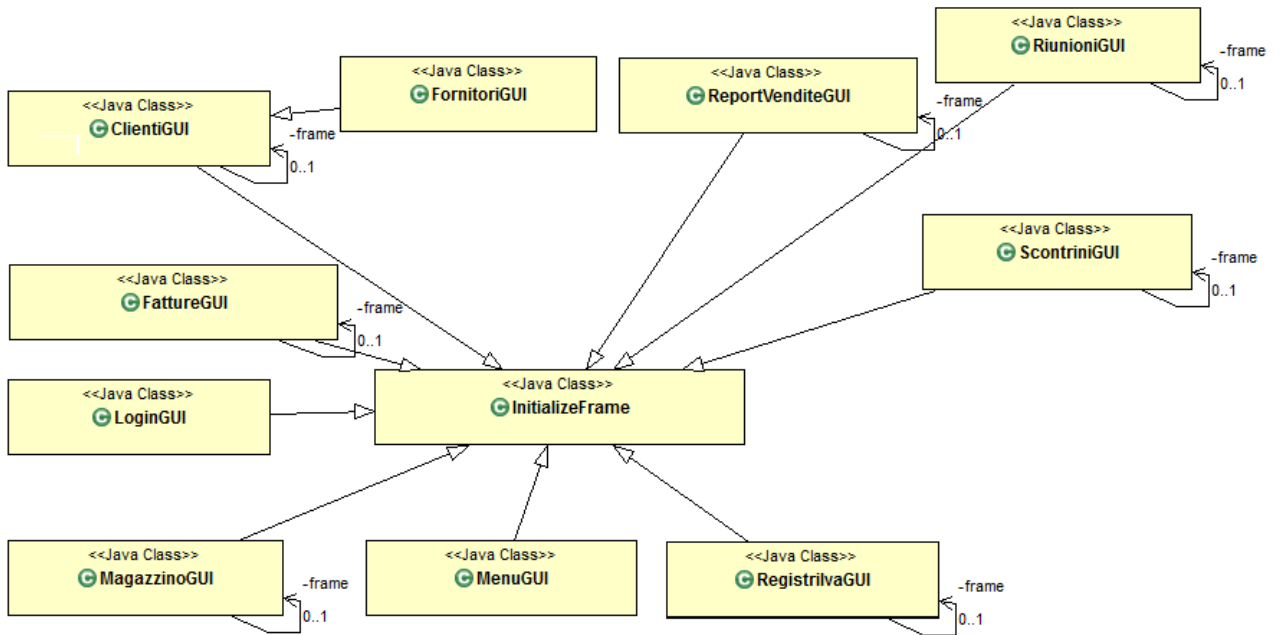
Infine è stata creata una classe ausiliare transactionsProducts che serve come tipo di dato che contiene sia un Prodotto, quantità, e prezzoUnitario; utili se si vuole effettuare una vendita o acquisto.



2.2.2 View

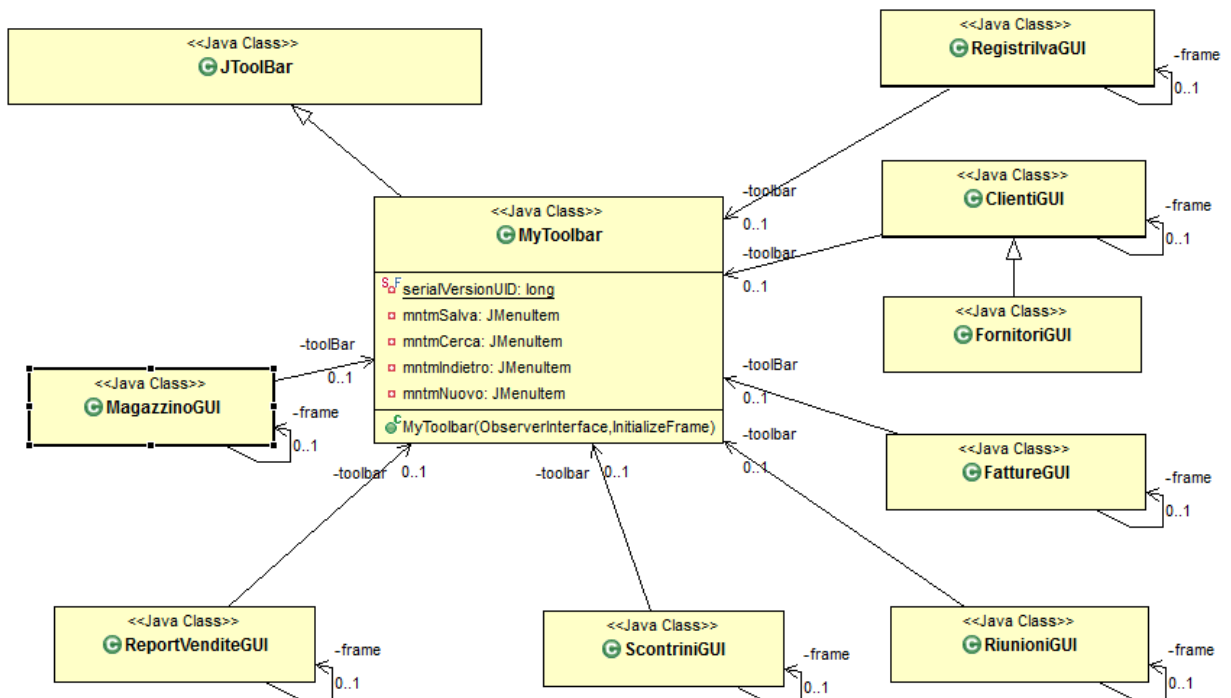
La view attraverso varie interfacce grafiche garantisce l'utilizzo del programma di gestione della parte economica di un negozio di informatica in maniera piuttosto semplice e intuitiva. Lo sviluppo della view è partito dalla creazione del login e del menu fino alle view che mostrano le funzionalità del programma lanciate dal menu. La comunicazione con il model avviene solo ed esclusivamente attraverso il controller. Per semplificare la modifica generica dei JFrame (azione alla pressione del bottone di uscita e rendere i frame ridimensionabili) delle varie view ho creato una classe che eredita JFrame e che a sua volta viene

ereditata da tutte le view del progetto le quali specificano solo la dimensione del frame, il titolo e il layout del primo JPanel inserito.

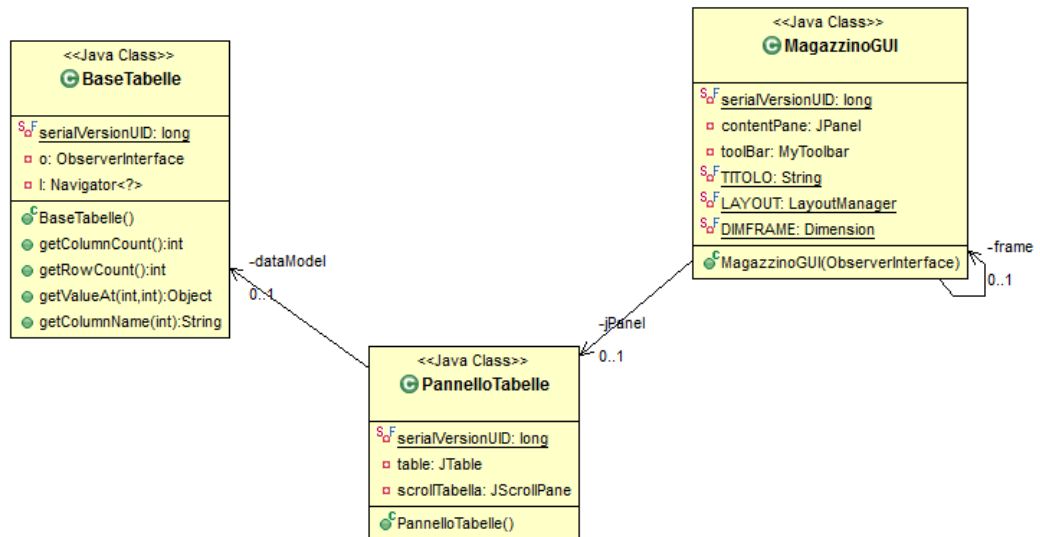


Schema UML del Frame generico ereditato da tutte le altre view.

Ho creato una classe MyToolBar che viene creata in tutte le view e gestisce le operazioni di salvataggio, ricerca e creazione di nuove informazioni. Di seguito riportato lo schema UML della classe MyToolBar.



In fine ho creato un jTable utilizzata per il Magazzino la quale viene costruita tramite un modello astratto che mi permette di editare la tabella. Di seguito riportato lo schema UML usato per la tabella nella classe MagazzinoGUI.



La comunicazione view controller avviene attraverso le interfacce `ViewInterface` (la quale è creata e implementata da me) e `ObserverInterface` (la quale è definita da me e implementata dal controller). La prima mette a disposizione i metodi necessari all'apertura delle varie view su richiesta, la seconda mette a disposizione metodi necessari per eseguire le richieste fatte da me.

2.2.3 Controller

In questa parte del progetto l'obiettivo era quello di creare un database e permettere di stabilire la corretta connessione tra le varie view, quello che l'utente vede a schermo, e il codice scritto dal Model, ciò che realmente rende possibile la produzione di dati a video e il corretto funzionamento dell'programma. per gestire l'input e l'output ho creato 3 classi private e 2 interfacce

classi

- Query
- Oggetto
- Database

Interfacce

- TableRow
- DataBaseSearch

Le interfacce contengono dei costruttori statici per le classi.

DataBase

La classe DataBase si occupa di gestire il database SQLite a livello di codice SQL andando a scansionare la struttura del DB aggiornandone la struttura aggiungendo tabelle o colonne.
Oltre a questo si occupa di gestire i dati aggiungendo record o aggiornandoli.
La classe è in grado di gestire i tipi di dato (String, Date, Boolean, Double, Float)

TableRow (Oggetto)

L'interfaccia TableRow si occupa di gestire una tupla di una tabella tenendo in memoria la chiave primaria che è sempre un intero che è generato dal database, la data di creazione e di ultima modifica, ed una mappa di Stringhe e Object che rappresenta i nomi delle colonne con i relativi dati.

È possibile ottenere le TableRow mediante l'oggetto DataBaseSearch oppure mediante il costruttore statico di TableRow, la differenza risiede nel fatto che tramite costruttore statico si crea una nuova tupla che al momento del salvataggio sarà inserita nell'apposita tabella, mentre tramite la classe di ricerca si ottengono TableRow già esistenti all'interno del database.

DataBaseSearch(Query)

L'interfaccia DataBaseSearch si occupa di creare una query di ricerca da passare al database e di restituire quindi una lista di TableRow.

Questa classe permette anche di utilizzare alcuni settari di ricerca direttamente tramite il codice SQL per esempio diverse condizioni di uguaglianza oppure di esistenza o meno di un campo.

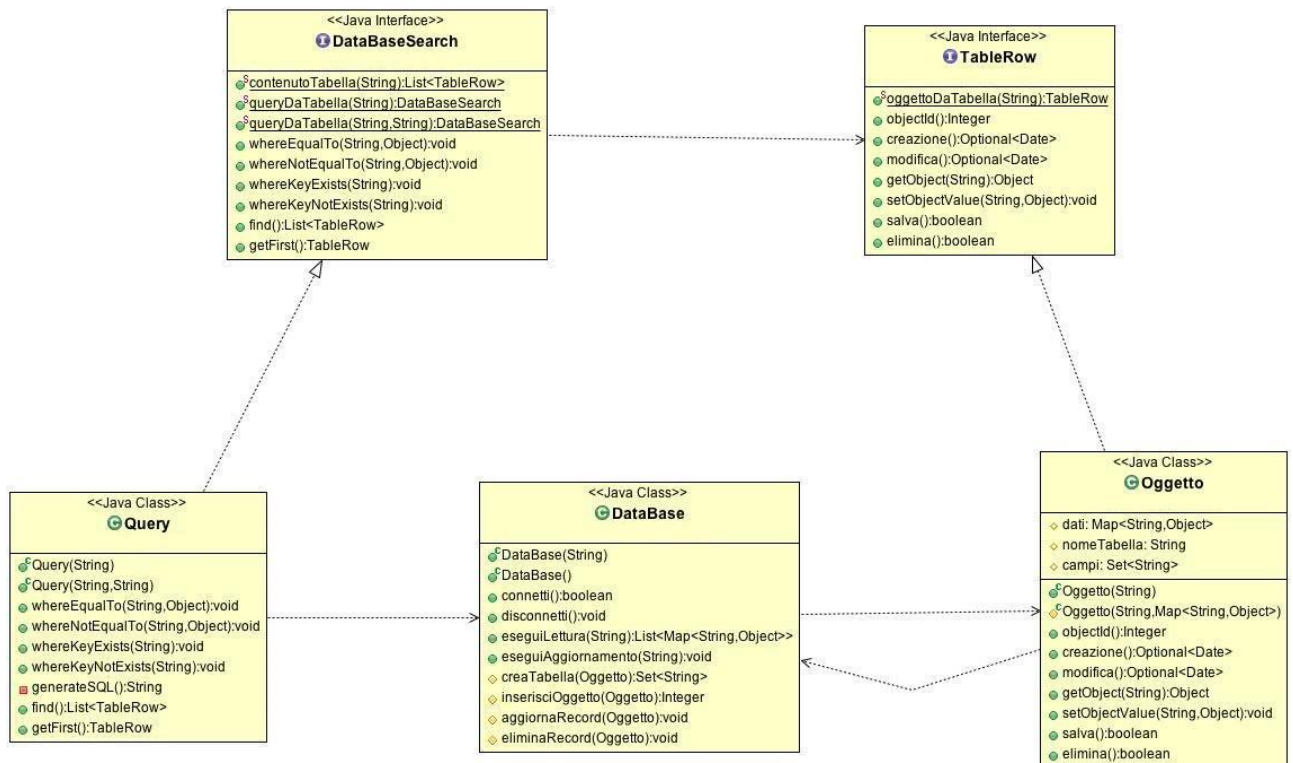
È possibile inoltre aggiungere un codice SQL per la ricerca.

Il numero di condizioni che si possono aggiungere ad una ricerca non ha limite ma esse vengono lette tramite l'operatore logico AND tra di loro

N.B.

Sia per TableRow e DataBaseSearch la tabella da utilizzare è indicata mediante costruttore.

UML funzionamento DataBase



Capitolo 3

Design

3.1 Testing automatizzato

Per controllare il funzionamento, soprattutto degli stream usati nell'implementazione di diversi metodi dentro le classi del model, è stato utilizzato il sistema I/O di java in contemporaneo con il console in modo tale da controllare l'efficacia e le eccezioni a run time che potevano generarsi nell'applicazione.

Mentre per controllare il funzionamento dell'interfaccia grafica si è creato un database a mano e testata ogni sua funzione dalla creazione di nuove fatture alla visualizzazione di fatture già esistenti controllando che ogni evento che l'utente può lanciare venga gestito.

3.2 Metodologia di lavoro

Il software è stato sviluppato in questo modo:

Model: Juan Goytia

Controller: Mattia Berretti

View: Alessandro Montalti

La progettazione è iniziata con ognuno che progettava per conto suo, ma ci siamo subito resi conto che era alquanto inutile perciò abbiamo optato per definire le principali entità che ognuno di noi doveva scrivere. Poi ognuno ha iniziato la sua parte cercando di essere il più autonomo possibile. La suddivisione del lavoro è stata abbastanza equa e l'utilizzo dell'architettura MVC ha reso minime le dipendenze tra i membri del gruppo. In fase di sviluppo si è utilizzato Mercurial con BitBucket come DVCS per garantire che ogni parte avesse a disposizione istantaneamente le modifiche effettuare da un membro. Questo è stato particolarmente utile sia nella parte iniziale del progetto sia alla fine di esso in quanto si potevano eseguire correzioni di bug, soprattutto nella parte finale. Alla fine quando abbiamo avuto una prima versione funzionante ci siamo incontrati per discutere di migliorie o bug. La parte più difficile è stata la mancanza di esperienza nel lavoro di gruppo che però con il tempo è andata affievolendosi.

3.3 Note di sviluppo

Nella realizzazione dell'interfaccia grafica è stato utilizzato WindowBuilder il quale ha facilitato parecchio il lavoro della view. Mentre nella parte del controller per scrivere e leggere è stata utilizzata la libreria esterna sqlite-jdbc che è stata inclusa nel classpath per rendere il software portabile.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Il gruppo è soddisfatto del progetto in quanto è riuscito a implementare le funzionalità base che si era proposto anche se non siamo riusciti a implementare per tempo le funzionalità future. Considerando che è il primo progetto svolto in gruppo con un diverso approccio in quanto si deve rispondere anche ad altri sviluppatori il gruppo pensa di aver collaborato bene. La parte che un po' è mancata nel gruppo è stata sicuramente l'organizzazione e la lentezza nel caricare sul repository.

Alessandro Montalti: Mi ritengo soddisfatto del codice che ho scritto anche se mi sono accorto che avrei potuto ottimizzare molto meglio alcune sue parti. Ho cercato di aiutare, dove mi era possibile, il gruppo e lo stesso è stato fatto con me, perciò penso anche che come gruppo non sia andata tanto male. La parte che mi ha richiesto più tempo è stata la gestione della toolbar, i suoi bottoni e i dialog collegati alla toolbar stessa. Il mio ruolo nel gruppo è stato quello di definire l'interfaccia grafica del progetto.

Mattia Berretti: Inizialmente ho avuto dei problemi nel comprendere la differenza tra controller e model e il motivo della loro divisione.

Ancora ora non comprendo molto a pieno la necessità del passaggio tra view e model.

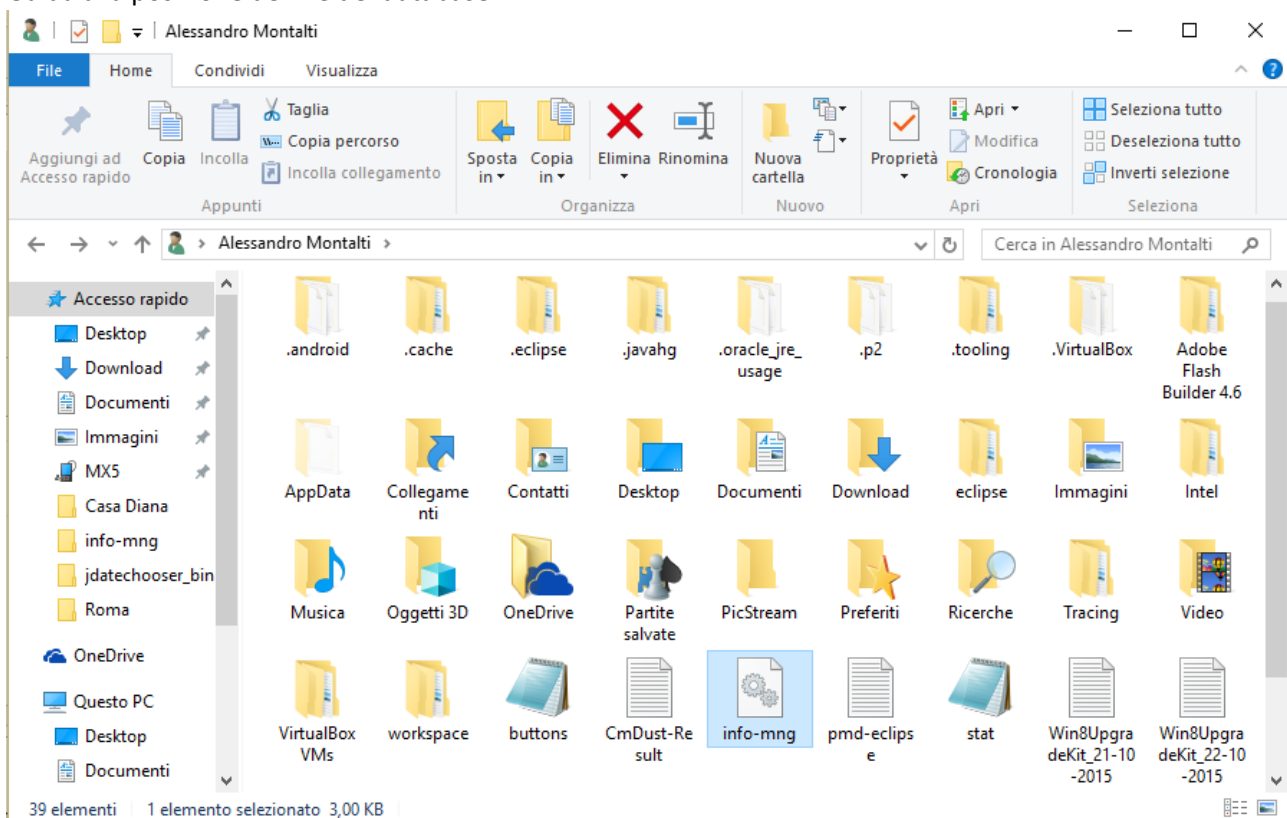
Tutto sommato è stato interessante vedere come mi trovo a lavorare in gruppo non ostante alcune incomprensioni di livello logico della costruzione delle classi del model, e sul modo di utilizzo delle mie per la gestione di input e output.

Juan Goytia: Ritengo questo progetto come una esperienza unica e molto vantaggiosa per la mia carriera professionale. Infatti non avevo mai lavorato autonomamente svolgendo il mio lavoro di model e contemporaneamente con altre persone per un unico progetto. Il fatto di interagire con i miei compagni valutando se un piccolo cambiamento da parte da uno o dell'altro potesse cambiare il lavoro dei restanti forse è stata la esperienza più significativa. In linee generali nonostante tutti i contrattempi nello sviluppo e anche se sono al corrente che posso sviluppare molto meglio la mia parte, considero l'esperienza molto positiva.

APPENDICE A

Guida Utente

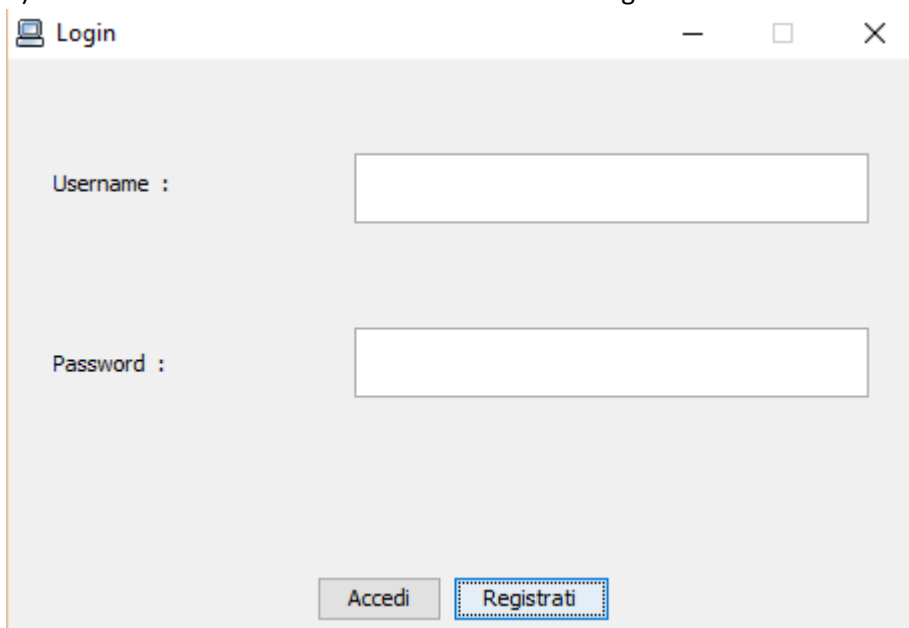
Guida alla posizione del file del database:



Il file si trova nel percorso "C://User/nomeutente" con il nome info-mng come evidenziato in figura

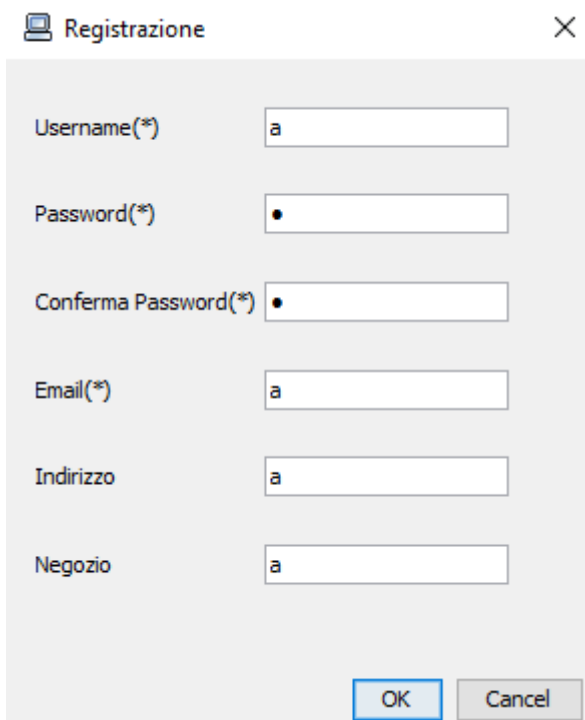
Guida alla creazione di un nuovo utente:

1) Per creare un nuovo utente selezionare la voce registrati.



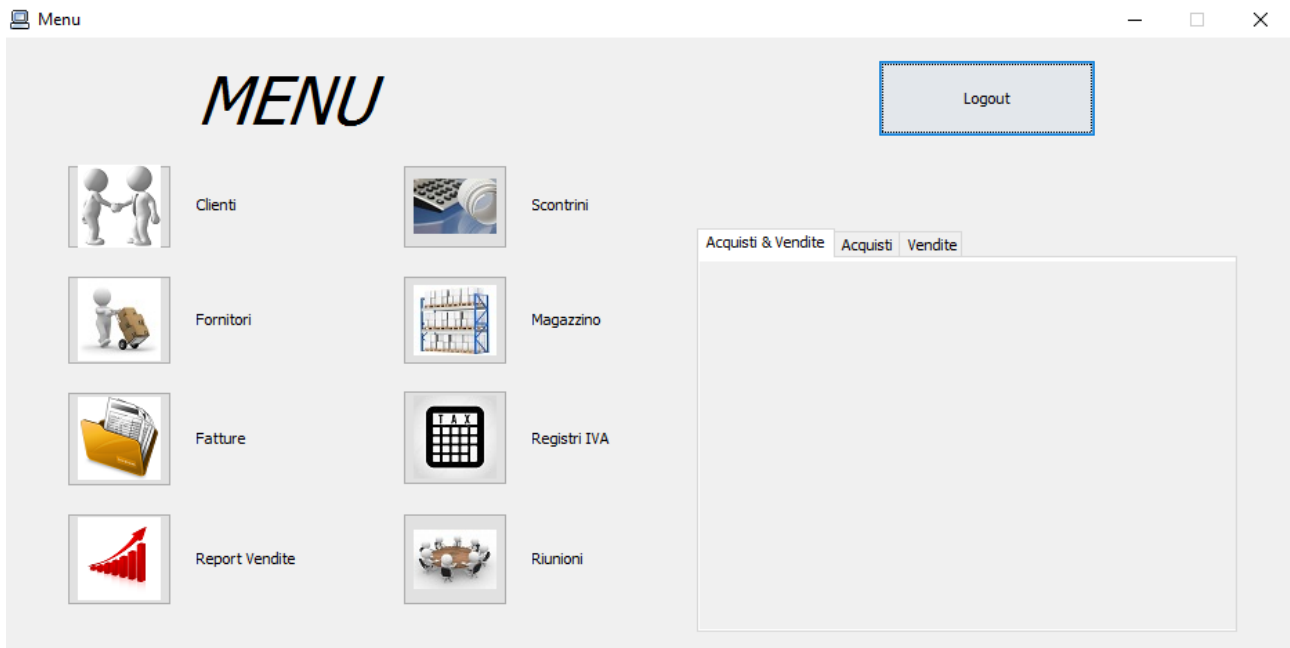
The image shows a 'Login' dialog box with a title bar containing a computer icon, the text 'Login', and standard window controls (minimize, maximize, close). The dialog contains two input fields: 'Username :' and 'Password :'. At the bottom, there are two buttons: 'Accedi' and 'Registrati'. The 'Registrati' button is highlighted with a blue dashed border.

2) Inserire i campi stando attenti ad inserire i campi obbligatori contrassegnati da (*).



The image shows a 'Registrazione' dialog box with a title bar containing a computer icon, the text 'Registrazione', and a close button. The dialog contains six input fields: 'Username(*)', 'Password(*)', 'Conferma Password(*)', 'Email(*)', 'Indirizzo', and 'Negozio'. The 'Username(*)', 'Email(*)', 'Indirizzo', and 'Negozio' fields contain the letter 'a'. The 'Password(*)' and 'Conferma Password(*)' fields contain a black dot. At the bottom, there are two buttons: 'OK' and 'Cancel'.

3) Dare l'ok nella schermata precedente effettuerà automaticamente il login e farà comparire il menu.



4) Ora si può procedere al uso delle funzionalità del programma.