

RELAZIONE PROGETTO

“Supermarket Billing System”

Briccoli Sara

Programmazione Orientata agli Oggetti
2021/2022

Indice

| | |
|--|-----------|
| Analisi | 5 |
| 1.1 Requisiti | 5 |
| 1.1.1 Requisiti funzionali | 5 |
| 1.1.2 Requisiti non funzionali | 6 |
| 1.2 Analisi e modello del dominio | 6 |
| Design | 8 |
| 1.1 Architettura | 8 |
| 2.2 Design dettagliato | 9 |
| 2.2.1 Lettura e Scrittura dei dati | 9 |
| 2.2.2 Account e Login | 10 |
| 2.2.3 Gestione Prodotto | 12 |
| 2.2.4 Reports | 14 |
| Sviluppo | 16 |
| 3.1 Testing | 16 |
| 3.2 Metodologia di lavoro | 16 |
| 3.3 Note di sviluppo | 16 |
| Commento finale | 18 |
| 4.1 Autovalutazione e lavori futuri | 18 |
| 4.2 Difficoltà incontrate e commenti per docenti | 18 |
| Guida Utente | 20 |
| A.1 Avvio dell'applicazione | 20 |
| A.2 Log In | 21 |
| A.3 CashierView | 22 |
| A.4 ManagmentView | 24 |

Capitolo 1

Analisi

1.1 Requisiti

Il singolo si pone come obiettivo quello di creare un'applicazione mirata alla gestione di un generico supermercato.

Il sistema permetterà di effettuare tutte le operazioni necessarie per monitorare ed organizzare il corretto svolgimento dell'attività. In particolare, l'utilizzo di questo software è stato pensato per due diversi contesti di impiego: amministrazione e personale alle casse, ognuno legato a proprie specifiche e permessi.

1.1.1 Requisiti funzionali

- L'applicazione all'avvio dovrà gestire l'autenticazione dell'utente, in base alle varie tipologie: amministratore, cassiere e cliente. Quest'ultimo non avrà i permessi di accedere.
- Il sistema permetterà di accedere ed effettuare operazioni tramite un apposito menu principale.
- L'amministratore si occuperà della gestione dei prodotti, gestione del personale, gestione della clientela.
- L'amministratore potrà inoltre visionare i report sugli acquisti e sui prodotti in magazzino.
- Il cassiere si occuperà della gestione del carrello, con possibilità di ricerca dell'utente tramite userId.
- L'amministratore potrà gestire gli account di tutti i tipi, attraverso la registrazione di nuovi utenti, la ricerca tramite userId e l'eliminazione di utenti presenti nel database.
- L'amministratore potrà inoltre aggiornare i dati degli utenti già registrati.
- Sarà resa possibile la registrazione, da parte dell'amministratore, di nuovi prodotti, inserendo le relative specifiche, quali: barcode, nome del prodotto, categoria, numero in magazzino e prezzo unitario.
- L'amministratore avrà la possibilità di ricercare, aggiornare ed eliminare prodotti esistenti all'interno del database.

- Il checkout alla cassa verrà registrato e riportato sul report acquisti, nel quale verranno mostrati dati quali: data acquisto, customer id e name, prezzo della spesa.
- Al momento del checkout verrà ricalcolato, per ogni prodotto venduto, il numero di rimanenze in magazzino.

1.1.2 Requisiti non funzionali

- L'interfaccia deve risultare semplice, intuitiva e veloce.
- Il gestionale dovrà essere multi-piattaforma: Windows, Linux and MacOS.

1.2 Analisi e modello del dominio

L'applicazione Supermarket Billing System si occuperà della gestione e fatturazione di un supermercato, in particolare gestirà entità quali prodotti, utenti e acquisti degli utenti.

Il sistema semplificherà la giornata dei dipendenti, in particolare degli amministratori, grazie alle funzioni per la gestione e controllo dei prodotti e tramite i report presenti.

Già nella prima versione del software per ogni acquisto da parte di un utente verranno ricalcolate le rimanenze dei prodotti.

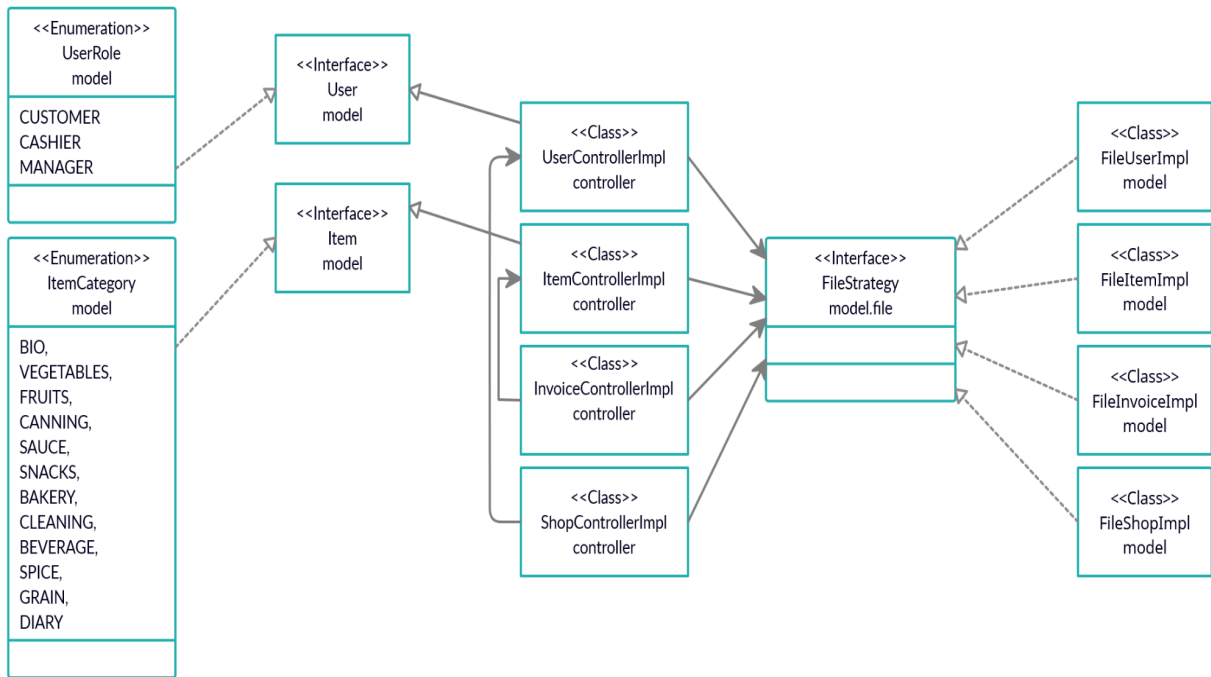


Figura 1.2.1: Architettura generale

Capitolo 2

Design

1.1 Architettura

Il singolo ha sviluppato il progetto seguendo il pattern architetturale MVC, nel quale il Controller fa da intermediario tra il Model e la View.

Il Model contiene i dati e gli algoritmi per la gestione delle entità interne dell'applicazione.

La View rappresenta graficamente il contenuto del Model specificando come gli oggetti contenuti in esso debbano venire presentati all'utente.

Il Controller trasforma gli input dell'utente provenienti dalla View in elementi processabili dal Model, il quale li elabora e restituisce al Controller per essere mostrati nella View.

Si è cercato di far sì che la View contenesse meno logica possibile demandando al Controller. Allo stesso modo, il Model non ha alcun collegamento con la View ma interagisce unicamente con il Controller.

La divisione nei package permette di far sì che i soli metodi dichiarati pubblici siano visibili (e quindi accessibili) al di fuori di essi.

Il progetto è stato quindi suddiviso in tre package principali, contenenti relativamente le classi di MVC, affiancati ad altri due package: application con il MainSBS.java e utils con classi a supporto di alcune funzionalità.

È presente, inoltre, il package test contenente i vari test di JUnit. I quali sono stati utilizzati molto nella fase di testing a seguito della creazione del model.

2.2 Design dettagliato

Poiché questo progetto è stato implementato da un singolo, verrà presentato il design suddiviso per macro categorie.

Per la parte grafica dell'intero progetto, si è optato per l'utilizzo della libreria Swing.

2.2.1 Lettura e Scrittura dei dati

Questa sezione riguarda il salvataggio e il caricamento dei dati, parte fondamentale per il funzionamento dell'intera applicazione.

All'interno del model ho realizzato la lettura e la scrittura dei dati riguardanti tutte le entità coinvolte, ovvero user, item, invoice e shop.

A ognuna di queste entità è associato un file testuale in cui sono memorizzati i valori corrispondenti ai campi.

All'interno del package model.file, è stata realizzata la lettura e la scrittura su file.

Per fare ciò è stato utilizzato il pattern strategy, il quale permette di separare la strategia in base al contesto, tramite l'implementazione, da parte di una specifica classe per ogni entità coinvolta, di un'unica interfaccia FileStrategy.

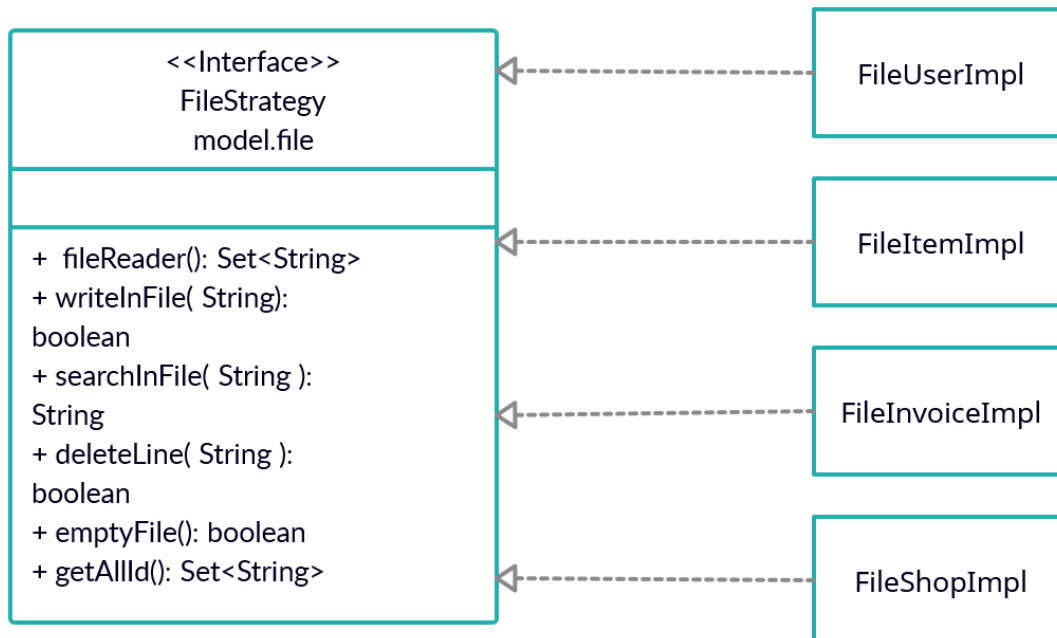


Figura 2.2.1.1: Schema UML della sottoparte riguardante il salvataggio e il caricamento dei dati realizzato tramite il pattern strategy

Queste classi di model vengono chiamate dai controller delle varie entità: UserController, ItemController, InvoiceController e ShopController. In questo modo ogni operazione viene effettuata specificamente per il suo contesto.

2.2.2 Account e Login

Molto importante è anche la parte di login e gestione degli account. Senza la quale non sarebbe possibile procedere oltre l'avvio dell'applicazione.

Per il corretto funzionamento dell'applicativo è di fondamentale importanza implementare delle politiche di gestione per ogni entità e funzionalità del dominio per essere correttamente inserite all'interno del sistema.

La prima fra tutte è l'unicità delle entità. In particolare è stato implementato tramite AddUserView e UserController un controllo dell'input, al fine di rendere univoco lo user id dell'utente per evitare duplicazioni.

E' inoltre di fondamentale importanza che non vengano creati utenti vuoti, è stato quindi implementato un controllo di input in AddUserView, grazie al quale non è possibile creare un nuovo utente se i campi userId, password e nome non sono stati completati.

Gli utenti sono costruiti tramite pattern builder, il quale semplifica la costruzione di oggetti complessi.

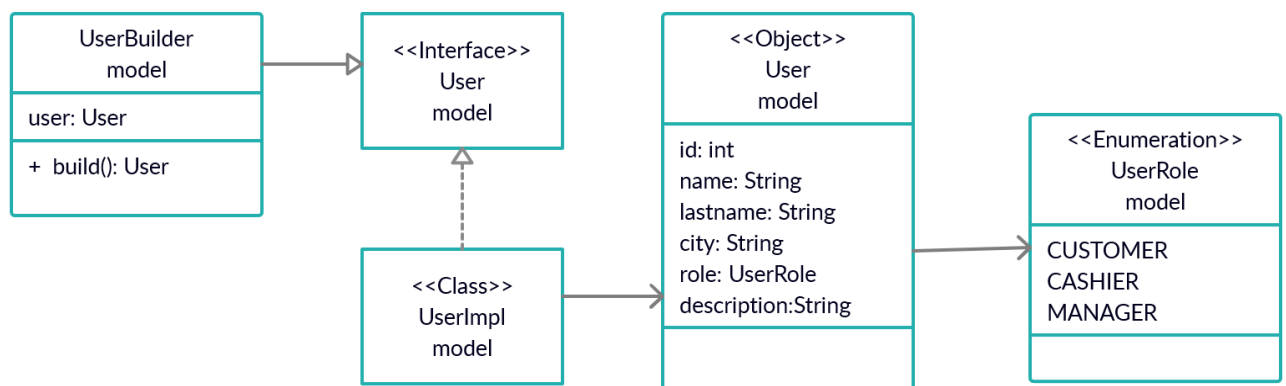


Figura 2.2.2.1: Schema UML della costruzione tramite pattern builder dell'entità user

Come si può notare, i ruoli si trovano all'interno di un Enum.

In base al ruolo di cui si è in carico, si hanno le seguenti autorizzazioni:

Ai Customer NON è consentito l'accesso; I dipendenti si dividono in due categorie: Cassieri e Amministratori.

Gli amministratori hanno accesso a tutta l'applicazione, a differenza dei cassieri che possono accedere solo alla sezione riguardante la cassa.

La parte di login è molto importante poiché, successivamente all'autenticazione, verrà mostrato il menù principale basato sui permessi di cui si dispone.

Essendo presenti due contesti di utilizzo con permessi differenti, in quanto a seconda della tipologia dell'utente autenticato saranno resi disponibili operazioni diversificate, è indispensabile in questo momento il salvataggio dell'account loggato e la gestione delle view.

Per fare ciò ho deciso di passare come parametro lo user ID dell'account loggato ad ogni View.

All'avvio dell'applicazione il valore sarà nullo, ciò porterà alla visualizzazione di un menù principale disabilitato. Al momento del login, la variabile verrà salvata.

Tramite controller verranno controllati i permessi dell'utente loggato e in base a ciò verrà mostrato il menu ad essi adeguato.

Possiamo vedere in ManagementView che l'autenticazione come amministratore permette, tra le altre cose, la gestione di account esistenti, oltre alla registrazione di nuovi account di ogni genere.

Per evitare e prevenire errori di ogni genere, si è trovato necessario aggiungere vari controlli.

- Il primo consiste nell'evitare di dare la possibilità di cancellare il proprio account, in quanto essendo quello loggato, non potrebbe più rientrare successivamente. Questo è stato reso possibile grazie alla presenza della variabile `loggedId` che viene comparata con quella dell'account che si vuole modificare, tramite un controllo nella view `EditUserView`.
- Un altro controllo consiste nell'evitare di dare all'account loggato in quel momento, la possibilità di modificare il suo ruolo, per evitare che il sistema rimanga senza amministratori. In quanto se questo fosse possibile, successivamente nessun utente riuscirebbe ad accedere a questa zona di

gestione account rendendo il sistema inutilizzabile. Anche questo controllo si trova in EditUserView e utilizza la variabile loggedId insieme allo userId che si è intenti a modificare.

- L'ultima sicurezza è quella dell'impossibilità di cambiare userId. Questo obiettivo è stato raggiunto tramite l'utilizzo di un JComboBox che mostra gli id degli utenti esistenti ed in base a questi si possono effettuare modifiche negli altri campi.

Per la gestione degli utenti sono state implementate diverse View:

In particolare si è pensato che sarebbe stato utile per gli amministratori poter visualizzare in due sezioni separate i clienti (CustomerView) e tutti gli utenti (UserView).

2.2.3 Gestione Prodotto

Questa parte riguarda la gestione dei prodotti presenti nel supermercato e successivamente venduti.

Il Model è composto dall'interfaccia Item, implementata da ItemImpl.

Il Controller è composto dall'interfaccia ItemController, implementato da ItemControllerImpl. Questi elementi, che devono essere visualizzabili in un'interfaccia grafica, sono modellabili dall'Controller, grazie al quale è possibile gestire il model item e il model di scrittura e lettura su file per il contesto item, permettendo l'inserimento, la ricerca, la modifica e l'eliminazione.

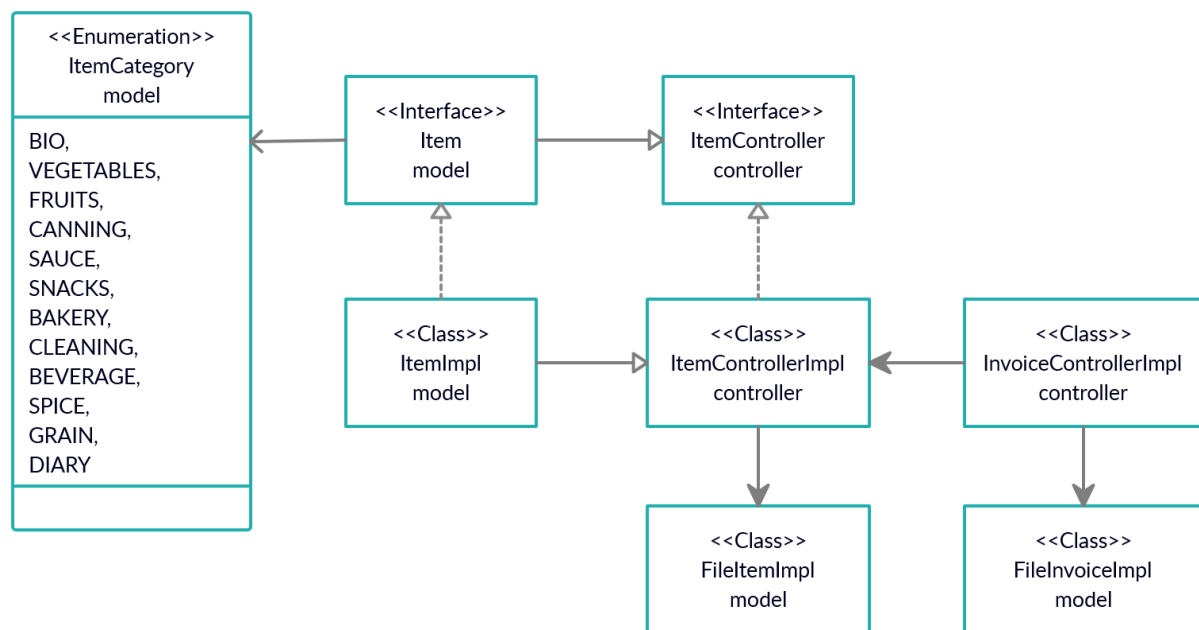


Figura 2.2.3.1: Schema UML del funzionamento di Invoice e Item

Tramite il ruolo del cassiere, è possibile effettuare un acquisto all'interno di InvoiceView.

Qui il cassiere aggiungerà tramite id l'utente che sta effettuando la spesa, potrà controllare che l'utente sia quello desiderato tramite la stampa a video dei dati dell'utente in base all'id.

Inoltre, il cassiere dovrà aggiungere barcode e quantità di ogni prodotto presente nel carrello del cliente.

Quando tutti gli elementi del carrello saranno stati aggiunto alla fattura, sarà possibile fare il checkout pagando.

Tramite la funzione di checkout, il sistema calcolerà le rimanenze in magazzino per ogni prodotto della fattura. Inoltre verranno salvati i dati necessari per il Report Acquisti.

Particolarità: si è pensato di dare a tutti gli utenti la possibilità di effettuare un acquisto, per evitare l'eventuale casistica di avere lo stesso utente con due id e ruoli diversi.

Così come per gli utenti, anche per i prodotti è stato necessario implementare controlli atti a prevenire ed evitare errori di ogni genere, tra cui:

- Il primo controllo consiste nel prevenire la creazione con due id uguali, è stato implementato all'interno di AddItemView tramite ItemController, il quale, prima di procedere con l'inserimento di un prodotto, controlla la sua esistenza nel database.
- Il secondo controllo consiste nella prevenzione della creazione di un prodotto sprovvisto di dati, poiché sarebbe inutilizzabile. All'interno di AddItemView è stato effettuato un controllo sull'inserimento di id univoco, nome del prodotto, quantità a disposizione del supermercato e prezzo unitario, questi sono i campi considerati fondamentali per il funzionamento corretto del sistema.
- Il terzo controllo si trova in InvoiceView, rende impossibile l'aggiunta nella fattura di prodotti di cui non è presente il barcode, poiché inesistenti, questo è stato implementato tramite un JComboBox che presenta tutti e solo gli id dei prodotti esistenti.
- Il quarto controllo, sempre in InvoiceView, riguarda l'impossibilità di aggiungere nella fattura prodotti di cui non ci sono rimanenze all'interno del supermercato. Affiancato al JComboBox con gli itemID appena citato, si trova un'ulteriore JComboBox con le rimanenze.
Il cassiere dovrà inserire uno dei valori presenti o non potrà aggiungere il prodotto.
- L'ultimo controllo è in EditItemView e rende impossibile cambiare itemId o barcode ad un prodotto. Questo obiettivo è stato raggiunto tramite l'utilizzo di un JComboBox che mostra i barcode dei prodotti esistenti ed in base a questi si possono effettuare modifiche negli altri campi.

2.2.4 Reports

All'interno di AccountingView si possono notare due tipi di report, entrambi stampabili.

Anch'essi si basano sul sistema di scrittura e lettura di cui si è parlato nel paragrafo 2.2.1.

Il primo è un report che riguarda gli acquisti degli utenti, il quale tiene in memoria: data, userId e prezzo speso.

Per questioni di leggibilità si è pensato di aggiungere anche name e last name correlati all'Id utente del dato acquisto.

Questo report si trova in PurchaseReportView ed è gestito tramite ShopController. L'utilità di questo report sta nel data analysis, tramite questo si possono studiare i giorni più affluenti e gli utenti più attivi, insieme al prezzo medio di spesa.

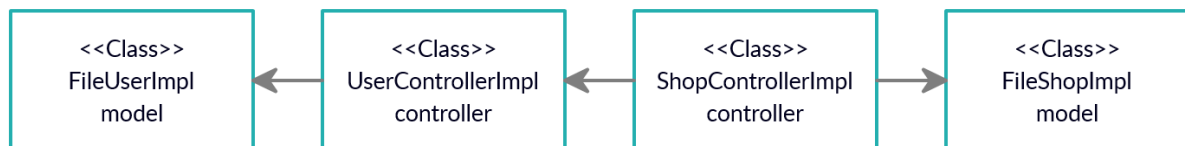


Figura 2.2.4.1: Schema UML di PurchaseReportView

Il secondo report, dentro ItemReportView, riguarda le rimanenze in magazzino, utile all'amministratore per aggiungere nuovi prodotti in magazzino tramite la view AddItemView quando necessario. Il controller utilizzato per questo report è lo stesso per i prodotti: ItemController.

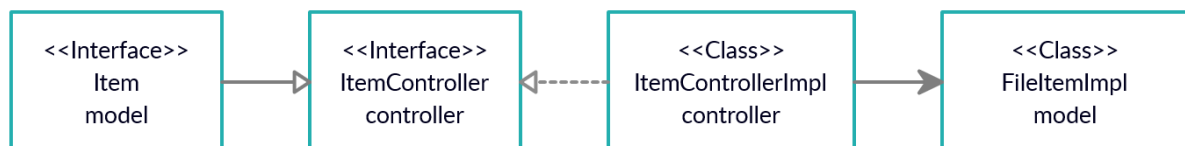


Figura 2.2.4.2: Schema UML di ItemReportView

Capitolo 3

Sviluppo

3.1 Testing

Per il testing è stato utilizzato nella fase iniziale quello automatizzato.

Sono state impiegate classi di test utilizzando il framework JUnit 5 cercando di sottoporre a controlli più metodi possibili dei model e dei controller.

Per la View invece si è reso più efficiente effettuare dei controlli manuali atti a verificare il corretto funzionamento dell'interfaccia e dei meccanismi di eccezione oltre al posizionamento dei vari componenti.

Il tutto è stato inoltre testato a fondo in fase di debugging, per controllare che i funzionamenti fossero quelli desiderati.

3.2 Metodologia di lavoro

Il progetto è stato suddiviso in funzionalità, poichè mi è sembrato intuitivamente più semplice. In questo modo, per ogni funzionalità è stata creata una parte di Model, una parte di Controller e una parte di View.

Si è utilizzato Git tramite il servizio di hosting GitHub come Distributed Version Control System. Poiché ero da sola e lavoravo ad una funzionalità per volta, il tutto è stato sviluppato su branch master.

Ogni commit è stato eseguito dopo l'aggiunta o modifica di una certa funzionalità, a progetto funzionante.

3.3 Note di sviluppo

- Optional utilizzati al posto di ritornare null
- Uso di lambda expression
- Generici
- java.io Buffer reader e writer
- JUnit per i test

Capitolo 4

Commento finale

4.1 Autovalutazione e lavori futuri

Credo che tutto sommato sia stato un bel progetto, con sicuramente ampio margine di miglioramento.

La view non è giovane poiché i dipendenti saranno gli unici possibilitati ad utilizzare il sistema, si è preferito quindi avvantaggiare l'intuitività e la velocità a discapito dell'aspetto puramente grafico. Considerato anche che ad oggi molti sistemi di fatturazione per supermercati hanno queste sembianze.

Causa lavoro e ultimi due esami da dare ho avuto poco tempo a disposizione, sono quindi contenta del risultato finale.

4.2 Difficoltà incontrate e commenti per docenti

E' stato interessante comprendere i concetti teorici della programmazione ad oggetti, nonostante ritenga che questo esame sia eccessivamente impegnativo rispetto al numero di crediti.

Appendice A

Guida Utente

A.1 Avvio dell'applicazione

All'avvio dell'applicazione il sistema parte in una HomeView disabilitata, occorre effettuare l'accesso per abilitarla.

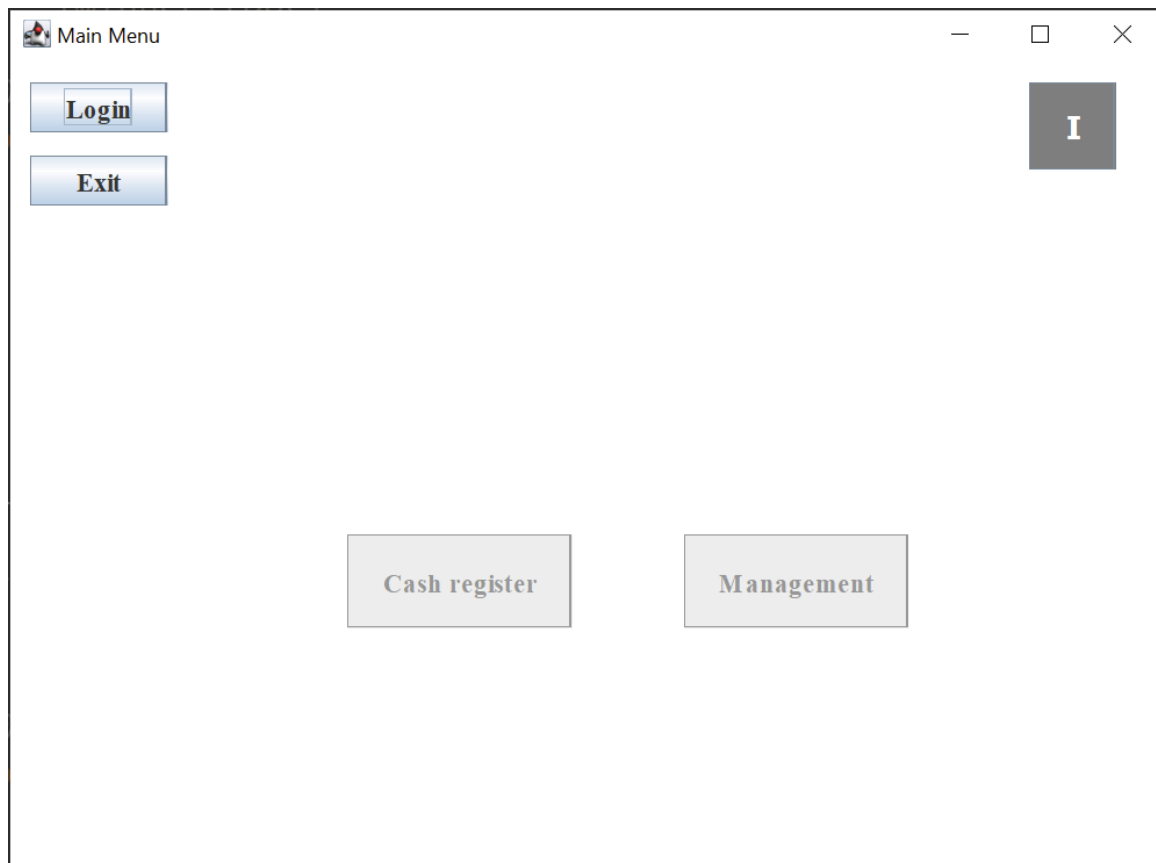


Figura A.1.1 HomeView disabilitata

Cliccando sul bottone “Login” ci si potrà autenticare.

A.2 Log In

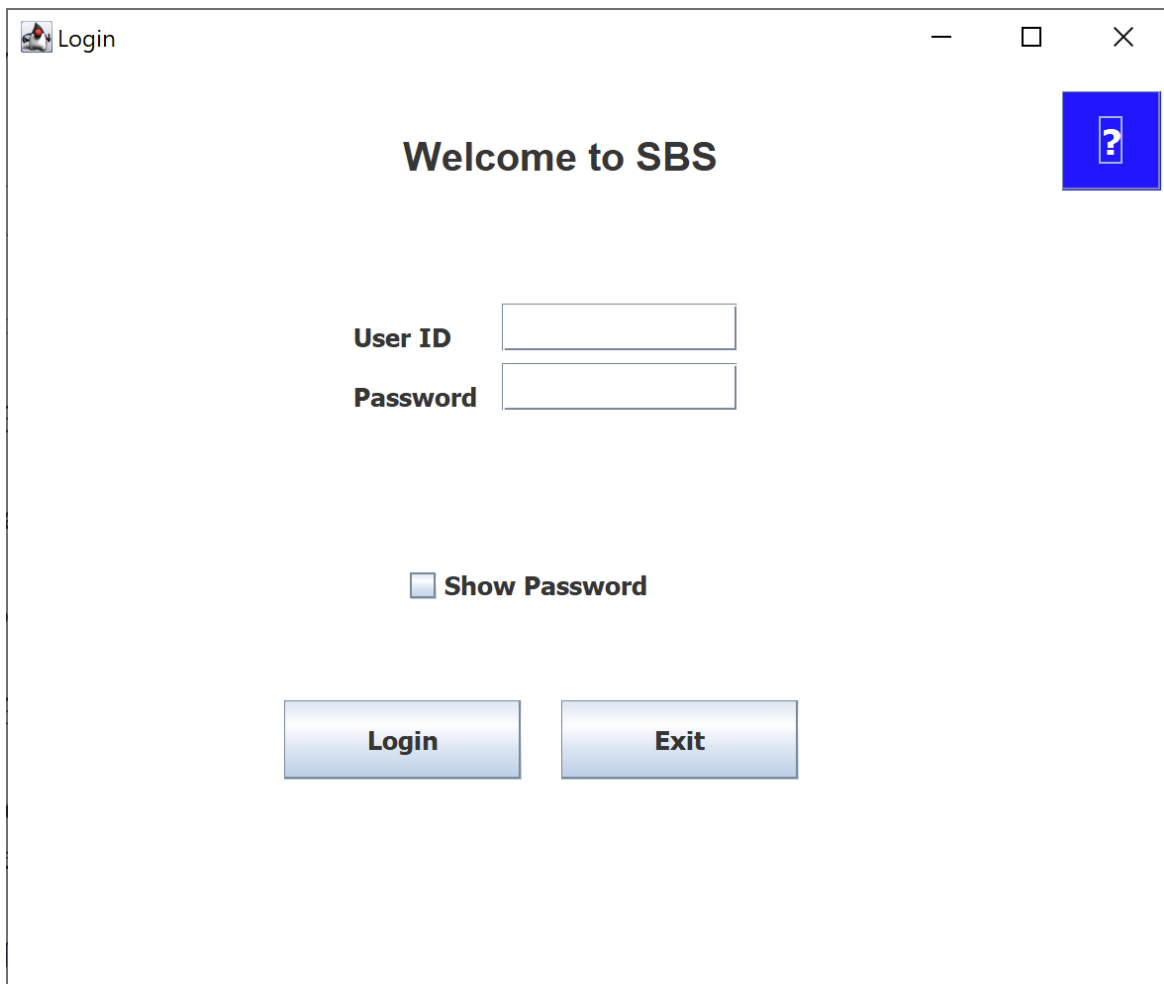


Figura A.2.1 LoginView

Se si utilizza il file User.txt presente sul repository GitHub, si può testare il sistema con i seguenti account:

| | User ID | Password |
|---------|---------|----------|
| Manager | 4 | 123 |
| Cashier | 1 | 000 |

Il login vi riporterà alla schermata principale, a seconda del ruolo con il quale sarete loggati, vedrete abilitata la sezione Management o meno.

A.3 CashierView

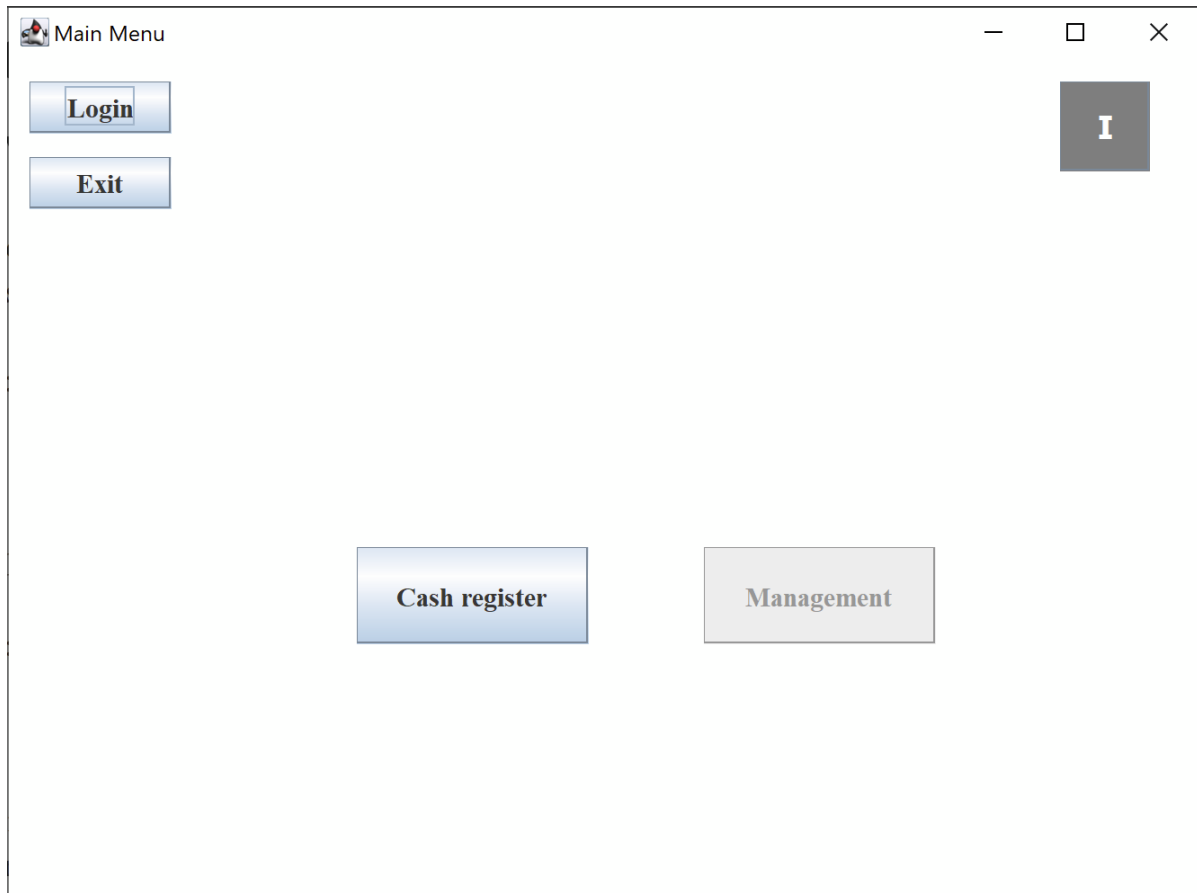


Fig. A.3.1 Presentazione HomeView per gli account di tipo UserRole.CASHIER

Se clicchiamo su “Cash register” > “Invoice Screen” ci troveremo alla schermata nella quale, tramite la figura del cassiere, gli utenti potranno effettuare acquisti:

The screenshot shows a window titled "Invoice Screen" with standard Windows window controls (minimize, maximize, close). The interface is divided into several sections:

- Table Header:** A table with four columns: "Id", "Product", "Quantity", and "UnitPrice". The table body is currently empty and has a light gray background.
- User Search Section:**
 - Text: "Search for CustomerID, check data by clicking 'Check User'"
 - Form: "User ID:" followed by a dropdown menu showing "1" and a "Check User" button.
- Item Search Section:**
 - Text: "Search for Item ID, choose quantity and click Add"
 - Form: Two dropdown menus, both showing "1", followed by an "Add to Cart" button.
- Action Buttons:** Three buttons are arranged horizontally: "Clear", "Print", and "New Invoice".
- Total and Payment Section:**
 - Text: "Total price:" followed by an empty text input field.
 - Button: "Pay" (disabled, grayed out).
 - Button: "Back" (located at the bottom right).

Fig. A.3.2 Presentazione InvoiceView

Cercando un UserId e cliccando su “Check User” sarà possibile il controllo dei dati dell’utente.

Per aggiungere un elemento al carrello basterà selezionare il suo codice Id e la quantità che si vuole acquistare.

Per effettuare il checkout basterà cliccare su “Pay”.

A.4 ManagmentView

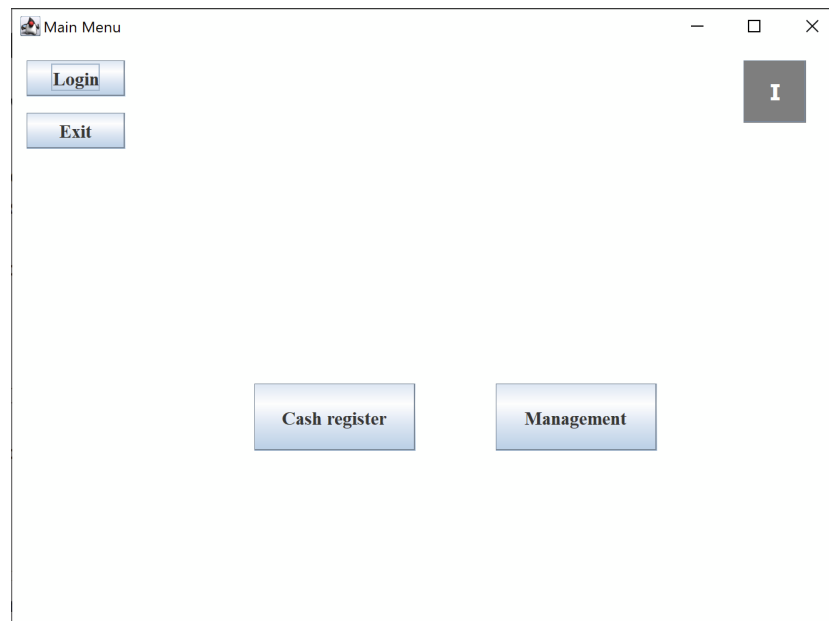


Fig. A.4.1 Presentazione HomeView per gli account di tipo UserRole.MANAGER

Cliccando su “Management” si verrà trasferiti al menù dedicato agli amministratori:

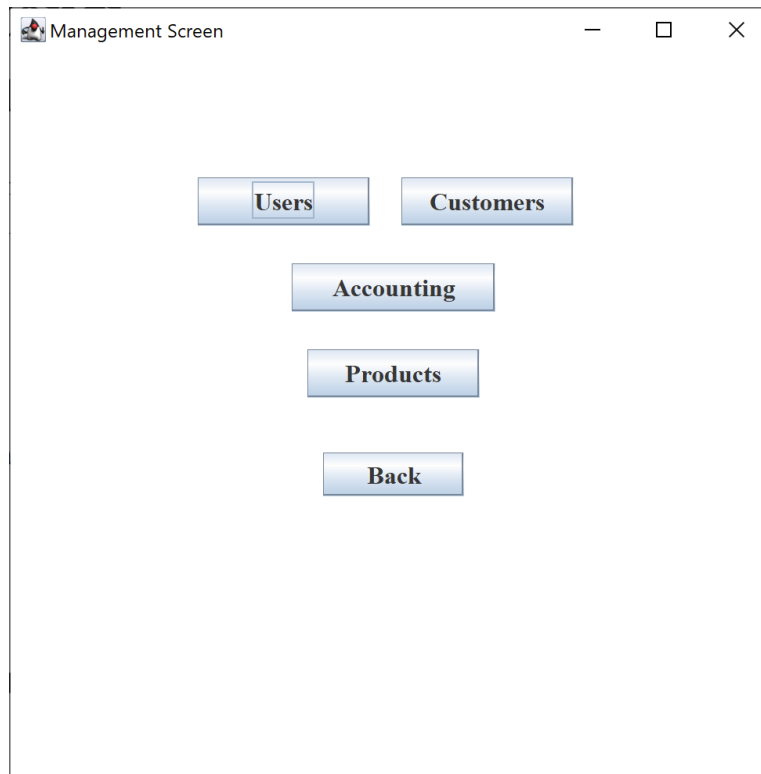
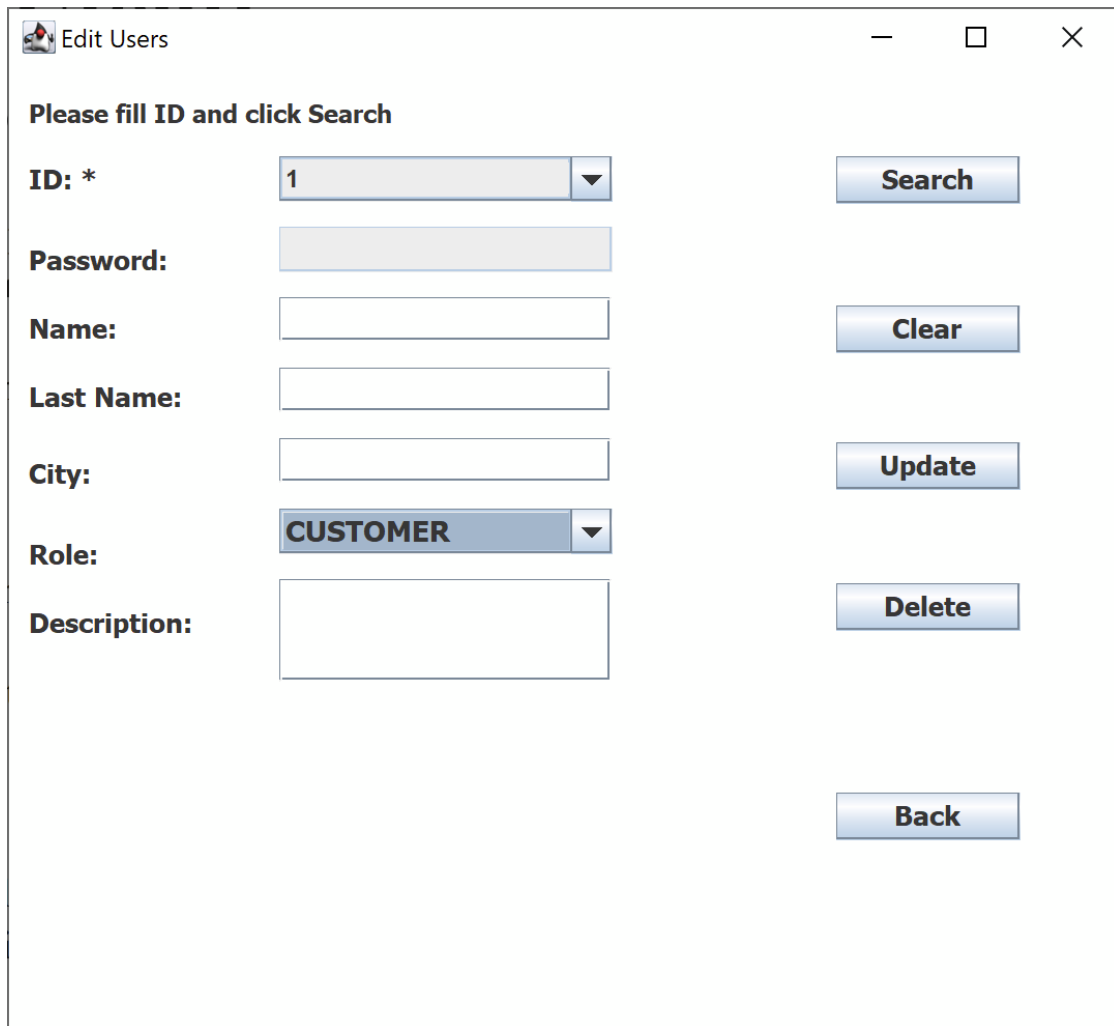


Fig. A.4.2 Presentazione menu ManagementView

In questa guida utente non verranno mostrate tutte le View, si propone di seguito EditUserView, del quale le altre seguono lo stile.



The screenshot shows a window titled "Edit Users" with standard window controls (minimize, maximize, close). Inside the window, there is a prompt "Please fill ID and click Search". Below this, there are several input fields and buttons:

- ID: ***: A dropdown menu currently showing "1".
- Password:**: A text input field.
- Name:**: A text input field.
- Last Name:**: A text input field.
- City:**: A text input field.
- Role:**: A dropdown menu currently showing "CUSTOMER".
- Description:**: A text input field.

On the right side of the form, there are five buttons arranged vertically: "Search", "Clear", "Update", "Delete", and "Back".

Fig. A.4.3 Schermata EditUserView per la gestione degli utenti

Per cercare un utente basterà selezionare un Id tra quelli proposti e cliccare su "Cerca", il sistema vi stamperà tutti i dati relativi a quell'Id utente.

Per modificare un utente, dopo averlo cercato, inserire i nuovi dati e cliccare su "Update".

ATTENZIONE: L'utente verrà aggiornato con i nuovi dati inseriti, è quindi importante prima cercare l'utente e poi compilare i campi che si intende modificare.

Per cancellare un dato utente, selezionare l'id e cliccare "Delete".

In ogni schermata sarà presente il tasto "Back" che riporterà alla schermata precedente.