RELAZIONE FINALE MATTIA FARABOLLINI 126603

1. INTRODUZIONE

Questo documento descrive in modo completo il software da me sviluppato per la gestione del bilancio familiare. Esso è stato sviluppato utilizzando il linguaggio Java 21 utilizzando anche javaFX per l'interfacciagrafica, XML per la persistenza dei dati, manipolati tramite parser DOM, e Gradle come build automation tool. Il progetto adotta il pattern architetturale MVC, garantendo separazione dei dati, logica applicativa e interfaccia utente. Il progetto è concepito per garantire la consistenza dei dati e la facilità di estensione futura, consentendo l'integrazione di nuove funzionalità senza richiedere modifiche profonde alle componenti esistenti.

2. FUNZIONALITÀ IMPLEMENTATE

Ho cercato di implementare tutte le caratteristiche generate che sono state elencate nella specifica del progetto. In poche parole:

- Gestione movimenti: è possibile inserire una transazione nella lista principale e ovviamente quest'ultima può essere consultata nel dettaglio. Inoltre, è possibile eliminare o modificare la transazione, una volta aggiunta, ed è anche possibile inserire spese ricorrenti o future. Tutte le transazioni avranno un tag che ne definisce la categoria principale e nella lista finale è possibile anche ordinare le transazioni in base agli attributi presenti (data, tipologia spesa, scadenza, tag, ecc..).
- Gestione budget: è possibile configurare e gestire il bilancio familiare arricchito delle spese e dei ricavi in base alla data e alle varie categorie. Infatti, nell'interfaccia è presente una sezione che permette di visualizzare il grafico a torta, il quale studia tutte le transazioni e mostra come il budget viene suddiviso nelle varie spese inserite dall'utente. In questa sezione è possibile anche consultare il bilancio finale.
- Statistiche: è possibile visualizzare le statistiche delle transazione selezionate in base al tag e/o al periodo di tempo. Queste vengono suddivise nel grafico a barre, manipolabile dall'utente in modo tale da studiarne l'andamento.

- Scadenzario: è possibile inserire le spese future, come ho citato precedentemente, e quest'ultime vengono visualizzate nella lista principale con tanto di scadenza nel formato giorni-ore-minuti.

3. PACKAGE PRINCIPALI E RESPONSABILITÀ INDIVIDUATE

- Model: Il package model rappresenta il nucleo del dominio applicativo, cioè la parte dell'applicazione che definisce concetti fondamentali come transazioni, importi e categorie. Qui vengono modellati gli oggetti reali che l'utente gestisce, come transazioni finanziarie e tag di classificazione, separando la logica di gestione dei dati dalla loro presentazione e dai calcoli di business. La classe TransactionBase funge da struttura generale per qualsiasi transazione, con attributi chiave come identificativo univoco, importo, data, descrizione, tipo (entrata o uscita) e una lista di tag associati. Questo permette di avere una base comune per transazioni standard e ricorrenti, facilitando l'estendibilità futura. La classe Transaction aggiunge dettagli specifici per le transazioni normali, mentre RecurringTransaction introduce la gestione della ricorrenza, definita tramite l'enum RecurrenceType, consentendo di trattare automaticamente eventi che si ripetono nel tempo, come stipendi o abbonamenti. La classe Money garantisce precisione nella gestione dei valori monetari, sfruttando sia tipi primitivi sia BigDecimal, e include metodi per sommare, convertire e rappresentare importi in modo leggibile. La classe Tag consente di categorizzare le transazioni e supporta relazioni gerarchiche tra tag parent e child, facilitando statistiche aggregate e filtri avanzati. DateRange, infine, gestisce intervalli temporali, utile per calcoli di bilancio o filtraggio delle transazioni. Le responsabilità del package model includono la rappresentazione fedele del dominio applicativo, la protezione della coerenza dei dati e la fornitura di strumenti per effettuare calcoli e verifiche sugli oggetti senza dipendere dalla logica della UI.
- Controller: Il package controller rappresenta il livello logico dell'applicazione, dove vengono implementate tutte le regole di business e le interazioni tra i dati. L'AppController funge da coordinatore principale, collegando transazioni, tag e persistenza in un'unica interfaccia per la view. All'interno di questo package, BudgetController e BudgetManager sono responsabili di calcolare il bilancio complessivo o filtrato per tag e intervalli di tempo, aggregando importi e gestendo eventuali transazioni ricorrenti. StatisticsController e StatisticsManager generano statistiche complesse, come conteggi di transazioni per categoria o

valori medi, offrendo insight utili per la gestione finanziaria. TagController e TagManager gestiscono la creazione, modifica e rimozione dei tag, supportando la gerarchia tra tag parent e child e fornendo metodi per recuperare solo i tag radice o i figli di un dato tag. La struttura di questo package garantisce che la logica di business sia centralizzata, facilitando la manutenzione e l'estensione delle funzionalità, evitando che le view debbano conoscere i dettagli della gestione dei dati.

- View: Il package view gestisce l'interfaccia grafica, costruita con JavaFX, e rappresenta il punto di contatto tra l'utente e l'applicazione. Ogni view è progettata per essere indipendente e aggiornabile dinamicamente, in modo che le modifiche ai dati siano immediatamente riflesse nella UI. AddTransactionView permette all'utente di inserire nuove transazioni o modificare quelle esistenti, gestendo input complessi come importo, data, tag e ricorrenza. TransactionListView offre una panoramica dettagliata delle transazioni con evidenziazione dei colori e supporto alla modifica rapida. FinalBalanceView visualizza il bilancio totale e lo distribuisce graficamente in un grafico a torta, colorando entrate, uscite e transazioni future per una comprensione immediata. StatisticsView consente di filtrare i dati e generare statistiche per tag e periodo, mentre TagManagementView gestisce l'inserimento e la modifica dei tag, mostrando chiaramente la gerarchia tra genitori e figli. L'interfaccia ViewRefreshable assicura che tutte le view implementino il metodo refresh, facilitando l'aggiornamento automatico dei dati. In questo package, le responsabilità principali includono la rappresentazione chiara dei dati, la gestione dell'interazione utente e il collegamento con i controller, assicurando che la UI sia sempre sincronizzata con lo stato interno dell'applicazione.
- Persistency: Il package persistency è dedicato alla conservazione dei dati tra diverse sessioni, assicurando che transazioni e tag non vadano persi alla chiusura dell'applicazione. PersistenceManager definisce un'interfaccia astratta che offre metodi per salvare e caricare i dati, indipendentemente dal formato. XMLPersistenceManager implementa tale interfaccia utilizzando le API DOM per leggere e scrivere documenti XML, gestendo sia la serializzazione che la deserializzazione degli oggetti model. Questo approccio garantisce la compatibilità dei dati tra versioni diverse del programma e la possibilità di estendere il supporto ad altri formati in futuro. Le responsabilità di questo package includono la gestione sicura della persistenza, la conversione tra oggetti in memoria e rappresentazioni su file, e la protezione dell'integrità dei dati contro eventuali errori di scrittura o lettura. Grazie a questa struttura, la logica di salvataggio è completamente separata dal resto dell'applicazione, rendendo

facile sostituire o migliorare il sistema di persistenza senza modificare il modello o le view.

4. PRINCIPI SOLID E CODE SMELLS

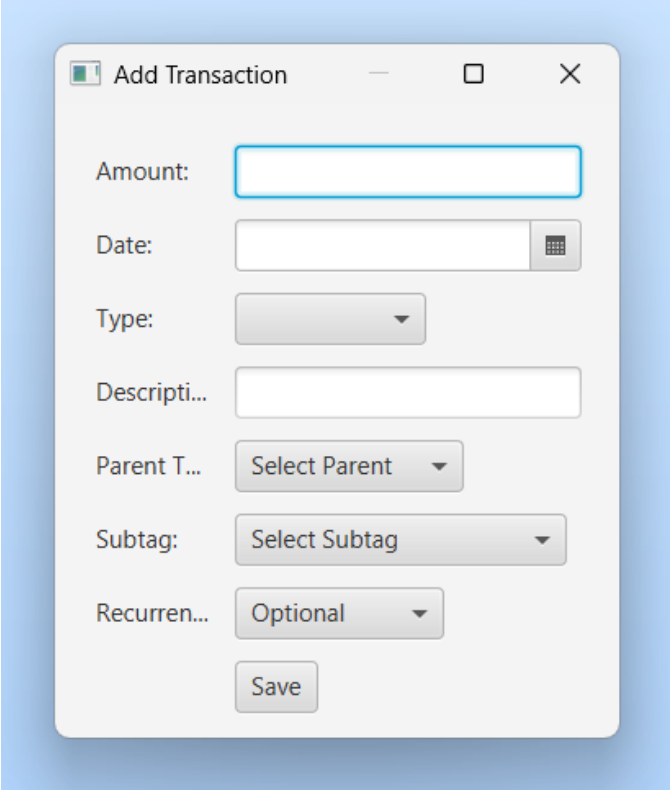
Nello sviluppo di questo progetto, particolare attenzione è stata rivolta all'adozione dei principi SOLID al fine di garantire un'architettura chiara, modulare e facilmente manutenibile. Il principio di Single Responsibility (SRP) è stato rispettato assegnando a ciascuna classe una responsabilità ben definita: le classi del package model si occupano esclusivamente della rappresentazione dei dati e delle regole fondamentali di coerenza, quelle del package controller gestiscono la logica di business e le operazioni sugli oggetti, mentre le classi del package view si limitano alla presentazione e all'interazione con l'utente, senza conoscenza dei dettagli interni dei dati o delle regole di calcolo. Il principio di Open/Closed (OCP) è stato perseguito strutturando i controller e i manager in modo che possano essere estesi con nuove funzionalità, ad esempio nuove statistiche o tipi di transazioni, senza modificare il codice esistente, grazie all'uso di classi astratte e metodi sovrascrivibili. Per quanto riguarda il principio di Liskov Substitution (LSP), le sottoclassi di TransactionBase, come Transaction e RecurringTransaction, possono essere utilizzate in contesti dove ci si aspetta una transazione generica senza compromettere la correttezza dei calcoli o delle operazioni. Il principio di Interface Segregation (ISP) è stato applicato creando interfacce mirate, come ViewRefreshable, che obbligano le view a implementare solo ciò che è strettamente necessario, evitando interfacce "gonfie" con metodi inutilizzati. Infine, il principio di Dependency Inversion (DIP) è stato considerato facendo sì che le view dipendano da astrazioni (controller) e non da implementazioni concrete, consentendo di sostituire facilmente manager o controller senza modificare la UI. Dal punto di vista dei code smells, il progetto è stato progettato per minimizzare problemi comuni come classi troppo grandi, metodi troppo complessi o duplicazione di codice. La separazione netta tra model, controller, view e persistency riduce il rischio di classi "god" o di logica distribuita in modo incoerente. L'uso di metodi di supporto, come refresh nelle view e metodi di utilità per la gestione dei tag e delle transazioni, aiuta a evitare duplicazioni e facilita la leggibilità. Inoltre, la struttura a package chiari e ben definiti consente di localizzare facilmente eventuali problemi, semplificando debugging e test, e garantisce che ogni modifica rimanga confinata alla componente pertinente senza effetti collaterali indesiderati. Complessivamente, l'attenzione ai principi SOLID e alla pulizia del codice contribuisce a un'applicazione robusta, modulare e facilmente estendibile.

5. PERSISTENZA ED ESTENDIBILITÀ

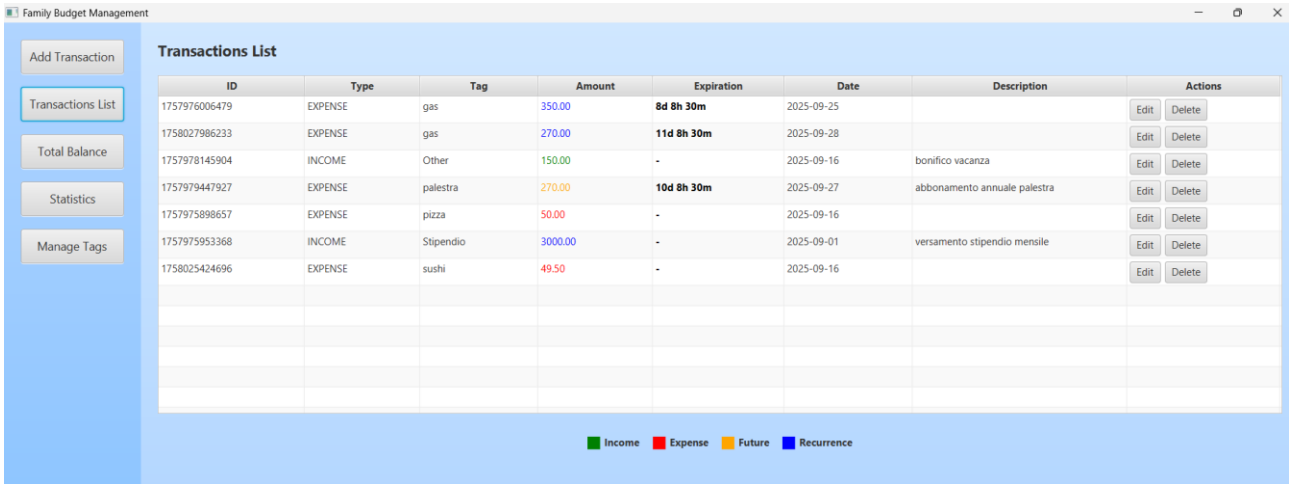
Per quanto riguarda la persistenza dei dati e l'estendibilità del progetto, è stato adottato un approccio modulare e altamente flessibile che combina l'utilizzo di XML con il pattern DOM, centralizzando tutte le operazioni di salvataggio, caricamento e gestione dei dati all'interno del package persistency, principalmente tramite le classi PersistenceManager e XMLPersistenceManager. Questa scelta architetturale permette alle classi del model e dei controller di concentrarsi esclusivamente sulla logica di business, senza doversi preoccupare dei dettagli di memorizzazione, garantendo al contempo la coerenza e l'integrità dei dati anche in caso di chiusure improvvise dell'applicazione o di anomalie durante le operazioni di scrittura su file. Il sistema converte i dati delle transazioni e dei tag in strutture XML, salvandoli su file e ricaricandoli automaticamente all'avvio dell'applicazione, assicurando una persistenza sicura, affidabile e facilmente leggibile sia da parte del sistema sia da eventuali strumenti esterni. Per quanto riguarda l'estendibilità, il progetto è stato strutturato secondo principi di progettazione modulare e orientata alle interfacce: sono state introdotte numerose interfacce che definiscono contratti chiari tra i vari componenti del sistema, permettendo a un altro sviluppatore di implementare nuove funzionalità o sostituire componenti esistenti senza la necessità di modificare il codice già presente. Questo approccio facilita la manutenibilità e incrementa la modularità dell'intero sistema. Inoltre, sono stati applicati principi di programmazione generica e astrazioni, come l'utilizzo di collezioni generiche di tipo List invece di implementazioni concrete come ArrayList o LinkedList, in linea con i principi di Dependency Inversion e Interface Segregation; in questo modo, le classi dipendono da astrazioni piuttosto che da dettagli concreti, rendendo il codice più flessibile, estendibile e pronto a supportare nuove strutture dati o collezioni diverse senza interventi significativi sul codice esistente. Anche le view e i controller sono progettati per operare tramite astrazioni e listener, consentendo l'integrazione di nuovi pannelli, statistiche o metodi di visualizzazione senza impattare sulla logica sottostante. In particolare, le view sfruttano liste osservabili, binding e listener di JavaFX, permettendo aggiornamenti automatici dei dati visualizzati e semplificando l'aggiunta di nuove funzionalità grafiche o di pannelli informativi. Grazie a questa architettura modulare, è possibile sostituire facilmente il backend di persistenza con altre tecnologie, come database relazionali o formati JSON, semplicemente implementando una nuova versione di PersistenceManager o di XMLPersistenceManager, senza necessità di modificare il model o le view. Complessivamente, l'adozione di interfacce, l'uso di collezioni generiche, la centralizzazione della persistenza e la separazione netta tra logica di business, presentazione e gestione dei dati rendono il sistema coerente, modulare, manutenibile e facilmente estendibile, pronto a integrare nuove funzionalità, nuove categorie di transazioni o filtri aggiuntivi con interventi minimi, migliorando al contempo la leggibilità, la qualità e la robustezza complessiva del progetto.

6. PANORAMICA INTERFACCIA

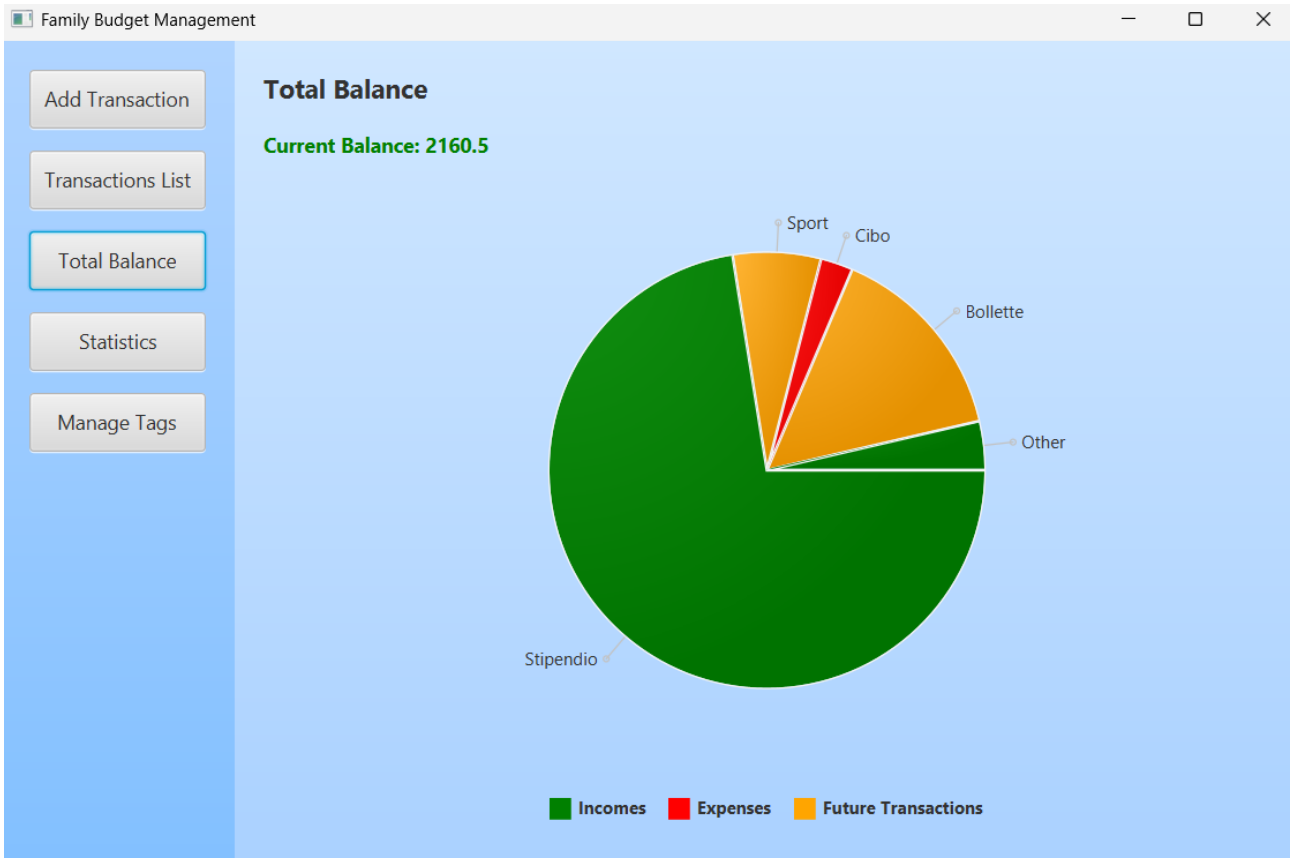
- L’interfaccia è sviluppata in 5 livelli differenti. Nel progetto che ho consegnato lo scorso appello ce n’erano solo 4 e presentava alcune operazioni in meno; questa volta invece ho deciso di apportare delle modifiche, non tanto nel design dell’interfaccia, ma soprattutto sulle funzionalità che essa presenta. Attualmente le modifiche principali e aggiornate, rispetto la scorsa versione del “Family Budget Management” sono:
- Add Transaction non ha una schermata tutta sua, ma si presenta come un pop up che è possibile aprire in qualsiasi momento, garantendo la modifica dei grafici presenti istantaneamente. Inoltre, è possibile aggiungere transazioni ricorrenti, scegliendo la tipologia prestabilita, e selezionare un Parent Tag o un Subtag, a seconda del proprio utilizzo.



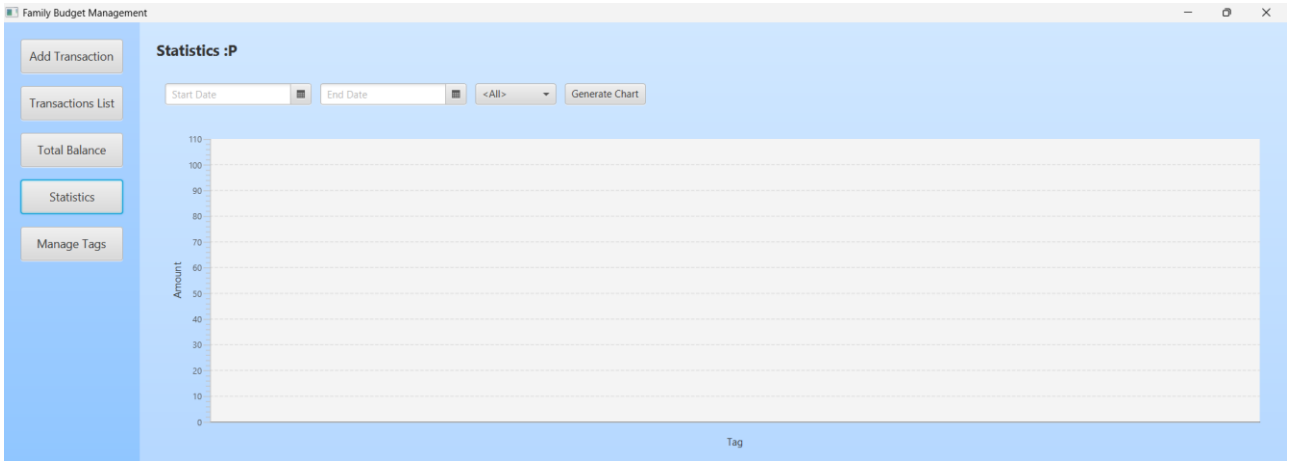
- Transaction List risulta essere la schermata principale che viene visualizzata all’apertura del programma e finalmente gestisce le transazioni ricorrenti, lo scadenzario e la possibilità di modifica o di rimozione di una transazione direttamente nella lista finale.

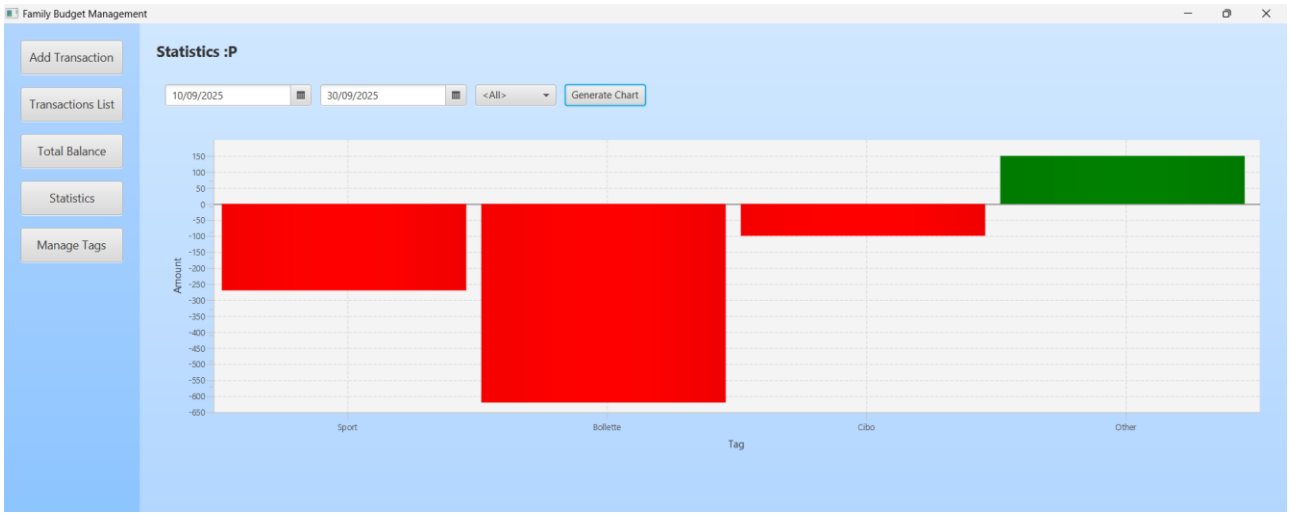


- Total Balance non presenta modifiche visibili rispetto il progetto precedente.

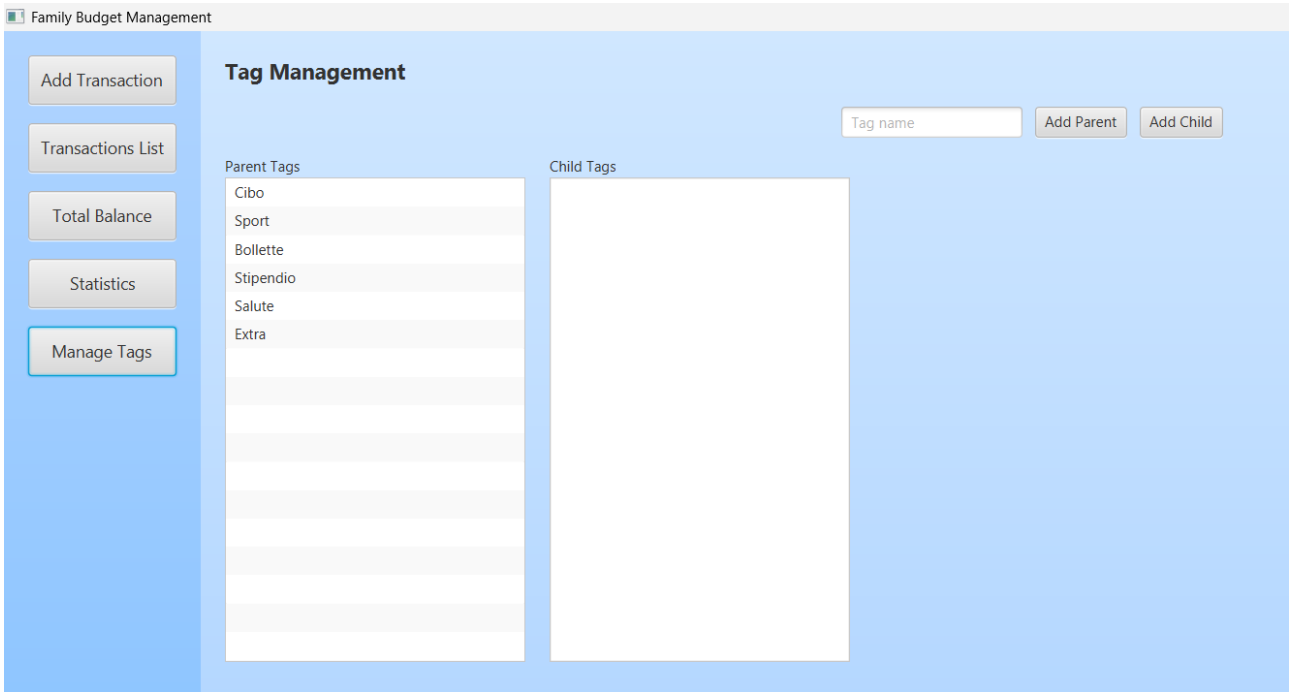


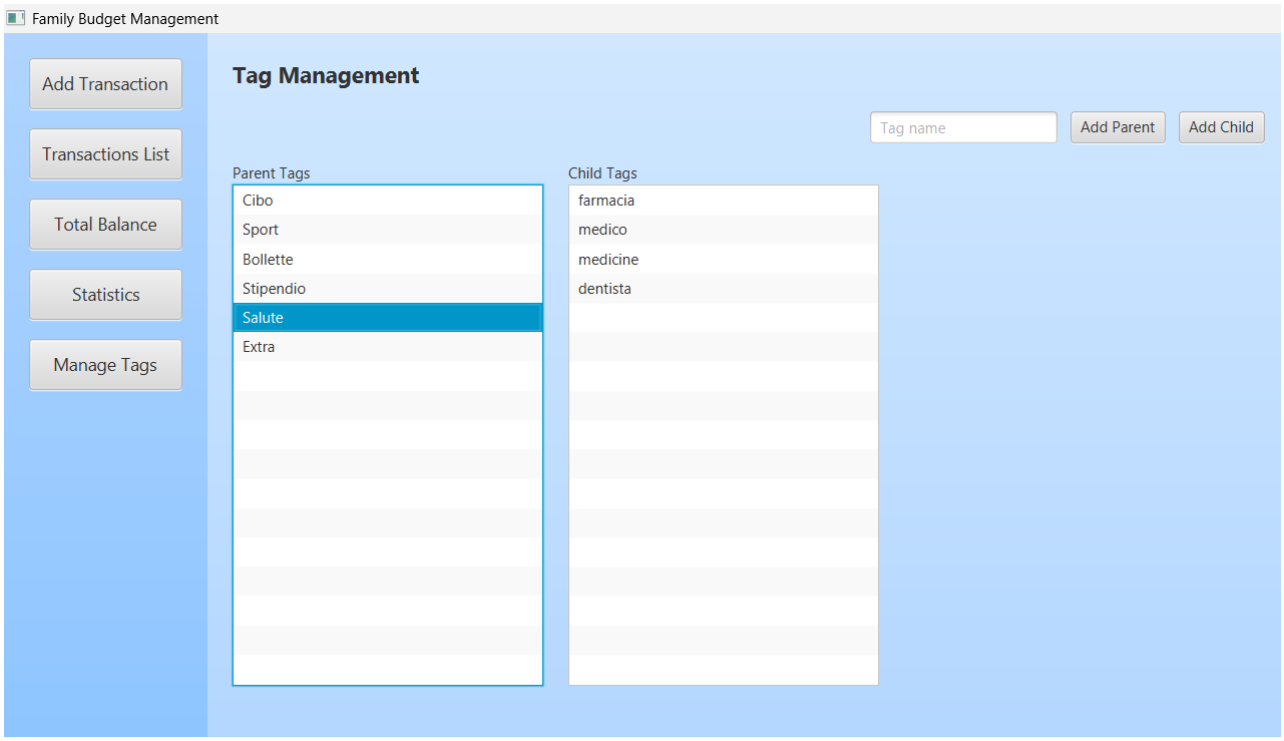
- Statistics differisce esclusivamente nella grafica, in quanto le transazioni in negativo vengono colorate di rosso e i tag selezionabili vengono visualizzati tramite menù a tendina.





- Tag Management è una nuova schermata (non presente nel progetto precedente), che permette di gestire la gerarchia dei tag a proprio piacimento. Nelle due tabelle sono presenti rispettivamente tutti i tag padre e tutti i tag figli, collegati al padre. È possibile, quindi, aggiungere entrambe le tipologie dei tag per caratterizzare le transazioni.





7. CONCLUSIONE

Il progetto "Family Budget Management" spero abbia raggiunto gli obiettivi prefissati, offrendo un sistema completo per la gestione delle finanze familiari. La separazione tra model, view e controller, unita a una gestione chiara dei tag e delle transazioni, garantisce facilità di manutenzione e possibilità di estendere le funzionalità future. La persistenza tramite XML assicura che i dati siano sempre salvati e caricabili, rendendo il sistema

affidabile. L'applicazione è pronta per essere estesa con nuove statistiche, tipologie di transazioni o interfacce più avanzate senza compromettere la stabilità del progetto.