Politecnico di Torino

Dipartimento di ingegneria gestionale e della produzione

Corso di Laurea Triennale in Ingegneria Gestionale
Classe L8 – Ingegneria dell'informazione A.A. 2022/2023

Sviluppo di un software per ottimizzare l'acquisto di libri su Amazon

Relatore
Prof. Fulvio Corno

Candidato
Davide Faudella S273255

Settembre 2023

Indice

1 Proposta di progetto

1.1 Studente proponente

S273255 Davide Faudella

1.2 Titolo della proposta

Sviluppo di un software per ottimizzare l’acquisto di libri su Amazon

1.3 Descrizione del problema proposto

L’obbiettivo del software è quello di permettere all’utente di generare automaticamente una lista di libri da comprare su Amazon, elaborata rispettando dei vincoli selezionabili tramite un'interfaccia grafica.
Alcuni dei vincoli che determineranno la scelta dei libri da inserire nella lista saranno il budget disponibile per l'acquisto, il genere, il rating dei libri, il numero di recensioni e altri che verranno mostrati in seguito.
Un'altra funzione sarà quella di una ricerca di libri tramite autore (o anno in cui il libro è stato in top 100) che potranno essere aggiunti a una lista “Libri da leggere” (lista esterna a quella precedente e che ha come funzione quella di tener traccia dei libri da leggere in un secondo momento).

1.4 Descrizione della rilevanza gestionale del problema

Il problema è rilevante da un punto di vista gestionale, in quanto si tratta di un problema di ottimizzazione.
Il software, infatti, si focalizza su una gestione ottimale delle finanze.
Utilizzeremo un data-set contenente i dati dei bestseller di Amazon dal 2009 al 2021 e tramite questi dati pianificheremo al meglio la nostra scelta nel comprare un insieme di libri.
Grazie al vincolo di budget, infatti, potremo selezionare la cifra a nostra disposizione e il software genererà la lista di libri che più si avvicina alle nostre finanze, ovviamente tenendo conto delle preferenze inserite dell'utente.
L’utente potrebbe per esempio volere un rating medio dei libri nella lista di almeno 4.3 stelle, solo libri con copertina rigida, di genere fiction, con minimo 1000 recensioni e che sono stati in top 100 tra il 2010 e il 2012.
Vi sarà inoltre la possibilità di aggiungere manualmente uno o più libri nella lista e l’algoritmo si occuperà poi di riempirla sempre seguendo i parametri selezionati.
Al termine della generazione, oltre alla lista di libri, verranno mostrati i dati riguardanti il numero di libri presenti nella lista, il costo totale da sostenere e il bestseller della lista.
Un processo del genere richiede tempo ma grazie a questo software il risultato sarà solamente a qualche click di distanza, ottimizzato e automatizzato.

1.5 Descrizione dei data-set per la valutazione

Il software utilizzerà un data-set proveniente dalla piattaforma Kaggle e contenente i top 100 bestseller su Amazon negli anni 2009-2021 [https://www.kaggle.com/datasets/abdulhamidadavize/top-100-best-selling-books-on-amazon-20092021].
Questo database verrà pulito e adattato in quanto presenta alcune righe con dati errati o non completi, verrà poi convertito per essere importato su HeidiSQL.
Il database è costituito da più di 1000 libri, con caratteristiche rappresentate nelle 10 colonne:

- Numero identificativo
- Prezzo
- Ranking
- Titolo
- Numero di recensioni
- Rating
- Autore
- Tipo di copertina
- Anno in cui è stato in top 100
- Genere

1.6 Descrizione preliminare degli algoritmi coinvolti

Il software sarà realizzato in linguaggio Java, le interfacce grafiche in JavaFx e i pattern utilizzati saranno MVC e DAO.
Il software lavorerà con algoritmi ricorsivi, il cui compito sarà quello di estrarre dalle innumerevoli combinazioni possibili quella ottimale, definita tramite parametri definiti dall'utente.
I vincoli saranno combinati, andando così ad avere un algoritmo più complesso da gestire.
Per esempio potremmo voler mantenere un certo livello di rating medio, ovviamente rimanendo nel budget e al contempo rispettando genere, intervallo di anni, tipo di copertina, numero di recensioni e ranking selezionati dall'utente.
La ricerca dei libri e in seguito l'eventuale aggiunta alla lista “Libri da leggere” o alla lista per la ricorsione sarà invece effettuata tramite un algoritmo di ricerca e interrogazioni SQL al database.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

All'apertura del programma l'utente avrà a disposizione la funzione di ricerca dei libri, come in una biblioteca.
Tramite una comboBox potrà selezionare un autore (e/o un range di anni), e confermando la scelta, verrà popolata una tabella con tutti i libri scritti dall'autore (e/o quelli degli anni selezionati).
A questo punto un'altra comboBox si popolerà con i titoli dei libri, e selezionandoli potranno essere aggiunti alla lista "Libri da leggere".
Presente anche un metodo per rimuovere un libro dalla lista o cancellarla interamente.

In un'altra sezione del programma invece sarà presente la parte della generazione automatica della lista di libri.
Qui saranno disponibili diversi controlli per tutti i parametri (budget, genere, rating medio, tipo di copertina, numero recensioni minime e anno) seguiti da un pulsante "conferma" che farà partire la ricorsione.
Prima di ciò però potranno essere aggiunti manualmente uno o più libri alla lista, che saranno presenti anche nel caso in cui non rispettassero alcun vincolo.
Al termine l'utente avrà davanti a sé una lista di libri, il numero di libri, il totale in euro e il bestseller.

2 Descrizione dettagliata del problema affrontato

Il problema che viene affrontato in questa tesi è un problema di ottimizzazione.

Infatti, si vuole aiutare l'utente a ottimizzare e velocizzare un processo, in questo caso quello della scelta di un insieme di libri.
In una piattaforma come Amazon o simili non è previsto un tipo di servizio che componga automaticamente una lista di prodotti da comprare in base alle preferenze dell'utente.

Il cliente deve quindi cercare libro per libro quelli di suo gusto e aggiungerli al carrello, tutto questo continuando a controllare che l'aggiunta di un determinato libro non vada a sforare il budget.
Questo processo oltre che essere lungo e macchinoso potrebbe anche non essere ottimale in quanto è molto difficile riuscire a trovare l'insieme di libri che più si avvicina al nostro budget, sempre rispettando le nostre preferenze.
Qui nasce l'idea di creare un qualcosa che possa andare a migliorare l'esperienza di un cliente, selezionando per lui l'insieme migliore, comunque lasciandogli la possibilità di aggiungere un prodotto manualmente.

Sfruttando un algoritmo ricorsivo il software genera in automatico una lista di libri ad hoc, in base ai filtri impostati dall'utente e ovviamente in base al suo budget.

Questo software vuole quindi essere un elemento di supporto per questo tipo di piattaforme, ponendosi l'obiettivo di adattarsi alle esigenze di ogni singolo cliente.

3 Descrizione del data-set

Il data-set, proveniente dalla piattaforma Kaggle, è composto da una tabella “libri” che contiene al suo interno i dati riguardanti i libri che sono stati nella top 100 dei libri più venduti su Amazon negli anni 2009-2021.
I dati, inizialmente in formato CSV, sono stati opportunamente importati su HeidiSQL, andando a effettuare una pulizia dove necessario.
Importante notare che la colonna “Anno” si riferisce all’anno in cui il libro è stato in top 100 e non al suo anno di uscita, per questo motivo si può trovare lo stesso libro più volte all’interno del data-set (nel caso in cui lo stesso libro sia stato in top 100 per più anni).
Di seguito è riportata la tabella “libri” del database.

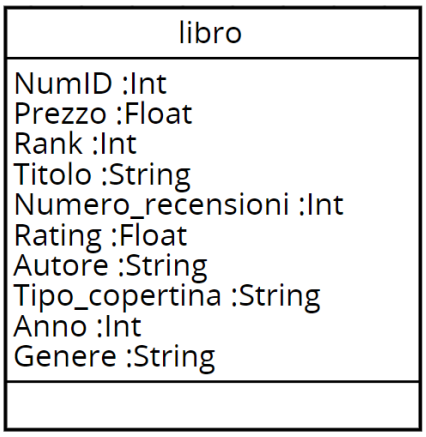

Figura 1: Tabella libri database

La tabella “libri” contiene le seguenti informazioni di un libro:

- Numero identificativo: un numero crescente e univoco usato come identificativo
- Prezzo
- Rank: il suo posizionamento nella classifica nell’anno in questione
- Titolo
- Numero di recensioni: numero di recensioni lasciate al prodotto
- Rating: numero di stelle dato dagli utenti
- Autore: nome dell’autore
- Tipo di copertina: copertina dura, morbida etc.
- Anno: anno in cui è stato in top 100
- Genere: Fiction o Non Fiction

4 Descrizione strutture dati e algoritmi coinvolti

4.1 Strutture dati

Il software è stato programmato in Java, usando il pattern **DAO** (Data Access Object) e il pattern **MVC** (Model View Controller). Per l'interfaccia grafica in JavaFX è stato utilizzato Scene Builder. Il progetto è costituito da tre package per separare il modello, l'accesso al database e l'interfaccia utente.

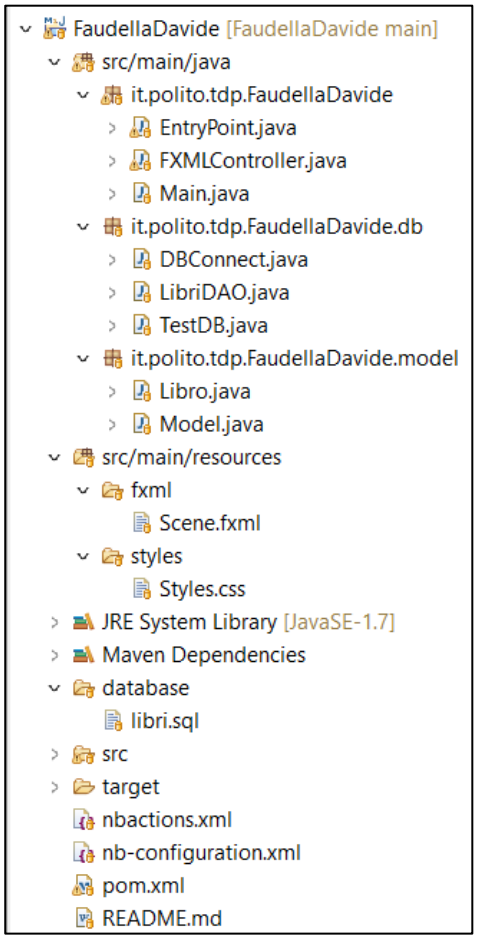

Figura 2: Struttura dati

Il package *it.polito.tdp.FaudellaDavide* contiene il Main, che è la classe principale del progetto, la classe EntryPoint, utilizzata per l'avvio del programma e la classe FXMLController, che viene utilizzata per collegare il Model all'interfaccia grafica.

Il package *it.polito.tdp.FaudellaDavide.db* contiene le classi necessarie per l'accesso al database e necessarie quindi per accedere ai suoi dati.

Il package *it.polito.tdp.FaudellaDavide.model* contiene la classe Libro, utilizzata per gestire l'oggetto. Inoltre, vi è la classe Model , dove si trova la logica principale dell'applicazione.

4.2 Algoritmi principali

La ricerca dei libri viene eseguita nel **Model** utilizzando *cercaLibri(String aut, int annoI, int annoF), cercaLibriAutore(String aut)* o *cercaLibriAnno(int annoI, int annoF).*
Questi metodi ricevono i parametri scelti dall'utente e ritornano una lista con i libri che li rispettano.
La lista "tuttiLibri" che viene usata nel ciclo è una lista che contiene tutti i libri, popolata tramite il metodo *getAllBooks()* situato nella classe **LibriDAO**.

```
public List<Libro> cercaLibri(String aut, int annoI, int annoF) {
    List<Libro> libri = new LinkedList<>();
    for (Libro l : tuttiLibri) {
        if (l.getAutore().equals(aut) && l.getAnno() <= annoF && l.getAnno() >= annoI) {
            libri.add(l);
        }
    }
    return libri;
}

public List<Libro> cercaLibriAutore(String aut) {
    List<Libro> libri = new LinkedList<>();
    for (Libro l : tuttiLibri) {
        if (l.getAutore().equals(aut)) {
            libri.add(l);
        }
    }
    return libri;
}

public List<Libro> cercaLibriAnno(int annoI, int annoF) {
    List<Libro> libri = new LinkedList<>();
    for (Libro l : tuttiLibri) {
        if (l.getAnno() <= annoF && l.getAnno() >= annoI) {
            libri.add(l);
        }
    }
    return libri;
}
```

Figura 3: Metodi cercaLibri(), cercaLibriAutore() e cercaLibriAnno() - Model

Questi metodi vengono chiamati dal metodo *doRicerca()* situato nella classe **FXMLController.**
In base ai parametri selezionati, dopo gli opportuni controlli, viene chiamato il metodo corrispondente nel **Model.**

```
    if (aut != null && annoI == null && annoF == null) {
        trovati.addAll(model.cercaLibriAutore(aut));
    } else if (aut == null && annoI != null && annoF != null) {
        if (annoF < annoI) {
            txtErrori.setVisible(true);
            txtErrori.appendText("Seleziona un range di anni valido\n");
            return;
        }
        trovati.addAll(model.cercaLibriAnno(annoI, annoF));
    } else {
        if (annoF < annoI) {
            txtErrori.setVisible(true);
            txtErrori.appendText("Seleziona un range di anni valido\n");
            return;
        }
        trovati.addAll(model.cercaLibri(aut, annoI, annoF));
        System.out.print(trovati.size());
    }
```

Figura 4: Parte del metodo doRicerca() – FXMLController

Per quello che riguarda l'aggiunta di libri alla lista "daLeggere", per evitare ogni tipo di errore vengono eseguiti dei controlli. Viene infatti verificato che nella comboBox interessata ci sia effettivamente selezionato un libro, ma soprattutto viene controllato se il libro non sia già presente nella lista tramite il metodo *aggiungi(Libro l)* situato nel **Model**.
Per la rimozione di un libro viene usato un metodo analogo *doRimuovi()* dell'**FXMLController** nel quale viene chiamato *rimuovi(Libro l)* sempre dal **Model** che controlla che il libro da rimuovere sia effettivamente nella lista**.**

```
@FXML
void doAggiungiLeggi(ActionEvent event) {
    txtErrori.clear();
    txtErrori.setVisible(false);
    if (cmbTitoloLeggi.getValue() == null) {
        txtErrori.setVisible(true);
        txtErrori.appendText("Scegliere un libro da aggiungere alla lista \n'Libri da leggere'\n");
        return;
    }
    boolean res = model.aggiungi(cmbTitoloLeggi.getValue());

    if (!res) {
        txtErrori.setVisible(true);
        txtErrori.appendText("Il libro è già presente nella lista 'Libri da leggere'\n");
        cmbTitoloLeggi.valueProperty().setValue(null);
        return;
    }

    cmbTitoloLeggiRimuovi.getItems().clear();
    cmbTitoloLeggiRimuovi.getItems().addAll(model.getDaLeggere());
    cmbTitoloLeggi.valueProperty().setValue(null);
    table2.setItems(FXCollections.observableArrayList(model.getDaLeggere()));
}
```

Figura 5: Metodo doAggiungiLeggi() – FXMLController

I metodi *aggiungi()* e *rimuovi()* del model, che prima di aggiungere o rimuovere un libro dalla lista, controllano che l'azione che si intende eseguire sia legale e non causi errori.

```
public boolean aggiungi(Libro l) {
    if (daLeggere.contains(l)) {
        return false;
    }
    daLeggere.add(l);
    return true;
}

public boolean rimuovi(Libro l) {
    if (daLeggere.contains(l)) {
        daLeggere.remove(l);
        return true;
    } else {
        return false;
    }
}
```

Figura 6: Metodi aggiungi(), rimuovi() – Model

Verranno ora visti i metodi legati all'algoritmo ricorsivo.
Nel metodo *doRicorsione()* vengono passati tutti i parametri già controllati da *doGenera()* dell'**FXMLController.** Come prima cosa vengono resettati tutti i parametri, quindi il flag a 0, una nuova lista contenente **tutti** i libri che rispettano i vincoli e l'inizializzazione della best.
La lista contenente tutti i libri viene popolata grazie alla *getBooksParametri()* della classe **LibriDAO** che interroga il database chiedendo i libri con i parametri selezionati.

Poi viene controllato se per caso il costo totale dei libri è inferiore al budget, in quel caso chiaramente la best conterrà tutti i libri e verrà risparmiato del tempo.
In caso non fosse così allora verrà chiamata *cerca()*, che è dove si svolge effettivamente tutto il processo di scelta dei libri.
Qui tutti i libri della lista totale verranno aggiunti e rimossi uno alla volta e la funzione richiamerà se stessa fino a che non saranno terminate le combinazioni possibili. Ogni volta che viene trovata una combinazione migliore viene aggiornata la lista best.
Questo può succedere quando la differenza tra il budget e il costo totale è inferiore alla best precedente, o, in caso di "pareggio" della differenza, quella con la media di stelle più elevata (trovata tramite *stelleMaggiori()*).

```
public List<Libro> doRicorsione(float budget, float rating, int numero, int annoI, int annoF, String copertina,
        String genere) {
    flag = 0;
    libriParametri = dao.getBooksParametri(rating, numero, annoI, annoF, copertina, genere);
    best = new ArrayList<>();

    if (budget > totale(libriParametri)) {
        best = new ArrayList<>(libriParametri);
        return best;
    }
    cerca(parziale, budget);
    return best;
}

private void cerca(List<Libro> parziale, float budget) {
    if (totale(best) == budget) {
        flag = 1;
    }

    float diffB = budget - totale(best);
    float diffP = budget - totale(parziale);

    if (diffB > diffP) { // parziale si avvicina di più al budget da spendere
        best = new ArrayList<>(parziale);
    }
    if (diffB == diffP) { // se è uguale prendo quello con più stelle medie
        if (stelleMaggiori(best, parziale) == true) {
            best = new ArrayList<>(parziale);
        }
    }
    for (Libro l : libriParametri) {
        if (isValid(parziale, l, budget)) {
            parziale.add(l);
            cerca(parziale, budget);
            parziale.remove(parziale.size() - 1);
        }
    }
}
```

Figura 7: Metodi ricorsivi: doRicorsione(), cerca() – Model

Ovviamente sono state effettuate delle ottimizzazioni.
Tramite la *isValid()*, prima che venga inserito un libro nella lista viene controllato che inserendolo non si vada a sforare il budget.
Il controllo con il flag invece è stato inserito per ottimizzare ulteriormente l'algoritmo; infatti, se è già stato trovato un insieme di libri con il costo pari al budget, allora verranno validate solo altre liste il cui totale sia pari al budget. Esse verranno poi selezionate in base alle stelle medie.
Il metodo *totale()* viene utilizzato per calcolare velocemente il costo totale di una lista.

```
private boolean isValid(List<Libro> parziale, Libro li, float budget) {
    float totale = li.getPrezzo();
    for (Libro l : parziale) {
        totale += l.getPrezzo();
    }
    if (flag == 1) { // se siamo gia al budget desiderato e cerchiamo altre combinazioni magari con
                     // più stelle
        // se ne troviamo alcune che non sono esattamente uguali al budget le scartiamo
        // già qui
        if (totale != budget) {
            return false;
        }
    }
    if (totale <= budget && !parziale.contains(li)) {
        for (Libro l : libriParametri) {
            if (l.getTitolo().equals(li.getTitolo())) {
                if (isStronger(li)) {
                    return true;
                } else {
                    return false;
                }
            }

        }
        return true;
    }
    return false;
}

public float totale(List<Libro> toCalc) {
    float somma = 0;
    for (Libro l : toCalc) {
        somma += l.getPrezzo();
    }
    return somma;
}
```

Figura 8: Metodi isValid(), totale() – Model

Il metodo *isStronger()* viene usato in quanto lo stesso libro può essere presente più volte nella lista se esso è stato in top 100 per più di un anno. Tramite questo metodo viene preso solo una volta, quella in cui si trovava nella posizione più alta della classifica. In questo modo non viene mai generata una lista contenente due volte lo stesso libro ma di anni differenti.

```
public boolean isStronger(Libro li) {
    for (Libro l : libriParametri) {
        if (l.getTitolo().equals(li.getTitolo())) {
            if (l.getRank() < li.getRank()) {
                return false;
            }
        }
    }
    return true;
}
```

Figura 9: Metodo isStronger() –Model

Il metodo *bestSeller()* viene usato per stabilire quale libro all'interno di una lista è quello con il posizionamento in classifica più alto.
Il metodo *stelleMaggiori()* invece viene usato quando si hanno due liste di libri e si vuole sapere quale delle due abbia una media di stelle più elevata.

```
    public Libro bestSeller(List<Libro> toCalc) {
        Libro bestSeller = null;
        if (toCalc.size() > 0) {
            bestSeller = toCalc.get(0);
        } else {
            return null;
        }
        for (Libro l : toCalc) {
            if (l.getRank() < bestSeller.getRank()) {
                bestSeller = l;
            }
        }
        return bestSeller;
    }

    private boolean stelleMaggiori(List<Libro> best2, List<Libro> parziale2) {
        float mediaStelleBest = 0;
        float mediaStelleParz = 0;
        for (Libro l : best2) {
            mediaStelleBest += l.getRating();
        }
        for (Libro l : parziale2) {
            mediaStelleParz += l.getRating();
        }
        mediaStelleBest /= best2.size();
        mediaStelleParz /= parziale2.size();

        if (mediaStelleParz > mediaStelleBest) {
            return true;
        } else {
            return false;
        }

    }
}
```

Figura 10: Metodi bestSeller(), stelleMaggiori() – Model

Tutti i dati che vengono passati al **Model** vengono prima controllati dai metodi appartenenti all'**FXMLController**. Qui viene riportato un esempio del controllo eseguito sul budget e rating.

```
    void doGenera(ActionEvent event) {
        txtErroriGen.setVisible(false);
        txtErroriGen.clear();
        tableGenera.getItems().clear();
        float budget = 0;
        try {
            budget = Float.parseFloat(txtBudget.getText());
        } catch (NumberFormatException ex) {
            txtErroriGen.setVisible(true);
            txtErroriGen.appendText("Inserire un bugdet valido\n");
            return;
        }

        if (budget < 0) {
            txtErroriGen.setVisible(true);
            txtErroriGen.appendText("Inserire un bugdet maggiore di 0\n");
            return;
        }

        float rating = 0;
        try {
            rating = Float.parseFloat(txtRatingMin.getText());
        } catch (NumberFormatException ex) {
            txtErroriGen.setVisible(true);
            txtErroriGen.appendText("Inserire un rating valido\n");
            return;
        }

        if (rating < 0 || rating > 5) {
            txtErroriGen.setVisible(true);
            txtErroriGen.appendText("Inserire un rating compreso tra 0 e 5\n");
            return;
```

Figura 11: Parte dei controlli input metodo doGenera() - FXMLController

5 Diagramma delle classi delle parti principali

Il collegamento al database viene effettuato mediante la classe DBConnect. I dati sono ottenuti dal database attraverso la classe LibriDAO del package: *it.polito.tdp.FaudellaDavide.db*.

La classe **Libro** a cui fa riferimento il metodo *getAllBooks()* è stata fatta appositamente per immagazzinare tutte le informazioni ricevute dal database.
Al suo interno sono stati inoltre creati getter e setter degli attributi, così da poter disporre dei dati anche all'interno di una classe differente.

La classe **Model** è quella contenente i principali algoritmi, responsabili della logica del software.

Di seguito la sua composizione (omettendo i getter):

Model
List<Libro> daLeggere; List<Libro> best; List<Libro> parziale; List<Libro> libriParametri; List<Libro> tuttiLibri; LibriDAO dao;
public Model() **public** List<Libro> cercaLibri(String aut, int annoI, int annoF) **public** List<Libro> cercaLibriAutore(String aut) **public** List<Libro> cercaLibriAnno(**int** annoI, **int** annoF) **public boolean** aggiungi(Libro l) **public boolean** rimuovi(Libro l) **public void** cancellaLista() **public boolean** aggiungiBest(Libro l) **public void** cancellaBest() **public boolean** isStronger(Libro li) **public** List<Libro> doRicorsione(float budget, float rating, int numero, int annoI, int annoF, String copertina, String genere) **private void** cerca(List<Libro> parziale, float budget) **public boolean** isValid(List<Libro> parziale, Libro li, float budget) **public float** totale(List<Libro> toCalc) **public Libro** bestSeller(List<Libro> toCalc) **private boolean** stelleMaggiori(List<Libro> best2, List<Libro> parziale2)

Vi sono ovviamente altre classi essenziali per il funzionamento del software, qui sono solo presenti i metodi del **Model** che servono per gestire la parte logica. Ma essi, senza il contributo della classe **LibriDAO**, non avrebbero i dati necessari per funzionare, e senza l'**FXMLController** non avrebbero modo di ricevere i dati inseriti dall'utente tramite interfaccia grafica (con gli opportuni controlli di input).

6 Esempi di utilizzo dell'applicazione

6.1 Videata dell'applicazione

La schermata principale dell'applicativo si compone di due schede, selezionabili in alto a sinistra:

1) Scheda Ricerca
2) Scheda Genera Lista

Nella prima scheda (Ricerca) è possibile effettuare una ricerca di libri per Autore, per Anni, o con la combinazione dei due parametri. I risultati appariranno nella tabella "Libri trovati".
Inoltre è possibile aggiungere dei libri (tra quelli presenti in quel momento in "Libri trovati") in una lista di "Libri da leggere", dalla quale potranno essere poi rimossi una volta letti.

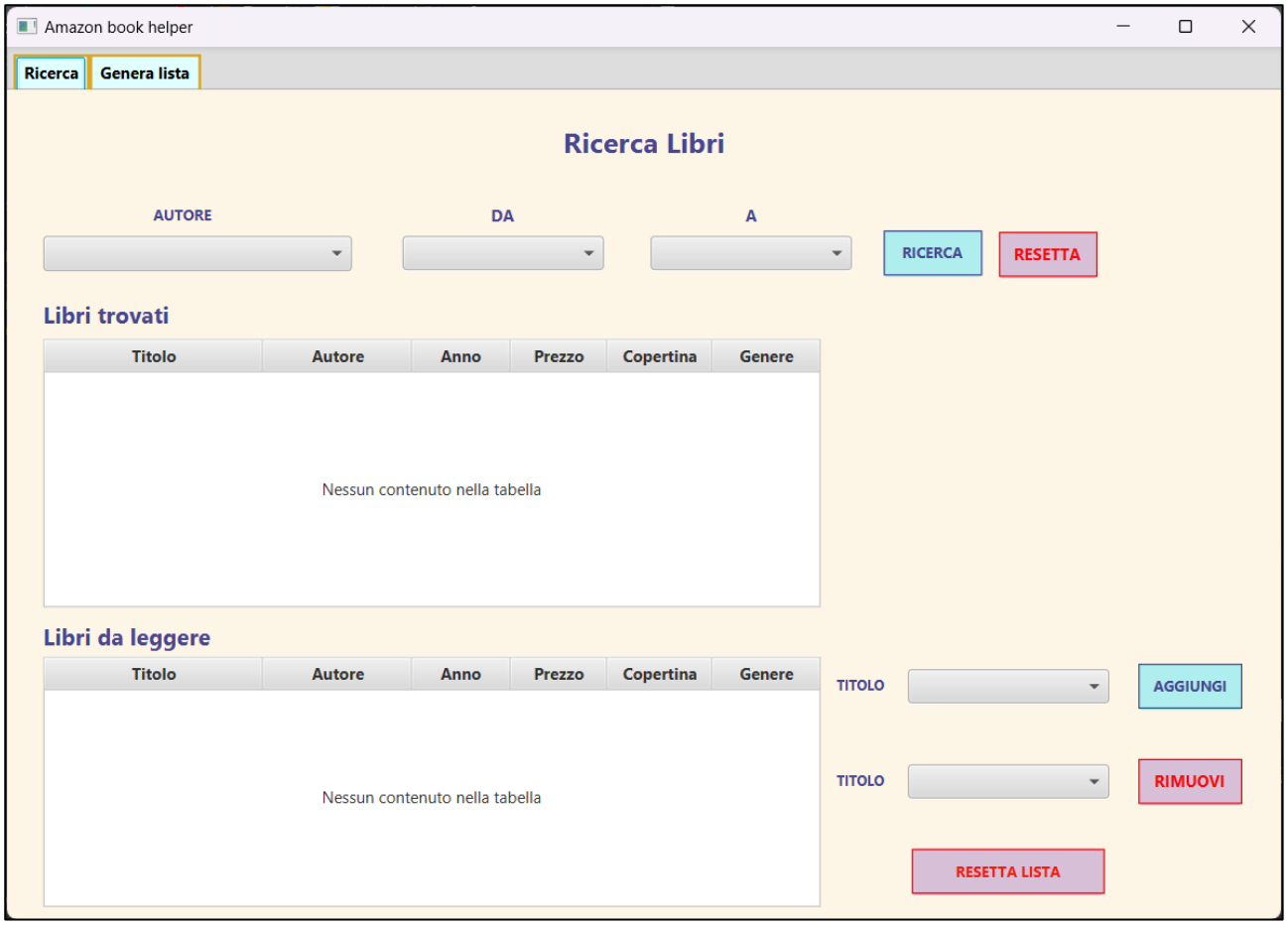

Figura 12: Schermata applicazione – Scheda Ricerca

Nella seconda scheda (Genera Lista), invece, sono presenti tutti i parametri da impostare per avviare (tramite il pulsante “GENERA”) la ricorsione. Nella parte superiore vi è la possibilità di aggiungere manualmente dei libri prima di far partire l’algoritmo ricorsivo.

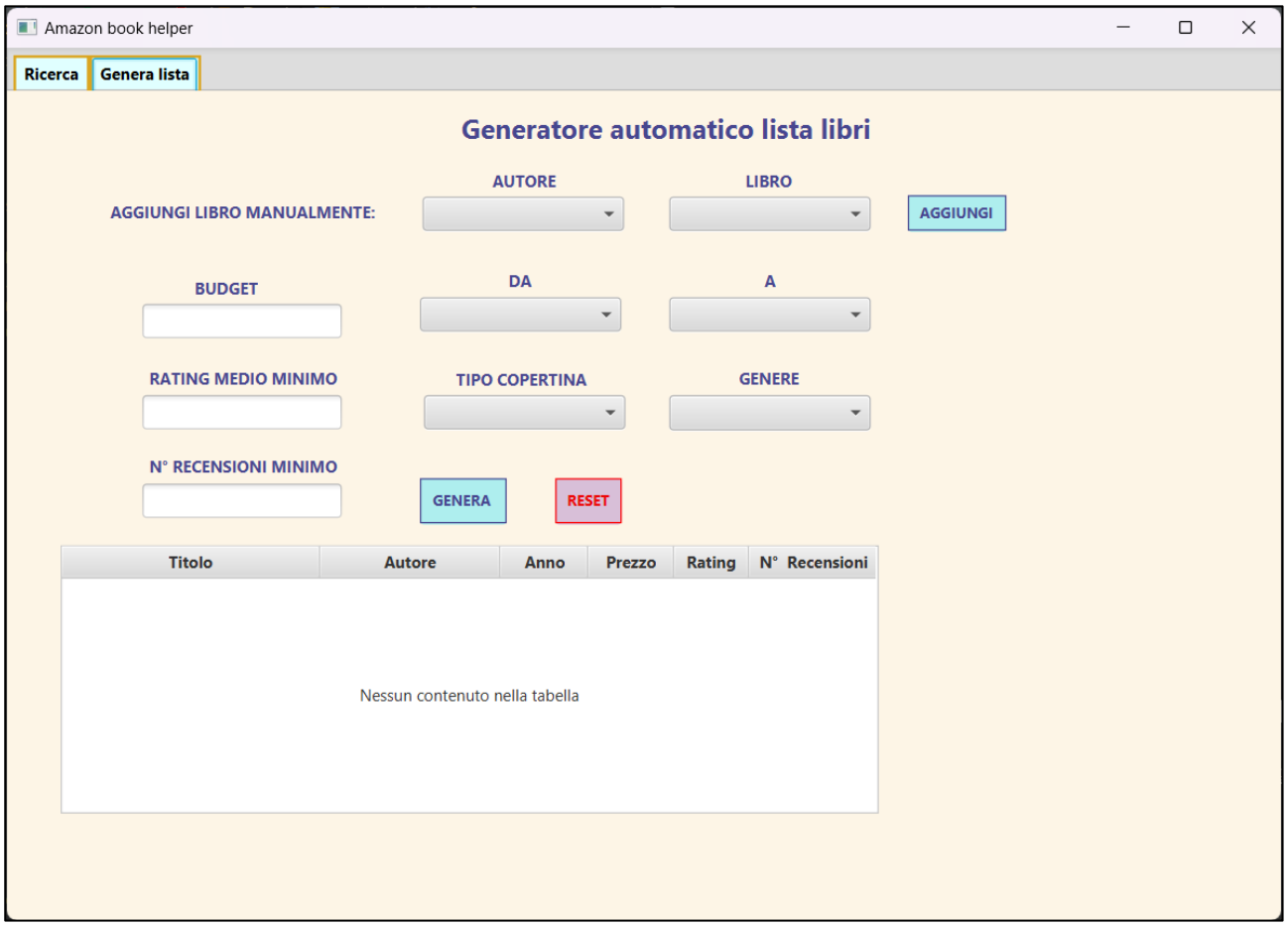

Figura 13 Schermata applicazione – Scheda Genera Lista

In entrambe le schede sono inoltre presenti delle aree di testo nascoste che vengono visualizzate solo in caso di errori di input o per segnalare problemi all’utente.

6.2 Video dimostrativo dell’applicazione software

È disponibile un video dimostrativo sul funzionamento del programma al seguente link:

https://youtu.be/Zs0TTHQrZe4

7 Risultati sperimentali ottenuti

Per quello che riguarda la parte della ricerca libri non ci sono osservazioni importanti da fare, in quanto riporta graficamente i risultati ottenuti dal database. Nell'immagine sottostante possiamo vedere una ricerca combinata con Autore e Anno che popola la prima tabella; la seconda, invece, è stata riempita a scopo dimostrativo.

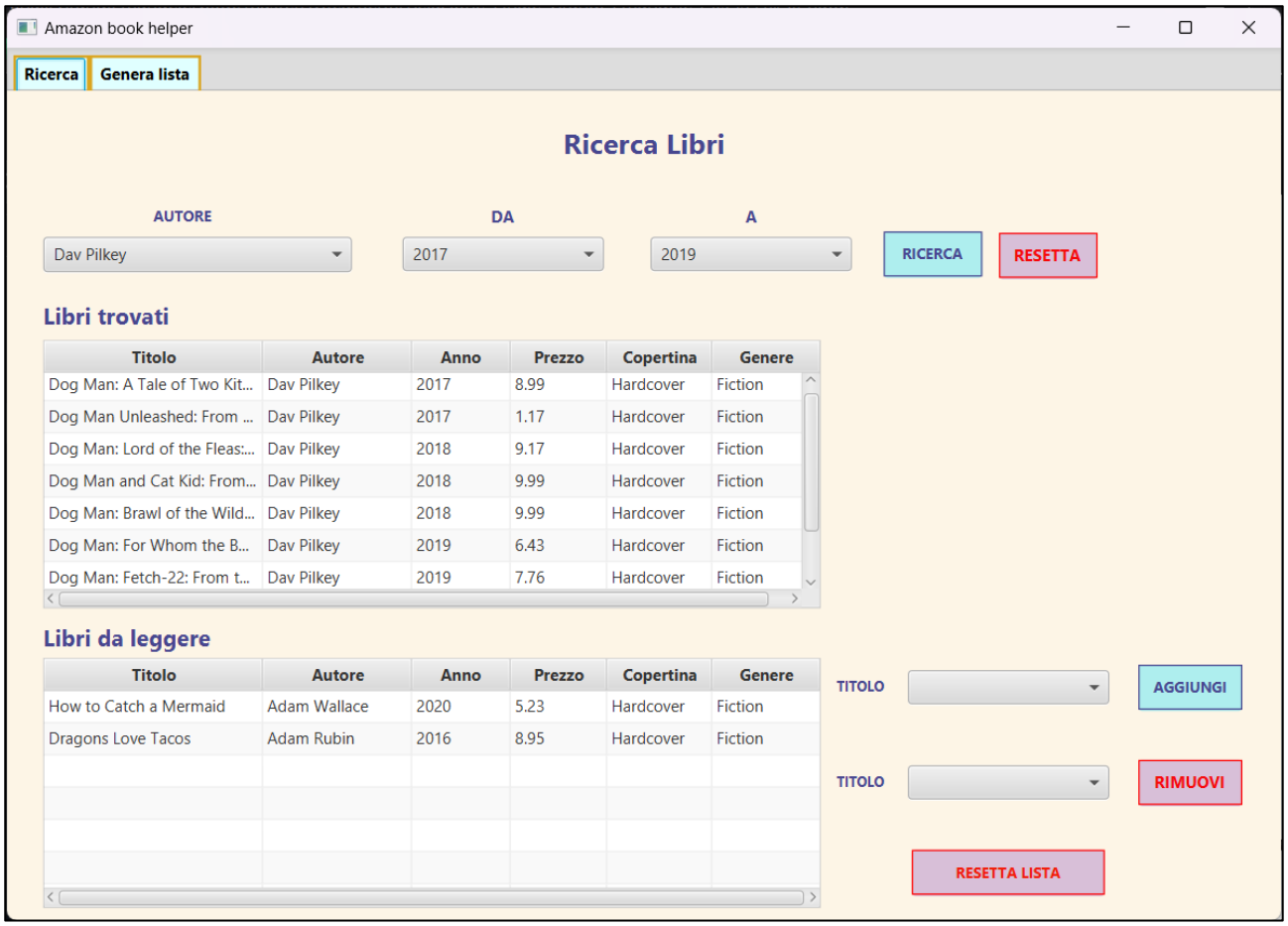

Figura 14 Esempio di ricerca

Si possono ovviamente utilizzare gli opportuni bottoni di reset per andare a pulire tutti i campi.

Per quanto riguarda la parte della ricorsione, invece, possiamo fare alcune osservazioni interessanti. L'algoritmo cerca sempre di portarci a spendere **esattamente** il budget da noi impostato; di conseguenza, spesso con budget ridotti non si troveranno combinazioni che ci portano alla cifra esatta al centesimo, in quanto si possono combinare pochi libri. Andando ad ampliare il budget invece il software è quasi sempre in grado di trovare una o più combinazioni perfette al centesimo, e tra queste poi seleziona quella con il rating medio più elevato.
Tutto questo chiaramente se ci sono abbastanza libri che rispettano i parametri selezionati, andando così a popolare la pool utilizzata dall'algoritmo. In caso di pool piccole ci si ritrova nella situazione in cui con poche combinazioni possibili è difficile trovare una soluzione perfetta.

In questo primo esempio possiamo notare che il costo totale è pari a 31,89 e si avvicina molto al budget di 31,9. Avendo un numero ristretto di combinazioni (per via del budget piccolo) l'algoritmo non ha trovato un insieme di libri che gli permettesse di raggiungere esattamente il budget.

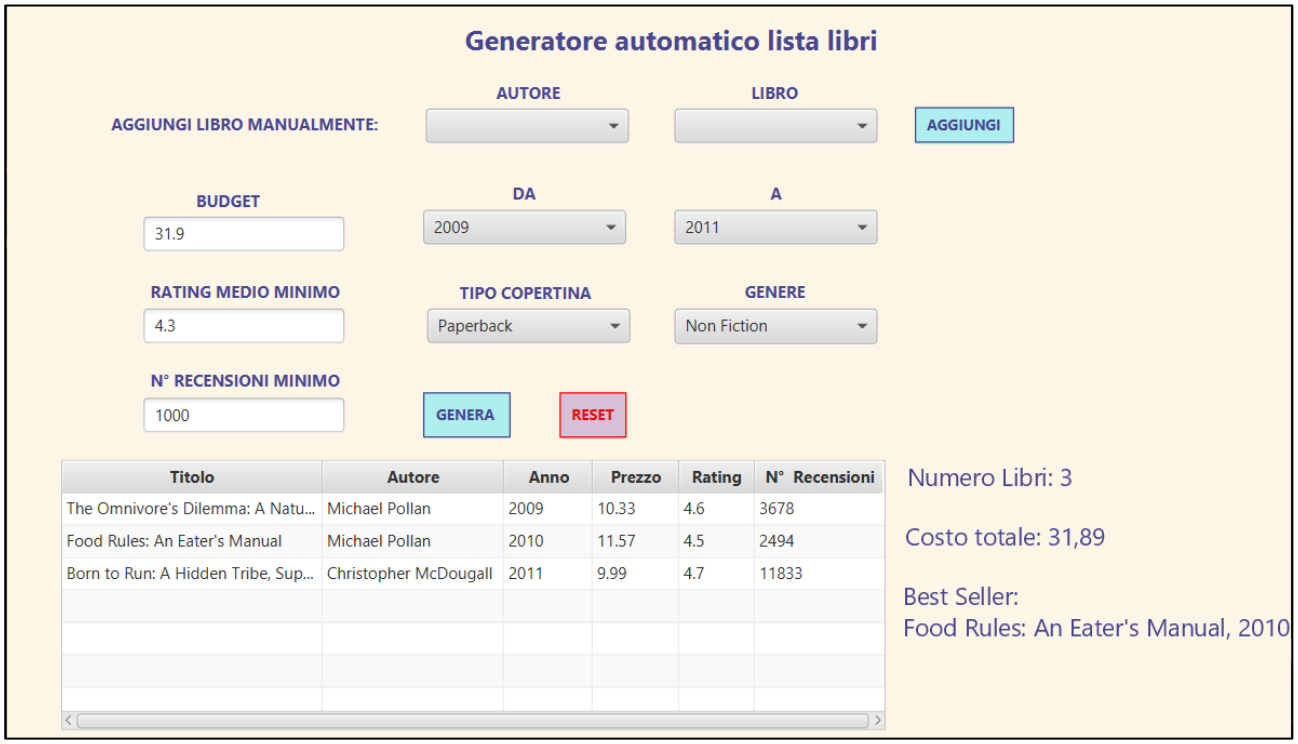

Figura 15 Esempio budget basso

Dopo aver aumentato un po' la disponibilità economica possiamo invece vedere come il costo totale sia esattamente identico al budget.

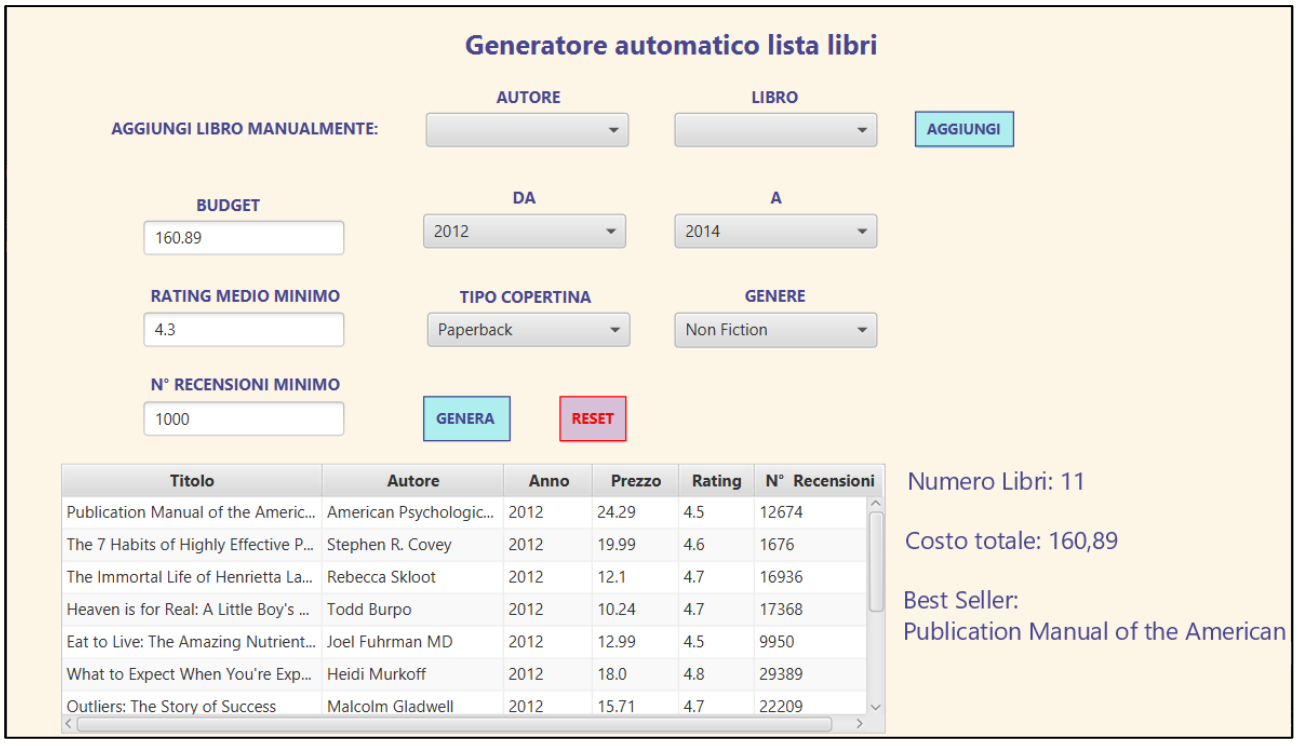

Figura 16 Esempio budget alto 1

Semplicemente modificando di due centesimi il budget la lista cambia e si trova un nuovo ottimo.

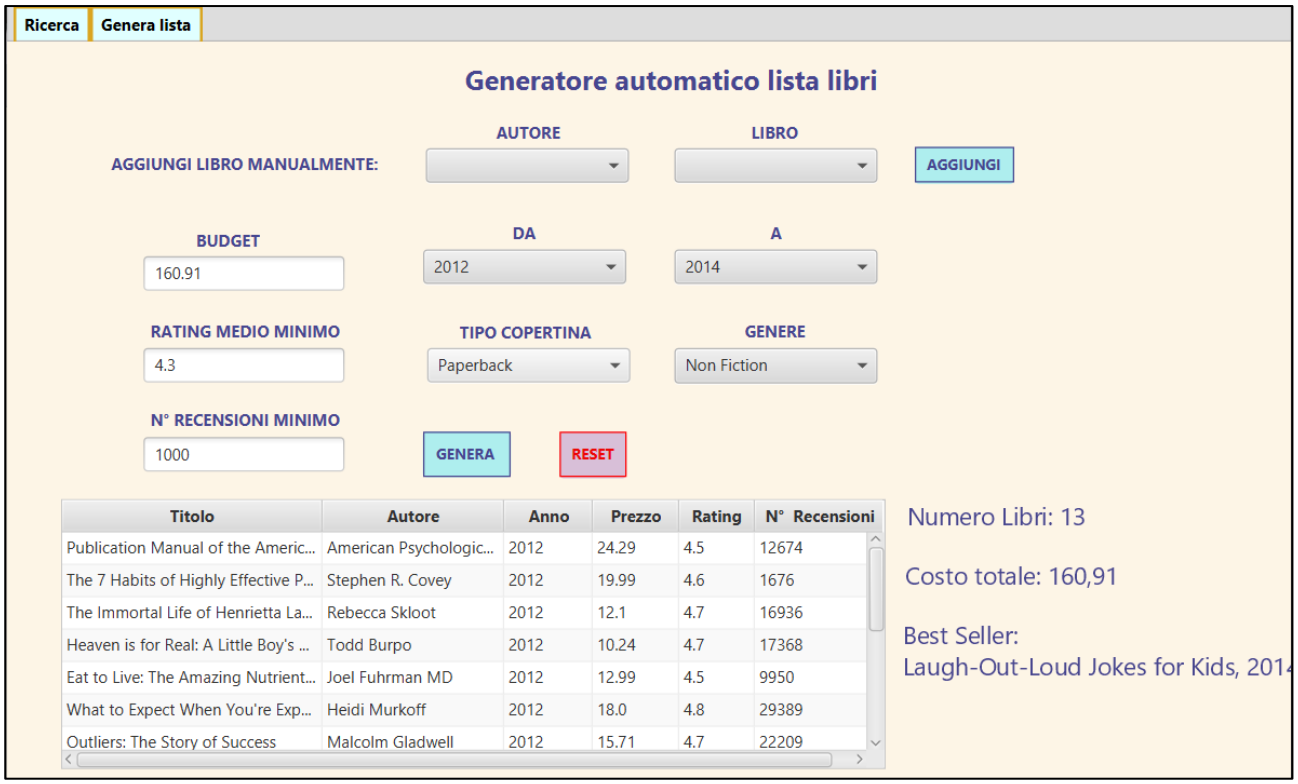

Figura 17 Esempio budget alto 2

Visto che la lista è a scorrimento non si vedono tutti i libri; di seguito vi è il confronto della parte non visibile nelle due immagini precedenti:

Titolo	Autore	Anno	Prezzo	Rating	N° Recensioni
Eat to Live: The Amazing Nutrient...	Joel Fuhrman MD	2012	12.99	4.5	9950
What to Expect When You're Exp...	Heidi Murkoff	2012	18.0	4.8	29389
Outliers: The Story of Success	Malcolm Gladwell	2012	15.71	4.7	22209
Crucial Conversations Tools for T...	Kerry Patterson	2012	20.97	4.7	9891
America the Beautiful: Rediscover...	Ben Carson M.D.	2013	10.61	4.8	3826
How to Win Friends & Influence ...	Dale Carnegie	2014	10.5	4.7	79094
Night (Night)	Elie Wiesel	2014	5.49	4.7	8573

Figura 18 Lista budget alto 1

Titolo	Autore	Anno	Prezzo	Rating	N° Recensioni
Outliers: The Story of Success	Malcolm Gladwell	2012	15.71	4.7	22209
To Heaven and Back: A Doctor's ...	Mary C. Neal	2012	9.29	4.5	4826
Forks Over Knives: The Cookbook...	Del Sroufe	2013	9.79	4.5	13075
Laugh-Out-Loud Jokes for Kids	Rob Elliott	2014	4.99	4.6	10751
Mindset: The New Psychology of ...	Carol S. Dweck	2014	8.55	4.6	13911
Night (Night)	Elie Wiesel	2014	5.49	4.7	8573
The Care and Keeping of You: Th...	Valorie Schaefer	2014	9.48	4.8	24552

Figura 19 Lista budget alto 2

Anche con cifre abbastanza importanti di budget, e quindi un numero di libri elevato, il programma riesce a trovare una combinazione ottimale e sempre col il rating medio più alto possibile.

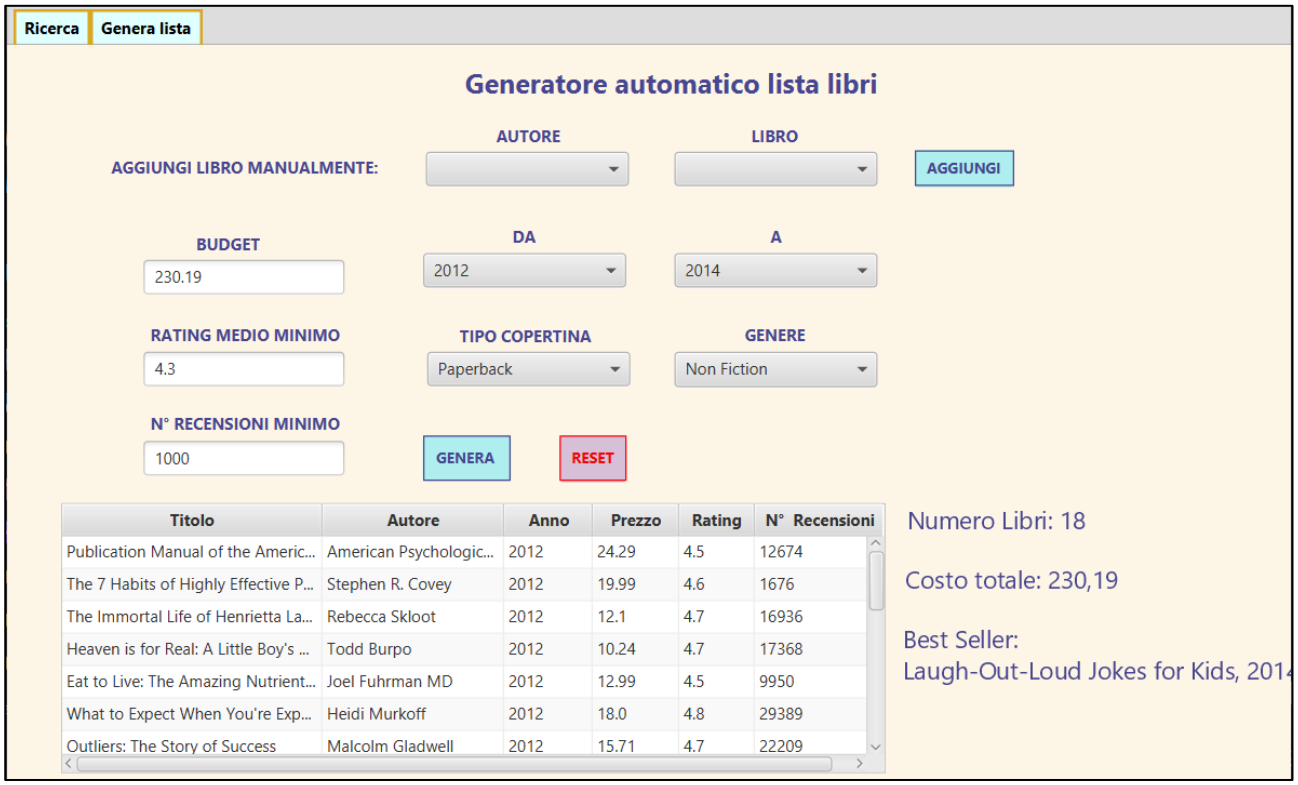

Figura 20 Esempio budget molto alto

8 Considerazioni finali

8.1 Valutazione dei risultati ottenuti

I risultati ottenuti sono ottimi, creare una lista di libri non è mai stato così semplice.
Possiamo avere liste di una ventina di libri generate in qualche millisecondo.
Un punto di debolezza, se così vogliamo chiamarlo, di questo software è la scalabilità, intesa come grandezza massima della lista creabile realmente. Infatti, a livello pratico, superati i 230 euro di budget il software rischia il crash poiché non riesce a trovare una soluzione ottima in quanto le combinazioni da controllare diventano troppe.

Questo è un software che ha molte possibilità di espansione a livello funzionale.
Una possibile implementazione futura potrebbe essere quella di andare a segnare i libri già letti, così che non vengano più proposti tra quelli da comprare alla fine dell'algoritmo ricorsivo.
Al momento, il limite di questo programma è dato dal database; in caso se ne avesse una versione continuamente aggiornata, con più dati e più libri, sicuramente si potrebbero aggiungere altre funzionalità. L'aggiunta del codice ISBN del libro, la data di uscita e informazioni simili andrebbero a rendere il programma più completo.
Anche solamente un'espansione del parametro "genere" (per esempio avendo Romanzi Gialli, Fantasy, Informatica, Economia, Psicologia etc. etc.) riuscirebbe a migliorare moltissimo la user experience dell'utente.

8.2 Conclusioni

Gli obiettivi del programma sono stati raggiunti. Il software permette di fare tutto ciò che era stato prefissato nella proposta di progetto.
Nonostante le grandi capacità di crescita del software, già al momento può essere ritenuto uno strumento utile e funzionale che va a risolvere un problema che fino ad ora era rimasto senza soluzione.