POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Gestionale

Classe L8 – Ingegneria dell’Informazione



**USA Travel Planner**

**Relatore Candidato**

Prof. Fulvio Corno Campana Riccardo

260448

A.A. 2022/2023

**Sommario**

[1 Proposta di progetto 1](#_Toc116049874)

[1.1 Studente proponente 1](#_Toc116049875)

[1.2 Titolo della proposta 1](#_Toc116049876)

[1.3 Descrizione del problema proposto 1](#_Toc116049877)

[1.4 Descrizione della rilevanza gestionale del problema 1](#_Toc116049878)

[1.5 Descrizione dei data-set per la valutazione 2](#_Toc116049879)

[1.6 Descrizione preliminare degli algoritmi coinvolti 4](#_Toc116049880)

[1.7 Descrizione preliminare delle funzionalità previste per l’applicazione software 5](#_Toc116049881)

[1.8 Modifiche, Correzioni, Integrazioni 5](#_Toc116049882)

[2 Descrizione dettagliata del problema affrontato 6](#_Toc116049883)

[3 Descrizione del data-set utilizzato per l'analisi 7](#_Toc116049884)

[3.1 Coupon 8](#_Toc116049885)

[3.2 Ticket 9](#_Toc116049886)

[3.3 Airport 9](#_Toc116049887)

[3.4 AirBnB 10](#_Toc116049888)

[4 Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati 11](#_Toc116049889)

[4.1 Pacchetti e Classi 11](#_Toc116049890)

[4.1.1 it.polito.tdp.TravelManager 12](#_Toc116049891)

[4.1.2 it.polito.tdp.TravelManager.db 13](#_Toc116049892)

[4.1.3 it.polito.tdp.TravelManager.model 13](#_Toc116049893)

[4.2 Algoritmi 15](#_Toc116049894)

[4.2.1 Creazione del grafo 15](#_Toc116049895)

[4.2.2 Ricerca dei voli ricorsiva 16](#_Toc116049896)

[4.2.3 Ricerca degli AirBnB ricorsiva 17](#_Toc116049897)

[5 Diagramma delle classi principali 20](#_Toc116049898)

[6 Interfaccia utente 21](#_Toc116049899)

[7 Risultati sperimentali 26](#_Toc116049900)

[8 Valutazioni finali 31](#_Toc116049901)

# Proposta di progetto

## Studente proponente

s260448 Campana Riccardo

## Titolo della proposta

USA Travel Planner

## Descrizione del problema proposto

Programmare un viaggio richiede sempre molto tempo, perché ci si vuole assicurare di poter ottenere l’esperienza migliore in relazione al budget personale. Solitamente, la ricerca dura più settimane per osservare l’oscillazione dei prezzi e si cerca di prenotare il più presto possibile per mantenere il costo totale basso.   
L’obiettivo di questa applicazione è offrire all’utente una lista di voli e sistemazioni che rispettino i vincoli inseriti per poter rendere semplice, intuitiva e veloce la prenotazione di un viaggio simulato nel primo quarto del 2022. Oltre alla ricerca ottimizzata di un volo aereo, se la città di destinazione è una delle sei metropoli supportate, l’applicazione permette anche la ricerca ottimizzata di una sistemazione AirBnB secondo le preferenze dell’utente.

## Descrizione della rilevanza gestionale del problema

L’interesse dal punto di vista gestionale si trova nel min-maxing parametrico che rispetti i vincoli inseriti dall’utente, oltre che nella semplificazione e automatizzazione di un processo che altrimenti sarebbe lungo e tedioso. Fornendo il data set adatto, con prezzi previsti tramite algoritmi statistici e voli programmati per il futuro, l’applicazione potrebbe essere a tutti gli effetti operativa per la scelta delle opzioni migliori, indirizzando l’utente e risparmiandogli ore di ricerca. In caso di dati dinamici, come sono effettivamente i prezzi di voli e hotel, sarebbe anche interessante usare l’applicazione per studiare le tendenze e individuare il momento migliore per l’acquisto.

## Descrizione dei data-set per la valutazione

La struttura del data-set che ho creato è la seguente:

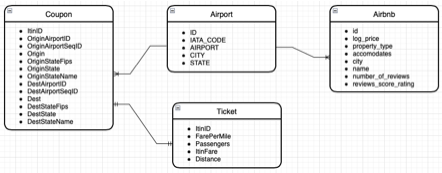


Figura 1. Diagramma ER del database iniziale

Per quanto riguarda la ricerca dei voli, ho utilizzato dati presi dall’Airline Origin and Destination Survey (DB1B), un data-set fornito direttamente dal United States Department of Transportation, che raccoglie informazioni sul 10% dei voli con aeroporto di partenza e aeroporto di atterraggio negli USA dal 1993 ad oggi.   
Date le dimensioni, ho estratto le informazioni relative al primo quarto del 2022, per poter avere una rappresentazione realistica di tratte e prezzi.

La tabella Coupon contiene circa nove milioni di righe e comprende le informazioni su aeroporto di partenza e destinazione dei voli. Queste sono le colonne che ho deciso di utilizzare tra quelle offerte dalla fonte:

Tabella 1. Tabella Coupon

|  |  |
| --- | --- |
| ItinID | Itinerary ID |
| OriginAirportID | Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused. |
| OriginAirportSeqID | Origin Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time. |
| Quarter | Quarter (1-4) |
| Origin | Origin Airport Code |
| OriginStateFips | Origin Airport, State FIPS Code |
| OriginState | Origin Airport, State Code |
| OriginStateName | Origin State Name |
| DestAirportID | Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused. |
| DestAirportSeqID | Destination Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time. |
| Dest | Destination Airport Code |
| DestStateFips | Destination Airport, State FIPS Code |
| DestState | Destination Airport, State Code |
| DestStateName | Destination State Name |

La tabella Ticket, facente parte dello stesso data-set originale di Coupon, contiene quasi quattro milioni di righe e contiene informazioni sui prezzi dei biglietti aerei. Queste sono le colonne che ho deciso di utilizzare tra quelle offerte dalla fonte:

Tabella 2. Tabella Ticket

|  |  |
| --- | --- |
| ItinID | Itinerary ID |
| FarePerMile | Itinerary Fare Per Miles Flown in Dollars (ItinFare/MilesFlown) |
| Passengers | Number of Passengers |
| ItinFare | Itinerary Fare Per Person |
| Distance | Itinerary Distance (Including Ground Transport) |

La tabella Airport è utile per la corrispondenza aeroporti-città. Ho estratto la tabella “airports” dal database dell’esercitazione “exflightdelays” svolta durante il corso di tecniche di programmazione, che contiene circa trecento righe. Queste sono le colonne che ho deciso di utilizzare tra quelle offerte dalla fonte:

Tabella 3. Tabella Airport

|  |  |
| --- | --- |
| Id | Airport Id |
| IATA\_Code | Airport IATA code |
| Airport | Airport name |
| City | Airport city |
| State | Airport state code ISO 3166-2 |
| StateName | Airport state name |

Per quanto riguarda la tabella AirBnB, data l’impossibilità di reperire un data-set soddisfacente che coprisse l’intero territorio degli USA, ho optato per una più vasta offerta di alloggi a discapito del numero di città supportate. Le città supportate sono sei tra le più grandi degli Stati Uniti: New York, Los Angeles, San Francisco, Washington D.C., Chicago e Boston.

Le informazioni relative agli AirBnB provengono da Kaggle. L’utente ha partecipato a una competizione il cui obiettivo era prevedere il costo di diverse offerte presenti su AirBnB nelle maggiori città statunitensi. Data la finalità di studio e previsione della colonna log\_price, i valori non possono essere ritenuti reali, ma sono abbastanza verosimili da permetterne l’utilizzo per la mia applicazione. La tabella contiene circa duecentotrenta mila righe.

Queste sono le colonne che ho deciso di utilizzare tra quelle offerte dalla fonte.

Tabella 4. Tabella AirBnB

|  |  |
| --- | --- |
| Id | Listing id |
| Log\_price | Natural logarithm of the predicted price |
| Property\_type | Property\_type |
| Accommodates | Number of people the property can accommodate |
| City | City |
| Name | Name of the listing |
| Number\_of\_reviews | Number of reviews |
| Reviews\_score\_rating | Rating of the listing calculated from the reviews |

## Descrizione preliminare degli algoritmi coinvolti

Intendo creare un grafo semplice orientato e pesato con gli aeroporti come vertici e i voli come archi, utilizzando il costo del volo per il peso.  
Il passo successivo sarà creare uno o più algoritmi ricorsivi di visita per raggiungere la destinazione, mirati ad ottenere il volo o una lista di essi che rispetti i parametri inseriti e sia la soluzione migliore.   
Se la destinazione fa parte delle sei città supportate, un successivo algoritmo ricorsivo scorrerà la lista degli AirBnB presenti nella città di arrivo e sarà utilizzato per la ricerca dell’hotel più adatto alla richiesta.

## Descrizione preliminare delle funzionalità previste per l’applicazione software

Agli occhi dell’utente, l’interfaccia offrirà la scelta di una città di partenza e una di arrivo, seguita da un’area dove inserire il costo massimo del viaggio e una dove specificare il numero massimo di scali. In una tabella sottostante sarà visualizzato il risultato, composto da uno o più voli.   
Sarà specificato che le funzionalità riguardanti gli AirBnB verranno abilitate solo se la città di arrivo farà parte delle sei presenti nel database.  
In caso la città di arrivo sia una di quelle supportate, l’interfaccia abiliterà i campi successivi, dove inserire le preferenze sulle sistemazioni da ricercare. Saranno proposti vincoli sul prezzo, sul tipo di proprietà, sulla quantità̀ di persone che può ospitare la casa, sul numero di recensioni e sul rating del locale. Una tabella mostrerà le sistemazioni risultato della ricerca.

## Modifiche, Correzioni, Integrazioni

Le dimensioni significative del data-set creato e la volontà di facilitare le query in corso d’opera hanno portato a diversi cambiamenti della struttura del data-set stesso. Le informazioni utilizzate sono rimaste le medesime, ma le colonne inutili o ridondanti sono state eliminate.  
La struttura finale del data-set è illustrata al [punto 3](#_Descrizione_del_data-set).

Oltre ai precedenti cambiamenti, sono state aggiunte manualmente circa cento righe alla tabella Airport. Le informazioni contenute nelle righe aggiunte sono state estratte in parte dalla tabella Coupon e in parte dalle pagine Wikipedia degli aeroporti interessati.

Algoritmicamente, non ci sono stati cambiamenti rispetto al programma iniziale, oltre a una leggera modifica nella creazione del grafo. Gli archi previsti erano i voli, ma ciò non si è rivelato possibile a causa di un problema concettuale. È infatti permessa l’esistenza di un solo arco con specifica origine e destinazione; Questo crea conflitto con l’idea di avere tutti i voli come archi e obbliga la ricerca di una soluzione alternativa. Nella versione finale del programma, tale problema è stato affrontato facendo la media aritmetica dei prezzi di tutti i voli con stesse origini e destinazioni, e assegnandola a voli fittizi che rappresentassero verosimilmente la situazione descritta dal database.

L’interfaccia ha guadagnato un’aggiunta non prevista, ma ritenuta utile ai fini del progetto: è stato reso possibile selezionare da entrambe le tabelle la riga a cui si è più interessati, visualizzando nella zona inferiore destra della finestra il totale in dollari delle proprie scelte. Inoltre, cambiando la selezione delle righe, il totale si aggiornerà in automatico.

# Descrizione dettagliata del problema affrontato

L’applicazione affronta un comune problema di ricerca applicato a un contesto conosciuto da moltissime persone. L’idea iniziale nasce dall’esperienza personale, spesso frustrante, che si ripete annualmente per prenotare le vacanze, ma il programma si rivela abbastanza duttile da poter essere applicato in ambiti diversi a seconda del data-set che viene fornito. Può essere utilizzato dalle aziende per programmare viaggi di lavoro, o da servizi di spedizione nazionali per gestire le consegne. Si rende evidente che le potenzialità di questa applicazione sono offerte dalla vasta scelta di input possibili e dalle poche modifiche sul codice che richiederebbero. Il necessario per rendere effettivamente operativo il software si riduce a un data-set reale, come può essere quello di siti dedicati alle prenotazioni, per esempio SkyScanner o Booking. Dopo una tale aggiunta, l’applicazione potrebbe completamente sostituire la ricerca sui suddetti siti e lasciare a loro soltanto la procedura di prenotazione e pagamento.

Contestualmente alla ricerca di soluzioni possibili, il programma affronta anche il sotto-problema della soluzione migliore, ritenuta per scelta quella a prezzo minore.

Gli output e la conseguente interfaccia grafica sono mirati alla semplicità di utilizzo e al mettere in evidenza le informazioni più utili ai fini della scelta; entrambe le tabelle dei risultati sono infatti ordinate per prezzo crescente. È possibile cambiare l’ordinamento delle tabelle cliccando sul parametro a cui più si è interessati, una feature importante soprattutto per la sezione dedicata agli AirBnB, vista la possibilità di visionare il rating ed il numero di recensioni delle sistemazioni offerte.

Questo tipo di applicazione mostra la sua utilità quando sono presenti sufficienti filtri di ricerca. Le scelte dei filtri erano chiare già durante la prima proposta di progetto e sono rimaste solide per tutta la durata dello sviluppo. Per quanto sia ricca un’interfaccia piena di filtri, è critico capire quali sono i parametri realmente importanti per la decisione. Una presenza esagerata di possibilità può anche portare l’utente a confondersi, per questo motivo non sono stati utilizzate informazioni del data-set come il numero di passeggeri dei voli, o il prezzo al miglio, che però sono rimaste disponibili in caso di necessità di studio futura.

Appurate le potenzialità di uso del software in caso di data-set reali, emergono le possibilità che offre utilizzando data-set ottenuti da previsioni statistiche. Vista la dinamicità dei dati, il programma potrebbe essere utilizzato per studiare le oscillazioni dei prezzi dei voli o la correlazione tra rating di un AirBnB ed il suo quartiere di appartenenza, oppure per valutare il traffico aereo attorno agli aeroporti. Hosts alle prime armi potrebbero sfruttare il software per capire che prezzo assegnare alle loro residenze, osservando annunci simili alla loro ipotetica offerta.

# Descrizione del data-set utilizzato per l'analisi

Vista l’importanza del data-set per il progetto, la preparazione preliminare precedente alla proposta di progetto è stata particolarmente attenta e approfondita. Le informazioni che seguono sono molto simili a quelle della proposta iniziale, ma presentano le modifiche apportate e descritte al [punto 1.8](#_Modifiche,_Correzioni,_Integrazioni).

Le tabelle Coupon e Ticket derivano dall’Airline Origin and Destination Survey (DB1B), un data-set fornito dal United States Department of Transportation che raccoglie informazioni sul 10% dei voli con aeroporto di partenza e aeroporto di atterraggio negli USA dal 1993 ad oggi.   
Le due tabelle contengono informazioni relative al primo quarto del 2022.

La base per la tabella Airport proviene dall’esercitazione “exflightdelays” svolta durante il corso di tecniche di programmazione. La sua struttura è stata leggermente modificata secondo le necessità del codice e circa cento righe sono state aggiunte manualmente. Le informazioni relative agli aeroporti aggiunti sono state reperite da Wikipedia.

La tabella AirBnB contiene informazioni relative agli AirBnB di sei tra le più grandi città degli Stati Uniti: New York, Los Angeles, San Francisco, Washington D.C., Chicago e Boston. Le informazioni relative agli AirBnB provengono dal lavoro di un utente di Kaggle che ha partecipato a una competizione basata sulla previsione del prezzo di diverse offerte presenti su AirBnB nelle maggiori città statunitensi. Data la finalità di studio e previsione della competizione, i valori sono verosimili, ma non reali.

La struttura finale del data-set è la seguente:

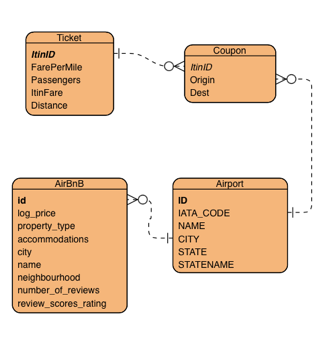


Figura 2. Diagramma ER finale

## Coupon

La tabella Coupon ha subito il maggior numero di cambiamenti, poiché molte delle sue informazioni sono state trasferite nella tabella Airports per comodità. Coupon contiene quasi nove milioni di righe.

Nella sua versione definitiva si presenta con la seguente struttura:



Figura 3. Struttura della tabella Coupon. Screenshot di Sequel Pro

Dal [punto 1.5](#_Descrizione_dei_data-set):

Tabella 5. Tabella Coupon finale

|  |  |
| --- | --- |
| ItinID | Itinerary ID |
| Origin | Origin Airport Code |
| Dest | Destination Airport Code |

## Ticket

La tabella Ticket è rimasta invariata durante lo sviluppo: Contiene quasi quattro milioni di righe e si presenta con la seguente struttura:

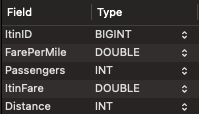


Figura 4. Struttura della tabella Ticket. Screenshot di Sequel Pro

Dal [punto 1.5](#_Descrizione_dei_data-set):

Tabella . Tabella Ticket finale

|  |  |
| --- | --- |
| ItinID | Itinerary ID |
| FarePerMile | Itinerary Fare Per Miles Flown in Dollars (ItinFare/MilesFlown) |
| Passengers | Number of Passengers |
| ItinFare | Itinerary Fare Per Person |
| Distance | Itinerary Distance (Including Ground Transport) |

## Airport

La tabella Airport ha visto l’aggiunta della colonna stateName, utile per convertire i codici dei vari stati degli USA nel loro corrispondente nome esteso. Contiene circa quattrocentocinquanta righe e la sua struttura finale è la seguente:

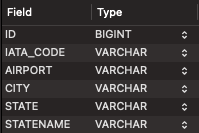


Figura 5. Struttura della tabella Airport. Screenshot di Sequel Pro

Dal [punto 1.5](#_Descrizione_dei_data-set):

Tabella . Tabella Airport finale

|  |  |
| --- | --- |
| Id | Airport Id |
| IATA\_Code | Airport IATA code |
| Airport | Airport name |
| City | Airport city |
| State | Airport state code ISO 3166-2 |
| StateName | Airport state name |

## AirBnB

La tabella AirBnB non ha subito cambiamenti dalla sua versione iniziale. Contiene più di settantamila righe e la sua struttura è la seguente:



Figura 6. Struttura della tabella AirBnB. Screenshot di Sequel Pro

Dal [punto 1.5](#_Descrizione_dei_data-set):

Tabella . Tabella AirBnB finale

|  |  |
| --- | --- |
| Id | Listing id |
| Log\_price | Natural logarithm of the predicted price |
| Property\_type | Property type |
| Accommodates | Number of people the property can accommodate |
| City | City |
| Name | Name of the listing |
| Number\_of\_reviews | Number of reviews |
| Reviews\_score\_rating | Rating of the listing calculated from the reviews |

# Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

L’applicazione è stata sviluppata in JavaFX seguendo i pattern MVC e DAO. La scrittura di programmi che rispettano i pattern permette non solo la prevenzione di problemi e criticità ricorrenti, ma anche la possibilità di modificare parti di codice senza influenzarne altre.

Il pattern MVC divide l’applicazione in tre componenti: *View*, cioè l’interfaccia grafica, *Model*, ovvero la parte di codice che contiene dati e logica applicativa, e *Controller*, che si occupa di interpretare le richieste dell’utente rispetto a ciò che contiene il Model.

Il pattern DAO (Data Access Object) suggerisce la creazione classi apposite per l’accesso e l’estrazione di dati dal database. L’obiettivo è evitare di mischiare informazioni di basso livello con il codice dell’applicazione principale, rilegando tutto ciò che riguarda query SQL e struttura del database in classi apposite.

## Pacchetti e Classi

L’applicazione si compone di tre pacchetti che rispecchiano i pattern sopra descritti.

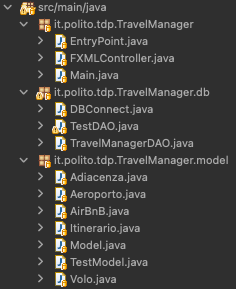


Figura 7. Struttura dei Packages. Screenshot di Eclipse

### it.polito.tdp.TravelManager

Il pacchetto contiene tre classi che si occupano dell’avvio dell’applicazione e dell’interfaccia grafica.

**EntryPoint**: necessario per l'avvio dell'applicazione e parte fondante del pattern MVC.

**Main**: Classe di avvio dell’applicazione.

**FXMLController**: La classe più colma di codice del pacchetto, contiene infatti tutte le informazioni riguardanti l’interfaccia utente ed i metodi per trasferire le richieste dell’utilizzatore al *Model*. Dopo aver chiamato i metodi del *Model*, il *Controller* si occupa anche di inserire gli output nell’interfaccia utente.   
L’interfaccia utente, gestita dal [*Controller*](#_Descrizione_ad_alto), è stata sviluppata su SceneBuilder, un software apposito dell’ecosistema JavaFX.

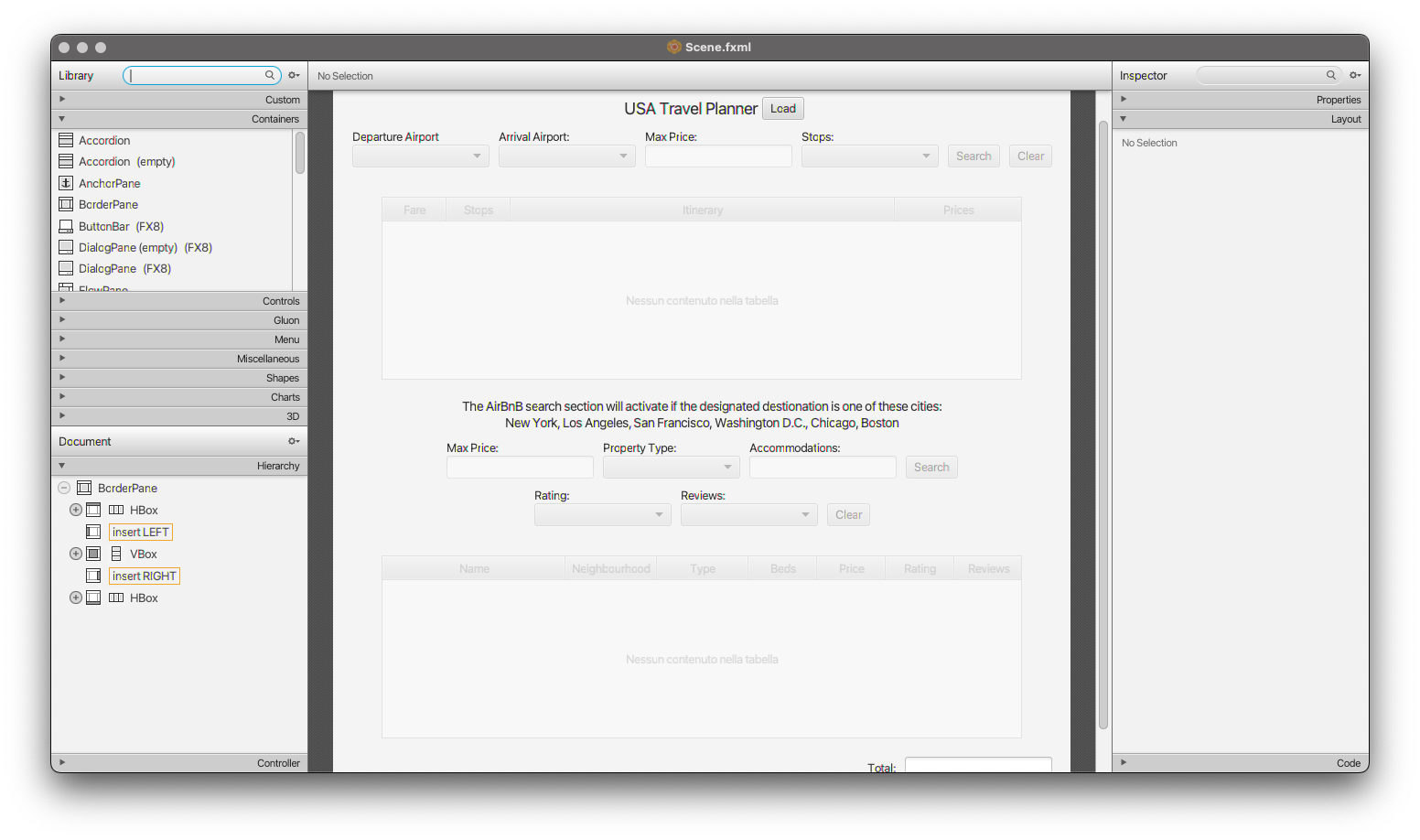


Figura 8. Creazione dell'interfaccia grafica. Screenshot di SceneBuilder

### it.polito.tdp.TravelManager.db

Il pacchetto contiene tre classi e si occupa di connessione al database e susseguente estrazione dei dati in esso contenuti.

**DBConnect:** Sfrutta la libreria HikariDataSource per gestire le connessioni con il database. Contiene i dati di accesso al proprio DBMS.

**TestDAO:** Una classe di test, creata per poter verificare che i metodi di TravelManagerDAO siano funzionanti e per poterne valutare il tempo di processo. Effettivamente inutile ai fini dell’utente utilizzatore finale, ma utile in caso vengano aggiunti nuovi metodi da testare.

**TravelManagerDAO:** Contiene metodi che, tramite query SQL, estraggono informazioni dal database e le convertono in strutture dati tipiche di Java, come liste e mappe.

### it.polito.tdp.TravelManager.model

Il pacchetto contiene sette classi e rispecchia il [Model](#_Descrizione_ad_alto) del pattern MVC.

**Adiacenza:** È una struttura dati temporanea utilizzata per le informazioni estratte da TravelManagerDAO. Il suo scopo finale è fornire i dati necessari alla creazione degli archi del grafo: la sua struttura infatti riporta un’origine, una destinazione e un costo. Contiene un costruttore, metodi *getter* e *setter* per accedere agli attributi privati e metodi *hashCode*, *equals* e *toString*, rispettivamente per verificare l’uguaglianza tra due Adiacenze (i primi due) e per tradurre in stringa le informazioni contenute nell’istanza di classe (l’ultimo).

**Aeroporto:** Struttura e raccoglie le informazioni relative agli aeroporti. Contiene attributi riguardanti Id, IATA\_code, nome, città, codice dello stato e nome dello stato di appartenenza. Trova il suo maggiore utilizzo nella creazione del grafo, i cui vertici sono aeroporti. Contiene un costruttore, metodi *getter* e *setter*, *hashCode*, *equals* e *toString*.

**AirBnB:** È una trasposizione quasi identica della tabella AirBnB presente nel data-set. I due attributi aggiuntivi sono prezzo e prezzoS, rispettivamente il valore esponenziale di log\_price e lo stesso valore in versione stringa, con l’aggiunta di un segno $ precedente alla cifra. Contiene un costruttore, metodi *getter* e *setter*, *hashCode*, *equals* e *toString*. È inoltre presente un metodo *compareTo* per l’ordinamento di liste di AirBnBs.

**Itinerario:** È utilizzata per rappresentare l’intera tratta percorsa, presenta infatti una lista di aeroporti visitati, oltre che una lista di archi attraversati, sotto forma di adiacenze. Ulteriori attributi di comodità sono l’aeroporto di partenza, quello di arrivo e quattro stringhe che rappresentano il viaggio ed i suoi costi. Contiene un costruttore, metodi *getter* e *setter*, *hashCode*, *equals*, *toString* e *compareTo*.

**Model:** È il cuore del package, rappresentante il [*Model*](#_Descrizione_ad_alto) del pattern MVC. Contiene tutti i metodi operativi di costruzione del grafo, caricamento delle strutture dati, visita ricorsiva del grafo e ricerca ricorsiva di un AirBnB, oltre a metodi supplementari, tra cui *getters* e metodi aggregati a quelli ricorsivi, utilizzati per la valutazione graduale delle possibilità. Questa classe si occupa anche di inserire in apposite strutture dati le informazioni estrapolate dal database chiamando i metodi di TravelManagerDAO.

**TestModel:** Come TestDAO, si tratta di una classe di test inutile ai fini di utilizzo, ma comoda in fase di sviluppo.

**Volo:** La classe Volo si sarebbe occupata di strutturare le informazioni relative ai voli presenti nei database, ma a causa di scelte di sviluppo non è utilizzata nella versione finale dell’applicazione. Il suo ruolo è stato ricoperto dalla classe Adiacenza, in quanto anche lo scopo originale di Volo era riempire la lista di archi del grafo.   
La classe contiene attributi relativi alle tabelle Coupon e Ticket, ovvero un Id, origine e destinazione, prezzo e prezzo al miglio, numero di passeggeri e distanza coperta dal volo. Disponeva di metodi *getter*, *setter*, *HashCode*, *equals* e *toString*.

## Algoritmi

L’applicazione ottiene i suoi risultati tramite l’utilizzo dei metodi ricorsivi, cioè una funzione che al suo interno richiama sé stessa, passandosi i parametri iniziali aggiornati. Ogni chiamata ricorsiva è definita come livello di profondità. Questo tipo di approccio permetto la suddivisione del problema iniziale in una serie di sotto-problemi, a loro volta poi scomposti. Il processo porta ad affrontare tanti problemi di difficoltà banale anziché un unico problema complesso.   
L’interesse algoritmico risiede interamente nel Model, dove gli algoritmi ricorsivi si occupano di portare alla luce le soluzioni migliori secondo i criteri scelti.

### Creazione del grafo

Il metodo *creaGrafo* si incarica di creare un grafo semplice, orientato e pesato, i cui vertici sono istanze della classe aeroporto, mentre gli archi contengono informazioni prelevate da istanze della classe Adiacenza.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 9. Metodo creaGrafo. Screenshot di Eclipse

Come è possibile osservare nella figura 9, il metodo non riceve parametri in entrata e si divide in tre fasi. Nelle sue prime righe inizializza il grafo, per poi inserire in esso tutti i vertici, contenuti nella variabile globale *airports*. Un’ultima sezione del metodo aggiunge, tramite un ciclo for, una serie di archi creati a partire dalle informazioni contenute nella variabile globale adiacenze.  
Sotto forma di commento sono ancora presenti i precedenti algoritmi utilizzati per l’aggiunta degli archi al grafo, che si sono rivelati però troppo lenti.

### Ricerca dei voli ricorsiva

La ricerca dei voli contestuale alla prima parte del progetto è gestita da tre metodi, di cui due privati.  
Immagine che contiene testo

Descrizione generata automaticamente

Figura 10. Metodo percorso. Screenshot di Eclipse

Il metodo *percorso* riceve come parametrii vincoli imposti sulla ricerca dei voli e si occupa di inizializzare le variabili e le liste necessarie, successivamente chiama il metodo *percorsoRicorsiva*, dove avviene l’effettiva ricerca. Una volta ottenuti i risultati dalla chiamata, li inserisce in apposite strutture dati, gli itinerari, che poi inserisce in una lista e restituisce in output.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 11. Metodo percorsoRicorsiva. Screenshot di Eclipse

*PercorsoRicorsiva* segue la struttura tipica di un metodo ricorsivo. Il metodo riceve come parametri i vincoli inseriti dall’utente, oltre a una lista di archi ed una lista di vertici da popolare durante la ricerca con le soluzioni sotto studio.  
Una prima serie di controlli valuta la presenza di casi terminali, in ordine negativi e positivi. In situazione di caso terminale negativo, il metodo torna alla precedente chiamata senza ulteriori ricerche, mentre in un caso terminale positivo, prima di ritornare aggiunge alla lista dei risultati il caso appena studiato.  
Dopo aver dichiarato ed inizializzato un iteratore per visitare il grafo, in questo caso uno che predilige la ricerca in profondità, si trovano i controlli sui casi intermedi. Un ciclo for, contenente un ciclo while e dei controlli if, si occupa di contare il numero di chiamate ricorsive che possono avvenire, a seconda del numero di scali scelto. In questo caso il numero di scali coincide con il livello massimo di ricorsione che si vuole affrontare. All’interno del ciclo while si trova un’ulteriore struttura tipica dei metodi ricorsivi; le righe 183, 184 e 185 della figura 11. Queste tre righe rappresentano l’aggiornamento del caso al momento in studio, la chiamata ricorsiva del metodo e il ritorno allo stato originale del caso al momento in studio, prima dell’aggiornamento. La presenza di queste righe garantisce l’esplorazione di tutte le possibilità.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 12. metodo sommaprezzo. Screenshot di Eclipse

Il metodo *sommaprezzo* viene chiamato all’interno di *percorsoRicorsiva* e serve a calcolare il costo totale dell’itinerario al momento sotto studio, per verificare che non ecceda il limite posto dall’utente. Riceve come parametro la lista di archi attraversati fino a quel momento, popolata da *PercorsoRicorsiva*, e restituisce il peso totale di essi.

### Ricerca degli AirBnB ricorsiva

La ricerca degli AirBnB contestuale alla seconda parte del progetto, similmente alla ricerca dei voli, è composta da tre metodi, di cui due privati.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 13. Metodo ricercaBnb. Screenshot di Eclipse

*RicercaBnb* riceve come parametri i vincoli imposti dall’utente ed al suo interno vengono inizializzate le variabili e le liste necessarie per la ricerca. Viene verificato in quale città l’utente ha deciso di alloggiare ed una serie di controlli if si occupa di gestire la forma della stringa relativa. Un controllo for si occupa di sezionare la lista di tutti gli AirBnB del sistema, tenendo in conto soltanto quelli della città di interesse. Ciò è necessario per la dimensione della lista originale che, se fatta scorrere con un algoritmo ricorsivo, genererebbe errore a causa della profondità di ricorsione eccessiva.  
Una volta chiamato il metodo *ricercaBnbRicorsiva*, viene restituito il risultato.

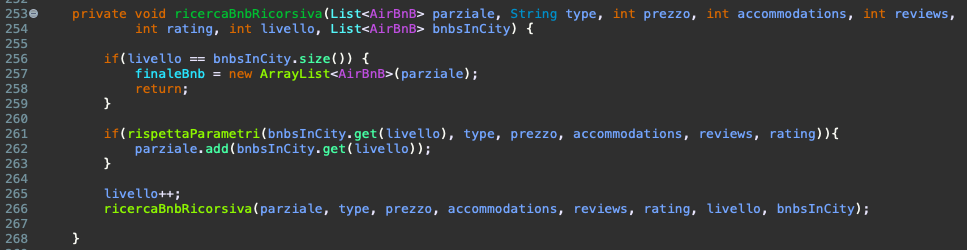


Figura 14. Metodo ricercaBnbRicorsiva. Screenshot di Eclipse

Il metodo *ricercaBnbRicorsiva* implementa la ricorsione relativa alla ricerca degli AirBnB adatti. Riceve in input i parametri scelti dall’utente, oltre a una lista di soluzioni da popolare con casi che rispettano i vincoli e la lista di tutti gli AirBnB presenti nella città di studio. Un ultimo parametro in entrata è il livello di ricorsione, che viene utilizzato per terminare il programma una volta che ha raggiunto la profondità adeguata.  
Nuovamente il metodo si divide in studio del caso terminale e studio del caso intermedio.Al contrario del metodo *PercorsoRicorsiva*, non è presente la procedura di aggiunta-chiamata-backtracking, perché in questo caso non è previsto un ordine da seguire o un grafo da esplorare, ma soltanto una lista di possibilità da scorrere.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 15. Metodo rispettaParametri. Screenshot di Eclipse

Il metodo *rispettaParametri* ha la semplice funzione di controllare se l’AirBnB passato in input rispetta le condizioni imposte dall’utente, anche loro passate in input.   
Viene chiamato all’interno di *PercorsoRicorsiva.*

# Diagramma delle classi principali



Figura 16. Diagramma delle classi principali. È stato mantenuto il color coding utilizzato da Eclipse

# Interfaccia utente

Appena avviata, l’applicazione appare prevalentemente bloccata e richiede il caricamento delle informazioni del data-set tramite il tasto load. Finché l’utente non segue l’indicazione, il software non da accesso a nessun filtro o bottone.

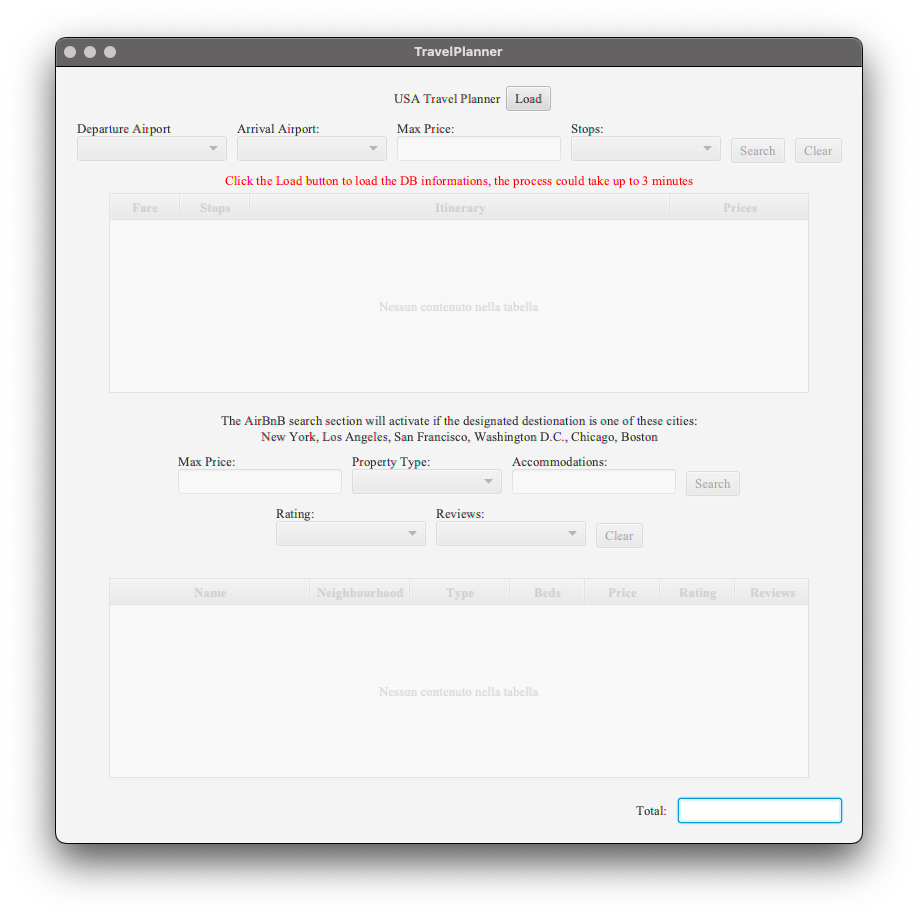


Figura 17. Applicazione al suo avvio

Una volta caricato il data-set, è possibile iniziare le proprie ricerche. Sono presenti dei controlli di sicurezza che prevengono l’inizio di una ricerca con informazioni mancanti o non valide. La mancata scelta di un filtro o l’inserimento di informazioni non valide comportano l’apparizione di un messaggio di errore rosso sopra la tabella dei risultati.

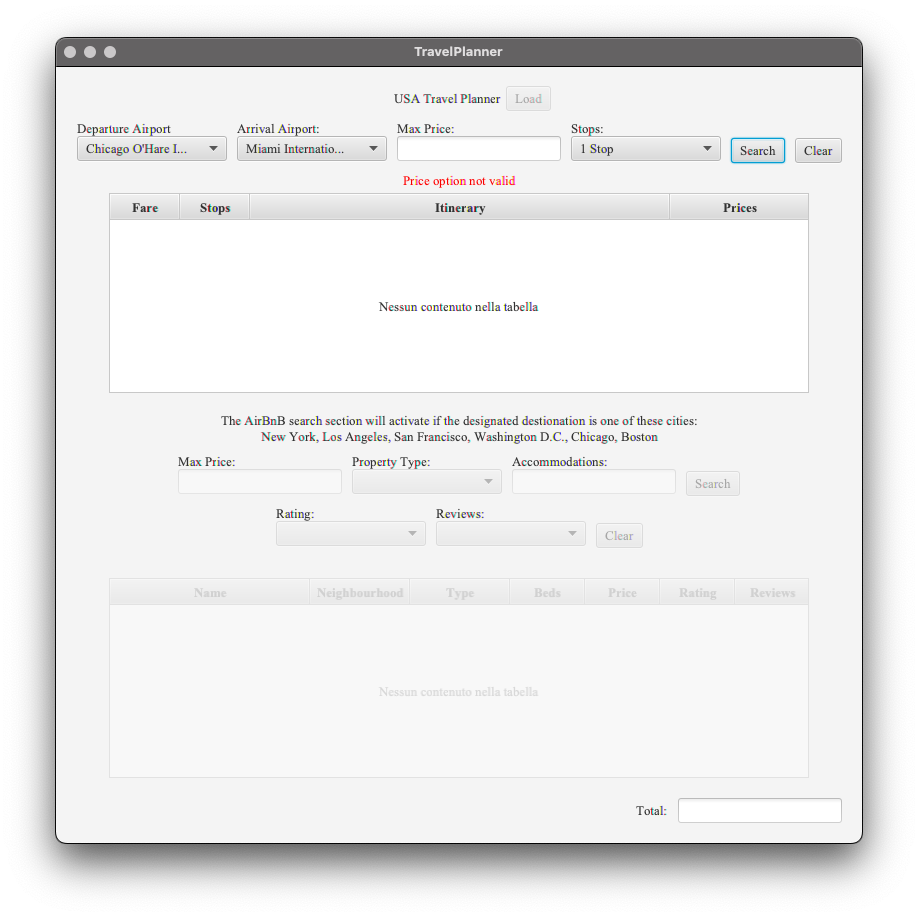


Figura 18. Controlli e filtri implementati

Se la ricerca va a buon fine, i risultati saranno esposti nella tabella. In caso contrario, un messaggio comunicherà la mancanza di soluzioni nel data-set.

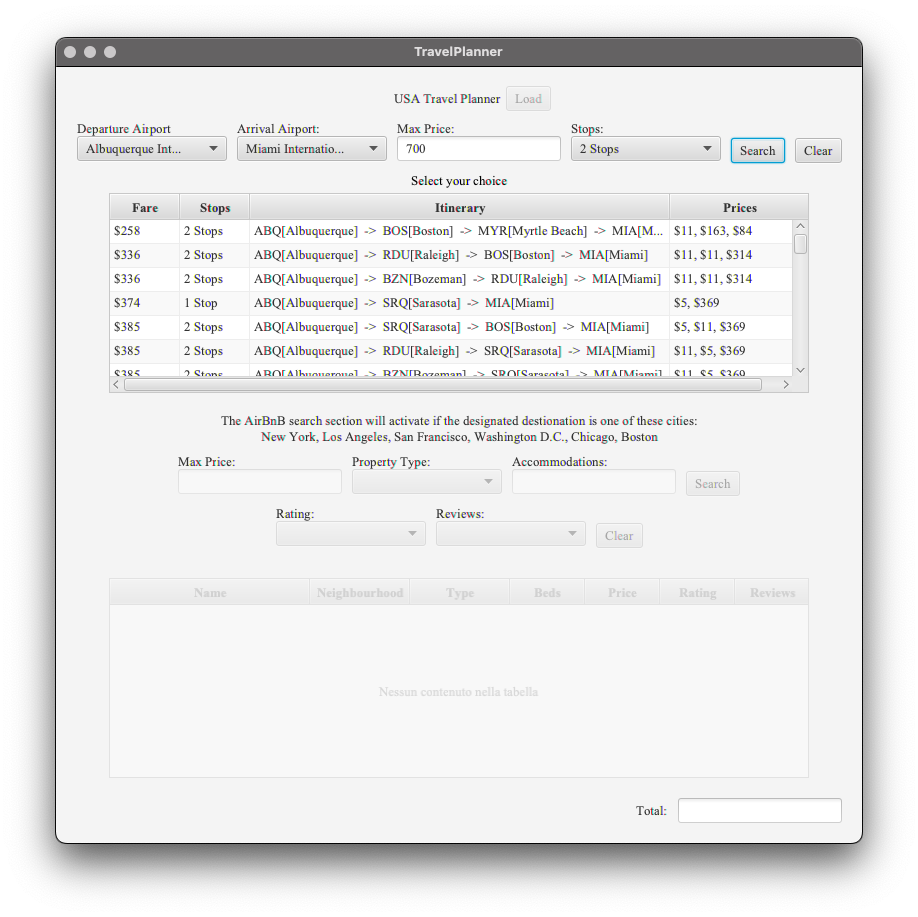


Figura 19. Ricerca generica a buon fine

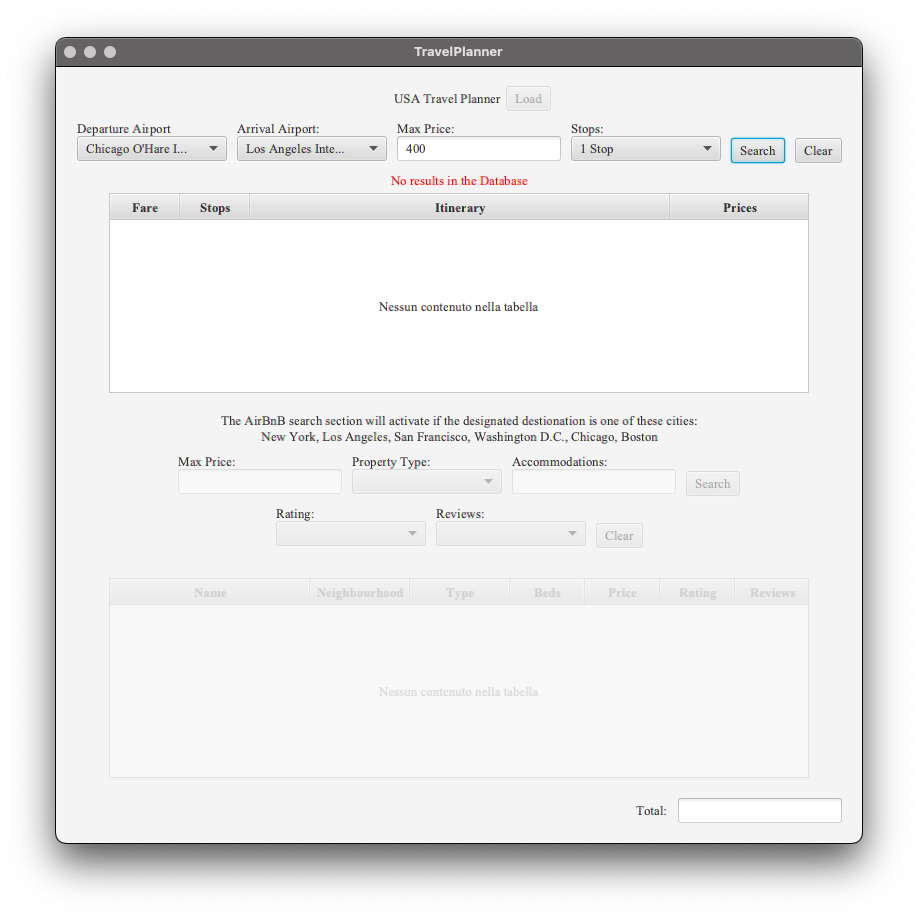


Figura 20. Ricerca generica fallita

In caso la ricerca sia andata a buon fine, è possibile selezionare un volo e il suo costo verrà inserito nella casella “Total” in basso a destra. La casella è dinamica, quindi la selezione di diversi voli aggiorna automaticamente il suo valore. Inoltre, se la destinazione scelta fa parte delle sei città supportate, la parte inferiore della schermata si sblocca, permettendo la ricerca di un AirBnB.

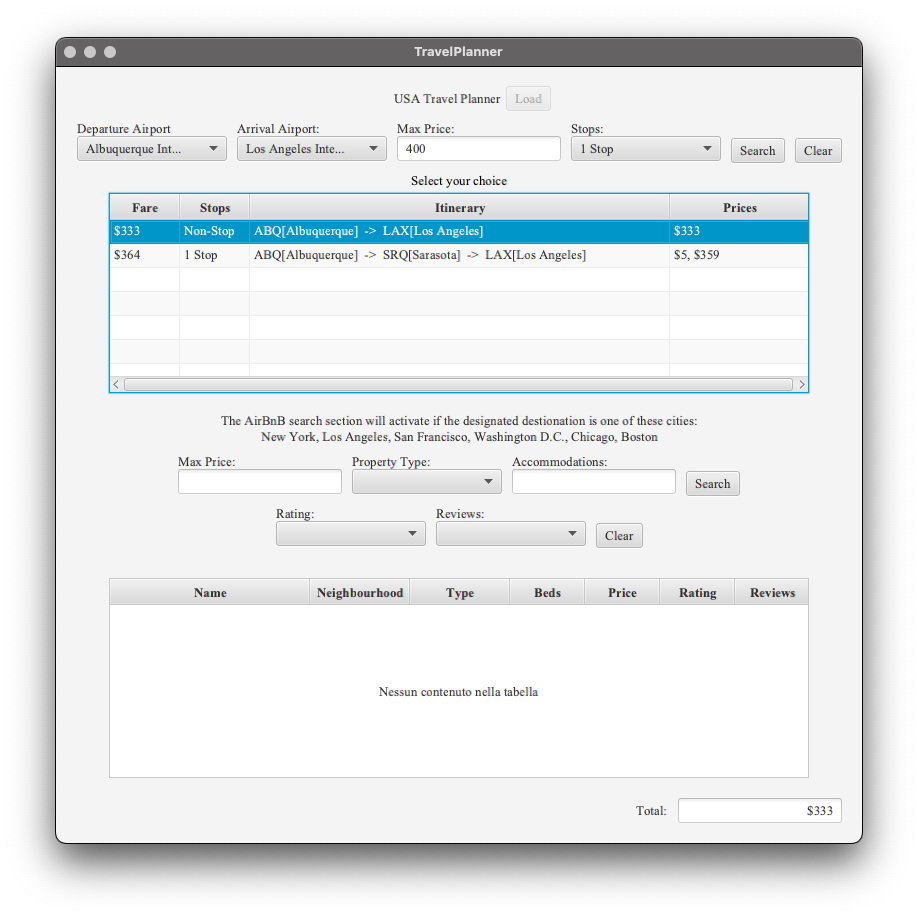


Figura 21. Funzionamento della casella “Total” e sblocco della seconda parte della schermata

La ricerca di un AirBnB funziona nel medesimo modo, con filtri e controlli di sicurezza.   
Sono presenti ulteriori messaggi di errore o indicazioni per l’utente e nuovamente, i risultati, se presenti, saranno mostrati nell’apposita tabella. In caso contrario verrà notificata all’utente la mancanza di soluzioni.

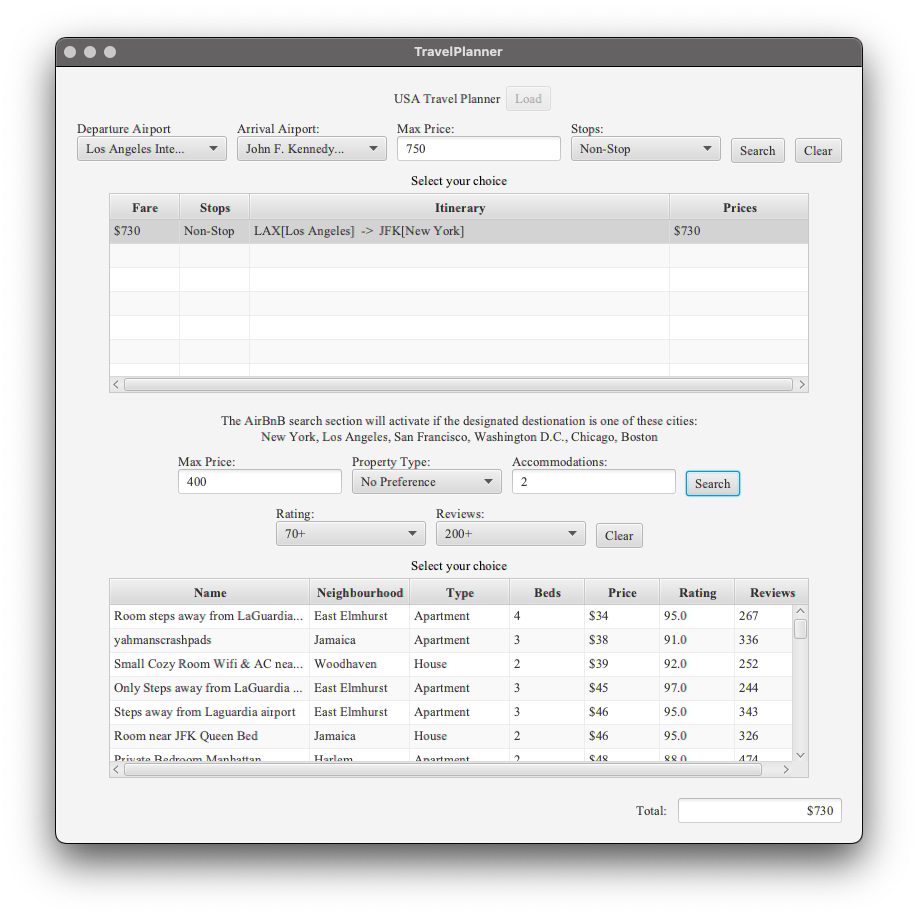


Figura 22. Ricerca di un AirBnB e risultati

Selezionando una scelta tra gli AirBnB mostrati, la casella “Total” si aggiornerà, mostrando la somma dei prezzi del volo e della sistemazione. Cambiando selezione, il valore si aggiornerà in automatico. Dopo aver già selezionato un AirBnB, cambiare la scelta del volo porterà la casella “Total” a mostrare solo il prezzo del nuovo volo. Per visualizzare il totale aggiornato sarà necessario selezionare nuovamente l’AirBnB.

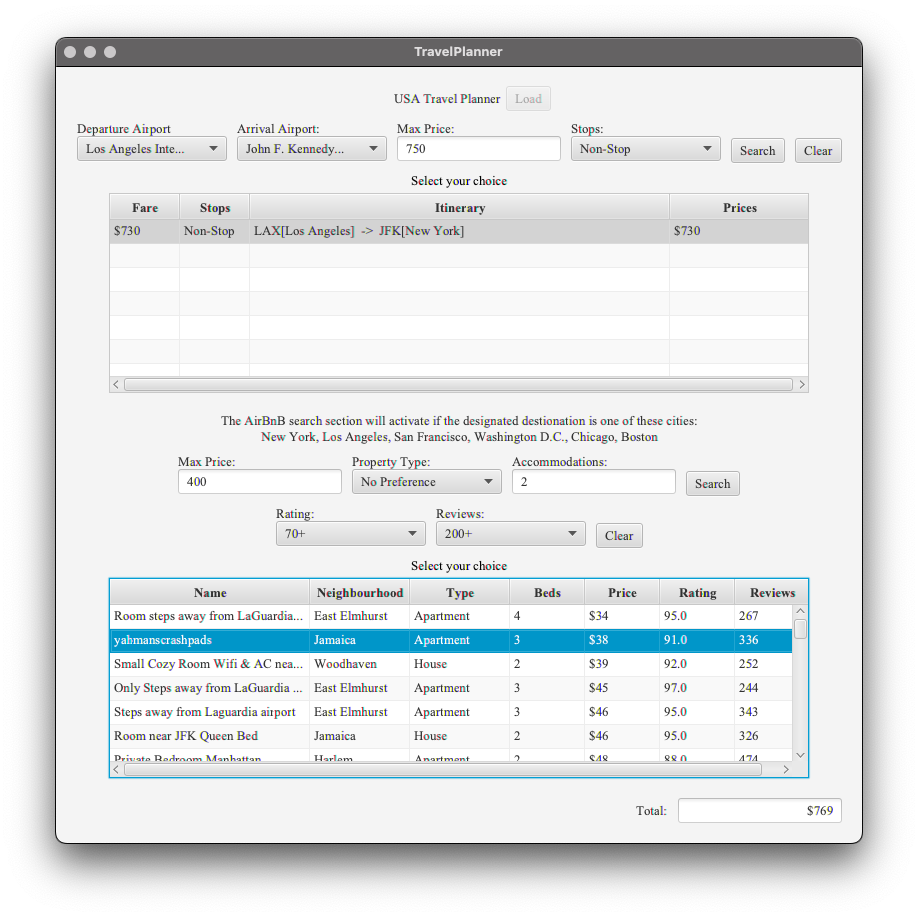


Figura 3. Ricerca di un itinerario completata. L’utente ha selezionato un volo e un AirBnB e il costo totale viene mostrato

È disponibile su YouTube un [breve video tutorial](https://youtu.be/DXojfuG4Ka8) per l’applicazione.

# Risultati sperimentali

A scopo informativo, gli aeroporti delle sei città supportate dalla ricerca di un AirBnB sono:

* New York: John F. Kennedy International Airport (New York International Airport), LaGuardia Airport (Marine Air Terminal).
* Los Angeles: Los Angeles International Airport.
* San Francisco: San Francisco International Airport.
* Washington D.C.: Ronald Reagan Washington National Airport, Washington Dulles International Airport.
* Chicago: Chicago Midway International Airport, Chicago O'Hare International Airport.
* Boston: Gen. Edward Lawrence Logan International Airport.

Nelle seguenti immagini sono riportati i primi risultati sperimentali ottenuti dalla ricerca di voli per l’aeroporto JFK di New York e per l’aeroporto LAX di Los Angeles. Come partenza è stato selezionato l’ORD di Chicago. La notevole dimensione degli aeroporti scelti permette la presenza di un maggior numero di risultati. Il prezzo scelto è $800, abbastanza alto da mostrare la maggior parte dei voli disponibili.

Immagine che contiene testo

Descrizione generata automaticamente

Figura 24. Voli diretti dall’ORD di Chicago al JFK di New York. Risultato mostrato istantaneamente

Immagine che contiene testo

Descrizione generata automaticamente

Figura 25. Voli con massimo 1 scalo dall’ORD di Chicago al JFK di New York. Risultato mostrato in circa 1 secondo

Immagine che contiene testo

Descrizione generata automaticamente

Figura 26. Voli con massimo 2 scali dall’ORD di Chicago al JFK di New York. Risultato mostrato in circa 30 secondi

Immagine che contiene testo

Descrizione generata automaticamente

Figura 27. Voli diretti dall’ORD di Chicago al LAX di Los Angeles. Risultato mostrato istantaneamente

Immagine che contiene testo

Descrizione generata automaticamente

Figura 28. Voli con massimo 1 scalo dall’ORD di Chicago al LAX di Los Angeles. Risultato mostrato in circa 1 secondo

Immagine che contiene testo

Descrizione generata automaticamente

Figura 29. Voli con massimo 2 scali dall’ORD di Chicago al LAX di Los Angeles. Risultato mostrato in circa 30 secondi

Nelle seguenti immagini sono riportati i primi risultati sperimentali ottenuti dalla ricerca di AirBnB in ognuna delle sei città supportate. I filtri scelti sono:

* Max Price = $800. Una cifra abbastanza alta da mostrare tutti gli AirBnB disponibili.
* Property Type = No preference. La scelta permette di mostrare tutti i tipi di proprietà disponibili
* Accommodations 2. Una scelta arbitraria che riflette la realtà di un viaggio di coppia o in famiglia.
* Rating = 60+. Mostra tutti gli AirBnB con una valutazione sufficiente.
* Reviews = 200+. Scelta arbitraria per ottenere solo sistemazioni che hanno ottenuto un numero significativo di recensioni, assicurando l’effettiva esistenza della struttura ed il suo utilizzo.

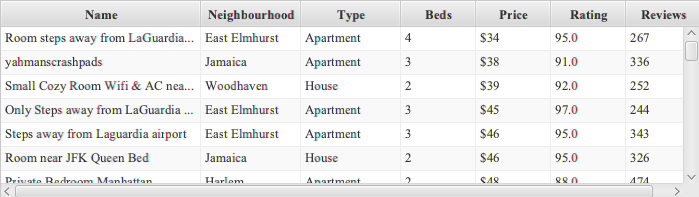


Figura 0. Risultati sperimentali per gli AirBnB di New York

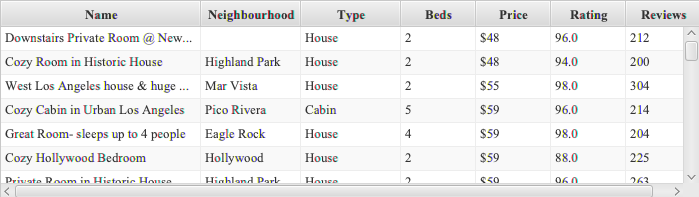


Figura 31. Risultati sperimentali per gli AirBnB di Los Angeles

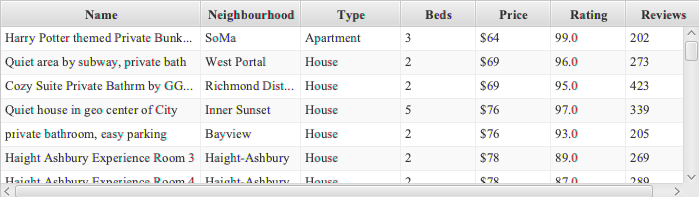


Figura 32. Risultati sperimentali per gli AirBnB di San Francisco

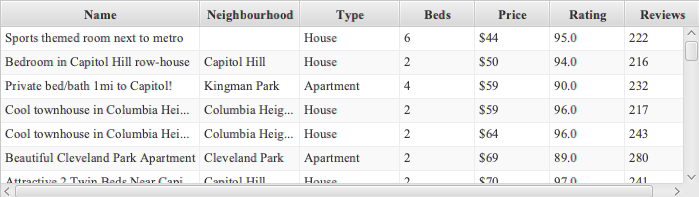


Figura 33. Risultati sperimentali per gli AirBnB di Washington D.C.

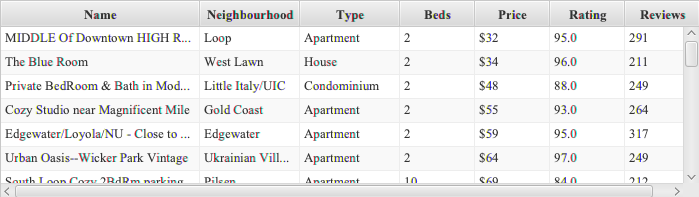


Figura 34. Risultati sperimentali per gli AirBnB di Chicago

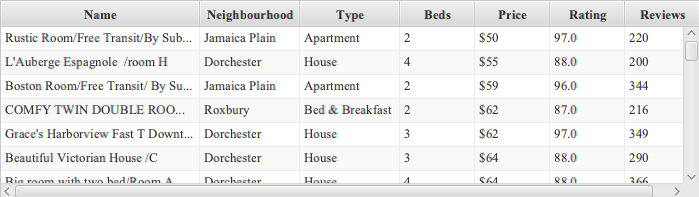


Figura 35. Risultati sperimentali per gli AirBnB di Boston

Il caricamento delle informazioni del data-set sull’applicazione tramite il tasto “load” richiede una media di circa 165 secondi, ovvero 2:45 minuti. Il risultato è ottenuto da 5 test i cui risultati sono riportati nella tabella 9.

Tabella . Risultati sperimentali per i tempi di caricamento del data-set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | T1 | T2 | T3 | T4 | T5 |
| Tempo[min] | 2:48 | 2:39 | 2:47 | 2:48 | 2:45 |

# Valutazioni finali

A seguito delle valutazioni fatte fino a questo momento, ritengo l’applicazione all’altezza delle aspettative. Un’idea iniziale chiara e solida ha permesso la previsione dei problemi che avrei affrontato e ha permesso una preparazione adeguata.

Il processo più dispendioso è stato creare un data-set sodisfacente. La ricerca di dati utili si è rivelata tediosa, a causa della natura delle informazioni ricercate. Data-set simili, contenenti informazioni sugli hotel e i voli in USA, sono asset professionali disponibili solo previo pagamento ed utilizzati dalle aziende del settore.

Una volta ottenuto un database soddisfacente, la pulizia e l’adeguamento di esso a degli standard mnemonici ha richiesto un ulteriore ingente quantitativo di tempo.  
Superata la fase preparativa, l’unico grande scoglio è stato mantenere i tempi di caricamento sotto a dei livelli che ho ritenuto accettabili. In particolare, dell’intero processo, solo la ricerca e l’aggiunta degli archi al grafo si è rivelata difficile da migliorare. Dopo aver provato diverse soluzioni concettuali e pratiche, sono riuscito ad ottenere come tempo migliore quello attuale, cioè circa 2 minuti e 40 secondi. Ritengo che questa attesa sia il più grande difetto dell’applicazione. Nonostante siti appositi, come per esempio SkyScanner o Booking, contengano caricamenti simili ed inevitabili, mai si raggiungono tempi così lunghi, che scoraggiano l’utente dall’utilizzo. Chiaramente il dispositivo utilizzato gioca una grande parte nell’ottimizzazione dei tempi, ma è anche necessario rendere l’applicazione utilizzabile con scioltezza su dispositivi meno prestanti.

Algoritmicamente, tralasciando la creazione del grafo già descritta, la parte ricorsiva non ha causato problemi o rallentamenti. L’attuale codice è una sorta di esercizio di stile, vista la facilità con cui si può sostituire la ricerca ricorsiva di un AirBnB con una query diretta al database, oppure con un semplice ciclo for each sull’intera lista, entrambe soluzioni molto più intuitive.

L’interfaccia è rimasta per lo più invariata nel corso dello sviluppo. Ho ritenuto necessaria solo l’aggiunta della casella di testo “Total”, per rendere più dinamica la scelta e aggiungere un tocco di realismo all’applicazione. Non si tratta certamente di un’interfaccia bella o accattivante, ma sicuramente è funzionale e offre tutte le informazioni necessarie.

Riguardo ciò che è stato scritto al [punto 2](#_Descrizione_dettagliata_del), l’applicazione soddisfa quasi a pieno i requisiti prefissati. La quantità di filtri rende verosimile la ricerca e le soluzioni offerte all’utente nelle apposite tabelle sono ben esposte. Una mancanza pesante riguarda gli orari dei voli. Non può esistere un programma di prenotazione che non contenga informazioni su date e orari, cosa che relega la mia applicazione allo scopo didattico e allo studio statistico.

L’aggiunta di informazioni riguardanti data e ora dei voli comporterebbe dei cambiamenti soprattutto sull’ordinamento delle liste finali, ma non intaccherebbe l’integrità dei metodi ricorsivi, in quanto l’utente restringerebbe la lista di possibilità con ulteriori filtri che, se implementati correttamente, migliorerebbero l’efficienza dei metodi stessi.

I punti di forza del programma sono sicuramente la sua facilità d’utilizzo e la rappresentazione efficace delle soluzioni richieste. Ho cercato, anche tramite feedback esterni, di mantenere l’interfaccia il più semplice possibile, senza però sacrificare la buona rappresentazione dei dati. L’indirizzamento dell’utente è un altro cardine dell’applicazione, che si rispecchia nei continui messaggi di errore o consigli che appaiono sopra le tabelle dopo ogni operazione.

In conclusione, giudico l’applicazione riuscita e rimango convinto che, con un data-set professionale dinamico e contenente tutte le informazioni necessarie, lo sviluppo non sarebbe stato significativamente diverso. Una soluzione tampone per ottenere il tanto agognato data-set realistico sarebbe costruire un bot per lo scraping dei dati ed utilizzarlo sui siti delle compagnie aeree e di prenotazione di hotel. Si rimarrebbe comunque in una situazione di studio, ma il risultato sarebbe decisamente migliore.

Immagine che contiene testo, clipart, segnale

Descrizione generata automaticamente

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-sa/4.0/>