Politecnico di Torino

Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale

Anno accademico 2021/2022

Sessione di Laurea Luglio 2022

Software gestionale per ufficio acquisti

Relatore:

prof. Fulvio Corno

Candidato:

Mauro Caradonna

Indice

Capitolo 1: Proposta di progetto

1.1 Studente proponente

s268974 Mauro Caradonna

1.2 Titolo della proposta

Software gestionale per ufficio acquisti

1.3 Descrizione del problema proposto

Il software ha come obiettivo quello di misurare le performance dell'ufficio acquisti e simulare il costo finale di un prodotto finito composto da più articoli acquistati. Le performance saranno misurate sulla base di parametri che si basano sullo scostamento del prezzo di acquisto dell'anno in esame rispetto al prezzo dell'anno precedente. I risultati saranno calcolati sia per buyer, sia per sedi aziendali, sia per categorie di prodotti. L'utente potrà caricare tramite file i nuovi ordini, per i quali verrà stampata l'analisi della varianza dei costi. Inoltre intendo simulare il ciclo di vita degli articoli per un'azienda, da quando dovranno essere acquistati a quando sarà venduto il prodotto finito finale che li contiene. Una serie di eventi specifici potranno avvenire per i singoli articoli e ciò avrà influenza sul prezzo finale del prodotto.

1.4 Descrizione della rilevanza gestionale del problema

Sto svolgendo il tirocinio presso un ufficio acquisti di una multinazionale e so che uno strumento di questo tipo è fondamentale per il management di una media/grande azienda per supportare le decisioni manageriali e per tenere sotto controllo le performance dei dipendenti. Inoltre all'ufficio acquisti viene richiesto di prevedere il prezzo dei materiali dei prodotti venduti perciò risulta utile avere una simulazione di possibili scenari che potrebbero accadere.

1.5 Descrizione dei data-set per la valutazione

Come data-set intendo usare i dati dell'azienda presso la quale sto svolgendo il tirocinio, eventualmente modificando alcuni valori. In particolare, userò dei data-set per i buyer aziendali, gli articoli acquistati, gli ordini, le fatture, i fornitori e le distinte base.

1.6 Descrizione preliminare degli algoritmi coinvolti

All'avvio il programma calcolerà una serie di parametri tramite un algoritmo sia per l'anno da analizzare, sia per l'anno precedente. Questi parametri saranno memorizzati in un'opportuna struttura dati per poter essere disponibili per essere visualizzati e per essere elaborati successivamente. All'inserimento di nuovi ordini saranno ricalcolati tutti i parametri e gli ordini saranno memorizzati nel database. Sulla base dei dati in possesso del modello e dai parametri inseriti dall'utente il programma userà algoritmi di simulazione per calcolare il costo di un prodotto finito, tenendo in considerazione alcuni eventi che possono accadere durante il Lead-time.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'idea è quella di permettere all'utente di ricevere i risultati in scene differenti. In una scena l'utente potrà visualizzare i valori sugli andamenti dei prezzi d'acquisto degli articoli. Da combo Box potrà scegliere se visualizzare le performance per ciascun dipendente, per sede aziendale, per articolo o per commodity. In un'altra scena l'utente potrà inserire da file .csv, che è facile ottenere da un foglio di lavoro Excel, i nuovi ordini fatti e, per quegli articoli, visualizzare l'analisi della Cost Variance. Infine in un'altra scena l'utente può impostare alcuni parametri, scegliere un prodotto finale da analizzare con la simulazione e in una text Area ricevere i risultati.

Capitolo 2: Descrizione dettagliata del problema affrontato

2.1 Contesto operativo/aziendale

L'ufficio acquisti è il dipartimento aziendale deputato alla gestione dell'approvvigionamento, ovvero l'acquisizione di componenti, beni e servizi necessari per soddisfare il fabbisogno aziendale.
I responsabili operativi dell'approvvigionamento vengono chiamati *Buyer*, la cui occupazione principale è trovare i fornitori e negoziare con loro i contratti di compravendita.
I materiali acquistati si possono distinguere in *diretti* e *indiretti* sulla base della finalità. I materiali diretti hanno un impatto sul margine dei prodotti venduti, perciò il loro reperimento deve avvenire secondo logiche diverse a quelle di un consumatore. La politica di acquisti aziendale ha un impatto sul breve, medio e lungo periodo bisogna tenere perciò approcci paralleli sulla gestione operativa, tattica e strategica.
Nel caso di multinazionali, l'internazionalità porta sicuramente dei vantaggi, ma è necessario avere un corretto approccio. Si parla per esempio di *handle buy*, ovvero il raggiungimento di un unico accordo con lo stesso fornitore per i diversi poli aziendali. È anche fondamentale avere un'organizzazione funzionale, ovvero una struttura organizzativa verticale dove i dipendenti sono raggruppati per area di specializzazione e diretti da un responsabile di funzione. Ogni buyer diventa dunque un grande esperto del proprio gruppo di acquisto, le cosiddette commodity.
Per raggiungere gli obiettivi aziendali il CPO (Chief Procurement Officer) ha la necessità di misurare i risultati e avere dei report. Perciò, al software è stato posto l'obiettivo di misurare le performance dell'ufficio acquisti sulla base dell'analisi della Cost Variance.

2.2 Cost Variance

La Cost Variance è calcolata come differenza fra il costo di acquisto nell'anno da analizzare rispetto al costo di acquisto nell'anno precedente. Il management così tiene traccia dei trend dei prezzi e potrebbe intervenire istantaneamente qualora i risultati fossero negativi. Nello specifico ho considerato quattro parametri basati sulla variazione del costo: CV avg, CV avg%, CV last, CV last%.

- CV avg (Cost Variance average) indica se e quanto l'azienda stia spendendo in più o in meno per un prodotto rispetto all'anno precedente, se lo scorso anno fosse stata acquistata una quantità di prodotto pari alla quantità acquistata nell'anno corrente, al prezzo medio di quell'anno.
  Alla fine dell'esercizio indica esattamente la variazione di costo per quel prodotto che incide sul Conto Economico.

  In formule: $$CV\ avg\ =\ (\ \overline{P1}\ -\ \overline{P0}\ )\ *\ Q1$$

dove: $\overline{P1}$ è il prezzo medio dell'articolo nell'anno da analizzare,

$\overline{P0}$ è il prezzo medio dell'articolo nell'anno precedente,

Q1 è la quantità dell'articolo acquistata nell'anno da analizzare.

- CV avg % indica in valore relativo se e quanto l'azienda stia spendendo in più o in meno per un prodotto rispetto all'anno precedente, se lo scorso anno fosse stata acquistata una quantità di prodotto pari alla quantità acquistata nell'anno corrente, al prezzo medio di quell'anno.

In formule: $$CV\ avg\ \% = \frac{CV\ avg}{\overline{P0} * Q1}$$

dove: $CV\ avg$ è la Cost Variance del prezzo medio dell'articolo nell'anno da analizzare,

$\overline{P0}$ è il prezzo medio dell'articolo nell'anno precedente,

Q1 è la quantità dell'articolo acquistata nell'anno da analizzare.

- CV last (Cost Variance last) indica se e quanto l'azienda stia spendendo in più o in meno per un prodotto rispetto all'anno precedente, se lo scorso anno fosse stata acquistata una quantità di prodotto pari alla quantità acquistata nell'anno corrente, ma all'ultimo prezzo di acquisto di quell'anno e quest'anno avessimo acquistato tutta la quantità di quest'articolo all'ultimo prezzo di acquisto di quest'anno.

  Questo parametro è utile per tener conto anche dell'evoluzione del mercato.

In formule: $$CV\ last\ = (P1 - P0) * Q1$$

dove: $P1$ è l'ultimo prezzo dell'articolo nell'anno da analizzare,

$P0$ è l'ultimo prezzo dell'articolo nell'anno precedente,

Q1 è la quantità dell'articolo acquistata nell'anno da analizzare.

- CV last % indica in valore relativo se e quanto l'azienda stia spendendo in più o in meno per un prodotto rispetto all'anno precedente, se lo scorso anno fosse stata acquistata una quantità di prodotto pari alla quantità acquistata nell'anno corrente, ma all'ultimo prezzo di acquisto di quell'anno e quest'anno avessimo acquistato tutta la quantità di quest'articolo all'ultimo prezzo di acquisto di quest'anno.

In formule: $$CV\ last\ = \frac{CV\ last}{P0 * Q1}$$

dove: $CV\ last$ è la Cost Variance dell'ultimo prezzo dell'articolo nell'anno da analizzare,
$P0$ è l'ultimo prezzo dell'articolo nell'anno precedente,
Q1 è la quantità dell'articolo acquistata nell'anno da analizzare.

Incrociando i risultati dei vari parametri dei singoli articoli tra loro è possibile definire dei parametri anche per gruppi di articoli. Il programma calcola così anche le performance dei diversi siti produttivi, buyers e commodity. Nel dettaglio, la Cost Variance di un gruppo di articoli è pari alla somma della Cost Variance dei singoli articoli. La Cost Variance percentuale di un gruppo di articoli è pari al rapporto tra la Cost Variance del gruppo e la somma per ciascun articolo del prodotto tra il prezzo di acquisto dell'anno precedente e la quantità dell'articolo acquistata nell'anno da analizzare.

2.3 Simulazione

L'ufficio acquisti collabora con gli altri dipartimenti aziendali per raggiungere obiettivi comuni. Per esempio dà un contributo fondamentale alla determinazione del costo del prodotto finito, sul quale poi si basa il prezzo di vendita. Il lavoro richiesto non è semplice se il prodotto finito ha tanti componenti. Ciascun componente infatti ha un proprio prezzo di acquisto che dipende da diversi fattori. Durante l'anno i listini delle aziende fornitrici vengono aggiornati, anche più volte durante lo stesso anno. La situazione si complica in un periodo di crisi di approvvigionamenti, causato da vari fattori come è stato il caso del Covid 19 o il caso della guerra. È possibile infatti che i pezzi ritardino o che addirittura non vengano consegnati. Ciò causa una situazione di emergenza, per cui conviene mandare l'articolo il prima possibile sulla linea di produzione acquistandolo ad un prezzo maggiore piuttosto che bloccare la produzione.
Il software aiuta il management in questo compito grazie ad una simulazione ad eventi discreti stocastica. In particolare è possibile caricare l'elenco dei prodotti presenti attualmente in magazzino, l'elenco dei prodotti della distinta base del prodotto finito da analizzare, la probabilità che ciascun ordine ritardi o che addirittura non venga consegnato e le previsioni dei prezzi di ciascuno dei prodotti della distinta base. Le condizioni di funzionamento ottimali si avrebbero se l'azienda disponesse correttamente di questi dati. Mentre è facile infatti disporre di dati sugli articoli in magazzino e nella distinta base, risulta più difficile ottenere gli altri valori. Le probabilità si possono ottenere facendo un'analisi statistica del comportamento dei vari fornitori, mentre le previsioni si possono ottenere per esempio grazie a società esterne oppure ai

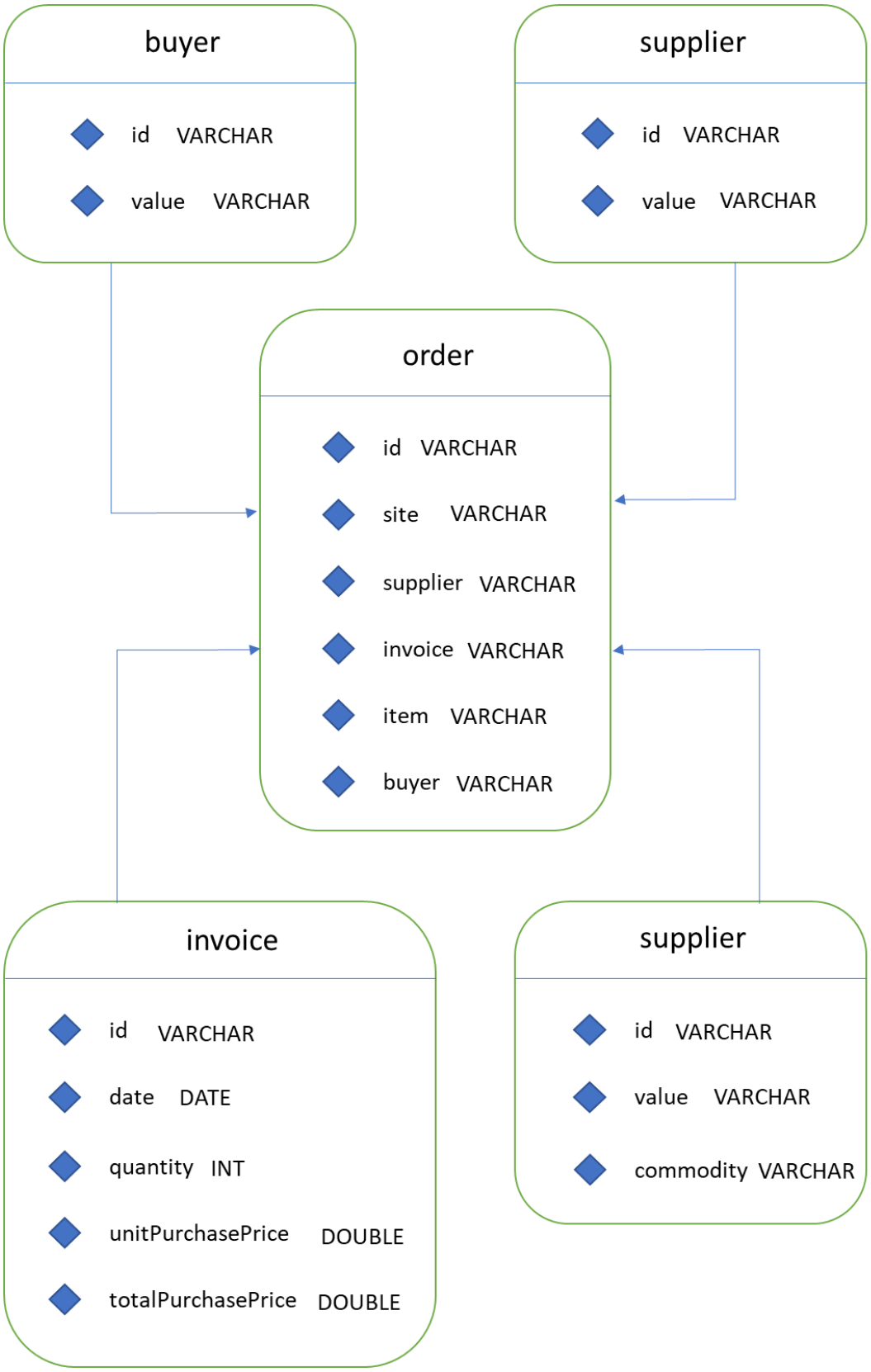
listini dei fornitori che anticipano gli scostamenti. Infine l'utente sceglie la data dalla quale la simulazione inizia. Ciascun articolo della distinta base può essere soggetto a vari eventi. Se l'articolo è presente in magazzino viene prelevato e mandato in produzione, ma nel caso non fosse presente dovrebbe essere ordinato. In fase di ordine viene stipulato un prezzo d'acquisto e dopo un periodo di Lead Time, vengono generati tre possibili eventi che corrispondono all'arrivo in magazzino, al ritardo del pezzo o all'annullamento della consegna. Chiaramente questo influisce sul prezzo d'acquisto poiché in caso di mancata consegna il prezzo deve essere ordinato ad un prezzo maggiore o minore sulla base del prezzo indicato nelle previsioni. Ma sia nel caso di ritardo che di annullamento i tempi della produzione del prodotto finito si allungano. L'algoritmo di simulazione avvia la produzione vera e propria solo quando tutti gli articoli sono presenti in produzione. Non tutte le aziende applicano questa politica di produzione, perciò questa semplificazione è una potenziale criticità. Dopo il periodo di produzione il prodotto finito è completo e l'interfaccia grafica presenta i risultati all'utente. In particolare viene mostrata la somma dei costi di acquisto di tutti gli articoli, la data di fine produzione e la lista di tutti gli eventi della simulazione.

2.4 Potenzialità, criticità e rilevanza

Il software ha delle grandi potenzialità per tutti i tipi di aziende, particolarmente per le aziende manifatturiere. Potrebbe essere utile anche per decidere quanto e se attribuire dei bonus ai buyer sulla base della Cost Variance oppure simulare il prezzo e la possibile data di vendita di un nuovo prodotto da iniziare a produrre. Oppure per fare una previsione su quali saranno i costi da riportare in conto economico alla fine dell'esercizio o ancora simulare la data di rilascio di un prodotto se si decidesse di far iniziare la simulazione dopo un certo numero di mesi. Le criticità principali sono però le approssimazioni fatte dagli algoritmi. In particolare è stato ipotizzato che è possibile fare ordini per un solo prodotto e che dopo l'annullamento di una consegna la nuova data di consegna venga rispettata. La giusta interpretazione dei risultati però potrebbe essere rilevante in generale per supportare le decisioni manageriali.

Capitolo 3: Descrizione del data-set

Il data-set utilizzato per l'analisi è composto da otto tabelle, di cui cinque ottenute dall'estrapolazione e elaborazione dei dati del database aziendale di Prima Industrie Spa.



La tabella order contiene le informazioni sugli ordini effettuati:

- id: identificativo dell'ordine
- site: sito per il quale è stato emesso l'ordine
- supplier: identificativo del fornitore
- invoice: identificativo della fattura
- item: identificativo dell'articolo
- buyer: identificativo del buyer

La tabella buyer contiene le informazioni sui buyer:

- id: identificativo del buyer
- value: descrizione del buyer

La tabella invoice contiene le informazioni sugli ordini effettuati:

- id: codice della fattura
- date: data di emissione della fattura
- quantity: quantità di articoli acquistati
- unitPurchasePrice: prezzo unitario
- totalPurchasePrice: prezzo totale

La tabella item contiene le informazioni sugli ordini effettuati:

- id: identificativo dell'articolo
- value: descrizione dell'articolo
- commodity: commodity alla quale l’articolo appartiene

La tabella supplier contiene le informazioni sugli ordini effettuati:

- id: identificativo del fornitore
- value: descrizione del fornitore

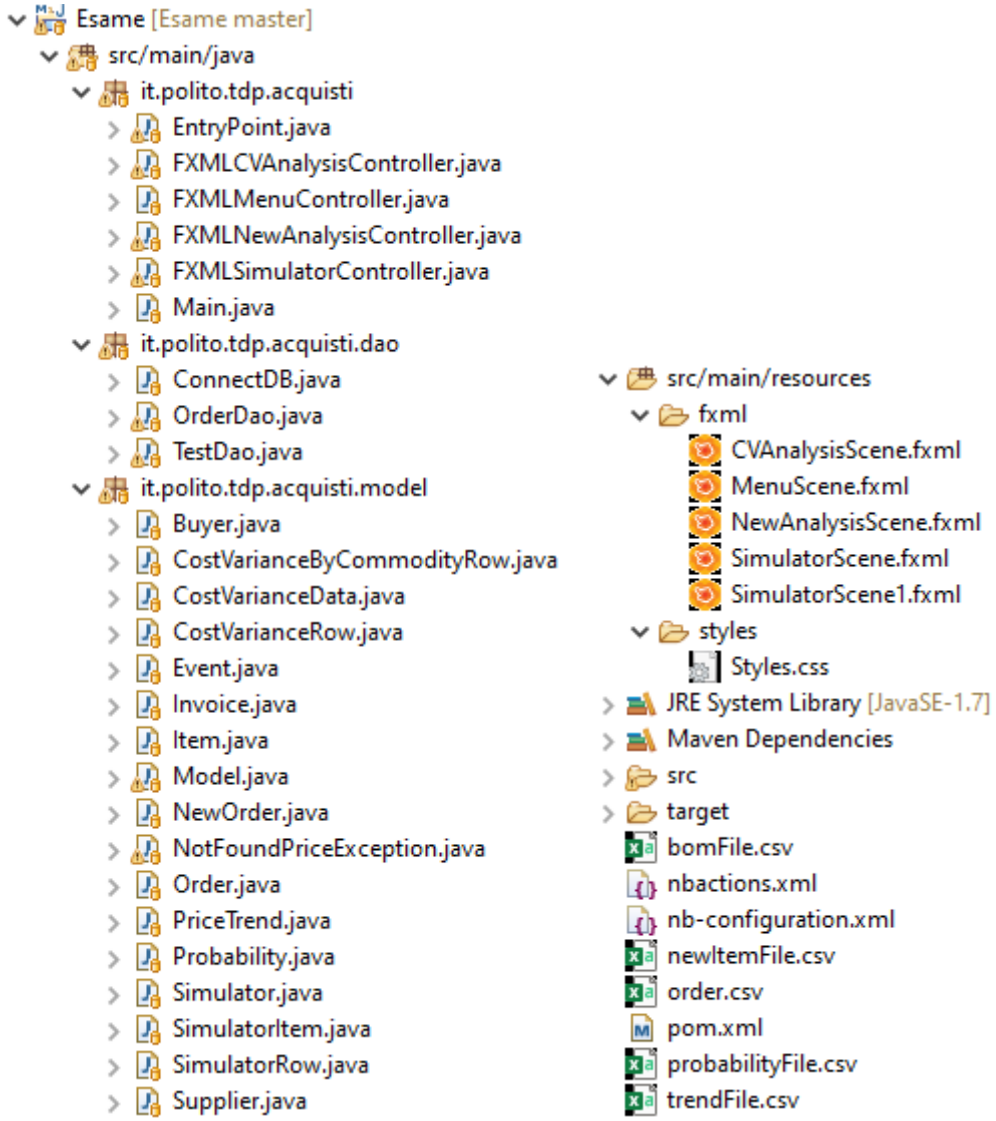
Vi sono altre tre tabelle utili al funzionamento del software

- warehouse : contiene i prodotti presenti a magazzino
- trend : contiene i prezzi per ciascun prodotto per un dato periodo
- bom : contiene la distinta base di alcuni articoli

Capitolo 4: Descrizione strutture dati e algoritmi utilizzati

4.1 Strutture dati

Il software è stato programmato in Java, usando il pattern MVC (Model View Controller) e il pattern DAO (Data Access Object). È stato usato il software SceneBuilder per realizzare le interfacce grafiche in JavaFX. Come consigliato dai pattern MVC e DAO, il progetto è costituito da tre package per separare il modello, l'accesso al database e l'interfaccia utente. All'interno del progetto sono presenti anche fogli di calcolo Excel salvati in formato .csv, per poter essere elaborati dal programma come file di testo.



Il package it.polito.tdp.acquisti contiene la classe principale del progetto, ovvero la classe Main, la classe EntryPoint, nella quale viene creato un nuovo modello passato al Controller del Menù.

Il package it.polito.tdp.acquisti.dao contiene le classi necessarie per l'accesso ai dati presenti nel database.

Il package it.polito.tdp.acquisti.model contiene la classe logica principale dell'applicazione (Model), la classe deputata alla simulazione (Simulator) e le classi di oggetti create per il funzionamento del programma.

4.2 Algoritmi principali

I dati in possesso del model vengono inizialmente separati in due mappe, una contenente i dati dell'anno corrente e l'altra i dati dell'anno precedente.

```
private void uploadRepetitiveness(LocalDate modelDate){
    for(Order o : orderMap.values()){
        if(o.getInvoice().getInvoiceDate().getYear() < modelDate.getYear() && o.getInvoice().getInvoiceDate().getYear() > (modelDate.getYear() - 2)){
            o.getItem().setPreviouslyPurchased(true);
            this.pastPeriodOrderMap.put(o.getOrderId(), o);
        } else{
            o.getItem().setCurrentlyPurchased(true);
            this.currentPeriodOrderMap.put(o.getOrderId(), o);
        }
    }
}
```

I dati vengono elaborati per entrambi gli anni e memorizzati in opportune strutture dati, che contengono una sintesi dei dati necessari per l'analisi. Di seguito l'algoritmo per ottenere i dati dell'anno precedente.

```
private void getPastPeriodCostVarianceData(Map<Item, CostVarianceData> pastCVDByItemMap, Map<String,
        HashMap<Item, CostVarianceData>> pastCVDBySiteMap, Map<Buyer, HashMap<Item, CostVarianceData>> pastCVDByBuyerMap){

    for(Order o : this.pastPeriodOrderMap.values()){
        Item item = o.getItem();
        Invoice invoice = o.getInvoice();
        LocalDate purchaseDate = invoice.getInvoiceDate();
        String siteId = o.getSiteId();
        Buyer buyer = o.getBuyer();

        if(pastCVDByItemMap.containsKey(item) ){
            CostVarianceData cvd = pastCVDByItemMap.get(item);
            if(cvd.getLastPurchaseDate().isBefore(purchaseDate)){
                cvd.setLastPurchaseDate(purchaseDate);
                cvd.setLastPrice(invoice.getUnitPurchasePrice());
            }
            cvd.setQuantity(cvd.getQuantity() + invoice.getQuantity());
            cvd.setTotalPrice(cvd.getTotalPrice() + invoice.getTotalPurchasePrice());
        } else {
            pastCVDByItemMap.put(item, new CostVarianceData(item, invoice.getTotalPurchasePrice(),
                    invoice.getQuantity(), invoice.getUnitPurchasePrice(), purchaseDate) );
        }

        if(pastCVDBySiteMap.containsKey(siteId)) {
            HashMap<Item, CostVarianceData> thisPastCVDByItemMap = pastCVDBySiteMap.get(siteId);
            if(thisPastCVDByItemMap.containsKey(item) ){
                CostVarianceData cvd = thisPastCVDByItemMap.get(item);
                if(cvd.getLastPurchaseDate().isBefore(purchaseDate)){
                    cvd.setLastPurchaseDate(purchaseDate);
                    cvd.setLastPrice(invoice.getUnitPurchasePrice());
                }
                cvd.setQuantity(cvd.getQuantity() + invoice.getQuantity());
                cvd.setTotalPrice(cvd.getTotalPrice() + invoice.getTotalPurchasePrice());
            } else {
                thisPastCVDByItemMap.put(item, new CostVarianceData(item, invoice.getTotalPurchasePrice(),
                        invoice.getQuantity(), invoice.getUnitPurchasePrice(), purchaseDate) );
            }


        } else {
            pastCVDBySiteMap.put(siteId, new HashMap<Item, CostVarianceData>());
            pastCVDBySiteMap.get(siteId).put(item, new CostVarianceData(item, invoice.getTotalPurchasePrice(),
                    invoice.getQuantity(), invoice.getUnitPurchasePrice(), purchaseDate) );
        }

        if(pastCVDByBuyerMap.containsKey(buyer)) {
            HashMap<Item, CostVarianceData> thisPastCVDByItemMap = pastCVDByBuyerMap.get(buyer);
            if(thisPastCVDByItemMap.containsKey(item) ){
                CostVarianceData cvd = thisPastCVDByItemMap.get(item);
                if(cvd.getLastPurchaseDate().isBefore(purchaseDate)){
                    cvd.setLastPurchaseDate(purchaseDate);
                    cvd.setLastPrice(invoice.getUnitPurchasePrice());
                }
                cvd.setQuantity(cvd.getQuantity() + invoice.getQuantity());
                cvd.setTotalPrice(cvd.getTotalPrice() + invoice.getTotalPurchasePrice());
            } else {
                thisPastCVDByItemMap.put(item, new CostVarianceData(item, invoice.getTotalPurchasePrice(),
                        invoice.getQuantity(), invoice.getUnitPurchasePrice(), purchaseDate) );
            }
        } else {
            pastCVDByBuyerMap.put(buyer, new HashMap<Item, CostVarianceData>());
            pastCVDByBuyerMap.get(buyer).put(item, new CostVarianceData(item, invoice.getTotalPurchasePrice(),
                    invoice.getQuantity(), invoice.getUnitPurchasePrice(), purchaseDate) );
        }
    }
}
```

I dati degli ordini passati e attuali vengono incrociati per vedere se risulta possibile calcolare una variazione di costo, ciò risulta sicuramente vero se il prodotto fosse stato acquistato sia nell'anno precedente sia nell'anno corrente. Successivamente i risultati vengono mandati al controller.

```
private List<CostVarianceRow> computeAllCostVarianceByItemRows(Map<Item, CostVarianceData> pastCVDByItemMap, Map<Item, CostVarianceData> currentCVDByItemMap){
    List<CostVarianceRow> result = new ArrayList<CostVarianceRow>();

    for(Item i : this.itemMap.values() ){
        if(i.isPreviouslyPurchased() ){
            if(i.isCurrentlyPurchased() ){
            //compute Cost Variance
            CostVarianceData pastCVD = pastCVDByItemMap.get(i);
            CostVarianceData currentCVD = currentCVDByItemMap.get(i);
            double currentQuantity = currentCVD.getQuantity();

            double averageCostVariance = (currentCVD.getAveragePrice() - pastCVD.getAveragePrice()) * currentQuantity;
            double averagePercentageCostVariance = averageCostVariance / (pastCVD.getAveragePrice() * currentQuantity );
            double lastCostVariance = (currentCVD.getLastPrice() - pastCVD.getLastPrice()) * currentQuantity;
            double lastPercentageCostVariance = lastCostVariance / (pastCVD.getLastPrice() * currentQuantity );

            result.add(new CostVarianceRow(i.getItemId(), i.getItemDescription(), currentCVD.getTotalPrice(), null, null,
                    averageCostVariance, averagePercentageCostVariance , lastCostVariance , lastPercentageCostVariance) );
            } else{
            }
        }else if(i.isCurrentlyPurchased()){
        }
    }

    return result;
}
```

Per quanto riguarda il simulatore, questo processa diversi eventi.

Evento finding : vengono cercati i prodotti assenti

```
case FINDING:
    for(SimulatorItem toFind : this.absentItems.values()) {
        if(toFind.getState() == State.ABSENT) {
            toFind.setState(State.IN_FINDING);
            this.queue.add(new Event(today.plus(DURATION_FINDING), EventType.PURCHASE, toFind));
        }
    }
    break;
```

Evento purchase : vengono acquistati gli articoli ad un determinato prezzo.

```
case PURCHASE:
    item.setState(State.PURCHASED);
    String itemId = item.getItemId();
    Period DURATION_LEAD_TIME;
    if(leadTimeByItemId.containsKey(itemId))
        DURATION_LEAD_TIME = leadTimeByItemId.get(itemId);
    else
        DURATION_LEAD_TIME = DURATION_DEFAULT_LEAD_TIME;
    double num = Math.random();
    Probability probability = this.probabilities.get(itemId);
    double delayProbability = probability.getDelayProbability();
    double annulmentProbability = probability.getAnnulmentProbability();
    if(num < delayProbability) {
        List<PriceTrend> priceTrends = this.trendsByItem.get(itemId);
        int pricetrendsSize = priceTrends.size();
        if(pricetrendsSize > 1) {
            for(int i = 0; i < pricetrendsSize - 1; i++) {
                if(today.isAfter(priceTrends.get(i).getDate()) && today.isBefore(priceTrends.get(i + 1).getDate())) {
                    item.setPurchasePrice(priceTrends.get(i).getPrice());
                }
            }
        } else if(today.isAfter(priceTrends.get(pricetrendsSize - 1).getDate())){
            item.setPurchasePrice(priceTrends.get(pricetrendsSize - 1).getPrice());
        }
        item.setId(purchaseId);
        purchaseId ++;
        queue.add(new Event(today.plus(DURATION_LEAD_TIME), EventType.DELAY, item));
    }
    else if(num < delayProbability + annulmentProbability) {
        probability.setAnnulmentProbability(0);
        probability.setDelayProbability(0);
        queue.add(new Event(today.plus(DURATION_LEAD_TIME), EventType.ANNULMENT, item));
    }
    else {
        List<PriceTrend> priceTrends = this.trendsByItem.get(itemId);
        int pricetrendsSize = priceTrends.size();
        if(pricetrendsSize > 1) {
            for(int i = 0; i < pricetrendsSize - 1; i++) {
```

```
else {
    List<PriceTrend> priceTrends = this.trendsByItem.get(itemId);
    int pricetrendsSize = priceTrends.size();
    if(pricetrendsSize > 1) {
        for(int i = 0; i < pricetrendsSize - 1; i++) {
            if(today.isAfter(priceTrends.get(i).getDate()) && today.isBefore(priceTrends.get(i + 1).getDate())) {
                item.setPurchasePrice(priceTrends.get(i).getPrice());
            }
        }
    } else if(today.isAfter(priceTrends.get(pricetrendsSize - 1).getDate())){
        item.setPurchasePrice(priceTrends.get(pricetrendsSize - 1).getPrice());
    }
    item.setId(purchaseId);
    purchaseId ++;
    queue.add(new Event(today.plus(DURATION_LEAD_TIME), EventType.ARRIVAL, item));
}
break;
```

Evento Arrival : gli oggetti arrivano al magazzino.

Evento Delay : Gli oggetti in ritardo entrano nel magazzino dopo un ritardo.

Evento Annulment: Imposta lo stato degli oggetti ad assenti perciò dovranno essere riacquistati.

Evento taking : gli oggetti saranno prelevati dal magazzino e arrivano sulla linea di produzione.

```
case ARRIVAL:
    item.setState(State.WAREHOUSE_ARRIVAL);
    item.setArrivalDate(today);
    this.warehouse.add(item);
    break;
case DELAY:
    queue.add(new Event(today.plus(DURATION_DELAY), EventType.ARRIVAL, item));
    break;
case ANNULMENT:
    item.setState(State.ABSENT);
    break;
case TAKING:
    List<SimulatorItem> toTake = new ArrayList<SimulatorItem>();
    for(SimulatorItem si : this.warehouse) {
        String siId = si.getItemId();
        if(this.absentItems.keySet().contains(siId)) {
            SimulatorItem absentItem = this.absentItems.get(siId);
            int absentItemQty = absentItem.getQuantity();
            int siQty = si.getQuantity();
            if(absentItemQty < siQty) {
                si.setQuantity(siQty - absentItemQty);
                this.absentItems.remove(siId);
                this.lineItem.add(new SimulatorItem(SimulatorItem.State.LINE_ARRIVAL, si.getArrivalDate(), si.getId(), si.getPurchasePrice(), siId, absentItemQty));
            }else if(absentItemQty == siQty) {
                toTake.add(si);
                this.absentItems.remove(siId);
                this.lineItem.add(new SimulatorItem(SimulatorItem.State.LINE_ARRIVAL, si.getArrivalDate(), si.getId(), si.getPurchasePrice(), siId, absentItemQty));
            }else {
                absentItem.setQuantity(absentItemQty - siQty);
                toTake.add(si);
                this.lineItem.add(new SimulatorItem(SimulatorItem.State.LINE_ARRIVAL, si.getArrivalDate(), si.getId(), si.getPurchasePrice(), siId, siQty));
            }
        }
    }
    this.warehouse.removeAll(toTake);
    if(this.absentItems.keySet().size() == 0)
        queue.add(new Event(today.plus(DURATION_PRODUCTION), EventType.SALE, null));
    break;
```

Evento Sale : la lavorazione del prodotto è finita e quest'ultimo può essere venduto.

```
case SALE:
    for(SimulatorItem si : this.lineItem) {
        si.setState(State.OUT);
        this.totalPrice += (si.getPurchasePrice()*si.getQuantity());
    }
    queue.clear();
    saleDate = today;
    System.out.println(today);
    break;

default:
    break;
}
```

Capitolo 5: Diagramma delle classi delle parti principali

Model
Map<String, Order> orderMap; Map<String, Order> currentPeriodOrderMap; Map<String, Order> pastPeriodOrderMap; Map<String, Buyer> buyerMap; Map<String, Supplier> supplierMap; Map<String, Item> itemMap; Map<String, Invoice> invoiceMap; Set<String> commoditySet; OrderDao dao; LocalDate modelDate; List<CostVarianceRow> costVarianceByItemRows; List<CostVarianceRow> costVarianceBySiteRows; List<CostVarianceRow> costVarianceByBuyerRows; List<CostVarianceByCommodityRow> costVarianceByCommoditiesRows; Set<String> buyerWithCVRow; Set<String> siteWithCVRow; Set<String> commodityWithCVRow; Map<Item, CostVarianceData> pastCVDByItemMap; Map<Item, CostVarianceData> currentCVDByItemMap; Simulator simulator; List<SimulatorItem> actual; Map<String, Integer> bom; Map<String, Probability> probabilities; Map<String , List<PriceTrend>> trendsByItem; LocalDate startDate;
public Model public void init private void uploadRepetitiveness private void getPastPeriodCostVarianceData private void getCurrentPeriodCostVarianceData private List<CostVarianceRow> computeAllCostVarianceByItemRows private List<CostVarianceRow> computeAllCostVarianceBySiteRows private List<CostVarianceRow> computeAllCostVarianceByBuyerRows private List<CostVarianceByCommodityRow> computeAllCostVarianceByCommodityRows public List<CostVarianceRow> getAllCostVarianceByItemRows public List<CostVarianceRow> getRowsWith public List<CostVarianceRow> getAllCostVarianceBySiteRows public List<CostVarianceRow> getAllCostVarianceByBuyerRows public List<CostVarianceRow> getAllCostVarianceByCommodityRows public Set<String> getAllBuyersWithCVRow public Set<String> getAllSitesWithCVRow public List<NewOrder> readFile public void addNewOrders public void simulate public void reset

Simulator

List<SimulatorRow> simulatorRowList;

LocalDate saleDate;

PriorityQueue<Event> queue;

List<SimulatorItem> warehouse;

List<SimulatorItem> lineItem;

Map<String, SimulatorItem> absentItems;

int purchaseId;

List<SimulatorItem> actualWarehouse;

Map<String, Integer> bom;

Period DURATION_FINDING;

Period DURATION_DEFAULT_LEAD_TIME;

Map<String, Period> leadTimeByItemId;

Period DURATION_DELAY;

Period DURATION_PRODUCTION;

Period CHECK_INTERVAL ;

LocalDate startTime ;

LocalDate endTime ;

Map<String, Probability> probabilities;

Map<String , List<PriceTrend>> trendsByItem;

double totalPrice;

---

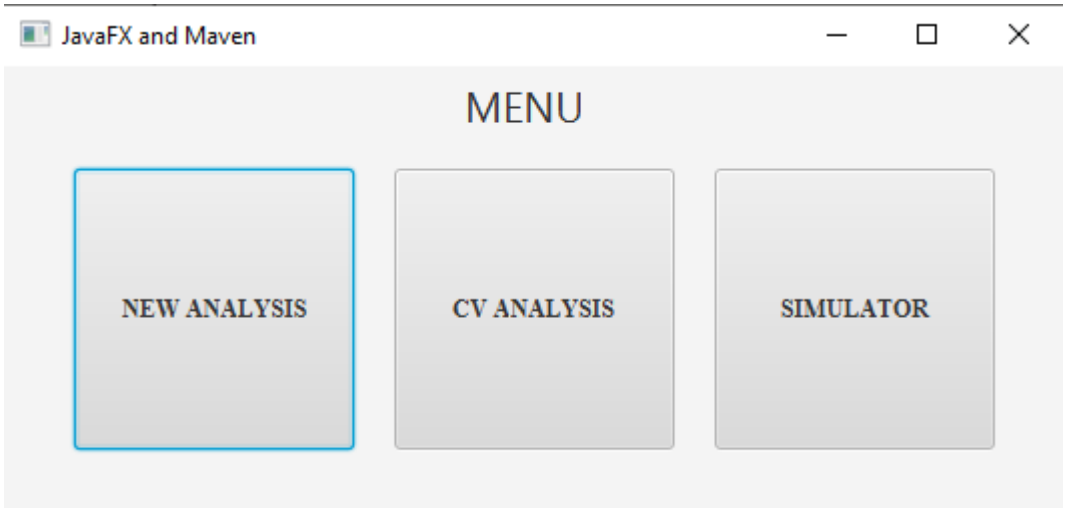
public void init
public void run
private void processEvent
public double getTotalPrice
public void setActualWarehouse
public void setBom
public void setDURATION_FINDING
public void setDURATION_DEFAULT_LEAD_TIME
public void setDURATION_DELAY
public void setDURATION_PRODUCTION
public void setCHECK_INTERVAL
public List<SimulatorRow> getSimulatoRowList
public LocalDate getSaleDate

Event
private LocalDate time private EventType type private SimulatorItem item
public Event public LocalDate getTime public void setTime public EventType getType public void setType public SimulatorItem getItem public void setItem public int compareTo public String toString()

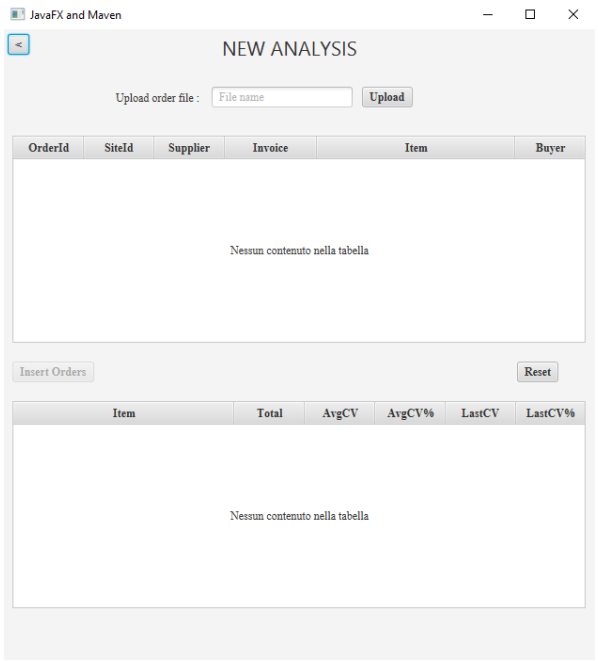
Capitolo 6: Videate dell'applicazione realizzata

6.1 Menù

La schermata principale del software è il menù, dal quale l'utente può aprire altre tre schermate tramite i rispettivi bottoni.



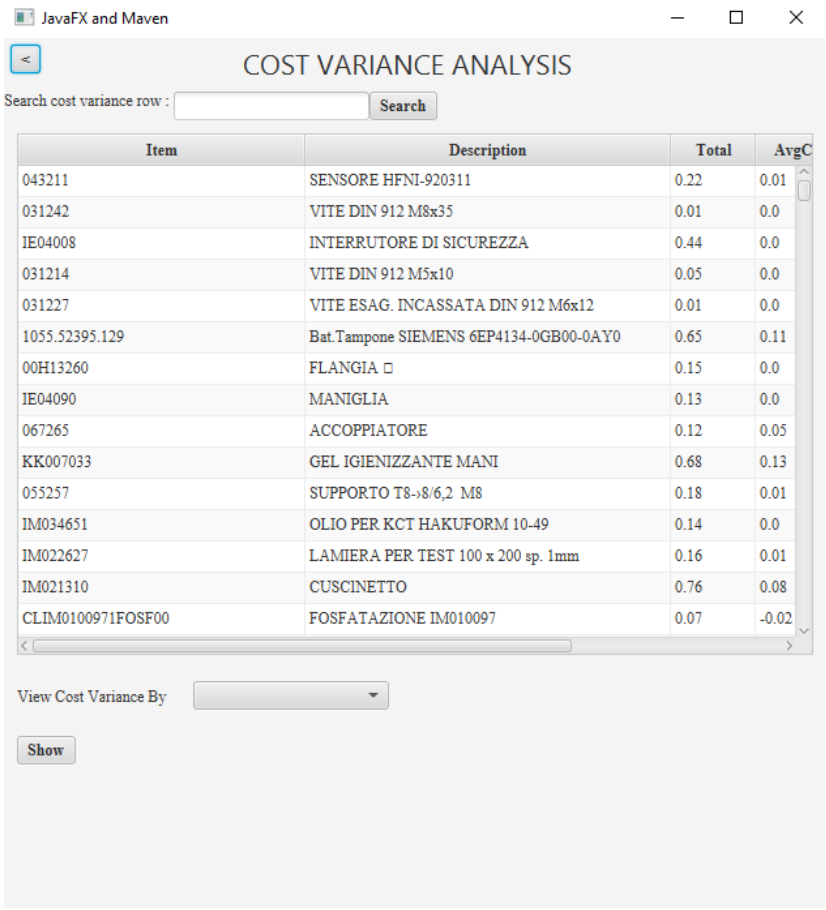
6.2 New Analysis



Dopo aver selezionato il pulsante New Analysis si potranno caricare dei nuovi ordini con un file .csv da importare nella cartella del progetto. La scelta di accettare i file .csv è dettata dall'ampia diffusione di Microsoft Excel, il software di fogli di calcolo leader di settore. Dopo aver selezionato Upload, sarà possibile

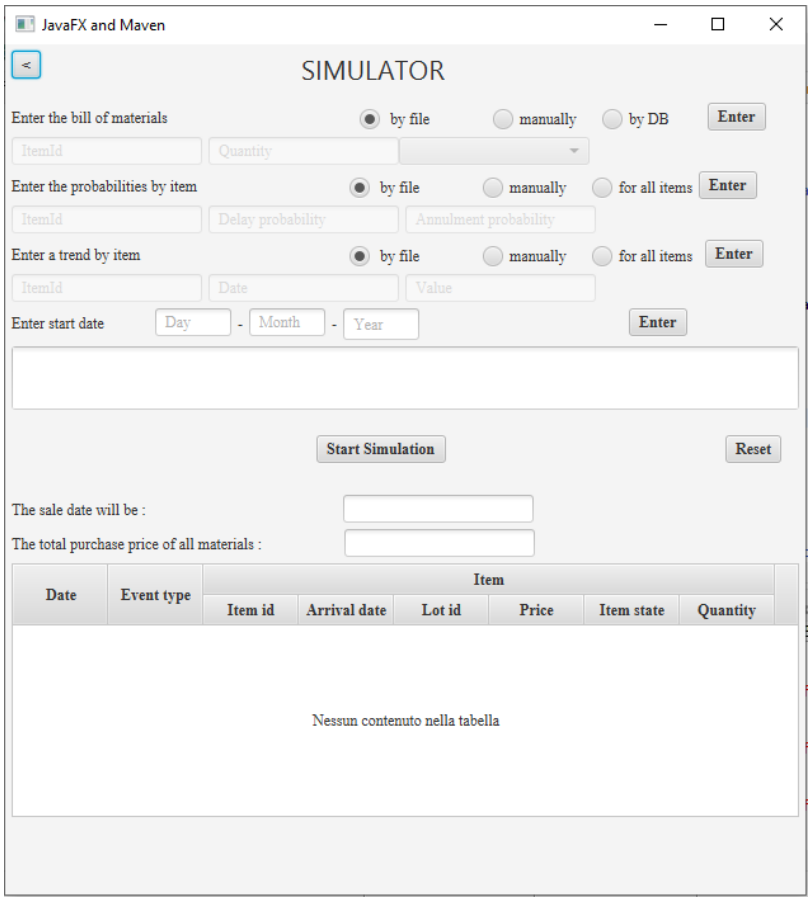
visualizzare in una tabella la correttezza dei dati inseriti e solo dopo aver cliccato Insert Orders verrà aggiornato il database con i nuovi ordini.

6.3 CV Analysis



Dopo aver selezionato il pulsante CV Analysis l'utente visualizzerà la schermata dalla quale potrà monitorare i risultati dell'analisi della Cost Variance. Inoltre potrà scegliere da Combo Box se visualizzare i risultati per articolo (default), sito, buyer o commodity.

6.4 Simulator



Dopo aver selezionato il pulsante Simulator, verrà aperta la schermata della simulazione dalla quale l'utente potrà inserire i dati manualmente o da file .csv, già presenti all'interno della cartella e da modificare con i nuovi dati per caricarli.

6.5 Video dimostrativo

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

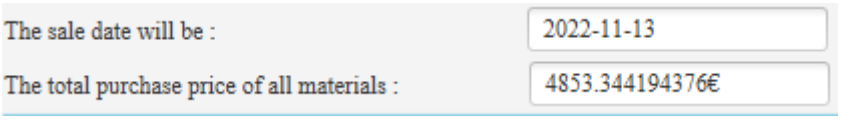
https://youtu.be/O4U3TtbLUH0

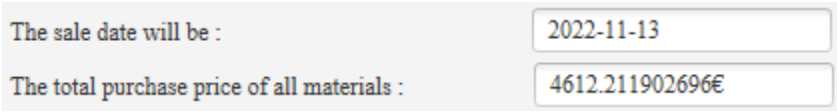
Capitolo 7: Risultati

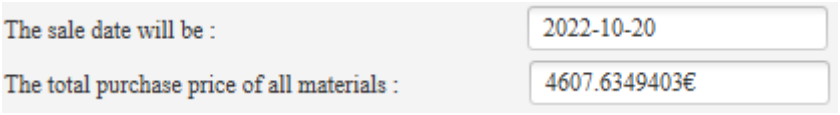
7.1 Valori e tabelle dei risultati sperimentali ottenuti

I tempi e i risultati ottenuti dalle varie elaborazioni del software sono riportati di seguito.

Inserendo nel simulatore una distinta base presente nel database, le probabilità di ritardo e annullamento, ho ottenuto i risultati sottostanti. Si può notare come diverse simulazioni portino a risultati differenti sia in termini di data di vendita che di prezzo totale.







Il tempo di elaborazione della simulazione è pari a `364 millisecondi.`

Capitolo 8: Considerazioni finali

8.1 Valutazioni sui risultati ottenuti

Nel complesso i risultati ottenuti dall'esecuzione del programma sono positivi. L'esecuzione sia dell'analisi dei dati, sia della simulazione non avviene in tempi lunghi. I risultati sono coerenti ai problemi aziendali descritti al punto 3 e dunque utili per supportare il management nelle decisioni manageriali.

Le criticità principali del software derivano dall'uso di semplificazioni delle operazioni e dei processi aziendali e dal rispetto delle istruzioni di uso del programma da parte dell'utente.

8.2 Conclusioni

Gli obiettivi principali posti al programma sono stati raggiunti, il software funziona e può essere sfruttato da tutti gli uffici acquisti italiani e internazionali. Infatti la lingua usata per sviluppare l'applicazione è l'inglese. Soluzioni personalizzate, dedicate alle aziende potranno essere implementate sviluppando ulteriormente il programma. Tuttavia questa versione base del programma supporterà comunque le aziende, che decideranno di utilizzarlo, nel processo di approvvigionamento e gestione aziendale.