Relazione Progetto "UnieurOOP"
Programmazione ad Oggetti

Ferri Samuele, Iorio Matteo, Strada Nicola, Vincenzi Fabio

Febbraio - Aprile 2022

Indice

1 Analisi Progetto

Il nostro progetto consiste nella realizzazione di un software gestionale, che ha come destinatario la catena Unieuro, nel dettaglio ci concentriamo specificatamente sul famoso negozio di Gambettola Unieuro, locale specializzato nella vendita di prodotti di elettrodomestica ed elettronica.

1.1 Requisiti

Requisiti Funzionali: Lo sviluppo del nostro software si pone come obiettivo la gestione di tutto ciò che riguarda l'attività Unieuro. Di seguito analizziamo alcuni punti principali.

- Verrà gestita la vendita dei vari prodotti verso i clienti del negozio. Un cliente, potrà acquistare più prodotti appartenenti anche a diverse categorie, inoltre per un cliente non sarà necessaria, ma bensì opzionale, la sua registrazione nel portale in questione.

Da qui segue la necessità di una corretta gestione di tutti i vari reparti presenti nel negozio.

- L'amministrazione dei vari reparti consiste nella capacità di doverli rifornire con prodotti specifici adibiti ai reparti stessi.
- L'assegnazione dei vari dipendenti in uno specifico reparto.
- Lo scambio di informazioni che avviene tra il il magazzino del negozio e i suoi vari reparti.

Approfondiamo ora la gestione del *magazzino*; questo sarà riservato per lo stoccaggio dei vari prodotti, i quali poi verranno spostati, in quantità utile, negli appositi reparti.

- I dipendenti avranno la possibilità di accedere allo stoccaggio prodotti in magazzino e a seconda della disponibilità potranno rifornire il reparto, ipoteticamente sprovvisto o in quantità insufficiente, di un certo prodotto.
- Verrà garantita una cooperazione tra i vari reparti ed il magazzino, in modo da gestire eventualmente richieste, da parte di clienti, di prodotti con quantità maggiori di quelle presenti nel reparto. Conseguentemente si avrà possibilità di completare l'ordine prelevando la parte rimanente dal magazzino.
- Dovrà inoltre essere fondamentale l'approvvigionamento del magazzino. Quando le scorte inizieranno a terminare, si procederà all'esecuzione di ordini verso i vari fornitori del negozio. Sarà quindi possibile acquistare una quantità X di prodotti per un prezzo Y.

Si prevede inoltre la gestione dello Staff presente all'interno dell'azienda, questo per un'eventuale organizzazione dello scheduling lavorativo e per una corretta autenticazione alla applicazione del negozio.

- Tutto il personale sarà perciò predisposto di un set di credenziali composto da: Email, la quale funge da username univoco, e Password. Utili per una corretta autenticazione.
- Tutto il personale, una volta autenticatosi, potrà svolgere una serie di operazione tra cui per esempio vendita di prodotti, visualizzazione di eventuali statistiche, rifornimento di reparti...

Requisiti non Funzionali: Si prevede un prodotto estremamente accessibile e facile da utilizzare, questo per una massima efficienza nel lavoro. Non vi dovranno essere ambiguità di alcun tipo, per evitare di creare inconsistenze o problemi di natura ancora più grave. Questo permetterà ai vari dipendenti, indipendentemente dalle loro conoscenze, di essere in grado di utilizzare il nostro software senza alcuna conoscenza pregressa.

1.2 Analisi e Modello del Dominio

Procediamo adesso con una dettagliata analisi per quanto riguarda il dominio applicativo della nostra applicazione. Il negozio per il quale verrà erogato il servizio è *Unieuro*, la quale amministrazione è divisa tra i diversi dipendenti, ognuno dei quali è poi assegnato ai vari reparti del negozio. Il chiaro fine del negozio è quello di fornire e vendere prodotti ai vari clienti. Tali prodotti, come già espresso in precedenza, si troveranno in locazioni diverse del negozio, nel loro specifico reparto, a seconda della tipologia del prodotto. Per esempio un telefono lo si potrà trovare nel reparto "*Telefonia*", mentre una lavatrice sarà presente nel reparto "*Idraulica*" e così di seguito. Uno dei punti chiave di un negozio è il poter evolvere nel tempo, quindi sarà necessario poter eliminare ed aggiungere nuovi reparti al nostro negozio, con i relativi prodotti. Un problema però potrebbe sorgere quando si va ad eliminare un reparto; i prodotti della sezione eliminata che fine andrebbero a fare? e i dipendenti assegnati ad esso?

Per risolvere il problema sarà quindi necessario allocare tutti i prodotti del reparto appena eliminato nel magazzino del negozio (Stock). Mentre per quanto riguarda i dipendenti, questi potranno essere riassegnati in futuro ad altri reparti già esistenti, evitando così di perdere risorse preziose.

Passiamo quindi ad uno dei punti forti del nostro software, il controllo della vendita. Una parte fondamentale per il ciclo di vita di un negozio è la vendita dei suoi beni, si deve rendere possibile perciò l'acquisto, da parte di un cliente, di uno o più prodotti diversi tra loro. La vendita avviene da parte dello staff, il quale accedendo alla sezione "*Sales*" sarà in grado di dichiarare alcuni prodotti come venduti.
Tutto ciò poi servirà anche per rendere possibile la creazione dei vari grafici statistici, ottenendo informazioni sui periodi migliori di vendita, sui reparti e i vari prodotti di tendenza, e molto altro ancora. Possiamo osservare la

costruzione di un semplice *UML*, il quale verrà implementato adottando una visione del modello **MVC**, ovvero previsto di un *Model*, un *Controller*, e una *View*, fare riferimento allo schema 1.

2 Design Applicativo

2.1 Introduzione

In questo capitolo osserviamo le strategie utilizzate per soddisfare al meglio i requisiti identificati nell'analisi. Si parte da una **Visione Architetturale**, il cui scopo è informare il lettore di quale sia il funzionamento dell'applicativo realizzato, ad un livello di alta astrazione. In particolare risulta necessario descrivere accuratamente in che modo i componenti principali del sistema si coordinano fra loro. A seguire, si dettagliano alcune parti del design, quelle maggiormente rilevanti al fine di chiarificare la logica con cui sono stati affrontati i principali aspetti dell'applicazione.

2.2 Architettura

Come accennato in precedenza l'architettura impiegata nel progetto si avvale del pattern **MVC**, composto da *Model-View-Controller*. La parte sostanziale sarà strutturata dal **Model**, che racchiude lo "scheletro" dell'applicativo, fornendo i metodi per accedere a tutti i suoi dati utili. Le varie **View**, costruite a seconda del loro utilizzo, permetteranno l'interazione con l'utente catturando ogni sua azione, le quali produrranno informazioni significative, gestite poi dai relativi **Controller**. Questi avvalendosi poi degli altri due componenti genereranno Output destinati ad una successiva visualizzazione nelle corrette View. Di seguito elencati troviamo gli 11 Controller, vediamo, per ciascuno, di cosa si occupano e quali elementi utilizzano:

- Controller per le statistiche: (*unieuroop.controller.analytic*)
- Controller per i clienti: (*unieuroop.controller.client*)
- Controller per il menù principale: (*unieuroop.controller.dashboard*)
- Controller per i reparti: (*unieuroop.controller.department*)
- Controller per le fatture: (*unieuroop.controller.invoice*)
- Controller per il login: (*unieuroop.controller.login*)
- Controller per le vendite: (*unieuroop.controller.sale*)
- Controller per la serializzazione: (*unieuroop.controller.serialization*)
- Controller per lo shop: (*unieuroop.controller.shop*)
- Controller per lo staff: (*unieuroop.controller.staff*)
- Controller per lo stock: (*unieuroop.controller.stock*)

In conclusione, lato View avremmo una pagina per ciascuna operazione che l'utente voglia eseguire. Le varie *Pagine*; quali spazieranno da *Gestione dello Stock*, *Gestione dei Reparti*, *Analytics*, produrranno input come per esempio: aggiunta e rimozione prodotti, modifiche ai reparti, vendita prodotti, calcolo di eventuali statistiche; tutti processati poi dallo specifico Controller, che garantisce inoltre la correttezza del dato finale prodotto.

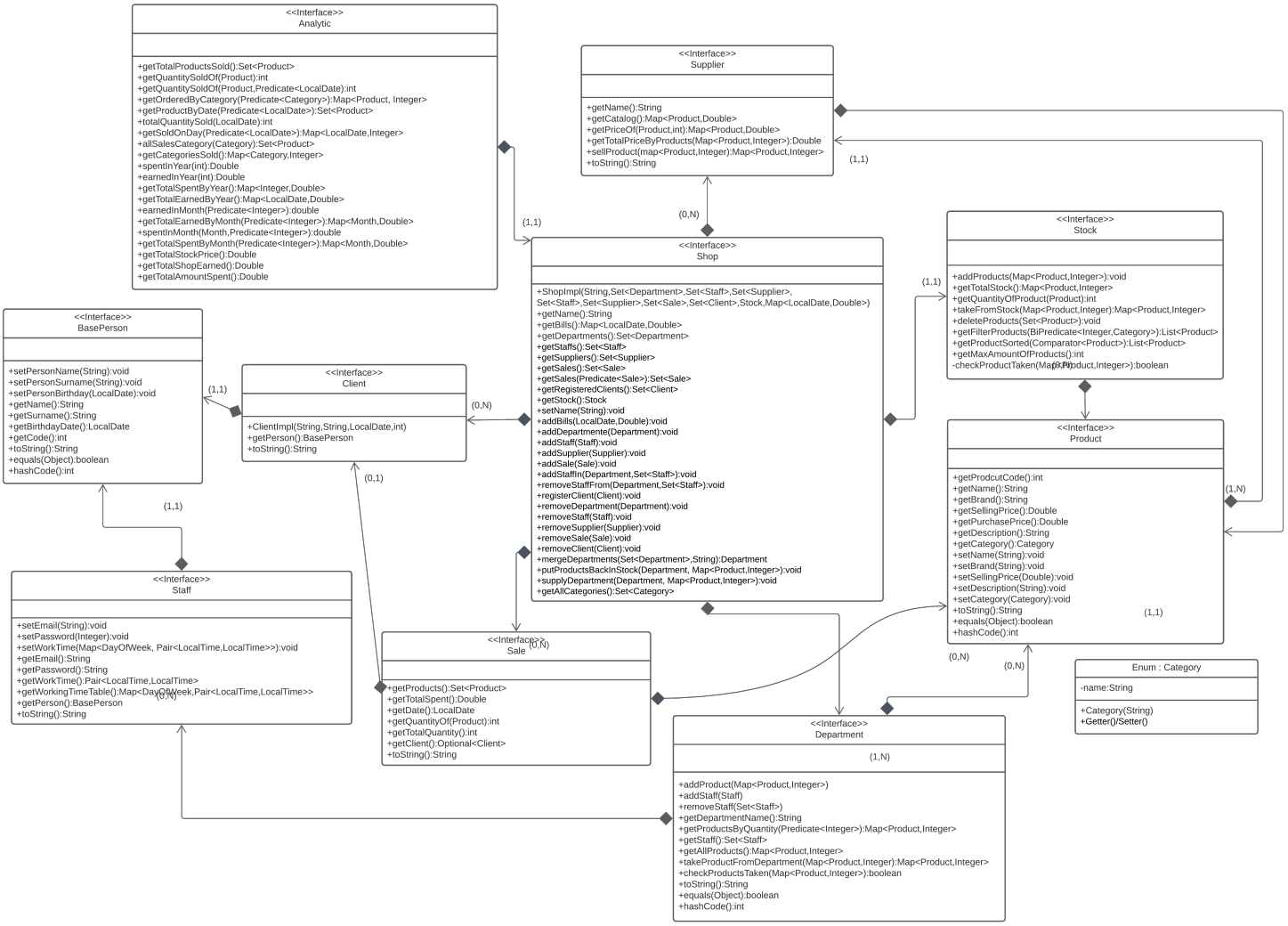

Figure 1: UML Project

2.3 Design Dettagliato

2.3.1 Samuele Ferri

Problema: Inizialmente durante lo sviluppo del *"Package Person"*, come previsto in analisi (Sezione 1), si era appunto progettato l'inserimento di una Classe Astratta, *AbstractPerson*, che risolveva la ripetizione di alcuni dati nelle classi *Staff e Client*, come: *Name, Surname, Birthday, Code*. Questa classe però, per ragioni progettuali non presentava alcun metodo astratto, generando così un errore concettuale poiché non rispettava gli stessi principi di una classe astratta.

Soluzione: Ho quindi optato per la **Composizione**, rinominando la nostra classe in *BasePerson* e riadattandola come un attributo delle stesse classi che prima la estendevano, ovvero *Client e Staff*. In questo modo non si perdeva nessuna informazione, e contemporaneamente, si risolve l'inconsistenza.

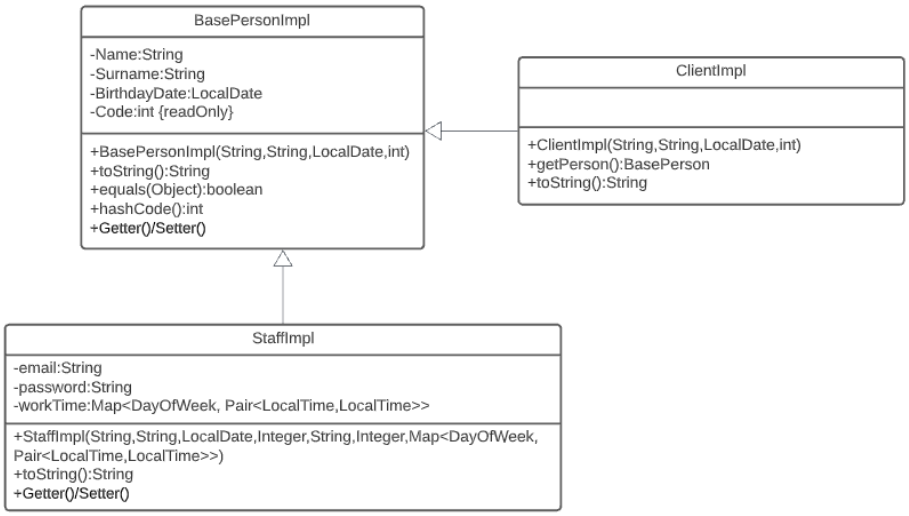

Figure 2: UML della composizione

Problema: Poter riuscire a filtrare i prodotti di un reparto in base alla loro quantità.

Soluzione: Utilizzare come parametro della funzione un Predicate, il quale lavorando sulle quantità dei prodotti, riesce a filtrare ed ottenere un risultato in maniera efficiente.

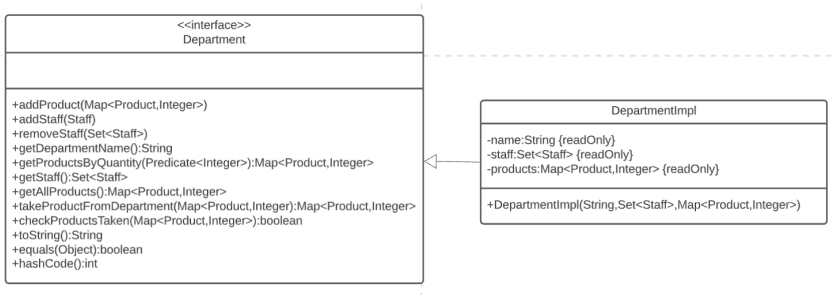

Figure 3: UML Department

2.3.2 Matteo Iorio

Problema: All'interno della classe *Analytic*, sorgeva la necessità di sviluppare una strategia efficace con la quale filtrare l'insieme dei prodotti venduti nelle diverse vendite.

Soluzione: Il mio primo approccio è stato nell'utilizzare una collezione di *Categorie*, in modo da poter classificare i prodotti venduti in base a quella posseduta. Successivamente però la soluzione finale che ho adottato è cambiata; Ho impiegato come parametro per la funzione un *Predicate*, il quale facesse riferimento alle *Categorie*, così facendo sono riuscito ad ottenere una maggiore scalabilità ed una grande efficienza.

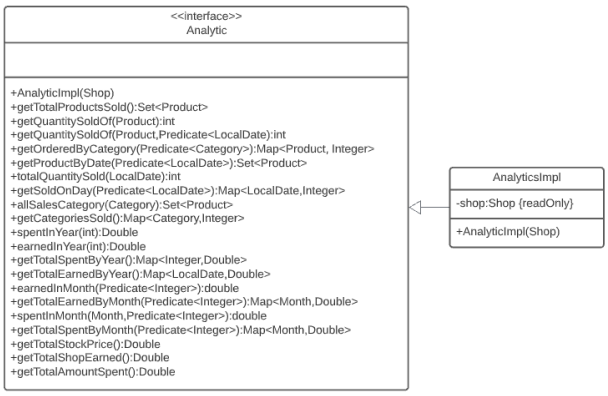

Figure 4: UML Analytic

Problema: Seguendo una falsa riga di ciò che realmente accade in un negozio, durante una vendita il Cliente può scatenare due macro-eventi:

1. Il Cliente è già registrato all'interno dell'applicazione, perciò disponiamo già dei suoi dati personali ovvero Nome, Cognome, Data di Nascita e Codice.

2. Contrariamente si può verificare la situazione in cui la vendita sia assegnata ad un Cliente non registrato all'interno del negozio.

Soluzione: All'interno della classe *Sale* sarà quindi gestito questo evento, dove il Cliente, esecutore delle vendita, può non essere presente all'interno dell'applicazione. Inizialmente, come prima idea abbozzata, si pensava di poter lasciare semplicemente l'attributo *Client*, all'interno di *Sale*, col valore *"Null"*. Assumendoci così una grande responsabilità, potendo richiamare metodi su un possibile attributo nullo. La mia soluzione finale perciò è stata quella di utilizzare un attributo di tipo *Optional* all'interno di *Sale*, quindi in conclusione se il Cliente non è presente, sarà sufficiente impostare tale attributo ad *Empty*.

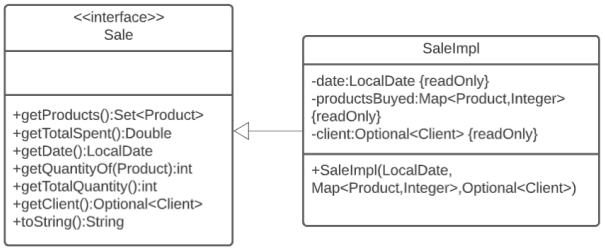

Figure 5: UML Sale

Problema: Un altro problema, nel quale mi sono cimentato, riguarda la creazione di *"Pane" e "Stage"*, componentistiche per noi fondamentali nell'ambito View. Disponendo difatti di molteplici pagine, accadeva di ripetersi nella creazione e caricamento di questi elementi.

Soluzione: Successivamente la mia procedura risolutiva è stata, seguendo il pattern **Strategy**, costruire una classe il cui compito era unicamente la creazione e o caricamento, di *Pane e Stage*. La classe in questione è *"Loader"*, costituita da due metodi statici: *loadPane* e *loadStage*, ognuno per l'oggetto da cui prendono il nome.

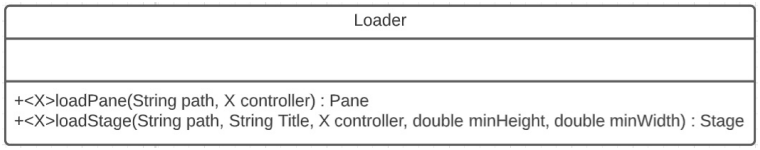

Figure 6: UML Loader

Problema : Grande lentezza per serializzare i nostri dati contenuti all'interno del nostro applicativo.

Soluzione : Implementare nel metodo di serializzazione dei dati il Multi-threading, ogni volta che l'operazione di scrittura viene chiamata, si apre un nuovo thread, il cui compito è quello di serializzare i dati in input nella locazione specificata.

2.3.3 Nicola Strada

Problema: Preoccuparsi di gestire un servizio di filtraggio prodotti all'interno del magazzino *(Stock)*, eseguito in base a *Categoria* e *Quantità del Prodotto*.

Soluzione: La prima implementazione da me realizzata consisteva in una funzione costituita da tre parametri: Due di questi che avrebbero definito il range per la quantità prodotto, mentre il terzo sarebbe servito a definire se la categoria del prodotto rientrava o meno nella collezione definita dal parametro. Questa soluzione si presentava però davvero poco efficace e per niente scalabile, per questo motivo, ho pensato di sostituire tutti gli argomenti della funzione con *BiPredicate*. Caratterizzato da due parametri, il primo di tipo intero, per il controllo sulla quantità, mentre il secondo destinato alla *Categoria*.

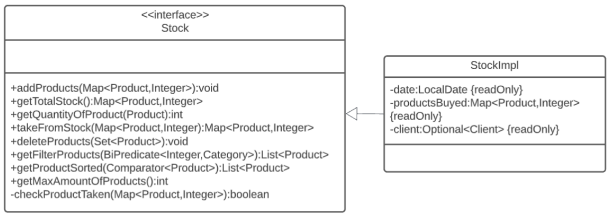

Figure 7: UML Stock

Problema: Poter ordinare i prodotti all'interno del magazzino *(Stock)*, con criterio *Crescente, Decrescente* o possibili altri stili di confronto e ordinamento.

Soluzione: Il primo approccio al problema è stato, in modo molto basilare, quello di creare due metodi; uno per l'ordinamento *Crescente*, e il secondo *Decrescente*. Così facendo limitavo però totalmente la possibilità di estendere lo stile di ordinamento, restringendo il campo a *Crescente e Decrescente*. Ragionando nuovamente sulla natura del problema, e sul funzionamento dell'ordinamento e *Sorting* in Java, ho notato l'opportunità di potersi servire dell'interfaccia *Comparator*. Di conseguenza ho costruito un singolo metodo, il quale input fosse un *Comparator*, in tal modo sono riuscito ad ottenere un unica funzione la cui scalabilità ed efficienza è molto alta.
Riferimento a **Figura 7**.

Problema: Un problema, derivante dalla Serializzazione, consisteva nell'istanziare un oggetto *ObjectMapper* ed impostarlo, per poi procedere infine al suo utilizzo durante ogni operazione di serializzazione.

Soluzione: La semplice soluzione l'ho trovata con l'impiego del pattern **Factory**: costruendo la classe *"ObjectMapperFactory"*; la quale presenta un singolo

metodo statico che restituisce un oggetto ObjectMapper, già preimpostato e pronto ad essere usato.

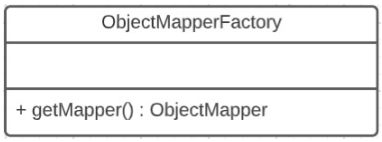

Figure 8: UML ObjectMapperFactory

2.3.4 Fabio Vincenzi

Problema: Riuscire ad estrapolare una strategia con la quale poter Serializzare e Deserializzare le classi utilizzate all'interno del nostro Software.

Soluzione: Mediante l'utilizzo del pattern **Strategy** ho creato una classe che si occupasse della Serializzazione di un singolo Model, ed un'altra, con il compito invece di Deserializzare l'oggetto in questione. Tale strategia è stata impiegata per quasi la totalità dei componenti del Model. (Osserviamo come esempio la Serializzazione di *Sale*).

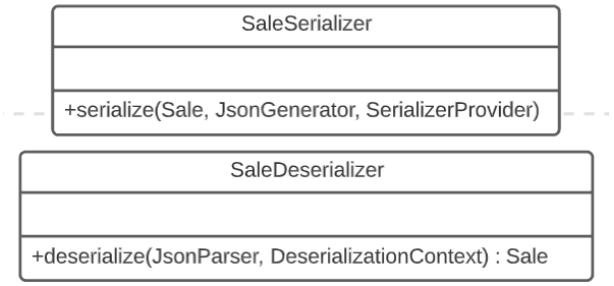

Figure 9: UML SaleSerializer

Problema: Durante il ciclo di sviluppo del progetto, ci si è posto il problema sulla introduzione della *Serializzazione Dati*. Dopo diversi tentativi si è giunti al punto in cui, in più di un Controller, si ricorreva alle medesime istruzioni.

Soluzione: Qui una volta riconosciuto il problema, ovvero quello di non ripetersi più volte in più classi, ci siamo consultati per decidere quale delle soluzioni adottare. Alla fine si è optato per il pattern **Strategy**, attraverso il quale ho poi incapsulato tutto ciò che riguardava la Serializzazione in una classe apposita, *"Serialization"*. Serialization si compone esclusivamente di due metodi statici:

Serialize e *Deserialize*. Il primo che si occupa della scrittura dati su file, mentre il secondo, complementarmente, della lettura. In questo modo, ovunque si voglia nel codice, si ricorrerà semplicemente all'utilizzo di tale classe.

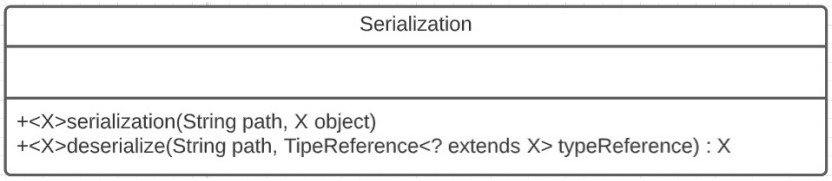

Figure 10: UML Serialization

Problema: Riuscire ad estrapolarsi una collezione di vendite, filtrate in base alla data di emissione.

Soluzione: La procedura risolutiva è stata di voler utilizzare una funzione, che prendesse come input un *Predicate* di tipo *LocalDate*, così sono riuscito ad ottenere una modalità di filtraggio molto scalabile.

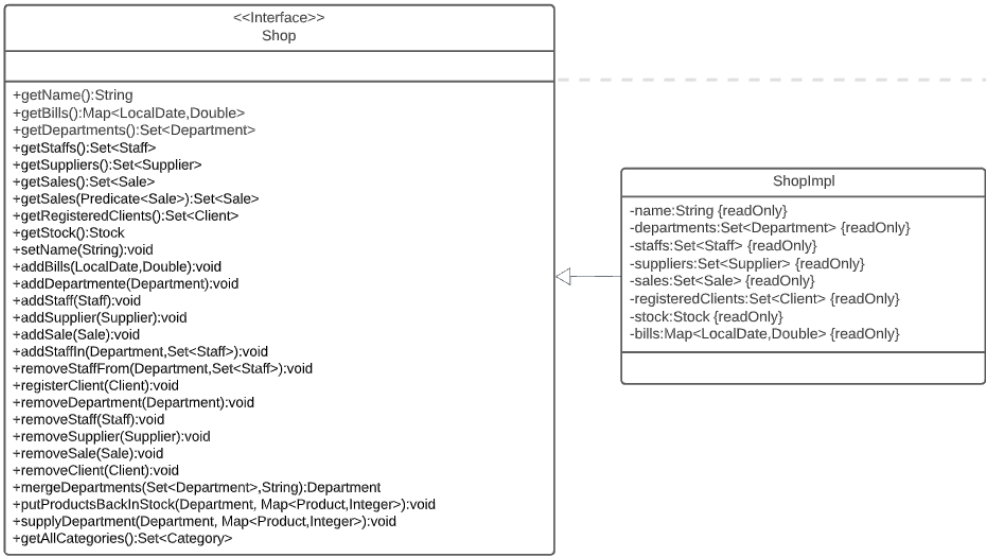

Figure 11: UML Shop

Problema : Dover salvare i file JSON, in una maniera tale che questi potessero sempre essere disponbili al nostro JAR, permettendo quindi di poter funzionare senza alcun errore, facendo in modo che il JAR possa essere eseguito indipendentemente dalla locazione in cui si trova.

Soluzione : Tramite il pattern Strategy ho creato una classe denominata **Save**, il cui compito è quello di :

- Creare la cartella **./unieuroop** all'interno della home directory dell'utilizzatore, se questa non esiste.
- Copiare tutti i file JSON contenuti nelle risorse del nostro applicativo all'interno cartella **./unieuroop**, se questi non sono già contenuti in tale cartella.

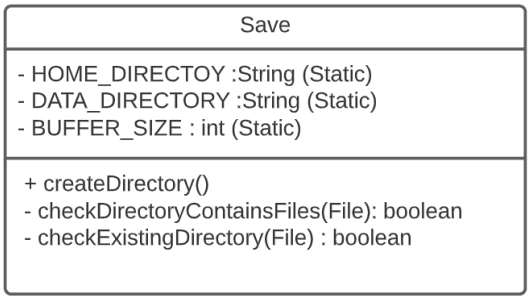

Figure 12: UML Save

3 Sviluppo Progettuale

3.1 Testing Automatizzato

Per quanto riguarda il *Testing*, parte a nostra visione molto importante, si è scelto di automatizzarlo testando ciascun metodo delle seguenti classi:

- **Analytic** (*unieuroop.test.analytic*): inserendo delle "vendite esempio" e visualizzando su queste tutte le potenziali statistiche da noi calcolabili.
- **Department** (*unieuroop.test.department*): verificando il corretto funzionamento di tutte le operazioni eseguibili su uno o più reparti, tra cui osserviamo, aggiunta, rimozione, unione di almeno due reparti...
- **Serialization** (*unieuroop.test.serialization*): utilizzando la classe in questione, da noi realizzata, e accertandosi che questa riesca nelle operazioni di scrittura e lettura da file.
- **Stock** (*unieuroop.test.stock*): verificando, una volta riempito di prodotti fittizi, il corretto funzionamento di tutte le sue operazioni eseguibili, tra cui aggiunta prodotti, rimozione prodotti, filtrare prodotti...
- **Shop** (*unieuroop.test.shop*): infine abbiamo esaminato e testato l'intero shop, creandone uno esempio che si avvicini al più a quello finale.

Tutti i test verificano ogni singolo metodo delle classi sopra descritte, in modo da confermare la loro corretta implementazione. Avvalendosi poi di asserzioni siamo riusciti a verificarne e controllarne anche la consistenza e correttezza dei dati restituiti.

3.2 Metodologia di Lavoro

Esaminiamo adesso in questa sezione, più nel dettaglio, il lavoro che ciascuno di noi ha svolto all'interno del progetto.

Lavoro Svolto da Samuele La mia parte del progetto parte dallo sviluppo della classe AbstractPerson, poi cambiata in BasePerson per avere una non inconsistenza e per utilizzare al meglio la composizione e non una classe astratta senza metodi astratti. Successivamente sono passato a sviluppare Cliente e Staff, quest'ultima sviluppata assieme al mio collega Fabio, due classi tra di loro molto simili, ma con differenze sostanziali : Il cliente sarà colui alla quale verranno assegnati i suoi vari acquisti, mentre Staff rappresenta colui che lavora all'interno del nostro negozio. Ad ogni Cliente e ad ogni Staff è assegnato un codice che viene calcolato all'interno del controller attraverso una tecnica che li rende unici. Di conseguenza non si potranno mai avere due codici uguali tra due Clienti o tra due Staff. Conseguentemente ho iniziato a sviluppare il controller di Staff e il controller dei Clienti, i cui compiti sono quelli di aggiungere/modificare/rimuovere Clienti o parte dello Staff. Poi ho iniziato a sviluppare alcune view ovvero le view che fanno riferimento ai Clienti e allo Staff.

Lavoro Svolto da Matteo Per quanto riguarda la mia parte io mi sono principalmente occupato di gestire tutto ciò che riguarda le Analitiche. Mi sono occupato di estrarre tutte le informazioni che verranno poi visualizzate all'interno dei nostri grafici, tramite l'utilizzo di stream, e lambde. Un'altra classe da me sviluppata è quella delle Vendite, ovvero la classe che permette di memorizzare tutte le informazioni dell'acquisto di svariati prodotti da parte di un possibile cliente, L'ultima classe da me sviluppata è quella dei Reparti, sviluppata con il mio collega Nicola Strada. Per poi sviluppare il controller delle Vendite, il controller delle Analitiche e il controller dei Reparti sempre in concomitanza con il mio collega Nicola. Infine sono passato a sviluppare le View, principalmente mi sono occupato di sviluppare le pagine delle Analitiche, la Pagine delle Sale e alcuni pezzi della pagina dei Reparti, in particolare l'aggiunta di nuovi reparti, la rimozione di alcuni reparti, il merge di più reparti e l'aggiunta/modifica dello staff nei reparti, questi ultimi sviluppati tramite la cooperazione con il mio collega Nicola.

Lavoro Svolto da Nicola Il mio compito era quello di sviluppare la gestione dei vari fornitori all'interno del nostro negozio, quindi anche l'acquisto di prodotti da essi, dopo aver sviluppato ciò sono passato a sviluppare la classe che gestire il Magazzino, quindi la prelevazione dei prodotti da questo luogo e la loro distribuzione nei vari reparti del negozio per poi concludere con lo sviluppo in parallelo al mio collega Matteo della classe dei Reparti. Dopo aver sviluppato tutto questo ho iniziato a scrivere il codice per i controller di Stock, il codice del controller dei Reparti e il codice del controller dei Supplier. Successivamente ho iniziato a scrivere il codice delle varie view. Tra cui la view dei vari fornitori tramite la quale è possibile acquistare da questi i prodotti, la view dello stock dalla quale è possibile controllare i vari prodotti nel magazzino, filtrarli secondo determinati parametri e successivamente la view dei reparti nella quale ho gestito assieme al mio collega Matteo l'aggiunta, la rimozione e il merge dei Reparti.

Lavoro Svolto da Fabio Sono partito dallo sviluppo della classe Shop, una delle classi fondamentali del nostro progetto nella quale sono contenuti i vari reparti, i vari clienti e tutti i lavoratori che compongono i vari reparti e non. Dopo aver sviluppato questa classe ho iniziato a sviluppare la classe Prodotto, la quale incapsula il concetto dell'entità venduta dal nostro negozio. Per poi terminare assieme al mio collega Samuele la costruzione della classe Staff. Una volta terminata la parte di model ho iniziato a sviluppare i controller che facevano riferimento allo Shop ed al LogIn. Infine mi sono occupato di studiare in che modo serializzare e deserializzare i nostri componenti.

Lavoro Svolto dal Team Come team invece abbiamo tutti svolto assieme l'intero sviluppo dei test, abbiamo tutti partecipato alla creazione del codice che ci ha permesso di controllare e verificare la correttezza dei nostri metodi. Inoltre all'interno dei vari controller specifici dei model e delle varie view si è

sempre cercato di darsi una mano a vicenda, quindi quasi tutti noi abbiamo messo mano su quasi tutti i componenti del nostro applicativo.

3.3 Note di Sviluppo

Osserviamo ora le particolarità di sviluppo impiegate da ognuno:

- **Note di Fabio:** Utilizzo di *JavaFX, java.util.Optional, Stream, Lambda Expressions*, libreria *Jackson* per serializzazione e libreria *iText7* per la generazione di fatture.
- **Note di Samuele:** Utilizzo di *JavaFX, Stream e Lambda Expressions.*
- **Note di Nicola:** Utilizzo di *JavaFX, Stream e Lambda Expressions.*
- **Note di Matteo**: Utilizzo di *JavaFX, java.util.Optional, Stream e Lambda Expressions.*

4 Commenti Finali

4.1 Autovalutazione e Lavori Futuri

4.1.1 Autovalutazione Ferri Samuele

Durante lo sviluppo di tutto il nostro progetto, il mio compito principale è stato quello di dover sviluppare la parte che facesse riferimento ai Clienti ed allo Staff e tutte le loro interazioni con il nostro applicativo. Sono molto fiero di ciò che sono riuscito a sviluppare, sopratutto per come sono riuscito a gestire la difficoltà nel non utilizzare più la classe astratta AbstractPerson ma la composizione.

4.1.2 Autovalutazione Iorio Matteo

Il mio compito principale è stato quello di dover sviluppare tutte le query per le analitiche. Sono veramente soddisfato di come sono riuscito a rappresentare esse nei vari grafici. Credo siano uno dei tanti punti di forza del nostro applicativo, il quale riesce a distinguersi dalla massa. All'interno della mia classe però uno o due metodi non sono stati implementati nelle view per mancanza di tempo, mi piacerebbe un giorno poterli implementare perchè comunque riuscirebbero a rendere le analitiche ancora più forti e belle da vedere. Credo che il nostro progetto sia veramente ben riuscito, con le conoscenze da noi attualmente acquisite.

4.1.3 Autovalutazione Strada Nicola

Nonostante io dovessi fare una delle parti a me più complicate, posso dire di essere completamente fiero di ciò che sono riuscito a sviluppare. Ho astratto nella miglior maniera, secondo me, il concetto di entità "Supplier". Vado anche molto orgoglioso della maniera con la quale lo "Stock" è stato da me sviluppato, apprezzo l'idea di poter filtrare i prodotti contenuti in esso per avere una visione più chiara dello che "Stock".

4.1.4 Autovalutazione Vincenzi Fabio

Sono abbastanza soddisfatto, di come sono riuscito a gestire tutta l'organizzazione dei file **JSON**, avrei preferito utilizzare un vero e proprio DB, essendo questo molto più comodo e facile da utilizzare. Uno dei tanti problemi, è stato quello di riuscire a trovare una libreria adeguata alle nostre necessità, poiché molte di queste riuscivano a serializzare solo i dati base e strutture dati semplici del linguaggio Java (Stringhe, Interi, Liste di interi, ecc ecc). Non andrò a negare le difficoltà nell'utilizzare la **libreria Jackson** per serializzare e deserializzare le nostre classi e le nostre collezioni contenute all'interno delle classi stesse. Ritengo comunque di aver svolto un ottimo lavoro sotto questo aspetto.

4.1.5 Autovalutazione Gruppo

Come gruppo sosteniamo di aver fatto tutti un ottimo lavoro, abbiamo collaborato tanto, discusso tanto e tutti siamo riusciti a portare a termine i nostri compiti, ci siamo aiutati molto e questo ci ha permesso di evitare di creare lacune tra di noi, il lavoro di squadra che c'è stato tra di noi è risultato una strategia vincente per portare a termine l'intero nostro progetto, con le funzionalità che ci siamo promessi. Come gruppo ci sarebbe veramente piaciuto introdurre tutte le funzionalità della classe "Analityc", ma per cause di tempo non siamo riusciti.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Iorio Matteo

Qui di seguito vorrei condividere alcuni commenti riguardanti il corso da me affrontato. Mi sarebbe piaciuto veramente tanto sfruttare le ore passate nei laboratori per approfondire al massimo i concetti affrontati nelle lezioni dal prof Viroli. Purtroppo la metà del tempo nei laboratori veniva impiegata per ascoltare le spiegazioni su altri argomenti. Mi piacerebbe poi che gli esercizi che offrite nei laboratori non siano già completamente fatti, sarebbe bello se ogni tanto deste un progetto vuoto e noi studenti il compito di svolgere l'esercizio da zero. Un altro piccolo appunto che vorrei fare è quello di soffermarsi molto di più su gli aspetti come : Stream, Lambde e sopratutto sui Pattern. Ho notato che la metodologia con la quale sono stati affrontati in classe questi argomenti è stata troppo "frettolosa", solamente perché eravamo giunti alla fine del corso. Dopo tutto questo comunque ritengo che il progetto sia vitale, molto utile e sopratutto estremamente formativo. Quindi credo che la scelta di fare questo progetto sia azzeccata.

5 Appendice A

5.1 Guida Utente

Login Al primo avvio del nostro programma la pagina che viene visualizzata è quella di LogIn. Per poter accedere nella nostra applicazione è necessario dover introdurre delle credenziali corrette (credenziali di un qualsiasi Staff). Per aiutare i nostri utilizzatori potete utilizzare come email : "prova@gmail.com" e come password "1234". Da notare che qualsiasi tentativo di controllare all'interno del file "Staff.json" e utilizzare una delle password scritte lì è totalmente inutile, siccome ogni password là scritta è criptata tramite un hashcode.

Dashboard Una volta entrati all'interno della nostra applicazione, la prima pagina che viene visualizzata è la "Dashboard", una pagina che offre un recap sul nostro negozio, seguendo l'ordine con la quale vengono visualizzati troviamo:

- **Sezione Staff:** possiamo vedere il numero totale di dipendenti che lavorano all'interno del nostro negozio
- **Sezione Supplier:** vi è possibile controllare il numero di "Supplier" disponibili, dai quali è possibile comprare prodotti per il nostro negozio
- **Sezione Department:** vi è il numero di reparti totali
- **Sezione Valore Magazzino:** possiamo vedere il valore totale del nostro magazzino (il valore di tutti i prodotti sommati tra di loro)
- **Sezione Guadagni:** possiamo vedere il totale guadagnato attraverso tutte le vendite eseguite dal nostro negozio
- **Sezione Soldi Spesi:** tutti i soldi spesi per comprare i prodotti dai vari supplier
- **Sezione Vendite:** è possibile visualizzare tutte le vendite eseguite dal negozio, cliccando poi sopra una di essa verranno visualizzati tutti i prodotti venduti e la loro relativa quantità venduta.

Si osservi attentamente che il valore totale dei soldi guadagnati è un indice molto importante rispetto ai soldi totali spesi, in quanto ci permette di capire se il negozio è in guadagno o in perdita. A tal proposito ci siamo lo scrupolo di rendere ciò ancora più chiaro attraverso la rappresentazione del guadagno con due semplici colori :

- Se Guadagno $>$ Soldi totali spesi la label sarà verde
- Se Guadagno $\leq$ Soldi totali spesi la label sarà rossa

Questo fa si che si abbia sempre un'idea di come sta procedendo il nostro negozio.

Stock Successivamente se esploriamo il menù possiamo imbatterci nella pagina del nostro magazzino, in cui è possibile visualizzare tutti i prodotti. In aggiunta è possibile effettuare operazioni di filtraggio del magazzino, è possibile selezionare determinati parametri per filtrare i vari prodotti del nostro magazzino. Un'altra operazione molto importante che si può effettuare è quella del rifornimento. In tale pagina si potranno scegliere uno o più fornitori dai quali sarà possibile acquistare i vari prodotti. **Piccola nota sui Suppliers : l'unica maniera con la quale possono esserne aggiunti di nuovi è andandoli ad inserire manualmente all'interno dei file "Suppliers.json".**

Sales Navigando attraverso il menu, passiamo nella sezione delle vendite. Viene visualizzata una pagina nella quale è necessario scegliere il/i reparti da cui prelevare i vari prodotti con le relative quantità, una volta scelti i prodotti si può procedere cliccando il tasto "Sell", da cui vi si aprirà una pagina, nella quale sarà possibile scegliere il cliente a cui associare la vendita.

- sarà poi inoltre possibile scegliere dove salvare la relativa fattura dell'ordine (questa cosa la si può evitare semplicemente chiudendo la finestra di scelta della cartella in cui salvare la fattura, così facendo la sale verrà comunque chiusa senza rilasciare alcuna fattura).

Notare bene che se non vi si sceglie alcun cliente, la scelta per dove salvare la fattura non verrà proprio visualizzata.

Client Scendendo ancora nel nostro menù troviamo la pagina che fa riferimento ai vari clienti del nostro negozio. In questa pagina sarà possibile aggiungere nuovi clienti, modificare i parametri :

- Nome
- Cognome
- Data di nascita

ed infine eliminare un cliente selezionato sempre dalla lista accanto.

Staff Ancora più in basso possiamo trovare la pagina utilizzata per gestire i vari dipendenti del nostro negozio. E' possibile aggiungere nuovi dipendenti, modificare i dati ;

- Nome
- Cognome
- Data di nascita
- Orario di lavoro
- Password
- Email

del dipendente selezionato oppure eliminarlo completamente.

Department Successivamente troviamo la pagina che fa riferimento ai reparti del negozio. Vi saranno visualizzati diversi componenti, partendo da sinistra troveremo la lista di tutti i reparti presenti all'interno del negozio, cliccando su uno di essi sarà possibile visualizzare tutti i suoi dipendenti e tutti i suoi prodotti.

- **Modifica dello Staff:** Attraverso una nuova pagina sarà possibile aggiungere o rimuovere dipendenti all'interno del reparto selezionato, semplicemente cliccando su uno di essi e poi cliccando su uno dei due bottoni disponibili "Add" o "Remove", in base al tipo di dipendente scelto. A sinistra i dipendenti che non sono assegnati ad alcun reparto e a destra i dipendenti che fanno riferimento al reparto.
- **Modifica dei Prodotti:** A sinistra si troveranno i prodotti del nostro magazzino, dal quale potremmo attingere per poterli aggiungere al reparto selezionato, mentre a destra ci saranno i prodotti del reparto, i quali potranno essere invece rimossi dal reparto e riposti all'interno del magazzino.

Passando invece alle opzioni che fanno riferimento ai reparti, troviamo :

- **"Add Departments":** apre una nuova finestra con diversi elementi, in alto a sinistra troviamo un'area di testo in cui poter inserire il nome del nuovo reparto che si vuole inserire, sotto di esso troviamo invece tutti i dipendenti che non sono assegnati ai vari reparti e quindi possibili nuovi dipendenti del nuovo reparto che si sta creando, a destra di essi troviamo tutti i prodotti del magazzino, da cui è possibile prelevare i prodotti con le quantità selezionate. Una volta selezionati tutti i dipendenti che si desiderano e tutti i prodotti che si vogliono avere basta cliccare "Add" per annullare tutto invece bisogna cliccare "Exit".
- **"Merge Departments":** tale bottone apre una nuova finestra in cui è possibile invece scegliere due o più reparti e unirli fra loro, andando ad unire tutti i prodotti, tutti i dipendenti, l'unica cosa che bisognerà impostare sarà il nuovo nuove del reparto che si andrà a creare e successivamente cliccare il bottone "Merge"
- **"Delete Department":** il quale apre una pagina in cui saranno visualizzati tutti i reparti presenti nel negozio, cliccando su uno di essi verrà visualizzato un messaggio di avviso, se tale avviso verrà chiuso cliccando sul bottone "X" il reparto non verrà eliminato ma se invece si clicca il pulsante "OK" il reparto selezionato verrà eliminato.

Balance Scendendo ancora, troviamo la pagina "Balance" nella quale è possibile visualizzare due semplici grafici a torta nel quale sono rappresentati, partendo da sinistra, il grafico che rappresenta i soldi spesi nell'ultimo anno per acquistare i prodotti dai fornitori, mentre il grafico affianco rappresenta tutti i soldi guadagnati nell'ultimo anno distribuiti nei vari mesi. Il grafico sottostante rappresenta l'andamento dei soldi spesi con i soldi guadagnati nei diversi anni di attività del negozio.

Categories Analutics La pagina successiva, “Categories Analytics” , ci permette di visualizzare le statistiche sulle varie categorie dei prodotti. Partendo dall’alto vediamo ogni singola categoria di prodotti venduti con il relativo numero di prodotti differenti venduti, se per esempio vendessi 2 iphone 13 pro (Categoria Smartphone), 1 Samsung 22 Ultra (Categoria Smartphone) e 5 Xiaomi (Categoria Smartphone) vedremo nel grafico la categoria Smartphone con valore 3, poiché sono 3 i prodotti differenti di quella categoria venduti. Il grafico sotto invece ci permette di avere un’analisi molto più approfondita delle analitiche delle categorie. Utilizzando la combobox è possibile selezionare una o più categorie, le quali ci permetteranno di visualizzare tutti i prodotti venduti di tale/tali categorie selezionate. Affianco al grafico ci sarà una ListView che verrà popolata mano mano che si scelgono le varie categorie. **Tale Lista ci permette di avere una legenda sui prodotti che si stanno visualizzando**, siccome nel grafico non verrà riportato il nome del prodotto ma il suo relativo codice. Una cosa che può tornare molto utile è : siccome tale grafico può riempirsi subito di tanti valori e perdere quindi il prodotto specifico che si voleva visualizzare, **sarà possibile cliccare sul corrispettivo prodotto nella lista accanto e tale selezione permetterà di evidenziare sul grafico il prodotto selezionato**. Questo iter può essere ripetuto anche per più di un prodotto. Ogni volta che si seleziona una nuova categoria, questa verrà aggiunta alla lista vicino al bottone “Clear All” in modo da dare un’idea su quali siano tutte le categorie selezionate. Cliccando poi sul bottone “Clear All” tutte le categorie selezionate verranno eliminate, resettando di fatto il grafico e le varie liste.

Date Aanalytics L’ultimo elemento del nostro menù è l’analitica in base alle date. Il grafico più in alto, visualizza tutti i ricavi effettuati in quell’anno. Nel grafico più in basso invece verranno visualizzate tutte le vendite che saranno comprese nei due intervalli di tempo indicati dai DatePicker. Come nelle analitiche delle categorie, ritroviamo sempre la lista accanto che ci permette di visualizzare tutte le vendite visualizzate nel grafico. Ritroviamo anche lo stesso “Tool” delle analitiche per le categorie, quindi cliccando su un valore della lista, verrà evidenziato anche all’interno del secondo grafico.

6 Appendice B

6.1 Esercitazioni di Laboratorio

B.0.1 Ferri Samuele

B.0.3 Strada Nicola

B.0.2 Iorio Matteo

- **Lab 07:** https://virtuale.unibo.it/mod/forum/discuss.php?d=88829
- **Lab 08:** https://virtuale.unibo.it/mod/forum/discuss.php?d=89272
- **Lab 09:** https://virtuale.unibo.it/mod/forum/discuss.php?d=90125
- **Lab 10:** https://virtuale.unibo.it/mod/forum/discuss.php?d=91128

B.0.4 Vincenzi Fabio