Entrata merci

Enrico Collina

01/10/2022

Analisi

1.1 Requisiti

La richiesta è quella di poter informatizzare il flusso di accettazione di un'azienda, creando un'applicazione che permetta di registrare gli arrivi della sua azienda, creando uno storico di questi ordini e permettendo di visualizzare la giacenza della merce introdotta.

Si deve poter quindi consultare in ogni momento l'anagrafica dei fornitori per permettere di avere dati sempre costantemente aggiornati, permettendo anche la modifica di questi fornitori.

Si deve anche poter consultare in ogni momento l'anagrafica degli articoli per permettere di avere dati aggiornati e modificabili.

Non sarà possibile attuare l'eliminazione di un fornitore e di un articolo, ma bensì si potrà renderlo obsoleto, questo ai fini di tenere traccia di uno storico, altrimenti se eliminassi un articolo presente in giacenza, lo storico non sarebbe più consultabile.

Si richiede inoltre di poter consultare la giacenza puntuale di ogni articolo con la sua quantità, giacenza che deve anche poter essere filtrabile. In ogni momento quindi l'utente deve avere la possibilità di sapere dove potrà trovare la merce.

La giacenza viene gestita tramite l'utilizzo di aree e locazioni, in cui una locazione è uno spazio fisico all'interno di un'area, in cui viene stoccata la merce. La struttura della fisica del magazzino non cambierà, quindi verrà caricata in automatico ed essa non sarà modificabile direttamente dall'utente.

In ogni momento si deve poter tenere traccia di tutti gli ordini effettuati, con il fornitore che ha portato la merce, questo per permettere all'utente di effettuare verifiche su ciò che è entrato in magazzino e in che istante è stato registrato l'arrivo.

Non è richiesta la gestione delle vendite, infatti per quello sarà compito di un'integrazione terza chiamare le rettifiche di quantità in giacenza, per poter gestire correttamente tutto il flusso operativo aziendale.

La gestione del magazzino deve partire da un menù navigabile, diviso per sezioni, in cui in ogni momento l'utente possa fare tutte le azioni previste dal software, non vi è quindi un utilizzo step-by-step del software, ma bensì in ogni momento devo poter ad esempio inserire un articolo, finito ciò registrare un arrivo, oppure fare queste operazioni ma in ordine inverso e questo deve portare sempre allo stesso risultato.

1.2 Analisi del dominio

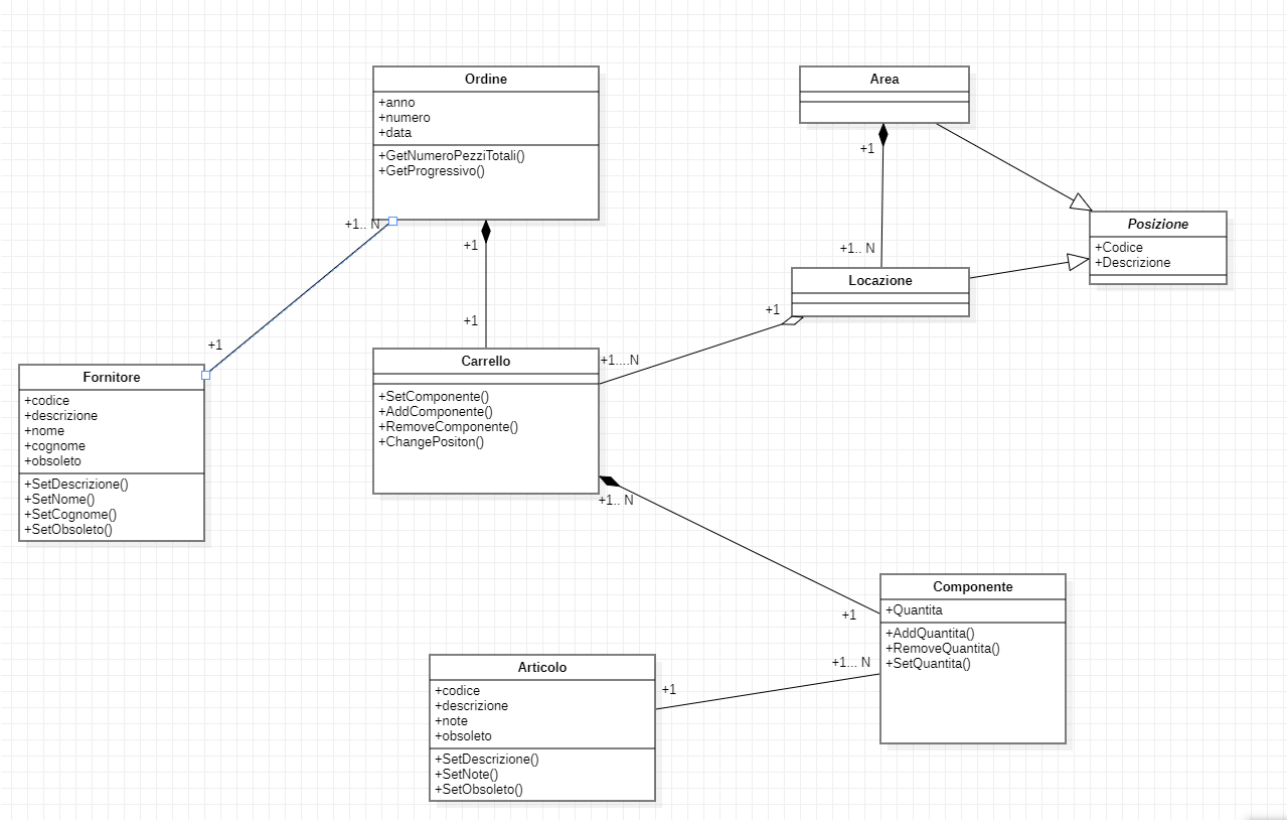
Il magazzino dovrà essere gestito tutto tramite il concetto chiave di carrello, che rappresenta per l'azienda il centro stella per la gestione degli stoccaggi. Infatti, il carrello non è altro che un contenitore in sono contenuti tutti gli articoli scaricati dal camion del fornitore.

Questo carrello sarà quindi sempre riconducibile al fornitore, questo per permettere a fronte di non conformità dell'articolo ricevuto, di attuare le misure necessarie.

Un carrello si compone di una lista di articoli, in una certa quantità, e durante l'inserimento dovranno essere presenti i soli articoli marcati come non obsoleti.

Un carrello proviene da un fornitore, anche in questo caso il fornitore presente in elenco dovrà essere un fornitore non obsoleto.

Di seguito viene quindi riportato il modello logico delle entità individuate in analisi, svolto per modellare al meglio le esigenze e darne una rappresentazione fedele della realtà su cui andrà a lavorare il software.



Design

2.1 Architettura

Per la realizzazione del software è stato deciso di implementare il pattern architetturale MVC. Questo permette una buona separazione dei contesti e rendere indipendenti le varie componenti. Infatti, difficilmente il modello operativo aziendale si modificherà nel corso, mentre le maschere potranno essere costantemente aggiornate e implementate nuove funzionalità. Questo sempre senza intaccare il modello.

Saranno previste quindi delle view che sono le maschere utilizzate dall'operatore, tramite le quali protrà utilizzare effettivamente e interagire con il software.

Partendo da un menu principale l'utente potrà navigare tra le varie funzioni e inserire/aggiornare/visualizzare i dati del magazzino, ognuno tramite la sua view separata.

Sono state individuate quindi 3 sezioni e il menù è stato diviso di conseguenza:

- Sezione Anagrafica
- Sezione Ordini
- Sezione Giacenza

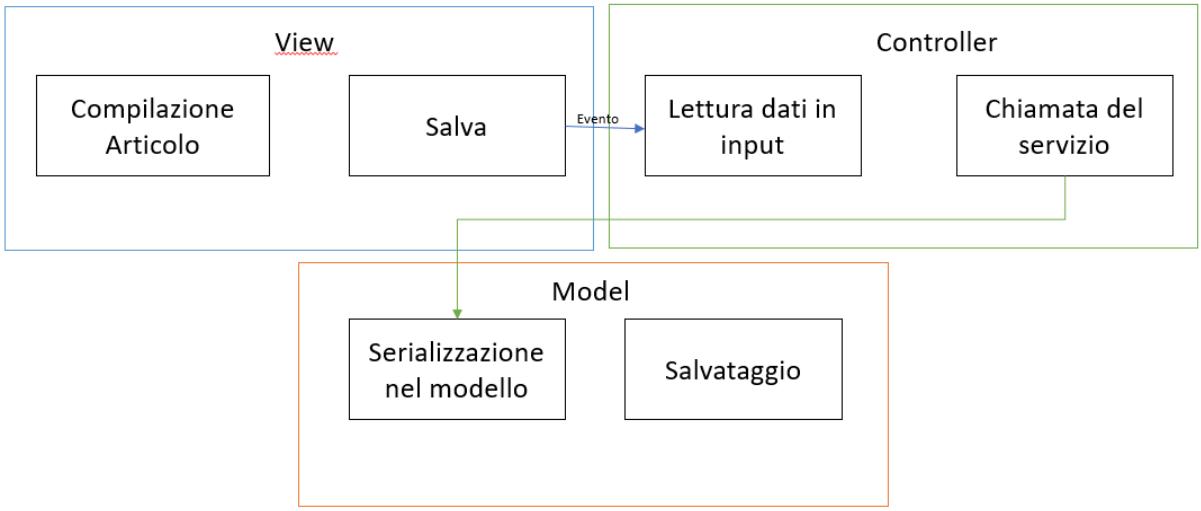
La sezione Anagrafica si occuperà della gestione degli articoli e fornitori all'interno del nostro magazzino, in cui potrà essere gestito l'inserimento, visualizzazione e modifica.

La sezione Ordini si occupa di gestire l'inserimento di un Ordine e del suo carrello, stoccandolo in una posizione del magazzino. Lo storico ordini permetterà di avere un quadro generale di tutte le accettazioni ad oggi fatte.

Sezione Giacenza permetterà di consultare la giacenza puntuale del magazzino.

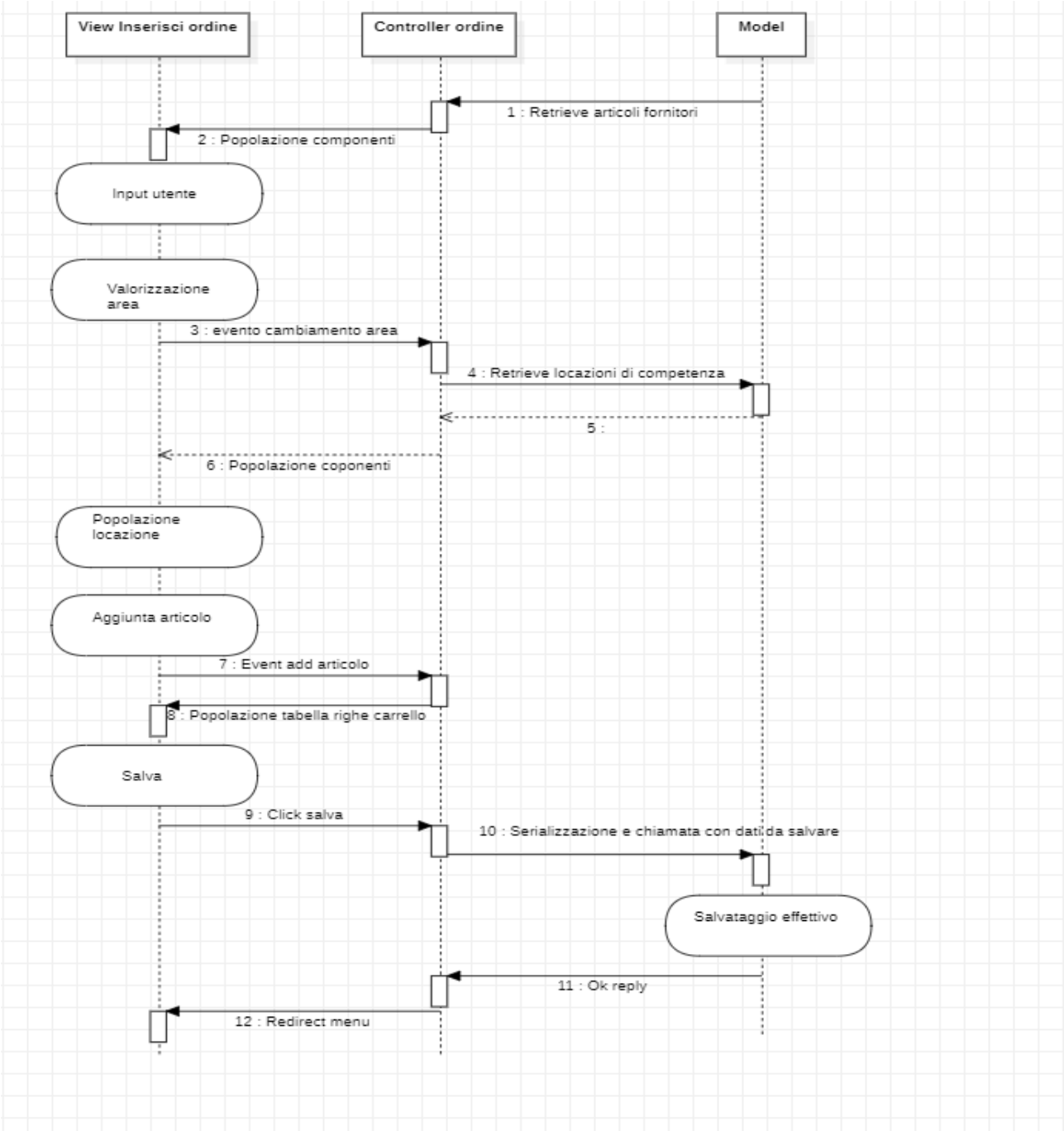
Ogni maschera avrà un suo controller dedicato che si occuperà della gestione di tutti gli eventi, inoltre si occuperà del dialogo con il nostro modello, tramite l'implementazione di "Service", che permettono la lettura dei dati presenti a magazzino.

Di seguito un esempio esplicativo di un'interazione.



Come da immagine si può notare come i 3 blocchi siano indipendenti tra di loro e togliendo e modificando una parte di questi 3, comunque i blocchi rimarranno sempre separati e indipendenti tra di loro.

Di seguito viene descritta l'interazione principale, quella di inserimento di un ordine. Si può notare il dialogo tra le varie componenti, descritte tramite un diagramma di flusso, con le varie timeline e le varie popolazioni dei contenuti.

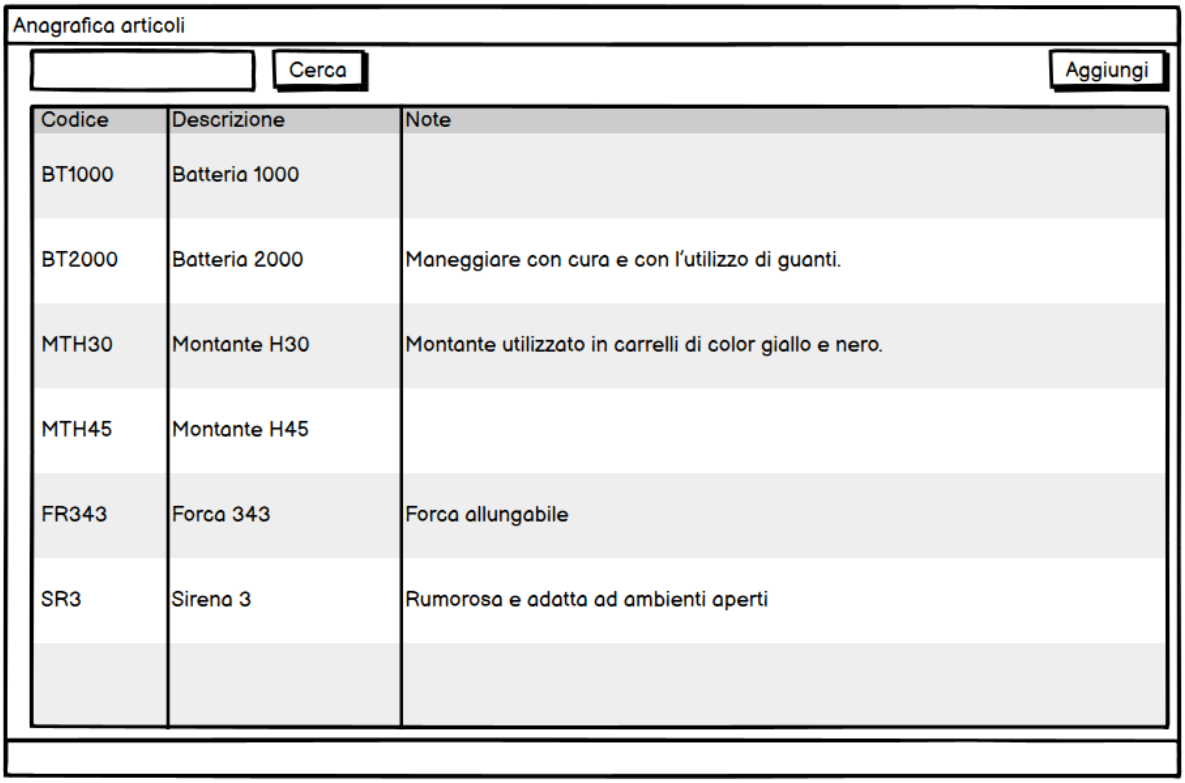


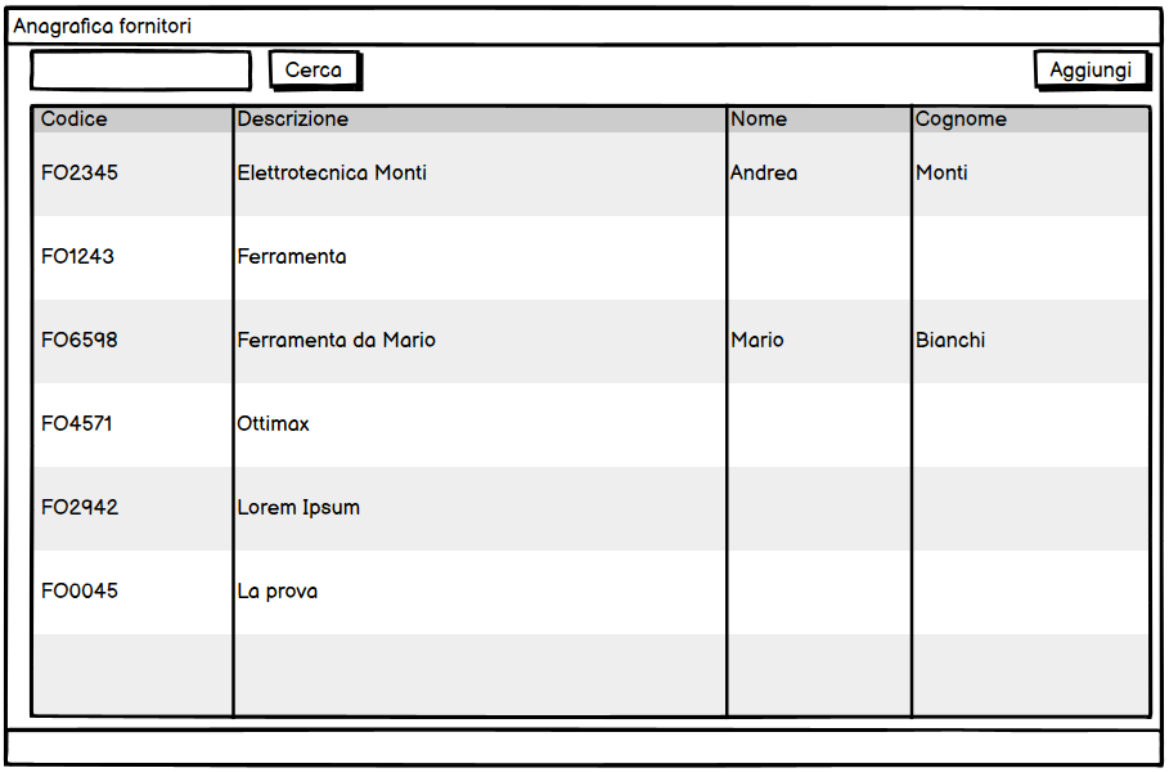
Il controller viene gestito tutto ad eventi e ogni evento produce un output, che può essere chiamare il modello oppure restituire un dato alla vista.

2.2 Mockup

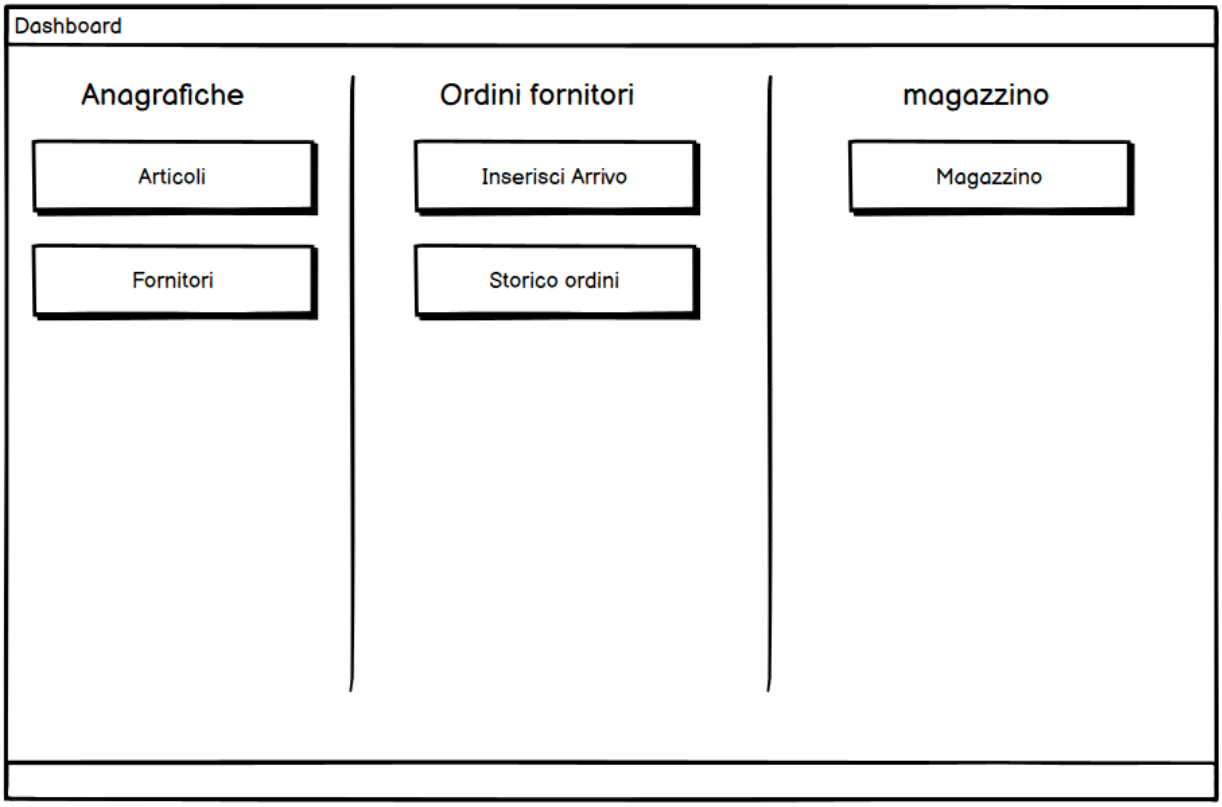
Di seguito vengono riportate anche dei mockup relativi alle pagine utente con le quali interagirà. Queste aiuteranno a capire l'operatività e l'utilizzo del software .

2.2.1 Anagrafica articoli e fornitori

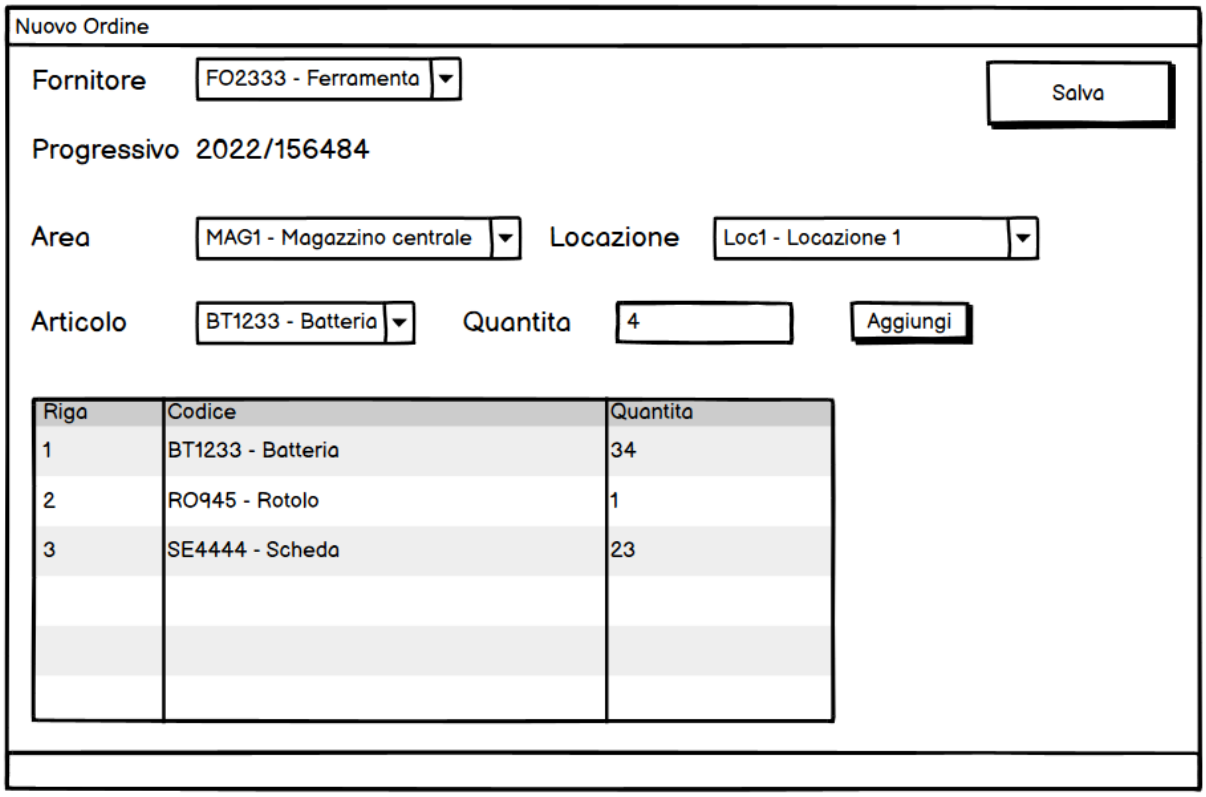




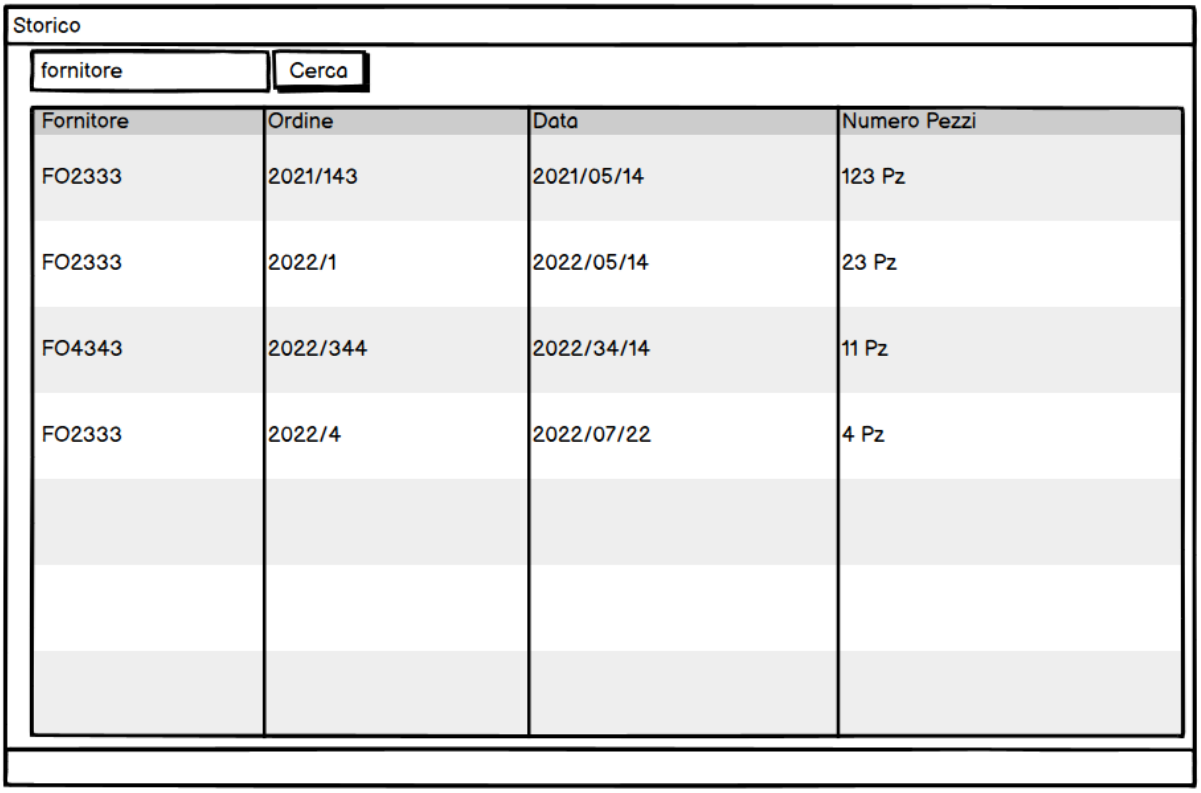
2.2.2 Menu



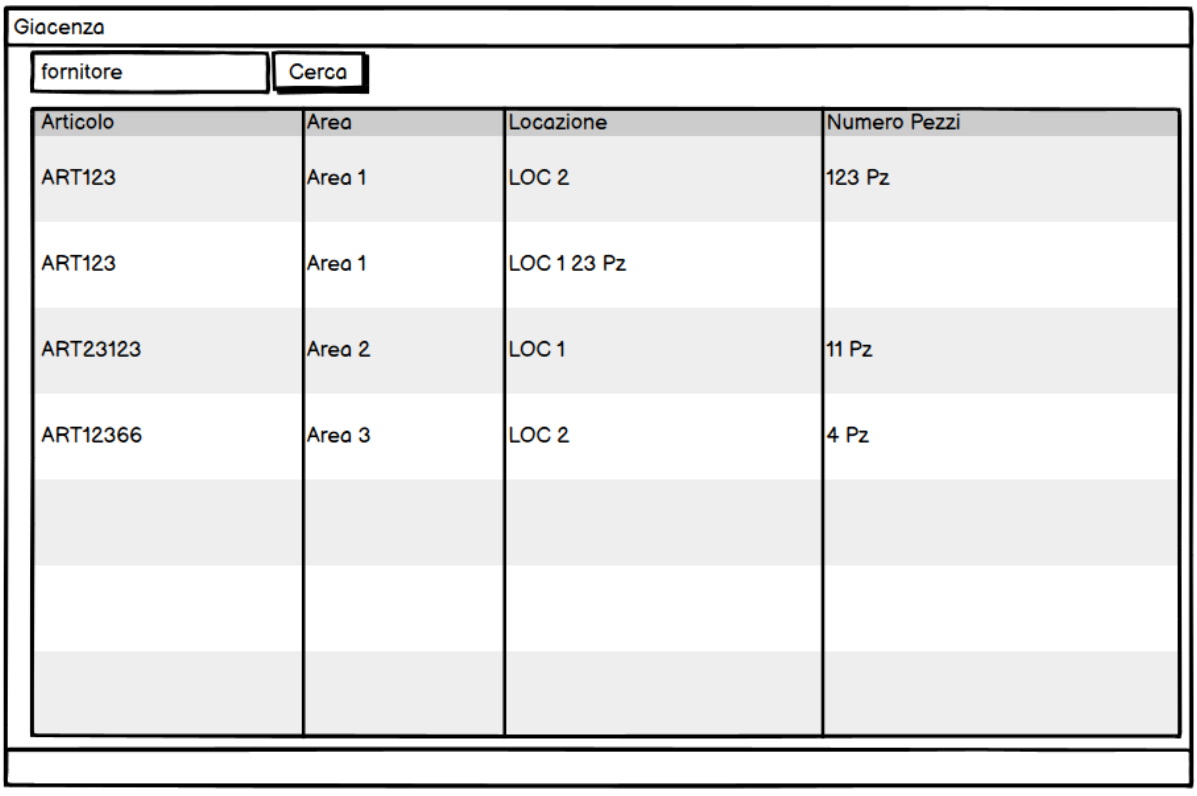
2.2.3 Inserimento arrivo



2.2.4 Storico arrivi



2.2.5 Giacenza



Sviluppo

3.1 Testing automatizzato

La parte fondamentale di questo software sono tutte le varie interrogazioni a modello, in particolare tutta la parte di “Service” che è la parte di dialogo tra Modello e Controller.

Tutti i servizi si occupano di fornire informazioni puntuali, corrette e filtrate in ogni istante. Quindi si è deciso di creare.

I test riportati sono completamente automatici ed eseguibili in ogni istante, per verificare ad ogni sviluppo di aver mantenuto la retrocompatibilità.

Per i test è stata utilizzata la libreria JUnit.

I test sono stati riportati sono:

- Test di lettura degli articoli, con test del servizio per la parte di filtro e di retrieve dei soli articoli non obsoleti
- Test su locazione di competenza di un’area, questo utile per tutte le funzioni di giacenza puntuale per la struttura del magazzino.
- Test sul corretto caricamento dei fornitori e della corretta lettura e filtro per codice, utile ogni volta che si deve andare in modifica dell’anagrafica e per consultazioni anagrafiche.

3.2 Metodologia di lavoro

Come primo approccio ho cercato di analizzare il progetto ed eviscerarlo in tutte le sue componenti. Ho cercato già di ragionare fin in analisi con un approccio MVC, pattern a mio parere molto comodo e che mantiene una separazione di interessi tale da rendere di facile comprensione e lettura ogni componente e soprattutto rende il progetto modulare. Infatti, in un primo momento avevo pensato di approcciarmi con JSwing, avendo creato già il modello, però per difficolta ho poi optato per l’interfaccia grafica di utilizzare JavaFx.

Essendo solo ed avendo molto da sviluppato ho cercato di sviluppare per funzionalità. Anche perché in alcuni momenti mi sono accordo di dover aggiustare il modello per renderlo più integrabile tra le varie maschere.

Sono partito dalle maschere di anagrafica per poter popolare il mio “DB”, seguendo poi dalla maschera centrale che è quella dell’inserimento dell’ordine e infine le maschere di consultazione. Finendo con la Giacenza che era l’unione di praticamente tutto il modello e la parte di service è stata a mio parere la più complessa.

3.3 Note di sviluppo

Per l’interfaccia grafica ho utilizzato JavaFx, che mi ha permesso a fronte di una view (.fxml) di creare un controller di facile lettura e di gestione chiara degli eventi. Ho aggiunto anche eventi alle tabelle per poter entrare in modifica delle righe presenti in tabella. C’è un event listener che si mette in ascolto del doppioclick ed esegue una funzione.

Per tutta la parte di servizio ho utilizzato lambda expression, stream soprattutto per filtrare, molto comodi soprattutto per i filtri poter andare a filtrare SOLO i filtri valorizzati, Aggiungendo alla stream già presente il filtro valorizzato di mio interesse.

Per la giacenza ho provato ad utilizzare il costrutto collectors groupingby che permette di creare un MAP<Key,Value> raggruppando per la chiave indicato. Questo mi era molto utile nella giacenza poiché la mia idea era appunto di raggruppare per articolo e locazione, poiché all'operatore interessa effettivamente quanti pezzi troverà in quella locazione di quel dato articolo. Però mi è risultata di difficile comprensione e soprattutto poco efficiente. Perché mi sono ritrovato una MAP molto complessa annidando i vari group by.

Per il salvataggio su file ho deciso di utilizzare la libreria gson, una libreria molto comoda che permette di parsare i miei oggetti da:

- JSON → Model
- Model → JSON

E la lettura e scrittura a quel punto è stata molto semplice, infatti leggevo il file e avevo già un ArrayList<Model> pronto per essere letto, mostrato oppure modificato.

Ho deciso di approcciare letture e scritture ad ogni azione, non mantenendo tutto in memoria un'istanza del mio DB, questo per scalabilità poiché potrei avere N client che lavorano assieme sul mio software che avrebbero dati costantemente aggiornati e corretti. (Ovviamente non è garantita la gestione della concorrenza).

Inoltre, ho utilizzato Maven per la gestione di tutte le dipendenze e come build tool.

Link alle librerie e tecnologie utilizzate:

https://openjfx.io/

https://github.com/google/gson

https://maven.apache.org/

Commenti finali

4.1 Autovalutazione e lavori futuri

In generale il software copre un caso d'uso che permetteva un buon approccio all'utilizzo degli oggetti e di tutte le varie tecnologie.

Permette anche sicuramente estensioni future, infatti io ho dato per assunto che la struttura del magazzino non variasse però sicuramente si potrebbe implementare una parte di gestione struttura di magazzino.

Altra cosa che renderebbe scalabile il prodotto potrebbe essere la gestione degli operatori e poter configurare tutte le maschere su base operatore, al fine di mostrare solo le funzionalità abilitate a quel dato operatore, per una separazione di compiti e di interessi.

La parte di vendita potrebbe essere anche quella un modulo futuro, con la gestione di un algoritmo di prelievo, magari configurabile sulla base dei più noti:

- FEFO – (First Expired First Out) Per clienti con merce deperibile
- FIFO – (First In First Out)
- LIFO – (Last In First Out)

Questo sarebbe un modulo molto grosso e che meriterebbe quasi un progetto separato a mio parere.

Nel complesso penso però che inserendo tutte le validazioni in modifica ed inserimento, tutti i filtri in maschera e i filtri in fase di inserimento, infatti vengono proposti in inserimento di ordine solo articolo non obsoleti, ma nelle maschere di consultazione tutti, il progetto sia abbastanza completo in tutte le sue parti.

4.2 Difficolta incontrate

Inizialmente ho fatto molta fatica perdendo molte ore per cercare di configurare JavaFX, probabilmente non aveva importato correttamente delle dipendenze, però anche seguendo numerose guide non riuscivo a fare partire il progetto, ogni volta mancavano delle componenti per fare partire le maschere.

Risolto il problema non ho avuto particolari difficoltà. Ho avuto difficolta a creare l'eseguibile JAR, sempre perché sembra che non abbia importato le librerie JavaFX nell'eseguibile e che mancassero componenti durante l'avvio.

Sono riuscito a farlo partire creando un .bat che esegue il jar con le varie dipendenze da JavaFx, e ho dovuto esportare anche le librerie, da cui legge il .bat per funzionare.

4.3 Note per l'esecuzione

Nel progetto ho incluso un folder chiamato "Eseguibile" contenente tutti i file necessari. Il progetto siccome legge e scrive in file .json cerca nel folder stored i file. Basterà lanciare il bat direttamente da quel folder e tutto partirà.

I file possono essere anche vuoti, al di fuori delle aree e locazioni, che ho dato per assunto essere immutabili e quindi già caricate. Per il resto si può partire completamete senza ordini, articoli e fornitori.