ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

SVILUPPO DI UN ALGORITMO PER LA PROPOSTA DI STOCCAGGIO

ELABORATO IN

ALGORITMI E STRUTTURE DATI

RELATORE:

MANIEZZO VITTORIO

PRESENTATA DA:

NEGRI ANDREA

CORRELATORE:

GAMBETTI CLAUDIO

ANNO ACCADEMICO: 2019-2020

Indice

1. Introduzione

L'obiettivo della tesi è quello di realizzare una nuova versione di un algoritmo per la proposta di stoccaggio, partendo dall'analisi del problema e del suo contesto, per poi passare alla progettazione e all'implementazione.

Lo stoccaggio è una delle varie fasi del ciclo di vita della merce in un magazzino e il suo scopo è quello di assegnare al carico appena arrivato una posizione in magazzino. L'algoritmo sviluppato serve quindi a proporre una posizione di stoccaggio durante l'omonima fase, che rispetti determinati vincoli e/o preferenze.

La scelta di sviluppare questo tipo di algoritmo deriva dall'aver svolto il tirocinio curriculare presso la Onit, un'azienda informatica che realizza soluzioni IT per diversi ambiti, tra cui quello industriale, di cui fa parte anche la gestione di un magazzino. Quando mi sono state fatte varie proposte di tesi, ho scelto quella che più mi affascinava cioè il rinnovamento dell'algoritmo per la proposta di stoccaggio. Durante lo sviluppo sono stato affiancato e guidato da Simone Castorri e Lorenzo Vernocchi, 2 team manager con i quali ho affrontato le fasi di analisi dei requisiti e stesura di un relativo documento, progettazione dell'algoritmo e implementazione.

La struttura della tesi è la seguente:

- **Sommario:** capitolo in cui viene descritto meglio il problema che si vuole affrontare, ed il contesto in cui si inserisce (le logiche di magazzino ma anche il software WMS);
- **Analisi:** questa parte della tesi è formata da tre capitoli e serve ad analizzare l'algoritmo attualmente implementato (come funzione e cosa fa), e il nuovo algoritmo (quali sono le nuove funzionalità richieste);
- **Progettazione:** capitolo in cui si definisce come si vogliono affrontare le funzionalità emersi nell'analisi;
- **Implementazione:** capitolo in cui vengono mostrate le scelte implementative effettuate per realizzare il nuovo algoritmo; viene quindi mostrato e spiegato il codice scritto;
- **Sviluppi futuri:** capitolo in cui si elencano possibili sviluppi futuri dell'algoritmo;
- **Conclusioni**

2. Sommario

2.1. Il magazzino e la sua struttura

Vediamo innanzitutto cos'è il magazzino:

*"Il magazzino è una **struttura logistica** che, insieme alle attrezzature di stoccaggio e movimentazione, alle risorse umane e gestionali, consente di regolare le differenze tra i **flussi di entrata delle merci** (ricevute dai fornitori o dai centri produttivi ad esempio) e quelli **di uscita** (le merci inviate alla produzione o la vendita)."*

(Mecalux, 2020)

Analizziamo meglio questa definizione.

Il magazzino è una **struttura logistica**: è cioè un edificio il cui compito primario è la logistica. Ma cos'è quest'ultima? È definita come "*Attività volta a garantire il funzionamento di un sistema, in modo tale che le risorse necessarie siano disponibili nella quantità, nel luogo e nel momento richiesti*." (TRECCANI, 2020). Nel nostro caso le risorse sono le merci, e il renderle disponibili in maniera adeguata significa controllare i **flussi di entrata** (merce che **entra** nel magazzino) e **i flussi di uscita** (merce che **esce** dal magazzino) per far sì che il **sistema** (l'azienda che possiede, tra le altre cose, il magazzino) funzioni in maniera corretta ed efficiente (per esempio, per garantire che ci siano sempre scorte sufficienti a far funzionare a pieno regime le linee di produzione, bisogna che il flusso di **entrata** sia almeno **equivalente** a quello di **uscita**).

Ricapitolando quindi, il magazzino è una struttura che ha il compito di assicurare il corretto funzionamento di un sistema regolando i flussi di entrata e di uscita delle merci e garantendone il costante approvvigionamento nelle quantità, luoghi e momenti richiesti.

Ora che sappiamo cos'è un magazzino, andiamo a vedere com'è strutturato, cioè com'è organizzato.

Un magazzino è solitamente diviso in varie aree dalle funzionalità diverse, andiamo a vedere quali sono le principali:

- **Area accettazione/ricevimento**: è l'area predisposta all'accoglienza delle merci che, per esempio, possono essere scaricate dai camion tramite le apposite baie di carico/scarico. Una volta scaricate vengono controllate più o meno a fondo (per verificarne la qualità, l'integrità e la corrispondenza con ciò che si è ordinato);
- **Area stoccaggio**: in quest'area sono stoccate tutte le merci del magazzino ed è qua che finiscono quelle che superano un eventuale controllo nell'area di accettazione. Quest'area è anche la più importante per quanto riguarda questa tesi e verrà quindi approfondita in seguito.
- **Area spedizione**: in quest'area si preparano gli ordini e le spedizioni delle merci. Una volta preparato un ordine, la merce viene caricata (ancora una volta, per esempio, tramite una baia di carico/scarico) e spedita.

L'area che ci interessa maggiormente è l'**area di stoccaggio**, poiché è quella su cui agisce l'algoritmo; esso, infatti, viene richiamato nel momento in cui si vuole andare a stoccare della merce in tale area e si vuole trovare una locazione adatta ad essa. L'area di stoccaggio verrà approfondita nel capitolo dedicato al WMS.

2.2. La merce e il suo ciclo di vita

Parliamo ora della merce.

Essa rappresenta un qualsiasi prodotto/articolo che entra in magazzino, sia che arrivi da un fornitore esterno, sia che arrivi come semilavorato o prodotto finito da una linea di produzione interna.

In *Figura 1* è rappresentato il ciclo di vita della merce, da prima che entri in magazzino, fin quando esce da esso.

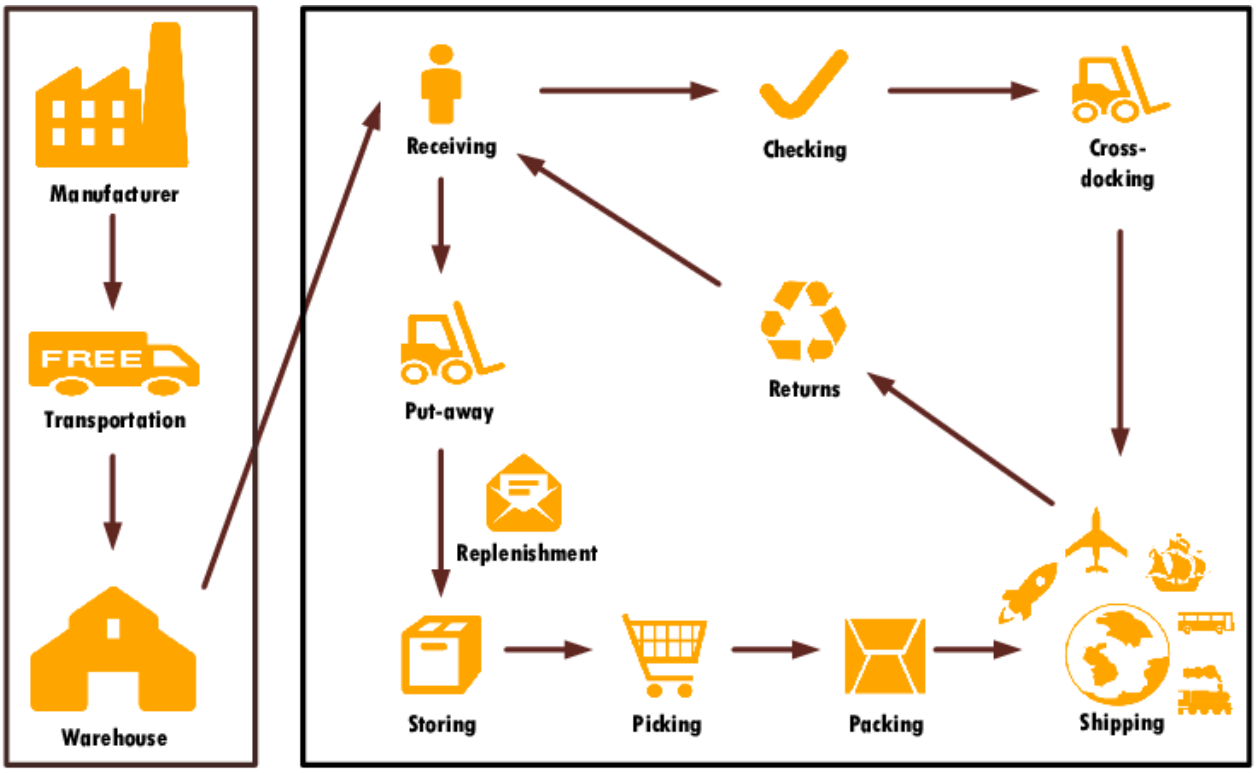


Figura 1: Ciclo di vita della merce (Koton, 2013)

Tra le varie fasi quella che ci interessa è il "*Put-away*", cioè lo **stoccaggio**. Prima di parlarne meglio cerchiamo però di descrivere velocemente l'interno ciclo, partendo dalla fase di "*Receiving*" (le parti precedenti non sono altro che l'arrivo della merce in magazzino, cioè un **flusso di entrata**):

- Il "*Receiving*" è la fase durante la quale la merce viene ricevuta all'interno del magazzino, ad esempio tramite lo scarico di essa da un camion presso una baia di scarico;
- A questa fase possono seguire due fasi diverse: il "*Put-away*", cioè lo stoccaggio della merce in magazzino o il "*Cross-docking*" cioè spedizione quasi immediata di merce appena arrivata e scaricata; vediamole ora in dettaglio;

- "*Cross-docking*": questa fase è preceduta da una verifica della merce appena arrivata in cui ci si assicura che sia integra e conforme a ciò che è stato dichiarato (fase di "*Checking*"). Ultimato questo controllo, la merce non viene portata in una locazione del magazzino, ma viene invece preparata per essere immediatamente spedita. La merce è quindi di passaggio nel magazzino e non sosta in esso;
- "*Put-away*": cioè lo stoccaggio, o messa a dimora, della merce che viene trasferita dalla baia di scarico all'interno del magazzino, in una delle locazioni disponibili. La tesi si concentrerà su questa fase;
- "*Storing*": il processo in cui la merce viene immagazzinata, magari per completare un'azione di rifornimento ("*Replenishment*"). Non è solo una fase ma anche uno **stato** in cui si trova la merce. Essa è cioè stoccata in magazzino, pronta ad essere recuperata e spedita o utilizzata e lavorata in qualche linea di produzione;
- "*Picking*": è il recupero della merce dalla locazione di stoccaggio. Può essere manuale o automatico, a seconda che il recupero venga effettuato da un operatore o da un robot;
- "*Packing*": in questa fase, le merci recuperate tramite *picking* vengono imballate e preparate per la spedizione;
- "*Shipping*": non è altro che la spedizione delle merci precedentemente recuperate dal magazzino e imballate;
- "*Returns*": è la restituzione della merce che può avvenire per svariati motivi (non conforme all'ordine, danneggiata, scaduta, ...). Ovviamente non avviene sempre, per cui è una fase *opzionale*.

2.3. Warehouse Management System (WMS)

Il **WMS** è un software il cui obiettivo è quello di agevolare la gestione del magazzino (il nome significa infatti Sistema per la gestione di magazzino). Alcune delle sue funzioni sono:

- **Gestire l'entrata delle merci**: in questo caso il WMS opera insieme all'ERP (Enterprise Resource Planning, software che si occupare della gestione completa delle attività aziendali, tra cui la parte amministrativa e quindi relativa agli ordini in entrata) per semplificare il controllo e l'aggiunta a stock della merce;
- **Stoccare le merci**: il WMS deve proporre una locazione dell'area di stoccaggio del magazzino dove riporre le merci appena arrivate. Per farlo ha bisogno di avere una visione completa dello stato attuale della merce già stoccata, della configurazione fisica del magazzino e delle caratteristiche della merce da stoccare. La tesi si pone come obiettivo quello di realizzare un algoritmo che faciliti questa operazione;
- **Gestire lo stock**: significa che il WMS ha una visione in tempo reale dello stato del magazzino, cioè delle scorte e della giacenza, dove è allocata la merce e in che quantità;
- **Gestire l'uscita delle merci**: il WMS si occupa di gestire la fase di picking (aiutando e guidando gli operatori nel recupero delle merci) in preparazione agli ordini, per poi preparare la documentazione necessaria alla spedizione e guidare la fase di carico. Infine, dialoga ancora una volta con l'ERP per segnalare l'ordine in uscita.

Il WMS si occupa quindi di gestire, nei fatti, la merce che entra ed esce dal magazzino. Ma continuare a parlare solo e genericamente di merce non sarebbe del tutto corretto per cui bisogna introdurre il termine UDC.

Per **UDC (Unità Di Carico)** si intende l'unità base di movimentazione e stoccaggio in magazzino, o anche unità logica di movimentazione atomica.

Unità logica perché varia a seconda del contesto in cui ci si ritrova: per alcuni utilizzatori potrebbe trattarsi di un pallet, per altri di uno scatolone, ecc.

Movimentazione atomica perché è l'unità minima che può essere movimentata in magazzino. Quindi, se per esempio un UDC è definita come pallet, non si potranno movimentare i singoli elementi che lo compongono, ma solo l'intero pallet.

All'interno di un magazzino tutto ruota intorno alle UDC, e così è nel WMS.

Oltre alle UDC però, il WMS deve essere anche a conoscenza di come è fatto il magazzino e in maniera particolare, di come è strutturata l'area di stoccaggio. Visto che la tesi è stata sviluppata all'interno di un'azienda, il WMS di riferimento è quello creato ed utilizzato da Onit cioè **On.Plant/WMS** per cui il modello adottato per descrivere l'area di stoccaggio è quello utilizzato all'interno di questo WMS.

Nel nostro caso avremo quindi un'area di stoccaggio divisa in più **aree**. In ogni area saranno presenti più **scaffali**, e ognuno di essi avrà più **campate**. Ogni campata sarà divisa in più **piani** e ogni piano potrò avere 1 o più **locazioni.** Le UDC vengono collocate nelle singole locazioni.

In *Figura 2* si può vedere un semplice di schema che rappresenta un'area di stoccaggio divisa in aree, ognuna con i propri scaffali. Come si può vedere, le aree sono identificate in maniera univoca all'interno dell'area di stoccaggio, mentre gli scaffali hanno un id univoco all'interno dell'area di appartenenza.

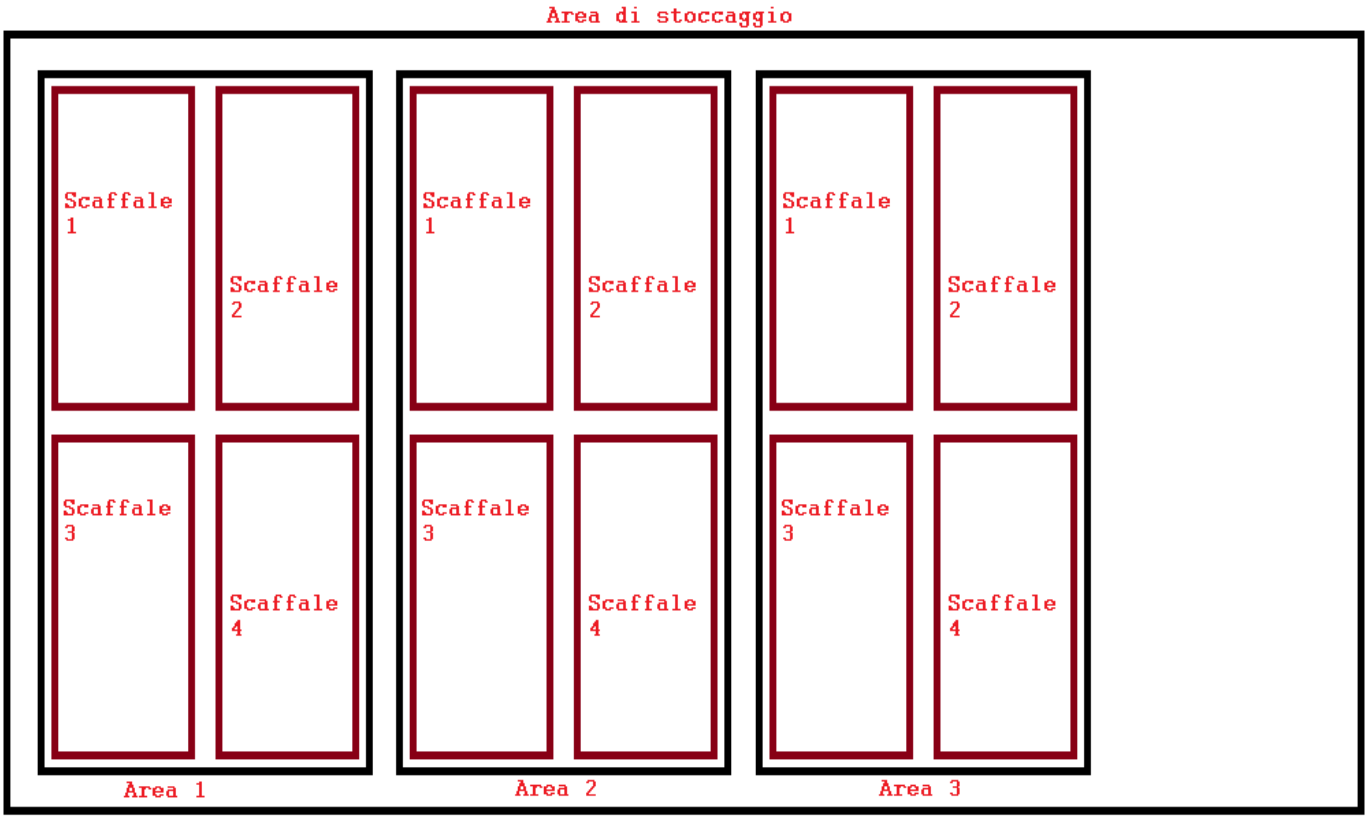


Figura 2: Schema area di stoccaggio

In *Figura 3* è invece mostrato lo schema di uno scaffale, formato da 4 campate, ognuna con 6 piani. In questo esempio nelle prime tre campate per ogni piano è presente una sola locazione, mentre i piani della quarta campata sono stati divisi (segno verticale rosso) per avere ognuno due locazioni.

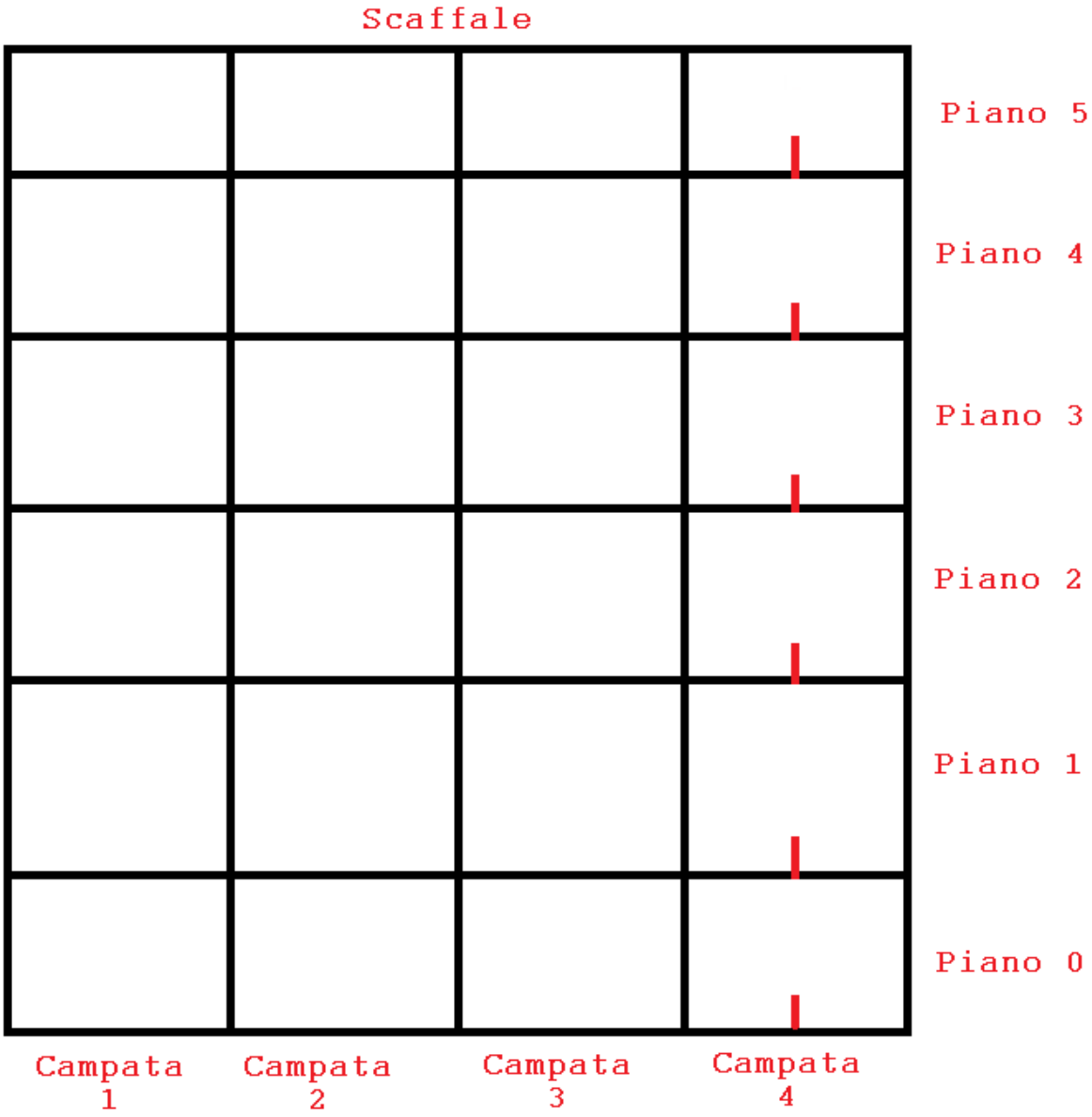


Figura 3: Schema di uno scaffale

2.4. Lo stoccaggio[1]

Abbiamo visto cos'è il magazzino, com'è strutturato, cosa si intende per merce e qual è il suo ciclo di vita e cos'è il WMS. Rimane da spiegare perché sia così importante lo stoccaggio, cosa lo rende fondamentale all'interno delle logiche di magazzino.

Partiamo però dando una definizione al termine **stoccaggio (put-away):**

"Lo stoccaggio è normalmente considerato come il processo durante il quale la merce appena arrivata viene spostata dall'area (baia) di scarico, o dal reparto di produzione a una delle locazioni di stoccaggio."

(Hatlevik, 2011)

Perché questo processo è così importante?

Una prima risposta la si può ottenere osservando il *ciclo di vita della merce*[2]: lo stoccaggio è infatti una delle prime fasi, e dipendendo ognuna dalla precedente, significa che gestire in maniera efficiente una fase a monte porta benefici in tutte quelle seguenti. Stoccare in maniera efficiente significa avere un magazzino più efficiente, poiché si migliorano le fasi successive, come il picking e quindi lo shipping, inoltre si ha una miglior visione dello stato attuale delle scorte.

Per contro, ignorare o sottovalutare questa fase, gestendola in maniera troppo semplicistica, per esempio, accatastando la merce senza logica, collocandola nel primo spazio libero che si trova, avrà un impatto negativo sull'intero magazzino:

- Si occuperà male lo spazio, il che vuol dire utilizzarlo in maniera inefficiente;
- Si creeranno congestioni: la merce in entrata dovrà aspettare che quella accatastata venga stoccata, e si avranno maggiori difficoltà e tempi più lunghi nel reperire la merce da spedire;
- Venendo impilata e maneggiata spesso dagli operatori per essere spostata, si avranno più probabilità di danneggiarla.

Ma cosa vuol dire stoccare in maniera efficiente?

Innanzitutto, significa affidare questa fase al WMS, e non agli operatori di magazzino. Questo porta ad ottenere dei benefici quali:

- La garanzia del rispetto di tutte le regole di stoccaggio presenti nel magazzino, cosa che risulterebbe più difficile, se non impossibile, affidandosi

[1] Le informazioni riportate in questo paragrafo fanno riferimento a: (Bowles, 2020), (Cyzerg, 2020), (Hatlevik, 2011) e (Everything Warehouse, s.d.)
[2] Figura 1: Ciclo di vita della merce

esclusivamente alla memoria umana, soprattutto in medi e grandi magazzini. Questo porta ad un miglior utilizzo degli spazi;
- Il miglioramento delle fasi successive: il picking è più efficiente, se le merci sono stoccate anche in previsione di come dovranno essere recuperate;
- Non si dipende più dalla familiarità che hanno gli operatori con il magazzino, affinché lo stoccaggio avvenga in maniera efficiente;
- Si velocizza lo stoccaggio: non è più possibile che un operatore venga indirizzato presso una locazione non adatta allo stoccaggio, perché quest'ultime vengono escluse a priori.

Affinché il WMS operi in maniera efficacie, attraverso un algoritmo per lo stoccaggio, ha bisogno che gli vengano forniti in input dei dati validi e completi da cui poter trarre informazioni utili. Queste informazioni possono essere di tanti tipi (volumi di vendita, dimensioni delle UDC, caratteristiche del prodotto, ecc.) e la scelta di quali raccogliere dipende da cosa si vuole ottenere: se si vogliono evitare sprechi di spazio sarà utile raccogliere informazioni sulle dimensioni, sia delle merci in entrata, sia delle locazioni disponibili; se si punta a voler velocizzare gli spostamenti in magazzino sarà invece utile avere a disposizione i dati relativi ai volumi di vendita (analisi ABC, come vedremo più avanti). L'algoritmo deve cioè poter rispondere alle differenti esigenze dei diversi magazzini su cui opera.

Ci sono però alcune informazioni che vengono sempre raccolte, perché, generalmente, consentono sempre di rendere più efficiente lo stoccaggio, e sono (a partire dalla più utile):

1. La *velocità* dell'articolo;
2. Il tipo di package;
3. L'utilizzo che si fa dell'articolo.

Di queste tre informazioni, la più importante è senz'altro la prima, che verrà infatti approfondita in seguito.

Il package di un UDC ne indica il tipo contenitore, ed è importante da sapere perché contenitori diversi possono avere requisiti di stoccaggio diversi che dovranno essere considerati dall'algoritmo.

Il tipo di utilizzo dell'articolo consente invece di stoccare in vista del futuro utilizzo dell'articolo stesso: se ad esempio un UDC contiene un tipo di articolo che verrà sicuramente utilizzato in una determinata linea di produzione, avrà senso stoccarla vicino a quella linea.

2.4.1. Velocità dell'articolo[3]

Il concetto di *velocità di un articolo* (detta anche *velocità di rotazione*) è legato all'**analisi ABC**, che si basa sul **principio di Pareto**. Andiamo ora a vedere questi tre concetti a partire dall'ultimo.

Il **principio di Pareto** si fonda su uno studio statistico effettuato da Vilfredo Pareto (1848-1923), ingegnere, economista e sociologo italiano, che nel 1897 studiò i dati relativi alla distribuzione della ricchezza in Italia, e constatò che il 20% ca della popolazione possedeva l'80% ca delle terre. Da questo studio venne fuori la cosiddetta "legge 80/20", secondo la quale l'80% degli effetti è dovuto dal 20% delle cause (i valori sono ovviamente indicativi, ma servono a rendere l'idea).

Questa legge empirica è stata applicata efficacemente a diversi ambiti, tra cui anche alla gestione dei magazzini. In maniera particolare, si verifica che l'80% del fatturato viene generato dal 20% degli articoli presenti in magazzino.

Su questa legge si basa l'**analisi ABC**: essa prevede di dividere gli articoli in tre classi (A, B e C) in base all'importanza che hanno. Quel 20% di merce che genera l'80% del fatturato, rappresenta la classe A, cioè quelli articoli che devono sempre essere presenti in magazzino. Il restante 80% degli articoli genera solo il 20% del fatturato, e si divide nelle rimanenti 2 classi, B e C.

Un altro modo di applicare questa analisi è quello di basarsi sulla **velocità** degli articoli: vengono cioè divisi in base al numero di volte che vengono movimentati (sia per picking che per semplici cambiamenti di locazione). Gli articoli che subiscono più movimenti sono quelli di classe A, quelli meno mossi sono di classe C: i primi dovranno essere messi in locazioni facilmente raggiungibili (vicine all'area di spedizione o produzione e ad altezza uomo), mentre quelli di classe C potranno essere stoccati in locazioni difficili da raggiungere (verso la fine del magazzino, quindi più lontane dall'area di spedizione o produzione, o nei piani più alti, raggiungibili con carrelli trasloelevatori).

Prendere in considerazione la velocità di rotazione di un articolo al momento dello stoccaggio, consente di migliorare le fasi successive, diminuendo il tempo impiegato nel picking, e velocizzando le spedizioni.

[3] Le informazioni riportate in questo paragrafo fanno riferimento a: (Logipack, s.d.), (Logistica Efficiente (Analisi di Pareto), s.d.) e (Logistica Efficiente (Analisi ABC scorte/fatturato), s.d.)

3. Tecnologie utilizzate

Le tecnologie utilizzate per lo sviluppo dell'algoritmo sono:

- C# come linguaggio di programmazione;
- Visual Studio 2015 come IDE;
- .NET Framework;
- LINQ per interrogare il DB;
- Entity Framework come mapper di database a oggetti.

Il WMS è realizzato in larga parte con le stesse tecnologie sopra riportate, e questo è il motivo per cui esse sono state scelte per realizzare anche l'algoritmo.

La metodologia di sviluppo utilizzata si basa sul modello iterativo, infatti, nonostante in questa tesi l'analisi, la progettazione e l'implementazione appaiano tutte accorpate, in realtà sono state eseguite ciclicamente.

Per ogni funzionalità implementata sono state eseguite:

- **Analisi**: descrizione del problema, per comprenderlo meglio, capire cosa effettivamente viene richiesto, e quindi determinare quale debba essere il comportamento desiderato dall'algoritmo;
- **Progettazione**: scrittura di pseudo-codice utile a capire come il problema debba essere risolto;
- **Implementazione**: questa fase è a sua volta divisa in:
    - **Scrittura dei test**: per ogni funzionalità, è stato creato un test set ritenuto adatto a verificare la correttezza del codice scritto;
    - **Implementazione**: traduzione dello pseudo-codice in C#;
    - **Testing**: una volta finita l'implementazione, sono stati lanciati i test scritti precedentemente, e sono state eseguite le dovute correzioni al codice.

4. Introduzione all'Analisi

La prima parte dell'analisi sarà dedicata allo studio dell'algoritmo *as-is*, cioè alla versione già implementata e funzionante dell'algoritmo per la proposta di stoccaggio presente all'interno del WMS. Questo per capire come è stato affrontato il problema in passato e quale sia la soluzione attualmente presente.

La seconda parte invece sarà dedicata alla raccolta e all'analisi dei requisiti per il nuovo algoritmo.

N.B. Nel resto della tesi, quando si parlerà di *algoritmo per la proposta di stoccaggio **V1** (versione 1)*, si sta facendo riferimento alla versione già implementata.
L'algoritmo che si vuole analizzare, progettare e implementare sarà invece l'*algoritmo per la proposta di stoccaggio **V2** (versione 2)*.

5. Analisi dell’algoritmo per la proposta di stoccaggio V1

5.1. Funzionamento dell’algoritmo as-is

In *Figura 4* è raffigurato lo schema riassuntivo di funzionamento dell’algoritmo V1.

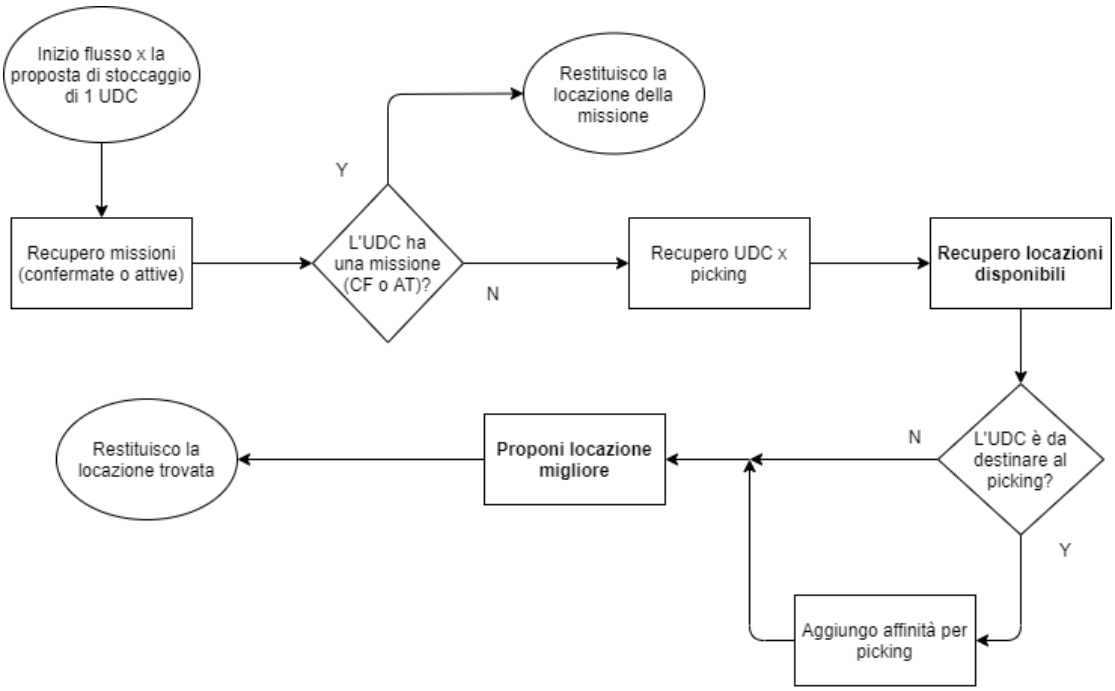


Figura 4: Flusso dell'algoritmo per la proposta di stoccaggio V1

Analizziamolo meglio.

5.2. Recupero missioni

La prima operazione che l’algoritmo svolge è il *recupero missioni*. Ma cos’è una missione per il WMS?

Una **missione** rappresenta uno spostamento di un’UDC, da un punto A ad un punto B del magazzino, e si può trovare in vari stati (attiva, confermata, eseguita, ecc.).

Come si può facilmente immaginare, in un magazzino si hanno continuamente UDC in movimento, per cui si avranno continuamente delle missioni, ognuna associata ad un’UDC, alla locazione di partenza e a quella di destinazione. Quello che interessa in questa fase, è trovare tutte le missioni che devono essere eseguite (stato confermata) e

tutte quelle che si stanno eseguendo (stato attiva), per verificare se l'UDC che si vuole stoccare è associata ad una di esse: se sì, la locazione proposta non è altro che la locazione di destinazione della missione.

5.3. Recupero UDC per picking

Se invece l'UDC non ha missioni assegnate, l'algoritmo prosegue e passa al *recupero UDC per picking*.

Come abbiamo visto in precedenza, il picking è il recupero della merce dalla locazione di stoccaggio.

Succede spesso che, quando viene effettuato il picking, non viene recuperata l'intera UDC, ma solo la quantità necessaria per sopperire all'ordine, per cui l'UDC pian piano si svuota (si pensi per esempio ad un UDC formata da N scatoloni, ognuno contenente dieci buste di un particolare prodotto. Quando si fa picking si potrebbe aver necessità di recuperare solo un piccolo quantitativo di buste, per cui queste vengono prelevate ma le restanti rimangono, e l'UDC stessa rimane nella locazione ma contiene sempre meno quantitativo dell'articolo). Le locazioni presso le quali si effettua il picking prendono il nome di **locazioni di picking**.

Quello che si vuole fare è evitare che le locazioni di picking si svuotino completamente, per cui in questa fase vengono recuperate, basandosi sulla giacenza attuale, quali sono le UDC che devono essere destinate al rifornimento delle locazioni di picking.

5.4. Recupero locazioni disponibili

Successivamente si effettua il *recupero delle locazioni disponibili*, cioè vengono prelevate tutte le locazioni presenti all'interno del magazzino, che hanno determinate caratteristiche. In *Figura 5* è mostrato il flusso completo di questa fase.

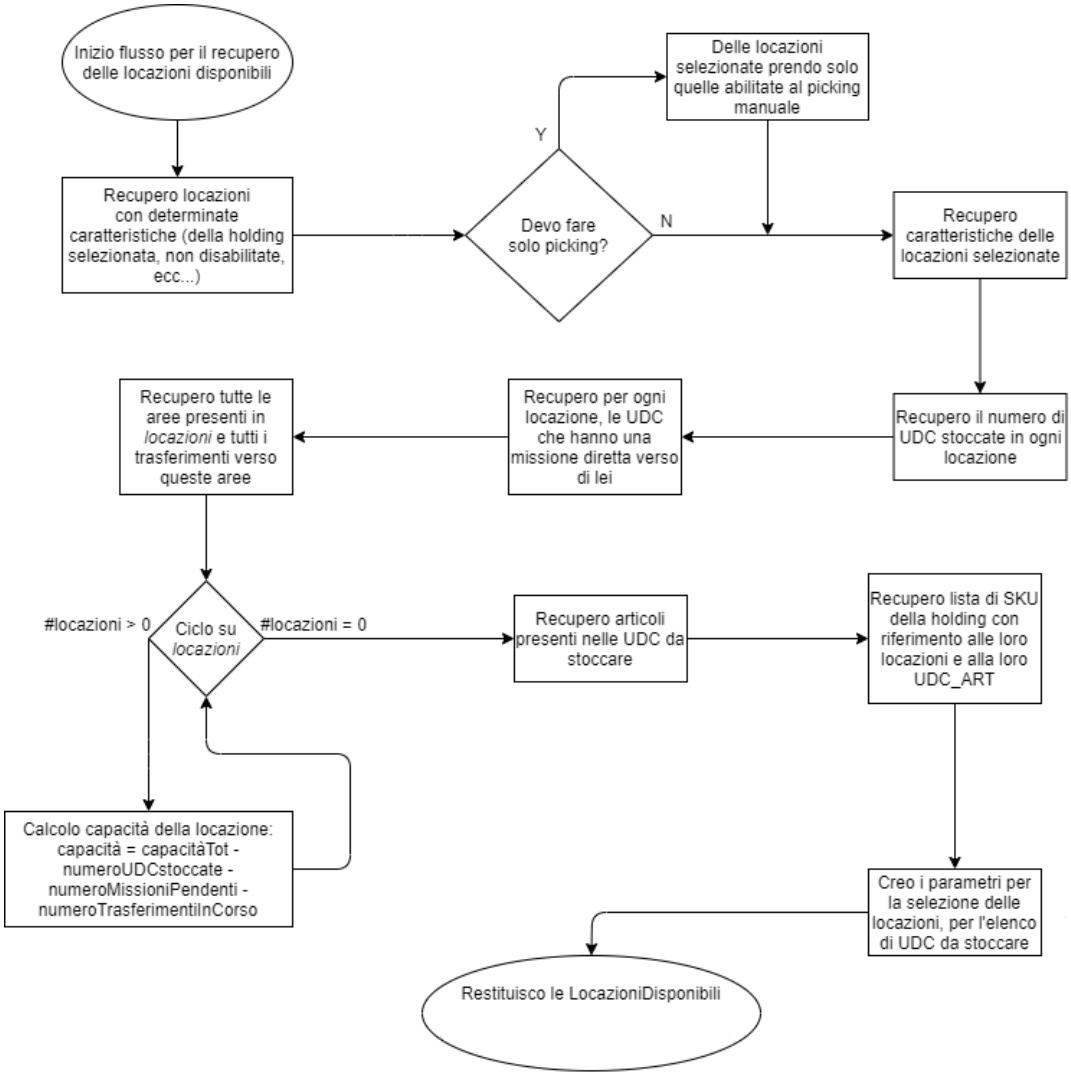


Figura 5: Flusso recupero locazioni disponibili

Per prima cosa si recuperano tutte le locazioni con determinate caratteristiche (non disabilitate e non virtuali per esempio). Può succedere poi che si vogliano prendere in considerazione solo le locazioni di picking, vengono quindi escluse tutte le altre (primo rombo del flusso).

La parte del flusso che va dal *Recupero caratteristiche delle locazioni selezionate* al *Ciclo su locazioni* serve a calcolare la capacità di ogni locazione. La capacità indica quante UDC una locazione può ancora accogliere e viene trovata partendo dalla capacità massima di una locazione (espressa come numero di UDC che possono

esservi stoccate), a cui viene sottratto il numero di UDC attualmente stoccate ed il numero di missioni pendenti e trasferimenti che hanno come destinazione quella locazione.

Nell'ultima parte del flusso vengono recuperati gli articoli presenti nell'UDC da stoccare e la rispettiva lista di SKU.

La **SKU (Stock Keeping Unit)** è un codice che identifica in maniera univoca la merce. In questo caso essa è rappresentata da un insieme di cinque informazioni:

- Codice dell'articolo: codice identificativo univoco dell'articolo;
- Conformazione: come l'articolo è fisicamente organizzato (1 pallet, con 50 cartoni, con 10 buste per cartone, ognuna dal peso di 0.5 kg);
- Attributo: campo jolly;
- Lotto: un numero che viene assegnato ad uno o più prodotti in fase di produzione, per facilitarne la tracciabilità;
- Scadenza: data entro la quale il prodotto va consumato.

La SKU consente di caratterizzare in maniera sintetica ma completa la merce, e quindi l'UDC, che si vuole stoccare e consente la creazione dei *parametri per la selezione della locazione* che verranno utilizzati durante la ricerca della locazione migliore. A questi parametri, ne viene eventualmente aggiunto un altro, che indica una preferenza (o affinità) alle locazioni picking (secondo rombo in *Figura 4*).

5.5. Proposta della miglior locazione

Una volta recuperate le locazioni con tutte le relative informazioni, si può procedere alla ricerca della miglior locazione in cui stoccare l'UDC.

Durante questo procedimento verranno utilizzati i *parametri per la selezione della locazione* creati precedentemente.

In *Figura 6* è raffigurato il flusso completo della proposta della miglior locazione.

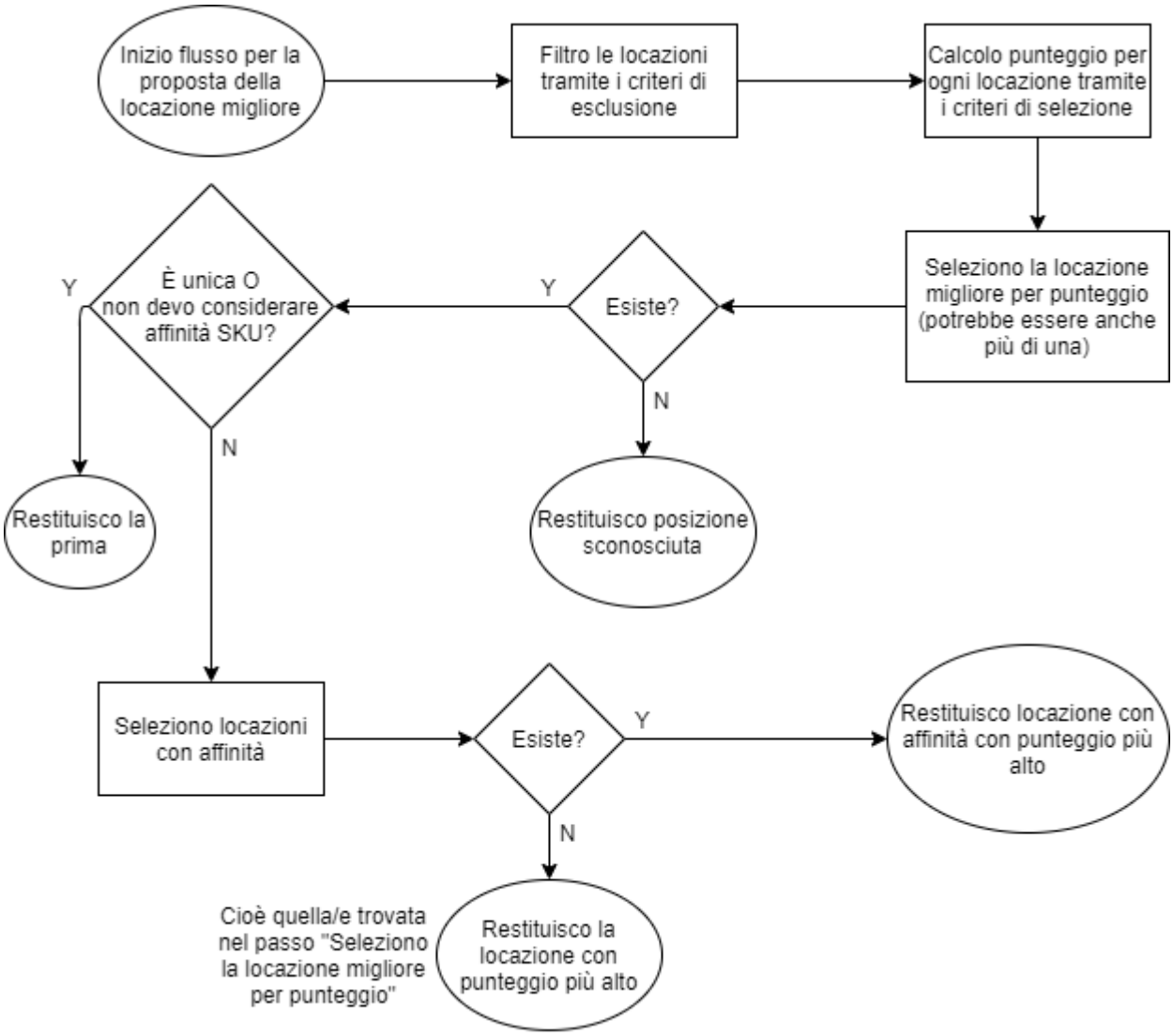


Figura 6: Flusso proposta miglior locazione

La prima cosa che viene fatta, è un filtraggio delle locazioni basato su dei criteri d'esclusione **opzionali**:

- Riservata per articolo: si verifica se la locazione è riservata per un determinato articolo, in caso affermativo si confronta se l'articolo è lo stesso di quello presente nell'UDC: se non lo è la locazione viene scartata;
- Riservata per soggetto: si verifica se la locazione è riservata per uno o più soggetti in caso affermativo si verifica se l'UDC appartiene ad uno di essi: se non appartiene a nessuno di quelli in elenco, la locazione viene scartata;
- Gruppo di stoccaggio: ad ogni articolo possono essere associati uno o più tag che rappresentano dei gruppi di stoccaggio (infiammabile, commestibile, ecc.); una locazione può essere riservata esclusivamente ad alcuni di essi, per cui potrà accogliere solo UDC contenenti articoli con quel/quei gruppi di stoccaggio;
- Altezza piano: se presente, si verifica semplicemente se l'altezza dell'UDC è compatibile con quella del piano (altezza UDC <= altezza piano);
- Peso massimo consentito sul piano: se presente, si verifica se il peso dell'UDC è compatibile con quello massimo del piano (peso UDC < peso max piano).

ATTENZIONE: mentre l'altezza e il peso sono VINCOLANTI (cioè, se presenti, devono essere rispettati) gli altri tre criteri possono esserlo come no, cioè, è possibile, per esempio, che ad una locazione siano associati dei gruppi di stoccaggio ma non venga richiesto l'obbligo di stoccarvi solo articoli con quel determinato gruppo.

Una volta escluse le locazioni non compatibili con gli eventuali filtri presenti, si procede ad un calcolo di punteggio per ogni locazione. Il calcolo si basa sui *parametri per la selezione della locazione* che sono poi, in parte, gli stessi criteri usati per i filtri precedenti, in maniera particolare vengono utilizzati:

- Articolo: si verifica se la locazione è riservata per determinati articoli e se l'articolo da stoccare è tra questi; se sì, si incrementa il punteggio della locazione;
- Gruppo di stoccaggio: si verifica se alla locazione sono associati dei gruppi di stoccaggio e, se sì, si controlla se sono associati anche all'articolo: in caso affermativo viene incrementato il punteggio della locazione;
- Soggetto: si verifica se la locazione è associata ad un soggetto, se sì, se verifica se uno tra proprietario e destinatario dell'UDC è tra essi; in caso affermativo si incrementa il punteggio della locazione.

Una volta che è stato assegnato un punteggio ad ogni locazione, si ordinano in base ad esso e si seleziona quella col punteggio più alto. Se a causa dei filtri non è stato possibile trovare alcuna locazione, si restituisce un valore predefinito (locazione

sconosciuta) per indicare che non ne è stata trovata nessuna (primo rombo in *Figura 6*). Altrimenti si prosegue e si verifica se quella col punteggio più alto è unica o se non bisogna considerare la SKU dell'UDC da stoccare: in questi casi si restituisce l'unica/la prima locazione recuperata.

In caso contrario (ho più locazioni con lo stesso punteggio, o devo considerare la SKU) effettuo un ulteriore filtraggio basandomi sulla SKU dell'articolo da stoccare, cioè classifico le locazioni in base alla somiglianza tra la SKU da stoccare e le SKU dei prodotti già stoccati. Se da questa operazione si ottengono 1 o più locazioni, viene restituita la prima. Altrimenti si restituisce la prima di quelle trovare precedentemente al filtraggio per SKU.

In entrambi i casi la locazione restituita è quella che, per l'algoritmo, è la migliore possibile per l'UDC da stoccare.

6. Analisi dell’algoritmo per la proposta di stoccaggio V2

In questo capitolo verranno prima raccolti i requisiti, poi verrà effettuata un’analisi che consenta di poter progettare e implementare la nuova versione dell’algoritmo. La fase di raccolta prevede di elencare i requisiti, e dar loro una breve descrizione. La fase di analisi serve invece a comprendere cosa effettivamente viene richiesto al nuovo algoritmo e se la richiesta è fattibile oppure no.

6.1. Raccolta dei requisiti

Come detto nell’***Introduzione***, la tesi è stata svolta all’interno di un’azienda, per cui l’algoritmo V1 è quello utilizzato nel WMS OnPlant/WMS. La creazione dell’algoritmo V2 non è perciò fine a sé stessa, ma servirà a sostituire un prodotto già utilizzato da varie aziende clienti, e si basa sulle richieste che queste stesse aziende hanno fatto. La fase di raccolta, quindi, non è frutto di un mio personale lavoro, ma del lavoro di altre figure professionali interne all’azienda, che ne hanno condiviso i risultati.

L’elenco dei requisiti comprende:

- richieste che saranno effettivamente analizzate, progettate e implementate;
- richieste che dopo una prima analisi non risulteranno realizzabili;
- richieste che seppur realizzabili, non sono state progettate né implementate perché è stato deciso di dare la priorità ad altre o non c’è stato il tempo all’interno di quello previsto dalla tesi.

I requisiti raccolti sono:

- Valutare gli ingombri volumetrici, cioè per ogni locazione sapere quant’è lo spazio occupato dalle UDC stoccate (sapere non solo le dimensioni ma anche come sono posizionate all’interno della locazione) e quanto quello disponibile per futuri stoccaggi;
- Durante lo stoccaggio valutare le modalità di prelievo per agevolarle. Le modalità sono:

    - FIFO (First In First Out): le UDC stoccate per prime devono essere prelevate per prime;
    - LIFO: (Last In First Out): le UDC stoccate per ultime sono le prime ad essere prelevate;
    - FEFO (First Expired First Out); in questo caso bisogna prelevare per prime le UDC contenente SKU con la data di scadenza più vicina;
- Valutare il peso;
- Valutare le zone geografiche di appartenenza di proprietari/destinatari delle UDC;
- Avere delle locazioni statiche, cioè a cui sono associate determinati articoli e solo essi possono esservi stoccati;
- Valutare di stoccare le UDC vicino a delle locazioni di picking dello stesso articolo (posizionare cioè le scorte vicine a dove si fa picking);
- Gestire in maniera adeguata lo stoccaggio contemporaneo di più UDC (che non venga proposta la stessa locazione a due o più UDC se non è possibile stoccarvi più di una UDC);
- Valutare le classi di rotazione (Classi A/B/C, come visto nel paragrafo ***Velocità dell'articolo***);
- Valutare il tipo di contenitore dell'UDC, affinché venga stoccata solo in piani adatti ad esso.

Di questi, sono stati scartati:

- La valutazione degli ingombri volumetrici **per come era stata proposta**: non è infatti possibile essere a conoscenza di **come** esattamente le UDC sono stoccate, cioè, se per esempio in una locazione sono stoccate due UDC, tramite il WMS non è possibile sapere se sono stoccate una di fianco all'altra, una sopra l'altra, ecc. **Sarà** invece **oggetto di analisi** una valutazione degli ingombri volumetrici più semplicistica, che tiene in considerazione le tre dimensioni sia delle UDC che della locazione, facendo però determinati assunti iniziali, che semplificano la realtà;
- La valutazione delle modalità di prelievo: essendo in parte già coperte da altri moduli del WMS (per esempio la politica FEFO viene rispettata tramite l'algoritmo che determina da quale UDC fare picking), non sono state inserite all'interno del nuovo algoritmo;
- Lo stoccaggio contemporaneo di più UDC, perché è stato considerato un caso molto particolare e non frequente.

Dei restanti requisiti sono invece stati analizzati, progettati e implementati:

- La valutazione degli ingombri volumetrici semplificata;
- La valutazione del peso;
- La valutazione delle classi di rotazione;
- La valutazione del tipo di contenitore.

È stato solo analizzato il requisito della valutazione delle zone geografiche.

L'utilizzo di locazioni statiche e lo stoccaggio vicino a locazioni di picking non sono state oggetto di analisi.

Prima di passare al prossimo capitolo, è giusto dare una spiegazione del perché è stato scelto di implementare alcuni requisiti piuttosto che altri:

- La gestione degli ingombri è stata scelta perché permette di iniziare ad affrontare il problema della **capacità rimanente** di una locazione, che finora era gestita in maniera molto semplicistica;
- La gestione del peso è stata scelta per due motivi: il primo è che garantisce un maggior controllo e una maggior sicurezza in magazzino (prima le portate massime di piani e campate non erano gestite dal WMS), il secondo è che era già stata richiesta in passato come funzionalità, ed era in corso di approfondimento, per cui si è deciso di affrontarla e implementarla;
- La valutazione delle classi di rotazione è stata scelta perché, come visto nel paragrafo ***Velocità dell'articolo***, è un parametro importante per qualsiasi magazzino, e che consente di migliorarne l'efficienza;
- La valutazione del tipo di contenitore è stata scelta perché, come per la gestione degli ingombri, rappresenta un primo passo verso una miglior gestione della capacità rimanente di una locazione.

6.2. Analisi dei requisiti

L'approccio utilizzato per analizzare i requisiti è quello delle User Stories, dei brevi paragrafi definiti dallo schema:

- **AS A [persona]:** definisco chi è l'utente che eseguirà l'azione;
- **I WANT TO [cosa si vuole fare]**: specifica l'azione che l'utente vuole eseguire;
- **SO THAT [il motivo per cui si svolge l'azione]:** specifica la motivazione per cui l'utente vuole eseguire l'azione.

Questo approccio di analisi permette di dare un significato più specifico ai requisiti raccolti, e di comprendere meglio quale problema essi descrivano, e perché lo si vuole risolvere.

ATTENZIONE: questo tipo di approccio è presente SOLO per i requisiti poi implementati. Per gli altri, è presente una normale analisi testuale.

6.2.1. Valutazione ingombri volumetrici

Per ingombro volumetrico si intende lo spazio occupato nelle tre dimensioni da un'UDC. Nell'algoritmo V1 viene considerata solo l'altezza dell'UDC, che, durante il filtraggio delle locazioni, viene confrontata con quella del piano, per verificare se sono compatibili (altezza piano >= altezza UDC). Non vengono considerate le altre due dimensioni, in nessun modo, e questo porta a più problemi:

- Non vengono escluse le locazioni che non sono abbastanza larghe per accogliere l'UDC;
- Non vengono escluse le locazioni che non sono abbastanza profonde per accogliere l'UDC;
- Non si tiene in considerazione lo spazio occupato dalle UDC già stoccate.

L'ultimo punto porta inoltre a considerare un altro aspetto, e cioè quello della **capacità rimanente** di una locazione. L'algoritmo V1, infatti, basa questo concetto su un indicatore numerico che indica la capacità massima (in numero di UDC) di una locazione. Questo tipo di gestione è ovviamente semplice, ma non tiene conto delle informazioni volumetriche delle UDC già stoccate, che consentirebbe di avere una gestione degli spazi più efficiente.

Si pensi ai casi in cui le UDC hanno dimensioni disomogenee tra loro e anche rispetto alle locazioni. Gestire la capacità solo tramite un numero può portare ad un notevole spreco di spazio perché si possono avere locazione con capacità rimanente uguale a 0, ma, per esempio, le UDC stoccate, viste le loro dimensioni, non occupano neanche metà della locazione.

In *Tabella 1* sono raffigurati i pro e i contro dei due diversi tipi di gestione della capacità.

Tabella 1: pro e contro delle diverse gestioni della capacità rimanente

Gestione capacità tramite numero di UDC stoccabili		**Gestione capacità tramite valutazione ingombro effettivo delle UDC**	
PRO	**CONTRO**	**PRO**	**CONTRO**
Gestione del problema notevolmente semplificata sia a livello logico che algoritmico	Casi in cui lo spazio disponibile viene sprecato; Dato poco significativo; Difficoltà da parte di un configuratore nel configurare correttamente questo dato	Ottimizzata la gestione dello spazio disponibile	Complicato da gestire a livello algoritmico

Per quanto la modalità di gestione della capacità tramite la valutazione degli ingombri effettivi sia la soluzione ideale, attualmente non è realizzabile, poiché non si hanno gli strumenti che consentono, per esempio, di sapere COME le UDC sono stoccate in una locazione (una di fianco all'altra, una sopra l'altra, una dietro l'altra, ecc. come mostrato in *Figura 7*) o quanto spazio effettivamente occupano (presa un'UDC larga 20 cm in una locazione larga 50 cm, se l'UDC è stoccata attaccata ad uno dei 2 lati, abbiamo uno spazio rimanente di 30 cm, ma se fosse stoccata esattamente al centro avremmo 15 cm per lato. Vedi *Figura 8*).

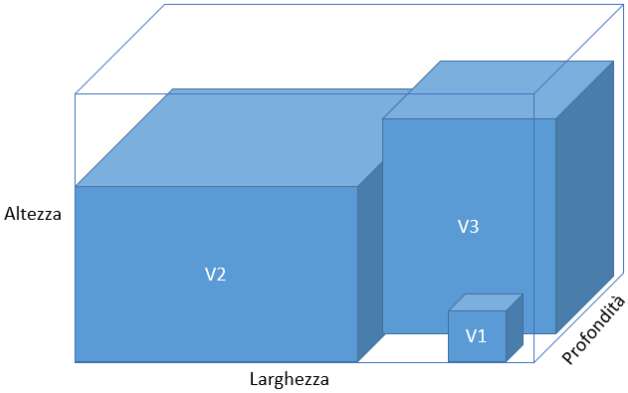


Figura 7: Esempio di una situazione non individuabile da WMS: si sa solo che nella locazione ci sono 3 UDC, il posizionamento non è conosciuto

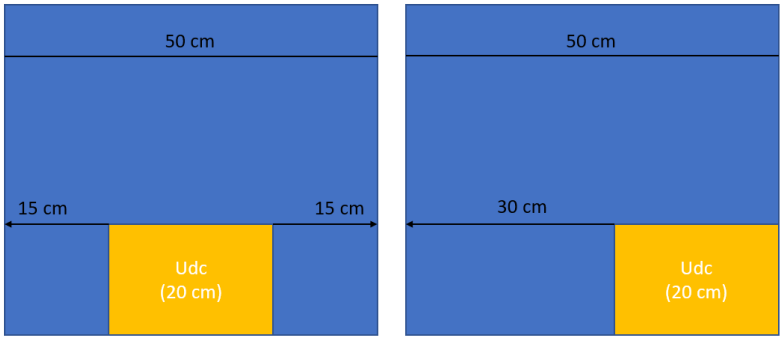


Figura 8: problema di volumetrica, legato alla locazione e all'UDC

Per affrontare comunque questo argomento, è stato deciso che, nella prima versione dell'algoritmo V2, verrà adottata un'approssimazione in quanto si assume che le UDC vengano stoccate l'una davanti all'altra e che occupino tutto lo spazio disponibile in larghezza e altezza (*Figura 9*).

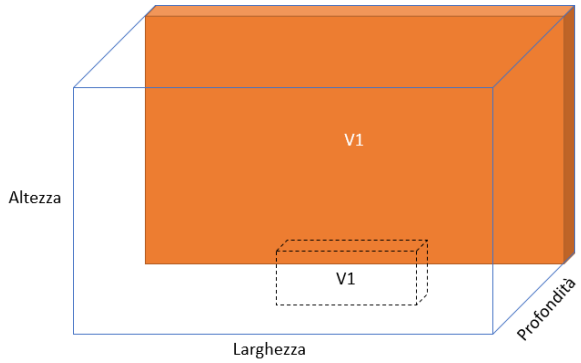


Figura 9: Le dimensioni reali dell'UDC sono quelle tratteggiate, ma per semplificare si considera occupato lo spazio arancione

I controlli che verranno eseguiti sono quindi:

- Controllo dell'altezza;
- Controllo della larghezza;
- Controllo della profondità;
- Controllo della profondità rimanente.

Per mantenere una continuità con l'algoritmo V1, si continuerà a dare la possibilità di controllare solo l'altezza, e verrà aggiunta la possibilità di controllare le dimensioni, come descritto nei quattro punti sopra. Andiamo ora a vedere questi due casi.

6.2.2. Controllo altezze piani

Come abbiamo visto in precedenza, il magazzino è diviso in aree, scaffali, campate, piani e locazioni. Su uno stesso piano possono esservi più locazioni, che quindi ne condividono alcune dimensioni, cioè altezza e profondità.

Questo controllo vuole verificare che l'altezza dell'UDC sia compatibile, cioè minore o uguale, con quella definita nel piano.

Ovviamente affinché il controllo abbia effetto, bisogna che sia l'UDC che le locazioni abbiano un'altezza inserita. Mentre per l'UDC è così (le dimensioni solitamente vengono prese al momento dello stoccaggio), per la locazione la cosa non è così ovvia, perché quest'informazione non è detto che sia stata inserita in fase di configurazione del magazzino.

Per mitigare il problema è stato deciso che il controllo potrà essere attivato a livello di area: l'attivazione sarà possibile SOLO dopo aver definito un'altezza per TUTTI i piani dell'area. Nel caso contrario non lo si potrà attivare, e in caso di aggiunta di un nuovo piano in un'area con questo controllo attivo si dovrà specificare OBBLIGATORIAMENTE un'altezza per quel piano.

In questa maniera al momento dello stoccaggio il comportamento è il seguente:

- Se l'UDC ha un'altezza dichiarata vengono presi solo i piani che hanno altezza compatibile oppure piani che fanno parte di un'area senza il controllo abilitato;
- Se l'UDC non ha un'altezza dichiarata vengono presi solo i piani che fanno parte di un'area senza il controllo abilitato.

Di seguito lo User Stories (US) definite per questo controllo.

US#1 - Configurazione altezza piano

AS A	Configuratore del sistema
I WANT TO	Poter assegnare ad un piano una specifica altezza
SO THAT	All'interno delle aree con il relativo controllo attivo, vengano confrontate le altezze dei piani e dell'UDC da stoccare per proporre solo locazioni compatibili

US#2 - Stoccaggio UDC con altezza dichiarata

AS A	Addetto allo stoccaggio di una Udc con altezza H
I WANT TO	Il WMS proponga locazioni di piani con altezza configurata superiore o uguale a H oppure locazioni di piani appartenenti ad un'area su cui non è attivo il controllo dell'altezza
SO THAT	Evitare lo stoccaggio dell'UDC in piani in cui non si riuscirebbe a svolgere la messa a dimora

US#3 - Stoccaggio UDC senza l'altezza dichiarata

AS A	Addetto allo stoccaggio di una UDC senza altezza dichiarata
I WANT TO	Il WMS mi proponga locazioni di piani appartenenti ad un'area su cui non è attivo il controllo dell'altezza
SO THAT	Circoscrivere le UDC senza altezza specificata alle sole aree in cui il controllo di questo parametro non è settato

6.2.3. Controllo dimensioni

Questo controllo serve a verificare che l'UDC sia compatibile con le dimensioni della locazione, e con le UDC già stoccate in essa. Anche in questo caso bisogna che l'UDC abbia le dimensioni definite, così come la locazione (larghezza e profondità) e il piano (altezza).

Questo controllo è in realtà suddiviso in quattro parti:

- **Controllo dell'altezza**: si utilizza il controllo visto in **6.2.2;**
- **Controllo della larghezza:** si verifica che la larghezza dell'UDC sia compatibile (cioè minore o uguale) con quella della locazione;
- **Controllo della profondità:** si verifica che la profondità dell'UDC sia compatibile (cioè minore o uguale) con quella della locazione;
- **Controllo della profondità rimanente:** partendo dalla profondità della locazione si sottraggono le profondità delle UDC già stoccate, e poi si verifica se la profondità rimanente è compatibile (cioè minore o uguale) con quella dell'UDC.

Il comportamento al momento dello stoccaggio è il seguente:

- Se l'UDC ha le dimensioni dichiarate, vengono prese solo locazioni che hanno dimensioni compatibili, o che non hanno dimensioni dichiarate;
- Se l'UDC non ha le dimensioni dichiarate, vengono prese solo locazioni che non hanno dimensioni dichiarate.

Di seguito lo User Stories (US) definite per questo controllo.

US#1 – Configurazione dimensioni

AS A	Configuratore del sistema
I WANT TO	Poter assegnare ad una locazione delle specifiche dimensioni
SO THAT	Nelle locazioni con dimensioni dichiarate vengano stoccate solo UDC compatibili

US#2 - UDC con dimensioni dichiarate

AS A	Addetto allo stoccaggio di una UDC con dimensioni X, Y e Z
I WANT TO	Il WMS proponga locazioni in cui l'UDC possa fisicamente entrare oppure locazioni che non hanno dimensioni dichiarate
SO THAT	Evitare lo stoccaggio dell'UDC in locazioni in cui sicuramente non si riuscirebbe a svolgere la messa a dimora

US#3 - Stoccaggio UDC senza dimensioni dichiarate

AS A	Addetto allo stoccaggio di una UDC senza dimensioni dichiarate
I WANT TO	Il WMS mi proponga locazioni che non hanno dimensioni dichiarate
SO THAT	Circoscrivere le UDC senza dimensioni specificate alle sole locazioni che non hanno dimensioni dichiarate

6.2.4. Controllo peso

Come abbiamo visto durante l'***Analisi dell'algoritmo per la proposta di stoccaggio V1,*** il peso è gestito in maniera molto semplice, si controlla solo se la portata massima definita sul piano sia compatibile (maggiore o uguale) con il peso dell'UDC. I problemi che possono sorgere sono principalmente due:

- Non si tiene conto delle UDC già stoccate: se un piano ha come portata massima 100 kg, l'UDC da stoccare pesa 50 kg, e sul piano sono già stoccate due UDC ognuna dal peso di 40 kg, quel piano DEVE essere escluso (40 kg + 40 kg + 50 kg = 130 kg > 100 kg), ma l'algoritmo V1 non lo esclude (verifica solo portata massima e peso da stoccare, cioè 50 kg <= 100 kg). Questo problema può creare situazioni di pericolo, e portare ad infortuni del personale e al danneggiamento del magazzino;
- Non si tiene conto delle portate delle campate: gli scaffali utilizzati nei magazzini, solitamente, non hanno solo definite delle portate massime per ogni piano, ma anche per ogni campata. Inoltre, **non è sempre vero** che la portata di una campata è uguale alla somma delle portate dei piani: si può avere come portata massima di una campata 500 kg, e avere 5 piani ognuno con portata massima 150 kg, per cui **entrambe** le informazioni (portata massima del piano e della campata) hanno senso di esistere e di essere prese in considerazione.

In questo caso, l'attivazione del controllo non è semplicemente a livello di area ma verrà strutturato in maniera più articolata:

- **Area** → abilitazione controllo portata **campate;**
  - **Scaffale**
    - **Campata** → abilitazione controllo portate **piani** + **portata campata** + abilitazione controllo piani a terra
      - **Piano → portata piano**
        - **Locazione**

L'abilitazione del controllo portate delle campate per una data area sarà possibile solamente se è stata definita una portata per ogni campata dell'area considerata. Viceversa, non sarà possibile abilitare il controllo. Anche nel momento in cui si dovesse aggiungere una nuova campata, se questa fa parte di un'area su cui è abilitato il controllo, il sistema controllerà che per questa dovrà essere necessariamente definita una portata.

L’abilitazione del controllo portate dei piani di una determinata campata è possibile solamente se è stata definita una portata per ogni piano e se è attivo il controllo portate della rispettiva campata. Viceversa, non sarà possibile abilitare il controllo. Anche nel momento in cui si dovesse aggiungere un nuovo piano, se questo fa parte di una campata su cui è abilitato il controllo, il sistema controllerà che per questo venga necessariamente definita una portata.

Abilitando il controllo portate piani per una data campata, è possibile inoltre specificare se controllare anche il piano terra (livello 0 o piano 0) oppure no: questo perché ci sono casi in cui, il piano terra è in realtà il pavimento stesso del magazzino, per cui non c’è bisogno di effettuare alcun controllo.

ATTENZIONE: nel caso in cui venga abilitato il controllo portate piani, il piano zero può comunque non avere una portata, in tal caso si intende che non deve essere effettuato alcun controllo su di esso.

La portata del piano, quindi, verrà considerata **solamente** se viene abilitato il controllo a livello di area (controllo portate campate) e di campata (controllo portate piani).

Di conseguenza gli scenari saranno i seguenti:

- Abilitazione del controllo di portata massima a livello unicamente di campata
- Abilitazione del controllo di portata massima sia a livello di piano che di campata (se definite le portate per i piani della campata)

NOTA: La gestione del doppio vincolo a livello di campata/piano consente una maggiore flessibilità:

- in fase di configurazione iniziale, ad esempio, si potrebbe voler gestire per maggiore semplicità il solo vincolo a livello di campata
- nel lungo termine, si potrebbe voler iniziare a gestire il controllo delle portate anche a livello di piano, e per facilitare il tutto, lo si può iniziare a fare campata per campata, senza dover eseguire tutto il lavoro in un’unica volta.

Per entrambi i controlli (campata e piano), inoltre, verrà preso in considerazione il peso delle UDC già stoccate (il comportamento è spiegato meglio nelle US).

Di seguito lo User Stories (US) definite per questo controllo.

US#1 - Configurazione portata piano

AS A	Configuratore del layout di magazzino
I WANT TO	Impostare una portata massima del piano
SO THAT	All'interno di un'area in cui sono attivi i controlli a livello di campata e di piani, vengano confrontati il peso dell'UDC da stoccare e la portata del piano.

US#2 - Configurazione portata campata

AS A	Configuratore del layout di magazzino
I WANT TO	Impostare una portata massima della campata
SO THAT	All'interno di un'area in cui è attivo il controllo a livello di campata, ed eventualmente ma non obbligatoriamente quello a livello di piani, vengano confrontati il peso dell'UDC da stoccare e la portata della campata

US#3 - Stoccaggio UDC pesata in aree con controllo portate campata abilitato

AS A	Addetto allo stoccaggio di una UDC con peso P_u in un magazzino con tutte le aree abilitate al solo controllo portate delle campate
I WANT TO	Il WMS mi proponga una locazione di una campata con portata P_c
SO THAT	P_u sommato al peso di tutte le UDC già stoccate nella campata sia inferiore di P_c

US#4 - Stoccaggio UDC pesata in aree con controllo portate campate e piani abilitato ma controllo piano a terra disabilitato

AS A Addetto allo stoccaggio di una UDC con peso P_u in un magazzino con tutte le aree abilitate al controllo portate campate con specifica sulle portate dei piani (escluso il piano a terra)

I WANT TO Il WMS mi proponga una locazione, di un piano (di livello superiore di quello a terra) con portata P_p oppure del piano a terra, appartenente alla campata con portata P_c

SO THAT P_u sommato al peso di tutte le UDC già stoccate nella campata sia inferiore di P_c e, per i piani superiori allo zero, P_u sommato al peso di tutte le UDC già stoccate nel piano sia inferiore a P_p

US#5 - Stoccaggio UDC pesata in aree con controllo portate campate e piani abilitato, e controllo piano a terra abilitato

AS A Addetto allo stoccaggio di una UDC con peso P_u in un magazzino con tutte le aree abilitate al controllo portate con specifica sulle portate di tutti i piani

I WANT TO Il WMS mi proponga una locazione, di un piano con portata P_p (di livello uguale o superiore di quello a terra) appartenente alla campata con portata P_c

SO THAT P_u sommato al peso di tutte le UDC già stoccate nella campata sia inferiore di P_c e P_u sommato al peso di tutte le UDC già stoccate nel piano sia inferiore a P_p

US#6 - Stoccaggio UDC pesata in aree con controllo portate abilitato/disabilitato

AS A Addetto allo stoccaggio di una UDC con peso P_u in un magazzino con alcune aree abilitate al controllo portate e altre no

I WANT TO Il WMS mi proponga una locazione appartenente ad un’area di magazzino

SO THAT La locazione proposta faccia parte di un’area abilitata al controllo e il peso P_u rispetti le regole di sicurezza (vedi altre US), oppure la locazione proposta faccia parte di un’area senza controllo portate

US#7 - Stoccaggio UDC non pesata

AS A	Addetto allo stoccaggio di una UDC senza peso specificato
I WANT TO	Il WMS mi escluda tutte le locazioni appartenenti ad aree in cui è abilitato il controllo portate
SO THAT	Evitare di stoccare l'UDC infrangendo le norme di sicurezza in termini di portate campate/piani ed isolare le UDC non pesate in specifiche aree di magazzino

6.2.5. Valutazione classi di rotazione

Come visto nel capitolo relativo allo *stoccaggio*[4], la *velocità di un articolo*[5], è una delle prime informazioni da raccogliere per migliorare l'efficienza di un magazzino. Per questo è stato deciso di aggiungere questa funzionalità all'algoritmo.

La velocità dell'articolo (chiamata d'ora in avanti **classe di rotazione**) verrà utilizzata in maniera molto semplice ma utile ed efficacie: si cercherà infatti di stoccare UDC con articoli di una determinata classe di rotazione, in locazioni che sono state configurate per quella determinata classe. In questa maniera, UDC che dovranno essere movimentate più spesso (classe A) saranno stoccate in locazione facilmente raggiungibili, e questo aumenterà l'efficienza del magazzino e faciliterà il lavoro da eseguire.

In questo caso, il controllo si deve poter attivare o non attivare senza dover prima definire alcun parametro a livello di area, campata, ecc. o di articolo: questo perché, se non viene definita alcuna classe di rotazione, sia livello di articolo che di locazione, verrà considerata una classe "indefinita" ed utilizzata normalmente come le altre (vedi US#3).

[4] Vedi capitolo ***Lo stoccaggio***
[5] Vedi capitolo ***Velocità dell'articolo***

Di seguito lo User Stories (US) definite per questo controllo.

US#1 - Configurazione classe di rotazione Articolo

AS A	Configuratore dell'anagrafica articoli
I WANT TO	Poter associare all'articolo una determinata classe di rotazione, tra quelle ammissibili (A/B/C)
SO THAT	Specificare per un determinato articolo a quale specifica classe di rotazione appartiene

US#2 - Configurazione classe di rotazione Locazione

AS A	Configuratore del layout di magazzino
I WANT TO	Poter associare alla locazione una determinata classe di rotazione, tra quelle ammissibili
SO THAT	Specificare per una determinata locazione a quale specifica classe di rotazione appartiene

US#3 - Stoccaggio articolo con classe di rotazione

AS A	Addetto allo stoccaggio di una UDC contenente l'articolo di classe X, definita tra l'insieme di valori finito A, B, C (in ordine di priorità decrescente)
I WANT TO	Che il sistema dia priorità prima di tutto alle locazioni in cui è definita la classe di rotazione X. Nel caso in cui non ve ne fossero di compatibili tra queste, voglio che il sistema cerchi tra le locazioni con classe di rotazione > X fino ad arrivare alle locazioni di classe A (upper bound). Nel caso in cui non si trovasse alcuna locazione compatibile, si esamineranno le locazioni con classe di rotazione < X fino ad arrivare alle locazioni di classe C (lower bound). Se anche questa navigazione non restituirà alcun risultato, si considereranno le locazioni senza classe di rotazione impostata.
SO THAT	Dare una priorità ad ogni locazione sulla base di X e della specifica classe associata all'ubicazione di magazzino per fare sì che, lo sforzo nel raggiungere la locazione in cui stocco l'articolo è inversamente proporzionale alla sua velocità di rotazione.

6.2.6. Valutazione tipo contenitore

Con questo controllo si vuole iniziare a tenere conto del tipo di package delle UDC al momento dello stoccaggio, come visto nel relativo capitolo[6]. In maniera particolare si vuole poter configurare quali tipi di contenitore sono stoccabili in una determinata locazione. Questo permette, per esempio, la creazione di aree logicamente divise non più solo in base ai tipi di articolo stoccati, ma anche in base al tipo di contenitore della rispettiva UDC.

Di seguito lo User Stories (US) definite per questo controllo.

US#1 – Assegnazione contenitori a piano

AS A	Configuratore del sistema
I WANT TO	Poter assegnare ad una locazione un determinato numero di contenitori
SO THAT	Solo UDC aventi come contenitore uno tra quelli impostati potranno esser stoccate all'interno della locazione

US#2 - Proposta di stoccaggio per UDC con contenitore dichiarata

AS A	Addetto allo stoccaggio di una UDC con contenitore X
I WANT TO	Il WMS mi proponga una locazione di un piano con contenitore X tra quelli ammessi oppure senza specifiche sui contenitori
SO THAT	Evitare lo stoccaggio dell'UDC in locazioni di piani con contenitori impostati differenti da quello considerato

US#3 - Proposta di stoccaggio per UDC senza contenitore dichiarato

AS A	Addetto allo stoccaggio di una UDC senza contenitore dichiarato
I WANT TO	Il WMS mi proponga una locazione di un piano senza specifiche sui contenitori
SO THAT	Evitare di stoccare UDC in locazioni di piani configurati per specifici contenitori

[6] Vedi capitolo ***Lo stoccaggio***

6.2.7. Valutazione zone geografiche

Come detto nella ***Raccolta dei requisiti,*** la valutazione delle zone geografiche è stata solo analizzata.

Il requisito chiedeva di prendere in considerazione la zona geografica del proprietario/destinatario dell'UDC al momento dello stoccaggio. Il motivo è che in determinati tipi di magazzini, può risultare utile aver stoccate tutte nella stessa zona, UDC che provenienza/destinazione geograficamente simile.

Per fare esempi più concreti e comprensibili è bene però dividere la valutazione della zona geografica del destinatario da quella del proprietario:

- Valutazione zona geografica del proprietario: questo è un caso di richiesta che è stata scartata perché ritenuta non utile. Stoccare, vicine tra loro, UDC provenienti dalla stessa zona geografiche non porta alcun vantaggio, almeno non nella realtà in cui questa tesi è stata sviluppata. Potrebbe aver senso invece stoccare insieme UDC dello stesso proprietario, ma questo aspetto non è stato portato avanti;
- Valutazione zona geografica del destinatario: questa richiesta invece è stata analizzata e ritenuta utile perché sono stati trovati casi reali di applicazione. Si pensi a dei magazzini di grossi fornitori: quando deve essere effettuata una spedizione verso una determinata località, è probabile che essa soddisfi ordini di diversi clienti, vicini geograficamente. Stoccare vicine tra loro UDC destinate alla stessa zona, consentirebbe di velocizzare la preparazione della spedizione.

Avrebbe quindi senso realizzare un controllo che permetta, in fase di stoccaggio, di valutare la zona geografica del destinatario: in questo modo UDC destinate alla stessa zona sarebbe stoccate vicine tra loro e velocizzerebbero la fase di preparazione delle spedizioni.

7. Progettazione dell’algoritmo per la proposta di stoccaggio V2

La progettazione dell’algoritmo è divisa in due macro-fasi:

1. Progettazione della struttura dell’algoritmo
2. Progettazione dei singoli filtri analizzati precedentemente.

Con **filtro** si intende un qualsiasi controllo/valutazione di quelli citati nell’***Analisi dei requisiti***: per esempio si parlerà, d’ora in avanti, di **filtro del peso (o delle portate)**.

Il nuovo algoritmo, quindi, sarà formato da un insieme di filtri che elimineranno pian piano, tutte le locazioni non adatte all’UDC da stoccare. Una volta finito di applicare i vari filtri attivi, si avrà un insieme di locazioni adatte all’UDC e ne verrà scelta una da proporre.

7.1. Progettazione della struttura

Il nuovo algoritmo avrà una struttura molto semplice, a cascata. Ogni filtro infatti è a sé stante e può essere attivato o meno: se è attivo restituisce le locazioni che lo rispettano, se non lo è si prosegue e si verifica il filtro successivo. Il flusso è mostrato in *Figura 10*.

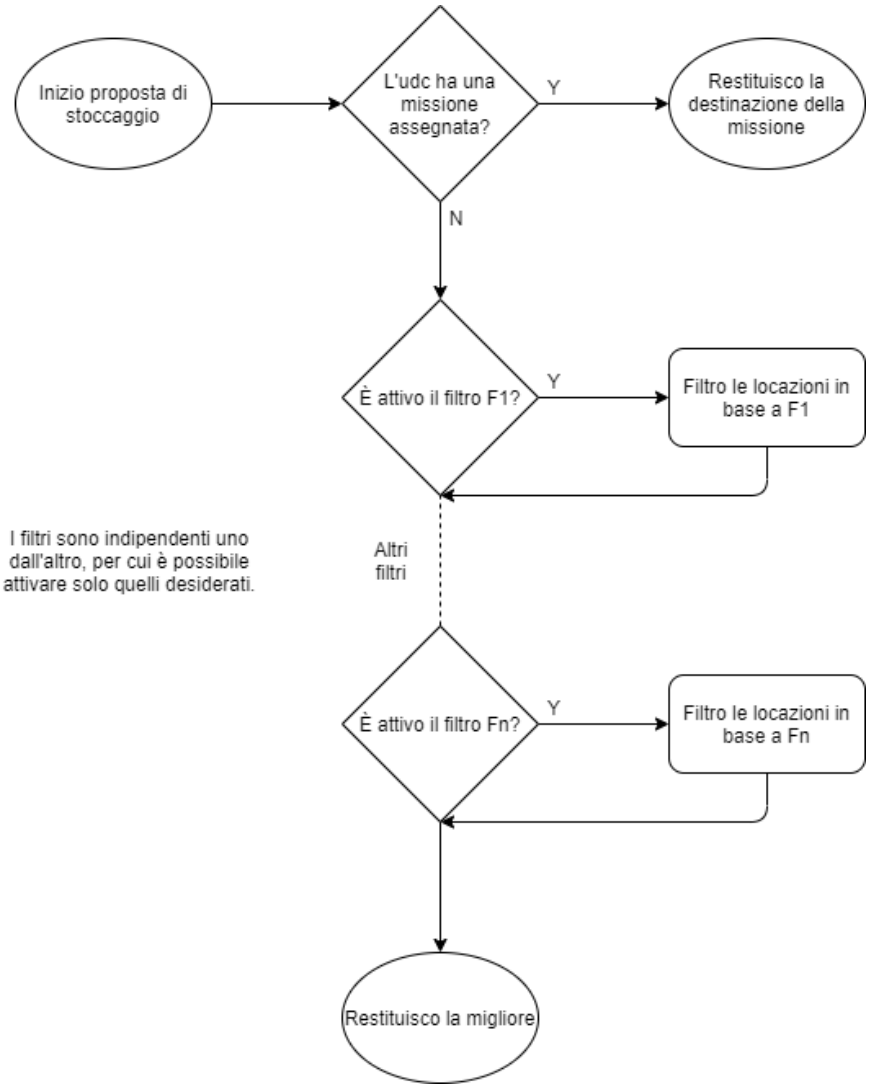


Figura 10: flusso algoritmo V2

Ogni filtro è un modulo indipendente e questo rappresenta un punto di forza: quando si vorrà aggiungere un nuovo filtro basterà creare solo la logica ad esso relativa e dire all'algoritmo di considerarlo. Non c'è bisogno di modificare altre parti dell'algoritmo.

Si può anche pensare alla struttura come ad un imbuto: ogni filtro che viene applicato riduce il numero di locazioni compatibili con l'UDC fino ad arrivare ad avere solo quelle che rispettano tutti i filtri attivati.

7.2. Progettazione dei filtri

In questa parte della progettazione si prenderanno in considerazione i singoli filtri analizzati e per ognuno verrà fornito dello pseudo-codice utile a capire come si vorrà risolvere il relativo problema.

I filtri progettati sono relativi ai seguenti requisiti:

- **Controllo altezze piani**
- **Controllo dimensioni**
- **Controllo peso**
- **6.2.5Valutazione classi di rotazione**
- **Valutazione tipo contenitore**

7.2.1. Progettazione filtro altezze piani

Come detto nella relativa analisi, questo filtro deve essere attivato a livello di area. Per far ciò si inserisce nell'area un flag, che indica se il controllo è attivo o no. In *Figura 11* è mostrato lo pseudo-codice di questo filtro.

```
1  FiltraPerAltezza()(udc, locazioni)
2  {
3      locazioniCompatibili = lista;
4
5      foreach(loc in locazioni)
6      {
7          if(udc.altezza.HasValue)
8          {
9              if(loc.Area.AbilitatoControlloAltezza)
10             {
11                 if(udc.altezza < loc.altezza)
12                     locazioniCompatibili.add(loc);
13             }
14             else
15                 locazioniCompatibili.add(loc);
16         }
17         else
18         {
19             if(!loc.Area.AbilitatoControlloAltezza)
20             {
21                 locazioniCompatibili.add(loc);
22             }
23         }
24
25     }
26     locazioni = locazioniCompatibili;
27 }
```

Figura 11: pseudo-codice filtro altezza piani

Come si può vedere il comportamento è molto semplice e rispetta quello indicato nelle User Story scritte nell'analisi.

Il primo controllo che si effettua è verificare se l'UDC ha l'altezza valorizzata (riga 7): se non ce l'ha vengono prese solo le locazioni di aree in cui il controllo dell'altezza non è attivo (righe 17-23).

Se l'altezza dell'UDC è valorizzata, si verifica allora se è attivo il controllo dell'altezza nell'area (riga 9): se lo è, si confrontano le altezze dell'UDC e della locazione (riga 11), e nel caso siano compatibili la locazione viene presa (riga 12). Se il controllo dell'altezza nell'area non è attivo, la locazione viene sempre presa (riga 15).

Alla fine, vengono salvate solo le locazioni compatibili (riga 26).

7.2.2. Progettazione filtro dimensioni

Questo filtro è logicamente diviso in più parti:

- Controllo altezza: viene applicato il filtro visto nel paragrafo precedente, tale quale (pseudo-codice in *Figura 12*);
- Controllo larghezza: il filtro è identico a quella dell'altezza tranne che per due fattori: il primo, è che non c'è un flag di controllo ma si verifica la presenza delle dimensioni, cioè se sono state configurate, il secondo, è che, ovviamente, viene controllata la larghezza e non l'altezza (pseudo-codice in *Figura 13*);
- Controllo profondità: in questo caso il filtro verifica se lo spazio rimanente in profondità, consente lo stoccaggio dell'UDC (pseudo-codice in *Figura 14*).

```
1   FiltraPerVolume(udc, locazioni)
2   {
3       locazioniCompatibili = locazioni;
4       tmp = lista;
5
6       foreach(loc in locazioniCompatibili)
7       {
8           if(udc.altezza.HasValue)
9           {
10              if(loc.Area.AbilitatoControlloAltezza)
11              {
12                  if(udc.altezza < loc.altezza)
13                      tmp.add(loc);
14              }
15              else
16                  tmp.add(loc);
17          }
18          else
19          {
20              if(!loc.Area.AbilitatoControlloAltezza)
21              {
22                  tmp.add(loc);
23              }
24          }
25      }
26
27      locazioniCompatibili = tmp;
28      tmp.clear();
```

Figura 12: pseudo-codice filtro dimensioni, controllo altezza

```
29
30      foreach(loc in locazioniCompatibili)
31      {
32          if(udc.larghezza.HasValue)
33          {
34              if(loc.larghezza.HasValue)
35              {
36                  if(udc.larghezza < loc.larghezza)
37                      tmp.add(loc);
38              }
39              else
40                  tmp.add(loc);
41          }
42          else
43          {
44              if(!loc.larghezza.HasValue)
45              {
46                  tmp.add(loc);
47              }
48          }
49      }
50
51      locazioniCompatibili = tmp;
52      tmp.clear();
53
```

Figura 13: pseudo-codice filtro dimensioni, controllo larghezza

```
54      locazioniConUdc = GetUdcStoccateFrom(locazioniCompatibili);
55
56      foreach(loc in locazioniCompatibili)
57      {
58          if(udc.profondita.HasValue)
59          {
60              if(loc.profondita.HasValue)
61              {
62                  profonditaRimanente = loc.profondita;
63                  if(locazioniConUdc.contains(loc))
64                  {
65                      foreach(u in locazioniConUdc.GetUdcFrom(loc))
66                      {
67                          profonditaRimanente -= u.profondita;
68                      }
69                  }
70
71                  if(profonditaRimanente >= udc.profondita)
72                      tmp.add(loc);
73              }
74              else
75                  tmp.add(loc);
76          }
77          else
78          {
79              if(!loc.profondita.HasValue)
80              {
81                  tmp.add(loc);
82              }
83          }
84      }
85      locazioni = tmp;
86  }
```

Figura 14: pseudo-codice filtro dimensioni, controllo profondità rimanente

Confrontando la *Figura 12* e la *Figura 13* si può osservare che il comportamento è identico, per cui nulla differisce da quello descritto nel paragrafo **7.2.1.**

La *Figura 14* mostra invece come viene gestita la profondità.

Innanzitutto, si recupera la giacenza (riga 54), cioè l'insieme delle UDC stoccate nelle locazioni da controllare.

Poi si controlla se l'UDC ha la profondità valorizzata oppure no (riga 58): se non ce l'ha vengono prese solo le locazioni senza profondità configurata (righe 77-83).

In caso contrario, si controlla la profondità della locazione: se non è valorizzata la locazione viene presa (riga 75). Se invece è valorizzata, viene recuperata (riga 62) e si verifica se nella locazione sono presenti UDC già stoccate (riga 63): se sono presenti, si calcola la profondità rimanente (righe 65-68). A questo punto si controlla se la profondità dell'UDC è compatibile con quella rimanente della locazione (riga 71), e in caso affermativo la locazione viene presa.

Alla fine, vengono salvate solo le locazioni compatibili (riga 85).

7.2.3. Progettazione filtro portate

Questo filtro è il più articolato, e come detto nella relativa analisi, saranno presenti abilitazioni a più livelli (quindi più flag), in maniera particolare:

- A livello di Area, sarà presente un flag che indica se è attivo o meno il controllo delle portate (in caso affermativo, vuol dire che sono state definite almeno le portate delle campate);
- A livello di Campata saranno presenti due flag:
    - Il primo che indica se è attivo o meno il controllo delle portate a livello di piano;
    - Il secondo (utilizzato solo se il primo è attivo) che indica se il piano terra deve essere incluso o meno nel controllo delle portate.

In *Figura 15* e in *Figura 16* viene mostrato lo pseudo-codice di questo filtro.

Anche in questo caso la prima cosa che viene controllata è se l'UDC ha un peso configurato oppure no (riga 6). Se non ce l'ha, vengono prese solo le locazioni appartenenti ad aree in cui non è attivo il controllo delle portate (righe 50-59, in *Figura 16*). Se l'UDC ha il peso valorizzato, la prima cosa che viene fatta è la raccolta di tutte le locazioni compatibili, cioè:

- Le locazioni di aree in cui non è attivo il controllo delle portate (riga 10);
- Le locazioni in cui il peso dell'UDC è compatibile con la portata della campata (riga 11);
- Le locazioni in cui il peso dell'UDC non è compatibile con la portata della campata, ma fanno parte di un piano terra in cui non è attivo il controllo della portata (riga 12).

Una volta raccolte le locazioni compatibili, viene recuperata la loro giacenza (riga 15), e vengono selezionate (riga 17) solo le locazioni che necessitano di ulteriori controlli (da quelle compatibili vengono tolte quelle in cui non bisogna controllare la portata, cioè quelle di piani terra senza portata e locazioni di aree senza controllo attivo).

Di queste locazioni vengono escluse quelle che non rispettano le portate delle campate o dei piani (righe 21-24), cioè locazioni in cui, aggiungendo il peso dell'UDC da stoccare, si sforerebbe la portata massima consentita. In ultimo viene effettuato il controllo delle portate tenendo conto della giacenza: prima si considera la giacenza e il peso sulla campata (righe 27-28) e si verifica se la locazione è da escludere o no (righe 29-33); poi, se attivo il controllo sui piani (riga 35), si effettuano gli stessi controlli sui piani (righe 37-43). Arrivati alla fine si avrà un insieme di locazioni da escludere da quelle compatibili (riga 48).

```
FiltraPerPeso(udc, locazioni)
{
    locazioniCompatibili = lista;
    locazioniDaEscludere = lista;

    if(udc.peso.HasValue)
    {
        foreach(loc in locazioni)
        {
            if(!loc.Area.AbilitatoControlloPeso ||
            (loc.Area.AbilitatoControlloPeso && loc.Campata.Portata >= udc.peso ||
            (loc.Campata.AbilitatoControlloPesoPiani && loc.Campata.IgnoraPianoTerra)))
                locazioniCompatibili.add(loc);
        }
        giacenzaPerLocazione = GetUdcFrom(locazioniCompatibili);

        locazioniDaControllare = GetLocazioniConControlliFrom(locazioniCompatibili);

        foreach(loc in locazioniDaControllare)
        {
            if(udc NON RISPETTA PORTATE CAMPATA/PIANO)
            {
                locazioniDaEscludere.add(loc);
            }
            else
            {
                giacenzaPerCamapata = GetGiacenzaCampataFrom(giacenzaPerLocazione, loc.Campata);
                pesoStoccatoCampata = giacenzaPerCamapata.sum(peso);
                if(pesoStoccatoCampata + udc.peso > loc.Campata.Portata)
                {
                    locazioniDaEscludere.add(loc);
                    continue foreach;
                }

                if(loc.Campata.AbilitatoControlloPesoPiani)
                {
                    giacenzaPerPiano = GetGiacenzaPianoFrom(giacenzaPerLocazione, loc.Piano);
                    pesoStoccatoPiano = giacenzaPerPiano.sum(peso);
                    if(pesoStoccatoPiano + udc.peso > loc.Piano.Portata)
                    {
                        locazioniDaEscludere.add(loc);
                        continue foreach;
                    }
                }
            }
        }

        locazioniCompatibili = locazioniCompatibili.exclude(locazioniDaEscludere);
    }
```

Figura 15: pseudo-codice filtro dimensioni, controllo portata, parte 1

```
    else
    {
        foreach(loc in locazioni)
        {
            if(!loc.Area.AbilitatoControlloPeso)
            {
                locazioniCompatibili.add(loc);
            }
        }
    }

    locazioni = locazioniCompatibili;
}
```

Figura 16: pseudo-codice filtro dimensioni, controllo portata, parte 2

7.2.4. Progettazione filtro classi di rotazione

Anche il filtro relativo alle classi di rotazione è piuttosto semplice e segue il comportamento definito nelle relative User Story.

In *Figura 17* è mostrato lo pseudo-codice di questo filtro.

```
1  FiltraPerClasseDiRotazione(udc, locazioni)
2  {
3      locazioniCompatibili = lista;
4
5      if(udc.ClasseRotazione != INDEFINITA)
6      {
7          foreach(loc in locazioni)
8          {
9              if(loc.ClasseRotazione == udc.ClasseRotazione)
10                 locazioniCompatibili.add(loc);
11         }
12
13         if(locazioniCompatibili IS EMPTY)
14         {
15             foreach(loc in locazioni)
16             {
17                 if(loc.ClasseRotazione < udc.ClasseRotazione)
18                     locazioniCompatibili.add(loc);
19             }
20             if(locazioniCompatibili IS EMPTY)
21             {
22                 foreach(loc in locazioni)
23                 {
24                     if(loc.ClasseRotazione > udc.ClasseRotazione)
25                         locazioniCompatibili.add(loc);
26                 }
27             }
28         }
29     }
30     else
31     {
32         foreach(loc in locazioni)
33         {
34             if(loc.ClasseRotazione == INDEFINITA)
35                 locazioniCompatibili.add(loc);
36         }
37     }
38
39     locazioni = locazioniCompatibili;
40 }
```

Figura 17: pseudo-codice filtro classe di rotazione

Il primo controllo effettuato è sulla presenza o meno di una classe di rotazione nell'UDC (riga 5): se non è presente (cioè è INDEFINITA), si cercano solo le locazioni per cui non è stata definita una classe di rotazione (righe 30-37). Se invece l'UDC ha una classe di rotazione, si cercano innanzitutto le locazioni con la stessa classe (righe 7-11); se non sono presenti (riga 13), si cercano le locazioni adibite allo

stoccaggio di articoli con classi superiori (righe 15-19); se neanche questa ricerca produce risultati (riga 20) si recuperano le locazioni adibite allo stoccaggio di articoli con classi inferiori (righe 22-26).

Alla fine, sono salvate solo le locazioni compatibili (riga 39).

7.2.5. Progettazione filtro tipo di contenitore

L'ultimo filtro è quello relativo al tipo di contenitore dell'UDC.

In *Figura 18* è mostrato lo pseudo-codice del filtro.

Per prima cosa si verifica se l'UDC ha un contenitore definito (riga 5): se non ne ha uno, si cercano solo le locazioni che non hanno specifiche sui contenitori ammessi (righe 15-19).

In caso contrario si cercano le locazioni che hanno, tra i contenitori ammessi, lo stesso dell'UDC, e le locazioni che non hanno specifiche sui contenitori ammessi (righe 7-11).

Alla fine, sono salvate solo le locazioni compatibili (riga 22).

```
1  FiltraPerTipoContenitori(udc, locazioni)
2  {
3      var locazioniCompatibili = lista;
4
5      if(udc.Contenitore != NON DEFINITO)
6      {
7          foreach(loc in locazioni)
8          {
9              if(loc.ContenitoriAmmessi.contains(udc.Contenitore) || loc.ContenitoriAmmessi IS EMPTY)
10                 locazioniCompatibili.add(loc);
11         }
12     }
13     else
14     {
15         foreach(loc in locazioni)
16         {
17             if(loc.ContenitoriAmmessi IS EMPTY)
18                 locazioniCompatibili.add(loc);
19         }
20     }
21
22     locazioni = locazioniCompatibili;
23 }
```

Figura 18: pseudo-codice filtro tipi di contenitore

8. Implementazione algoritmo per la proposta di stoccaggio V2

Come per la progettazione ci si concentrerà prima sullo sviluppo della struttura generale dell'algoritmo e poi su quella dei singoli filtri.

Come detto nel capitolo **3**, l'implementazione di ogni filtro è divisa in:

- Scrittura dei test
- Traduzione dello pseudo-codice in codice C#
- Testing del codice scritto, con relative correzioni

Il codice che verrà mostrato è quello finale a cui sono già state apportate le dovute correzioni. Alcune parti dei test e dei filtri fanno riferimento ad altre parti del WMS, che verranno brevemente spiegate ma non mostrate.

8.1. Implementazione della struttura

La prima parte implementata è una sorta di semplice configuratore, mostrato in *Figura 19.*

```
787     public static class CfgPropostaDiStoccaggioV2
788     {
            3 references | Andrea.Negri, 123 days ago | 2 authors, 2 changes
789         public static bool ConsideraPeso { get; private set; }
            24 references | 0/24 passing | Andrea.Negri, 123 days ago | 1 author, 1 change
790         public static void SetConsideraPeso(bool stato)
791         {
792             ConsideraPeso = stato;
793         }
794
            4 references | Andrea.Negri, 114 days ago | 1 author, 1 change
795         public static bool ConsideraDimensioni { get; private set; }
            12 references | 0/12 passing | Andrea.Negri, 114 days ago | 1 author, 1 change
796         public static void SetConsideraDimensioni(bool stato)...
800
            3 references | Andrea.Negri, 123 days ago | 1 author, 1 change
801         public static bool ConsideraAltezza { get; private set; }
            10 references | 0/10 passing | Lorenzo Vernocchi, 114 days ago | 2 authors, 2 changes
802         public static void SetConsideraAltezza(bool stato)...
806
            3 references | Andrea.Negri, 108 days ago | 1 author, 1 change
807         public static bool ConsideraClasseRotazione { get; set; }
            6 references | 0/6 passing | Andrea.Negri, 108 days ago | 1 author, 1 change
808         public static void SetConsideraClasseRotazione(bool stato)...
812
            3 references | Andrea Negri, 93 days ago | 1 author, 1 change
813         public static bool ConsideraTipoContenitore { get; set; }
            7 references | 0/7 passing | Andrea Negri, 93 days ago | 1 author, 1 change
814         public static void SetConsideraTipoContenitore(bool stato)...
818     }
819 }
820
```

Figura 19: configuratore della proposta di stoccaggio

Tramite esso è possibile scegliere quali controlli abilitare, utilizzando gli appositi metodi di configurazione, come quello mostrato nelle righe 790-793.

Queste informazioni vengono poi utilizzate nel metodo principale dell'algoritmo, come mostrato in *Figura 20* e *Figura 21*.

```
43          public LocazioneId ProponiLocazionePer(Udc udc, params int[] aree)
44          {
45              var destinazioniMissioni = DestinazioniMissioniDiStoccaggioPer(udc.Holding, new Udc[] { udc });
46              if (destinazioniMissioni[udc].Equals(LocazioneId.Sconosciuta) == false)
47              {
48                  return destinazioniMissioni[udc];
49              }
50
51              var dettagliUdc = GetDettagliUdcDaStoccare(udc.Holding, udc);
52
53              var locazioniQuery = GetLocazioniDisponibili(udc.Holding, aree);
54              if (locazioniQuery.IsEmpty())
55              {
56                  return new LocazioneId(LocazioneId.Sconosciuta.Area, LocazioneId.Sconosciuta.Locazione);
57              }
58
```

Figura 20: metodo principale della proposta di stoccaggio; recupero delle informazioni necessarie dell'UDC e delle locazioni

```
59
60              if (CfgPropostaDiStoccaggioV2.ConsideraTipoContenitore && !locazioniQuery.IsEmpty())
61              {
62                  locazioniQuery = locazioniQuery.FiltraPerTipoContenitori(dettagliUdc.First());
63              }
64
65              if (CfgPropostaDiStoccaggioV2.ConsideraClasseRotazione && !locazioniQuery.IsEmpty())
66              {
67                  locazioniQuery = locazioniQuery.FiltraPerClasseDiRotazione(dettagliUdc.First());
68              }
69
70              if (CfgPropostaDiStoccaggioV2.ConsideraDimensioni && !locazioniQuery.IsEmpty())
71              {
72                  locazioniQuery = locazioniQuery.FiltraPerDimensioni(dettagliUdc.First(), _context);
73              }
74
75              if (CfgPropostaDiStoccaggioV2.ConsideraAltezza && !CfgPropostaDiStoccaggioV2.ConsideraDimensioni
76                  && !locazioniQuery.IsEmpty())
77              {
78                  locazioniQuery = locazioniQuery.FiltraPerAltezza(dettagliUdc.First());
79              }
80
81              if (CfgPropostaDiStoccaggioV2.ConsideraPeso && !locazioniQuery.IsEmpty())
82              {
83                  locazioniQuery = locazioniQuery.FiltraPerPeso(dettagliUdc.First(), _context);
84              }
85
86
87              if (locazioniQuery.IsEmpty())
88              {
89                  return LocazioneId.Sconosciuta;
90              }
91
92              return locazioniQuery.Select(x => x.Locazione).FirstOrDefault();
93          }
```

Figura 21: metodo principale della proposta di stoccaggio; invocazione dei filtri attivati tramite il configuratore

In *Figura 20* viene mostrata la parte del metodo che recupera le informazioni relative a:

- Le missioni in corso (metodo *DestinatazioniMissioniDiStoccaggioPer* riga 45);
- I dettagli dell'UDC da stoccare (metodo *GetDettagliUdcDaStoccare* riga 51);
- I dettagli delle locazioni che dovranno essere filtrate (metodo *GetLocazioniDisponibili* riga 53).

Per prima cosa si verifica se l'UDC ha missioni assegnate (riga 46), in tal caso viene restituita la locazione di destinazione della missione (riga 48).

Altrimenti vengono recuperati i dati dell'UDC e delle locazioni. Se non ci sono locazioni disponibili viene restituito un valore di default, definito come "*locazione sconosciuta*" (riga 56).

Ma quali sono i dati dell'UDC e delle locazioni? Come vengono salvati?

Per entrambe, è stata creata un'apposita classe in cui vengono salvate le informazioni necessarie. Le classi sono:

- *UdcPropostaStockDto*: è la classe che raccoglie i dati dell'UDC. Il metodo *GetDettagliUdcDaStoccare* restituisce un oggetto di questo tipo;
- *LocazionePropostaStockDto*: è la classe che raccoglie i dati della locazione. il metodo *GetLocazioniDisponibili* restituisce un insieme di oggetti di questo tipo.

Il suffisso *Dto* sta per Data Transfer Object: le due classi, infatti, non sono le entità principali con le quali le UDC e le locazioni sono modellate e implementate nel WMS, ma servono a rappresentarle nella proposta di stoccaggio, trasferendo solo i dati necessari.

All'interno della classe *LocazionePropostaStockDto* sono presenti i DTO delle aree, delle campate e dei piani, che verranno poi spiegati insieme alla classe.

Vediamo ora le due classi.

In *Figura 22* viene mostrata la classe *UdcPropostaStockDto* che rappresenta un'UDC.

```
664     public class UdcPropostaStockDto
665     {
            2 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
666         public int AnnoUdc { get; set; }
            2 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
667         public int NumeroUdc { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
668         public int Livello { get; set; }
669
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
670         public string Articolo { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
671         public int Conformazione { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
672         public string Attributo { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
673         public string Lotto { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
674         public DateTime Scadenza { get; set; }
675
            7 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
676         public double? Peso { get; set; }
            3 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
677         public double? Larghezza { get; set; }
            4 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
678         public double? Profondita { get; set; }
            3 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
679         public double? Altezza { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
680         public string DimensioneUm { get; set; }
            1 reference | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
681         public string PesoUm { get; set; }
682
            5 references | Andrea.Negri, 108 days ago | 1 author, 1 change
683         public ClassiABC ClasseRotazioneABC { get; set; }
684
            3 references | Lorenzo Vernocchi, 136 days ago | 1 author, 1 change
685         public int? Contenitore { get; set; }
686     }
687
```

Figura 22: dati dell'UDC

Gli attributi *AnnoUdc* e *NumeroUdc* (righe 666-667) rappresentano l'identificativo dell'UDC.

Il *Livello* (riga 668) indica il numero di SKU all'interno dell'UDC: è possibile, infatti, avere UDC in cui sono presenti articoli diversi e che quindi hanno SKU diverse, in questo caso si parla di UDC multilivello. Lo stoccaggio di questo tipo di UDC segue solitamente regole diverse, per cui non è stato affrontato in questa tesi; questo attributo non è quindi utilizzato.

Gli attributi *Articolo, Conformazione, Attributo, Lotto* e *Scadenza* (righe 670-674), rappresentano la SKU presente nell'UDC.

Peso, Larghezza, Profondita e *Altezza* (righe 676-679) servono a salvare le informazioni sul peso e sulle dimensioni dell'UDC.

DimensioneUm e *PesoUm* sono le unità di misura utilizzate rispettivamente per le dimensioni e per il peso.

ClasseRotazioneABC rappresenta la classe di rotazione dell'articolo, e quindi dell'UDC, essendo monolivello. È stato creato un enum *ClassiABC* che contiene i valori: A, B, C e INDEFINITA.

Contenitore indica il codice del contenitore dell'UDC.

Tutte queste informazioni vengono recuperate da DB tramite il metodo *GetDettagliUdcDaStoccare* citato prima.

In *Figura 23* viene mostrato la classe *LocazionePropostaStockDto* che rappresenta una locazione.

```
690     public LocazionePropostaDiStockDto()...
        0 references | Lorenzo Vernocchi, 115 days ago | 2 authors, 2 changes
694     public LocazionePropostaDiStockDto(LocazioneId locazione,
695         PianoPropostaDiStockDto piano, CampataPropostaDiStockDto campata,
696         AreaPropostaDiStockDto area, double? larghezza, double? profondita)...
705
        7 references | Lorenzo Vernocchi, 137 days ago | 1 author, 1 change
706     public LocazioneId Locazione { get; set; }
707
        7 references | Andrea.Negri, 108 days ago | 1 author, 1 change
708     public ClassiABC ClasseRotazioneABC { get; set; }
709
        5 references | Andrea.Negri, 121 days ago | 1 author, 1 change
710     public double? Larghezza { get; set; }
711
        7 references | Andrea.Negri, 121 days ago | 1 author, 1 change
712     public double? Profondita { get; set; }
713
        19 references | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
714     public PianoPropostaDiStockDto PianoDettaglio { get; set; }
715
        12 references | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
716     public CampataPropostaDiStockDto CampataDettaglio { get; set; }
717
        8 references | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
718     public AreaPropostaDiStockDto AreaDettaglio { get; set; }
719
```

Figura 23: dati della locazione

Il costruttore (righe 694-704) inizializza semplicemente gli attributi presenti.

Locazione (riga 706) è un attributo di tipo *LocazioneId*. Per brevità non viene mostrato il codice di questa semplice classe (che non è stata creata per questo algoritmo, ma era già presente ed utilizzata nel WMS) che contiene al suo interno solo due attributi: *Area* un identificativo numerico dell'area della locazione, e *Locazione* un identificativo numerico della locazione.

ClasseRotazioneABC (riga 708) indica la classe di rotazione della locazione, cioè quali articoli dovranno esservi stoccati. È stato usato lo stesso enum definito per la classe *UdcPropostaStockDto.*

Larghezza e *Profondita* (righe 710-712) indicano le dimensioni della locazione.

PianoPropostaDiStockDto, *CampataPropostaDiStockDto e AreaPropostaDiStockDto*, sono i DTO rispettivamente del piano, della campata e dell'area della locazione.

In *Figura 24* viene mostrata la classe-DTO del piano.

```
720     public class PianoPropostaDiStockDto
721     {
            6 references | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
722         public long PianoId { get; set; }
            4 references | Andrea.Negri, 123 days ago | 1 author, 1 change
723         public int? Piano { get; set; }
724
            2 references | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
725         public bool IsPianoZero { get { return Piano == 0 || Piano == 1; } }
726
            5 references | Andrea.Negri, 123 days ago | 1 author, 1 change
727         public double? PesoMaxPiano { get; set; }
            3 references | Andrea.Negri, 123 days ago | 1 author, 1 change
728         public int? AltezzaMaxPiano { get; set; }
729
            7 references | Andrea Negri, 94 days ago | 1 author, 1 change
730         public IList<int> ContenitoriAmmessi { get; set; }
731
            0 references | Lorenzo Vernocchi, 115 days ago | 2 authors, 2 changes
732         public PianoPropostaDiStockDto(long codice, int? piano,
733             double? peso, int? altezza)...
740
            1 reference | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
741         public PianoPropostaDiStockDto()...
745
746     }
747
```

Figura 24: DTO del piano di una locazione

PianoId (riga 722) è l'identificativo numerico del piano.

Piano (riga 723) indica la posizione del piano nella campata.

IsPianoZero (riga 725) viene utilizzato per sapere se il piano in questione è il piano terra o no.

PesoMaxPiano (riga 727) indica la portata massima consentita sul piano, mentre *AltezzaMaxPiano* (riga 728) indica l'altezza massima consentita nel piano.

ContenitoriAmmessi (riga 730) è la lista dei codici identificativi dei contenitori ammessi nella locazione.

Il costruttore (riga 732-739) inizializza gli attributi della classe.

In *Figura 25* viene mostrata la classe-DTO della campata.

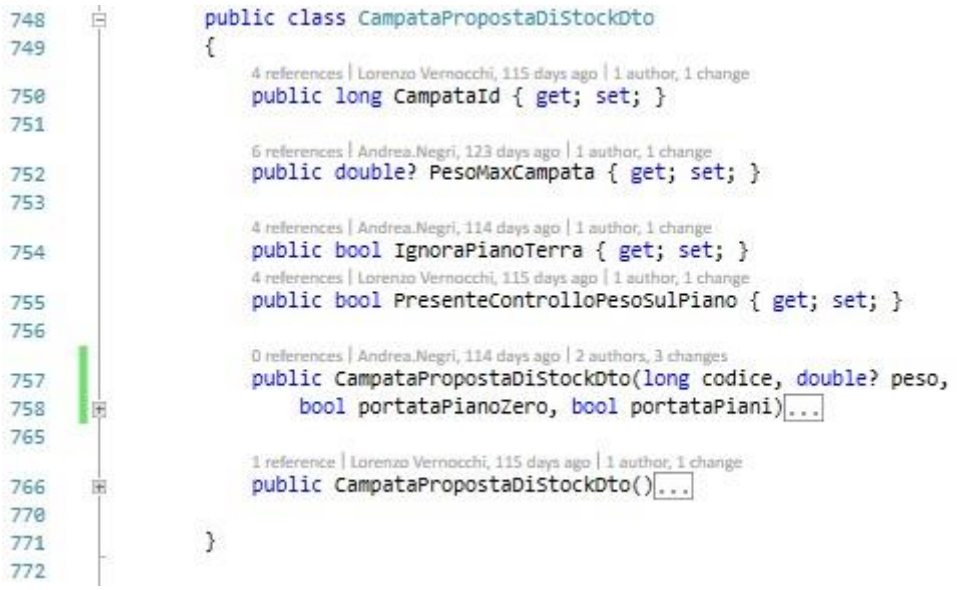


Figura 25: DTO della campata di una locazione

CampataId (riga 750) è l'identificativo numerico della campata.

PesoMaxCampata (riga 752) indica la portata massima consentita nella campata.

IgnoraPianoTerra (riga 754) indica se bisogna considerare anche il piano terra quando si controllano le portate oppure no (true = ignoralo, false = consideralo).

PresenteControlloPesoSulPiano (riga 755) indica se per i piani della campata è attivo o no il controllo delle portate (true = controllo attivo, false = controllo non attivo).

Il costruttore (riga 757-766) inizializza gli attributi della classe.

In *Figura 26* viene mostrata la classe-DTO dell'area.

Figura 26: DTO dell'area di una locazione

PresenteControlloPeso (riga 775) indica se per le campate dell'area è attivo o no il controllo delle portate (true = controllo attivo, false = controllo non attivo).

IsAbilitatoControlloAltezza (riga 776) indica se per i piani dell'area è attivo o no il controllo dell'altezza (true = controllo attivo, false = controllo non attivo).

Il costruttore (riga 778-782) inizializza gli attributi della classe.

Il codice fin qua osservato, permette di realizzare tutte le funzionalità previste dai filtri; andiamo ora a vedere quest'ultimi.

8.2. Implementazione dei filtri

In *Figura 21* è stata mostrata la parte dell'algoritmo in cui vengono invocati i filtri. Prima di vederli uno ad uno, vediamo la classe in cui sono implementati, mostrata in *Figura 27* e *Figura 28*.

```
407     public static class LocazionePropostaDiStockDtoExtensions
408     {
409         #region FILTRO PESO
410
                2 references | Andrea Negri, 17 days ago | 3 authors, 3 changes
411         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerPeso
412             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)...
509
510         #endregion
511
512         #region FILTRO DIMENSIONI & ALTEZZA
513
                2 references | Andrea.Negri, 109 days ago | 1 author, 2 changes
514         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerDimensioni
515             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)...
529
                1 reference | Andrea.Negri, 109 days ago | 2 authors, 3 changes
530         private static IQueryable<LocazionePropostaDiStockDto> FiltraPerLarghezza
531             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)...
544
                1 reference | Lorenzo Vernocchi, 115 days ago | 1 author, 1 change
545         private static IQueryable<LocazionePropostaDiStockDto> FiltraPerProfonditaRimanente
546             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)...
603
                3 references | Andrea.Negri, 109 days ago | 2 authors, 4 changes
604         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerAltezza
605             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)...
618
619         #endregion
```

Figura 27: filtri per il controllo delle portate e delle dimensioni

```
620
621         #region FILTRO CLASSE DI ROTAZIONE
622
                2 references | Andrea.Negri, 109 days ago | 1 author, 2 changes
623         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerClasseDiRotazione
624             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)...
647
648         #endregion
649
650         #region FILTRO TIPO DI CONTENITORI
651
                2 references | Andrea Negri, 94 days ago | 1 author, 1 change
652         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerTipoContenitori
653             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)...
667
668         #endregion
669     }
670
```

Figura 28: filtri per il controllo delle classi di rotazione e dei contenitori

Come si può osservare, tutti i filtri si aspettano in input le locazioni da filtrare (variabile *_locazioni* in tutte le signature) e l'UDC da stoccare (variabile *udc* in tutte le signature).

Il filtro per le portate e quello per la profondità utilizzano inoltre una variabile *_context*, tramite la quale è possibile accedere ai dati del DB, e quindi alla giacenza.

Per ogni filtro sono stati prima scritti dei test automatici, per verificarne la corretta implementazione. In *Figura 29* è mostrato un esempio: in questo caso si tratta di un test per il filtro sulle altezze dei piani.

```
1831        [TestMethod]
            0 references | Andrea.Negri, 115 days ago | 1 author, 3 changes
1832        public void TestAltezza01()
1833        {
1834            var container = TestContainerNewStockProposal.Create_Statefull_Container();
1835            var builder = container.Resolve<BuilderPropostaLocazioniNewStockProposal>();
1836
1837            var articolo = "ART1";
1838            var attributi = new Attributi(articolo, 1, "-", "123", DateTime.MaxValue, null);
1839
1840            builder.BuildMagazzino(_holding, new[] { 1 });
1841            builder.BuildArticoli(_holding, articolo);
1842
1843            var magazzino = container.Resolve<TestStrutturaMagazzinoNewStockProposal>();
1844            magazzino.DisabilitaStoccaggioPer2(_holding, 1, new int[] { 8 });
1845            magazzino.SetAbilitazioneControlloAltezzaPer(_holding, new int[] { 1 }, true);
1846
1847            for (int i = 1; i <= 8; i++)
1848            {
1849                magazzino.SetAltezzaPiano(_holding, i, 100);
1850            }
1851
1852            var giacenza = container.Resolve<GiacenzaTestServiceNewStockProposal>();
1853
1854            var udcDaStoccare = new Udc(_holding, 2020, 1);
1855
1856            giacenza.CreaUdc(new Posizione(1, 8), udcDaStoccare, attributi);
1857            giacenza.SetAltezzaUdc(udcDaStoccare, 50);
1858
1859            var proponiLocazioni = container.Resolve<ProponiLocazioniV2>();
1860            CfgPropostaDiStoccaggioV2.SetConsideraAltezza(true);
1861
1862            var locazioneUdc = proponiLocazioni.ProponiLocazionePer(udcDaStoccare);
1863
1864            Assert.AreEqual(1, locazioneUdc.Area);
1865            CollectionAssert.Contains(new int[] { 1, 2, 3, 4, 5, 6, 7 }, locazioneUdc.Locazione);
1866        }
1867
```

Figura 29: Test per il filtro dell'altezza

Dalla riga 1834 alla 1846 si prepara l'ambiente di test, creando un magazzino virtuale: viene creata un'area (riga 1840), un articolo (righe 1837-1838 e 1841) e il magazzino stesso (righe 1843-1845). Questi passi sono comuni a tutti i test.

Nelle righe 1847-1850 viene configurata una stessa altezza per tutti i piani dell'area, mentre nelle righe 1854-1857 viene creata l'UDC, e viene configurata la sua altezza.

Viene chiamato il configuratore nella riga 1860, e poi viene richiamato l'algoritmo nella riga 1862. Questa parte è comune a tutti i test, cambia solo la chiamata al configuratore, in cui si attivano filtri diversi in base a quale si sta testando.

Il corretto funzionamento dell'algoritmo viene verificato nelle righe 1864-1865, in cui ci si aspetta che venga proposta una delle locazioni compatibili con l'UDC (in questo caso una qualsiasi, perché sono tutte compatibili). Anche questa parte è comune a tutti i test, cambiano ovviamente i risultati aspettati in base al test.

In totale sono stati realizzati:

- 2 test per verificare il corretto funzionamento dell'algoritmo in presenza di missioni;
- 7 test per il filtro sulle altezze dei piani;
- 10 test per il filtro sulle dimensioni dell'UDC e delle locazioni;
- 21 test per il filtro sulle portate delle campate e dei piani;
- 4 test per il filtro sulle classi di rotazione;
- 5 test per il filtro sul tipo di contenitore.

8.2.1. Implementazione filtro altezze piani

In *Figura 30* è mostrato il codice del filtro.

```
604         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerAltezza
605             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)
606         {
607             if (!udc.Altezza.HasValue)
608             {
609                 _locazioni = _locazioni.Where(x => !x.AreaDettaglio.IsAbilitatoControlloAltezza);
610             }
611             else
612             {
613                 _locazioni = _locazioni.Where(x => !x.AreaDettaglio.IsAbilitatoControlloAltezza
614                 || x.PianoDettaglio.AltezzaMaxPiano > udc.Altezza);
615             }
616
617             return _locazioni;
618         }
619
```

Figura 30: Filtro altezza

Quello che viene fatto è verificare se l'UDC ha l'altezza valorizzata (riga 607):

- Se non ce l'ha, vengono prese solo le locazioni facenti parte delle aree senza il controllo dell'altezza attivo (riga 609)
- Se ce l'ha, vengono prese le locazioni di aree senza il controllo attivo (riga 613), e le locazioni che hanno un'altezza compatibile (riga 614).

In entrambi i casi si utilizza la clausola WHERE di LINQ per selezionare solo le locazioni desiderate.

8.2.2. Implementazione filtro dimensioni

In *Figura 31* è mostrato il codice del filtro.

```
514         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerDimensioni
515             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)
516         {
517             var locazioniCompatibili = _locazioni;
518
519             locazioniCompatibili = locazioniCompatibili.FiltraPerAltezza(udc);
520
521             locazioniCompatibili = locazioniCompatibili.FiltraPerLarghezza(udc);
522
523             locazioniCompatibili = locazioniCompatibili.FiltraPerProfonditaRimanente(udc, _context);
524
525             _locazioni = locazioniCompatibili;
526
527             return _locazioni;
528         }
529
```

Figura 31: Filtro dimensioni

Come si può osservare, il filtro richiama in realtà altri metodi, ognuno adibito al controllo di una singola dimensione. Alla fine, si avranno locazioni che rispettano tutte le dimensioni dell'UDC.

La prima chiamata (riga 519) effettuata è al filtro visto nel paragrafo precedente, quello per l'altezza; il comportamento è quindi identico.

La seconda chiamata (riga 521) effettuata è al filtro per la larghezza; il suo codice è mostrato in *Figura 32*.

La terza chiamata (riga 523) viene effettuata al filtro per la profondità rimanente; il suo codice è mostrato in *Figura 33* e *Figura 34*.

Alla fine, vengono restituite le sole locazioni che rispettano tutti e tre i filtri (riga 527).

Partiamo dal filtro per la larghezza.

```
530         private static IQueryable<LocazionePropostaDiStockDto> FiltraPerLarghezza
531             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)
532         {
533             if (udc.Larghezza.HasValue)
534             {
535                 _locazioni = _locazioni.Where(loc => !loc.Larghezza.HasValue || loc.Larghezza > udc.Larghezza);
536             }
537             else
538             {
539                 _locazioni = _locazioni.Where(loc => !loc.Larghezza.HasValue);
540             }
541
542             return _locazioni;
543         }
544
```

Figura 32: Filtro larghezza

Si verifica se l'UDC ha la larghezza valorizzata (riga 533):

- Se ce l'ha, vengono prese le locazioni senza larghezza valorizzata e quelle con larghezza compatibile (riga 535);
- Se non ce l'ha, vengono prese solo le locazioni senza larghezza valorizzata.

Anche in questo caso viene utilizzato LINQ per selezionare solo le locazioni desiderate.

Il filtro per la profondità è più complesso, per cui richiede più passaggi intermedi.

```
545         private static IQueryable<LocazionePropostaDiStockDto> FiltraPerProfonditaRimanente
546             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)
547         {
548             if (!udc.Profondita.HasValue)
549             {
550                 return _locazioni.Where(x => !x.Profondita.HasValue);
551             }
552 
553             _locazioni = _locazioni.Where(x => !x.Profondita.HasValue || (x.Profondita > udc.Profondita.Value));
554 
555             var locazioniAree = _locazioni.Select(x => x.Locazione);
556 
557             var predicateGiacenza = PredicateBuilder.False<V_UDC_GIACENZA>();
558             foreach (var area in locazioniAree.GroupBy(x => x.Area))
559             {
560                 var locazioniByArea = area.Select(x => x.Locazione).Distinct();
561 
562                 if (locazioniByArea != null && locazioniByArea.Any())
563                 {
564                     if (locazioniByArea.Count() == 1)
565                     {
566                         var locazioneSingola = locazioniByArea.Single();
567                         predicateGiacenza = predicateGiacenza.Or(x => x.U_CELLA == area.Key && x.U_LOC.Value == locazioneSingola);
568                     }
569                     else
570                     {
571                         predicateGiacenza = predicateGiacenza.Or(x => x.U_CELLA == area.Key && locazioniByArea.Contains(x.U_LOC.Value));
572                     }
573                 }
574             }
575 
```

Figura 33: Filtro profondità, raccolta delle locazioni

```
576             var profonditaOccupataPerLocazioni = _context.V_UDC_GIACENZA
577                 .Where(x => x.U_CELLA.HasValue && x.U_LOC.HasValue && x.U_LUNGHEZZA.HasValue).Where(predicateGiacenza)
578                 .GroupBy(x => new { Area = x.U_CELLA.Value, Locazione = x.U_LOC.Value })
579                 .Select(x => new
580                 {
581                     Area = x.Key.Area,
582                     Locazione = x.Key.Locazione,
583                     ProfonditaOccupata = x.Where(xx => xx.U_LUNGHEZZA.HasValue).Sum(xx => xx.U_LUNGHEZZA.Value),
584                 }).ToList();
585 
586             var locazioniDaEscludere = new List<LocazionePropostaDiStockDto>();
587             foreach (var loc in _locazioni.Where(x => x.Profondita.HasValue))
588             {
589                 var profonditaOccupataPerLocazione = profonditaOccupataPerLocazioni.Where(x => x.Area == loc.Locazione.Area
590                 && x.Locazione == loc.Locazione.Locazione).SingleOrDefault();
591                 if (profonditaOccupataPerLocazione != null)
592                 {
593                     if (profonditaOccupataPerLocazione.ProfonditaOccupata + udc.Profondita > loc.Profondita)
594                     {
595                         locazioniDaEscludere.Add(loc);
596                     }
597                 }
598             }
599 
600             _locazioni = _locazioni.Except(locazioniDaEscludere).AsQueryable();
601 
602             return _locazioni;
603         }
604 
```

Figura 34: Filtro profondità, selezione delle locazioni compatibili

Il primo controllo effettuato è sempre sulla presenza o meno di un valore, in questo caso, nella profondità dell'UDC (riga 548): se non è presente, si selezionano solo le locazioni per le quali non è stata configurata nessuna profondità.

In caso contrario, bisogna verificare quali locazioni hanno una profondità rimanente compatibile con l'UDC.

La prima cosa che viene fatta è recuperare tutte le locazioni in cui, senza prendere in considerazione la giacenza, l'UDC potrebbe entrare (riga 553).

A questo punto bisogna recuperare le UDC già stoccate. Per farlo bisogna effettuare un'interrogazione a DB, e per non appesantirla si recuperano solo le UDC stoccate nelle locazioni selezionate.

L'interrogazione viene effettuata sulla vista (non tabella) *V_UDC_GIACENZA* (riga 557) che contiene un insieme di informazioni sulle UDC che giacciono attualmente in magazzino; per effettuarla, però, bisogna specificare quali sono le locazioni da prendere in considerazione.

Per far questo viene costruito dinamicamente un predicato SQL tramite la classe *PredicateBuilder* (riga 557). Utilizzando il costruttore *False* si sta dicendo che le varie clausole WHERE che verranno aggiunte devono essere messe in OR tra loro.

Il predicato viene quindi costruito tramite il ciclo foreach delle righe 559-574:

- Si raggruppano le locazioni per area (riga 558);
- Si selezionano le locazioni dell'area attualmente selezionata (riga 560);
- Se è stata selezionata una sola locazione (riga 564), si aggiunge una clausola WHERE al predicato (riga 567) del tipo "area == area della locazione selezionata AND locazione = locazione selezionata";
- Se sono state selezionate più locazioni (riga 569), si aggiunge una clausola WHERE al predicato (riga 571) del tipo "area == area della locazione selezionata AND locazione IN {insieme delle locazioni selezionate}".

Alla fine, si avrà un unico predicato, di questo tipo: "(area == area prima clausola AND locazione == locazione prima clausola) OR (area == area seconda clausola AND locazione == locazione seconda clausola) OR (area == area prima clausola AND locazione IN {locazioni terza clausola}) OR …"

Per comprendere meglio come viene utilizzato, vediamo la query che viene effettuata nelle righe 576-584:

- Tramite *_context* si seleziona la vista sulla quale si vuole effettuare la query (riga 576);
- Si definiscono poi le clausole where (riga 577): le prime tre (*U_CELLA.HasValue, U_LOC.HasValue e U_LUNGHEZZA.HasValue*) servono a recuperare le UDC dalla giacenza che hanno area, locazione e profondità valorizzate. L'ultima inserisce il predicato costruito in precedenza.

In questa maniera, il where finale sarà costituito da un numero variabile di clausole, in base al numero di locazioni che devono essere controllate;

- Il risultato della query si raggruppa per area e locazione (riga 578). Ogni riga è quindi composta da questi due campi e da tutti i campi delle UDC stoccate nella locazione;
- Si selezionano solo le colonne desiderate (righe 579-583), cioè l'area (riga 581), la locazione (riga 582) e la profondità occupata (riga 583) che viene calcolata sommando tra loro le profondità delle UDC (riga 583, tramite l'utilizzo del metodo Sum di LINQ);
- Si costruisce una lista con le informazioni appena recuperate (riga 584).

Il risultato della query (una lista) viene salvato nella variabile *profonditaOccupataPerLocazioni.*

Raccolte le informazioni sulla giacenza, si cercano le locazioni da escludere da quelle recuperate all'inizio (riga 553), tramite il ciclo delle righe 587-598:

- Si recupera la profondità occupata nella locazione (righe 589-590) tramite *profonditaOccupataPerLocazioni*;
- Se il dato è presente (riga 591), vuol dire che ci sono UDC stoccate, e si verifica se sommando la profondità dell'UDC da stoccare a quella già occupata si supera la profondità totale (riga 593): se sì, la locazione viene esclusa (riga 595);
- Se il dato non è presente significa che non ci sono UDC stoccate nella locazione e che quindi è compatibile con l'UDC da stoccare.

Queste locazioni vengono quindi escluse perché non compatibili (riga 600) e vengono restituite quelle compatibili (riga 602).

8.2.3. Implementazione filtro portate

Il filtro per il controllo portate è logicamente diviso in tre parti:

- Recupero delle locazioni (*Figura 35*);
- Recupero della giacenza (*Figura 36*);
- Controllo delle portate (*Figura 37* e *Figura 38*).

```
411         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerPeso
412             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc, IOnWmsEntities _context)
413         {
414             var locazionidaEscludere = new List<LocazionePropostaDiStockDto>();
415
416             if (!udc.Peso.HasValue)
417             {
418                 return _locazioni.Where(x => !x.AreaDettaglio.PresenteControlloPeso);
419             }
420
421             _locazioni = _locazioni.Where(x => !x.AreaDettaglio.PresenteControlloPeso
422             || (x.AreaDettaglio.PresenteControlloPeso &&
423             (x.CampataDettaglio.PesoMaxCampata >= udc.Peso.Value ||
424             (x.CampataDettaglio.PresenteControlloPesoSulPiano && x.CampataDettaglio.IgnoraPianoTerra))));
425
426             var LocazioniCampate = _locazioni.Select(x => x.CampataDettaglio.CampataId);
427
```

Figura 35: Filtro portate, recupero locazioni

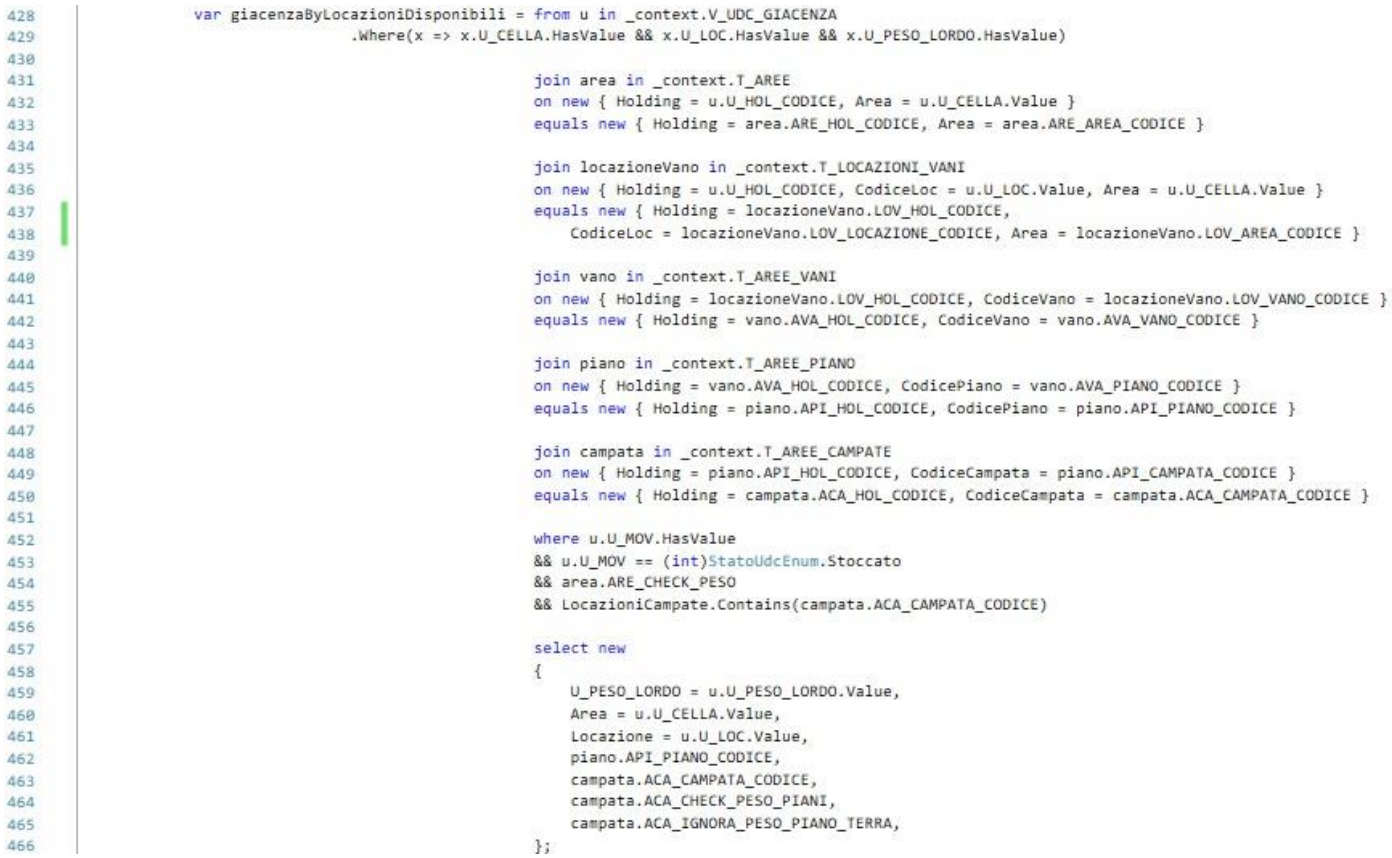
Per prima cosa si verifica se l'UDC ha il peso valorizzato (riga 416): se non ce l'ha vengono restituite le locazioni facenti parte di aree senza il controllo del peso abilitato.

Altrimenti si prosegue, e vengono recuperate tutte le locazioni compatibili con l'UDC, senza tenere conto della giacenza (righe 421-424):

- Si recuperano le locazioni di aree senza il controllo del peso abilitato (riga 421);
- Si recuperano le locazioni di aree in cui è presente il controllo del peso (riga 422) ed è rispettato uno di questi due vincoli:
    - Il peso dell'UDC è compatibile con la portata massima della campata (riga 423);
    - Il peso dell'UDC **non** è compatibile con la portata massima della campata ma la locazione in questione fa parte di un piano terra la cui portata non deve essere controllata (riga 424).

Dalle locazioni recuperate saranno poi escluse le locazioni che non rispettano i vincoli di portate che sono salvate nella lista creata all'inizio del filtro (riga 414).

Vengono quindi selezionati gli identificativi delle campate (riga 426) che serviranno per il recupero della giacenza.



```
428         var giacenzaByLocazioniDisponibili = from u in _context.V_UDC_GIACENZA
429                     .Where(x => x.U_CELLA.HasValue && x.U_LOC.HasValue && x.U_PESO_LORDO.HasValue)
430
431                                         join area in _context.T_AREE
432                                         on new { Holding = u.U_HOL_CODICE, Area = u.U_CELLA.Value }
433                                         equals new { Holding = area.ARE_HOL_CODICE, Area = area.ARE_AREA_CODICE }
434
435                                         join locazioneVano in _context.T_LOCAZIONI_VANI
436                                         on new { Holding = u.U_HOL_CODICE, CodiceLoc = u.U_LOC.Value, Area = u.U_CELLA.Value }
437                                         equals new { Holding = locazioneVano.LOV_HOL_CODICE,
438                                             CodiceLoc = locazioneVano.LOV_LOCAZIONE_CODICE, Area = locazioneVano.LOV_AREA_CODICE }
439
440                                         join vano in _context.T_AREE_VANI
441                                         on new { Holding = locazioneVano.LOV_HOL_CODICE, CodiceVano = locazioneVano.LOV_VANO_CODICE }
442                                         equals new { Holding = vano.AVA_HOL_CODICE, CodiceVano = vano.AVA_VANO_CODICE }
443
444                                         join piano in _context.T_AREE_PIANO
445                                         on new { Holding = vano.AVA_HOL_CODICE, CodicePiano = vano.AVA_PIANO_CODICE }
446                                         equals new { Holding = piano.API_HOL_CODICE, CodicePiano = piano.API_PIANO_CODICE }
447
448                                         join campata in _context.T_AREE_CAMPATE
449                                         on new { Holding = piano.API_HOL_CODICE, CodiceCampata = piano.API_CAMPATA_CODICE }
450                                         equals new { Holding = campata.ACA_HOL_CODICE, CodiceCampata = campata.ACA_CAMPATA_CODICE }
451
452                                         where u.U_MOV.HasValue
453                                         && u.U_MOV == (int)StatoUdcEnum.Stoccato
454                                         && area.ARE_CHECK_PESO
455                                         && LocazioniCampate.Contains(campata.ACA_CAMPATA_CODICE)
456
457                                         select new
458                                         {
459                                             U_PESO_LORDO = u.U_PESO_LORDO.Value,
460                                             Area = u.U_CELLA.Value,
461                                             Locazione = u.U_LOC.Value,
462                                             piano.API_PIANO_CODICE,
463                                             campata.ACA_CAMPATA_CODICE,
464                                             campata.ACA_CHECK_PESO_PIANI,
465                                             campata.ACA_IGNORA_PESO_PIANO_TERRA,
466                                         };
```

Figura 36: Filtro portate, query per il recupero della giacenza

Per recuperare la giacenza, come nel filtro per profondità, si effettua un'interrogazione sulla vista V_UDC_GIACENZA: in questo caso però bisogna effettuare numerose join che permettano di recuperare le informazioni relative al piano e alla campata in cui ogni UDC è stoccata.

Per prima cosa si cercano UDC in cui area, locazione e peso sono valorizzati (riga 429). Poi si effettuano le join necessarie (righe 431-450), tra la vista e le tabelle che rappresentano la struttura del magazzino (aree, piani e campate).

Si cercano solo le UDC che sono attualmente stoccate (righe 452-453 tramite la tabella *T_MOV* che indica lo stato dell'UDC. Si cercano UDC in stato "stoccato"), che fanno parte di aree in cui è attivo il controllo del peso (riga 454) e la cui campata compare tra nella lista di identificativi recuperati precedentemente (riga 455).

Di tutte le informazioni recuperate, si selezionano solo il peso (riga 459), l'area (riga 460), la locazione (riga 461), il piano (riga 462), la campata (riga 463), e i flag relativi al controllo della portata dei piani (riga 464) e al controllo della portata del piano terra (riga 465).

```
468             var locazioniCompatibiliConControlloGiacenza = _locazioni.Where(x => x.AreaDettaglio.PresenteControlloPeso
469             && (!x.PianoDettaglio.IsPianoZero || (x.PianoDettaglio.IsPianoZero && !x.CampataDettaglio.IgnoraPianoTerra)));
470
471             foreach (var locazione in locazioniCompatibiliConControlloGiacenza)
472             {
473                 if((locazione.PianoDettaglio.PesoMaxPiano.HasValue && locazione.PianoDettaglio.PesoMaxPiano.Value < udc.Peso)
474                     || (locazione.CampataDettaglio.PesoMaxCampata.HasValue && locazione.CampataDettaglio.PesoMaxCampata.Value < udc.Peso))
475                 {
476                     locazionidaEscludere.Add(locazione);
477                 }
```

Figura 37: Filtro portate, controllo portate, parte 1

Non occorre controllare tutte le locazioni recuperate a inizio metodo, ma solo quelle di un'area con controllo delle portate attivo (riga 468), e che non fanno parte di un piano terra (prima dell'OR nella riga 469) o che, anche essendo un piano terra, devono essere controllate (dopo l'OR nella riga 469).

Queste locazioni vengono controllate una alla volta, e la prima cosa che si verifica è se l'UDC rispetta i vincoli di portata del piano (riga 473) e della campata (riga 474): se uno dei due non è rispettato, la locazione viene aggiunta a quelle da escludere.

Altrimenti si prosegue.

```
478             else
479             {
480                 var giacenzaCampata = giacenzaByLocazioniDisponibili.Where(x => x.ACA_CAMPATA_CODICE == locazione.CampataDettaglio.CampataId);
481                 if(giacenzaCampata != null && giacenzaCampata.Any())
482                 {
483                     var pesoTotaleSuCampata = giacenzaCampata.Sum(x => x.U_PESO_LORDO);
484                     if (pesoTotaleSuCampata + udc.Peso > locazione.CampataDettaglio.PesoMaxCampata)
485                     {
486                         locazionidaEscludere.Add(locazione);
487                         continue;
488                     }
489                 }
490
491                 if (locazione.CampataDettaglio.PresenteControlloPesoSulPiano)
492                 {
493                     var giacenzaPiano = giacenzaByLocazioniDisponibili.Where(x => x.API_PIANO_CODICE == locazione.PianoDettaglio.PianoId);
494                     if(giacenzaPiano != null && giacenzaPiano.Any())
495                     {
496                         var pesoTotaleSuPiano = giacenzaPiano.Sum(x => x.U_PESO_LORDO);
497                         if (pesoTotaleSuPiano + udc.Peso > locazione.PianoDettaglio.PesoMaxPiano)
498                         {
499                             locazionidaEscludere.Add(locazione);
500                             continue;
501                         }
502                     }
503                 }
504             }
505         }
506
507         _locazioni = _locazioni.Except(locazionidaEscludere).AsQueryable();
508
509         return _locazioni;
510     }
```

Figura 38: Filtro portate, controllo portate, parte 2

Se la locazione rispetta i vincoli di portata del piano e dalla campata, bisogna verificare se li rispetta anche tenendo conto della giacenza.

Per cui si recupera la giacenza della campata (riga 480) e, se esiste (riga 481), si calcola il peso attualmente stoccato in essa (riga 483): se aggiungendo il peso dell'UDC a quello già stoccato si supera la portata massima (riga 484), la locazione viene esclusa (riga 486) e si prosegue verificando la prossima locazione (riga 487).

Se la portata massima della campata è rispettata, si prosegue e si guarda se è presente il controllo della portata del piano (riga 491): se non è presente la locazione viene considerata valida e si passa alla successiva. Se invece bisogna controllare il piano, effettuano gli stessi passaggi effettuati per la campata: si recupera la giacenza del piano (riga 493) e, se esiste (riga 494), si calcola il peso attualmente stoccato in esso (riga 486): se aggiungendo il peso dell'UDC a quello già stoccato si supera la portata massima (riga 497), la locazione viene esclusa (riga 499) e si prosegue verificando la prossima locazione (riga 500).

Vengono quindi escluse le locazioni non compatibili (riga 507) e restituite le compatibili (riga 509).

8.2.4. Implementazione filtro classi di rotazione

In *Figura 39* è mostrato il codice del filtro.

```
627         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerClasseDiRotazione
628             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)
629         {
630             var locs = _locazioni;
631
632             if(!udc.ClasseRotazioneABC.Equals(ClassiABC.INDEFINITA))
633             {
634                 locs = _locazioni.Where(l => l.ClasseRotazioneABC == udc.ClasseRotazioneABC);
635                 if (!locs.Any())
636                 {
637                     locs = _locazioni.Where(l =>
638                     (int)l.ClasseRotazioneABC < (int)udc.ClasseRotazioneABC).OrderByDescending(l => l.ClasseRotazioneABC);
639                     if (!locs.Any())
640                     {
641                         locs = _locazioni.Where(l =>
642                         (int)l.ClasseRotazioneABC > (int)udc.ClasseRotazioneABC).OrderBy(l => l.ClasseRotazioneABC);
643                     }
644                 }
645             }
646             else
647             {
648                 locs = _locazioni.Where(l => l.ClasseRotazioneABC.Equals(ClassiABC.INDEFINITA));
649             }
650
651             return locs;
652         }
653
```

Figura 39: Filtro classi di rotazione

Il primo controllo è sul tipo di classe dell'UDC: se non è definito (riga 632), vengono recuperate solo le locazioni che non sono associate ad alcuna classe di rotazione (riga 648).

Se invece è presente una classe di rotazione, si cercano le locazioni con la stessa classe (riga 634).

Se non sono state trovate (riga 635), si cercano locazioni associate a classi superiori (riga 638, la classe A ha valore 1, INDEFINITA ha valore 4).

Se neanche così ne sono state trovate (riga 639), si cercano le locazioni di classe inferiore (riga 642).

Infine, vengono restituite le locazioni trovate (riga 651).

8.2.5. Implementazione filtro tipo di contenitore

In *Figura 40* è mostrato il codice del filtro.

```
658         public static IQueryable<LocazionePropostaDiStockDto> FiltraPerTipoContenitori
659             (this IQueryable<LocazionePropostaDiStockDto> _locazioni, UdcPropostaStockDto udc)
660         {
661             if(!udc.Contenitore.HasValue)
662             {
663                 _locazioni = _locazioni.Where(l => (l.PianoDettaglio.ContenitoriAmmessi == null
664                 || l.PianoDettaglio.ContenitoriAmmessi.Count == 0));
665             }
666             else
667             {
668                 _locazioni = _locazioni.Where(l => (l.PianoDettaglio.ContenitoriAmmessi != null
669                 && l.PianoDettaglio.ContenitoriAmmessi.Contains(udc.Contenitore.Value))
670                 || l.PianoDettaglio.ContenitoriAmmessi == null || l.PianoDettaglio.ContenitoriAmmessi.Count == 0);
671             }
672 
673             return _locazioni;
674         }
675 
676         #endregion
677     }
```

Figura 40: Filtro tipo di contenitore

Per prima cosa si verifica se l'UDC ha un contenitore definito o no (riga 661):

- Se ce l'ha (righe 668-670) vengono selezionate le locazioni in cui sono stati definiti dei contenitori ammessi (riga 668) e tra essi è presente quello dell'UDC (riga 669), e le locazioni in cui non sono stati definiti contenitori (riga 670);
- Se non ce l'ha (righe 663-664) vengono selezionate le locazioni in cui non sono stati definiti contenitori (righe 663-664).

9. Sviluppi futuri

Con questo capitolo finale si vogliono presentare e discutere brevemente alcuni possibili sviluppi futuri dell'algoritmo, utili a migliorarlo e ad aumentarne le funzionalità, rendendolo più completo ed efficace.

Come visto nella progettazione e nell'implementazione, le locazioni vengono filtrate ed escluse in base ai filtri attivi: questo vuol dire che arrivati alla fine, si avrà un insieme di locazioni tutte compatibili con l'UDC da stoccare. Come scegliere quale proporre? L'algoritmo, attualmente, non risponde a questa a questa esigenza, ma propone semplicemente la prima locazione in elenco.

Un possibile sviluppo futuro è quindi quello che vede lo studio e l'implementazione di una funzionalità in grado di proporre la miglior locazione tra quelle compatibili, basandosi su determinati criteri, scelti, per esempio, dall'utente/cliente.

Ogni nuova esigenza in termini di controlli/filtri rappresenta un possibile sviluppo futuro:

- La valutazione delle zone geografiche (che è anche stata oggetto di analisi;
- Tenere conto della cronologia di stoccaggio: se un articolo, in passato, è sempre stato stoccato in determinate locazioni, può essere utile tenere in considerazione queste informazioni, per proporre una di queste locazioni (se compatibile);
- Utilizzare delle locazioni statiche per determinati articoli;
- Migliorare l'attuale controllo delle dimensioni, dando la possibilità di tenere conto anche di altri fattori (larghezza occupata per esempio, o come sono stoccate le UDC in locazione).

Ogni nuova esigenza può essere studiata e implementata senza dover alterare lo scheletro già esistente: basterà infatti aggiungere un nuovo metodo per ogni nuovo controllo che sarà poi richiamato dall'algoritmo. Si è quindi più aperti a nuovi sviluppi.

10. Conclusioni

Questa tesi mi ha permesso di mettere in pratica, per la prima volta nel mondo del lavoro, le capacità acquisite durante gli ultimi 8 anni di studi (superiori e università). Ho affrontato le fasi di analisi, progettazione e implementazione, utilizzando una metodologia di sviluppo iterativa che mi ha consentito di portare a termine un progetto così complesso.

Ho avuto l'opportunità di affrontare un problema interessante ma di cui non avevo praticamente alcuna conoscenza: questo mi ha consentito di apprendere non solo nuove informazioni, ma anche come recuperarle e come studiarle e analizzarle per poi applicarle alla mia situazione.

Una volta studiato il problema ho potuto comprendere quanto sia importante progettare bene la soluzione prima di implementarla, perché questo comporta una scrittura del codice più semplice e fluida. Inoltre, consente di avere del codice facilmente mantenibile e leggibile.

Altro insegnamento ed esperienza fondamentale che ho potuto apprendere è stata l'importanza dei test nello sviluppo. Scriverli prima di iniziare a sviluppare ha portato due grandi benefici: una maggior facilità nello scrivere il codice, avendo già chiaro in mente cosa avrebbe dovuto fare, e una maggior semplicità nel testare e correggere il codice scritto.

Oltre a migliorare le mie capacità implementative, ho avuto l'occasione di migliorare anche quelle di esposizione: per ogni fase affrontata ho dovuto condividere i risultati ottenuti con uno dei due team manager, e alla fine del progetto ho dovuto esporre il risultato finale al responsabile d'area, nonché correlatore della tesi, Claudio Gambetti.

Alla fine di questa tesi mi ritengo soddisfatto del risultato ottenuto, ma mi sento di poter dire che qualcosa in più potevo fare.

Ho imparato tanto, e molto di più ancora ho da imparare sull'infinito mondo dell'informatica.

11. Ringraziamenti

Vorrei ringraziare tutti coloro che mi hanno consentito di conseguire questo traguardo della mia vita.

Ringrazio Onit Group per avermi dato la possibilità di lavorare con loro.

Ringrazio il professore e relatore Vittorio Maniezzo, che ha accettato la mia proposta di tesi e mi ha consigliato e seguito durante tutto il suo sviluppo.

Ringrazio la mia famiglia, che mi è stata vicina durante tutti questi anni, sostenendomi durante i momenti più difficili e dandomi la forza di arrivare fino in fondo.

Ringrazio i due team manager, con cui ho avuto l'opportunità e il privilegio di lavorare, che mi hanno guidato fin da subito, e mi hanno fatto crescere professionalmente e personalmente: Simone Castorri e Lorenzo Vernocchi.

Infine, un grande ringraziamento va ai miei 4 compagni "d'avventura", con cui ho affrontato superiori e università, che hanno reso il percorso universitario meno ostico e più leggero, spensierato e perché no, divertente: Andrea Baldoni, Enrico Collina, Matteo Lucchi e Andrea Scarpellini.

12. Bibliografia e Sitografia

Bowles, R. (2020, Maggio 13). *The Beginner's Guide to Directed Putaway Algorithms*. Tratto il giorno Ottobre 7, 2020 da Sito Logiwa: https://www.logiwa.com/blog/directed-putaway-algorithm-warehouse#1

Cyzerg. (2020, Luglio 20). *Warehouse Operations: Optimizing the Putaway Process*. Tratto il giorno Ottobre 09, 2020 da Sito Cyzerg: https://articles.cyzerg.com/putaway-process-optimization-warehouse-operations

Everything Warehouse. (s.d.). *10 really easy tips to optimize your warehouse put away process*. Tratto il giorno Ottobre 7, 2020 da Sito Everything Warehouse: https://www.everythingwarehouse.net/optimize-warehouse-put-away-process/

Hatlevik, K. (2011, Aprle 07). *Put-away Concepts*. Tratto il giorno Ottobre 08, 2020 da Kurt Hatlevik – Dynamics 365 Blog: https://kurthatlevik.com/2011/04/07/put-away-concepts-%E2%80%93-part-i/

Koton, J. (2013). IJATES^2 Vol. 2, No. 3. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*. doi:10.11601/ijates.v2i3.80

Logipack. (s.d.). *Analisi ABC*. Tratto il giorno Ottobre 10, 2020 da Sito Logipack: http://www.logipack.it/PDF/08_ABC.pdf

Logistica Efficiente (Analisi ABC scorte/fatturato). (s.d.). *L'analisi ABC scorte/fatturato: definizione ed applicazione al mondo retail*. Tratto il giorno Ottobre 10, 2020 da Sito Logistica Efficiente: https://www.logisticaefficiente.it/redazione/supply-chain/scorte/analisi-abc-scorte-fatturato.html

Logistica Efficiente (Analisi di Pareto). (s.d.). *ANALISI DI PARETO*. Tratto il giorno Ottobre 10, 2020 da Sito Logistica Efficiente: https://www.logisticaefficiente.it/wiki-logistica/supply-chain/analisi-pareto.html

Mecalux. (2020, Settembre 16). *Cos'è il magazzino?* Tratto il giorno Settermbre 16, 2020 da Sito web MECALUX: https://www.mecalux.it/manuale-logistica-magazzino/magazzino

TRECCANI. (2020). *Enciclopedi TRECCANI*. Tratto il giorno Settembre 16, 2020 da TRECCANI: https://www.treccani.it/enciclopedia/logistica/