Politecnico di Torino

Dipartimento di AUTOMATICA E INFORMATICA

Tesi di Laurea Magistrale

Web-app per il monitoraggio e la gestione di un grande magazzino

Relatore
prof. Morisio Maurizio

Candidato
Ferrero Cristian

10 settembre 2020

Indice

Elenco delle figure

Ringraziamenti

Prima di iniziare a presentare la tesi vorrei dedicare qualche frase alle persone che hanno permesso la realizzazione di questo splendido percorso e che mi sono state vicine in questi anni.

Un sentito grazie va al mio relatore Maurizio Morisio, il quale oltre che ad avermi dato parecchi consigli nella realizzazione della tesi è anche riuscito in un anno di corsi a trasmettermi la passione per l'ingegneria del software.

Ringrazio molto l'azienda Orbyta e tutti i suoi dipendenti, in particolare gli sviluppatori dell'Area 51 che in questi mesi sono sempre stati disponibili a venire incontro a qualunque esigenza e hanno contribuito sia alla crescita personale che a quella del progetto. Un particolare ringraziamento va a Mirko e a Karina che mi hanno seguito per tutta la durata del percorso della tesi lavorando per la corretta realizzazione di questo progetto.

Senza il supporto da parte della mia famiglia non sarebbe stato possibile intraprendere questo percorso, per questo motivo i miei più sentiti ringraziamenti vanno a mia mamma Mara, mio fratello Nicholas, mia nonna Piera e a tutto il resto della mia famiglia che mi ha sempre sostenuto nel bene e nel male e che non mi ha mai fatto mancare nulla.

Un doveroso ringraziamento va al mio collega Paride che in tutti gli anni in cui ho frequentato l'università è sempre stato presente ogni giorno in ogni momento, senza di lui non sarei arrivato fino a qui.

Infine voglio ringraziare tutti i miei amici, colleghi e tutte le persone che mi sono state vicine nel mio percorso accademico, sia quelle presenti ancora oggi che quelle di cui non ho più notizie, perché mi hanno portato dove sono ora. Un caro ringraziamento va ai miei amici Alberto, Mattia, Daniele e Flavio che hanno condiviso con me in tutti questi anni le gioie di questo percorso.

Colgo l'occasione per porgere i miei ringraziamenti anche a tutti i docenti del Politecnico di Torino che hanno contribuito negli anni ad accrescere la passione in questo settore attraverso le nozioni e le conoscenze condivise in aula.

1 Introduzione

La tesi svolta tratta lo sviluppo di una web-app full stack utilizzando metodologie agili riguardante la gestione ed il monitoraggio di un grande magazzino che tratta parti di ricambio in ambito automotive. Questo progetto è stato effettuato presso l'azienda Orbyta Tech srl di Torino.

L'accettazione di questa tesi è dovuta alla passione che è nata frequentando i corsi di programmazione distribuita e ingegneria del software presso il Politecnico di Torino e alla presa di coscienza della potenzialità enorme che questa parte dell'informatica possiede nel mondo digitale oggigiorno. Oltre a questa motivazione vi è la voglia di imparare a sviluppare con linguaggi differenti non trattati nel corso di studi, possibilità che è stata offerta grazie a questa scelta.

L'obiettivo del progetto è quello di migliorare la gestione attuale di un magazzino proponendo una soluzione basata sulle nuove tecnologie e potenzialità che il mondo del web offre attualmente. Si è deciso di affiancare in maniera più fedele possibile le tecnologie già presenti per lo svolgimento di questo compito, proponendone migliorie e cercando di fornire strumenti nuovi in modo da ottimizzare anche la gestione del magazzino e facilitare l'operato del personale.

Il progetto prevede lo sviluppo di un'applicazione full-stack con metodologie agili, si è quindi deciso di utilizzare strumenti che facilitassero la condivisione del codice e dei documenti. Inoltre sono stati organizzati dei meeting durante tutto il periodo della durata della tesi, in modo da strutturare il lavoro nel modo corretto e trarre tutti i vantaggi che una metodologia agile può offrire.

Nella prima parte dell'elaborato è possibile venire a conoscenza dei risultati degli studi effettuati antecedentemente allo sviluppo della tesi, per poter migliorare il più possibile le funzionalità offerte dall'applicazione e venire incontro alle necessità del cliente finale.Gli studi svolti riguardano in primo luogo le politiche di gestione dei magazzini con i relativi processi e secondariamente le tecnologie utilizzabili per lo sviluppo di un'applicazione in ambito mobile e desktop.

Successivamente verranno mostrati i requisiti che l'applicazione deve soddisfare e le necessità espresse dal cliente. In particolar modo verranno presentate sia la situazione corrente che quella futura, in modo da poter comprendere i benefici introdotti con lo sviluppo di questo progetto e i risultati ottenuti.

Proseguendo verrà dedicata una sezione all'intero sviluppo dell'applicazione avvenuto in questi mesi fornendo dettagli tecnici maggiori. La parte in questione sarà corposa dovendo spiegare le diverse logiche e tecnologie utilizzate.

Prima delle conclusioni sarà dedicato un capitolo relativo all'introduzione del data mining in questo progetto. Sarà possibile vedere l'applicazione di alcuni algoritmi che estrapoleranno delle regole di associazione che il cliente potrà utilizzare per effettuare degli sconti o per posizionare la merce all'interno del magazzino, in modo ridurre i tempi necessari per smaltire gli ordini.

2 Gestione dei magazzini

La gestione di un magazzino comprende diversi aspetti, che si possono analizzare in dettaglio per poter beneficiare al massimo della produttività e del guadagno, oltre a garantire un fluido avanzamento dell'attività, minimizzando la possibilità di errori e perdite.

Uno dei fattori chiave è l'ottimizzazione dei processi che vengono attuati dal momento della ricezione della merce all'istante in cui si consegna il prodotto al cliente che ha effettuato l'acquisto.

La corretta gestione dell'inventario all'interno di un magazzino è un punto fondamentale per garantire un corretto operato e per organizzare in modo efficiente l'afflusso e il deflusso delle merci.

2.1 Processi primari

Per poter massimizzare la produzione e gestire al meglio tutte le risorse è necessario conoscere nel dettaglio questi processi e cercare di ottimizzarli il più possibile. Una corretta gestione di queste attività può portare, oltre ai benefici citati sopra, anche un risparmio di tempo e denaro da parte di chi gestisce il magazzino.

Prendendo in considerazione il ciclo di vita delle merci nel periodo di tempo in cui si viene a contatto con il magazzino è possibile identificare i seguenti processi [1]:

- Ricezione
- Stoccaggio
- Immagazzinamento
- Prelevamento
- Imballaggio
- Spedizione

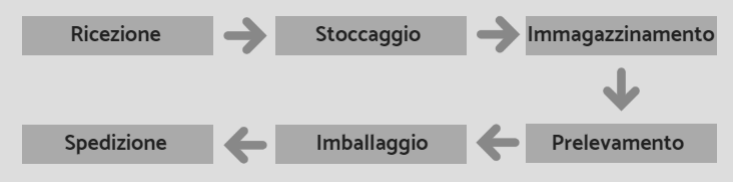


Figura 1: Processi Primari

Ciascuno dei processi identificati è essenziale nella corretta gestione di un magazzino, e può essere svolto in maniera differente a seconda di diversi fattori quali il tipo di magazzino, la posizione, la dimensione, la disponibilità di mano d'opera o macchinari e molti altri.

L'importanza di conoscere la natura di questi processi e la loro utilità contribuirà ad effettuar la scelta migliore per massimizzare l'efficienza di essi e il loro corretto funzionamento.

2.1.1 Processo di ricezione

Il processo di ricezione è il primo che viene messo in atto nel momento in cui le merci arrivano al magazzino, e include anche il modo e il periodo in cui vengono consegnate dai fornitori [2].

Uno dei problemi più importanti da risolvere in questa fase è l'ottimizzazione delle risorse allocate per il corretto svolgimento di esso, in modo da non assegnarne un errato numero, per questa ragione è consigliato avvalersi di sistemi di prenotazione digitali in modo da tracciare le consegne e organizzarsi nei tempi corretti per la ricezione.

Un aspetto importante da considerare è che all'interno di questa fase vi è un trasferimento di responsabilità delle merci dal fornitore al gestore del magazzino, quindi è opportuno controllare il loro stato e la corretta quantità.

Questo processo essendo il primo è responsabile del corretto funzionamento dei successivi, ed è necessario cercare di evitare che le merci restino ferme nelle zone di scarico/carico. Tipicamente nei magazzini si fa uso dei Transpallet elettrici per il trasporto in sicurezza delle merci precedentemente imballate utilizzando i pallet e per velocizzare le operazioni.

Il processo ideale consiste nel ricevere la quantità di merci prestabilita, in perfetta forma, e in un arco di tempo preciso, possibilmente in un formato facile da gestire all'interno del magazzino. Per migliorare questa fase si può cercare, nel caso sia possibile, di fissare degli standard al fornitore, molto precisi, in modo da poter ridurre al minimo lo sforzo. Tali standard possono riguardare sia il metodo di imballaggio del carico, quali dimensioni, peso e volume, sia il modo e il tempo in cui vengono consegnati.

Cercare di automatizzare il processo il più possibile aiuta a ridurre i tempi, ma sempre effettuando i dovuti controlli sulla qualità e lo stato delle merci, in modo da poterle catalogare rapidamente e in modo efficiente.

Nella gestione dei magazzini vengono impiegate spesso delle tecnologie come i barcode scanners che si interfacciano con il Warehouse Management System (WMS), in modo da automatizzare alcuni processi che sarebbero dispendiosi se eseguiti manualmente, questo aiuta a diminuire notevolmente i tempi di verifica delle merci e il dimensionamento dei pallet.

2.1.2 Processo di stoccaggio

Il processo di stoccaggio viene messo in pratica nel momento in cui si devono trasportare le merci dalla zona di scarico al punto finale dove verranno immagazzinate [3]. Questo processo deve essere svolto garantendo l' integrità delle merci e cercando di minimizzare lo sforzo e il tempo necessario al suo completamento.

Per massimizzare queste richieste è opportuno affidarsi a un Warehouse Management System (WMS) come per il processo di ricezione. Questi strumenti possono essere essenziali per un corretto svolgimento del processo di stoccaggio, garantendo anche che le merci vengano posizionate nel punto migliore del magazzino massimizzando l'efficienza.

Il corretto utilizzo di un WMS necessita di una raccolta di dati svolta in modo accurato ed efficiente e che essi vengano analizzati continuamente, più la raccolta dei dati è automatizzata, meno sarà possibile introdurre degli errori umani.

Molti carichi hanno dimensione e caratteristiche standard, (come consigliato di garantire nel processo di ricezione), questo può essere sfruttato dai software di gestione del magazzino in modo da evitare degli spostamenti inutili verso zone che non potranno ospitare il carico per limiti di spazio, e per minimizzare il viaggio dal punto di scarico al punto di posizionamento finale.

Un'analisi può essere condotta dai managers per ottimizzare lo spostamento, prendendo in considerazione che alcune merci vengono consegnate e richieste maggiormente rispetto ad altre, pesando il fatto che alcune merci possono avere delle caratteristiche fisiche particolari, come peso o fragilità, che necessitano di spazi appositi. Questo studio può essere effettuato utilizzando una tecnica conosciuta come ABC Analysis, approfondita nella sezione seguente.

L'utilizzo di barcode scanners o di tecnologie affini può essere impiegato per un monitoraggio dello spazio disponibile e della capacità del magazzino. Per un corretto funzionamento è però necessario che ogni spostamento di merce sia tracciato dal personale che lavora all'interno del magazzino.

Nel caso in cui il WMS sia in grado di fornire previsioni sul posizionamento delle merci prima ancora che esse siano arrivate, basandosi sugli ordini effettuati, si avrebbe un grosso vantaggio da sfruttare, applicando un processo di stoccaggio diretto (portando quindi le merci direttamente al punto di immagazzinamento, senza dover occupare la zona di scarico).

Nel processo di stoccaggio è possibile optare per avere un posto designato per ogni tipo di merce, favorendo l'efficienza per via della facile memorizzazione possibile per il personale del magazzino, oppure utilizzare un approccio dinamico, che permette una maggiore flessibilità. A seconda delle esigenze dei gestori dei magazzini, una di queste alternative può essere messa in atto, ma anche una combinazione di esse.

2.1.3 ABC Analysis

Una delle tecniche utilizzate nella gestione dei magazzini per classificare i tipi di merci è l' ABC Analysis, questa metodologia consiste nel creare tre gruppi concettuali, identificati con la lettera A, B e C e attribuire a ciascun gruppo alcuni oggetti dipendendo dalla loro rilevanza, e infine decidere dove fisicamente posizionarli in base al gruppo di appartenenza [4].

Nel gruppo identificato con la lettera A vengono assegnate le merci che hanno più importanza, le merci con cui principalmente viene effettuato il maggior guadagno, quelle vitali per l'azienda.

I prodotti che vengono usati davvero poco, e che ricoprono un ruolo secondario vengono invece attribuiti al gruppo B, tali merci vengono richieste meno frequentemente ma sono comunque una buona fonte di guadagno.

Il gruppo C comprende infine tutte le merci che rappresentano un plus, quasi mai richieste ma che necessitano di essere presenti.

Una volta definita la divisione delle merci nei precedenti gruppi logici, esse vengono posizionate all'interno del magazzino dipendentemente dal gruppo di appartenenza:

- Merci del **gruppo A**: dovranno essere messe nei punti più accessibili e di facile carico/scarico.
- Merci del **gruppo B**: saranno posizionate in una posizione di mezzo.
- Merci del **gruppo C**: andranno ad occupare i punti più lontani e difficilmente accessibili.

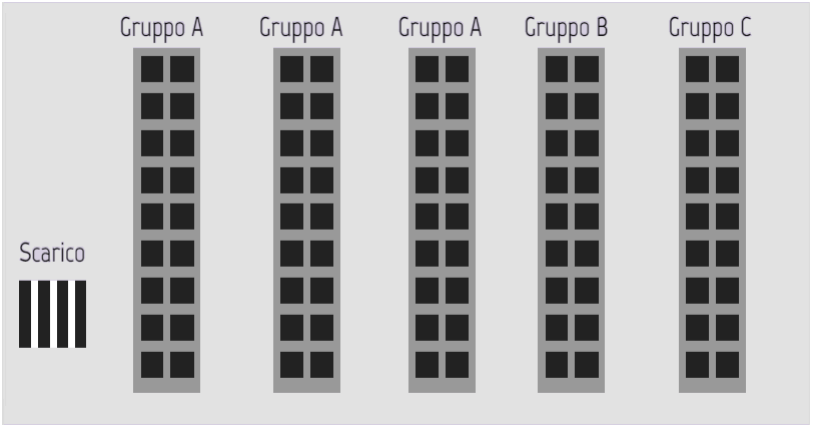


Figura 2: Esempio applicazione ABC Analysis ad un magazzino

2.1.4 Processo di immagazzinamento

A differenza degli altri processi, quello di immagazzinamento non è fisicamente connesso al singolo prodotto nel suo ciclo di vita all'interno del magazzino, ma viene effettuato a parte, globalmente, però è connesso alla scelta del posizionamento ideale delle merci, come spiegato in seguito.

Questo processo ha come obiettivo quello di sfruttare al massimo la capacità del magazzino, evitando di dover essere obbligati ad ampliarne la dimensione quando non necessario, e cerca di gestire alcuni periodi temporali in cui possono esserci più ordini e più flusso di merci, senza che nulla vada perso o gestito erroneamente [5].

Prima di analizzare lo spazio sfruttato è utile sapere che molti fattori influenzano questo processo, quali il posizionamento degli scaffali, la dimensione degli spazi, dei corridoi e il numero di zone di carico/scarico. Nella sezione successiva verranno analizzati i diversi sistemi di immagazzinamento possibili all'interno dei magazzini.

Per poter ottimizzare il processo di immagazzinamento è opportuno evitare di inserire gli oggetti all'interno di contenitori di dimensione troppo elevata, questo andrebbe a sprecare spazio utile per altre merci. Alcuni WMS possono aiutare nell'individuare il corretto contenitore per le merci, se gli si fornisce i dati necessari, di buona norma è sempre meglio cercare di racchiudere le merci di dimensioni diverse in contenitori di dimensioni diverse, occupando quello che meglio racchiude il prodotto, garantendone l'integrità, in modo da ottimizzare lo spazio.

Per poter sapere se si sta sfruttando adeguatamente lo spazio all'interno del magazzino è necessario raccogliere alcuni dati e combinarli insieme, senza trascurare che all'interno del magazzino deve esserci una percentuale di spazio che agevoli il personale nelle operazioni. Grazie a un appropriato WMS, e grazie alla raccolta dati effettuata nel processo di stoccaggio è possibile calcolare il volume occupato dalle merci all'interno del magazzino in ogni momento, questo dato e il primo che ci serve per poterne studiare il corretto posizionamento.

Una volta che si conosce l'effettivo utilizzo del magazzino bisogna confrontarlo con l'utilizzo potenziale che il magazzino offre. Per conoscere il potenziale del magazzino è necessario calcolare i volumi degli spazi destinati alle merci, tipicamente quello degli scaffali presenti all'interno del magazzino. Così facendo è possibile mettere in correlazione i due dati e vedere quanto spazio si sta utilizzando.

Come accennato in precedenza un elemento importante è anche evitare che lo spazio potenziale sia in eccesso o in difetto, per poterlo capire è sufficiente dividere il volume potenziale per il volume totale del magazzino, così si otterrà un valore compreso tra 0 e 1.

Per un corretto utilizzo del magazzino è necessario rimanere tra 0.22 e 0.27, se si supera questo limite si ha poco spazio per garantire al personale un corretto operato. Contrariamente, se non si raggiunge la soglia minima, significa che si sta utilizzando uno spazio inferiore a quello che il magazzino offrirebbe rispettando i termini di efficienza e sicurezza, e si può quindi pensare a una riorganizzazione dello spazio.

Infine per poter migliorare il processo di immagazzinamento è possibile correlare i concetti appresi in precedenza con dei particolari indicatori che vengono comunemente chiamati KPI (Key Process Indicator). Le KPI vengono scelte a seconda degli interessi del manager del magazzino e possono riguardare sia l'utilizzo dello spazio che la produttività o la rotazione di merci e verranno descritte in maniera più approfondita nelle prossime sezioni.

2.1.5 Sistemi di immagazzinamento

Una delle decisioni più importanti nella gestione di un magazzino è la scelta della tipologia di sistema di immagazzinamento. In base alle necessità e ai tipi di prodotti che sono immagazzinati viene scelta la tipologia che porta maggiori benefici. I modelli più popolari e maggiormente adottati sono i seguenti [6]:

- **Scaffalature statiche**: questa tecnica viene utilizzata con l'impiego di scaffali fissati, spesso adottata in magazzini di dimensioni molto ridotte dove non è necessario l'utilizzo di carrelli elevatori. Principalmente viene messa in atto per prodotti che hanno un peso molto ridotto e che necessitano di molte transazioni, quindi quelli maggiormente venduti nel caso siano sufficientemente leggeri.
- **Scaffalature mobili**: impiegati quando si ha la necessità di immagazzinare molti prodotti in uno spazio ristretto, sono più costosi dato che necessitano di un'infrastruttura per la mobilità. Questa soluzione fornisce molta flessibilità, offrendo la possibilità di non preoccuparsi degli spazi necessari tra uno scaffale e l'altro, dato che si possono creare quando necessario.
- **Scaffalature per pallet**: a differenza dei primi due modelli, la scaffalatura a pallet è impiegata in caso di magazzini di grandi dimensioni, dove tipicamente vengono utilizzati dei meccanismi automatizzati o dei carrelli elevatori per immagazzinare la merce. I prodotti vengono solitamente racchiusi in contenitori di grandi dimensioni e il punto di immagazzinamento viene scelto in base all'altezza dell'imballaggio.
- **Scaffalature multilivello**: la scelta migliore se si vuole sfruttare nel modo più efficiente lo spazio verticale del magazzino, progettato per con-

centrarsi in altezza, e pensato per prodotti di piccole dimensioni ma in grandi quantità. Data la natura delle merci, vengono solitamente usati con uno stoccaggio manuale manuale, ma l'impiego non è escluso in magazzini di piccole dimensioni. Tipicamente la gestione dei livelli è molto flessibile ed è possibile modificarli a piacimento.

- **Soppalco**: il soppalco è uno dei migliori sistemi di immagazzinamento, ma anche il più costoso. Permette di avere differenti piani all'interno del magazzino, in modo da poter automatizzare alcuni processi con l'utilizzo di macchinari, o di gestire in modo più personalizzato gli spazi, le illuminazioni e le merci. Per via dell'esigenza di un'apposita infrastruttura fisica viene adottato in magazzini di grandi dimensioni .
- **Partizioni di filo**: come per il soppalco anche questa scelta prevede l'utilizzo di un'infrastruttura, che consiste nella creazione di gabbie di ferro flessibili e facilmente montabili/smontabili, tipicamente utilizzate per prodotti fragili o per prodotti con caratteristiche particolari (per esempio alcune volte viene adottata quando si devono immagazzinare delle gomme per automobili).

La seguente figura riassume brevemente i tipi di sistemi di immagazzinamento in base alle caratteristiche che li contraddistinguono, considerando il costo per attuare il sistema e la dimensione del magazzino:

	Magazzini piccoli	**Magazzini grandi**
Costi minori	Scaffalature statiche	Scaffalature per pallet Scaffalature multi livello
Costi maggiori	Scaffalature dinamiche	Soppalco Partizioni di filo

Figura 3: Sistemi di immagazzinamento

2.1.6 Principali KPI per la gestione di un magazzino

Quando ci si trova a dover gestire un magazzino, alcune KPI possono essere molto utili da valutare per poter capire se si sta effettuando una corretta gestione del magazzino, o se si devono migliorare alcuni aspetti.

Gli indicatori che si possono valutare in questo settore sono davvero tanti, in seguito ne verranno spiegati alcuni dei più importanti secondo Derrik Weiss [7], che vengono suddivisi in cinque macro categorie: sicurezza, ricezione, stoccaggio, inventario e di ordinazione.

Uno degli aspetti fondamentali all'interno di un magazzino è la sicurezza, quindi è molto importante analizzare alcuni indicatori di questo settore:

- **Tempo perso a causa di incidenti**: solitamente misurato in ore, indica il tempo che sarebbe potuto essere utile sfruttandolo per lavorare se non ci fossero stati incidenti, più è basso e meno ci sono stati rallentamenti.
- **Tempo trascorso dall'ultimo incidente**: tipicamente misurato in giorni, più è elevato e maggiore è la sicurezza all'interno del magazzino.
- **Numero di incidenti per anno**: Idealmente non dovrebbero essercene, ma può essere utile tener traccia di essi per capire se si deve migliorare la sicurezza all'interno del magazzino.

Già dal processo di ricezione, il primo messo in atto, è utile effettuare delle misure e raccogliere delle KPI per poter vedere se è possibile un'ottimizzazione. Alcuni degli indicatori che possono essere analizzati in questo processo sono i seguenti:

- **Efficienza di ricezione**: utile per calcolare la produttività del personale, rappresenta il volume medio di merci ricevuti in un'ora lavorativa, massimizzandolo si aumenta la produttività.
- **Tempo di ciclo**: rappresenta il tempo medio necessario per gestire una ricezione, minimizzandolo si ha un'efficienza maggiore.
- **Costo per linea**: mostra il costo per la gestione di un singolo prodotto all'interno di una linea, e si ottiene dividendo il costo totale della gestione di una linea, inerente a un rifornimento, per il numero di prodotti presenti in essa. Cercando di ridurre questo indicatore viene diminuito il costo della gestione dei prodotti.

Proseguendo con i processi messi in atto si arriva allo stoccaggio, in cui possono essere analizzate delle KPI per poterne migliorare l'efficienza. Gli indicatori principali in questa fase sono:

- **Costo per linea**: esattamente come nel processo di ricezione sopra citato, è il costo per gestire un elemento della linea, ma applicato al processo di stoccaggio.
- **Tempo di ciclo**: rappresenta il tempo medio trascorso per lo stoccaggio delle merci.
- **Percentuale di precisione**: indica la percentuale di oggetti messi correttamente al loro posto rispetto al totale di oggetti riposti nel processo di stoccaggio.

La gestione dell'inventario è un aspetto importante quando si deve organizzare un magazzino, infatti verrà trattata in una sezione a parte, per poterla migliorarne si possono tenere in considerazione alcune KPI:

- **Precisione**: indica la percentuale di inventario che è fisicamente presente ed è stato registrato, si ottiene dividendo la dimensione dell'inventario digitale per il numero di prodotti effettivamente presenti nel magazzino, un risultato vicino a 1 indica che si sta operando nel modo corretto.
- **Perdita**: rappresenta la percentuale di perdita a livello economico data dal fatto che alcuni prodotti sono registrati virtualmente ma non presenti nel magazzino, e si ottiene dividendo la differenza dei costi registrati e fisici dell'inventario per il costo dell'inventario registrato.
- **Tasso di rotazione**: questo indicatore è su base temporale, e un valore elevato indica che quel prodotto è venduto maggiormente ed è molto richiesto. Uno dei modi per ottenerlo è dividere il numero di vendite per la dimensione media dell'inventario in un determinato periodo temporale.
- **Costo di gestione**: può essere utile sapere quale è la percentuale di costo per la gestione dell'inventario rispetto al costo totale di esso. Più è basso e maggiori sono i guadagni.

Infine può essere utile avere una visione più globale e verificare l'andamento complessivo del magazzino, per poterlo fare ci si può affidare a due KPI che prendono in considerazione gli ordini:

- **Accuratezza**: si ricava dividendo la differenza degli ordini totali e ritornati indietro per il numero di ordini totali. Questo indicatore è molto importante, cercando di ridurlo il più possibile a zero si ottiene una soddisfazione dei clienti totale, aumentando quindi l'affidabilità fornita.
- **Percentuale di restituzioni**: indica la percentuale di ordini restituiti rispetto al totale di quelli ricevuti. Anche in questo caso minimizzare questo indicatore comporterebbe un notevole incremento dell'affidabilità fornita, e migliorerebbe di molto l'efficienza della gestione del magazzino.

2.1.7 Processo di prelevamento

Il processo di prelevamento entra in gioco nel momento in cui uno o più ordini vengono ricevuti dai clienti e quindi la merce necessita di essere trovata e recuperata per poi poter attuare il processo di imballaggio.

Per poterlo ottimizzare si deve aver applicato l'ABC Analysis , descritta precedentemente, e successivamente si può procedere con la scelta di metodologia di prelevamento e l'introduzione della tecnologia [8].

Per quanto concerne la scelta di metodologia di prelevamento, si deve prendere in considerazione il fatto che alcuni magazzini possono ricevere ordini di dimensioni differenti. Alcuni degli approcci più utilizzati secondo Hector Sunol [8] sono i seguenti:

- **Prelevamento a pezzo/ordine**: questo approccio è il più intuitivo da mettere in atto, e consiste nell'avere una lista di tutti i prodotti relativi ad un ordine e nell'andare all'interno del magazzino a prelevare gli oggetti necessari uno a uno per poi poterli preparare per il processo di imballaggio. Purtroppo questa tecnica si può adottare efficientemente solo nel caso in cui gli ordini contengano un numero molto basso di prodotti (idealmente uno), altrimenti, in caso di grandi ordini, si potrebbero facilmente commettere errori. Il grande vantaggio che però offre (se applicato con uno o pochi prodotti) è una precisione molto elevata, questo perché effettuando il prelevamento individualmente si riducono le possibilità di errore. Inoltre, il fatto di poter raccogliere e preparare tutti i prodotti appartenenti ad un ordine insieme diminuisce la complessità della gestione del carico.
- **Prelevamento a cluster**: uno dei vantaggi principali offerti rispetto al prelevamento a pezzo è che si possono prelevare oggetti insieme appartenenti a diversi ordini, e effettuare anche una selezione in base alle priorità degli ordini se richiesto. Spesso in questi casi ci si affida a un WMS che fornisce la lista con il percorso migliore, e durante la raccolta i prodotti vengono separati per ordine in modo da non creare confusione ed incorrere in errori.
- **Prelevamento a zona**: la scelta migliore se si devono gestire tanti ordini di grandi dimensioni, questo approccio richiede che il magazzino venga diviso in zone (non necessariamente fisiche ma anche virtuali) e che il personale sia suddiviso in modo da operare sempre nella stessa zona. Assegnando prodotti diversi a zone diverse si migliora l'efficienza e la velocità, dato che dovendo gestire una piccola parte di magazzino si riesce a memorizzare facilmente la posizione dei prodotti. I WMS gestiscono ordini multipli e generano una lista per ogni zona in cui sono presenti dei prodotti necessari, essi vengono raccolti e passati di zona in zona con dei macchinari adibiti al trasporto, quali carrelli o nastri trasportatori.

- **Prelevamento ad onda**: molti WMS offrono la possibilità di attuare un prelevamento ad onda, in questo caso si utilizza un approccio temporale. Le liste di raccolta vengono erogate in tempi diversi nell'arco della giornata, anche in zone differenti. Grazie a questa tecnica si riesce anche a bilanciare il carico di lavoro del personale, e si può venire incontro anche a particolari esigenze di cui il personale può necessitare.

Nella figura sottostante sono riassunti i quattro tipi di prelevamento evidenziando quello più opportuno, tenendo in considerazione la possibilità di dividere il magazzino in zone e di gestire più ordini contemporaneamente o singolarmente.

	Senza divisione in zone	Con divisione in zone
Ordini singoli	Prelevamento a pezzo	Prelevamento a pezzo
Ordini multipli	Prelevamento a cluster	Prelevamento a zona Prelevamento a onda

Figura 4: Metodologie di prelevamento merci

Una volta decisa la tipologia di prelevamento da adattare all'interno del magazzino, è possibile utilizzare diverse tecnologie per lo svolgimento di questo processo. Nei magazzini uno degli elementi più utilizzati è il lettore di codice a barre, questo permette di poter scannerizzare in tempo reale i codici dei prodotti che si stanno prelevando per essere sicuri di non commettere errori. L'utilizzo di questi dispositivi richiede l'integrazione con il WMS che fornirà ed elaborerà i dati necessari.

In alcuni magazzini vengono impiegati dei piccoli computer indossabili in alternativa all'opzione precedente, questi dispositivi possono essere legati al braccio e talvolta offrono anche un lettore di codice a barre utilizzabile dal lavoratore. Utilizzando questa tecnologia si può interagire direttamente con il WMS in caso di necessità, ma si richiede che esso sia progettato per il supporto di questi dispositivi. Oltre al vantaggio sopra citato questi computer offrono la possibilità di avere una mobilità maggiore data la natura del loro posizionamento e quindi si riduce la possibilità di incidenti o danneggiamenti dei prodotti.

Infine se si vuole investire in questo settore è possibile acquistare dei dispositivi vocali appositi che il personale può indossare, essi si interfacciano con il sistema di gestione del magazzino. Così facendo è possibile comunicare vocalmente con il WMS e ottenere precise informazioni nel caso di bisogno, come posizione dei prodotti o quantità da prelevare o caratteristiche particolari.

2.1.8 Processo di imballaggio

Il processo di imballaggio viene utilizzato per cercare di garantire l'integrità totale del prodotto fino al raggiungimento della destinazione, come se fosse presente nel magazzino anche nel momento del trasporto [9].

Oltre a garantire l'integrità del prodotto è anche necessario cercare di non applicare imballaggi eccessivi che aumentino il peso della merce, quindi è necessario trovare un compromesso tra i due, in modo da ridurre i costi e le restituzioni da clienti insoddisfatti ma massimizzare la soddisfazione di chi ha effettuato l'acquisto.

I tipi di imballaggi possono dividere in due categorie a seconda della zona a cui vengono applicati:

- **Imballaggi esterni**: servono per proteggere sia quelli interni che il prodotto da spedire, devono avere come caratteristiche la possibile identificazione della merce interna (tipicamente con etichette e codice a barre) e la possibilità di raggruppamento facile in pallet, in modo che la merce possa essere organizzata con successo efficientemente.
- **Imballaggi interni**: sono quelli visibili al cliente finale, e devono seguire alcune restrizioni imposte da specifiche leggi. Per poter utilizzare un solido imballaggio interno occorre scegliere tra i diversi tipi disponibili quello che più è appropriato rispetto al tipo di merce che viene imballata, solitamente si differenziano per materiale, dimensioni, peso e altre caratteristiche.

Data la natura degli imballaggi esterni, si può notare che i clienti che decidono di effettuare ordini che comprendono interi pallet beneficino maggiormente degli imballaggi e godano di una sicurezza maggiore.

Legato al processo di imballaggio c'è anche un aspetto ambientale, dato che gli imballaggi diventano inutili nel momento che la consegna è avvenuta con successo, è opportuno cercare di crearli in modo da poterli facilmente riciclare. Per questa ragione molti materiali da imballaggio attualmente in circolazione sono fatti di carta, cartone alluminio e plastica riciclabile.

Per poter ottimizzare al massimo questo processo è quindi necessaria una scelta appropriata degli imballaggi interni a seconda delle esigenze, ma soprattutto una corretta applicazione di imballaggi esterni in modo da ottimizzare lo spazio del magazzino e velocizzare le operazioni di gestione di flusso di merci.

Molti dei più popolari siti di e-commerce utilizzano gli imballaggi anche per una questione di privacy. Se si suddividono in modo appropriato i compiti, la parte del personale adibita all'imballaggio (o al controllo del corretto avvenimento nel caso in cui il processo sia automatizzato) non verrà a conoscenza di chi ha effettuato l'acquisto.

2.1.9 Processo di spedizione

L'ultimo processo da mettere in atto nel ciclo di vita di un prodotto all'interno del magazzino è quello di spedizione. Una vendita può considerarsi terminata solamente se il prodotto corretto viene spedito al cliente che lo ha richiesto, nei tempi prestabiliti e attraverso la giusta modalità di trasporto [1].

Trattandosi dell'ultimo anello della catena, questo processo è fortemente correlato con tutti quelli precedenti, i quali ne influenzano il corretto svolgimento provando a massimizzarne l'efficienza ed a diminuirne i costi e i tempi.

Durante questo processo, come spiegato da Hector Sunol [10], vengono identificate tre fasi principali:

- **Pesatura e dimensionamento**: a causa del processo di imballaggio il peso e le dimensioni della merce possono subire variazioni. In questa fase è opportuno tenere traccia di tale variazione e registrare il peso finale del prodotto. Uno dei modi migliori per effettuare la pesatura è avvalersi di appositi macchinari che lo fanno in modo automatico mentre le merci attraversano il processo di imballaggio, questo però richiede l'integrazione con il sistema di gestione dei magazzini. Tale procedimento è obbligatorio perché le etichette sulle merci devono riportare i dati aggiornati al momento della spedizione.
- **Documentazione**: ogni merce ha la necessità di essere documentata, dipendentemente dalla natura stessa del prodotto, queste regolamentazioni sono a norma di legge ed è necessario che parte del processo di spedizione si assicuri una corretta veridicità di esse. Queste informazioni possono riguardare diversi aspetti quali il carico complessivo, il certificato di origine, di assicurazione, la fattura erogata o altro. Per poter fornire questa documentazione nel modo più rapido ed efficiente possibile, limitando gli errori, è opportuno avvalersi di un appropriato WMS.
- **Caricamento**: come per lo scarico delle merci, è importante anche il modo in cui esse vengono caricate sul mezzo di trasporto finale, ed è essenziale che al personale siano fornite le informazioni necessarie su come farlo correttamente. Per poter massimizzare l'efficienza è possibile sia affidarsi a software per la gestione dello scarico che avere delle aree adibite al posizionamento di carichi già pronti per la spedizione, in modo da velocizzare il processo.

Per non intralciare il processo di spedizione è bene che le merci che arrivano al magazzino siano fisicamente e nettamente **separate** da quelle che devono essere spedite. Se non è possibile farlo per una mancanza di spazio fisico è consigliabile che le due operazioni siano svolte in fasce orarie differenti, in modo che sia garantito il completamento di un'operazione prima dell'avvio dell'altra.

2.2 Gestione dell'inventario

La gestione dell'inventario è il secondo aspetto di vitale importanza per una corretta gestione di un magazzino. In parte questo concetto si collega al processo di immagazzinamento perché molte delle decisioni prese in esso possono derivare da quelle adottate per avere una corretta organizzazione dell'inventario.

Per non confondere i concetti successivamente verranno riportate le differenze fondamentali tra le due, è da ricordare infatti che la seconda si occupa non solo del posizionamento corretto delle merci all'interno del magazzino, ma anche del controllo della quantità, delle dimensioni, del peso e del numero delle stesse, in modo da poter sapere quando richiedere una nuova fornitura di un determinato prodotto [11].

Anche in questo caso è possibile affidarsi alla tecnologia per poter migliorare e perfezionare la gestione dell'inventario, avvalendosi di appositi software chiamati Inventory Management System (IMS). Questi software hanno uno scopo leggermente differente rispetto a quelli per la gestione del magazzino, si possono notare in seguito analizzando le principali differenze tra i due.

Quando si parla di gestione dell'inventario si fa spesso riferimento alle Stock Keeping Unit (SKU) e quindi è importante apprenderne il significato e l'integrazione all'interno dell'inventario, per questa ragione una sezione sottostante verrà dedicata a questo concetto, cercando di capirne l'utilità e il funzionamento.

Dato che gli IMS ricoprono un ruolo importante anche nella gestione di un magazzino, la scelta di essi è una decisione molto importante, da non prendere alla leggera. Per questo motivo la parte finale di questo capitolo sarà dedicata ad analizzare le caratteristiche principali che questi software devono offrire per poter essere scelti ed implementati con successo.

2.2.1 Differenze tra gestione dell'inventario e del magazzino

La differenza tra gestione del magazzino e gestione dell'inventario sembra sottile ma in realtà non lo è, molto spesso questi due concetti vengono fusi o scambiati erroneamente, per questo motivo in seguito verranno analizzate più nel dettaglio alcune delle loro peculiarità che li contraddistinguono [11]:

- **Complessità e caratteristiche**: quando si utilizzano degli Inventory Management Systems si possono avere delle informazioni più nette e concise riguardante i prodotti (come la loro quantità presente in una determinata zona del magazzino), per questo motivo sono molto più semplici. Se invece si utilizza un software per la gestione del magazzino (WMS) si possono avere più informazioni dettagliate che comprendono dati che

possono arrivare anche da diversi IMS, motivo per cui sono più complessi e possono fornire una visione più globale.

- **Controllo e immagazzinamento**: nel caso di gestione dell'inventario come detto precedentemente si può avere il controllo solamente sul numero di articoli che sono presenti nel magazzino per un determinato prodotto. Come invece visto nella sezione sulla gestione dell'immagazzinamento, utilizzando un WMS si può avere il controllo anche su dove posizionare il prodotto e dove recuperarlo, tenendo in considerazione diversi fattori.
- **Integrazione**: la gestione del magazzino ingloba anche operazioni che si possono svolgere contemporaneamente in altri dipartimenti, ed è quindi connessa ad altri aspetti della gestione dell'azienda (come la gestione della qualità, la fornitura di produzione, le vendite e le distribuzioni). Per questa ragione, a differenza della gestione dell'inventario, può essere integrata facilmente nella logistica generale della gestione di un'azienda.
- **Soluzioni software**: come visto nel processo di prelevamento i WMS offrono la possibilità di analizzare e modificare l'inventario con l'ausilio di diverse tecnologie, potendo interagire con esso in modo rapido ed efficiente. I software per la gestione dell'inventario contrariamente offrono solo la possibilità di avere informazioni digitali su cosa è tenuto nel magazzino, senza fornire in alcun modo un motivo per poter apportare dei cambiamenti.

In conclusione quindi è possibile notare come sia marcata la differenza tra i due concetti, ma come essi siano entrambi necessari per la gestione di un magazzino.

In alcune occasioni è possibile che si voglia sapere solamente lo stato attuale del magazzino (considerando le merci presenti all'interno), in questo caso un IMS è quello che più viene incontro a queste esigenze dato che si sta parlando di una gestione dell'inventario.

Come visto nel processo di immagazzinamento in alcune situazioni è necessario dover gestire il magazzino in maniera efficiente e quindi affidarsi a un WMS per poterne ottimizzare lo spazio e i tempi (quindi per una gestione globale del magazzino).

2.2.2 Stock Keeping Unit: SKU

Molto spesso nell'ambito della gestione dell'inventario si fa riferimento alle SKU (Stock Keeping Unit), esse rappresentano un singolo prodotto, depositato in stock, in una specifica posizione all'interno del magazzino [12].

Gestire un magazzino identificando tutte le SKU fornisce la possibilità di avere un'organizzazione più dettagliata e da spazio a possibili ottimizzazioni da parte di software per la gestione di esso.

Le SKU si differenziano dai prodotti in quanto esse sono strettamente connesse a una posizione all'interno del magazzino. Ci sono molti motivi per cui depositare in parti differenti quantitativi dello stesso prodotto all'interno di un unico magazzino (per spazio ma anche per ragioni logiche, come merci apposite per clienti fissi), in questo caso avremo una SKU per ogni luogo all'interno del magazzino dove verrà posizionata la merce con egual caratteristiche.

La presenza di merci uguali in posti differenti non è la sola motivazione che identifica una SKU, è possibile averne più di una inerente nello steso posto qualora il prodotto presenti caratteristiche diverse quali dimensioni, colori e peculiarità affini.

Come riportato da Joannès Vermorel [12], ogni SKU è legata alle scorte disponibili corrispondenti, quindi al numero di unità già pronte per essere prelevate nel punto indicato dalla SKU. Le scorte ordinate sono invece quelle già in consegna, che dovranno rifornire la SKU.

Per questa ragione le scorte disponibili in un determinato momento rappresentano lo stato di un SKU. Nella pratica non viene mai svolto un conteggio manuale delle scorte presenti all'interno del magazzino, ma grazie agli appositi software e al tracciamento del flusso delle merci (entrate ed uscite) viene implicitamente calcolata la disponibilità in un determinato momento temporale.

Nel caso sia necessario controllare manualmente (anche se fortemente sconsigliato perché molto costoso a livello di tempo e denaro), esistono due modi principalmente per poter effettuare il conteggio:

- **Disponibilità in numeri interi**: usata nella maggior parte dei casi, dato che attualmente molta merce in circolazione è venduta imballata in pallet e messa in contenitori che ne indicano la quantità contenuta.
- **Disponibilità in peso**: quando la merce non ha caratteristiche identiche per ogni pezzo è necessario effettuare un conteggio basandosi sul peso, come per esempio nei settori alimentari.

Ogni SKU viene identificata da una sequenza di caratteri alfanumerici univoca (a differenza dei codici a barre che sono solo numerici), che viene letta tramite appositi apparecchi per la lettura.

2.2.3 Caratteristiche principali di un IMS

Per poter scegliere il software che meglio va incontro alle necessità che un magazzino può avere è opportuno che alcune caratteristiche siano presenti e alcune funzionalità siano fornite. Secondo Bryan barry [13] le principali caratteristiche che deve avere un software per la gestione dell'inventario sono le seguenti:

- **Miglioramento del controllo dell'inventario e previsioni**: un IMS deve fornire delle solide informazioni riguardanti gli SKU, come le caratteristiche principali e i costi dettagliati. Oltre a questo deve anche essere possibile una previsione o un progetto sui futuri SKU che saranno richiesti dai clienti. Se i magazzini ospitano merci che possono essere vendute su diversi canali (come fisicamente oppure online o con metodi differenti) è necessaria la possibilità di poter fornire una visione separata per ogni canale. Come ultima caratteristica che rientra in questo settore vi è la possibilità di fornire delle previsioni basate su degli algoritmi leggibili e delle funzioni di analisi comprensibili e studiabili.
- **Codici a barre e scansioni**: molti WMS si basano su un sistema di codici a barre e sono dotati di meccanismi per la loro interpretazione. Come detto in precedenza gli SKU sono identificati da dei particolari codici alfanumerici, tendenzialmente i software per la gestione del magazzino offrono l'interoperabilità con questi codici, è opportuno che anche un IMS sia in grado di supportare correttamente questa funzione.
- **Analisi dell'inventario migliorata e fruibile**: dato che è lo scopo principali di questi sistemi è opportuno che sia possibile consultare in modo efficiente e rapido l'inventario, quindi è necessario valutare quali sono le metriche che vengono usate per misurare le caratteristiche dell'inventario. Una metodologia di scelta può anche essere il valutare quante funzioni specifiche di ricerca e gestione offre il sistema e quale livello di personalizzazione è possibile raggiungere. Un IMS deve offrire la possibilità di vedere tutti questi dati in un tempo molto breve e in un modo chiaro e conciso, l'impatto con la dashboard deve essere semplice ed intuitivo.
- **Configurabilità**: una delle caratteristiche che spesso vengono richieste ma non fornite nella versione standard è la personalizzazione del sistema, quindi uno degli aspetti da valutare è quanto esso sia configurabile in modo che sia visibilmente piacevole e comprensibile da diverse entità. La personalizzazione spesso è riguardante la gestione degli SKU all'interno del magazzino e richiede di poter avere statistiche ed informazioni differenti a seconda delle necessità.
- **Integrazione ed interfacce**: spesso il sistema che si decide di adottare non presenta tutte le caratteristiche di cui si ha bisogno, in questo caso è opportuno valutare se è possibile effettuare delle integrazioni, quali sono

le eventuali interfacce che si possono usare e in quale modo. Una delle integrazioni più importante è quella con il WMS.

- **On-Premise vs SaaS and cloud system**: il fattore economico nella scelta di un IMS gioca un ruolo importante, molto spesso si opta per un sistema che immagazzina i dati altrove (un SaaS o un cloud system), in questo modo i costi e i tempi di implementazione del servizio sono ridotti. Se invece si ha la possibilità o la necessita di avere i dati gestiti localmente si può optare per una scelta di un IMS che immagazzini i dati On-Premise, e quindi gestibili in sede.

Come visto quindi ci sono diverse ragioni per cui preferire un IMS rispetto ad un altro, e sono di diversa natura, occorre quindi effettuare uno studio iniziale prendendo in considerazione questi fattori nel caso si debba cambiare o integrare da nuovo un IMS.

2.3 Analisi e confronto dei principali WMS

In questa sezione verranno prima presentati i maggiori pacchetti software per la gestione di un magazzino e successivamente verrà fatto un confronto tra essi per capirne meglio i pregi e i difetti.

Dato che la gestione del magazzino si basa su questi software è opportuno sceglierli con cura perché, come visto precedentemente, possono portare enormi benefici sia organizzativi che di profitto per il magazzino. Una scelta errata può danneggiare in maniera consistente il ciclo di vita delle merci e l'intera gestione del magazzino.

In seguito nella presentazione dei programmi si farà riferimento al termine ERP che sta a significare Enterprise resource planning. Si tratta di un software che pianifica le risorse dell'impresa, gestisce ed integra tutte le funzioni aziendali e i processi di business dell'azienda, tra cui anche la gestione del magazzino [14], conoscerne il significato renderà più semplice la comprensione.

2.3.1 Principali software per la gestione di un magazzino

Data la necessità di avere un software per la gestione del magazzino valido e adatto per l'occasione verranno mostrati brevemente alcuni dei principali WMS e le funzionalità rilevanti offerte da essi. I primi due WMS elencati sono a carattere più generale e sono molto validi in svariati ambiti, oltre che essere leader nel mercato attuale, mentre gli ultimi due hanno come peculiarità il fatto di essere usati molto in ambito automotive.

Un fattore che è opportuno ricordare è che i WMS possono differenziarsi per il modo in cui operano e vengono eseguiti. I due modi principali possibili sono gli stessi dei software per la gestione dell'inventario (spesso integrati già in quelli della gestione del magazzino), cioè installati in locale o eseguiti in cloud (Saas). Tutti i seguenti software sono da considerarsi parte di una top 10 dei WMS aggiornata al 2020 effettuata da Software Testing Help [15]:

- **Fishbowl Warehouse**: questo software è utilizzabile da organizzazioni di qualunque dimensione data la sua enorme flessibilità. Offre un periodo di prova gratuito e la possibilità di creare ordini di acquisto e vendita in qualunque valuta, quindi utilizzabile universalmente. Contrariamente però è disponibile solamente in lingua inglese e questo può rappresentare un grosso svantaggio in alcune aziende. Offre un'enorme varietà di plugin che possono accomodare molte necessità tra cui integrarsi con QuickBooks e Xero in ambito accounting, ma non supporta moduli nativi per la vendita e moduli di e-commerce. Fornisce la possibilità di gestire ordini, tracciarli e gestire le vendite. Purtroppo ha come svantaggio il fatto di non poter essere installato localmente, ma fa parte della famiglia dei Saas.

- **NetSuite**: viene impiegato come software di supporto nelle principali attività del magazzino come ricezione, immagazzinamento stoccaggio e spedizione di merci. Una delle caratteristiche offerte è la scansione di codici a barre con radio frequenze mobili. Non è un vero e proprio ERP ma fornisce molte delle loro funzioni. Per quanto riguarda l'utilizzo è molto semplice da usare, fornisce plugin di terze parti e accomoda la possibilità di gestione del magazzino in modo wireless. Tra gli aspetti positivi vi solo la presenza di una garanzia offerta dall'azienda e il fatto di aver ben visibili le informazioni di contatto. Un altro degli aspetti utili è che gestisce bene i casi in cui avvengono restituzioni delle merci.
- **Infor SCM**: un software che fornisce funzioni specifiche e assistenza sul processo di vendita. Tra le operazioni possibili vi è la visualizzazione in 3D del WMS e una buona gestione dell'inventario, non ottima dato che non esiste supporto per la pianificazione dei requisiti dei materiali o la possibilità di avere dei report automatici nel caso di stock esauriti. Come principale svantaggio ha il fatto di non fornire la possibilità di esportare dei report o una visualizzazione grafica di essi. Inoltre non fornisce una struttura per la gestione dei documenti ma offre la possibilità di tracciare in modo centralizzato le vendite e una buona gamma di funzionalità inerenti all'ambito finanziario.
- **HighJump**: è un sistema basato su cloud che offre anche un'integrazione personalizzata con un ERP presente, indipendentemente dalla dimensione dell'azienda. Si presenta come un sistema robusto ma senza possibilità di gestire processi di pagamenti con carte, anche in questo caso non è presente una funzionalità per generare dei report. Come punto di forza vi è anche il fatto che è in continuo aggiornamento cercando di accomodare quelle che sono le richieste dei clienti grazie ai loro feedback.

Come detto a inizio sezione ci sono molti software che possono venire incontro alle esigenze di un magazzino, la scelta richiede uno studio dettagliato a seconda delle necessità. Quelli elencati sono stati selezionati in base all'utilizzo globale e al settore di interesse in questa tesi, cioè la gestione di un magazzino in ambito automotive.

La presentazione dei WMS è avvenuta tenendo in considerazione alcuni dei fattori che possono essere necessari per una corretta gestione del magazzino, ma ci sono molte altre funzionalità che ognuno di questi software offre in aggiunta. Per poter avere un'idea più chiara è opportuno effettuare uno studio approfondito su quali sono i requisiti richiesti e quali di questi software ne accomodano la maggior parte in maniera soddisfacente.

Precedentemente sono stati elencati alcuni dei punti chiave e delle caratteristiche di questi quattro software, per un confronto più visivo e per alcune informazioni aggiuntive si può fare riferimento alla sezione seguente.

2.3.2 Confronto tra i principali WMS

Nel confronto tra i principali software verranno presi in considerazione solamente i quattro di cui si è parlato nelle pagine precedenti. Oltre a una vista riassuntiva verranno introdotte alcuni informazioni che possono essere rilevanti nella scelta, come ad esempio il prezzo nel caso in cui non sia variabile. Nell'immagine seguente è possibile vedere il quadro generale [15]:

	Prezzo	Vantaggi	Svantaggi	Caratteristiche principali
Fishbowl Warehouse	$4,395 per utente all'anno	Periodo di prova gratuito Ordine e vendite in diverse valute Molti plugin	Solo lingua inglese No vendite native e moduli di e-commerce No installazione in locale	Integrazione con software per accounting Produzione report Gestione degli ordini Gestione delle vendite e degli acquisti
NetSuite	Licenza base: $999 + $99 per utente al mese	Facile da usare Garanzia offerta dall'azienda Possiede plugin di terze parti Informazioni contatto visibili Supporto magazzini wireless	Non un vero ERP, fornisce solo alcune funzionalità	Scansione codici a barre con radio frequenze mobili Gestione delle restituzioni delle merci Gestione dei compiti Pianifica il conteggio dei cicli
Infor SCM	Variabile, necessario contattare l'azienda per un preventivo	Assistenza per processo di vendita Supporto per funzioni specifiche d'industria	No report grafici No esportazione di report No gestione dei documenti No supporto pianificazione dei requisiti dei materiali No report automatici se stock esauriti	Visualizzazione WMS in 3D Gestione dell'inventario Tracciamento delle vendite centralizzato Funzionalità in ambito finanziario
HighJump	Variabile, necessario contattare l'azienda per un preventivo	Sistema robusto Aggiornamenti grazie ai feedback	No report No gestione dei processi di pagamenti con carte	Basato su cloud Supporta l'integrazione ERP Funzionalità per il processo di prelevamento

Figura 5: Confronto tra WMS

Alcuni dei WMS in circolazione non forniscono informazioni sul prezzo perché provvedono a fornire un piano economico dettagliato in base alle dimensioni dell'azienda e alle funzionalità richieste, per questa ragione è necessario contattare privatamente la compagnia.

Anche nel caso della scelta di un WMS è opportuno tenere in considerazione altri fattori che possono influenzare tra cui particolari esigenze del magazzino e funzionalità specifiche che si desiderano. Uno di questi fattori può essere l'esigenza di inglobare in questo software anche parte della gestione della produzione, se non avere addirittura un unico programma per gestire tutto, ovviamente possibile solo nel caso in cui non vi sia solo immagazzinamento ma anche una catena di produzione propria.

Oggigiorno molti WMS vengono utilizzati all'interno di altri software che hanno uno scopo di gestione generale dell'azienda, tra cui è possibile avere anche quella del magazzino. Questa decisione è però molto importante da prendere e si può effettuare solo nel caso in cui la gestione del magazzino non sia l'intera attività dell'azienda ma una parte di essa.

3 Sviluppo nativo vs multipiattaforma

Una delle scelte più importanti da effettuare quando si vuole creare un'applicazione è la metodologia di sviluppo che si dovrà adottare. Attualmente esistono due tipologie di sviluppo per le applicazioni: nativo e multipiattaforma, ognuno con le sue caratteristiche, vantaggi e svantaggi.

Per fare una scelta corretta occorre conoscerne il funzionamento di entrambi e capire quale più viene incontro alle necessità che si hanno. Le differenze che esistono tra le due metodologie spaziano dal modo in cui vengono distribuite le applicazioni all'integrazione o meno con l'hardware fino alla necessità o meno di compilazioni/interpretazioni. In questo capitolo verranno spiegati in modo più approfondito i due metodi e ne verranno evidenziati i pregi e i difetti. Saranno inoltre approfonditi i framework e/o linguaggi utilizzati relativi a ogni tipologia di sviluppo.

Per poter decidere correttamente è opportuno avere una visione chiara, coincisa e globale di queste metodologie e avere la possibilità di metterne rapidamente a confronto pregi e difetti, per questa ragione la parte finale di questo capitolo avrà come scopo il confronto tra queste due tipologie di sviluppo.

3.1 App native

Le app native ricoprono la maggior parte del settore delle mobile applications, esse sono create per poter essere usate su una determinata piattaforma e vengono sviluppate con tecnologie e linguaggi ad hoc.

Le due principali tipologie di app native che vengono create oggigiorno sono Android e iOS, le prime per essere usate su dispositivi con l'omonimo sistema operativo, le seconde per essere utilizzate su dispositivi prodotti dall'azienda Apple.

Le app native sono create utilizzando degli ambienti di sviluppo conosciuti come IDE (Integrated Development Environment) che vengono eseguiti sulla stessa piattaforma per cui l'applicazione viene creata. Ognuno di questi IDE mette a disposizione del programmatore una moltitudine di strumenti ed interfacce che possono essere utilizzati per lo sviluppo e la documentazione del software, spesso identificati con la parola SDK (Software Development Kit) [16].

Lo sviluppo di applicazioni native offre la possibilità di beneficiare dei seguenti aspetti [17]:

- Utilizzo di tutte le funzionalità del dispositivo per cui sono create in maniera rapida ed efficiente, come fotocamera, GPS, bluetooth, file system ed altro.

- Familiarità da parte dell'utente per via di componenti sviluppati specificatamente per quella piattaforma, garantendo una consistenza con le altre applicazioni e un senso di familiarità.
- Potenza di calcolo e velocità massima dato che queste applicazioni vengono installate e lanciate localmente sul dispositivo (oltre al fatto di essere pensate per questo scopo) e sono in grado di lavorare autonomamente.
- Possibilità di esecuzione offline, ossia avere la garanzia di un corretto funzionamento anche senza una connessione internet (salvo che l'applicazione non sia studiata per lavorare solamente in presenza di internet).
- Miglior metodologia di distribuzione dell'app, grazie agli appositi store è più semplice conoscerne l'esistenza ed è più rapida, facile ed intuitiva l'installazione.
- Massima velocità di interazione con l'utente e tempi d'attesa minimi nel cambio tra schermate (i quali sarebbero molto più elevati per le web applications, soprattutto nel caso si verifichi un rallentamento della rete).
- Opportunità di sfruttare al meglio ogni tipo di interazione con il dispositivo, quali gestures e comandi vocali.
- Supporto fornito dallo store che ne garantisce sicurezza e protezione, data la necessità di approvazione da parte dello stesso prima della pubblicazione.
- Salvataggio di dati sul dispositivo in modo efficiente per poterne beneficiare nel momento in cui avverrà un nuovo rilancio dell'app.

Date le caratteristiche sopra elencate si direbbe che optare per uno sviluppo di un'applicazione nativa sia la scelta migliore. Purtroppo insieme a questi vantaggi esistono anche una serie di caratteristiche negative che si incontrerebbero nel caso si decidesse di optare per questa scelta. Alcuni degli aspetti negativi nella scelta di un'applicazione nativa sono i seguenti [17]:

- Necessità di sviluppo di un'applicazione per ogni piattaforma diversa, se si decide di sviluppare un'applicazione per Android essa non potrà girare su un iPhone e viceversa. Questa caratteristica implica che bisogna conoscere i linguaggi di programmazione per ogni piattaforma per cui si vuole creare l'applicazione
- Costi e tempi di sviluppo maggiori dati dalla necessita di utilizzo di linguaggi specifici.
- Investimento maggiore di tempo e denaro per la manutenzione a tempo indeterminato data l'assenza di una retrocompatibilità garantita nel caso di aggiornamento di alcuni dispositivi o di nuove introduzioni.

- Necessità di un processo di approvazione da parte dello store su cui verrà pubblicata l'app, che spesso richiede tempo.
- Obbligo da parte dell'utente di dover seguire un numero cospicuo di passaggi per poter usufruire dell'applicazione, che inizia con l'accesso allo store e termina con l'installazione dell'applicazione dopo aver acconsentito a tutte le richieste di termini e condizioni.

Come visto, ci sono quindi molti aspetti che contraddistinguono le applicazioni native, sia in positivo che in negativo. Per poter scegliere questo approccio è necessario quindi considerare tutti questi aspetti, nell'immagine seguente sono riassunte tutte le caratteristiche viste precedentemente in modo da visualizzare un quadro più generale, netto e coinciso.

Vantaggi	Svantaggi
- Confidenzialità e famigliarità	- Tempi e costi mantenimento
- Potenza di calcolo	- Tempi e costi sviluppo
- Esecuzione offline	- Processo di approvazione
- Velocità d'interazione	- Molti step per avere l'app
- Metodologia di distribuzione	- Sviluppo differente per ogni piattaforma
- Sicurezza, protezione e supporto dallo store	
- Utilizzo funzionalità del dispositivo	
- Salvataggio locale dei dati	
- Interazioni multiple	

Figura 6: Caratteristiche sviluppo native

3.1.1 Android

Android è uno dei sistemi operativi per cui vengono sviluppate la maggior parte delle applicazioni native. Attualmente più del 70% dei dispositivi in circolazione hanno Android [18], il restante ospita iOS (a meno dell'1% che racchiude ancora altri sistemi operativi), quindi se si punta a raggiungere la maggior parte dell'utenza è essenziale sviluppare un'app che sia ospitabile su Android .

Per sviluppare app in Android vengono principalmente utilizzati come IDE Android Studio ed Eclipse, il codice scritto è basato su Java e quindi è possibile programmare su PC Windows. Dato che Android utilizza principalmente Java è possibile usare l'omonimo linguaggio di programmazione, anche se in alcuni casi vengono usati Kotlin e C++. Essendo Android creato da Google, è possibile utilizzare molti degli strumenti di sviluppo forniti da quest'azienda come [19]:

- **Android Jetpack**: un set librerie, strumenti e guide che aiuta gli sviluppatori a creare facilmente app di qualità elevata, fornendo linee guida specifiche e semplificando l'implementazione di funzioni più complesse [20].
- **Firebase**: una piattaforma per lo sviluppo di applicazioni Web e mobili acquistata dalla Google.
- **Android SDK**: è il set di strumenti di sviluppo fornito per Android , un ambiente di sviluppo integrato connesso con Android Studio.

Sviluppare applicazioni native in Android porta alcuni vantaggi, in primo luogo viene offerta la possibilità di avere accesso a più funzionalità rispetto ad iOS, per via dalla natura open source del sistema. Come detto precedentemente il processo di pubblicazione di un'applicazione sullo store per un'app nativa è lungo, ma nel caso di Android è possibile eseguire questo passaggio in poche ore, facilitandone il rilascio. Grazie alle linee guida fornite è anche possibile aumentare la familiarità per gli utenti utilizzando componenti grafici e layout con cui si può interagire facilmente. Infine Android permette uno sviluppo frammentato a seconda del dispositivo finale, il quale può spaziare dalle automobili agli orologi fino alle televisioni e molto altro (oltre che a tablet e smartphone).

Scegliere di sviluppare in questo modo un'applicazione nativa porta anche alcuni aspetti negativi, il più importante è la richiesta di molto tempo necessario per testare le applicazioni (dovuto alla presenza di diverse versioni e uno svariato numero di dispositivi, a differenza di iOS). Avere una frammentazione può considerarsi anche uno svantaggio dato il tempo che è richiesto agli sviluppatori per creare versioni compatibili con la maggior parte dei dispositivi. L'ultimo aspetto da tenere in conto nella scelta è il costo di sviluppo, per un' applicazione Android oltre che a dipendere dalla complessità di essa e dalle funzionalità che offre il costo dipende anche dal tempo necessario per svilupparla e testarla (spesso per le app iOS invece viene stanziato un budget fisso per la creazione).

3.1.2 iOS

Il sistema operativo iOS è l'altra faccia delle medaglia riguardante lo sviluppo di app native, a differenza di Android occupa una fetta di mercato inferiore, ma comunque in costante crescita negli ultimi anni. Attualmente quasi il 29% dei dispositivi mobile ha come sistema operativo iOS [18], nonostante sia in minoranza ricopre comunque un ruolo fondamentale nello sviluppo di app native. La maggior parte delle applicazioni Android odierne sono anche sviluppate per iOS.

Per poter sviluppare un'applicazione di questo tipo è necessario lavorare su dispositivi Apple usando come IDE Xcode che attualmente è uno dei migliori software in circolazione per la creazione di app iOS. I linguaggi di programmazione utilizzati sono Swift (sviluppato da Apple) e Objective-C (evoluzione del C). Sviluppare un'applicazione iOS offre l'opportunità di beneficiare dei seguenti strumenti [19]:

- **Swift Playgrounds**: applicazione sviluppata dalla Apple per dare la possibilità di programmare in Swift interfacciandosi con un ambiente semplice ed intuitivo.
- **TestFlight**: applicazione utilizzata per installare e testare app sviluppate per iOS.
- **iOS SDK**: insieme di strumenti di sviluppo fornito dalla Apple utilizzati grazie a un'integrazione con Xcode.

Quando si decide di sviluppare un'app nativa in iOS si ottengono alcuni vantaggi difficili da raggiungere scegliendo Android. In primo luogo vi è un profitto economico perché i possessori di dispositivi Apple spendono mediamente di più nell'acquisto di applicazioni rispetto agli utenti Android. A seguire vi è un beneficio dato dal fatto che i dispositivi creati dall'azienda Apple sono in numero nettamente inferiore rispetto a quelli che supportano Android, quindi si riduce notevolmente il problema legato alla frammentazione e alla necessità di sviluppare app compatibili con molti dispositivi. Infine è possibile beneficiare di una serie di interfacce grafiche che gli sviluppatori iOS hanno creato in modo molto dettagliato focalizzandosi principalmente sulla familiarità per l'utente. Utilizzando le guide appropriate è possibile una rapida integrazione con un notevole risparmio di tempo.

Anche in questo caso ci si imbatte in un paio di aspetti negativi decidendo di adottare questa metodologia, innanzitutto vi è una flessibilità molto limitata data dal fatto che il sistema è chiuso, e quindi non open source. In secondo luogo vi è la necessità di caratteristiche molto particolari per la pubblicazione dell'app nello store, in materia di sicurezza e altro. Per questo motivo è necessario un tempo molto più lungo per il rilascio dell'applicazione sullo store (a volte anche alcuni giorni).

3.1.3 Confronto tra architetture e caratteristiche

Come visto precedentemente sviluppare un'app nativa comporta molti benefici, sia che sia essa creata in Android che in iOS, solitamente le caratteristiche descritte precedentemente sono sufficienti per la scelta della metodologia da utilizzare (nel caso sia necessaria la creazione per una sola piattaforma). Tuttavia molto spesso si prendono decisioni anche basandosi su aspetti più tecnici quali l'architettura e il kernel, per questo motivo verranno riportate a grandi linee le architetture delle due famiglie.

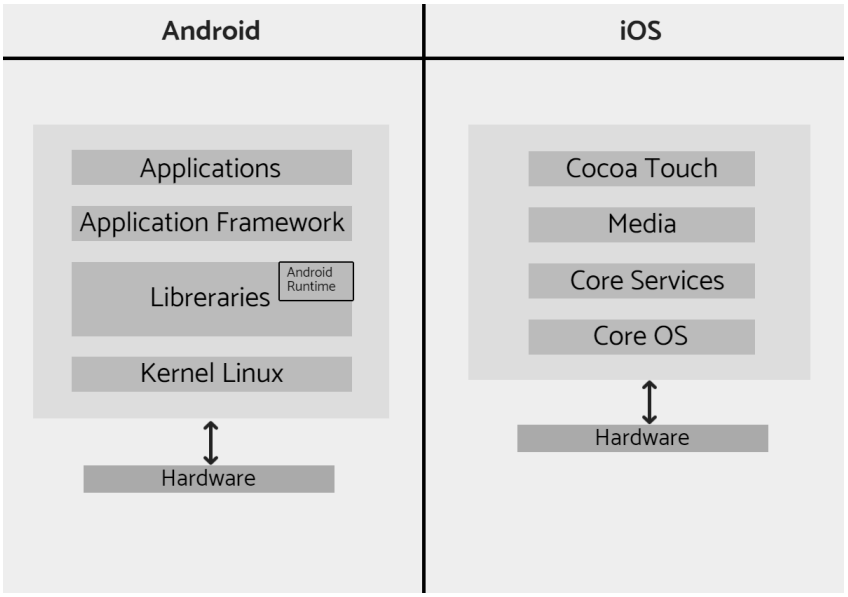


Figura 7: Architetture Android & iOS

Per quanto riguarda la parte destra dell'immagine è opportuno notare come Android utilizzi un kernel Linux e quindi open source alla base della sua architettura. Questo kernel contiene i driver essenziali per tutte le funzionalità come Audio, Camera, Gestione della batterie ecc. Subito sopra sono presenti le librerie open source tra cui SSL, SQLite e molte altre utilizzate per lo sviluppo. Molto importante da notare la presenza di una sezione chiamata Android Runtime in questo livello, essa contiene una Java Virtual Machine specificatamente creata per Android. Proprio per via di questa sezione è possibile scrivere delle app Android in Java grazie alle librerie fondamentali in essa contenute. Il livello Application Framework fornisce servizi ad alto livello che possono essere utilizzati dagli sviluppatori, mentre il livello più in alto consiste nella vera e propria applicazione installata sul dispositivo [21].

Uno dei principi su cui Android è stato basato è la robustezza, per questo motivo oltre all'utilizzo di un kernel Linux viene introdotto il concetto di indipendenza tra ogni differente applicazione Android. Identificata come "utente" ogni applicazione viene eseguita con un processo/istanza indipendente sulla JVM. Ogni "utente" è installato in una propria directory sul file system delimitando così dei confini virtuali e facendo in modo che un'app non influenzi il comportamento di un'altra e non si creino conflitti.

Sulla destra invece è possibile notare l'architettura iOS, anch'essa divisa in 4 livelli. Partendo dal fondo abbiamo il livello Core OS che anche in questo caso contiene le funzionalità di basso livello che sono necessarie alle applicazioni. Il livello Core Services offre molti modi per usufruire di alcuni servizi attraverso dei framework e fornisce interfacce per una moltitudine di integrazioni. Per concludere posiamo vedere il livello Media che offre supporto per l'utilizzo di audio/video/animazioni a livello avanzato con semplicità e il livello denominato Cocoa Touch che consiste in una collezione di framework chiave per lo sviluppo delle applicazioni iOS (in particolare per la definizione del layout dell'applicazione, la gestione delle funzionalità di multitasking, notifiche push e il rilevamento di input touch).

Come si nota quindi anche a livello architetturale c'è una sostanziale differenza, per poter avere un quadro generale è possibile affidarsi all'immagine sottostante che racchiude i principali pregi e difetti delle due modalità di sviluppo affrontate fino ad ora.

	Android	iOS
Vantaggi	Sistema aperto Frammentazione Tempi di rilascio Linee guida per Design	Guadagno Numero di dispositivi UI Design
Svantaggi	Numero di dispositivi Testing Costi variabili	Flessibilità Sistema chiuso Tempi di rilascio

Figura 8: Android vs iOS

3.2 App ibride multipiattaforma

In contrapposizione con le app native ci sono quelle multipiattaforma, che hanno come principale caratteristica il fatto di poter essere create per diverse piattaforme in una volta sola e scrivendo una sorta di codice universale.

La distinzione principale che è possibile fare all'interno di questa categoria è tra applicazioni ibride e applicazioni web, in questa sezione verranno trattate le prime perché strutturalmente sono una via di mezzo tra le app native e quelle web, anche se la nascita delle app ibride è avvenuta cronologicamente dopo quella delle altre.

Negli ultimi anni il concetto di applicazione ibrida ha assunto molti significati differenti e subito molte variazioni, quello che è sempre stato certo è che un'app ibrida cerca di racchiudere tutti i vantaggi delle applicazioni native e web in un'unica metodologia di sviluppo. In questo capitolo verrà indicata come applicazione ibrida un'app che viene sviluppata con un unico linguaggio di programmazione e in un ambiente diverso da quello nativo. Il risultato finale sarà un'insieme di applicazioni, ognuna differente per ogni piattaforma, distribuibili e ospitabili sui relativi store.

Le applicazioni ibride vengono create utilizzando principalmente due approcci, uno di questi viene chiamato Web Wrapper e consiste nella creazione di un nucleo scritto in HTML, CSS e JavaScript (linguaggi universalmente interpretabili dai browser, senza preoccupazione del dispositivo su cui vengono interpretati) e successivamente incapsulato all'interno di applicazioni native. In questo caso per poter avere accesso a tutte le funzionalità del dispositivo (come avviene automaticamente per le app native, data la loro natura) vengono utilizzati dei plugin appositi, questo metodo viene principalmente usato da Ionic. Il punto di forza di questa metodologia è quindi l'incapsulamento di un'applicazione web all'interno di una nativa che è dipendente dalla piattaforma su cui viene installata. Questo procedimento permette al nucleo dell'app di essere eseguito in un browser incorporato nell'applicazione nativa, invisibile all'utente finale, in modo da non avere una grafica come quando si naviga in rete, con una barra degli indirizzi e gli altri componenti [22]. Per poter visualizzare l'applicazione in questo modo nelle due principali piattaforme descritte per le app native è necessaria la creazione di una WebView per Android e di una WKWebView per i dispositivi ospitanti iOS. Molti dei framework e degli SDK attualmente utilizzati per la creazione di app ibride effettuano la compilazione e lo sviluppo di queste viste in cloud o in modo automatico autonomamente.

Il secondo metodo consiste nello sviluppare tutto utilizzando un framework o SDK che darà come risultato direttamente le applicazioni native traducendo il codice generato in codice nativo della piattaforma, come nel caso di Xamarin, React Native e Flutter.

I punti di forza delle app ibride sono il costo di sviluppo (che è inferiore rispetto alla creazione della stessa app nativa su più piattaforme), la rapidità con cui

possono essere create per diverse piattaforme simultaneamente e il fatto che essendo un unico codice è molto facile aggiornare o apportare modifiche in un unico momento per tutte le piattaforme. Inoltre le app ibride possono in certi casi anche lavorare offline con qualche accorgimento.

Molte delle funzionalità proprie dei dispositivi possono quindi essere sfruttate grazie all'utilizzo di plugin o servizi offerti da librerie come API. Questi strumenti forniscono interfacce e possibilità per utilizzare strumenti come GPS, Touch Screen, Bluetooth, Gestures e vi dicendo. Tuttavia, trattandosi di applicazioni che sono sviluppate con linguaggio differente da quello nativo, si vengono a creare alcuni svantaggi che invece nelle app native non sono presenti. In primo luogo la User Experience e le performance non sono ai livelli di un'applicazione nativa, ma inferiori e poi spesso nello sviluppo di applicazioni ibride si incorre in alcuni problemi che sono ricorrenti nello sviluppo nativo, per questo motivo in un team di sviluppo di app ibride vi è spesso la necessità di avere uno sviluppatore nativo [23].

In questa sezione verranno approfonditi maggiormente alcuni framework o SDK che si possono utilizzare per la creazione di applicazioni ibride nelle due tipologie descritte prima, cercando di evidenziarne le principali caratteristiche per poter avere un quadro generale nel caso in cui sia necessario effettuare una scelta. Facendo parte della stessa macro categoria le differenze di performance o tempi e costi sono davvero minime, quindi spesso la scelta viene effettuata a seconda del linguaggio utilizzato, degli IDE e degli strumenti che lo sviluppatore preferisce e con cui ha esperienza o pensa di trovarsi meglio.

3.2.1 Xamarin

Il metodo che la Microsoft suggerisce e supporta maggiormente per la creazione di applicazioni ibride è Xamarin, una piattaforma open source utilizzabile in combinazione con Visual Studio. Trattandosi di un livello di astrazione il suo compito è la comunicazione tra il codice condiviso e la piattaforma sottostante. Le funzionalità native dei diversi sistemi operativi mobile vengono esposte come classi .NET, le quali consentono un accesso completo alle API e ai comandi nativi, per avvicinarsi il più possibile alle applicazioni native [24].

Sviluppando con Xamarin si hanno a disposizione una serie di emulatori iOS e Android forniti dall'IDE per poter eseguire le applicazioni una volta ultimate, per poterle testare ed eventualmente eseguire il debug. I linguaggi che attualmente sono supportati e che possono essere utilizzati per uno sviluppo con Xamarin sono C# e F#. Essendoci un'integrazione con Visual Studio è possibile utilizzare una serie di strumenti in esso contenuti utili per sviluppare, come per esempio il designer per creare interfacce grafiche oppure IntelliSense per la documentazione delle API native e i nomi delle variabili.

Xamarin offre un prezioso strumento chiamato Xamarin.Forms, che è possibile utilizzare per creare delle interfacce grafiche singole che siano condivisibili su tutte le piattaforme, questo è dato dal fatto che compilando le soluzioni create in questo modo viene generata in un colpo solo un'applicazione per ogni piattaforma. In caso non si volesse approfittare di questa possibilità, come specificato dalla Microsoft [25], è possibile condividere tutta la parte di codice che non sia un'interfaccia grafica e sviluppare diverse UI per ogni piattaforma desiderata per poi integrarle con il codice condiviso.

Un altro strumento fondamentale che mette a disposizione Xamarin è Xamarin.Essentials, una libreria che fornisce le API necessarie per poter utilizzare al meglio alcune delle funzionalità del dispositivo (accelerometro, sblocco schermo, file system ecc).

Le applicazioni Xamarin possono essere compilate in file .apk (estensione per le applicazioni Android), file .ipa (estensione per le applicazioni iOS). Nell'immagine sottostante è possibile vedere il funzionamento di Xamarin con i due principali sistemi operativi mobile.

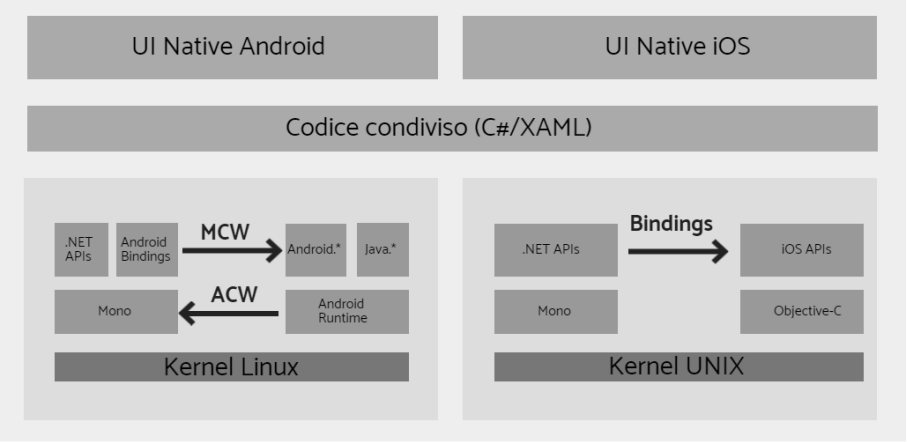


Figura 9: Integrazione con Xamarin

Per quanto riguarda il funzionamento con Android si ha che il codice viene compilato inizialmente da C# a un linguaggio chiamato IL (Intermediate Language), il quale a sua volta viene compilato in assembly nativo quando viene lanciata l'applicazione. Come specificato dalla Microsoft [25], le applicazioni Xamarin per Android vengono eseguite all'interno dell'ambiente di esecuzione Mono (creato appositamente per ospitare Xamarin e sfruttare .NET), affiancato dalla macchina virtuale che è presente nella zona di runtime (come si può vedere dall'architettura Android sopra descritta).

Grazie ai collegamenti forniti da .NET è possibile raggiungere gli spazi dei nomi Android.* e Java.* attraverso dei specifici collegamenti chiamati Managed Callable Wrappers (MCW). Mono offre anche la possibilità di essere raggiunto dalla zona di runtime con delle chiamate, identificate dal nome Android Callable Wrappers (ACW).

Il funzionamento con iOS è leggermente semplificato, viene sfruttato da mono il concetto di compilazione anticipata da C# a codice ARM assembly. La comunicazione tra C# e Objective-C avviene attraverso dei Bindings, termine con cui vengono identificati i Selectors e i Registrars (utilizzati per la comunicazione in una direzione e nell'altra).

3.2.2 React Native

React Native è un framework open source creato da Facebook per lo sviluppo di applicazioni mobili e per dare la possibilità di utilizzare ReactJS nella creazione di applicazioni native. Il codice viene principalmente scritto in JavaScript e successivamente renderizzato in codice nativo. Anche in questo caso sarà possibile pubblicare l'app finale sui vari store presenti nei dispositivi mobile per la distribuzione.

Il punto di forza di React Native è la User Interface, avere interfacce responsive chiare, efficienti, intuitive e visibilmente belle è uno dei pilastri portanti di questo framework. React Native può anche vantare una delle migliori community nel settore dello sviluppo multipiattaforma.

Questa metodologia si basa sull'avere dei componenti che racchiudano il codice nativo esistente e che comunichino con le API native attraverso JavaScript e i paradigmi dell'interfaccia utente dichiarativa di React [26]. Questo significa che quando l'applicazione verrà creata verranno utilizzati gli elementi nativi grafici del sistema operativo mobile. Il fatto di mappare dei componenti grafici JavaScript con i componenti nativi del sistema operativo costituisce la chiave per il funzionamento di React Native. I componenti utilizzati nello sviluppo nativo quindi vengono chiamati Native Component, a differenza di quelli utilizzati in ReactJS chiamati Web Component.

La creazione di applicazioni React è molto flessibile per quanto riguarda l'ambiente di sviluppo, è possibile utilizzare molti IDE tra cui Atom e Visual Studio Code in maniera rapida ed efficiente. Per poter programmare con React è necessario utilizzare Node.js, e quindi dei gestori di pacchetti quali Npm o Yarn. Il metodo consigliato principalmente per la creazioni di app in questo moto è attraverso l'utilizzo di Expo, per poter creare, sviluppare e pubblicare rapidamente un'applicazione nativa in React [27].

In React native è anche possibile scrivere del codice usando Objective-C/Swift o Java per poter creare dei componenti nativi e integrarli nel caso non siano presenti quelli necessari, altrimenti solitamente si usano quelli esistenti.

Per poter capire il funzionamento di React native è opportuno dare un'occhiata all'architettura su cui si basa. Come si può notare dalla figura sottostante essa è strutturata su 4 livelli principali interconnessi tra di loro.

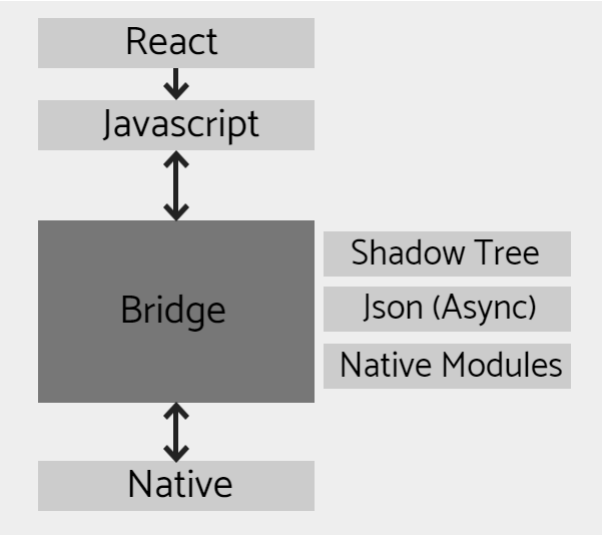


Figura 10: Integrazione con React Native

I livelli React, JavaScript e Native sono facilmente intuibili perché rappresentano rispettivamente la parte di codice scritta in React con i vari componenti, la parte tradotta in JavaScript di questo codice e la parte nativa del sistema operativo mobile.

Il cuore di questa architettura è ciò che è identificato con Bridge, che permette un collegamento tra la parte nativa e quella JavaScript. Questo componente permette la comunicazione asincrona e bidirezionale tra le due parti grazie all'utilizzo di messaggi JSON ed è creato con linguaggi C/C++ e quindi multipiattaforma. Lo Shadow Tree invece ha il compito di creare l'albero di elementi che costituiscono l'interfaccia grafica attraverso i messaggi ricevuti. Il funzionamento di React Native quindi sta principalmente nell'integrazione di questi componenti e nella loro connessione, i quali ne rendono possibile il funzionamento.

3.2.3 Flutter

Google ha lanciato Flutter come ambiente di sviluppo basato su un nuovo linguaggio di programmazione chiamato Dart. Flutter è un Software Development Kit (SDK) e non solo un framework. Esso permette la creazione di applicazioni utilizzando qualsiasi editor di testo, purché combinato con lo strumento di linea di comando offerto dalla Google per lo sviluppo di app ibride.

Per quanto riguarda il funzionamento di Flutter, le applicazioni scritte con il Framework in Dart vengono passate al motore per la compilazione anticipata (scritto in C, C++, Dart e Skia) che si interfaccia direttamente con la piattaforma di destinazione, è possibile avere un quadro generale osservando l'immagine sottostante.

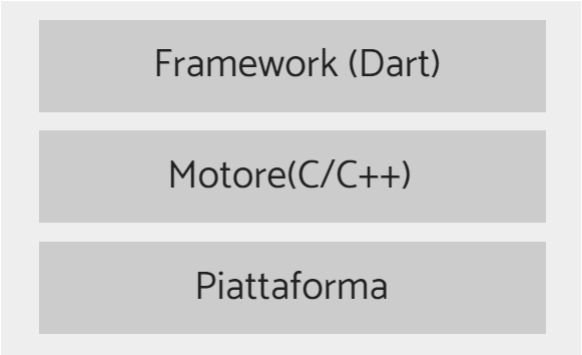


Figura 11: Integrazione con Flutter

Il motore ha un ruolo principale in questa architettura e si occupa di fornire un supporto per il rendering a basso livello utilizzando delle librerie grafiche di Google chiamate Skia. Esso si interfaccia con gli SDK nativi delle piattaforme Android e iOS permettendone un facile collegamento.

Come nei due casi precedentemente citati anche qui una volta che la creazione dell'applicazione è terminata viene messo in atto un procedimento per la pubblicazione negli store. A differenza di Xamarin, in entrambi i casi viene utilizzata una compilazione anticipata (Ahead Of Time: AOT), per via del fatto che iOS non supporta la compilazione Just In Time (JIT).

L'organizzazione di Flutter si basa sul concetto di Widget, con cui si identificano i blocchi base delle UI create con questo SDK, per via di questi elementi non vengono utilizzati componenti dell'interfaccia nativa. La divisione principale identifica Widget Stateless e Statefull, che si differenziano appunto per la presenza o meno dello stato. Tutti i Widget vengono creati in Dart e quindi fanno parte del primo livello dell'architettura (il framework), per poterne creare di più complessi spesso molti Widget vengono raggruppati e connessi [28].

3.2.4 Ionic

Ionic è un SDK open source che viene utilizzato per lo sviluppo di applicazioni ibride, si basa sulla creazione di un nucleo stile web app successivamente incapsulato in un wrapper, in modo da per poter utilizzare il tutto come se fosse un'applicazione nativa.

I componenti Ionic vengono scritti con linguaggi universalmente comprensibili dai browser come HTML, CSS e JavaScript [29], questo favorisce la loro creazione e comprensione essendo linguaggi ampiamente conosciuti nel mondo delle web app. Anche in questo caso per poter sviluppare applicazioni in Ionic è possibile utilizzare qualunque editor in grado di supportare ES6 e TypeScript, come Atom e Visual Studio Code.

Le applicazioni Ionic inizialmente venivano create con Cordova che ha appunto il compito di impacchettare il codice scritto e fornire dei plugin per l'accesso alle funzionalità native nel caso necessario. Attualmente nella maggior parte dei casi viene utilizzato Capacitor che crea il Wrapper/ponte tra l'app scritta in tecnologie web e il dispositivo nativo (Capacitor è stato creato da Cordova e quindi ne sfrutta la maggior parte degli aspetti) [30]. Nell'immagine sottostante è possibile vedere l'architettura che viene utilizzata da Ionic per la creazione di applicazioni mobile sfruttando Cordova.

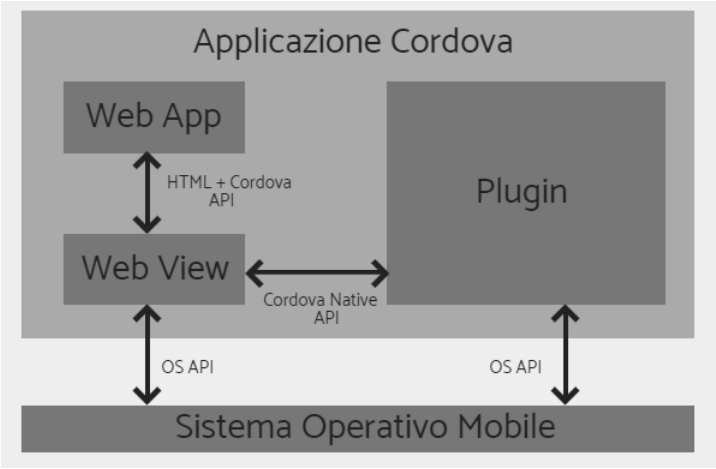


Figura 12: Integrazione con Ionic

Come si vede nell'immagine è necessario l'utilizzo di plugin e di una Web View per poter rendere possibile la scrittura applicazioni ibride con i linguaggi di programmazione web. La comunicazione con il sistema operativo avviene attraverso delle API gestite da Cordova/Capacitor.

3.2.5 Quadro generale dello sviluppo ibrido

Come detto precedentemente, spesso in questo caso la scelta viene effettuata per via dei linguaggi di programmazione o editor più familiari al programmatore. Oltre a una visione globale per quanto riguarda il funzionamento di ogni Framework o SDK, è opportuno confrontare le varie caratteristiche insieme per poter effettuare una scelta in maniera semplice. Nella seguente tabella verranno riportati gli aspetti precedentemente descritti e alcuni paragoni aggiuntivi tra di loro tratti da un articolo molto dettagliato presente su Apptunix [31].

	Xamarin	React Native	Flutter	Ionic
Tipologia	Framework	SDK	Framework	Web Wrapper
Linguaggi di programmazione	C# (+ ambiente .NET)	Javascript (+ Swift/Objective-C o Java)	Dart	HTML5, CSS e Javascript + Typescript
IDE Suggeriti	Visual Studio	Visual Studio Code Atom	Visual Studio Code Atom (+ Google CLI)	Visual Studio Code Atom
Interfaccia Grafica	Utilizza controllers delle UI native	Utilizza controllers delle UI native	Widgets proprietari	HTML + Css
Tipologia di software	Open-source (Funzionalità migliori a pagamento)	Open-source	Open-source	Open-source (Funzionalità migliori a pagamento)
Riusabilità del software	96% del codice	90% del codice	50-90% del codice	98% del codice
Performance	Medie utilizzando forms Quasi native senza	Quasi native	Massime in ambito ibrido	Moderate
Casi d'uso	App semplici	App di qualunque complessità	App di qualunque complessità	App semplici
Supporto della Community e mercato	Forte	Migliore in circolazione	Non molto popolare	Forte
Esempi	Molte app per le banche	Facebook e Instagram	Hamilton: The Musical	Just watch

Figura 13: Confronto metodologie di sviluppo ibrido

Nella figura sopra è possibile vedere dieci aspetti fondamentali per cui si differenziano le metodologie di sviluppo ibrido e il loro confronto. La scelta su quale metodologia adottare può essere effettuata prendendo in considerazione queste caratteristiche. Vengono anche riportate alcune delle app ibride più famose sviluppate con le relative metodologie, per dare un'idea delle potenzialità degli strumenti forniti.

3.3 Web App multipiattaforma

Le Web App fanno parte della famiglia delle applicazioni multipiattaforma, in contrapposizione a quelle native, precedentemente è stato illustrato il funzionamento delle applicazioni ibride, che sono una via di mezzo tra le web app e le quelle native. Le applicazioni web differiscono dalle altre tipologie in quanto sono dei veri e propri siti, i quali sono perfettamente responsive e possono essere consultati da dispositivi mobili con dei layout molto simili a quelli delle applicazioni native.

Con web app si intende un'applicazione distribuita funzionante su piattaforme differenti (anche desktop), è possibile fare una lieve distinzione introducendo il concetto di Mobile Web Application, che racchiude soltanto lo sviluppo di applicazioni per dispositivi mobili, in questo capitolo si farà riferimento a web application eludendo alla parte mobile, di maggiore interesse in quest'analisi.

Queste app vengono eseguite su un server ed è quindi essenziale una connessione internet costante per garantirne il corretto funzionamento. Le applicazioni web vengono raggiunte e utilizzate attraverso i browser (essendo appunto dei siti web), questo le rende indipendenti dalla piattaforma su cui vengono eseguite, sia anche essa mobile o non. Il principio di funzionamento di queste app consiste nel fatto che chi naviga sul sito si comporta da client, chiedendo implicitamente di poter usufruire di alcuni servizi offerti dal server su cui risiede l'applicativo.

Per creare un'applicazione web vengono quindi utilizzati i linguaggi interpretabili dai browser: HTML, CSS e JavaScript principalmente, essendo questi molto conosciuti rappresentano un vantaggio nello scegliere questa metodologia di sviluppo. Il fatto di essere raggiungibili dal browser elimina l'obbligo di installazione sul dispositivo dallo store (che spesso è molto lenta) e la necessità di aggiornamenti da effettuare e scaricare (dato dal fatto che in tempo reale il sito web è aggiornato all'ultima versione). Dal punto di vista economico i costi per la creazione di un'applicazione web sono più bassi rispetto a qualunque altra metodologia, questo per via della natura e semplicità del codice da scrivere, che richiede meno sforzo e tempo [32].

Sviluppare usando un approccio web porta anche alcuni svantaggi tra cui la necessità di avere un responsive layout perfetto per ogni dispositivo possibile, perciò ci sarà da investire molto in questo aspetto. Ovviamente trattandosi di un'applicazione risiedente su un server non sarà possibile utilizzarla se non si avrà una connessione costante. Come ultimi aspetti da tenere in considerazione, anche se non meno importanti, vi sono il fatto che è possibile che alcune funzionalità del dispositivo non funzionino correttamente in alcuni casi (riducendo così la User Experience) e il fatto che essendo presenti sui siti web e non pubblicizzate negli store la visibilità delle applicazioni web è nettamente inferiore rispetto a quelle ibride o native.

Per sviluppare applicazioni web vengono utilizzati alcuni framework molto popolari, alcuni dei principali verranno presentati e analizzati in questo capitolo, gli stessi sono anche adattati e aggiornati per supportare lo sviluppo di applicazioni web progressive, introdotte nella sezione seguente.

3.3.1 PWA

Negli ultimi tempi si è diffuso molto il concetto di Progressive Web Application (PWA), con questo termine si intende un nuovo modo rivoluzionario di sviluppare web, che si basa sul concetto di fornire all'utente una navigazione sul sito come se stesse utilizzando un'app nativa, per poi dare la possibilità di installare l'applicazione direttamente dal sito in maniera semplice e rapida.

Le PWA sono una tipologia di sviluppo di applicazioni a parte rispetto alle web app, come le ibride e le native, ma dato che per svilupparle vengono usati gli stessi framework delle web app (a cui sono stati integrati i moduli necessari) verranno trattate nella stessa sezione. Anche alcuni framework per lo sviluppo di applicazioni ibride (come Ionic) si sono adattati per fornire supporto nella creazione di app progressive e sono molto validi ed utilizzati.

Le progressive web app oltre che cercare di fornire all'utente un'esperienza molto vicina a quella delle app native forniscono la possibilità di lavorare offline, cosa che inizialmente nelle app ibride non era pensata (anche se poi si sono evolute). Le app progressive sono caratterizzati da due aspetti fondamentali [33]:

- **Manifest**: questo particolare file viene inserito nella root del progetto, è in formato JSON e contiene degli script e delle descrizioni relativi all'applicazione. Grazie a questo file l'app può essere scaricata e si potrà accedere ad essa come fosse un'app nativa. dato che verrà creata un'icona nella pagina principale del dispositivo uguale a quella delle app native, oltre a fornirne un layout identico. Nel file sono anche contenute informazioni riguardante l'autore, la versione, l'icona, la descrizione e la lista di tutte le risorse necessarie per il corretto funzionamento.
- **Service Worker**: è un file JavaScript che viene eseguito in un thread differente da quello principale su cui l'applicazione viene eseguita nel browser. Ha il compito di intercettare le richieste, effettuare operazioni di cache (garantendo il funzionamento offline) e mandare notifiche push. Viene pensato e implementato per essere completamente asincrono, non può accedere direttamente al DOM ma utilizza dei messaggi di tipo post e garantisce sicurezza, dato che viene eseguito con HTTPS. Funge da tramite ponendosi tra la pagina web e il server/servizio.

Come detto precedentemente i framework utilizzati per la creazione delle PWA sono praticamente gli stessi utilizzati per le web app (integrati con strumenti che offrono la possibilità di usufruire e implementare Manifest e Service Worker), i principali verranno analizzati nelle seguenti sezioni.

3.3.2 ASP.NET Core e Blazor

La Microsoft mette a disposizione un framework per la creazione di applicazioni web open source chiamato ASP.NET Core, successore di ASP.NET. Uno dei componenti di questo framework è Blazor, a sua volta definito come framework per la creazione di web app utilizzando C# e HTML.

Blazor è uno strumento in continua evoluzione e crescita, nella creazione delle web app nel mondo Microsoft è sicuramente un pilastro portante. Il funzionamento di Blazor permette la creazione di applicazioni Single Page, uno dei metodi principalmente usati attualmente nella creazione di web app.

Blazor deriva dall'accostamento di Browser e Razor, il secondo è un sistema di scripting lato server creato per generare e gestire pagine web dinamiche, grazie a Razor è possibile scrivere in C# e Visual Basic incorporando il codice scritto nelle pagine web.

Oltre che poter utilizzare tutto il potenziale di Razor è anche possibile usufruire degli altri strumenti forniti da ASP.NET Core per sviluppare in questa tecnologia, come per esempio EntityFramework (EF) per la gestione e l'interazione con database SQL.

Viene consigliato di sviluppare app in Blazor utilizzando Visual Studio che offre un completo supporto per questa tecnologia, anche se è possibile utilizzare altri editor. Sviluppando con questa metodologia è possibile utilizzare C# e ASP.NET sia lato client che lato server. Una volta sviluppato il codice occorre effettuare un'operazione di host, in base alla metodologia usata si utilizzerà un approccio differente.

Lo sviluppo di applicazioni Blazor si può dividere in due grandi categorie, esse utilizzando dei metodi completamente differenti che dipendono dall'esecuzione e dalla comunicazione:

- **Blazor Server**: l'applicazione è eseguita nel server all'interno di un'app ASP.NET Core e attraverso l'utilizzo di alcuni segnali chiamati SignalR avviene la comunicazione tra browser e server. I SignalR servono per aggiornare gli elementi della UI ma necessitano di un costante supporto internet, questa metodologia non può lavorare offline.
- **Blazor Webassembly**: il browser esegue il modello principale per Blazor, inoltre vengono scaricare le varie dipendenze e il Runtime .NET nel browser. La gestione degli eventi e l'aggiornamento dell'interfaccia utente avviene all'interno di un unico processo. Le risorse dell'app avvengono distribuite sotto forma di file statici attraverso dei Server Web o attraverso un servizio che ospita contenuto statico ai client. Grazie a questa metodologia è possibile fornire supporto offline non necessitando di comunicazione continua con il server per modificare la pagina.

Nell'immagine seguente è possibile vedere a confronto le due metodologie e la loro modalità di comunicazione:

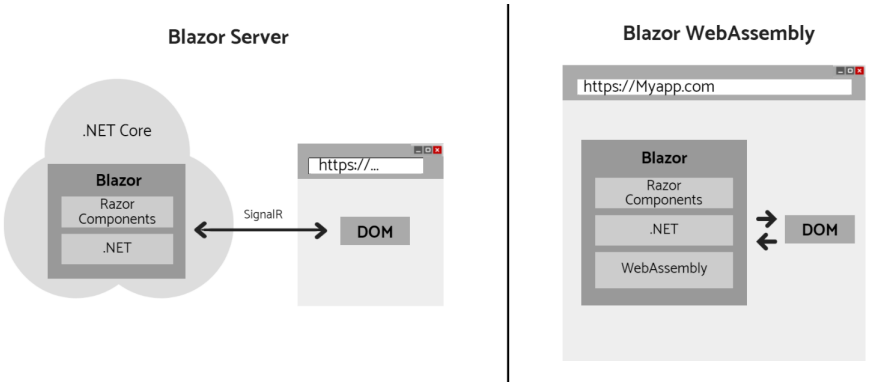


Figura 14: Blazor Server vs WebAssembly

Se si opta per un approccio WebAssembly si avrà un download più pesante all'inizio perché è necessario avere tutto sul lato client per poter garantire un corretto funzionamento, nel caso di Blazor Server invece il download è notevolmente ridotto. Nel primo modo si utilizzano a pieno le funzionalità del server e tutte le API di .NET sono disponibili, mentre nel secondo si utilizza al massimo la capacità di lavoro del client dato che tutto il lavoro viene scaricato su di esso.

Inoltre nel caso di WebAssembly l'app è limitata a quello che il browser può fornire e per poter effettuare un'operazione di host per questo tipo di app sono necessari supporti idonei e software client. Essendo una tecnologia molto nuova il supporto su Webassembly rispetto a Server è molto ridotto. Tuttavia per un'app gestita nel client non necessario per forza usare un server web di tipo ASP.NET Core ma si più usare anche una Content Delivery Network [34].

3.3.3 Angular con Express e Node, nasce lo stack MEAN

AngularJS è un framework open source creato dalla Google per offrire la possibilità di sviluppare applicazioni web e Single Page Application. Uno dei pilastri su cui si basa è l'architettura MVW (derivata dalla classica Model View Controller) che sta a significare Model View Whatever e che ha come scopo il dividere senza vincoli la logica di business dalla prestazione [35].

Questo framework è basato su JavaScript ma solitamente se ne utilizza una sua evoluzione chiamata TypeScript, esso offre funzionalità avanzate già a lato client senza necessità della parte server, se non in casi strettamente necessari. AngularJS viene impiegato nella scrittura della parte front-end e grazie a delle chiamate è possibile comunicare con il server quando necessario.

Non essendo un framework full-stack per poterlo utilizzare come tale è necessario avere qualcosa che a lato server si interfacci, una delle scelte migliori è Node.js che serve per l'installazione di dipendenze e per eseguire il web server locale preconfigurato. Nel caso si utilizzi Node.js come piattaforma lato server un valido framework da utilizzare per la gestione del back-end è Express.

Node, Angular ed Express sono una buona combinazione di framework per lo sviluppo web utilizzando JavaScript su entrambi i lati e possono lavorare bene con database SQL, la comunicazione avviene in maniera asincrona data la natura di Node.js, mentre le chiamate avvengono solitamente utilizzando AJAX per via dell'utilizzo del modello Single Page application.

Con l'avvento dei database non relazionali e l'introduzione di questi framework è nata un'idea di sviluppare mettendo insieme questi tre strumenti con MongoDB, dando vita così allo stack MEAN (MongoDB + Express + Angular + Node). Nell'immagine seguente è possibile vedere una rappresentazione del MEAN stack:

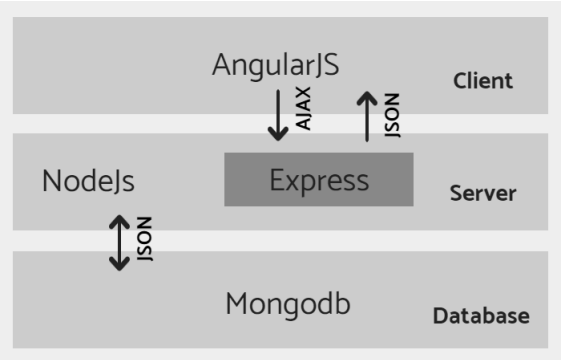


Figura 15: MEAN Stack

3.3.4 React con Express e Node, nasce lo stack MERN

ReactJS non è un framework ma una libreria JavaScript per la creazione di interfacce utente sviluppata da Facebook, anche in questo caso quindi siamo in presenza di un metodo per sviluppare front-end. Tale libreria si basa sul concetto dei componenti ed è dichiarativa. L'utilizzo di JSX (un'evoluzione del JavaScript) permette di produrre elementi React mentre la separazione dei concetti e delle responsabilità avviene grazie all'utilizzo dei componenti.

Questa libreria può essere usata come base per la creazioni di applicazioni single page, una delle sue caratteristiche è che si occupa solamente del rendering dei dati sul DOM, data la sua natura sono necessarie altre librerie che siano in grado si gestire lo stato e il routing. Uno dei pilastri di React è l'utilizzo di un DOM virtuale, che permette un aggiornamento efficiente ed istantaneo della vista sul browser una volta apportate le modifiche [36].

Come nel caso di Angular spesso viene accostato a un framework back-end per la creazione di applicazioni full stack, uno degli accostamenti migliori anche in questo caso è Node.js con Express. Ovviamente non è l'unica opzione, si potrà infatti usare un framework back-end che utilizza linguaggi differenti dal JavaScript lato server, come quelli illustrati a fine sezione.

La creazione di applicazioni con questi framework semplifica l'integrazione con tutti i tipi di database, relazionali e non. Nel caso si decida di optare per la seconda opzione e di utilizzare sempre MongoDB, è possibile sviluppare utilizzando uno stack che viene chiamato MERN, appunto dal nome (MongoDB + Express + React + Node). Nella figura sottostante è riportato uno schema raffigurante il MERN stack:

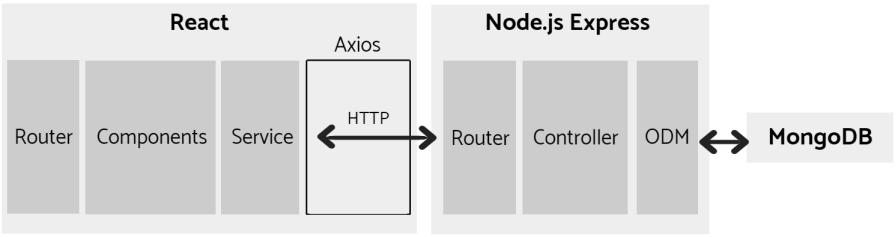


Figura 16: MERN Stack

Nell'immagine è possibile vedere in React la libreria Axios che permette la comunicazione asincrona, il React Router che è una collezione di componenti di navigazione che costituiscono in modo dichiarativo l'applicazione e il Service che rappresenta appunto il servizio. In Node invece è possibile vedere il Controller che intercetta le chiamate e l'ODM (Object Document Mapper) per gestire JSON al volo (tipicamente Mongoose se ci si interfaccia con MongoDB).

3.3.5 Vue e Spring

Vue.js è un framework progressivo per la creazione di interfacce utente sviluppato da Evan You ed è facilmente utilizzabile per lo sviluppo di Single Page Application se utilizzato con strumenti moderni e supportato con librerie. Alla base di questo framework vi è la configurazione MVVM (Model-View-ViewModel) a tre livelli, un modello di sviluppo software.

Questo framework utilizza dei componenti Vue (che sono elementi HTML) per incapsulare del codice riutilizzabile a cui il compilatore associa una funzionalità. I modelli utilizzati da Vue sono basati su HTML e comprensibili dai browser, i quali una volta compilati divengono funzioni di rendering DOM virtuali. Può anche essere utilizzato JSX per scrivere direttamente le funzioni di Rendering [37].

Anche questo Framework è scritto in JavaScript e utilizza questo linguaggio, che è uno dei più popolari nel mondo dello sviluppo web attualmente. Per poter creare un'applicazione full stack è possibile usare uno dei tanti framework per lavorare lato server come Node.js trattato nei casi precedenti, ma per esplorare un approccio differente verrà mostrata l'integrazione con Spring, un framework per lavorare lato server con Java.

Nell'immagine sottostante è possibile vedere un'integrazione di Vue.js lato client con Spring lato Server e il database:

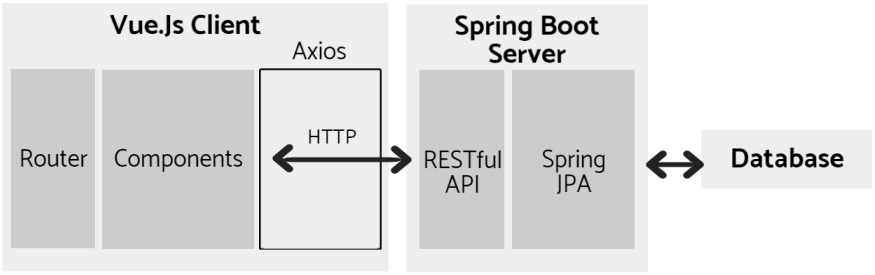


Figura 17: Vue.Js e Spring

Come si nota dalla figura, anche in questo caso viene utilizzata la libreria Axios per comunicare in maniera asincrona. Vengono utilizzati lato client i componenti come detto prima e anche il Router che è necessario per le single page applications. Nel lato Server si nota che viene usato il framework spring boot che semplifica notevolmente la creazione di web application. Sempre su questo lato vengono raggiunte delle API create seguendo la logica REST (RESTful API) che interfacciandosi con quelle Java (Java Persistent API) comunicano con il database.

3.3.6 Altri framework per lo sviluppo web

Come si è notato precedentemente nella spiegazione di alcuni dei Framework per lo sviluppo web, è possibile sceglierne diversi ed integrarli insieme per creare un'applicazione full stack dato che molti di essi forniscono solamente supporto o lato server o lato client.

Precedentemente sono stati descritti (oltre a Blazor con .NET) React, Angular e Vue perché utilizzano tutti JavaScript, che attualmente è uno dei linguaggi più diffusi, con le relative varianti. Esistono tuttavia anche altri framework/librerie degni di nota che utilizzano linguaggi differenti:

- **Laravel**: un framework per creare applicazioni web derivato da Symfony e ideato da Taylor Otwell. Segue il modello MVC ed è il più popolare framework in PHP. Presenta anche un'interfaccia a linea di comando chiamata Artisan ed è in continuo aggiornamento. Con un sistema di gestione di pacchetti modulare e un gestore delle dipendenze dedicato viene utilizzato principalmente per programmazione lato server [38].
- **Ruby on Rails**: chiamato anche RoR è un framework open source scritto in linguaggio Ruby. Creato da David Heinemeier Hansson utilizza un modello MVC e viene principalmente usato lato server [39].
- **Django**: un framework in Pyton lato server open source per lo sviluppo di applicazioni web seguendo il modello MVC, è stato sviluppato dalla DSF (Django Software Foundation) [40].
- **CodeIgniter**: come Laravel si tratta di un framework in PHP per la creazione di applicazioni web seguendo un modello MVC. Si tratta di un prodotto open source sviluppato da EllisLab, compatto e veloce nel funzionamento [41].
- **jQuery**: una delle più diffuse librerie JavaScript open source per le applicazioni web che si occupa di manipolare e gestire gli eventi e animare gli elementi del DOM in pagine HTML [42].
- **Bootstrap**: framework che consiste in un insieme di modelli di progettazione per la creazione di siti e applicazioni, utilizzato principalmente per la sua utilità in ambito responsive è una delle librerie più diffuse attualmente nello sviluppo web lato client. I modelli di Bootstrap interpretabili dal browser sono scritti in HTML CSS e JavaScript principalmente [43].

Molti di questi strumenti vengono utilizzati attualmente per lo sviluppo e la gestione delle maggiori applicazioni web sul mercato e sono in continua evoluzione. Come detto a inizio capitolo anche per la creazione di progressive web applications si possono utilizzare molti di questi strumenti.

3.3.7 Quadro generale dello sviluppo web e PWA

Quando si decide di utilizzare un framework web o una combinazione di essi è necessario conoscere la tipologia di sviluppo che esso offre (Front-end o Back-end) e i linguaggi utilizzati.

Uno degli aspetti da tenere anche in considerazione nella scelta è la necessità di esigenze particolari da parte del cliente, che devono essere soddisfatte. Per questa ragione è necessaria un'attenta analisi dei requisiti dell'applicazione e una ricerca/studio approfondito per scegliere il framework migliore che soddisfi tutti i requisiti.

Alcuni framework utilizzano maggiormente delle architetture specifiche o alcuni modelli di design particolari, nella figura sottostante vengono riportati quelli prevalentemente adottati in ogni framework, anche se a seconda delle necessità possono variare.

	Blazor	ASP.NET Core	AngularJs	ReactJs	NodeJs
Tipologia	Framework Front-end	Framework Back-end	Framework Front-end	Libreria Front-end	Framework Back-end
Linguaggi di programmazione	C# HTML	Visual Basic, C#, C++ e Java	TypeScript (JavaScript)	JSX (JavaScript)	JavaScript
Modello maggiroremte utilizzato	MVVM	MVC	MVW	–	MVC

	Express	Spring	VueJs	Laravel	Ruby On Rails
Tipologia	Framework Back-end	Framework Back-end	Framework Front-end	Framework Back-end	Framework Back-end
Linguaggi di programmazione	JavaScript	Java	JavaScript	PHP	Ruby
Modello maggiroremte utilizzato	MVC	MVC	MVVM	MVC	MVC

	Django	CodeIgniter	Jquery	Bootstrap
Tipologia	Framework Back-end	Framework Back-end	Libreria Front-end	Framework Front-end
Linguaggi di programmazione	Pyton	PHP	JavaScript	HTML CSS JavaScript
Modello maggiroremte utilizzato	MVC	MVC	–	MVC

Figura 18: Confronto metodologie di sviluppo web

Nell'ambito delle PWA alcuni dei framework utilizzati e ampiamente supportati nel 2020 sono Angular, Vue e React. Con l'integrazione di appositi moduli necessari viene offerta la possibilità di sviluppare applicazioni progressive in modo rapido ed efficiente potendo contare su un ottimo supporto da parte della community.

3.4 Quale metodologia scegliere

Dopo aver analizzato nel dettaglio le caratteristiche di tutte le metodologie di sviluppo con le relative varianti è possibile avere un quadro accurato sullo sviluppo web. In generale se si vuole sviluppare un'applicazione mobile e si hanno i fondi e i tempi sufficienti è sempre buono sviluppare con tecnologie native, per via dell'affidabilità di queste metodologie e del beneficio che portano garantendo la massima efficienza e le migliori performance.

Quando è necessario sviluppare un'applicazione solo mobile ma con tempi e costi ridotti, indipendentemente dal linguaggio o dagli editor, spesso la soluzione ibrida è quella che più viene incontro a queste esigenze, offrendo delle buone performance e dando la possibilità di creare applicazioni quasi native.

Nel caso invece che si voglia sviluppare un'applicazione mobile ma che si desideri anche avere una controparte desktop (magari per l'utilizzo da parte di personale differente, in circostanze e luoghi diversi) allora sicuramente lo sviluppo web è la scelta migliore per via della natura delle applicazioni create, comodamente accessibili da tutti i dispositivi dotati di un browser.

Un fattore fondamentale che ricopre un ruolo rilevante nella scelta è la necessità o meno di avere la presenza della rete, in alcuni luoghi è possibile che la connessione non sia costante e in questo caso optare per uno sviluppo web risulterebbe una pessima scelta. Questo requisito può essere facilmente accomodato utilizzando uno sviluppo nativo o alcune delle metodologie ibride.

In alcuni casi è possibile che sia necessario sviluppare un'applicazione web che si interfacci con dei dispositivi particolari, magari non più in produzione, tipo dispositivi mobili con Windows Phone o alcuni terminali che sfruttano tecnologie e sistemi operativi particolari. In questo caso è necessario verificare che il framework o la metodologia scelta offra la possibilità di sviluppare con queste restrizioni. Molti dei framework ibridi elencati non sono compatibili con smartphone che presentano il sistema operativo di Windows.

Nel caso di sviluppo web è possibile che venga richiesta solamente una nuova interfaccia lato client e quindi non si possa modificare lo sviluppo back-end e si sia limitati nell'utilizzare un determinato database (le metodologie che prevedono database non relazionali spesso sono difficili da utilizzare se in presenza di un database SQL), questo rende la scelta molto limitata.

Se non sono presenti esigenze particolari per via dei fattori elencati sopra, sta al team di sviluppo scegliere il framework migliore una volta decisa la tipologia, questa scelta spesso avviene in base ai linguaggi e agli strumenti nei quali si ha maggiore esperienza e familiarità. Non è da escludere che si prenda una decisione a causa di un investimento che ha come scopo l'apprendimento di una nuova metodologia da poter sfruttare anche in progetti futuri, in tal caso viene presa una decisione che aiuta ad aumentare il bagaglio informativo del team di sviluppo.

Nelle immagini sottostanti sono riportati degli alberi decisionali riassuntivi della pagina precedente che si possono utilizzare per decidere quale metodologia adottare. Per una visualizzazione più chiara sono state riportate in due figure separate il caso dove si voglia sviluppare un'applicazione anche per desktop e il caso in cui non sia necessario.

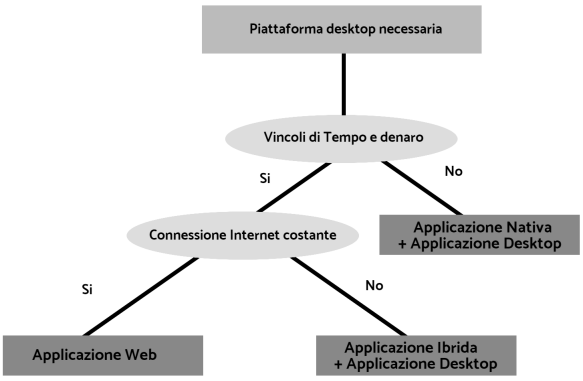


Figura 19: Schema metodologie di sviluppo con app desktop

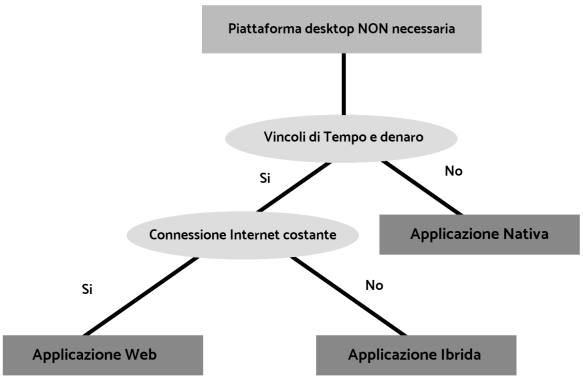


Figura 20: Schema metodologie di sviluppo senza app desktop

Nelle figure sono mostrati i passaggi da seguire senza considerare richieste specifiche da parte dell'utente, non prendendo in considerazione l'idea di scegliere per imparare un nuovo framework e senza considerare i problemi che possono nascere dovendosi interfacciare con dispositivi e sistemi operativi particolari. Inoltre viene rappresentato un percorso considerando di dover sviluppare un'intera applicazione e non solo una parte di essa (come nel caso di nuove interfacce lato client).

4 Presentazione del progetto

Il software utilizzato da Prasco S.p.A fornisce supporto per la gestione di un magazzino in ambito automotive. La tecnologia utilizzata attualmente è un'applicazione creata appositamente per funzionare su dispositivi Datalogic dotati di un lettore di codici a barre per scannerizzare i prodotti. Nella figura sottostante è possibile vedere l'apparecchio utilizzato.

Figura 21: Dispositivo RFID Datalogic Falcon

L'obiettivo principale della tesi è quello di re-ingegnerizzare l'applicazione che attualmente stanno utilizzando i dipendenti dell'azienda in modo da poter migliorare l'efficienza, l'operato, la quality of life e lo sforzo necessario da parte di quest'ultimi.

L'idea è di riproporre tutte le funzionalità presenti attualmente cercando di migliorarne la logica e il funzionamento, eliminando tutto ciò che non è utilizzato/necessario e introducendo nuovi elementi che possano portare valore all'applicazione.

Inoltre si vuole anche lasciare aperta la possibilità di proporre in futuro una versione desktop dell'applicazione per poter offrire la possibilità di interagire con essa anche da un comune PC da ufficio, questo permetterebbe ai responsabili di tenere traccia delle operazioni e visualizzare l'operato comodamente senza dover utilizzare dispositivi mobile. Per questo motivo si è optato per uno sviluppo di una web app multipiattaforma, come descritto nel capitolo precedente.

Proponendo una web app utilizzabile su comuni smartphone o tablet si ridurrebbe di molto il costo necessario per acquistare un nuovo dispositivo in caso di guasto, questo porterebbe beneficio all'azienda anche in termini di denaro.

In aggiunta si vuole anche proporre un eventuale metodo/algoritmo che possa sfruttare i dati raccolti durante l'operato del software in modo da fornire delle migliorie aggiuntive. Questa proposta si potrà nel caso integrare in futuro nell'applicazione, introducendo al suo interno l'implementazione degli algoritmi di data mining, tale argomento verrà approfondito nell'ultimo capitolo.

4.1 Analisi delle funzionalità e dei requisiti

Trattandosi di un'applicazione per la gestione del magazzino, è necessario che la gestione dei processi principali in questo settore sia garantita. Nel capitolo riguardante la gestione del magazzino è possibile apprendere quali siano questi processi e la loro natura, in seguito verranno descritti quelli utilizzati dall'azienda in questione, la loro implementazione attuale e quella futura.

Oltre alla presenza di queste funzionalità è necessario che l'applicazione rispetti alcuni requisiti non funzionali, alcuni presenti nella versione vecchia e alcuni che sono stati espressamente richiesti come miglioria.

Nelle sezioni seguenti verranno approfonditi i requisiti in base alla loro natura (funzionale e non), senza tenere in considerazione l'implementazione e la tecnologia attualmente utilizzata, prima di proseguire però è necessario avere una panoramica visiva di come sia strutturata l'applicazione.

Dato che non è stato possibile accedere a un dispositivo, la logica dell'applicazione è stata derivata da una lunga e approfondita analisi del codice attuale, seguita dalla ricostruzione manuale e cartacea di tutte e 38 le schermate. La mappa concettuale sottostante rappresenta tutti i flussi possibili percorribili partendo dalla schermata che si visualizza all'avvio del software.

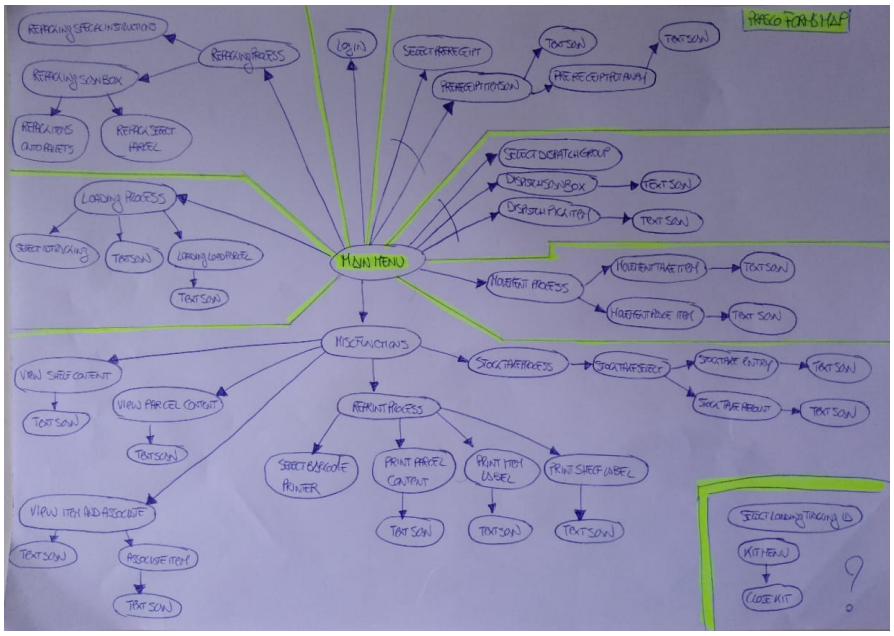


Figura 22: Mappa dei flussi possibili

4.1.1 Requisiti funzionali primari

L'applicazione presenta un menù principale da dove è possibile accedere alle funzionalità principali oltre che a quelle secondarie attraverso una specifica voce. L'intera applicazione necessita dell'implementazione di un lettore di codice a barre, nei requisiti viene esplicitamente chiesto di avere dei feedback sonori sia in caso di errore che di successo dell'operazione.

Inoltre, dato che è possibile che i codici a barre si rovinino e diventino irriconoscibili per il lettore, viene richiesta la possibilità di inserire in ogni occasione manualmente il barcode, digitando le cifre riportate su di esso.

All'interno dell'applicazione deve essere possibile navigare tra le varie funzionalità potendo riprendere l'operazione che si sta effettuando anche in seguito, salvandola anche se incompleta, inoltre quando si abbandona una dalle operazioni principali sarebbe opportuno chiedere conferma prima di uscire.

Infine l'applicazione deve poter permettere agli utenti di effettuare le operazioni solamente nel caso si sia precedentemente effettuata un'operazione di login. L'autenticazione deve essere eseguita con un username, una password e con il codice del dispositivo, non è possibile effettuare l'accesso su due dispositivi con le stesse credenziali contemporaneamente.

Tutti i requisiti analizzati comprendono la necessità di mostrare delle informazioni testuali nella schermata, non verranno elencati per non creare confusione ma saranno visibili nelle schermate di presentazione dell'applicazione. Le operazioni essenziali da garantire e di maggiore importanza sono:

- **Depositi**: La gestione dei depositi è relativa a quanto un carico viene ricevuto e necessita di essere depositato all'interno degli scaffali. La funzionalità si lega ai processi di stoccaggio e immagazzinamento visti nei capitoli precedenti. Questa sezione deve permettere di poter effettuare le seguenti operazioni, nell'ordine indicato:
  - Selezione di un precarico dalla lista di quelli disponibili
  - Lettura di un prodotto e controllo dell'appartenenza al precarico selezionato
  - Proposta di uno scaffale suggerito per quel prodotto con visualizzati i relativi posti liberi
  - Richiesta di lettura dello scaffale e nuovamente del prodotto, conferma nel caso si voglia cambiare lo scaffale suggerito e inserimento della quantità da depositare
  - Registrazione del deposito
  - Possibilità di rieseguire i 4 passi precedenti anche per gli altri prodotti facenti parte dello stesso deposito del precarico

- **Prelievi**: La sezione è dedicata al prelevamento di tutti i prodotti facenti parte di un ordine, si tratta del primo passo da eseguire per spedire infine i prodotti al destinatario e corrisponde al processo di prelevamento descritto nel capitolo della gestione del magazzino. La funzionalità di prelievo deve poter garantire sequenzialmente:
  - Selezione di una spedizione dalla lista di quelle da effettuare
  - Lettura di una scatola e controllo della validità di essa
  - Visualizzazione dello scaffale e dell'articolo da prelevare
  - Richiesta di lettura dello scaffale e nuovamente del prodotto prelevato, inserimento della quantità con possibilità di saltare l'articolo corrente, qualora non presente nello scaffale indicato (successivamente manualmente da db verrà risolto l'errore)
  - Registrazione del prelievo
  - Riesecuzione dei 3 passi precedenti per tutti gli altri prodotti facenti parte della stessa spedizione
- **Imballi**: Funzionalità da eseguire considerando gli ordini già prelevati, prima di poterli spedire, corrisponde all'implementazione del processo di imballaggio descritto precedentemente. In questa fase vengono anche stampate le etichette da posizionare sui vari prodotti, la stampa può avvenire singolarmente per ogni prodotto oppure globalmente per tutto l'ordine, a seconda dei casi. L'operazione si può svolgere in due modi differenti, re-imballando nella stessa scatola oppure travasando il contenuto in una scatola diversa, a seconda di come verrà eseguito il processo. Viene richiesto di poter eseguire in modo ordinato:
  - Scansione di un UDC e controllo della sua correttezza
  - Possibilità di selezionare la stampante da utilizzare (attualmente viene inserita appena si effettua il login e si può modificare in questa funzionalità o in una dedicata)
  - Possibilità di visualizzare le informazioni aggiuntive, se presenti, riguardanti il metodo di imballaggio richiesto dal cliente, se specificato da quest'ultimo
  - Lettura della scatola di destinazione
  - Nel caso la scatola sia la stessa (come in presenza di materiale sfuso) è necessario selezionare il tipo di scatola di destinazione, una volta confermato verranno stampate tutte le etichette
  - Nel caso la scatola sia differente si deve fornire la possibilità di scannerizzare un prodotto alla volta e per ognuno si deve poter selezionare la quantità da imballare e di etichette da stampare, che non necessariamente deve essere uguale

- **Spedizioni**: Realizzazione del processo di spedizione discusso nella prima parte della tesi, attualmente è possibile effettuare la chiusura di un automezzo in due modi, manualmente o caricando i prodotti precedentemente imballati. Tuttavia su richiesta del cliente la prima metodologia deve essere eliminata dato che non viene mai utilizzata. Per questo motivo le operazioni da garantire sequenzialmente sono:
  - Lettura di una spedizione con relativo controllo di validità
  - Possibilità di chiudere l'automezzo con relativa richiesta di conferma (qualora vi siano già sufficienti prodotti all'interno), oppure possibilità di caricare dei prodotti all'interno dell'automezzo assegnato automaticamente alla spedizione scansionata. Deve essere garantita la possibilità di chiudere un automezzo con una parte dei prodotti della spedizione (nel caso siano troppi), in questa occasione il resto verrà caricato su un altro veicolo.
  - Nel caso si sia scelto il caricamento si deve fornire l'opportunità di selezionare a una a una le scatole da caricare sull'automezzo, con relativa conferma.

  Nel caso si sia scelto di caricare sull'automezzo dei prodotti imballati è successivamente necessario fornire la possibilità di chiudere il veicolo.
- **Trasferimenti**: Questa funzione permette di gestire il magazzino e i prodotti al suo interno, fornendo la possibilità di spostare gli articoli da una parte all'altra dello stabile. Anche in questo caso possiamo trovare riscontro nel processo di immagazzinamento descritto nel capitolo della gestione del magazzino. Viene richiesto di poter effettuare:
  - Selezione tra deposito e prelievo di un prodotto
  - In entrambi i casi deve essere possibile scansionare lo scaffale su cui eseguire l'operazione, leggere l'articolo e inserire la quantità da spostare o spostata. Inoltre deve essere fornita una lista di prodotti e di quantità disponibile di tali articoli in tutti e due i rami di questa funzionalità

La lista precedente racchiude tutte le specifiche che sono state dedotte dall'analisi del codice dell'applicazione con l'integrazione di alcune informazioni ricevute grazie a delle visite in azienda effettuate da parte di alcuni dipendenti dell'azienda Orbyta. Tali funzionalità si possono vedere nella parte superiore dell'immagine raffigurante la mappa globale delle funzionalità e dei form dell'applicazione attuale.

L'insieme delle funzionalità primarie della versione attuale dell'applicazione comprende una sezione denominata Kit, ma non è funzionante, la logica non è implementata e nella nuova versione non è richiesta l'introduzione di tale funzionalità, per questo motivo non verrà descritta o trattata.

4.1.2 Requisiti funzionali secondari

Oltre all'insieme di operazioni importanti descritte nella sezione precedente è anche necessario garantire una serie di servizi secondari ma comunque che ricoprono un'importanza. Queste funzionalità sono prettamente di gestione tecnica e non si rispecchiano nei processi descritti all'interno del capitolo della gestione del magazzino.

L'accesso a queste operazioni deve essere permesso passando per il menù principale utilizzando un tasto per le funzionalità aggiuntive, nello specifico identificato come "Altro". La lista delle funzionalità e dei requisiti secondari richiesti è la seguente:

- **Contenuto scaffale**: In questa parte dell'applicazione è possibile visionare il contenuto di uno scaffale, per poter far ciò è necessario garantire:
  - Lettura dello scaffale da visualizzare e verifica della sua correttezza
  - Visualizzazione della lista di articoli presenti all'interno di esso con relativa descrizione e quantità
- **Contenuto scatola**: Analogamente allo scaffale è possibile visualizzare il contenuto di una scatola con questa funzionalità, anche in questo caso è necessario:
  - Eseguire una lettura della scatola da visualizzare e controllarne la sua validità
  - Visualizzare la lista di articoli presenti all'interno di essa mostrandone descrizione e quantità
- **Stampa etichette**: Questa funzionalità permette di effettuare diverse operazioni legate alla stampa, in particolare:
  - Selezionare una stampante de impostare come predefinita da una lista di disponibili
  - Scansione di un articolo e dopo averne verificata la validità dare la possibilità di inserire la quantità di etichette da stampare per quel prodotto.
  - Leggere uno scaffale e nel caso sia corretto permettere l'inserimento della la quantità di etichette da stampare per quello scaffale
  - Dare la possibilità di scannerizzare una scatola e verificarne la correttezza, successivamente permettere di poter stampare le etichette di tutti gli articoli presenti in essa in una volta sola. In alternativa alla stampa totale lasciar selezionare da una lista uno degli articoli all'interno della scatola scansionata e inserire la quantità di etichette da stampare per quel prodotto

- **Dettagli articolo**: Tale funzionalità ha lo scopo di associare un prodotto a un nuovo codice a barre oltre a permettere di poter vedere le informazioni dell'articolo, è necessario garantire sequenzialmente:
  - Lettura di un articolo con controllo di correttezza
  - Scansione di un codice a barre da poter associare al prodotto, senza nessun controllo
  - Registrare l'associazione del prodotto al nuovo codice a barre
- **Inventario**: Grazie a questa sezione deve essere possibile accedere all'elenco degli inventari attivi per poter aggiornare la quantità in uno scaffale e volendo effettuare un conteggio di un prodotto scaffale per scaffale. Queste operazioni devono essere possibili garantendo nel seguente ordine:
  - Selezione tra inventario e riconteggio di un prodotto
  - In entrambi i casi visualizzazione dell'elenco degli inventari attivi da cui è possibile selezionare un documento
  - Nel caso di inventario è necessario richiedere la lettura prima dello scaffale e poi del prodotto e per entrambi occorre effettuare il controllo di validità. Successivamente è necessario mostrare il numero di pezzi già contati e richiedere la nuova quantità, in modo da aggiornare quella vecchia e poter inserire nell'inventario la registrazione. Infine occorre registrare l'aggiornamento
  - Nel caso in cui si sia scelto il riconteggio occorre prima richiedere la scansione del prodotto e successivamente occorre mostrare uno per uno tutti gli scaffali in cui è presente il prodotto dando la possibilità per ognuno di inserire la quantità attualmente presente in quello scaffale e consentendo di proseguire fino a termine degli scaffali contenenti tale articolo.

Come si evince dalla lista seguente la complessità e l'utilizzo delle funzionalità secondarie è sicuramente molto inferiore rispetto a quelle primarie, ma è molto importante che anche esse siano presenti nella versione re-ingegnerizzata dell'applicazione.

Nella figura che rappresenta la mappa globale dell'applicazione e delle schermate è possibile identificare le funzionalità secondarie, che risiedono nella metà inferiore del disegno.

L'analisi dei requisiti funzionali rappresenta la maggior parte delle richieste da soddisfare, tuttavia una vi è una parte di requisiti non funzionali che devono essere implementati e che verranno trattati nella seguente sezione.

4.1.3 Requisiti non funzionali

I requisiti non funzionali ricoprono un ruolo importante nello sviluppo di un progetto, in particolare di un'applicazione, necessitano quindi di essere analizzati correttamente.

Nel caso in esame si ha come obiettivo lo sviluppo di un software che verrà utilizzato solamente all'interno dell'azienda, maggiormente all'interno del magazzino. Per questa ragione nella valutazione e nell'analisi di questi requisiti verrà considerato il fatto che il prodotto non verrà utilizzato globalmente come i siti web o le applicazioni più diffuse ma solo tra un gruppo ristretto di utenti specifici.

Analizzando l'applicazione attuale e le richieste esplicite del cliente è stato possibile stabilire i seguenti requisiti non funzionali che l'applicazione nuova dovrà soddisfare:

- **Compatibilità**: L'applicazione deve poter essere eseguita su qualunque tipo di smartphone e tablet, sia Android che IOS
- **Portabilità**: Il sistema deve poter essere utilizzato anche in eventuali altri magazzini dell'azienda senza necessità di operazioni particolari
- **Performance**: Ogni azione/richiesta da parte dell'utente deve essere eseguita in meno di 2 secondi
- **Usabilità**: L'applicazione deve essere utilizzabile dai dipendenti con meno di 10 minuti di spiegazione
- **Tolleranza ai guasti**: Il sistema deve continuare a funzionare anche in caso di errore, segnalandolo all'utente
- **Estensibilità**: Deve essere possibile aggiungere nuovi componenti (come per esempio l'implementazione del data mining) senza particolari complessità
- **Integrità dei dati**: Devono essere garantiti l'accesso e la modifica concorrente ai dati da parte di più operatori contemporaneamente
- **Disponibilità**: In ogni momento deve essere garantito l'operato del software, ad eccezione di quando manca la connessione internet, considerato fattore esterno
- **Flessibilità**: L'applicazione deve essere strutturata in modo da garantire una modifica rapida in caso di cambio di requisiti funzionali
- **Manutenibilità**: Deve essere possibile isolare una funzionalità guasta e poterla riparare garantendo il corretto operato di tutte le altre

Molti dei requisiti sopra elencati non sono soddisfatti nell'applicazione attuale, questo è anche dovuto al fatto che essa è stata sviluppata molto tempo fa e deriva da una rapida personalizzazione del codice offerto da Navision. Le modifiche che sono state fatte durante questa personalizzazione sono parecchio confuse e alcune non funzionanti, probabilmente dovute da un tempo di progettazione breve e da un budget di tempo e/o denaro ridotto.

Per avere un'idea riassuntiva più chiara è possibile far riferimento alla seguente immagine che mostra la lista dei requisiti non funzionali con il relativo stato in base alla versione vecchia e rispetto a quella nuova:

Requisito	Stato
Compatibilità	Non presente prima
Portabilità	Da mantenere
Performance	Non presente prima
Usabilità	Da mantenere
Tolleranza ai guasti	Da mantenere
Estensibilità	Presente in parte prima
Integrità dei dati	Da mantenere
Disponibilità	Da mantenere
Flessibilità	Presente in parte prima
Manutenibilità	Da mantenere

Figura 23: Stato requisiti non funzionali

All'interno delle sezioni sottostanti, che mostrano il confronto AS-IS vs TO-BE, sarà possibile capire perché alcuni di questi requisiti sono essenziali per produrre un prodotto che rispecchi le richieste del cliente ma sopratutto che possa essere analizzato, modificato e integrato molto facilmente. L'idea è di evitare in futuro di dover investire molte risorse nella comprensione del codice, cosa che è successa in questo progetto di tesi, per questo motivo anche il fatto di produrre codice di qualità è stato visto come requisito fondamentale.

4.2 Architettura attuale: AS-IS

In questa sezione verranno descritti i linguaggi, le tecnologie, le logiche e l'organizzazione del codice per quanto riguarda l'applicazione attualmente utilizzata in modo da poterne comprendere il funzionamento al meglio ed in seguito apportare tutte le possibili migliorie. Il software attuale è formato tre parti fondamentali interconnesse tra loro: l'applicazione, il client e i servizi.

Nella figura sottostante è possibile vedere graficamente le connessioni tra i vari componenti e il flusso di operazioni una volta ricevuta una richiesta.

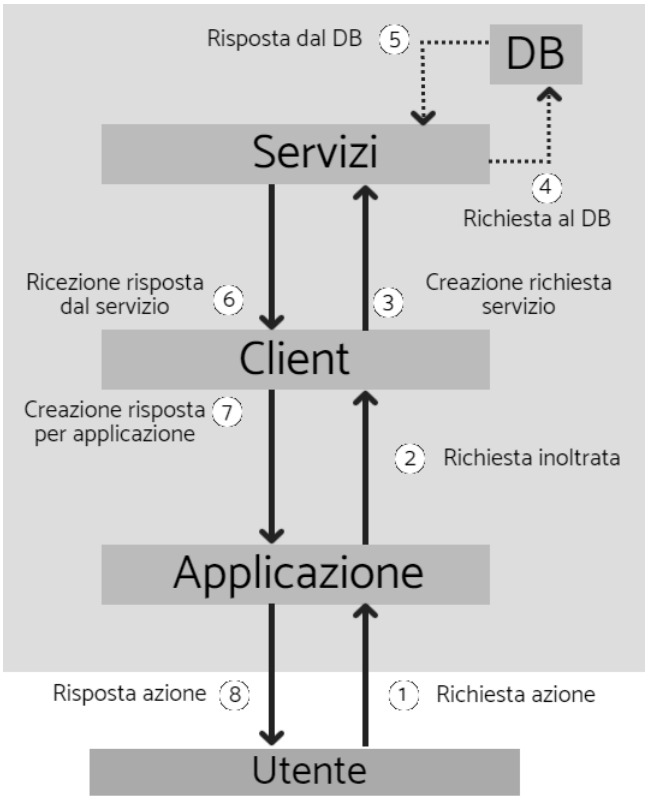


Figura 24: Architettura AS-IS

La parte che si interfaccia con l'utente è l'applicazione, la quale ha il compito di fornire graficamente accesso ai servizi e mostrare tutte le informazioni necessarie per il completamento dei processi, per questa ragione la logica di front-end è tutta situata al suo interno.

Il client invece funziona da ponte tra l'applicazione e i servizi, al suo interno risiede la maggior parte della logica di back-end. Il suo compito è quello di ricevere richieste dall'applicazione e dopo aver fatto le modifiche necessarie costruire una chiamata ai servizi tramite protocollo SOAP.

Infine grazie ai servizi offerti da Navision è possibile interagire con il database ed eseguire le operazioni necessarie per il completamento dei processi di gestione del magazzino. I servizi ricevono delle richieste e restituiscono un risultato al client dopo averle elaborate in maniera asincrona, quest'ultimo una volta elaborata la risposta ricevuta restituirà il risultato all'applicazione.

Per una comprensione più approfondita verranno trattate separatamente le tre parti, in modo da poter facilmente evidenziare le corrispondenze con la nuova realizzazione dell'applicazione.

4.2.1 Applicazione

L'applicazione è stata sviluppata interamente utilizzando come ambiente Visual Studio 2008 e come linguaggio C# con framework .NET v4.0 (disponibile in VS2008 e VS2010). La soluzione sviluppata utilizza dei form interconnessi tra loro per creare un flusso relativo a ogni processo.

All'avvio dell'applicazione viene chiesto di effettuare il login e, una volta completata questa operazione, viene mostrato il menù principale che è possibile notare ricostruito nell'immagine seguente.

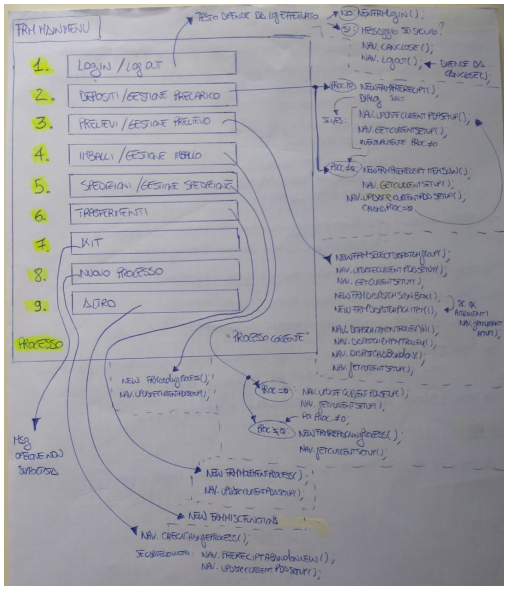


Figura 25: Menù principale applicazione attuale

Le funzionalità principali sono accessibili da questo menù, mentre quelle secondarie attraverso la voce "Altro". Uno dei problemi di questa versione è il fatto che il menù principale salva lo stato di alcuni processi e spesso per poter proseguire nella continuazione di uno di essi nei caricamenti si rimbalza più e più volte tra il menù principale e gli altri form. Una delle migliorie necessarie è quella di rendere indipendenti tutti i processi senza che il menù principale tenga traccia di qualcosa.

Dall'immagine seguente è possibile vedere una lunga lista di file, essi corrispondono ai vari form, alcuni sono stati raggruppati all'interno di cartelle per ordine, invece molti altri sono stati messi tutti insieme, anche questo aspetto è stato migliorato nella nuova versione.

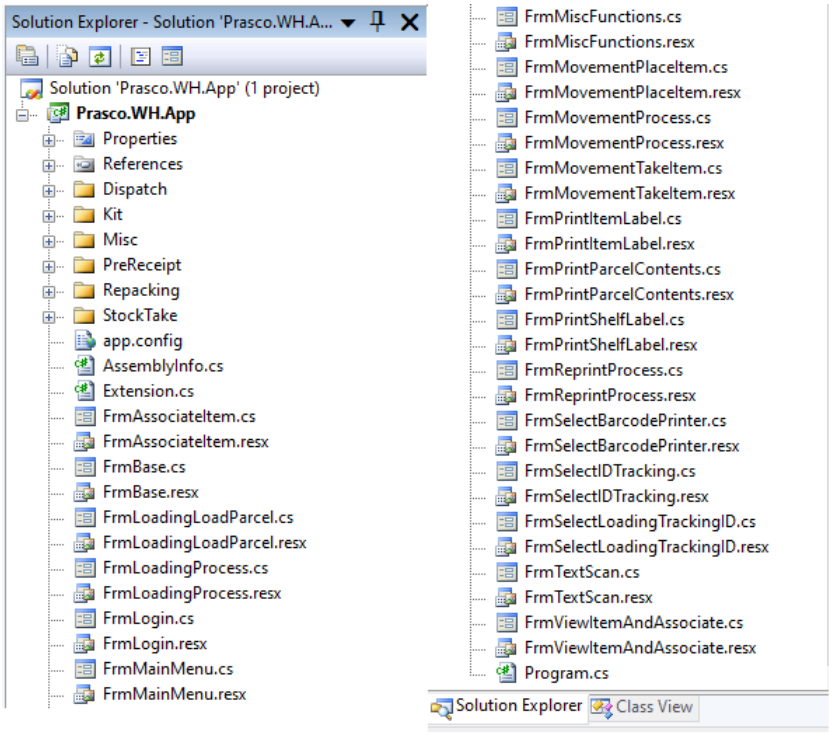


Figura 26: Struttura applicazione attuale

Per una questione di comprensione è stato posizionato a destra il continuo della lista di sinistra che avrebbe dovuto proseguire verso il basso. Dall'immagine è anche possibile vedere che in fondo è presente un file denominato "Program.cs" che serve solamente come main dell'applicazione e per far partire il tutto una volta avviato il dispositivo.

Per ogni schermata visibile dal dispositivo vi è un relativo file .cs (file sorgente di C#), dove vi è presente il codice che è stato utilizzato per ricostruire la logica e manualmente tutte le schermate, attraverso le posizioni assolute. Ogni file .cs inoltre è accoppiato a un file .resx che rappresenta attraverso il linguaggio XML il file sorgente. Nelle seguenti immagini è possibile vedere un esempio di entrambi i file.

```
        <xsd:element name="data">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
              <xsd:element name="comment" type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required" msdata:Ordinal="1" />
            <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
            <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
            <xsd:attribute ref="xml:space" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="resheader">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required" />
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figura 27: Esempio file .resx

```
public FrmAssociateItem(string ItemIn, string DescriptionIn)...

protected override void Dispose(bool disposing)...

private void InitializeComponent()...

protected override void Laser_OnGoodRead(ScannerEngine sender)...

private void buttonAssociate_Click(object sender, EventArgs e)...

private void buttonClose_Click(object sender, EventArgs e)...

private void buttonEditText_Click(object sender, EventArgs e)...

private void IUAssociateItem_Closing(object sender, CancelEventArgs e)...

private void IUAssociateItem_KeyUp(object sender, KeyEventArgs e)...

private void IUAssociateItem_Load(object sender, EventArgs e)...

private void ProcessScan(string ProcessData)...
```

Figura 28: Esempio file .cs

I file contenenti la descrizione XML non sono stati usati per la comprensione dell'applicazione, è quindi necessario concentrarsi solamente sui file sorgente per poterne capire il corretto funzionamento. Il progetto non compila sui normali PC perché è studiato per essere eseguito sui dispositivi Datalogic, però è possibile avere una breve idea di come funziona la logica nella pagina seguente.

All'interno di ogni form ci sono dei metodi sempre presenti che per comprendere il funzionamento di ogni singola schermata è bene conoscere, essi sono (con X viene indicato un nome generico di una schermata):

- **X()**: Costruttore del form X, richiama sempre InitializeComponent() e se necessario viene utilizzato per abilitare bottoni o settare variabili
- **InitializeComponent()**: Serve per posizionare nella schermata tutti i vari elementi quali bottoni, label, input, e così via. Inoltre vengono anche assegnate tutte le proprietà, i testi e gli eventi ad essi associati
- **Dispose()**: Usato solamente per liberare le risorse non utilizzate, non è da considerare per la comprensione dell'applicazione
- **X_Load()**: Metodo chiamato quando la schermata X viene caricata, tipicamente usato quando è necessaria una chiamata al client per ricevere dei dati appena si carica la pagina (solitamente per caricare liste di dati)
- **X_KeyUp()**: Questo metodo permette di associare dei tasti a dei metodi, in modo da assegnare un particolare metodo alla pressione di un particolare tasto nella schermata X
- **Laser_onGoodRead()**: Viene chiamato quando un articolo viene riconosciuto dal codice a barre, dopo una brevissima formattazione viene chiamato il metodo ProcessScan()
- **ProcessScan()**: Utilizzato per effettuare qualsiasi cosa una volta che è stato riconosciuto un codice a barre scansionato. All'interno di questo metodo risiede la maggior parte della logica di tutti i form che necessitano di una lettura barcode

Oltre a questi metodi standard ogni schermata possiede la lista di metodi propri necessari per implementare la logica di quella particolare fase del processo. Quando è necessario effettuare una chiamata al client viene semplicemente chiamato il metodo della classe Nav, dato che il client è collegato alla soluzione non vengono generati errori. Un esempio di chiamata è riportato nell'immagine sottostante.

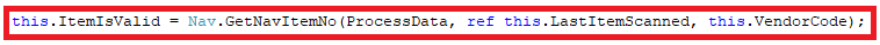


Figura 29: Chiamata al client

Questo tipo di chiamata permette sia di ricevere un valore come risposta sia di passare per referenza dei parametri che verranno riempiti dal client. In questo modo l'applicazione effettua le richieste, inoltre tutte le chiamate vengono racchiuse da dei blocchi try-catch di controllo he rilevano gli errori.

4.2.2 Client

Come anticipato precedentemente il client riceve le richieste dall'applicazione e si occupa di creare le richieste per i servizi ed elaborare le risposte ricevute da essi. Il progetto client è collegato all'applicazione ma è stato sviluppato a parte, ed è possibile aprirlo anche con le versioni più recenti di Visual Studio, a differenza dell'applicazione. Anche in questo caso il linguaggio utilizzato è C# con framework .NET versione 4.0 che funge da supporto.

Il client ha una complessità molto minore rispetto all'applicazione, proprio per il suo ruolo, infatti non è stato difficile capirne la logica e la natura. Al suo interno è presente una sezione WHSvc, dove vi sono le classi rappresentanti i modelli di alcuni tipi di dato utilizzati nell'applicazione, esse consistono nell'insieme di tipi di dati base che formano un oggetto più complesso.

La struttura dell'applicazione client è possibile vederla nella figura sottostante, per rendere più leggibile la figura è stata posizionata a destra la continuazione del lato sinistro che sarebbe dovuto propagarsi verso il basso.

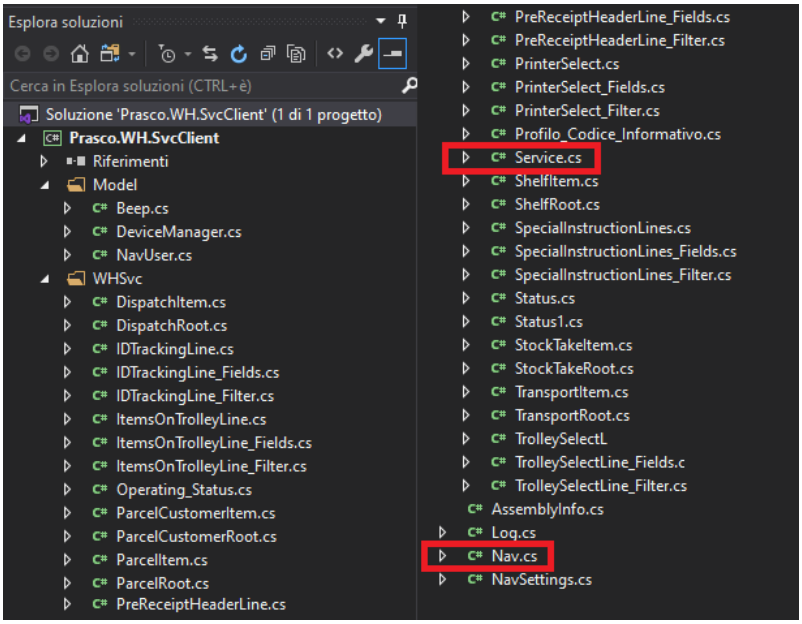
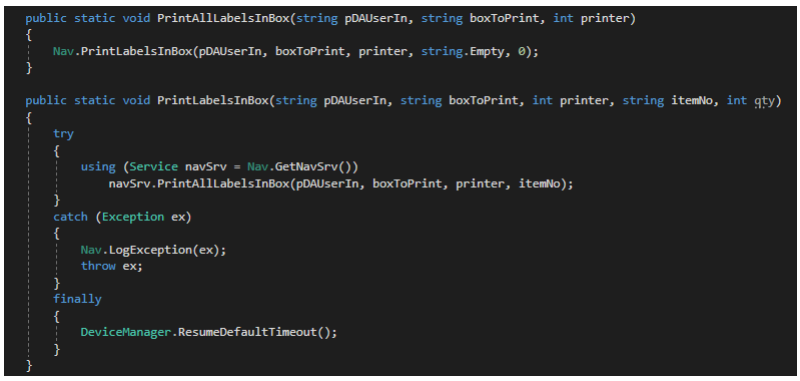


Figura 30: Struttura client attuale

Sono stati evidenziati due file nella figura sopra perché sono i due più importanti nella logica del client, gli altri sono principalmente modelli. Quando un metodo della classe Nav viene invocato dall'applicazione vengono eseguiti due passaggi principali:

- **Esecuzione del metodo**: La maggior parte dei metodi di questa classe si limitano a chiamare il metodo omonimo nella classe Service, altri invece sistemano leggermente i parametri prima di farlo. Come è possibile vedere nella figura sottostante se viene chiamato il metodo con tre parametri vi è una conversione e una chiamata a quello con 5, altrimenti viene chiamato direttamente il secondo che chiama Service

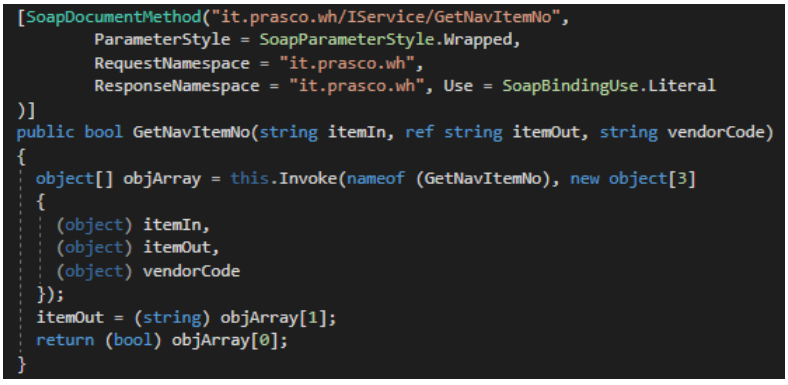


```
public static void PrintAllLabelsInBox(string pDAUserIn, string boxToPrint, int printer)
{
    Nav.PrintLabelsInBox(pDAUserIn, boxToPrint, printer, string.Empty, 0);
}

public static void PrintLabelsInBox(string pDAUserIn, string boxToPrint, int printer, string itemNo, int qty)
{
    try
    {
        using (Service navSrv = Nav.GetNavSrv())
            navSrv.PrintAllLabelsInBox(pDAUserIn, boxToPrint, printer, itemNo);
    }
    catch (Exception ex)
    {
        Nav.LogException(ex);
        throw ex;
    }
    finally
    {
        DeviceManager.ResumeDefaultTimeout();
    }
}
```

Figura 31: Metodi della classe Nav

- **Chiamata a Service**: La classe Service chiamata da Nav è quella che procede con la richiesta del servizio e ne elabora la risposta prima di restituirla, se richiesto. Quando è necessario effettuare una chiamata a un servizio SOAP occorre specificare il tipo di richiesta e il tipo di risposta oltre ai parametri necessari. Questa operazione viene fatta con delle notazioni tipiche di .NET e grazie al metodo Invoke() viene mandata la richiesta, è possibile vedere un esempio di chiamata nell'immagine seguente



```
[SoapDocumentMethod("it.prasco.wh/IService/GetNavItemNo",
        ParameterStyle = SoapParameterStyle.Wrapped,
        RequestNamespace = "it.prasco.wh",
        ResponseNamespace = "it.prasco.wh", Use = SoapBindingUse.Literal
)]
public bool GetNavItemNo(string itemIn, ref string itemOut, string vendorCode)
{
  object[] objArray = this.Invoke(nameof (GetNavItemNo), new object[3]
  {
    (object) itemIn,
    (object) itemOut,
    (object) vendorCode
  });
  itemOut = (string) objArray[1];
  return (bool) objArray[0];
}
```

Figura 32: Metodo della classe Service

4.2.3 Servizi

I servizi corrispondono al livello più alto dell'intero software, comunicano direttamente con il database e lo modificano. Tutta la logica concorrente risiede all'interno di essi, purtroppo l'accesso ai servizi non è stato fornito dall'azienda e quindi è stata necessaria un'analisi accurata per capire in base ai parametri di richiesta e al nome del metodo cosa potesse fare il servizio.

Quello che è possibile dire su di essi è che utilizzano un protocollo SOAP(Simple Object Access Protocol), che rispetto a quello REST(Representational State Transfer) è più vecchio e viene sempre meno utilizzato all'interno dei progetti attuali, tuttavia in alcuni settori è ancora usato ampiamente.

Il protocollo SOAP a differenza di quello REST permette di rilevare subito un errore in fase di compilazione nel caso non vi sia corrispondenza tra l'implementazione e il richiamo dei servizi, mentre se si utilizza il protocollo REST si riceverebbe in caso di errata implementazione una bad request a runtime, per questa ragione viene preferito il SOAP per alcune implementazioni.

Una delle risorse che è stata fornita è il file WSDL (Web Services Description Language), che contiene tutte le definizioni delle richieste e delle risposte, ha il compito di descrivere in modo formale utilizzando XML l'interfaccia del servizio. Nella figura sottostante è possibile vedere una definizione di una richiesta.

```
- <xs:element name="ScanBox">
   - <xs:complexType>
      - <xs:sequence>
            <xs:element name="userName" type="xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name="text" type="xs:string" maxOccurs="1" minOccurs="0"/>
            <xs:element name="parcel" type="xs:string" maxOccurs="1" minOccurs="0"/>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
```

Figura 33: Richiesta descritta all'interno del WSDL

Questo file verrà utilizzato anche nella versione TO-BE e se ne parlerà nell'implementazione quando verrà introdotto il proxy e nella mappatura tra modelli di risposta/richiesta service e non.

I servizi in questione sono di tipo WCF (Windows Communication Foundation) e sono stati creati seguendo le logiche, le regole e le restrizioni imposte dalla Microsoft. Così facendo anche questa parte viene sviluppata utilizzando la stessa tecnologia, in modo da permettere una facile integrazione con il resto dell'applicazione.

Trattandosi di servizi asincroni è necessario che quando vengono chiamati sia gestita l'asincronicità della risposta. Tipicamente vengono utilizzati degli operatori di await/async ma questo è compito della classe chiamante, nel caso in esame se ne occupa la classe Service.

Per cercare di capire correttamente il codice è stato stilato un documento con l'elenco di tutti i servizi richiamati all'interno del codice, divisi per categoria di processo e per schermata, in modo da rendere più semplice l'implementazione, come si vedrà nella presentazione dei controller.

L'immagine seguente rappresenta una manipolazione di tale documento e ha come scopo quello di visualizzare graficamente la quantità di servizi e la relazione con i relativi form/funzionalità di appartenenza. I processi primari sono elencati nelle prime due colonne mentre nella terza è possibile vedere quelli secondari.

Figura 34: Elenco dei servizi WCF utilizzati

4.3 Architettura finale: TO-BE

All'interno di questa sezione verranno presentati a grandi linee gli elementi che comporranno la versione finale dell'applicazione in modo da capirne la loro utilità e la loro interazione. Verrà dato particolare focus alla scelta e al percorso che ha portato ad essa, questo perché il capitolo seguente sarà interamente dedicato all'approfondimento di questa sezione per poter apprendere al meglio quali siano le migliorie introdotto con la creazione di questa applicazione e come i requisiti funzionali e non siano stati raggiunti .

In questo progetto di tesi si è analizzato attentamente il codice vecchio in modo da poter migliorare al massimo la nuova versione e poter effettuare una scelta progettuale corretta. Data la difficoltà e l'impiego di risorse che è stato necessario si è stabilito come obiettivo principale quello di fornire un codice di qualità facilmente comprensibile ed analizzabile da altri.

La struttura principale dell'applicazione finale è quella visibile nella figura sottostante, l'immagine è stata semplificata per poter comprendere in maniera più chiara il ruolo dei vari componenti, proseguendo nella tesi ci sarà un approfondimento sull'applicazione, sul back-end e sul DB Mock.

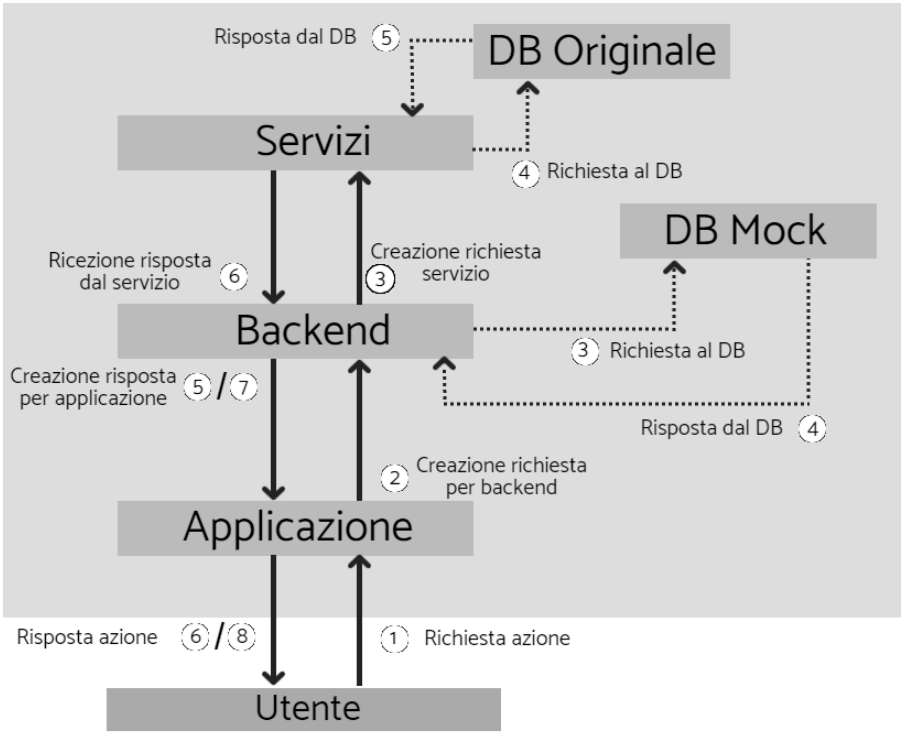


Figura 35: Architettura TO-BE

4.3.1 Applicazione front-end con ReactJS

La creazione dell'applicazione vera e propria (che comprende tutta la logica front-end) è avvenuta utilizzando come ambiente di sviluppo Visual Studio Code e utilizzando ReactJS come libreria. Come descritto nel capitolo sullo sviluppo nativo vs multipiattaforma è possibile utilizzare questa libreria per creare delle interfacce utente, utilizzando JavaScript come linguaggio.

Lo studio di ReactJS è parte della tesi, le competenze ad inizio percorso riguardante questa libreria erano nulle, questo ha richiesto la frequentazione di corsi online per l'apprendimento delle basi. Per questo motivo durante il progetto di tesi ci sono state dell grosse e frequenti sessioni di refactoring per poter produrre codice di qualità e per perfezionare il software.

Si è scelto di utilizzare questa soluzione perché come accennato precedentemente si voleva tenere aperta la possibilità di avere in una sola volta sia una versione per mobile che una per desktop. La scelta migliore in questo caso escludeva l'impiego di linguaggi nativi (per soddisfare la richiesta di portabilità su diverse piattaforme) e di React Native (che avrebbe fornito una soluzione solo mobile).

La scelta di utilizzare questa libreria al posto di altre tecnologie è dovuta al fatto che questo settore dello sviluppo web è in forte crescita, inoltre trattandosi di una tecnologia abbastanza recente si sarebbero incrementare le competenze in un ramo dell'informatica molto ricercato e diffuso.

ReactJS si basa sul concetto di componente, ogni componente deve essere indipendente e deve il più possibile rispecchiare il concetto base di riusabilità dello stesso. La logica dell'applicazione è riassunta nella figura sottostante che rappresenta l'ordine di chiamata dei componenti e la loro gerarchia.

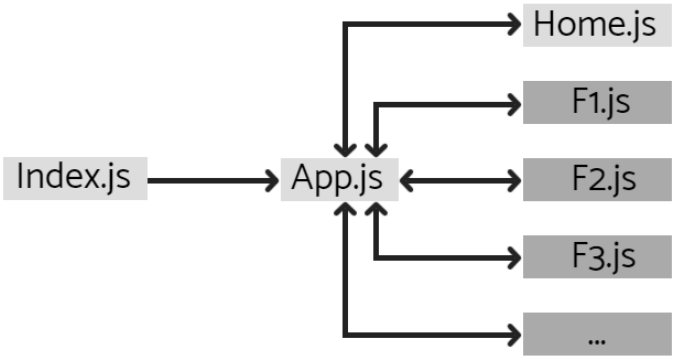


Figura 36: Struttura front-end con ReactJS

Come possibile intuire dalla figura, F1,F2,F3 ecc sono i vari componenti funzionali che corrispondono alle funzionalità del progetto, ogni form della versione precedente è stato mappato a un componente funzionale, il quale al suo interno contiene le chiamate alle API lato front-end.

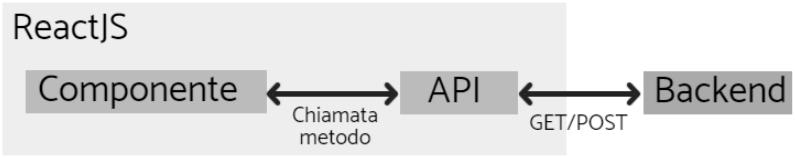


Figura 37: Interazione tra componente e back-end

La chiamata alle API da parte del componente è solamente la chiamata a un metodo, sempre gestita da ReactJS, mentre la chiamata di quest'ultimo al back-end avviene tramite chiamate GET/POST implementate con logica REST e verranno descritte nella sezione dedicata.

L'applicazione è stata sviluppata utilizzando NPM (Node Package Manager) come gestore di pacchetti che ha permesso l'installazione di diverse librerie utilizzate nello sviluppo. Tra le dipendenze installate è possibile trovare:

- **reactn**: Rappresenta un'estensione di React per la gestione dello stato globale [44]
- **react-loading-overlay**:Un semplice e personalizzabile elemento per gestire i caricamenti e le transizioni [45]
- **react-webcam**: Pacchetto utilizzato per gestire la webcam necessaria per implementare il lettore di codici a barre [46]
- **reactstrap e react-bootstrap** : Due pacchetti che permettono di utilizzare elementi con alcune grafiche base senza doverli implementare da zero e migliorare la grafica dell'applicazione [47] [48]
- **react-router-dom**: Impiegato per gestire il routing tra i vari componenti, permettere una corrispondenza univoca tra un URL e un componente dell'applicazione [49]
- **react-icons**: Permette di includere facilmente le icone necessarie nei progetti React e consente di includere solo le icone che il progetto sta utilizzando [50]

Grazie alla scelta di utilizzare React, combinata con questi pacchetti, con una corretta organizzazione dei file e con una scrittura di codice uniforme, commentato e leggibile, si è potuto garantire a lato front-end la qualità del codice che era stata posta come obiettivo principale.

4.3.2 Back-end con ASP.NET

Lo sviluppo del back-end è stato fatto utilizzando come IDE Visual Studio 2019 e come linguaggio C#. Si è scelta questa strada dato che i servizi WCF utilizzati sono stati implementati seguendo le regole e le strutture dettate dalla Microsoft, per questo motivo utilizzando questi strumenti si sarebbe potuto facilmente integrare il WSDL e si sarebbero potuti collegare in maniera semplice il front-end e i servizi.

Anche in questo caso la conoscenza iniziale del linguaggio C# e del framework .NET era nulla, quindi è stato necessario seguire dei corsi online sia per poter utilizzare questi strumenti che per conoscere meglio i servizi WCF, nel caso fosse stato necessario modificarli (cosa che non è successa).

Il compito del back-end è quello di ricevere delle richieste da parte dell'applicazione e fornire delle risposte. Dato che per poter accedere al database dell'azienda servono delle autorizzazioni (e anche per motivi di sicurezza) si è deciso di implementare un metodo alternativo per ricevere i dati oltre allo sviluppo di tutte le chiamate ai servizi.

In questa sezione verrà descritto il percorso che permette di richiamare e interfacciarsi con i servizi WCF, mentre nella seguente sezione verrà introdotto quello con il database di dati di prova. Nell'immagine seguente è possibile vedere la struttura del back-end per comunicare con i servizi.

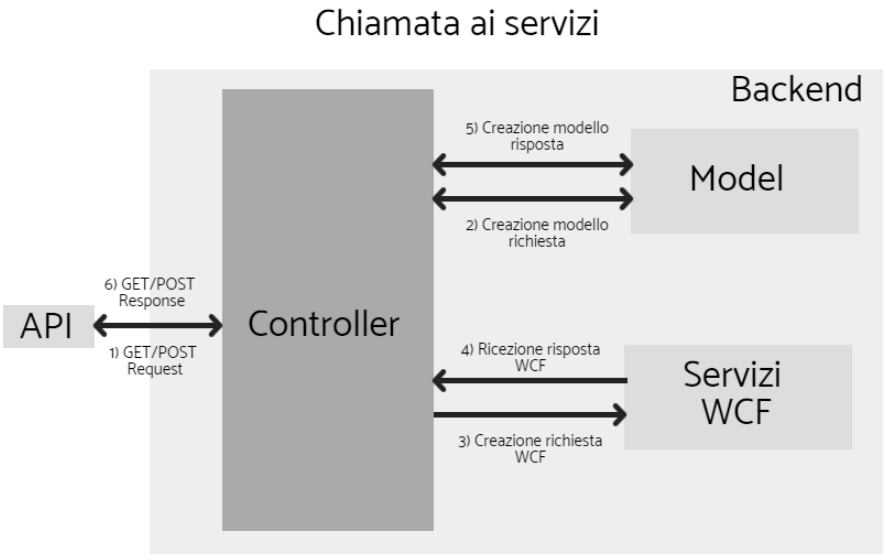


Figura 38: Comunicazione back-end con servizi WCF

Nell'implementazione delle chiamate ai servizi si è notato che non tutto ciò che viene restituito dal servizio serve all'applicazione. La tipologia e il numero di parametri necessari sono differenti, per questa ragione sono stati creati una serie di modelli per le richieste e le risposte ai vari servizi. In questo modo solamente i parametri necessari all'applicazione vengono mandati in rete, azzerando l'ammontare di dati inutili processati e spediti. Inoltre il tempo per completare l'operazione diventa minore e anche la complessità viene ridotta favorendo una maggiore leggibilità del codice.

Per adottare questa strategia è necessario che il controller esegua una mappatura tra le risposte/richieste delle API e quelle dei servizi WCF, in modo da creare un legame tra i modelli e i servizi.

4.3.3 Introduzione del DB Mock

La parte di back-end richiesta per il progetto è composta dai componenti precedentemente descritti, tuttavia un accesso al database reale per la creazione di una demo è una cosa che non si dovrebbe mai fare. Per questa ragione dopo aver implementato tutti i modelli e le chiamate ai servizi si è pensato di sviluppare una parte speculare rispetto alla chiamata dei servizi WCF. In questo modo in base ad una variabile di settaggio all'interno del file "appsettings.json" è possibile decidere di ricevere dei dati di test, dei mock, oppure di fare una vera e propria richiesta al servizio. Per implementare la logica illustrata nella seguente immagine è stato creato un database e grazie all'Entity Framework messo a disposizione dalla Microsoft è stato possibile un facile collegamento.

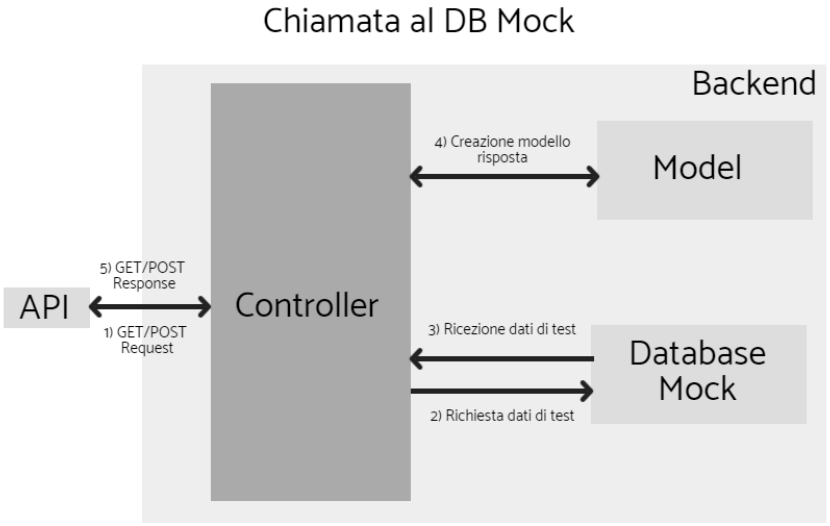


Figura 39: Comunicazione back-end con DB Mock

Attualmente tutte le richieste effettuate dall'applicazione sono gestite con i dati di test, questo per permettere di effettuare operazioni nella demo dell'applicazione senza problemi. Utilizzando questa strategia è possibile registrare depositi, prelievi e svolgere ogni tipo di operazione sull'applicazione, testarla e mostrare le sue potenzialità senza intaccare minimamente la base dati del cliente.

Non essendoci la chiamata ai servizi non è necessaria una mappatura tre modelli e tipi di richiesta/risposta del servizio WCF, però è necessario costruire il modello della risposta e ricevere quello della richiesta dalle API, per questa ragione una volta recuperati i dati dal database il controller dovrà eseguire le operazioni di creazione dei modelli prima di poter restituire il JSON con la risposta all'applicazione.

Nel capitolo successivo riguardante l'implementazione sarà presente una sezione di approfondimento dove verrà mostrata l'implementazione di questo database e si potranno vedere le query, questo permetterà di avere una visione più tecnica del software.

Sia utilizzando questa metodologia che quella precedentemente descritta si è potuta garantire una qualità del codice, una leggibilità e una comprensibilità di esso elevata lato back-end.

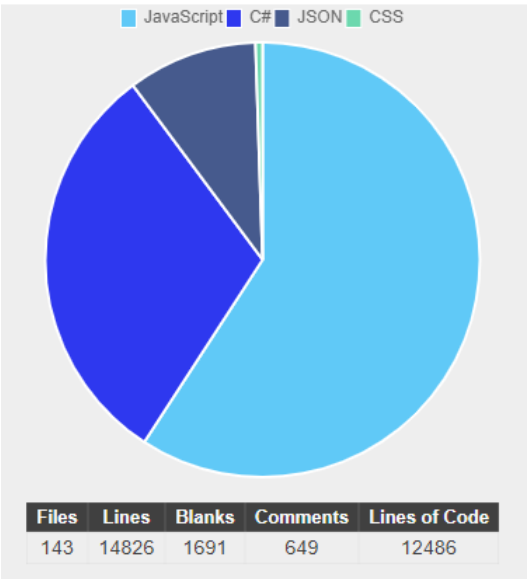
5 Implementazione dell'applicazione

In questo capitolo verrà mostrata l'implementazione del software e sarà presente parecchio codice, sarà il capitolo più tecnico in quanto verranno descritte informazioni nello specifico in modo da dare una visione più concreta dell'operato. Sarà inoltre possibile avere dei riscontri rapidi e significativi su come lo sviluppo di questo progetto possa notevolmente migliorare l'efficienza, la comprensione e l'utilizzo dell'applicazione.

La prima sezione sarà la più significativa dato che al suo interno verrà spiegato il funzionamento e l'aspetto di tutta l'applicazione (lato front-end), la seconda invece mostra come sono state impiegate le competenze acquisite per strutturare un back-end in modo chiaro, leggibile, comprensibile ed efficiente anche da un punto di vista ingegneristico.

Lo sviluppo descritto successivamente ricopre la maggior parte del tempo dedicato per lo svolgimento della tesi. Durante la spiegazione si farà riferimento alla logica introdotta nel capitolo precedente, alle strategie e alle architetture descritte in esso.

Grazie a uno strumento online automatico di conteggio delle LOC (Lines of code) è stato possibile ricavare il seguente diagramma a torta che considera la somma dei file creati per lo sviluppo del front-end e del back-end.



Files	Lines	Blanks	Comments	Lines of Code
143	14826	1691	649	12486

Figura 40: Diagramma LOC progetto

Nel conteggio sono stati considerati esclusivamente i file creati a mano da zero e senza contare il codice auto generato, come per esempio il proxy che verrà descritto in seguito.

Come si può notare dal grafico i file JavaScript sono quelli in maggioranza, corrispondono all'implementazione del front-end insieme alla fetta più esigua che rappresenta i documenti CSS. I file C# sono quelli relativi all'implementazione dei modelli e dei controller back-end mentre quelli JSON sono divisi tra ReactJS e .NET. Riguardante il conteggio c'è anche da considerare le innumerevoli operazioni di refactoring che hanno impiegato una grande quantità di risorse e che non è possibile includere nel conteggio.

5.1 Sviluppo front-end: struttura del progetto

La parte front-end dell'applicazione ricopre circa il 75/80% delle risorse impiegate per lo sviluppo (senza contare lo studio iniziale dei linguaggi e dei framework, la comprensione dell'applicazione legacy e la parte finale di data mining).

Come accennato precedentemente si è deciso di mantenere la struttura delle interfacce omonima a quella presente attualmente, per questo motivo verrà seguita fedelmente la mappa concettuale introdotta nelle analisi delle funzionalità e dei requisiti all'interno del precedente capitolo.

Proseguendo con la spiegazione si potrà vedere ogni sezione in maniera frammentata e spiegata più approfonditamente. L'intero codice ed i commenti sono stati scritti in lingua inglese per poter essere compresi da tutti gli sviluppatori.

Nella figura sottostante è possibile avere un'idea della struttura del progetto in ReactJS, anche in questo caso la colonna di destra rappresenta la continuazione di quella di sinistra verso il basso, affiancata per renderla più visibile.

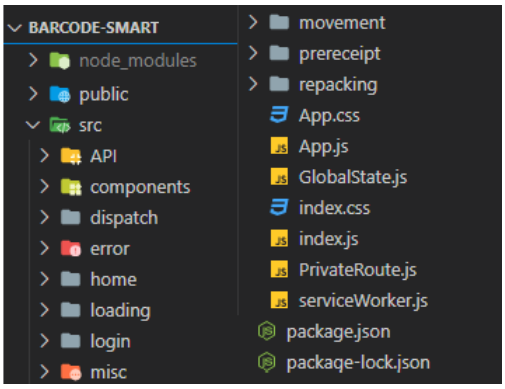


Figura 41: Struttura progetto ReactJS

Data la non utilità funzionale non verranno presi in considerazione nella spiegazione i seguenti file/cartelle:

- **Cartella node_modules**: contiene tutti i pacchetti utilizzati da React, è molto pesante e contiene molti file, solitamente non viene mai caricata nei repository e viene generata con il comando apposito da terminale
- **File package.json e package-lock.json**: contengono la lista delle dipendenze e degli script, in questa tesi sono stati creati solamente degli script brevi per far partire l'applicazione con https e senza, in modo da testare entrambi gli ambienti
- **File serviceWorker.js**: consiste in uno script che il browser esegue in background, separato da una pagina web, aprendo la porta a funzionalità che non richiedono una web page o l'interazione dell'utente [51]

Ogni altra voce mostrata nell'immagine precedente rappresenta una mappatura a una funzionalità principale, ad eccezione di alcune cartelle specifiche che si è scelto di creare per frammentare la logica migliorandone così la leggibilità e la comprensione.

Nella figura sottostante è possibile vedere il cuore del codice contenuto nel file index.js, motore da cui parte l'applicazione e da dove viene eseguito il primo vero e proprio componente.

```
initState();

sessionStorage.setItem('printerNumber', '5');

ReactDOM.render(<App />, document.getElementById('root'));
```

Figura 42: File index.js

La prima linea viene utilizzata per inizializzare le variabili globali utilizzate con ReactN una volta lanciata l'applicazione, questo concetto verrà approfondito in seguito. Nella riga sottostante è possibile vedere l'inizializzazione di una variabile di sessione, ed infine possiamo osservare l'istruzione che fa partire il tutto, grazie a cui viene renderizzato il componente App.

Dato che i file meno utilizzati sono stati introdotti è possibile proseguire con l'analisi di quelli che hanno permesso l'implementazione vera e propria dei requisiti funzionali e non.

5.1.1 Componenti base e suoni

Per via della natura di ReactJS è stato necessario introdurre una serie di componenti base personalizzati che potessero essere importati in tutte le interfacce per poterle costruire. Nella figura seguente è possibile vedere la lista completa di quelli implementati.

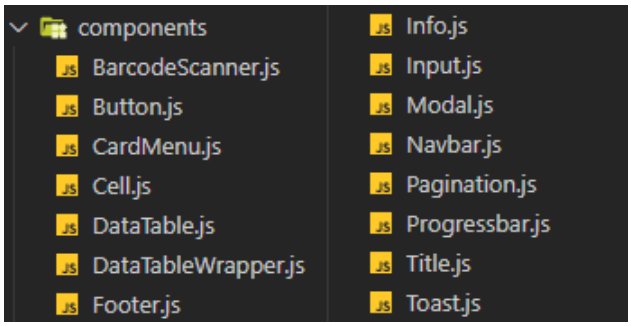


Figura 43: Componenti base

Dato che questi elementi sono utilizzati all'interno dell'applicazione si è preferito fare un breve approfondimento su ognuno di essi cercando di illustrarne l'utilità. Nelle interfacce dell'applicazione possiamo quindi trovare:

- **BarcodeScanner**: Questo è il componente fondamentale dell'applicazione, permette di visualizzare a schermo una fotocamera dove è possibile passare un codice a barre. Una volta letto il codice verrà restituito al componente padre che lo ingloba. Inoltre è anche stato inserito in questo componente un bottone per l'inserimento di un codice a barre manuale che una volta premuto permette attraverso un modal di eseguire la stessa operazione che verrebbe svolta nel caso di riconoscimento di un barcode da parte della fotocamera. Nella seguente immagine è possibile vedere a sinistra la fotocamera (che sta inquadrando un codice a barre) e a destra il modal che apparirebbe se venisse premuto il pulsante "Scan manuale".

Figura 44: Barcode Component

- **Button**: Permette di visualizzare un bottone graficamente e in base a dei parametri passati è possibile decidere se si tratti di un bottone di conferma o di back o di uno usato per tornare al menù principale. Per implementare il barcode illustrato nella precedente figura sono stati utilizzati due di questi bottoni nel modal e uno per lo scan manuale. Questo componente permette inoltre di eseguire una funzione passata come parametro nel momento del click sul bottone. Utilizzando il parametro appropriato è anche possibile disabilitarlo e abilitarlo a piacere.
- **CardMenu**: Dato che più volte nell'applicazione è necessario avere dei menù di scelta è stato implementato questo componente che ricevuta una lista crea il menù assegnando ai vari bottoni delle funzioni e dei nomi passati, è possibile vedere un esempio di menù nella seguente figura.

Figura 45: Card Menu

- **Input**: Permette di visualizzare un input con un relativo placeholder ed eventualmente una label associata. Consente una validazione nel momento della modifica del testo al suo interno utilizzando un metodo passato, inoltre è possibile disabilitarla a piacimento utilizzando un parametro specifico. I due possibili utilizzi di questo componente sono visibili nella seguente figura.

Figura 46: Input

- **Modal**: I modal possono essere di due tipi, a scopo di conferma (con solo i pulsanti per proseguire e annullare) oppure a fine di inserimento (come è possibile vedere nel barcode). Vengono abilitati e disabilitati in base a una variabile booleana e differiscono per titolo, contenuto e funzioni chiamate una volta premuti i pulsanti, tutti parametri da fornire al componente.
- **Cell, DatatTable, Pagination e DataTableWrapper** : Servono per implementare una tabella che possa ricevere una lista di titoli insieme a una di dati e creare una visualizzazione grafica delle informazioni. I primi tre componenti vengono utilizzati all'interno del wrapper insieme ad un input per la ricerca mentre il wrapper stesso è quello che effettivamente viene chiamato dai componenti funzionali e gestisce l'interazione tra essi in modo nessun altro debba occuparsene. Quando una riga viene selezionata le relative informazioni vengono mandate al componente padre, inoltre grazie alla ricerca le tuple che non rispecchiano i filtri vengono eliminate dinamicamente. Viene anche fornita la possibilità di scegliere il numero di elementi per pagina e quanti tasselli di paginazione visualizzare prima delle frecce, un esempio di tabella è mostrato nella seguente figura.

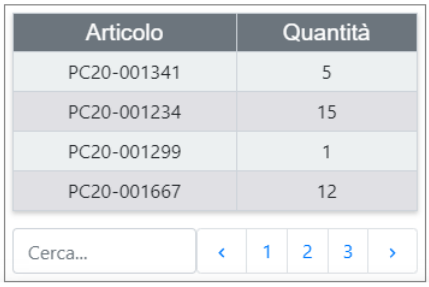


Figura 47: Data table wrapper

- **Toast**: Questo componente è utilizzato per comunicare all'utente il corretto o errato avvenimento di un'operazione oltre ad informarlo se qualcosa va storto. Hanno una durata limitata che è possibile settare all'interno del componente, dopodiché spariscono. Ci sono tre tipi di toast che vengono utilizzati ed è possibile vederli nell'immagine sottostante.

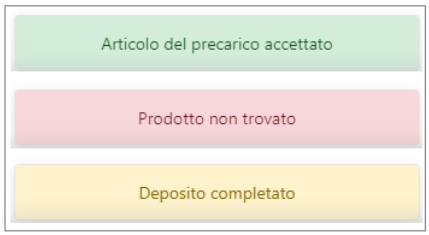


Figura 48: Toast

- **Progressbar**: Dato che alcuni processi sono interrompibili si è pensato di introdurre questo nuovo elemento che potesse dare un'idea chiara e precisa dello stato dell'operazione. Tuttavia però per poterlo utilizzare occorre una leggere modifica ai servizi, quindi si tratta di una proposta implementata che necessiterà di approvazione. Il componente riceve oltre che al testo anche il numero di elementi completati e totali, una sua istanza è visibile nell'illustrazione seguente.

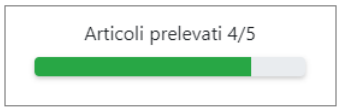


Figura 49: Progressbar

- **Info**: Questo componente riceve una lista di informazioni e le inserisce all'interno di una card per migliorarne la visualizzazione. Si può utilizzare in due modi, solamente con le informazioni base oppure in modalità estesa con all'interno anche icone o bottoni, la scelta viene effettuata in base ad un parametro. I due utilizzi sono rappresentati nella seguente immagine.

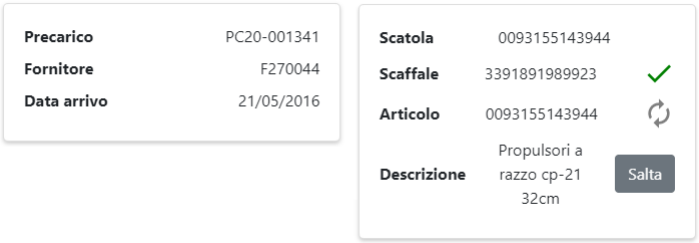


Figura 50: Info

- **Footer**: Il footer è uno dei due componenti fissi, in tutta l'applicazione è presente, varia solo il suo contenuto. Le due varianti utilizzate sono con un solo bottone o con due, ed è possibile assegnare a quello di sinistra un modal di conferma (nel caso siano due). Alla pressione di uno dei pulsanti una funzione passata come parametro viene chiamata, i due possibili scenari sono mostrati in figura.

Figura 51: Footer

- **Navbar**: La navbar è il secondo elemento fisso, posizionato in alto, essa permette di potersi spostare da un processo all'altro in maniera rapida. Se si preme il bottone appare il menù a tendina (come accade nelle classiche navbar) visibile nella metà inferiore dell'immagine seguente.

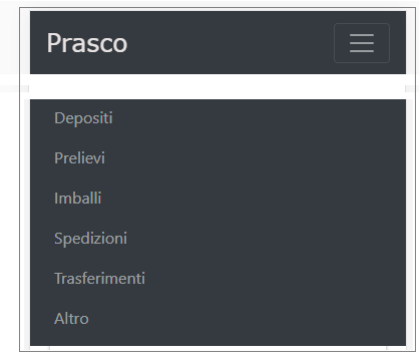


Figura 52: Navbar

- **Title**: Questo componente permette di inserire un titolo di dimensione grande o ridotta in base ad un parametro. Contiene solo testo del testo passato con del relativo stile.

Per cercare di rendere i componenti indipendenti e riutilizzabili si è deciso di introdurre lo stile all'interno della loro definizione. Si tratta di una pratica comune che è buona norma utilizzare.

Per questa ragione una volta terminata tutta la definizione del componente viene dichiarato un vettore JSON contenente lo stile che potrà essere utilizzato come CSS in linea. Questa strategia permette inoltre di decidere quale stile utilizzare in base ai parametri ricevuti come proprietà del componente, un esempio di utilizzo è mostrato nella figura seguente.

```
<ButtonReactstrap
    onClick={onClick}
    olor={type}
    size={size}
    style={{ ...style.default, ...style[styleKey] }}
    disabled={disabled}>
    {text}
</ButtonReactstrap>
```

Figura 53: Chiamata CSS interno

Adottando questo metodo il componente avrà uno stile di default sempre ed eventualmente ci sarà una modifica qualora il parametro in ingresso sia presente. Questo meccanismo è possibile grazie al fatto che in CSS le ultime proprietà dichiarate con lo stesso nome sovrascrivono le omonime precedenti. Il componente in questione ottiene uno stile aggiuntivo diverso in base al parametro

ricevuto, non c'è limite al numero di stili diversi creabili e grazie al fatto che sono sotto forma di vettore è possibile eseguire una sorta di switch inline. Nell'immagine sottostante si può osservare un esempio di definizione dei diversi tipi di stile per un bottone.

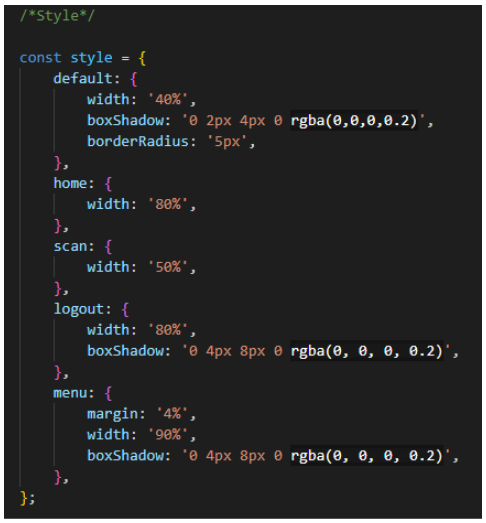


Figura 54: CSS interno ai componenti

A completare la serie di elementi base vi sono i due suoni in formato mp3 che vengono utilizzati quando un'operazione e/o una scansione viene eseguita. Questi suoni vengono riprodotti grazie al metodo play() utilizzabile generando un componente di tipo Audio. Dato che la risposta al metodo restituisce una promises vengono sempre utilizzati i costrutti .then() e .catch() per gestire il caso di errore o di successo nella riproduzione di esso.

I file si possono trovare all'interno della cartella public mostrata nella figura sottostante e hanno una durata molto breve, intorno a un secondo. Simulando dei beep di errore o successo si riesce ad avere anche un feedback sonoro, soddisfacendo così uno dei requisiti avanzati dal cliente. Il loro percorso è salvato come variabile globale grazie a ReactN.

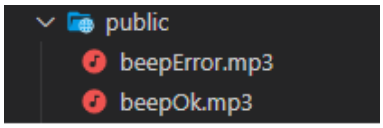


Figura 55: Audio

5.1.2 Struttura di un componente funzionale non base

Lo sviluppo di tutti i componenti funzionali non base in React è avvenuto utilizzando la stessa struttura. Come anticipato precedentemente è stata effettuata una mappatura tra interfaccia/form vecchio e componente nuovo, mantenendo lo stesso nome per facilitare la comprensione.

Nello sviluppo si è cercato di apportare come miglioria principale una linearità nel flusso di movimento, eliminando il passaggio attraverso il menù principale per eseguire operazioni riguardanti una voce precedentemente selezionata. In secondo luogo si è provato a migliorare la user experience nella ricerca e nel filtraggio dei dati grazie alla sostituzione delle liste e delle parti inerenti alla ricerca con delle datatable.

Si è scelto un approccio non a livello di classe ma a livello di funzione per i componenti dato che è molto più performante e viene preferito nello sviluppo in ReactJS. La prima sezione che troviamo in questa implementazione riguarda gli stati locali che contengono tutte le informazioni necessarie al componente per poter effettuare un render correttamente. Nella figura sottostante si può vedere un esempio di dichiarazione di stato locale utilizzando gli hooks di ReactJS.

```
/*Function component to allow the functionalities of the prereceiptPutAway form*/

function PrereceiptPutAway(props) {

    /*Local state*/

    const [loading, setLoading] = useState(true);
    const [modalShow, setModalShow] = useState(false);
    const [modalSkipShow, setModalSkipShow] = useState(false);
    const [newItem, setNewItem] = useState(true);
    const [userName, setUserName] = useState('');
    const [skipOnNextCall, setSkipOnNextCall] = useState(false);
    const [scannedElement, setScannedElement] = useState({
        disabledInput: true,
        showToast: false,
        title: 'Scannerizza scaffale',
        feedback: '',
        toastType: '',
        placeholder: '',
        confirmedShelf: false,
        confirmedArticle: false,
    });
    const [dispatchInfo, setDispatchInfo] = useState({
        shelf: '',
        maxQty: '',
        article: '',
        description: '',
    });
    const [dispatchQuantity, setDispatchQuantity] = useState({
        value: '',
        invalid: true,
    });
    const [receivedState, setReceivedState] = useState({
        dispatchGroupNumber: '',
        parcelNumber: '',
    });
```

Figura 56: Stato locale

Proseguendo con l'analisi del componente è possibile trovare la definizione degli stati globali, qualora necessari (come ad esempio nel caso in cui viene utilizzato l'audio ed è necessario il path), che vengono creati utilizzando ReactN, introdotto nella prossima sezione. Nella figura sottostante è possibile vedere come venga richiesto lo stato globale relativo al percorso dei file audio da riprodurre.

```
/*Global state */

const [audioOkPath] = useGlobal('audioOkPath');
const [audioErrorPath] = useGlobal('audioErrorPath');
```

Figura 57: Stato Globale

Dopo questa sezione si trova la definizione di alcune parti di codice che dipendono spesso dagli stati locali e che vengono passati ai componenti figli come proprietà, per questo motivo vengono raggruppati nel modo illustrato. Nell'immagine sottostante è possibile vedere come vengano creati prima gli oggetti audio e poi quello info che verrà passato come parametro al componente omonimo (i componenti in verde preceduti da Md rappresentano delle icone).

```
/*Initial value of informations that will be put inside labels (info component)*/

const info = {
    'Scatola': {
        text: receivedState.parcelNumber,
        icon: ''
    },
    'Scaffale': {
        text: dispatchInfo.shelf,
        icon: scannedElement.confirmedShelf ?
            (<MdDone style={{ color: 'green' }} size='32' />)
            :
            (<MdAutorenew style={{ color: 'grey' }} size='32' />)
    },
    'Articolo': {
        text: dispatchInfo.article,
        icon: scannedElement.confirmedArticle ?
            (<MdDone style={{ color: 'green' }} size='32' />)
            :
            (<MdAutorenew style={{ color: 'grey' }} size='32' />)
    },
    'Descrizione': {
        text: dispatchInfo.description,
        icon: <Button onClick={() => setModalSkipShow(true)} color="secondary" >Salta</Button>
    },
};

/*Audio files to play when an article is scanned*/

const audioOk = new Audio(audioOkPath);
const audioError = new Audio(audioErrorPath);
```

Figura 58: Dichiarazione oggetti

Arrivati a questo punto è possibile trovare un metodo tra i più importanti, chiamato useEffect(). Questo metodo permette di eseguire operazioni appena viene renderizzato il componente per la prima volta, oppure quando uno stato locale viene modificato, tutto dipende dalle dipendenze che vengono inserite nel secondo argomento tra le parentesi quadre.

In base a quando questo argomento cambierà lo useEffect verrà eseguito, inoltre anche appena caricato il componente avverrà una prima esecuzione. Nel caso sia necessario è possibile introdurre un return e una definizione di funzione all'interno di questo metodo per poter avviare quest'ultima quando il componente verrà distrutto, ma bisogna fare molta attenzione all'asincronicità di questo strumento. Questa strategia non verrà utilizzata nella versione finale dell'applicazione.

All'interno di questo metodo avvengono le chiamate back-end che sono necessarie al caricamento di un componente e vengono eseguite le dovute inizializzazioni, come fosse una sorta di costruttore. Tuttavia bisogna fare attenzione ad inserire tra le dipendenze uno stato locale, appena esso cambierà l'intero useEffect verrà eseguito, è quindi necessario apportare i dovuti controlli. Nella figura sottostante si può vedere l'implementazione di uno degli useEffect sviluppati.

```
/*Called when component is mounted to check info needed
and retrieved needed data from backend */

useEffect(() => {

    /*Retrieve username */

    const user = sessionStorage.getItem('loggedUser');
    setUserName(user);

    /*Control received params from prev page*/

    let params = props.location.state;

    setReceivedState({
        dispatchGroupNumber: params.dispatchGroupNumber,
        parcelNumber: params.parcelNumber,
    });

    /*Load the first shelf*/

    loadNewShelf("", "", false);

}, [loadNewShelf, props.location.state]);
```

Figura 59: UseEffect

In alcuni casi, come nell'immagine precedente, è necessario effettuare delle operazioni all'interno dello useEffect abbastanza lunghe e complesse. Per poter migliorare la qualità del codice si è utilizzato un hooks di React chiamato useCallback che permette di dichiarare delle funzioni che verranno chiamate in momenti asincroni e che hanno delle dipendenze. Data la natura di JavaScript è necessario che in questi rari casi le funzioni siano dichiarate prima di essere chiamate, tuttavia per semplicità sono state introdotte dopo nella spiegazione. Un'implementazione di esse è possibile vederla nell'illustrazione seguente che corrisponde alla chiamata precedentemente vista dove vengono richiesti dei dati al back-end che verranno elaborati nel momento della ricezione.

```
/*Load Info from backend */

const loadNewShelf = useCallback((shelf, article, skip) => {
    setLoading(true);
    DispatchGetCurrPickShelfREV(sessionStorage.getItem('loggedUser'), shelf, article, skip).then(
        resp => {
            setSkipOnNextCall(false);
            setNewItem(false);
            setScannedElement({
                disabledInput: true,
                showToast: false,
                title: 'Scannerizza scaffale',
                feedback: '',
                toastType: '',
                placeholder: '',
                confirmedShelf: false,
                confirmedArticle: false,
            });
            setDispatchQuantity({
                value: '',
                invalid: true,
            });
            if (resp.success) {
                setDispatchInfo({
                    maxQty: resp.data.maxQtyToPick,
                    shelf: resp.data.nextShelfNo,
                    article: resp.data.nextItemNo,
                    description: resp.data.itemDescription,
                });
                setLoading(false);
            } else {
                if (resp.redirect) props.history.push({ pathname: '/error_page', state: { errorMessage: resp.data } });
                else {
                    sessionStorage.setItem("allowedToken", true);
                    props.history.push({ pathname: '/dispatch_scan_box', state: props.location.state });
                }
            }
        }
    );
}, [props.history, props.location.state]);
```

Figura 60: UseCallback

A questo punto vengono implementati tutti i metodi specifici del componente, per esempio per gestire i vari eventi come click sui bottoni o interazioni con l'interfaccia. Essendo tutti diversi non verranno riportati esempi, trattandosi di semplici metodi con contenuti simili a quelli precedentemente elencati.

Infine è possibile trovare l'ultima parte comune a tutti i componenti, il return. Data la natura dei componenti funzionali non esiste un metodo render all'in-

terno di essi che li renderizzi, però è presente il return che svolge gli stessi compiti.

Il return restituisce un unico componente , che sia React o HTML non fa differenza. All'interno di esso verranno inseriti tutti i componenti uno dopo l'altro, volendo è possibile utilizzare del CSS per formattare la pagina inline o metterlo sotto forma di JSON come nei componenti base. Nell'immagine seguente si può vedere un esempio di return di un componente funzionale.

```
/*Render*/

return (
    <LoadingOverlay
        className="loading"
        active={loading}
        spinner
        text='Caricamento...'
    >
        <Title text={scannedElement.title} />
        <Info info={info} more />
        {!newItem ?
            <BarcodeScanner onUpdate={confirm} />
            :
            null
        }
        <Input
            placeholder={scannedElement.placeholder}
            changeInput={validateQuantity}
            value={dispatchQuantity.value}
            disabled={scannedElement.disabledInput}
            label
            labelValue={'Massimo ' + dispatchInfo.maxQty}
            type={"quantity"}
        />
        <Toast
            text={scannedElement.feedback}
            typeToast={scannedElement.toastType}
            showToast={scannedElement.showToast}
            toggleToast={() => setScannedElement(
                {
                    ...scannedElement,
                    showToast: !scannedElement.showToast,
                }
            )}
        />
        <Footer
            onClickBack={() => setModalShow(true)}
            onClickConfirm={onClickConfirm}
            textBack="Annulla"
            textConfirm="Preleva"
            disabledConfirm={dispatchQuantity.invalid}
        />
        <Modal
            show={modalShow}
            onHide={() => setModalShow(false)}
            text='Annullare il prelievo di questo articolo?'
            onConfirm={onClickBack}
            backText="Annulla"
            confirmText="Conferma"
            header="Annullamento prelievo articolo"
        />
        <Modal
            show={modalSkipShow}
            onHide={() => setModalSkipShow(false)}
            text="Saltare l 'articolo corrente?"
            onConfirm={onClickSkip}
            backText="Annulla"
            confirmText="Conferma"
            header="Saltare articolo"
        />
    </LoadingOverlay>
);
```

Figura 61: Render

5.1.3 Routing, session storage e stato globale

Prima di proseguire e vedere come sono state implementate le interfacce dell'applicazione è opportuno fare un breve accenno al metodo scelto per effettuare il collegamento tra esse e allo stato globale.

Il metodo migliore per mappare un URL a un componente è utilizzare il BrowserRouter, offerto dal pacchetto react-route-dom. Tuttavia per evitare che alcune pagine fossero disponibili senza aver i giusti parametri, o per evitare che un utente possa utilizzare l'applicazione senza aver effettuato il login, si è implementato un meccanismo di routing privato, spesso utilizzato per questi

scopi. Le pagine che non necessitano di particolari parametri, come quelle che si presentano all'inizio di ogni processo, non necessitano di questa cosa, quindi viene eseguito solamente il controllo sull'autenticazione.

Nell'immagine sottostante è possibile vedere il componente Router restituito da App.js, si può osservare che all'inizio è presente la navbar, comune a tutte le interfacce, mentre poi viene effettuato uno switch tra i vari URL.

```
<Router>
  <Navbar />
  <div style={style.defaultContainer}>
    <Switch>
      <Route exact path="/login" component={Login} />
      <Route exact path="/error_page" component={ErrorPage} />
      <PrivateRoute exact path="/select_prereceipt" component={SelectPrereceipt} />
      <PrivateRoute exact protectedPath path="/prereceipt_item_scan" component={PrereceiptItemScan} />
      <PrivateRoute exact protectedPath path="/prereceipt_put_away" component={PrereceiptPutAway} />
      <PrivateRoute exact path="/select_dispatch_group" component={SelectDispatchGroup} />
      <PrivateRoute exact protectedPath path="/dispatch_scan_box" component={DispatchScanBox} />
      <PrivateRoute exact protectedPath path="/dispatch_pick_item" component={DispatchPickItem} />
      <PrivateRoute exact path="/loading_process" component={LoadingProcess} />
      <PrivateRoute exact protectedPath path="/loading_load_parcel" component={LoadingLoadParcel} />
      <PrivateRoute exact path="/repacking_process" component={RepackingProcess} />
      <PrivateRoute exact protectedPath path="/repacking_special_instructions" component={RepackingSpecialInstructions} />
      <PrivateRoute exact protectedPath path="/repacking_scan_box" component={RepackingScanBox} />
      <PrivateRoute exact protectedPath path="/repack_select_parcel" component={RepackSelectParcel} />
      <PrivateRoute exact protectedPath path="/repack_items_onto_pallets" component={RepackItemsOntoPallets} />
      <PrivateRoute exact path="/movement_process" component={MovementProcess} />
      <PrivateRoute exact path="/movement_take_item" component={MovementTakeItem} />
      <PrivateRoute exact path="/movement_place_item" component={MovementPlaceItem} />
      <PrivateRoute exact path="/misc_functions" component={MiscFunctions} />
      <PrivateRoute exact path="/view_shelf_content" component={ViewShelfContent} />
      <PrivateRoute exact path="/view_parcel_content" component={ViewParcelContent} />
      <PrivateRoute exact path="/view_item_and_associate" component={ViewItemAndAssociate} />
      <PrivateRoute exact protectedPath path="/associate_item" component={AssociateItem} />
      <PrivateRoute exact path="/stock_take_process" component={StockTakeProcess} />
      <PrivateRoute exact protectedPath path="/stock_take_select" component={StockTakeSelect} />
      <PrivateRoute exact protectedPath path="/stock_take_entry" component={StockTakeEntry} />
      <PrivateRoute exact protectedPath path="/stock_take_recount" component={StockTakeRecount} />
      <PrivateRoute exact path="/reprint_process" component={ReprintProcess} />
      <PrivateRoute exact path="/select_barcode_printer" component={SelectBarcodePrinter} />
      <PrivateRoute exact path="/print_item_label" component={PrintItemLabel} />
      <PrivateRoute exact path="/print_shelf_label" component={PrintShelfLabel} />
      <PrivateRoute exact path="/print_parcel_content" component={PrintParcelContent} />
      <PrivateRoute exact path="/" component={Home} />
    </Switch>
  </div>
</Router>
```

Figura 62: Router

In alcuni casi si nota che è presente un parametro "protectedPath", questo sta a significare che è possibile accedere al componente solamente se si passa attraverso il flusso corretto dell'operazione e non direttamente.

Dato che il contesto di quest'applicazione è puramente aziendale e dato che la sicurezza attuale dell'applicazione e dell'accesso ai servizi è nulla, si è optato per utilizzare la session storage come zona in cui immagazzinare delle informazioni. Per via della sua natura la session storage si azzera non appena viene chiusa la finestra del browser. Si è quindi implementato un sistema di token nel quale una pagina viene caricata solamente se nel session storage è presente il token di login, salvato in essa appena l'autenticazione viene effettuata con successo.

Questo meccanismo della sessione viene utilizzato anche per controllare l'esistenza dei parametri necessari a un componente per poter essere renderizzato. Non appena si preme su un bottone per accedere a una pagina che necessita di parametri passati dal componente attuale, viene settato un token temporaneo che verrà eliminato non appena effettuato il routing, inoltre vengono preparati i parametri per il caricamento del nuovo componente. Così facendo nel momento in cui si tenta di accedere ad una pagina per cui non si ha l'autorizzazione (sia per accesso effettuato attraverso vie traverse che per mancanza di autenticazione) il sistema lo riconosce e porta l'utente su una pagina di errore dedicata o su quella del login a seconda dei casi.

Il componente PrivateRoute ha il compito di caricare il componente corretto, scegliendo tra quello di login, quello di errore o quello richiesto in base a cosa presente nella session storage e a seconda del fatto che sia un percorso protetto o meno.

Il problema che sorge nell'utilizzare un router è che i componenti perdono la relazione di paternità e quindi non è possibile passare i parametri da padre a figlio come avviene normalmente in React. Per questo motivo il passaggio di parametri tra componenti avviene utilizzando l'oggetto history, gestito dal browser, che possiede un metodo push in grado di definire un percorso e gli eventuali parametri.

Nell'immagine seguente è possibile vedere un esempio di redirezione con relativo recupero dei parametri effettuato tra due componenti che vengono chiamati in successione nonostante non siano relazionati direttamente.

```
sessionStorage.setItem("allowedToken", true);
let params = {
    ispatchGroupNumber: dispatchGroupNumber,
    parcelNumber: scannedParcel.parcelNumber
};
props.history.push({ pathname: '/dispatch_pick_item', state: params });
```

```
/*Control received params from prev page*/

let params = props.location.state;

setReceivedState({
    dispatchGroupNumber: params.dispatchGroupNumber,
    parcelNumber: params.parcelNumber,
});
```

Figura 63: Passaggio di parametri

Per quanto riguarda lo stato globale invece è stato utilizzato il pacchetto ReactN che permette una facile gestione di esso. All'interno del file GlobalState.js è presente un vettore di stati che può essere facilmente modificato con gli hooks useGlobal e getGlobal, creando così una sorta di vettore di variabili globali comune a tutti i componenti.

Questo strumento è accessibile semplicemente importando "reactn" al posto di "react" all'inizio del componente desiderato. Trattandosi di una sua estensione tutti i metodi quali useEffect, useState, useCallback ecc sono disponibili anche all'interno di ReactN.

Come visto precedentemente questi stati vengono inizializzati nel file Index.js e in seguito utilizzati nei componenti che richiedono l'utilizzo dello stato globale. Nella versione finale dell'applicazione verranno utilizzati questi stati solamente per i percorsi dei file audio, in modo che se dovessero cambiare basterebbe modificare una sola riga e non sarebbe necessario cambiare tutti i componenti. Nella figura sottostante è possibile vedere il contenuto del file, nel caso serva è possibile estendere il vettore a piacimento.

```
import { setGlobal } from 'reactn';

const DEFAULT_STATE = {
    audioErrorPath: "./beepError.mp3",
    audioOkPath: "./beepOk.mp3",
};

const initState = () => {
    setGlobal(DEFAULT_STATE);
};

export { initState };
```

Figura 64: Definizione stato globale

5.1.4 Menù principale, login/logout e pagina di errore

Dopo aver appreso la parte tecnica riguardante lo sviluppo del front-end è possibile analizzare il risultato finale graficamente, potendolo commentare e in alcuni casi confrontare rispetto alla versione precedente.

La parte più importante dell'applicazione è il menù principale, tramite esso è possibile accedere alle funzionalità primarie e secondarie. Il menù, tuttavia, è accessibile solamente dopo aver eseguito correttamente il login, come il resto dell'applicazione.

Nella versione originale erano presenti due voci aggiuntive nel menù principale: Kit e Nuovo processo, si è deciso di toglierle perché la prima non è stata implementata neanche nella versione precedente mentre la seconda non servirà dato che si è deciso di rendere i processi indipendenti.

Attraverso la voce altro del menù principale si può accedere alle funzionalità secondarie, mentre tutte le altre sono dedicate ai processi principali. Nel caso di errore all'interno dell'applicazione verrà caricato un componente specifico che lo segnalerà.

Nell'immagine seguente è possibile vedere sulla sinistra la schermata di login in cui è possibile inserire le credenziali ed effettuare l'autenticazione. Al centro si può osservare il menù principale abilitato, disponibile una volta eseguito il login. Infine a destra è possibile vedere un esempio della schermata di errore, nel caso specifico viene segnalata la mancata autorizzazione causata da un accesso ad un percorso non permesso.

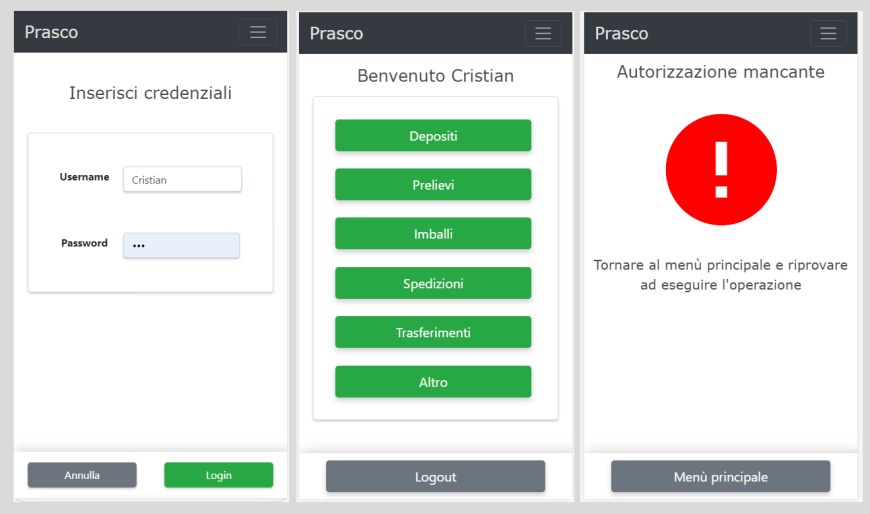


Figura 65: Schermate principali

Se ci si trova in qualunque momento all'interno del menù principale si presume che qualunque altra operazione sia stata terminata, come detto precedentemente però si vuole permettere l'interruzione delle operazioni emulando una sorta di bozza, infatti se si riprova ad eseguire la stessa operazione lasciata in sospeso precedentemente sarà possibile vedere la percentuale di completamento e riprendere da dove si era rimasi.

Tutti questi componenti funzionali sono stati realizzati utilizzando una combinazione di quelli base. Si può infatti notare come il footer sia sempre presente insieme alla navbar, mentre il contenuto variabile sia un'istanza dei componenti descritti nella sezione precedente.

Per quanto riguarda i colori scelti per l'applicazione, essi non sono casuali, infatti si è deciso di mantenere un tema molto simile a quello presente nel sito web del cliente.

5.1.5 Depositi

L'applicazione deve fornire la possibilità di registrare dei depositi una volta ricevute in ingresso le merci. La sezione considerata è composta da tre schermate che è possibile osservare nell'illustrazione sottostante.

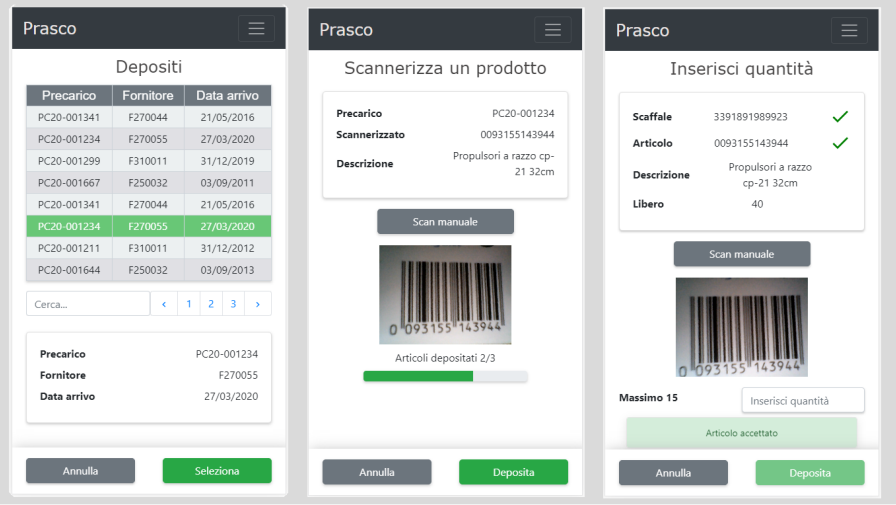


Figura 66: Depositi

A sinistra possiamo vedere la schermata identificata come "SelectPrereceipt", essa permette di selezionare un precarico da una lista richiesta al back-end, per poter registrare un deposito relativo a quel precarico. Quando viene selezionato uno dei precarichi proposti è possibile vedere le informazioni nella card relative a esso ed è possibile proseguire con il deposito ad esso associato.

Al centro è presente l'interfaccia che corrisponde a "PrereceiptItemScan" nella mappa dei form, essa permette di scannerizzare un prodotto che andrà poi depositato. Non è importante quale, basta che ne venga scansionato uno che faccia parte del precarico deciso precedentemente. Come miglioria è stata introdotta una progressbar che indica il numero di prodotti già depositati sul totale da immagazzinare. Nel caso si provi ad annullare l'operazione viene mostrato un modal di conferma, mentre se il prodotto scansionato è corretto e fa parte del deposito associato al precarico selezionato è possibile proseguire.

Infine a destra è possibile vedere la schermata "PrereciptPutAway" che permette di scannerizzare in modo sequenziale lo scaffale di destinazione e nuovamente il prodotto mentre in seguito richiede di inserire la quantità. Qualora tutti i parametri siano corretti il tasto di deposito registra l'operazione e riporta alla schermata intermedia per scannerizzare un nuovo prodotto. Nel caso lo scaffale scansionato sia diverso da quello proposto viene richiesta una conferma tramite modal.

Era stata proposta ed implementata come miglioria la scansione singola prima del prodotto e poi dello scaffale, dato che il fatto di dover scansionare due volte lo stesso prodotto in due schermate differenti a distanza di poco tempo non è molto sensato, ma il cliente ha espressamente richiesto che questa fase venga rieseguita due volte.

Nella figura precedente è riportato un esempio per ogni schermata, ma non vengono presi in considerazione alcuni casi come per esempio la presenza di un errore o il deposito dell'ultimo prodotto (caso in cui la schermata centrale notifica la terminazione del processo non caricando la fotocamera).

Dato che questo è il primo componente presentato viene fornito anche uno scenario alternativo delle schermate che è possibile vedere nella seguente figura sottostante.

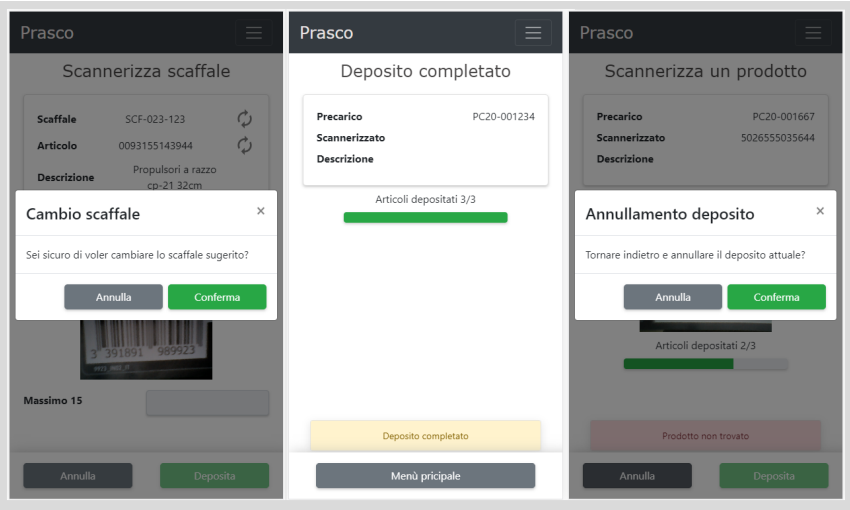


Figura 67: Depositi scenario alternativo

Possiamo osservare sequenzialmente prima il modal che appare se lo scaffale scansionato risulta diverso da quello consigliato, poi la schermata che appare in "PrereceiptItemScan" quando il deposito viene completato. Infine viene mostrato sulla destra il modal che appare quando si cerca di annullare un deposito.

Nell'ultima schermata è anche mostrato un toast che segnala un errore nel caso il prodotto scansionato non sia corretto. Per simulare questa situazione è stato passato un articolo inesistente e poi si è cercato di annullare immediatamente l'intero deposito. Osservando attentamente le schermate è anche possibile notare come siano abilitati o disabilitati i vari bottoni ed input in base alla validità del dato. Il controllo della quantità viene effettuato real-time man mano che si inserisce una cifra causando o meno l'abilitazione del bottone di conferma.

5.1.6 Prelievi

L'operazione di prelevamento è un'altra delle funzionalità primarie richieste, deve essere fornita la possibilità di effettuare dei prelievi dal magazzino in vista della spedizione delle merci. Questa funzione è stata implementata in modo quasi identico a quella di deposito, l'unica grande differenza è che i prodotti da prelevare vengono dati in ordine preciso e non si può scegliere. Nell'immagine sottostante possiamo vedere le tre fasi necessarie per il corretto completamento di essa.

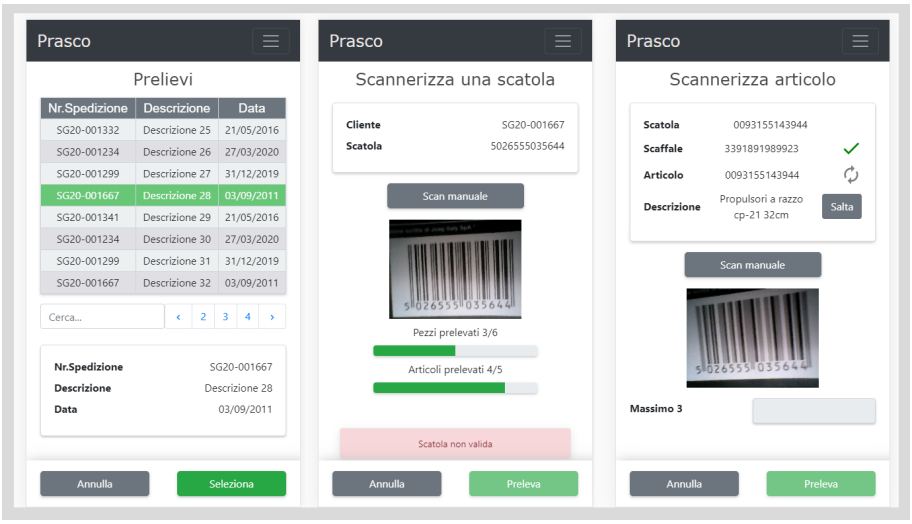


Figura 68: Prelievi

La parte sinistra dell'immagine rappresenta il componente che corrisponde al form "SelectDispatchGroup" in cui è permesso selezionare la spedizione per cui è necessaria l'operazione di prelevamento, anche in questo caso la card sottostante riporta le informazioni relative alla spedizione scelta.

In centro invece è possibile vedere la schermata che appare proseguendo con il processo, corrispondente al "DispatchScanBox" form della versione vecchia. A differenza dei depositi qui è necessario scannerizzare l'UDC relativo al prelievo, identificato anche come scatola. Dalla figura si vede che sono presenti due progressbar, anche in questo caso sono state introdotte per cercare di migliorare la user experience e di dare un'idea all'utente della percentuale di completamento dell'operazione. Per mostrare uno scenario differente è stato catturato il caso in cui la scatola scansionata non sia valida.

Infine sulla destra è possibile vedere l'interfaccia finale relativa ai prelevamenti, denominata "DispatchPickItem". Questa interfaccia permette di prelevare in modo ordinato i prodotti presentati, è necessario che lo scaffale e l'articolo sia-

no gli stessi indicati. Qualora per qualche motivo non sia presente il prodotto nello scaffale indicato è possibile utilizzare il tasto salta per passare al prodotto successivo. L'errore in questo caso verrà risolto direttamente sul database manualmente, come succede attualmente. Questa comunque è una situazione che non dovrebbe mai verificarsi.

A differenza dei depositi, quando viene prelevato un prodotto e viene premuto il tasto preleva nell'ultima schermata, non si ritorna a quella precedente ma automaticamente viene caricato il prossimo prodotto da prelevare, fino ad esaurire il prelievo.

Anche in questo caso se si prova ad annullare il processo dalle ultime due schermate appare un modal di conferma, come quando si decide di saltare il prodotto attuale. Appena l'operazione termina si ritorna alla schermata centrale e viene segnalato il completamento dell'operazione come per i depositi, nascondendo la fotocamera per scannerizzare.

5.1.7 Imballi

L'operazione di imballaggio è la più complessa tra tutte, è necessario garantire che si possano effettuare due tipi di imballi differenti, a seconda del fatto che il materiale venga imballato nella scatola in cui arriva o che venga travasato. I due rami verranno esplorati separatamente, tuttavia la parte iniziale del processo è la stessa in entrambi i casi ed è possibile vederla nella seguente immagine.

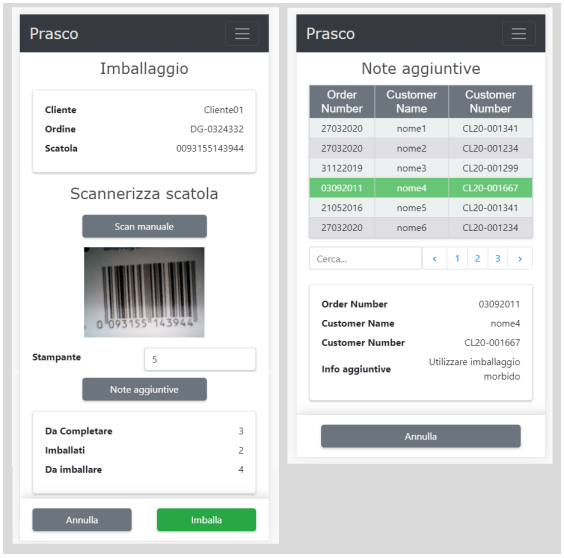


Figura 69: Imballi fase iniziale

A sinistra dell'immagine è possibile vedere la schermata che appare iniziando questa funzionalità, corrisponde al componente denominato "RepackingProcess", per motivi di leggibilità è stata mostrata l'intera schermata estesa, anche se nell'applicazione sarà visibile solo il contenuto sullo schermo e sarà necessario scrollare per vedere il resto. Questa immagine mostra le informazioni che sono ricevute una volta scansionato l'UDC del prodotto che si vuole imballare (il codice relativo alla scatola).

Il processo di imballo prevede che vengano stampate delle etichette, per questa ragione è possibile selezionare una stampante dall'interfaccia. La stampante indicata è quella di default che viene impostata una volta eseguito il login, è possibile cambiarla sia da qui che da un'apposita funzionalità secondaria.

Sempre da questa schermata è possibile accedere alla seconda (attraverso il pulsante "note aggiuntive") identificata come "RepackingSpecialInstructions", da cui è possibile vedere una lista che segnala se sono presenti delle richieste particolari relative all'imballaggio, è possibile che il cliente richieda degli imballi specifici per alcune tipologie di merci.

Confermando l'imballo viene mostrata la schermata visibile nella figura sottostante, da cui è possibile scegliere se imballare nella scatola corrente oppure se travasare in una nuova. Il primo caso affrontato sarà quello di reimballaggio nella stessa scatola, che è possibile vedere nella seguente immagine.

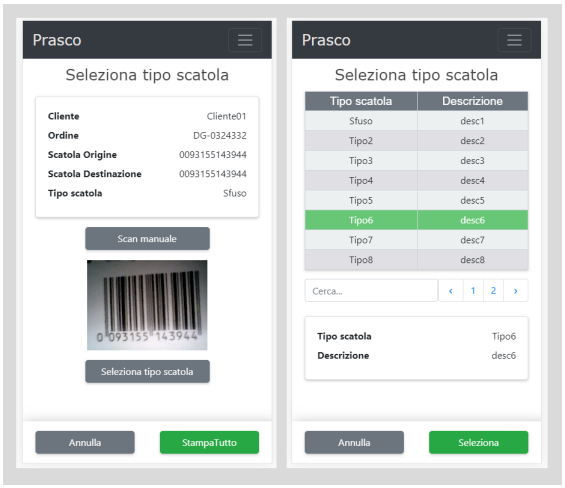


Figura 70: Imballaggio dentro la stessa scatola

Possiamo vedere nella parte sinistra della figura precedente la schermata "RepackingScanBox" nel caso venga selezionata la stessa scatola (tipicamente viene effettuata questa operazione quando si ha del materiale sfuso). Per questa ragione viene richiesta di selezionare una tipologia di scatola attraverso il bottone apposito che apre l'interfaccia sulla destra.

Terminata questa selezione, quando viene premuto il pulsante stampa tutto, verrà automaticamente effettuata la stampa di tutte le etichette necessarie per l'imballo dalla stampante selezionata precedentemente, in una volta sola.

La seconda schermata riporta un elenco di possibili tipologie di scatola tra cui bisogna scegliere, corrisponde al vecchio form "RepackSelectParcel". Le informazioni relative alla scatola selezionata in tabella vengono mostrate nella card inferiore.

L'altra tipologia di imballo invece riguarda la situazione in cui si decida di travasare il vecchio contenuto in una nuova scatola destinazione. Le schermate che si utilizzeranno in questo caso si possono vedere nell'immagine sottostante.

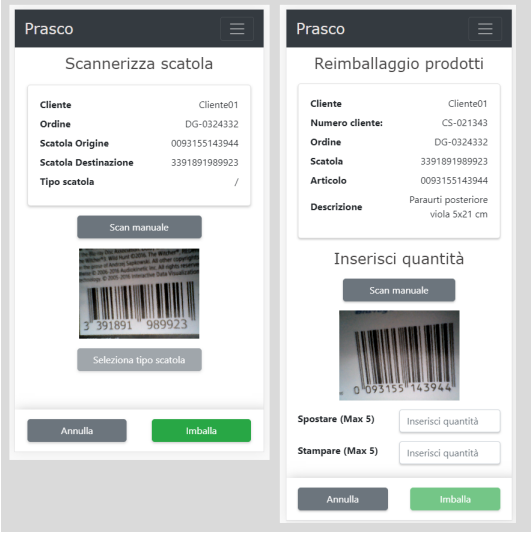


Figura 71: Imballaggio in scatola differente

Anche in questo caso la schermata iniziale è sempre quella denominata "RepackingScanBox", ma il suo layout è differente dato che si è scansionato come scatola di destinazione un UDC differente. Non è necessario selezionare il tipo di scatola perché è già associato alla scatola scelta.

Proseguendo su questo percorso si apre l'interfaccia "RepackItemsOntoPallets" (estesa per migliorare la visibilità), da cui è possibile stampare le etichette singolarmente per ogni prodotto. Per terminare questo processo è necessario scansionare un prodotto alla volta ed inserire la quantità da stampare ed imballare (non necessariamente devono corrispondere) fino ad esaurimento.

Anche nella funzionalità generale riguardante imballi sono presenti dei modal di conferma se si tenta di annullare l'operazione o nel caso si voglia cambiare scatola di destinazione e non usare quella sorgente.

5.1.8 Spedizioni

Il processo di spedizione è stato leggermente modificato su richiesta del cliente. Inizialmente era possibile effettuare una chiusura dell'automezzo da un pulsante apposito situato nella prima schermata, si poteva selezionare l'automezzo desiderato e chiudere in quel modo. Questa funzionalità non veniva mai utilizzata ed è stato esplicitamente richiesto di eliminarla. Per questa ragione la versione finale del processo di spedizione deve solo garantire il caricamento delle merci su un automezzo e la chiusura dello stesso, le interfacce di questa funzionalità sono visibili nella seguente figura.

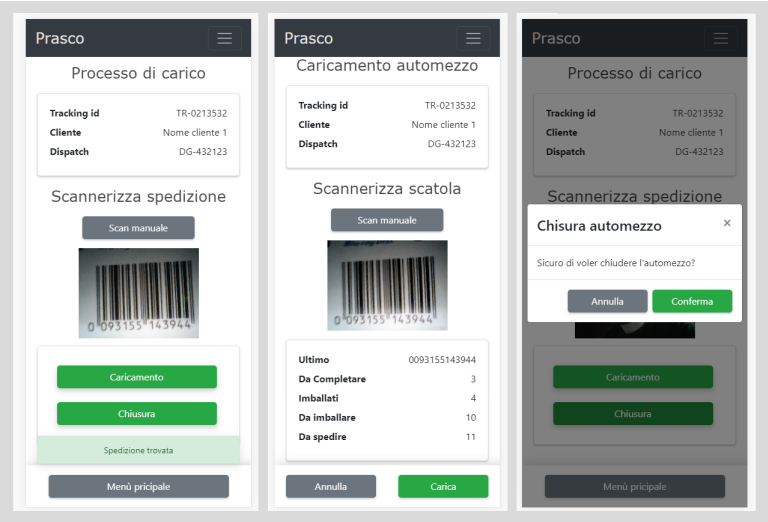


Figura 72: Spedizioni

La funzionalità è composta da due soli componenti, "LoadingProcess" a sinistra e "LoadingLoadParcel" al centro. Come è possibile vedere dall'immagine se si richiede una chiusura di una spedizione scansionata appare il modal di conferma sulla destra, mentre se si sceglie di caricare delle merci sull'automezzo il processo continua con l'interfaccia centrale.

Se si tenta di annullare una spedizione dopo aver scansionato il relativo codice apparirà il modal di conferma. Quando invece si effettua un caricamento è necessario scansionare la scatola imballata da caricare, una volta registrato il caricamento le statistiche verranno aggiornate.

Una richiesta espressa dal cliente è stata quella di poter chiudere un automezzo anche senza aver caricato tutti gli articoli relativi ad una spedizione. Questo potrebbe essere utile nel caso ci siano troppi prodotti per cui non basterebbe un solo automezzo, per come è stata implementata la chiusura dell'automezzo questa operazione è possibile in ogni istante.

5.1.9 Trasferimenti

I trasferimenti sono l'ultima funzionalità primaria richiesta in questo progetto. Grazie a questo processo è possibile spostare le merci all'interno del magazzino da una parte all'altra in modo che lo spostamento venga registrato. La figura sottostante mostra come sia possibile eseguire queste operazioni.

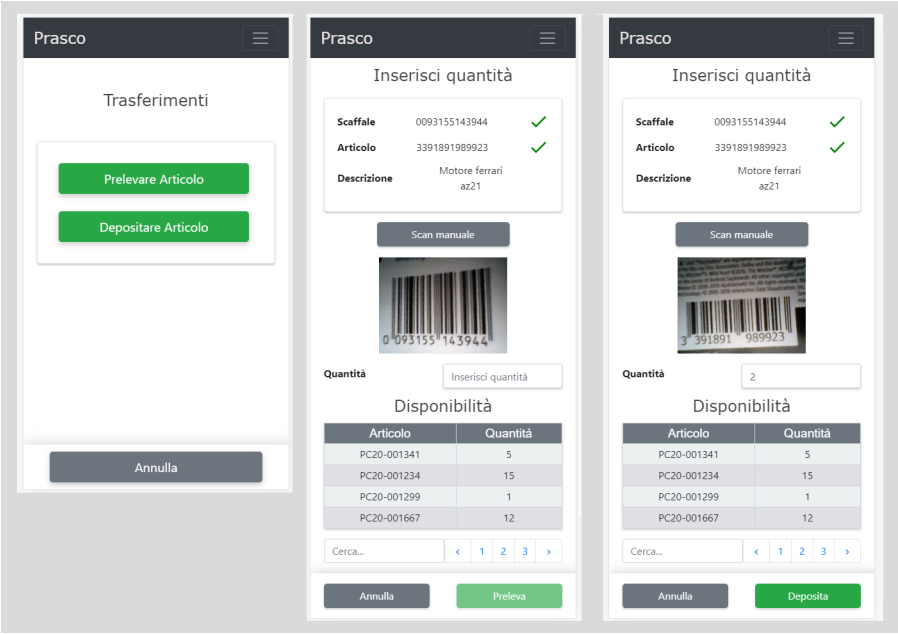


Figura 73: Trasferimenti

Nella schermata di sinistra ("MovementProcess") è possibile vedere il menù che appare una volta iniziato questo processo. Grazie a esso è possibile decidere se registrare il prelievo di un prodotto o se memorizzarne il deposito.

Al centro e a destra (anche in questo caso allungate per visibilità) possiamo vedere le schermate che permettono rispettivamente di prelevare e depositare un prodotto. Per entrambe le operazioni è necessario scannerizzare prima lo scaffale e poi il prodotto per cui bisogna registrare il deposito/prelievo.

Infine per garantire che l'operazione avvenga con successo occorre inserire la quantità dell'oggetto e confermare. La schermata centrale viene denominata "MovementTakeItem" mentre quella di destra "MovementPlaceItem" rispecchiando i nomi assegnati ai vecchi form dell'applicazione. In entrambe le schermate successive alla scelta si può vedere una lista di articoli presenti su cui effettuare l'operazione. Nel momento che si cerca di tornare indietro avendo scansionato qualcosa di corretto appare un modal di conferma.

5.1.10 Altro

Presentate tutte le funzionalità primarie che si legano ai processi di gestione del magazzino descritti a inizio tesi è possibile dare un rapido sguardo alle funzionalità secondarie.

Come detto a inizio capitolo le funzionalità secondarie non si associano a dei veri e propri processi di gestione del magazzino, tuttavia risultano essere molto utili se non essenziali per il corretto funzionamento di alcuni di essi, oltre che a migliorare l'operato dei dipendenti.

Attraverso la voce "Altro" della schermata principale è possibile accedere al menù delle operazioni secondarie, chiamato "MiscFunctions" che viene rappresentato nella figura sottostante.

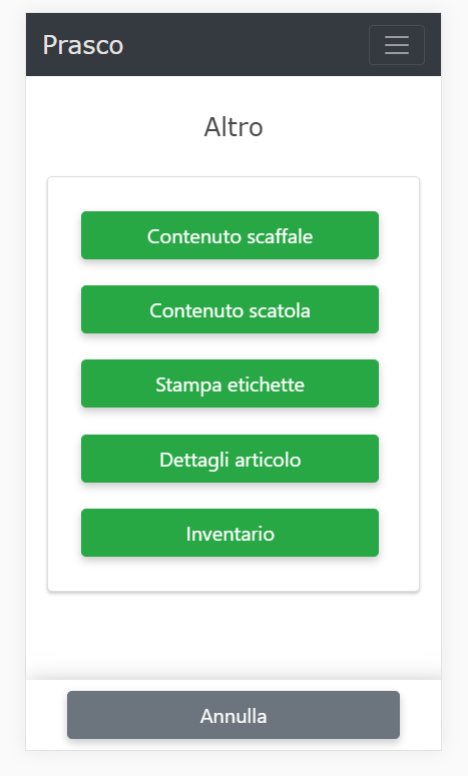


Figura 74: Altro

Le prime due funzionalità elencate servono solamente a scopo informativo per poter conoscere il contenuto di uno scaffale o di una scatola. Nella prossima immagine possiamo vedere a sinistra l'interfaccia che corrisponde al form "ViewShelfContent" mentre a destra quella denominata "ViewParcelContent".

Anche in questo caso si è deciso di mettere all'interno della figura tutta la schermata per comprenderne meglio il funzionamento, nell'applicazione si potrà scrollare per proseguire verso il basso.

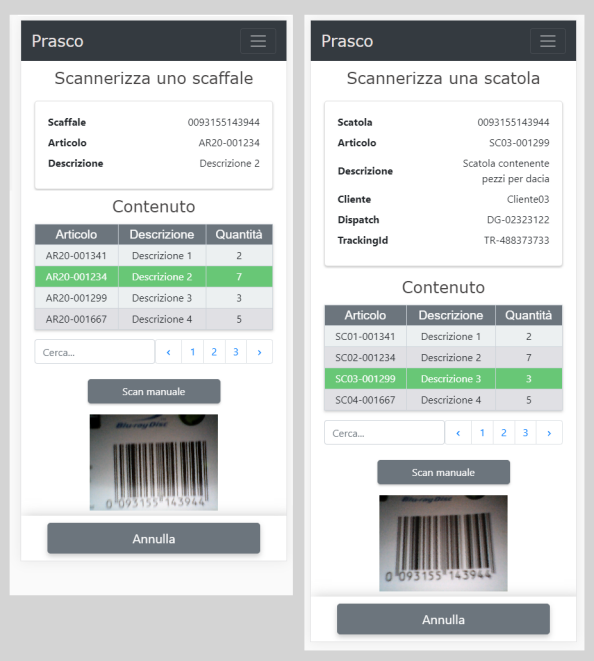


Figura 75: Contenuti

In entrambi i casi è sufficiente scansionare il codice a barre dello scaffale o della scatola in modo che appaia una lista con il contenuto visionabile. Il menù delle funzionalità secondarie è accessibile senza la presenza di alcun modal quando viene premuto il pulsante annulla.

A seconda della funzionalità una serie di informazioni aggiuntive verranno mostrate nella relativa card quando si selezionerà un prodotto dalla lista per visionarne le caratteristiche.

La terza funzionalità secondaria, denominata "Stampa Etichette", permette di eseguire svariate operazioni che possiamo raggruppare in due tipologie. La prima permette di selezionare la stampante da utilizzare che verrà memorizzata all'interno dell'applicazione mentre la seconda riguarda la stampa delle etichette relative al contenuto di un contenitore o ad un item.

Nell'immagine sottostante è possibile notare a sinistra il menù riguardante la stampa delle etichette, "ReprintProcess", con indicata la stampante attualmente impostata come predefinita per l'applicazione. A destra invece è possibile vedere la prima delle opzioni che mostra l'elenco delle stampanti disponibili e la loro posizione, la seconda schermata è denominata "SelectBarcodePrinter".

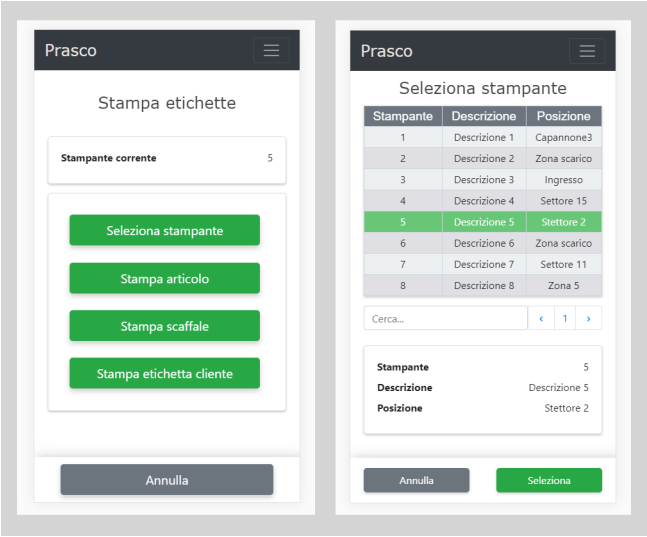


Figura 76: Menù di stampa e selezione stampante

Per quanto riguarda la seconda tipologia di stampa, essa racchiude le tre voci rimanenti per stampare rispettivamente il codice a barre di un articolo, di uno scaffale o del contenuto di una scatola per il cliente.

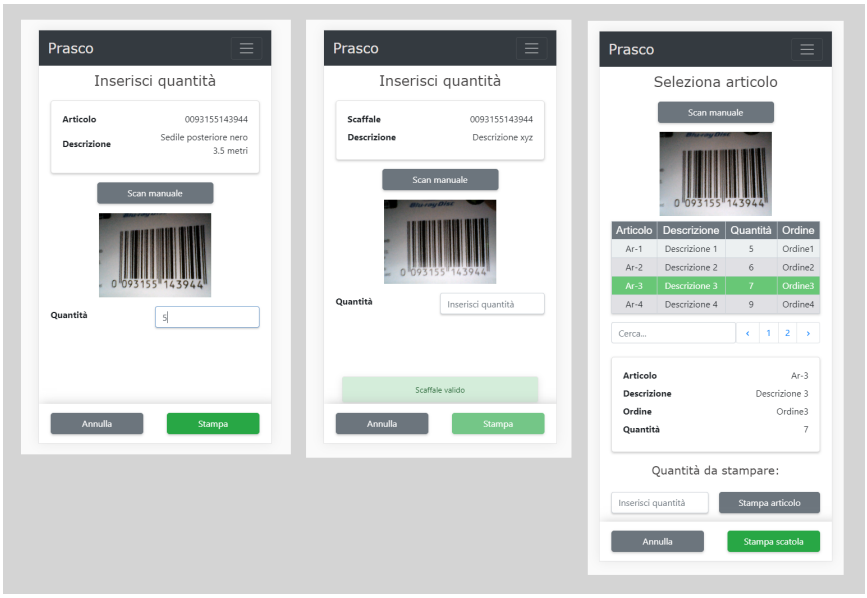


Figura 77: Stampa contenuti

Nella figura precedente è mostrato a sinistra il componente "PrintItemLabel", al centro quello chiamato "PrintShelfLabel" e infine a destra quello corrispondente al form "PrintParcelContent".Nei primi due casi è sufficiente scansionare l'articolo o lo scaffale a seconda della funzione e inserire il numero di etichette da stampare prima di confermare.

Per quanto riguarda il terzo caso, una volta scansionata la scatola è possibile decidere se stampare solo l'etichetta di uno degli articoli presenti in essa, selezionandolo dalla lista, con la relativa quantità, oppure se stampare tutte le etichette di tutta la scatola in un colpo solo.

Da tutte le interfacce è possibile ritornare nel menù delle funzionalità secondarie attraverso il relativo tasto per annullare.

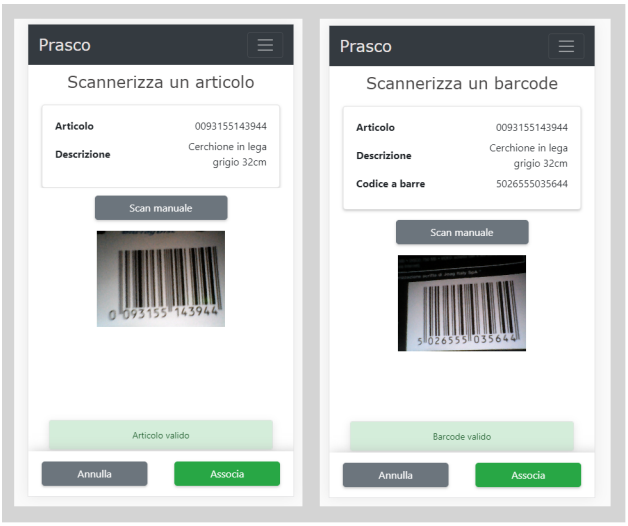


Figura 78: Dettagli articolo e associazione

La quarta funzionalità secondaria che viene messa a disposizione è quella che permette di vedere i dettagli di un particolare articolo e che fornisce la possibilità di cambiarne il codice a barre associato.

Nella figura soprastante è possibile vedere a sinistra la schermata "Associate item" che permette di scannerizzare un articolo e vedere la relativa descrizione. Se si decide di continuare si arriva sull'interfaccia di destra, denominata "ViewItemAndAssociate". Dall'ultima interfaccia è possibile scannerizzare un barcode e associarlo a quel prodotto, in entrambi i casi un modal richiede la conferma se si vuole annullare l'operazione.

Infine come ultima funzionalità secondaria abbiamo la gestione dell'inventario. Si tratta di un'operazione limitante, in quanto è possibile solamente modificare le quantità di un prodotto oppure eseguire un riconteggio, tuttavia in alcune occasioni può essere utile.

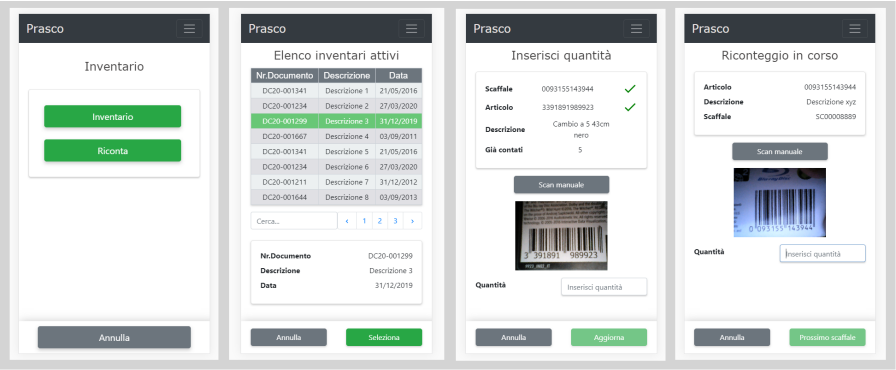


Figura 79: Inventario

Dall'immagine si evince che il menù di scelta identificato come "StockTakeProcess" è visibile sulla sinistra, mentre proseguendo verso il centro è possibile notare la schermata in cui viene richiesto di selezionare un documento relativo all'inventario (interfaccia chiamata "StockTakeSelect").

Sulla destra si possono notare in ordine le interfacce "StockTakeEntry" e "StockTakeRecount". Nella prima viene richiesto di scannerizzare uno scaffale e in seguito un articolo per poi modificarne la quantità. Nell'altra invece, una volta scansionato il prodotto, vengono mostrati a uno a uno tutti gli scaffali in cui è presente l'item, fornendo la possibilità di modificarne la quantità mentre la si controlla. In entrambe le ultime due schermate appare un modal di conferma se si tenta di uscire dalla funzionalità.

5.1.11 Creazione delle API con Axios

Per la comunicazione con il back-end si è deciso di utilizzare Axios, che permette di renderizzare delle richieste asincrone [52]. I file che gestiscono le chiamate Axios hanno il compito di esporre dei metodi che verranno chiamati dall'applicazione (quindi dai componenti funzionali) e che restituiranno i dati forniti dal back-end.

I metodi delle API sono stati sviluppati secondo una ferrea struttura, per questo motivo sono tutti molto simili tra loro. A seconda che il back-end restituisca un risultato corretto (status code 200) o che restituisca un errore (presentando un codice 500, 404 e così via) l'applicazione si comporterà di conseguenza mostrandolo o visualizzando la pagina di errore con un relativo messaggio.

La suddivisione delle API è organizzata seguendo le macro categorie funzionali mantenendo una corrispondenza uno a uno con i controller del back-end descritti in seguito. La struttura complessiva è riportata nella figura seguente.

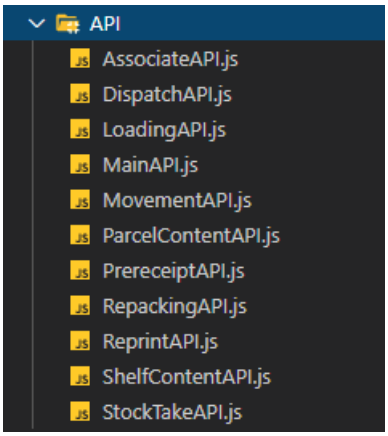


Figura 80: Struttura API

Per quanto riguarda la struttura dei metodi delle API, essa consiste in una chiamata di tipo POST o GET ,a seconda dei casi, ad un URL che è formato da una base globale e da un controller specifico che dipende del tipo di API. Nell'immagine sottostante è possibile vedere una chiamata al back-end e in particolare come i parametri passati al metodo vengano organizzati sotto forma di JSON per poi essere passati come parametro.

```
const axios = require('axios');
const url = 'https://localhost:44382/api';
const controller = '/prereceipt';

/*SelectPrereceiptAPI*/

export const PutAwayPreReceipt = (pDAUserIn, item, shelfNo, quantity) => {
    const params = {
        pDAUserIn: pDAUserIn,
        item: item,
        shelfNo: shelfNo,
        quantity: quantity,
    };
    return axios.post(url + controller + '/PutAwayPreReceipt', params).then(resp => {
        return {
            success: true,
            redirect: false,
            data: resp.data,
        };
    }).catch(error => {
        return {
            success: false,
            redirect: error.response === undefined || error.response.status === 500,
            data: error.response === undefined ? 'Errore di connessione' : error.response.data,
        };
    });
};
```

Figura 81: Metodo API

Una volta ricevuta la risposta viene effettuato il forwarding al metodo chiamante tramite il return, l'oggetto restituito è sempre composto da:

- **Success**: un flag che viene utilizzato per vedere se la chiamata è andata a buon fine o meno
- **Redirect**: un booleano che stabilisce se è necessario fare una redirezione alla pagina di errore o meno
- **Data**: la risposta vera e propria, può essere sia un messaggio (in caso di errore) che il contenuto vero e proprio che ci si aspetta quando si effettua la chiamata

Nel caso non si riesca a contattare il server viene mostrato un errore specifico nel componente dedicato. A seconda dei casi e delle operazioni è necessario effettuare una redirezione oppure mostrare l'errore utilizzando degli appositi toast e beep di errore per fare ciò si utilizza il flag di redirect, creato per questo scopo.

La scelta dei nomi dei parametri è stata fatta considerando quelli della versione vecchia, per una questione di comprensione si è deciso di non cambiare il nome delle vecchie API del client e delle variabili, in modo da rendere più comprensibile la mappatura tra modelli per l'applicazione e modelli di servizio, argomento approfondito nella prossima sezione.

Durante la spiegazione dei controller verrà anche illustrato il motivo dell'utilizzo di diversi URL con solo un metodo ciascuno. Solitamente la proceduta che viene adottata è quella di avere più operazioni che vengono effettuate su un'unica risorsa REST.

5.2 Sviluppo back-end: struttura del progetto

Come anticipato nell'introduzione di questo capitolo, per quanto riguarda lo sviluppo del back-end è stato investito meno tempo rispetto al front-end dato che l'implementazione dei servizi con la relativa connessione al database era già stata sviluppata. L'ambiente utilizzato è Visual Studio 2019 e il linguaggio usato C#, dall'immagine seguente è possibile avere un'idea della struttura del progetto.

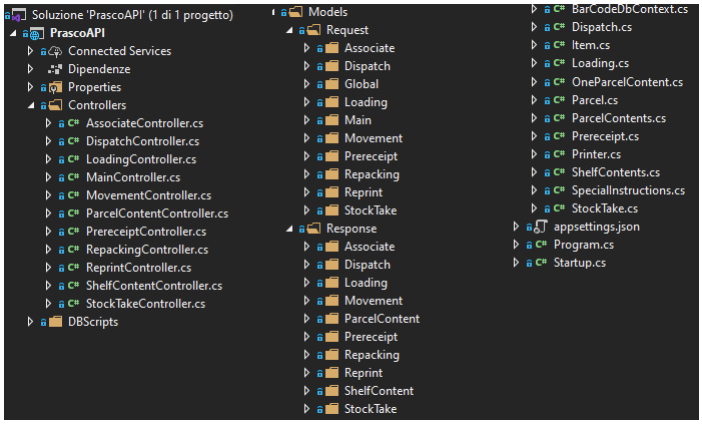


Figura 82: Struttura file back-end

Si può notare dall'immagine la presenza di diversi controller che del nome richiamano le macro funzionalità dell'applicazione. Inoltre si vede anche la struttura dei modelli di dati, divisi in cartelle differenti a seconda della loro natura di risposta o di richiesta. I file nella cartella modelli verranno accennati nella sezione riguardante l'implementazione del database Mock. Il progetto presenta anche il file Program.cs e quello Startup.cs dato che si tratta di una soluzione in Visual Studio. Non è necessario apportare particolari modifiche a questi due file se non per l'integrazione con il database dei dati mock o per la creazione di una variabile d'ambiente che modifichi il comportamento di tutto il back-end in base al suo valore, come si vede nella figura sottostante.

```
public static void Main(string[] args)
{
    Environment.SetEnvironmentVariable("mock","true");
    CreateHostBuilder(args).Build().Run();
}
```

Figura 83: Variabile d'ambiente

Il file appsetting.json memorizza delle costanti utili all'applicazione per funzionare, mentre nella cartella delle dipendenze possiamo trovare tutto ciò che è necessario al progetto per essere eseguito (contenuto creato in automatico dall'ambiente di sviluppo).

La zona relativa ai connected services invece verrà spiegata nella sezione successiva relativa alla gestione e alla creazione del proxy. Per quanto riguarda la cartella DBScript, essa contiene solo dei file contenenti istruzioni SQL che possono essere eseguiti per creare le tabelle dei dati mock nel db appropriato.

Quando il back-end viene eseguito parte sulla porta 44382 in modalità sicura utilizzando https, questi dati sono stati impostati in fase di creazione del progetto ed è necessario fornirli al front-end per poter comunicare correttamente.

5.2.1 Chiamata ai servizi con protocollo SOAP e proxy

Per quanto riguarda i servizi WCF, l'unica cosa fornita è stata una copia del documento WSDL che definisce i parametri necessari ed i relativi nomi (sia dei parametri che dei servizi).

Grazie agli strumenti presenti in VS2019 è stato possibile importare il documento in modo che venisse automaticamente generato un file che solitamente si chiama proxy, esso rappresenta l'implementazione di tutti i metodi di servizio nel progetto attuale che possono essere utilizzati dai controller. Oltre a questo il proxy fornisce la definizione di tutti i modelli di cui il servizio necessita (diversi da quelli che si vedono nella foto della struttura del back-end).

Nelle immagini seguenti è possibile vedere un estratto del proxy, il file è molto lungo, più di 5.000 righe, perciò sono stati presi dei frammenti di esso.

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.Tools.ServiceModel.Svcutil", "2.0.2")]
[System.ServiceModel.MessageContractAttribute(WrapperName="RepackSourceBoxOneOrderResponse", WrapperNamespace="it.prasco.wh", IsWrapped=true)]
public partial class RepackSourceBoxOneOrderResponse
{

    [System.ServiceModel.MessageBodyMemberAttribute(Namespace="it.prasco.wh", Order=0)]
    public bool RepackSourceBoxOneOrderResult;

    [System.ServiceModel.MessageBodyMemberAttribute(Namespace="it.prasco.wh", Order=1)]
    public string customerNo;

    [System.ServiceModel.MessageBodyMemberAttribute(Namespace="it.prasco.wh", Order=2)]
    public string customerName;

    [System.ServiceModel.MessageBodyMemberAttribute(Namespace="it.prasco.wh", Order=3)]
    public string orderNo;

    public RepackSourceBoxOneOrderResponse()
    {
    }

    public RepackSourceBoxOneOrderResponse(bool RepackSourceBoxOneOrderResult, string customerNo, string customerName, string orderNo)
    {
        this.RepackSourceBoxOneOrderResult = RepackSourceBoxOneOrderResult;
        this.customerNo = customerNo;
        this.customerName = customerName;
        this.orderNo = orderNo;
    }
}
```

Figura 84: Proxy definizione risposta

Nella figura superiore è possibile vedere la definizione di un modello di risposta a un servizio, con tutti i relativi campi e annotazioni, mentre nella figura seguente si vede l'implementazione di alcuni servizi.

Come detto precedentemente viene utilizzato il protocollo SOAP, perciò tutto avviene in modo asincrono. Questo proxy utilizza XML come linguaggio per parsificare i dati, mentre contrariamente JSON verrà utilizzato per la comunicazione tra front-end e back-end.

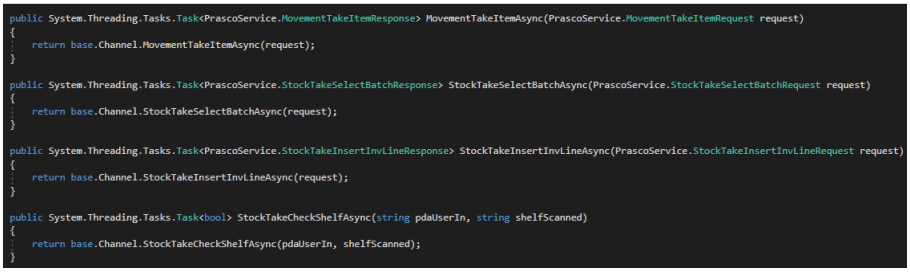


Figura 85: Proxy definizione servizio

I codici delle immagini possono risultare difficili da leggere data la loro dimensione, ma queste figure sono solamente a scopo informativo per dare un'idea di cosa si può trovare all'interno del proxy o di come funziona dato che viene generato automaticamente.

I modelli che vengono utilizzati dai servizi sono quelli presenti all'interno del proxy, che non necessariamente corrispondono con quelli che l'applicazione necessita e utilizza. La sezione seguente sarà utile a capire la differenza tra le due tipologie.

5.2.2 Modelli

I modelli non sono altro che degli oggetti che vengono mandati tra due componenti software per comunicare informazioni. Nel caso in esame ci sono due tipi di modelli, il primo viene descritto nella sezione precedente e si trova all'interno del proxy (modelli utilizzati per la comunicazione tra i servizi WCF e il back-end), mentre quelli facenti parte della seconda tipologia secondi sono stati creati tutti manualmente e sono impiegati nella comunicazione tra front-end e back-end.

Per quanto riguarda il secondo tipo di modelli è possibile osservare la figura che rappresenta la struttura del back-end mostrata poche pagine prima. Guardandola si nota come siano presenti tante cartelle, una per ogni controller sia all'interno della cartella request che di quella response. I file contenuti al loro interno rappresentano tutti modelli creati appositamente per la comunicazione con il front-end e quasi sempre sono degli oggetti che hanno come variabili un sottoinsieme di quelle del modello del proxy, raramente qualcuna in più.

Nell'immagine sottostante è possibile vedere un esempio di un modello, esso rappresenta solamente un oggetto, in questo caso un insieme di quattro variabili che rappresentano la richiesta effettuata per il servizio denominato "Loading-PlaceOnVehicle". Questi dati verranno forniti dal front-end al controller che li mapperà in modo esplicito ai dati presenti nel modello del proxy con cui verrà effettuata la chiamata al servizio.

```
namespace PrascoAPI.Models.Request.Loading
{
    public class LoadingPlaceOnVehicleRequest
    {
        public string PDAUserIn { get; set; }
        public string ParcelCode { get; set; }
        public string Customer { get; set; }
        public string Dispatch { get; set; }
    }
}
```

Figura 86: Modelli

5.2.3 Controller e implementazione delle API

I controller hanno il compito di ricevere le richieste effettuate dal front-end tramite Axios , elaborarle e fornire una risposta. A seconda che la variabile d'ambiente sia settata o meno per utilizzare i dati mock il controller si comporterà di conseguenza. In ogni metodo al suo interno è presente un controllo che permette di eseguire le richieste utilizzando o il db dei dati di test o la chiamata ai servizi veri in base al caso in questione.

```
[Route("api/reprint")]
[ApiController]
public class ReprintController : ControllerBase
{
    /*Controller variables*/

    private readonly IConfiguration _configuration;
    private readonly bool mock;
    private readonly ServiceClient serviceClient = new ServiceClient();

    /*Controller constructor*/

    public ReprintController(IConfiguration configuration)
    {
        _configuration = configuration;
        mock = _configuration.GetSection("Mock").Get<bool>();
    }

    SelectBarcodePrinter API

    PrintItemLabel API

    PrintParcelContent API

    PrintShelfLabel API

}
```

Figura 87: Controller

Nell'immagine soprastante è possibile vedere la struttura di uno dei controller, viene presa la variabile mock dall'ambiente per capire quale sia il comportamento desiderato nel costruttore e poi vengono scritti tutti i metodi all'interno del controller. Un'altra cosa che si può notare e che viene creato il serviceClient che verrà utilizzato per fare le chiamate ai servizi che il proxy gestirà successivamente.

Per avere un codice più ordinato e pulito si è deciso di utilizzare le region che sono uno strumento che permette di definire delle macro zone che possono essere facilmente racchiuse e compattate a seconda delle esigenze. Nel caso in esame sono state create le regioni con corrispondenza uno a uno con i vari componenti funzionali del front-end.

Osservando la figura sottostante invece si può notare come siano stati strutturati i metodi del controller. Innanzitutto vengono messe una serie di notazioni per documentare la natura del metodo, nel caso mostrato si tratta di un metodo POST che consuma e produce JSON e che può mandare risposte avente come status code 200, 409 e 500 a seconda dei casi.

```
/*StockTakeEntry API*/

    [HttpPost("StockTakeInsertInvLine")]
    [Consumes(MediaTypeNames.Application.Json)]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status409Conflict)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    public async Task<JsonResult> StockTakeInsertInvLine(Models.Request.StockTake.StockTakeInsertInvLineRequest request)
    {
        if (mock)
            return new JsonResult(true) { StatusCode = 200 };
        else
        {
            try
            {
                StockTakeInsertInvLineRequest serviceRequest = new StockTakeInsertInvLineRequest {
                    documentNumber = request.DocumentNumber,
                    navItemNo = request.NavItemNo,
                    pdaUserIn = request.PdaUserIn,
                    quantityCounted = Convert.ToInt32(request.QuantityCounted),
                    shelf = request.Shelf
                };
                var res = await serviceClient.StockTakeInsertInvLineAsync(serviceRequest);
                if (res.errorReturned == "") return new JsonResult(true) { StatusCode = 200 };
                else return new JsonResult(res.errorReturned) { StatusCode = 409 };
            }
            catch (Exception ex)
            {
                return new JsonResult(ex) { StatusCode = 500 };
            }
        }
    }
```

Figura 88: Metodo del controller

Tutte le risposte sono di tipo JesonResult sia in caso di risposta alla chiamata al servizio che di utilizzo di dati mock. La prima operazione che viene effettuata dai metodi è il controllo per sapere quale delle due strade precedentemente descritte utilizzare. Se se si è scelto di utilizzare i mock e sono necessarie informazioni quali liste di articoli o spedizioni e via dicendo sarà necessario connettersi al database, come verrà illustrato in seguito, mentre nel caso basti una risposta semplice o booleana come nell'immagine soprastante essa verrà direttamente mandata.

Nel caso sia richiesto di connettersi al servizio WCF è necessario effettuare una o più elle seguenti operazioni a seconda dei casi:

- **Mappatura della richiesta**: come possibile vedere nell'immagine precedente a volte è necessario mappare il modello di richiesta ricevuto con quello che il servizio si aspetta, in questo caso i campi vengono esplicitamente inseriti, mentre quelli mancanti vengono inizializzati a null
- **Chiamata asincrona al servizio**: questa operazione è sempre necessaria, viene utilizzato l'oggetto serviceClient per chiamare il servizio, passando i relativi parametri come richiesta e ricevendo una risposta (tranne in rari casi in cui i servizio sia di tipo void)
- **Mappatura della risposta**: in alcuni casi è necessario mappare anche la risposta, magari perché si tratta di un modello e non un booleano come nell'esempio, le operazioni da eseguire sono le stesse fatte per la richiesta ma a contrario, dato il modello del proxy viene costruito quello da mandare al front-end.

Un'importante considerazione da fare è la necessità di introdurre sempre un try catch per gestire il caso in cui qualcosa vada storto runtime nella chiamata del servizio. In questo modo è possibile restituire un errore di tipo 500 e il front-end potrà reagire con una redirezione mostrando la pagina di errore.

Nell'implementazione dei controller si sono effettuate alcune scelte che nella situazione corrente sono risultate le più efficaci ed appropriate, se necessario tuttavia è possibile metterle in discussione ed eventualmente modificare il software.

In primo luogo si è deciso di far corrispondere ad un indirizzo diverso, quindi a una risorsa differente, ogni singolo servizio, questo principio va contro la politica REST, la quale stabilisce che sulle risorse dovrebbe essere possibile una politica di tipo CRUD (Create, Read, Update e Delete). Analizzando però i servizi esistenti si nota che le uniche operazioni effettuate sono delle GET e delle POST a degli URL differenti. Per questa ragione si è deciso di implementare delle risorse su cui fosse possibile una sola operazione di GET o di POST a seconda dei casi.

Un'altra scelta che è stata fatta è quella di restituire in alcuni casi un errore di tipo 409. Questo status code segnala un conflitto, ed è quello che meglio rispecchia la situazione in cui viene richiesta un'operazione che non è possibile effettuare per via dello stato attuale della risorsa. Per questa ragione a seconda che la risposta del servizio sia positiva o di errore verrà ritornato un 200Ok o un 409Conflict come status, per segnalare al front-end l'esito dell'operazione e per permettere ad esso di poter attuare un comportamento differente a seconda dei casi.

5.2.4 Implementazione del database per i dati mock

Per poter simulare un'interazione con il database sono state create delle tabelle in un apposito DB per poter contenere i dati mock. Tramite Visual Studio vengono effettuate automaticamente le query utilizzando un contesto e uno strumento chiamato EntityFrameworkCore che permette la gestione dei DB in .NET.

La struttura del database è molto semplice ed è possibile vederla in figura, sono presenti le tabelle relative alle richieste di liste di dati, se per esempio l'accesso al DB serve per una sola conferma o per una singola tupla non viene adottato questo approccio e viene direttamente restituito il valore.

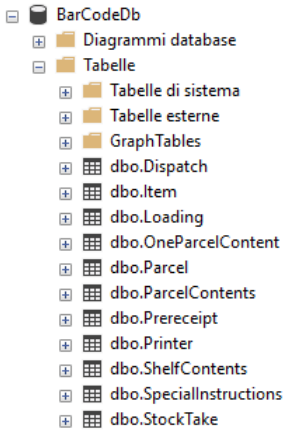


Figura 89: Struttura Database Mock

Nel caso vengano richiesti i dati mock si utilizza il contesto appositamente creato per le interazioni con il database. Si è deciso di utilizzare un approccio DB-First, ciò significa che il database viene implementato prima e poi grazie agli strumenti offerti da Visual Studio vengono generate automaticamente le classi corrispondenti alla tabelle.

Nell'immagine seguente è possibile vedere l'istruzione che se lanciata sul terminale di gestore di pacchetti di Visual Studio è in grado di creare automaticamente il contesto con tutte le mappature delle tabelle.

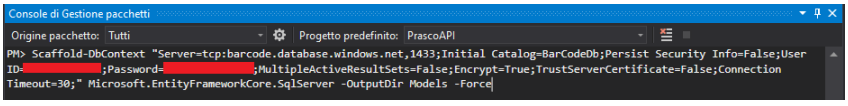


Figura 90: Creazione DbContext

Le classi vengono generate nella cartella Modelli insieme al contesto, ogni file contiene solamente le definizioni dei tipi e dei nomi delle colonne nelle tabelle, in maniera molto simile ai modelli presenti nell'applicazione. La figura sottostante mostra un esempio di mappatura all'interno del DbContext.

```
modelBuilder.Entity<Prereceipt>(entity =>
{
    entity.Property(e => e.DateTimeArrived)
        .HasColumnName("Date_Time_Arrived")
        .HasColumnType("date");

    entity.Property(e => e.Name)
        .IsRequired()
        .HasMaxLength(255)
        .IsUnicode(false);

    entity.Property(e => e.VendorNo)
        .IsRequired()
        .HasColumnName("Vendor_No")
        .HasMaxLength(255)
        .IsUnicode(false);
});

OnModelCreatingPartial(modelBuilder);
```

Figura 91: Mappatura interna al DbContext

La chiamata al database è fatta utilizzando i metodi della classe contesto, nell'immagine seguente è possibile vedere un'implementazione di un metodo mock che restituisce i dati presi dal database relativi alla lista di precarichi.

```
if (mock)
{
    PreReceiptHeaderLine_ReadMultipleResponse response = new PreReceiptHeaderLine_ReadMultipleResponse();
    List<PrececeiptMock> lines = new List<PrececeiptMock>();

    BarCodeDbContext context = new BarCodeDbContext();
    var result = context.Prereceipt.OrderBy(el => el.Name);
    foreach (var item in result)
    {
        PrececeiptMock prereceipt = new PrececeiptMock
        {
            No = item.Id.ToString(),
            Name = item.Name.ToString(),
            Vendor_No = item.VendorNo.ToString(),
            Date_Time_Arrived = item.DateTimeArrived.ToString().Split(" ")[0]
        };
        lines.Add(prereceipt);

    }
    response.Lines = lines;

    return new JsonResult(response) { StatusCode = 200 };
```

Figura 92: Chiamata al DB Mock

Data l'incompatibilità dei nomi delle colonne con Visual studio dovuta al simbolo "_" si è deciso di mappare i valori ritornati con i modelli di response. Questa scelta è stata adottata per mantenere i nomi dei modelli di risposta uguali a quelli originariamente presenti, per una comprensione più intuitiva del software.

5.3 Considerazioni sullo sviluppo

In questa parte finale del capitolo verranno presentate brevemente alcune considerazioni generali relative allo sviluppo della tesi e alle scelte effettuate nell'implementazione.

In primo luogo verranno presente delle analisi relative allo studio della sicurezza effettuato sia durante l'implementazione che nello studio del progetto. Successivamente verranno brevemente illustrati gli strumenti utilizzati e gli artefatti relativi all'utilizzo della metodologia agile per lo sviluppo della tesi.

5.3.1 Analisi della sicurezza dell'applicazione

Uno degli aspetti più importanti nello sviluppo di un software in generale è la gestione della sicurezza. Non è mai da sottovalutare il fatto che la sicurezza è un processo, non un prodotto, quindi solitamente è necessaria un'analisi costante che accompagni tutte le fasi del progetto.

Durante lo studio del software attuale, descritto nel capitolo precedente a questo, si ha avuto modo di esaminare il codice e capire la sicurezza adottata in quel caso. Per quanto riguarda il codice dell'applicazione e del client la sicurezza era totalmente assente, ed era possibile attaccare l'applicazione in tutti i modi.

Questa scelta probabilmente è stata effettuata dato che l'applicazione viene utilizzata solamente da dipendenti all'interno del magazzino per gestire i prodotti, e quindi i rischi di possibili attacchi sono davvero molto bassi. Non è possibile vedere le misure di sicurezza che sono state adottate all'interno dei servizi perché l'accesso al codice sorgente non è stato fornito. Tuttavia l'analisi del client e dell'applicazione mostra chiaramente come non sia minimamente stata presa in considerazione un'introduzione di precauzioni contro attacchi di tipo informatico.

Per questa ragione durante lo svolgimento della tesi si è dato priorità allo sviluppo delle funzionalità piuttosto che alla sicurezza, tuttavia alcuni controlli base e alcune restrizioni sono state implementate.

Prima di tutto si è introdotto il meccanismo delle route private in modo da non permettere alcuna operazione ad utenti che non hanno effettuato il login. Lo stesso strumento viene utilizzato per evitare che manualmente si acceda a pagine/componenti che non sarebbero accessibili direttamente, perché facenti parte di un processo sequenziale. L'unico problema nella sicurezza adottata è che si è scelto di utilizzare la session storage che è modificabile dall'utente.

Dato che questa è una scelta non sicura si è studiato un metodo che si potrebbe implementare nel momento in cui il cliente lo richieda, per eliminare questo problema. Tale strategia si basa sull'utilizzo di token come si sta facendo attualmente ma comprende una modifica necessaria ai metodi Axios front-end e a

quelli presenti nei controller del back-end in modo da mandare e ricevere token di autorizzazione ad ogni operazione. Con quest'idea si potrebbe eliminare il problema della modifica della session storage dato che se non si manda insieme a ogni richiesta il token di autorizzazione ottenuto con il login il server rifiuterà la richiesta.

Per introdurre un livello maggiore di sicurezza si potrebbe creare un sistema di scadenza per i token, che dopo un certo ammontare di minuti risulteranno invalidi, ma questo potrebbe andare in contrasto con le richieste del cliente, che necessita di avere un utilizzo prolungato del dispositivo senza necessità di effettuare nuovamente il login ogni intervallo di tempo stabilito.

Non essendo richiesto un particolare tipo di privacy o delle restrizioni particolari si è optato per un classico tipo di autorizzazione a livello di username e password. In aggiunta, il codice del dispositivo fisico viene mandato nella richiesta di autenticazione per evitare che lo stesso utente possa accedere contemporaneamente a più di un dispositivo ed eliminando così i problemi di concorrenza che si introdurrebbero se quel caso si verificasse.

5.3.2 Metodologia agile

Durante l'intero svolgimento della tesi si è utilizzata una metodologia agile in modo da portare un valore incrementale al progetto e tenere sempre aggiornato il product owner oltre che permettere una comunicazione costante con il cliente, qualora necessario.

Lo sviluppo del progetto è stato svolto individualmente per questo motivo è stata applicata solamente in parte la metodologia agile in quanto tutto il lavoro è stato svolto da un'unica persona.

La decisione presa è stata quella di creare delle sessioni di aggiornamento costanti ogni volta che la tesi si fosse svolta in sede (nello specifico due volte a settimana) in modo da poter fare degli incontri di confronto riconducibili agli Stand-up meeting della metodologia SCRUM. Per quanto riguarda il product backlog, esso è stato implementato utilizzando una lista sulla piattaforma GitLab da cui è stato possibile attingere per creare i vari task. Una volta completati i task essi sono stati segnati come tali e nel caso in cui è stato richiesto di imparare qualcosa sono stati introdotti task aggiuntivi.

Il materiale è stato condiviso con il product owner attraverso GitLab ed è stato possibile avere dei rapidi aggiornamenti in quanto quotidianamente venivano caricate su questa piattaforma le versioni modificate del codice.

Nell'immagine seguente è possibile vedere la board di GitLab che è stata utilizzata per visualizzare ed interagire con il product backlog, attraverso le varie sezioni è stato possibile creare gli equivalenti degli sprint backlog in SCRUM. Le tre colonne indicano la prima il product backlog, la seconda lo sprint backlog e la terza rappresenta solamente l'elenco delle storie completate.

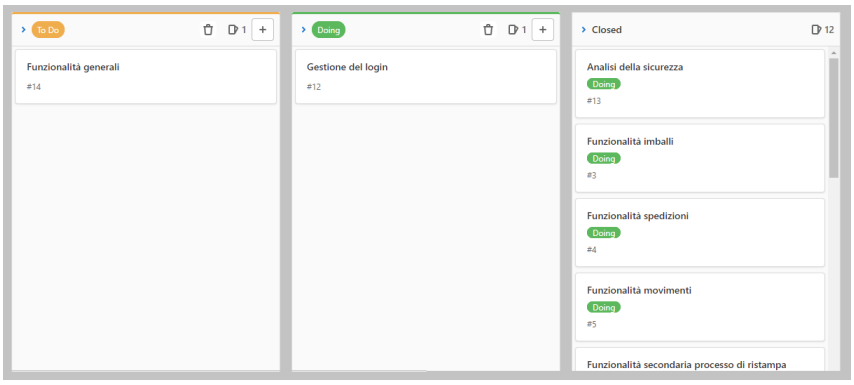


Figura 93: GitLab board

Per quanto riguarda il singolo elemento del product backlog è possibile avere un'idea di una suddivisione in task dall'immagine seguente. Mano a mano che le richieste venivano soddisfatte i relativi task venivano spuntati dalla lista.

Nell'immagine è possibile vedere come per una funzionalità primaria sia necessaria l'implementazione delle chiamate e della grafica lato front-end e quella della logica e delle API lato back-end (oltre che ad una serie di test e studi necessari per ogni funzionalità).

Funzionalità imballi

- [x] Analisi funzionalità attuali e migliorie (con domande al cliente se necessario)
- [x] Studio nuovi componenti da integrare
- [x] Back-end (C#)
- [x] Back-end (Chiamate ReactJS)
- [x] Funzionalità front-end (ReactJS)
- [x] Grafica front-end (ReactJS)
- [x] E2E tests

Figura 94: Task GitLab

Inoltre per la condivisione di alcuni file esterni all'implementazione del progetto si è utilizzato uno spazio condiviso su Google Drive. All'interno di esso è stato messo del materiale tra cui è possibile trovare l'analisi dell'applicazione vecchia con tutti i form ricostruiti a mano, i file di comunicazione per le decisioni prese, i documenti da discutere con il cliente e qualche video effettuato sia su delle funzionalità vecchie che sulla nuova implementazione di esse.

6 Proposta introduzione data mining

Come ultima funzionalità legata a questo progetto di tesi vi è la possibilità di fornire al cliente delle informazioni che potrebbero risultargli utili in ambito finanziario e logistico grazie all'impiego di tecniche di data mining.

In questo capitolo verranno presentati gli obiettivi che si intende raggiungere e verrà fornita un'implementazione di una parte di essi, mentre per altri sarà fornito uno studio teorico che potrà poi essere attuato in seguito grazie ai suggerimenti proposti.

Questa parte della tesi ha permesso di per poter mettere in pratica alcune conoscenze apprese nei corsi della laurea magistrale in ambito di manipolazione di dati e big data e quindi appartenenti alla specialità intrapresa in questo corso di studi, per questo motivo la trattazione di questo capitolo è molto importante.

Saranno inoltre presenti alcune nozioni di teoria che serviranno a giustificare alcune scelte intraprese oltre che a fornire una visione più chiara degli algoritmi utilizzati e degli obiettivi decisi.

6.1 Panoramica degli obiettivi e ambiente di sviluppo

In questa sezione verranno inizialmente presentati gli obiettivi che sono stati posti in modo da poter fornire dei dati che siano attendibili e utili al cliente nel caso decidesse di apportare le modifiche suggerite. I consigli forniti riguardano la scoperta di prodotti o categorie di prodotti che vengono solitamente venduti insieme e che quindi potrebbero essere utilizzati per l'applicazione di sconti o per migliorare la disposizione all'interno del magazzino, in modo da diminuire costi e sforzi necessari nei prelevamenti ed eventualmente nei depositi.

In secondo luogo verrà fornita una visione dell'ambiente di sviluppo che si è scelto, con particolare focus sui linguaggi che si potrebbero utilizzare e quelli che si sono decisi oltre che un'analisi analoga per gli ambienti di sviluppo e le tecnologie. Inoltre verrà presentato il software utilizzato per l'interazione con il DB, grazie al quale sarà possibile avere un rapido riscontro riguardante i risultati ottenuti dalle query.

Grazie ad una scelta corretta dell'ambiente ed a una definizione degli obiettivi chiara e coincisa sarà possibile stabilire quali algoritmi si avvicinino di più allo scopo che si vuole raggiungere. In alcuni casi sarà anche possibile implementarli nella versione base interagendo con il DB relazionale. Per quanto riguarda invece la comunicazione con i database non relazionali distribuiti non verrà fornita un'implementazione vera e propria ma solamente idee riguardanti possibili sviluppi futuri.

6.1.1 Definizione degli obiettivi e regole di associazione

L'obiettivo principale di questa analisi è quello di trovare delle regole di associazione che possano mostrare come certi prodotti, o certe categorie di prodotti, vengano comprate di conseguenza all'acquisto di altre, formando quindi una sorta di legame chiamato correlazione.

Una volta che si è trovata l'associazione tra uno o più prodotti (identificati come antecedenti) ed altri comprati di conseguenza (chiamati conseguenti) è possibile suggerire al cliente dei miglioramenti che possano portare guadagno e ridurre le perdite in materia di tempo e risorse.

Come prima idea vi è quella di proporre uno spostamento fisico di merci in modo da accostare i prodotti antecedenti e quelli conseguenti facenti parte delle regole di associazione più frequenti. Se si attua una scelta di questo tipo è possibile avere vicini i prodotti comprati solitamente insieme, questo ridurrebbe di molto lo spreco di energie e risorse impiegate nelle operazioni di prelievo e deposito, che attualmente vengono effettuate in maniera casuale.

Questa metodologia è molto utilizzata oggigiorno specialmente nell'organizzazione degli scaffali nei negozi ed è nota come correlazione "Beer and Diapers". Si tratta del risultato di uno studio per trovare le correlazioni negli acquisti, il quale ha mostrato che molto spesso l'acquisto di birre fosse accompagnato all'acquisto di pannolini, una correlazione sicuramente insolita ma veritiera.

Inoltre, avendo un'idea dei prodotti o delle categorie che spesso vengono acquistate insieme dalla stessa azienda o persona è possibile fornire al cliente la possibilità di applicare degli sconti o di creare dei pacchetti per favorire un acquisto maggiore.

Una regola di associazione nello specifico si trova quando si hanno delle transazioni, quindi un gruppo di prodotti che vengono acquistati insieme, come nel caso della lista della spesa. Spesso si definisce una regola di associazione nel modo seguente:

{Elemento A, Elemento B}=>{Elemento C}

Figura 95: Regola di associazione

L'insieme degli antecedenti è quello a sinistra della freccia, mentre quello a destra contiene i conseguenti, non per forza le dimensioni coincidono con quelle della figura, è possibile avere regole di associazione di qualunque natura per quanto riguarda la cardinalità degli insiemi.

Una regola di associazione presenta diverse metriche che permettono di valutarla, tra esse possiamo trovare:

- **Supporto**: indica la frazione delle transazioni che contengono sia gli antecedenti che i conseguenti e si rappresenta con un numero decimale compreso tra zero e uno
- **Confidenza**: rappresenta la porzione di transizioni contenenti gli antecedenti in cui sono presenti i conseguenti, anche in questo caso il valore viene assunto nello stesso range del supporto

Questi due parametri verranno utilizzati nell'implementazione degli algoritmi e nel filtraggio dei risultati in modo da ottenere un insieme di regole non troppo numeroso e che possano essere applicate.

6.1.2 Analisi dei dati

Per quanto riguarda l'analisi dei dati verranno prese in considerazione le vendite effettuate dall'azienda, ma non saranno necessarie informazioni specifiche riguardanti i clienti.

All'interno del database sono presenti molte tabelle, in seguito verranno elencate quelle trattate in quest'analisi e in cui saranno presenti i dati utilizzati durante la ricerca delle regole di associazione.

- **ODS_SALES**: questa tabella sarà quella maggiormente utilizzata, rappresenta lo storico delle vendite e al suo interno sono presenti dei campi riguardanti l'item e il compratore oltre che la quantità, il costo e la sede in cui è stato venduto il prodotto.
- **NAV_PRASCO2012$Item**: la tabella in questione contiene l'elenco dei prodotti con la relativa descrizione e codice di categoria di appartenenza
- **NAV_PRASCO2012$Item Category**: per visualizzare maggiori informazioni relative alla categoria di appartenenza di un prodotto viene utilizzata questa tabella che ne riporta anche la descrizione per esteso
- **HIST_AVAILABILITY**: le informazioni contenute nella tabella in questione servono per verificare la disponibilità attuale presente nei vari magazzini in modo da sapere quale prodotto è realmente disponibile

Le informazioni contenute nelle tabelle sopra citate verranno utilizzate anche per effettuare operazioni di filtraggio di dati. Nel caso in questione verranno prese solamente le sedi realmente esistenti, alcune di quelle indicate rappresentano degli scatoloni in cui vengono riposti gli oggetti difettosi o rotti.

La disponibilità invece sarà possibile utilizzala nel caso si decidesse di filtrare le regole di associazione ottenute in base alla presenza o meno del prodotto nel magazzino, in modo da prendere in considerazione solamente spostamenti e sconti su prodotti attualmente disponibili.

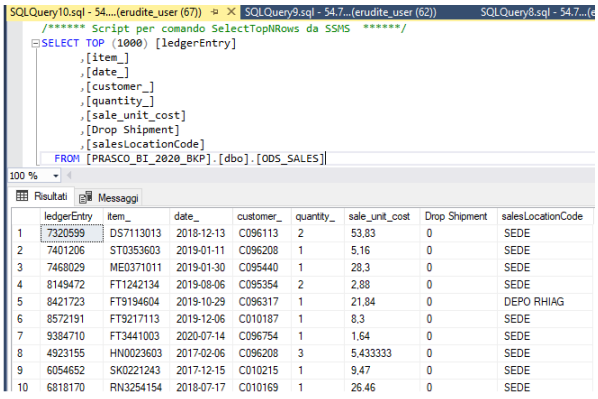
La scelta effettuata è stata quella di implementare gli algoritmi solamente dopo aver raggruppato gli acquisti per categoria, dato che l'elaborazione non sarebbe stata possibile se si fossero considerati i prodotti, almeno non si sarebbe potuto ricavare nulla se non con l'impiego di tecnologie big data data la limitazione fisica dei PC a livello di allocazione di memoria per effettuare i calcoli.

6.1.3 Linguaggi e IDE scelti

Per l'implementazione di algoritmi di data mining è possibile utilizzare differenti linguaggi in svariati ambienti di sviluppo. I più diffusi sono il linguaggio R (che viene principalmente utilizzato in RStudio) e Python, che viene utilizzato in diversi ambienti. Alcuni approcci al data mining possono anche essere scritti in Java utilizzando come IDE Eclipse per esempio, ma tipicamente questa strada viene scelta nell'ambito dei big data.

Tra le possibili scelte si è deciso di utilizzare lo stesso IDE utilizzato per il back-end, ovvero Visual Studio 2019 che supporta pienamente lo sviluppo di codice Python, linguaggio utilizzato nell'implementazione degli algoritmi.

Per quanto riguarda l'accesso ai dati e le interrogazione al DB verranno utilizzate le classiche query in linguaggio SQL relative ai database relazionali e tra i tanti programmi per la gestione dei DB verrà utilizzato quello facente parte della famiglia Microsoft, ovvero SQL Server Management Studio, mostrato in figura.



	ledgerEntry	item_	date_	customer_	quantity_	sale_unit_cost	Drop Shipment	salesLocationCode
1	7320599	DS7113013	2018-12-13	C096113	2	53,83	0	SEDE
2	7401206	ST0353603	2019-01-11	C096208	1	5,16	0	SEDE
3	7468029	ME0371011	2019-01-30	C095440	1	28,3	0	SEDE
4	8149472	FT1242134	2019-08-06	C095354	2	2,88	0	SEDE
5	8421723	FT9194604	2019-10-29	C096317	1	21,84	0	DEPO RHIAG
6	8572191	FT9217113	2019-12-06	C010187	1	8,3	0	SEDE
7	9384710	FT3441003	2020-07-14	C096754	1	1,64	0	SEDE
8	4923155	HN0023603	2017-02-06	C096208	3	5,433333	0	SEDE
9	6054652	SK0221243	2017-12-15	C010215	1	9,47	0	SEDE
10	6818170	RN3254154	2018-07-17	C010169	1	26,46	0	SEDE

Figura 96: SSMS

L'immagine precedente mostra un esempio di query di selezione totale effettuata sulla tabella delle vendite in cui è possibile notare la presenza dei campi precedentemente accennati.

Nella figura successiva è possibile vedere una cattura dell'ambiente di sviluppo VS2019, è stato creato un progetto Python apposito per l'implementazione degli algoritmi di data mining e per l'elaborazione dei dati.

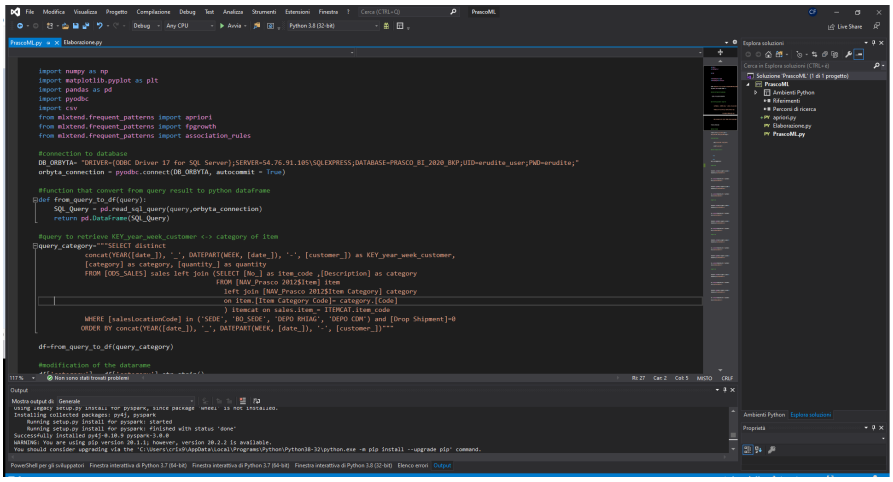


Figura 97: Progetto Python in Visual Studio 2019

Anche se non verrà utilizzato in questo studio è possibile vedere un esempio dell'ambiente di sviluppo RStudio che utilizza il linguaggio R nella figura sottostante, volendo può essere utilizzato come alternativa a Python.

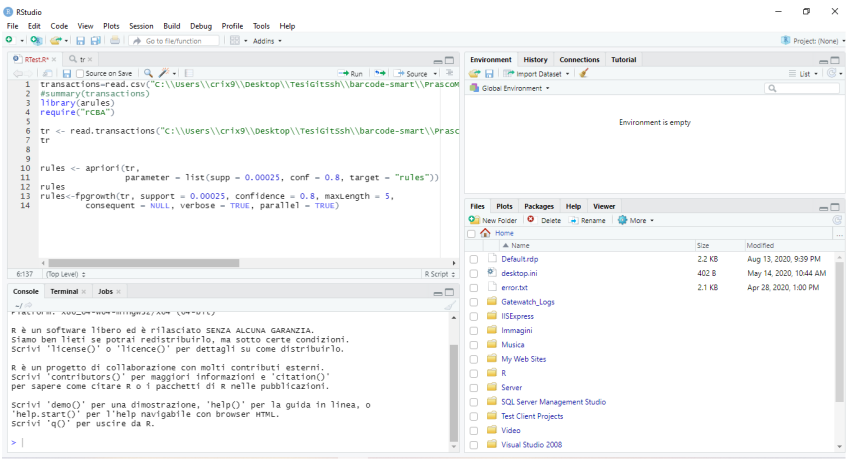


Figura 98: RStudio

Per quanto riguarda l'introduzione delle tecniche big data non è stata prevista nessuna implementazione. Tuttavia nella sezione apposita verrà introdotto Spark come framework possibile da utilizzare, insieme a Hadoop, che è necessario per il suo funzionamento in questo caso specifico (oltre ad essere una possibile alternativa). Questi framework spesso si utilizzano con il linguaggio Java ed hanno una semplice implementazione in Eclipse come ambiente di sviluppo.

6.2 Algoritmi implementati

In questa sezione verranno presentati due dei principali algoritmi utilizzati per trovare le regole di associazione nel campo del data mining, Apriori e FPGrowth. L'implementazione di essi è stata eseguita in Visual Studio 2019 utilizzando come linguaggio Python. Sono state trovate delle regole di associazione utili da mostrare al cliente e verranno illustrate per entrambi i casi.

Come anticipato precedentemente è stato fatto uno studio raggruppando per categoria dato che non è possibile effettuare una computazione completa a livello di prodotto (come granularità) la restrizione è dovuta ai limiti fisici di tipo hardware.

La prima parte del codice è in comune per entrambi gli algoritmi perché si tratta della connessione al database, seguita della richiesta e dall'elaborazione dei dati. L'obiettivo è quello di ottenere un insieme di vendite sotto forma di transazione in un particolare formato accettato dai due algoritmi.

La prima cosa da fare oltre all'importazione dei pacchetti corretti e alla loro installazione iniziale (procedimento effettuato alla creazione del progetto ma non descritto) è la creazione di una connessione al database SQL, mostrata nella seguente figura.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pyodbc
import csv
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

#connection to database
DB_ORBYTA= """DRIVER={ODBC Driver 17 for SQL Server};
              SERVER=54.76.91.105\SQLEXPRESS;
              DATABASE=PRASCO_BI_2020_BKP;
              UID=[redacted];
              PWD=[redacted];"""
orbyta_connection = pyodbc.connect(DB_ORBYTA, autocommit = True)
```

Figura 99: Connessione al db con Python

La richiesta di connessione necessita di alcuni parametri tra cui le credenziali, oscurate per privacy e il nome del database, insieme all'indirizzo IP del server ospitante. Il primo parametro permette di specificare il driver che si può utilizzare per connettersi al database [53], ne sono presenti parecchi e nel caso indicato viene usato quello ODBC versione 17. Osservando l'ultima riga di codice è anche possibile vedere come venga settata a true l'opzione di auto commit, in questo modo non è necessario eseguirla ogni volta manualmente nel momento in cui vengono lanciate le istruzioni scritte in linguaggio SQL.

Per poter utilizzare al meglio gli strumenti offerti dal linguaggio Python occorre convertire i risultati della query in data frame, per questo motivo viene definita una funzione che esegue la richiesta ricevuta in input per poi trasformare il risultato in un data frame della libreria pandas (anche chiamato Python data frame). La query in questione è la seconda parte in comune tra i due algoritmi e ha il compito di richiedere le vendite presenti nella specifica tabella dedicata, apportando alcuni filtri che verranno mostrati nell'immagine seguente.

```
#function that convert from query result to python dataframe
def from_query_to_df(query):
    SQL_Query = pd.read_sql_query(query,orbyta_connection)
    return pd.DataFrame(SQL_Query)

#query to retrieve KEY_year_week_customer <-> category of item
query_category="""SELECT distinct
            concat(YEAR([date_]), '_', DATEPART(WEEK, [date_]), '-', [customer_]) as KEY_year_week_customer,
            [category] as category, [quantity_] as quantity
            FROM [ODS_SALES] sales left join (SELECT [No_] as item_code ,[Description] as category
                                              FROM [NAV_Prasco 2012$Item] item
                                                left join [NAV_Prasco 2012$Item Category] category
                                                on item.[Item Category Code]= category.[Code]
                                                ) itemcat on sales.item_= ITEMCAT.item_code
            WHERE [salesLocationCode] in ('SEDE', 'BO_SEDE', 'DEPO RHIAG', 'DEPO CDM') and [Drop Shipment]=0
           ORDER BY concat(YEAR([date_]), '_', DATEPART(WEEK, [date_]), '-', [customer_])"""

df=from_query_to_df(query_category)
```

Figura 100: Query principale con Python

Il compito di questa richiesta è quello di mostrare tutte le vendite in un formato tale per cui siano raggruppate in modo che nella prima colonna sia presente una chiave che verrà identificata come "KEY_year_week_customer" (una concatenazione dell'anno e del mese di quell'anno in cui la vendita è stata effettuata con il relativo codice del cliente che ha effettuato l'acquisto). Per il tipo di studio che verrà mostrato non è importante la quantità di per sé o chi ha comprato ma l'importante è che vi è una transazione da parte di qualcuno contenete certi prodotti.

Per poter fare quest'analisi però è necessario avere come seconda e terza colonna rispettivamente la categoria e la quantità. La seconda operazione di SELECT interna serve solamente a reperire il nome della categoria, in modo da averlo più leggibile.

Particolare attenzione va rivolta alla clausola di WHERE, dove è possibile vedere un filtro che ha lo scopo di selezionare solamente le sedi realmente esistenti

(non per esempio le etichette date a delle scatole di rottami ma inserite come "sedi") e le vendite che non sono state cancellate (informazione reperibile tramite l'apposito flag).

Una volta selezionate queste informazioni occorre modificare il data frame ottenuto e manipolarlo in modo che presenti un formato accettabile dagli algoritmi applicati. Per questa ragione verranno eseguite le operazioni riportate in figura.

```
#modification of the datarame
df['category'] = df['category'].str.strip()
df.dropna(axis=0, subset=['KEY_year_week_customer'], inplace=True)
df['KEY_year_week_customer'] = df['KEY_year_week_customer'].astype('str')

#creation of basket transations
basket = (df
          .groupby(['KEY_year_week_customer', 'category'])['quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('KEY_year_week_customer'))

#modification of the quantity, from X to 0 or 1
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
basket_sets = basket.applymap(encode_units)
```

Figura 101: Manipolazione data frame con Python

La prima operazione svolta è analoga alla terza e consiste in un'eliminazione degli spazi, mentre la seconda rimuove tutte le tuple che presentano come valore della chiave in prima colonna null.

Una volta pulito il risultato da valori mancanti è possibile modificarne il formato. La richiesta da parte degli algoritmi è un input che presenti sulle varie colonne i possibili valori degli oggetti nelle transazioni (nel caso specifico quindi le categorie) mentre nelle righe un 1 o uno 0 in base a se in quella transazione quella categoria sia presente o meno. Per ottenere un input di questo tipo vengono utilizzati i metodi forniti dai data frame quali groupby(), sum(), reset_index(), fillna() e set_index(). Dopo la creazione delle transazioni però è necessario settare a 1 tutti i campi che presentano una quantità positiva (dovuta alla somma delle quantità), per questa ragione viene utilizzata la funzione definita nella parte inferiore dell'immagine.

Terminata questa operazione avremo un input da fornire agli algoritmi del tipo mostrato nella seguente figura, l'intestazione delle colonne è stata omessa perché troppo larga e confusionale data la natura descrittiva della variabile, ma ogni cifra in ogni colonna di ogni riga corrisponde a una determinata catego-

ria di prodotti, lo zero sta a indicare un'assenza di quella categoria in quella transazione, un uno sta indicare una presenza.

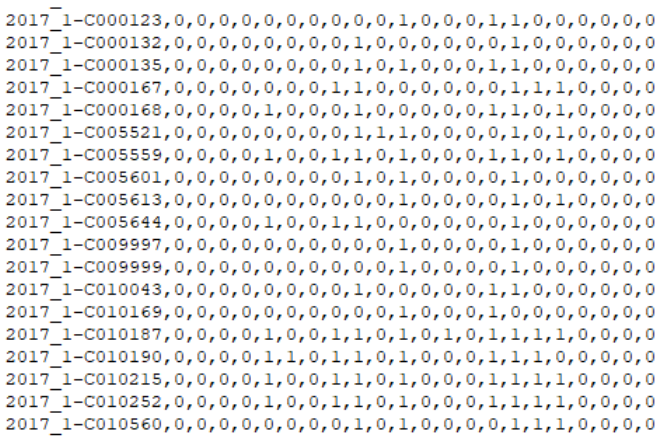


Figura 102: Formato dataset transazioni

Con questa elaborazione termina la parte in comune per entrambi gli algoritmi, in seguito verranno mostrate le parti di codice differenti per ognuno ed i risultati ottenuti, oltre ad un breve cenno al funzionamento di essi per una migliore comprensione.

Nella tabella seguente sono riportate alcune informazioni relative ai due algoritmi che verranno descritti nel dettaglio in seguito, per poter avere un'idea delle loro principali caratteristiche.

Apriori	FP-Growth
Più lento	Più veloce
Basato su alberi	Basato su vettori
Richiede parecchie scansioni delle transazioni	Richiede due scansioni delle transazioni
Utilizza un approcio in profondità	Utilizza un approccio in ampiezza

Figura 103: Riassunto caratteristiche algoritmi

6.2.1 Apriori

L'algoritmo di Apriori è quello più famoso per trovare le regole di associazione relative a un set di transazioni. Il suo funzionamento ruota intorno ad un principio legato agli itemset frequenti.

Un insieme di oggetti può definirsi frequente qualora sia presente un numero di volte uguale o superiore a un supporto fissato, denominato spesso "min_sup". La ricerca degli itemset frequenti è molto importante perché una volta trovati è possibile ricavare le regole di associazione applicando delle semplici combinazioni agli insiemi ottenuti, cercando di creare tutti i gruppi di antecedenti e conseguenti possibili.

Il principio accennato precedentemente stabilisce che se un itemset non è frequente allora tutti i suoi sovra-insiemi non lo sono, se per esempio l'insieme A,B non è frequente sicuramente tutti gli insiemi contenenti loro due più altro non sono frequenti (chiamato anche principio di antimonotonicità).

Per trovare gli itemset frequenti l'algoritmo di Apriori genera prima i candidati e poi effettua un'operazione di pruning, partendo da insiemi di dimensione 1 (che soddisfano la soglia minima di supporto) vengono generati grazie ad un prodotto cartesiano quelli di dimensione due e poi vengono filtrati in base alla soglia di supporto.

Una volta ottenuti gli insiemi di due elementi si procede alla creazione di insiemi sempre più grossi apportando i relativi tagli qualora i requisiti di supporto non siano soddisfatti.

La generazione dei candidati di insiemi con cardinalità maggiore di due avviene accostando la parte in comune con le due non in comune, per esempio dati gli insiemi {A,B,C} e {A,B,D} otterremo {A,B,C,D}.

Una volta trovati gli itemset frequenti è possibile rapidamente trovare le regole di associazione apportando come detto precedentemente delle semplici combinazioni, per esempio dall'insieme {A,B,C} possiamo ottenere:

- {A,B} => {C}
- {A,C} => {B}
- {B,C} => {A}
- {A} => {B,C}
- {B} => {A,C}
- {C} => {A,B}

Tutte le regole hanno stesso supporto perché provenienti dallo stesso itemset tuttavia la confidenza è diversa in ogni caso, per questa ragione occorre verificare quale regola soddisfa la confidenza desiderata applicando i relativi filtri.

A seconda della libreria utilizzata è possibile che sia richiesta o meno questa misura in input, nel caso illustrato le regole vengono generate tutte, con le relative statistiche, è quindi opportuno eseguire un filtro sulla correlazione nel caso si voglia restringere il campo.

Nell'immagine seguente è possibile vedere la chiamata alle due funzioni che generano prima gli itemset frequenti utilizzando Apriori e poi le relative regole di associazione con una funzione apposita.

```python
#apriori implementation, frequet itemsets generation
frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)
frequent_itemsets.to_csv("AprioriFrequentItemsets0.05.csv")

#apriori implementation, association rules generation
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.to_csv("AprioriAssociationsRules0.05.csv")
```

Figura 104: Applicazione dell'algoritmo Apriori in Python

Come si nota dalla figura viene prima chiamato l'algoritmo e poi vengono create le regole, è possibile vedere la presenza di due istruzioni per salvare i risultati nei relativi file in formato csv e la definizione della soglia di supporto minimo settata a 0.05, equivalente al 5%.

Sono stati effettuati diversi test, utilizzando 0.05, 0.1, 0.3 e 0.5 come supporto minimo, per vedere la variazione delle relative regole di associazione al cambiare di questo parametro.

```
,antecedents,consequents,confidence
3,'PARAFANGHI','GRIGLIA',0.7152593277116784
4,'GRIGLIA','PARAURTI-SPOILER-COMPONENTI',0.8903782211138821
7,'ILLUMINAZIONE','PARAURTI-SPOILER-COMPONENTI',0.7896299228181278
8,'PARAFANGHI','PARAURTI-SPOILER-COMPONENTI',0.8876054060298024
11,'RETROVISORI','PARAURTI-SPOILER-COMPONENTI',0.7978708502591398
13,"'GRIGLIA', 'ILLUMINAZIONE'",'PARAURTI-SPOILER-COMPONENTI',0.9376926821353238
14,"'PARAURTI-SPOILER-COMPONENTI', 'ILLUMINAZIONE'",'GRIGLIA',0.7241854636591479
18,"'GRIGLIA', 'PARAFANGHI'",'PARAURTI-SPOILER-COMPONENTI',0.9623708010335916
20,"'PARAFANGHI', 'PARAURTI-SPOILER-COMPONENTI'",'GRIGLIA',0.7755075481520042
```

Figura 105: Regole di associazione trovate con Apriori

Nell'esempio mostrato sono state filtrate le regole con supporto maggiore a 0.5 e si può notare le relativa confidenza. Grazie all'applicazione degli algoritmi si possono osservare anche altri dati relativi alle regole di associazione ma non verranno mostrati in quanto non rilevanti per questo tipo di analisi.

6.2.2 FP-Growth

Un altro algoritmo utilizzato spesso è quello chiamato Frequent Pattern Growth, o semplicemente FP-Growth. Questo algoritmo è applicabile su database densi (non sparsi) e ha un approccio molto simile a quello di Apriori per quanto riguarda l'utilizzo degli itemset frequenti.

L'algoritmo è leggermente più complesso di quello precedentemente descritto perché richiede più passaggi, ma una volta trovati gli insiemi di prodotti frequenti la generazione delle regole di associazione avviene in modo analogo.

Innanzitutto è necessario creare una tabella in cui vengano elencati tutti i prodotti nell'insieme delle transazioni, con i relativi supporti, accuratamente filtrati in base alla soglia decisa come supporto minimo. Una volta elencati è necessario ordinarli in ordine decrescente di supporto.

A questo punto avviene la parte più importante dell'algoritmo, ovvero la generazione di alberi chiamati FP-Tree. La creazione dell'albero deve seguire i seguenti passi per ogni transazione nel database:

- Ordinazione dei prodotti presenti in modo concorde alla tabella creata all'inizio
- Inserimento dei prodotti nell'albero partendo dalla radice (se non è presente la foglia con il prodotto indicato inserirla, altrimenti incrementare di uno il contatore a ogni passaggio)
- Generazione degli itemset frequenti dalla tabella nel caso in cui presentino un supporto minimo accettabile

Nell'immagine successiva è possibile vedere un esempio di sviluppo di un albero nel caso in cui venga elaborata una nuova transazione.

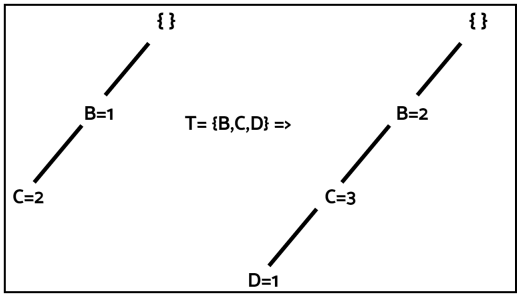


Figura 106: FP-Tree

Successivamente si procede con la costruzione dei CPB (Conditonal Pattern Base), partendo dalle foglie, per ogni elemento che possiede come somma complessiva di tutti i valori un numero che soddisfa il min_sup . Ovvero si estrae

l'elenco di tutti i percorsi che arrivano all'elemento per cui si sta costruendo il CPB con relativo supporto (uguale a quello indicato nella foglia). Per avere un esempio della costruzione del conditional pattern base relativo ad un prodotto è possibile vedere la seguente immagine.

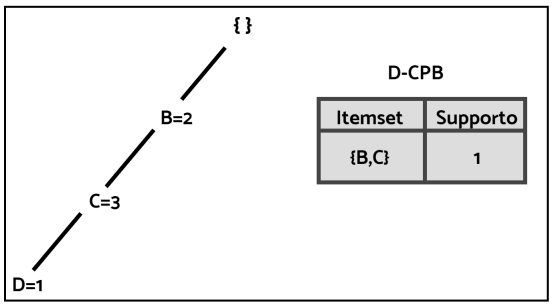


Figura 107: FP-CPB

In seguito avviene la ricreazione della tabella iniziale utilizzando le nuove transazioni "parziali" e si ricostruzione dell'albero FP-Tree sull'insieme di transazioni facenti parte del X-CPB. Successivamente si ricrea il CPB che sarà nella forma XY-CPB e si ripete tutto il procedimento ricorsivamente fino ad esaurimento valori.

Anche in questo caso l'implementazione dell'algoritmo è molto semplice in Python, l'immagine successiva mostra le istruzioni necessarie per la scoperta di itemset frequenti con un supporto minimo di 0.5 e la creazione delle relative regole di associazione.

```
### Support: 0.5 ###

#fpgrowth implementation, frequet itemsets generation
frequent_itemsets = fpgrowth(basket_sets, min_support=0.5, use_colnames=True)
frequent_itemsets.to_csv("FpGrowthFrequentItemsets0.5.csv")

#fpgrowth implementation, association rules generation
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.to_csv("FpGrowthAssociationRules0.5.csv")
```

Figura 108: Applicazione dell'algoritmo FP-Growth in Python

Anche in questo caso le prove sono state effettuate con supporti differenti per cercare di mostrare risultati diversi. Nella figura sottostante sono rappresentati gli itemset frequenti trovati utilizzando come soglia di supporto 0.3 (equivalente al 30% delle transazioni), con le relative informazioni.

```
,support,itemsets
0,0.7441027012570206,frozenset({'PARAURTI-SPOILER-COMPONENTI'})
1,0.5147900508157262,frozenset({'GRIGLIA'})
2,0.46306499063920836,frozenset({'PARAFANGHI'})
3,0.3047873763038246,frozenset({'COFANI'})
4,0.5405723455469377,frozenset({'ILLUMINAZIONE'})
5,0.38186680930730144,frozenset({'RETROVISORI'})
6,0.3174645627173041,frozenset({'PASSARUOTA'})
7,0.45835784969243115,"frozenset({'GRIGLIA', 'PARAURTI-SPOILER-COMPONENTI'})"
8,0.3296603369884996,"frozenset({'GRIGLIA', 'ILLUMINAZIONE'})"
9,0.30912008558438087,"frozenset({'GRIGLIA', 'PARAURTI-SPOILER-COMPONENTI', 'ILLUMINAZIONE'})"
10,0.4110189890345012,"frozenset({'PARAFANGHI', 'PARAURTI-SPOILER-COMPONENTI'})"
11,0.3312115538914148,"frozenset({'GRIGLIA', 'PARAFANGHI'})"
12,0.3187483284300615,"frozenset({'GRIGLIA', 'PARAFANGHI', 'PARAURTI-SPOILER-COMPONENTI'})"
13,0.4268520994918427,"frozenset({'PARAURTI-SPOILER-COMPONENTI', 'ILLUMINAZIONE'})"
14,0.3046803958277614,"frozenset({'PARAURTI-SPOILER-COMPONENTI', 'RETROVISORI'})"
```

Figura 109: Itemset Frequenti trovati con FP-Growth

6.2.3 Studio della stabilità dei prodotti

Gli studi effettuati fino ad ora hanno preso in considerazione la categoria di appartenenza del prodotto nelle varie transazioni. Questo approccio può mostrarsi utile, ma sarebbe sicuramente più accurato un risultato a livello del singolo prodotto. Attualmente la lista dei prodotti differenti è molto lunga, ma è possibile fare alcune considerazioni ed implementarle nella pratica in modo da ridurre la cardinalità del dataset e provare ad eseguire le operazioni senza l'utilizzo di tecniche di big data.

Innanzitutto è possibile considerare solamente una porzione di vendite recenti, ad esempio quelle effettuate negli ultimi sei mesi. Per non entrare nel dettaglio non verranno mostrate tutte le query, ma se si cerca di visualizzare la quantità di prodotti venduti negli ultimi sei mesi per ogni prodotto si ottiene una classifica che però non tiene conto del fatto che un prodotto a volte viene acquistato una sola volta o massimo due dallo stesso cliente in grossissime quantità, si tratta quindi di un outlier.

Altra cosa da prendere in considerazione è il volume degli oggetti, purtroppo questa informazione non è disponibile nel database, ma per quelli più frequenti è possibile dalla descrizione farsi un idea delle probabili dimensioni, sicuramente non sarà un problema spostare delle viti rispetto a delle portiere di un automobile.

Per questa ragione viene eseguita la seguente query che ha il compito di prendere gli item stabili, ovvero che vengano comprati almeno una volta al mese. Per rendere leggibile il codice sono stati messi solamente i primi due insiemi che verranno intersecati, occorre aggiungere anche gli altri 4 contenenti gli oggetti venduti negli altri mesi. Grazie a questo approccio è possibile ridurre il numero di articoli differenti da 38.480 a 5911, una notevole riduzione. Una volta trovati gli item stabili è possibile salvare una vista contenente queste informazioni, in modo da non sovraccaricare il database. Per semplicità la vista chiamata verrà identificata come "SOLD_STABLE_ITEMS_LST_6MTHS".

```
#query to find stable items
query_stable_items="""  SELECT distinct [item_]
                        FROM [PRASCO_BI_2020_BKP].[dbo].[ODS_SALES]
                        WHERE [Drop Shipment] = 0 and
                              [salesLocationCode] in ('SEDE', 'BO_SEDE', 'DEPO RHIAG', 'DEPO CDM') and
                              month([date_])=month(DATEADD(month, -6, GETDATE())) and
                              year([date_])=year(DATEADD(month, -6, GETDATE()))

                        INTERSECT

                        SELECT distinct [item_]
                        FROM [PRASCO_BI_2020_BKP].[dbo].[ODS_SALES]
                        WHERE [Drop Shipment] = 0 and
                              [salesLocationCode] in ('SEDE', 'BO_SEDE', 'DEPO RHIAG', 'DEPO CDM') and
                              month([date_])=month(DATEADD(month, -5, GETDATE())) and
                              year([date_])=year(DATEADD(month, -5, GETDATE()))

                        INTERSECT
```

Figura 110: Query per trovare i prodotti stabili

A questo punto occorre considerare il numero di volte che un oggetto è stato venduto, in modo da dare maggior peso ai prodotti stabili con maggiori vendite. Per poter capire il numero corretto di vendite da poter considerare è stata fatta una suddivisione in categorie dei pezzi in base al numero di vendite. Successivamente è stata effettuata una somma per vedere quanti prodotti entrassero in una determinata categoria.

Le categorie sono state create associando il numero di decine presenti nella somma dei prodotti come chiave, i pezzi che sono stati venduti 542 volte negli ultimi sei mesi rientrano nella categoria 6, mentre quelli venduti 2 volte nella categoria 1. La query che esegue questo conteggio è mostrata nell'immagine seguente.

```
#query to retrieve the number of sales divided by category
quey_sales_divided_for_category = """
                    SELECT sold_quantity_category, COUNT(distinct item) as histogram_height
                    FROM
                    (
                        SELECT item, sold_quantity_category, sold_quantity
                        FROM
                        (
                              SELECT [item_] as item,
                                     convert(int, sum([quantity_])/10) as sold_quantity_category,
                                     sum([quantity_]) as sold_quantity
                              FROM [PRASCO_BI_2020_BKP].dbo.ODS_SALES
                              WHERE [Drop Shipment] = 0 and
                                    [salesLocationCode] in ('SEDE', 'BO_SEDE', 'DEPO RHIAG', 'DEPO CDM') and
                                    [date_]>=DATEADD(month, -6, GETDATE())
                              GROUP BY [item_]
                        ) sales RIGHT JOIN SOLD_STABLE_ITEMS_LST_6MTHS as stable ON stable.item_=sales.item
                    ) tabella
                    GROUP BY sold_quantity_category
                    ORDER BY sold_quantity_category"""

df_sales_divided_for_category =from_query_to_df(query_sales_divided_for_category )
```

Figura 111: Query per trovare i prodotti stabili caratterizzati

Dal risultato ottenuto possiamo vedere che la maggior parte dei prodotti è presente nelle categorie inferiori, quindi associate ad un numero di vendite minore. Se si decide di considerare per esempio i prodotti con categoria maggiore di sette (venduti molto spesso negli ultimi sei mesi, più di 600 volte ciascuno) si ottiene un insieme di solamente 2028 prodotti, dato gestibile e utilizzabile con gli algoritmi precedentemente descritti.

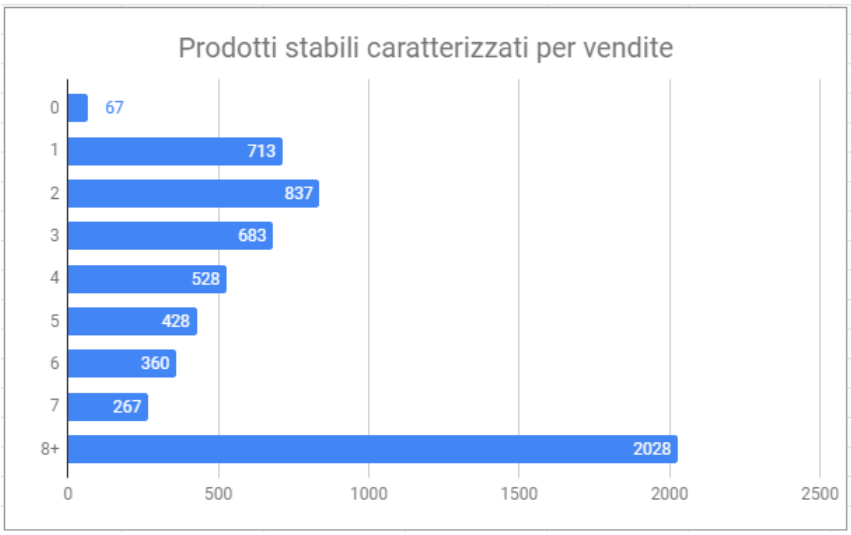


Figura 112: Prodotti stabili

Con questo tipo di studio è possibile anche prender in considerazione i prodotti, selezionando solo quelli stabili più venduti, in modo da ridurre la cardinalità dell'insieme di prodotti diversi.

Questa strategia permette di gestire in memoria l'elaborazione, cosa non possibile considerando tutti e 70 mila i prodotti iniziali. L'implementazione degli algoritmi di Apriori e FP-Growth è consigliata con questo dataset ma non verrà mostrata in quest'analisi.

6.3 Proposte per future implementazioni

L'approccio adottato fino ad ora in questo settore è stato quello di trovare le regole di associazione per le categorie di prodotti vendute insieme, in modo da portare i benefici precedentemente descritti all'azienda.

Tuttavia come visto nell'ultima sezione ci sono diversi metodi che possono essere applicati per effettuare delle analisi sui dati ed estrarne contenuti utili. L'idea pensata per una futura implementazione, anche per esplorare un settore differente è quella di utilizzare dei database non SQL e dei framework come Spark e Hadoop che consentono la manipolazione di dati in ambito big data.

L'implementazione proposta è sempre in Python ma necessita di avere dei cluster Big Data di SQL Server per poter effettuare le operazioni sui dati. La proposta è quella di utilizzare queste tecnologie per poter attuare gli algoritmi precedentemente descritti anche a livello di prodotto senza particolari filtri, in modo da vedere i risultati ottenuti e trarre le conclusioni necessarie.

Per poter fare questo tipo di operazioni però, come detto precedentemente, è necessaria una potenza di calcolo molto elevata, per questo motivo si è deciso di proporre questo tipo di approccio.

Se si decide di adottare una strategia simile è possibile utilizzare la tabella che contiene le disponibilità attuali dei prodotti per poter intersecare i risultati, in modo da fornire regole di associazione solamente su prodotti realmente presenti nelle varie sedi. Questa operazione era priva di senso con le categorie dato che sicuramente erano presenti dei prodotti di ogni categoria in ogni sede. Una cosa da considerare riguardante questo tipo di studio è la presenza nel database di alcuni ordini che passano direttamente dal fornitore al cliente finale senza transitare nei magazzini dell'azienda. In questo caso lo studio della disponibilità deve considerare anche questi prodotti per quanto riguarda la proposta di eventuali sconti, mentre per la gestione del carico nel magazzino è necessario applicare un'operazione per filtrare i prodotti in questione.

Un ulteriore futuro studio che si vuole proporre è quello di cercare di stimare o misurare il volume degli oggetti, in modo da poter considerare anche in maniera più precisa questo dato nella proposta di variazione di disposizione all'interno del magazzino.

Se fosse necessario è anche possibile concentrarsi maggiormente sul fatturato per poter estrarre le regole di associazione che più portano valore all'azienda, in modo da essere in grado di fornire delle offerte parecchio accurate, nel caso magari è anche possibile introdurre del machine learning per qualche tipo di predizione relativa alle vendite in modo da elaborare maggiormente gli sconti.

Per questo motivo in questa sezione verrà fatta prima una panoramica sul funzionamento dei database non relazionali e un'introduzione al campo dei big data e successivamente verranno proposte le due implementazioni dell'algoritmo Apriori e FP-Growth utilizzando tecniche di big data.

6.3.1 Elaborazione nel campo dei big data

Prima di procedere alla proposta di implementazione degli algoritmi nel campo dell'elaborazione big data è necessario avere una vaga idea di come funzioni questo tipo di approccio.

Innanzitutto i dati vengono organizzati secondo dei costrutti non SQL, viene smontato il concetto di tabella e avviene una partizione delle informazioni. Il principio su cui si basano le elaborazioni big data è quello di suddividere il carico di lavoro su diverse macchine, ognuna che lavora su un insieme di dati diverso, per questo motivo vengono utilizzati i cluster.

I dati spesso sono anche replicati, tipicamente tre volte oltre a quelli originali , una volta che i calcoli parziali vengono eseguiti è necessario ricompattare tutte le varie elaborazioni per ottenere un risultato finale.

I due framework principali sono Hadoop e Spark, il primo salva i dati all'interno del file system denominato HDFS (Hadoop Distributed File System) ed effettua delle operazioni di map-reduce, mentre il secondo utilizza gli RDD (Resilient Distributed Datasets), che rappresentano una collezione distribuita di dataset per trasformare i dati, come è possibile vedere in figura.

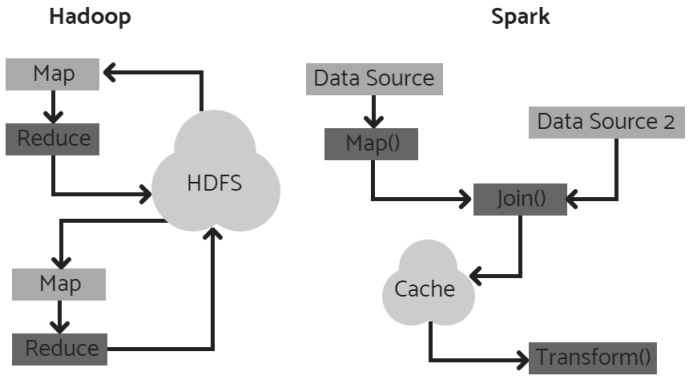


Figura 113: Hadoop vs Spark

La grande differenza tra i due è che Spark lavora in memoria, gli RDD sono elaborati in RAM, a differenza di Hadoop che legge e scrive file sul HDFS. Negli ultimi anni è stato favorito Spark, quindi molte delle elaborazioni attuali utilizzano questo framework.

In Python è possibile applicare operazioni di big data utilizzando Spark grazie alla libreria denominata PySpark. Per poter utilizzare questo approccio tuttavia come anticipato precedentemente è necessario avere a disposizione un cluster per l'elaborazione e il calcolo dei risultati.

Infine è utile sapere che nel caso si desiderasse è possibile utilizzare degli strumenti per la parallelizzazione delle computazioni, in modo da alleggerire il carico su una macchina. Nell'azienda presso cui si è sviluppata la tesi è stato utilizzato questo approccio in passato con l'ausilio del Cloud Computing, basandosi per esempio sull'utilizzo di Amazon Web Service (AWS) oppure con l'impiego di Docker per la creazione di macchine virtuali in modo da semplificare la gestione di sistemi distribuiti. Sulle macchine virtuali create è possibile attuare diversi tipi di personalizzazioni, in modo da effettuare il deployment di nuovi nodi solo quando necessario, permettendo uno stile di sviluppo del tipo platform as a service (Paas) per sistemi come Apache Cassandra, MongoDb o Rank.

6.3.2 Possibile implementazione di Apriori in Spark

Ci sono diversi modi per implementare l'algoritmo Apriori in Spark, tra essi il migliore trovato e analizzato è quello che utilizza la libreria sopra citata, PySpark, che mette a disposizione una serie di strumenti per poter scrivere l'algoritmo manualmente ma in maniera efficiente.

Questo approccio offre la possibilità di implementare l'algoritmo utilizzando Anaconda3 come distribuzione, abbinato a Spark-2.4.0 e Pycharm come IDE di sviluppo Python (ma è anche possibile utilizzare Vs2019 in alternativa, come nella prima parte dell'analisi).

```
def apriori(sc, f_input, f_output, min_sup):
    # read the raw data
    data = sc.textFile(f_input)
    # count the total number of samples
    n_samples = data.count()
    # min_sup to frequency
    sup = n_samples * min_sup
    # split sort
    itemset = data.map(lambda line: sorted([int(item) for item in line.strip().split(' ')]))
    # share the whole itemset with all workers
    shared_itemset = sc.broadcast(itemset.map(lambda x: set(x)).collect())
    # store for all freq_k
    frequent_itemset = []

    # prepare candidate_1
    k = 1
    c_k = itemset.flatMap(lambda x: set(x)).distinct().collect()
    c_k = [{x} for x in c_k]

    # when candidate_k is not empty
    while len(c_k) > 0:
        # generate freq_k
        Dprint("C{}: {}".format(k, c_k))
        f_k = generate_f_k(sc, c_k, shared_itemset, sup)
        Dprint("F{}: {}".format(k, f_k))

        frequent_itemset.append(f_k)
        # generate candidate_k+1
        k += 1
        c_k = generate_next_c([set(item) for item in map(lambda x: x[0], f_k)], k)
```

Figura 114: Linea guida per Apriori in Spark

L'immagine superiore rappresenta un estratto di una parte di codice condivisa sul web e analizzata nel dettaglio [54], per poterla utilizzare tuttavia occorre settare l'ambiente adatto. Analizzando il codice si vede come vengano utilizzate anche librerie di sistema per poter eseguire il lavoro tramite diversi thread, inoltre osservando attentamente si possono vedere delle operazioni quali flatMap(), distinct() e map() tipiche di Spark. Il codice sopra citato è stato preso online e visionato attentamente, è però possibile prendere spunto per una possibile implementazione dell'algoritmo, dato che le operazioni da eseguire sono le stesse effettuate.

Sono presenti alcune guide online che è possibile seguire per il settaggio delle variabili d'ambiente e per la configurazione dell'ambiente in modo da poter eseguire il codice in Python su un cluster di macchine SQL.

Le funzioni sopra chiamate sono state riportate in seguito, è possibile vedere anche dalla seguente immagine come vengano eseguite delle operazioni di collect(), filter() e map() anche all'interno di essi, principi cardine del framework Spark.

```
def generate_next_c(f_k, k):
    next_c = [var1 | var2 for index, var1 in enumerate(f_k) for var2 in f_k[index + 1:] if
              list(var1)[:k - 2] == list(var2)[:k - 2]]
    return next_c


def generate_f_k(sc, c_k, shared_itemset, sup):
    def get_sup(x):
        x_sup = len([1 for t in shared_itemset.value if x.issubset(t)])
        if x_sup >= sup:
            return x, x_sup
        else:
            return ()

    f_k = sc.parallelize(c_k).map(get_sup).filter(lambda x: x).collect()
    return f_k
```

Figura 115: Funzioni per Apriori in Spark con Python

L'idea quindi è di realizzare attraverso la distribuzione di Anaconda del codice in Python che venga eseguito su un cluster che utilizza tecniche di big data ed il framework Spark per la manipolazione di RDD.

6.3.3 Accenno a FP-Growth con PySpark e DFPS

Per quanto riguarda invece l'algoritmo FP-Growth è più semplice trovare strumenti per l'implementazione nel campo dei big data visto che spesso è considerato un'evoluzione del primo per via delle sue caratteristiche e prestazioni.

I migliori candidati che verranno analizzati sono il Distributed FP-growth algorithm based on Spark anche chiamato DFPS e gli strumenti forniti dalla libreria PySpark. Per quanto riguarda la prima implementazione come si intuisce dal no-

me si ha come scopo l'implementazione dell'algoritmo di data mining in maniera distribuita utilizzando Spark.

Nel caso si decidesse di adottare questo approccio è possibile visionare dei documenti online riguardanti risultati di studi effettuati dal 2015 al 2017 che possono essere utili per un'analisi dettagliata dell'implementazione [55]. Essendo dei documenti per cui è riservato l'accesso ci si è limitati a citare questa metodologia, che viene spesso presentata come una delle migliori nell'implementazione di FP_Growth in Spark.

Lo strumento migliore da utilizzare se invece si decide di utilizzare PySpark è MLlib, che racchiude delle API basate sugli RDD, nell'immagine seguente è possibile vedere l'esempio di implementazione fornito da questa libreria per utilizzare FP-Growth in Python [56]. Anche in questo caso è possibile prendere spunto dal codice fornito per l'implementazione, fornendo come parametro i dati corretti (non quelli mostrati) e modificando le soglie di supporto a piacimento.

```
from pyspark.ml.fpm import FPGrowth

df = spark.createDataFrame([
    (0, [1, 2, 5]),
    (1, [1, 2, 3, 5]),
    (2, [1, 2])
], ["id", "items"])

fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6)
model = fpGrowth.fit(df)

# Display frequent itemsets.
model.freqItemsets.show()

# Display generated association rules.
model.associationRules.show()

# transform examines the input items against all the association rules and summarize the
# consequents as prediction
model.transform(df).show()
```

Figura 116: FP-Growth con PySpark

Dall'ultima immagine è possibile vedere come venga prima inizializzato il modello e poi come vengano estratti prima gli itemset frequenti e poi le relative regole di associazione.

In questo caso non vi è un'implementazione tecnica da effettuare, ma è sufficiente richiamare la funzione come per l'implementazione base. Tuttavia è necessario settare tutto l'ambiente in modo che sia possibile utilizzare il cluster per le elaborazioni dei dati effettuate da PySpark.

7 Conclusioni

Grazie allo studio teorico effettuato nei primi capitoli è stato possibile apprendere e mostrare le diverse tecnologie all'avanguardia per lo sviluppo di applicazioni e le loro differenze. Inoltre si è potuta incrementare e condividere la conoscenza riguardante i processi e le azioni necessarie per gestire correttamente un magazzino di grandi dimensioni.

Per quanto riguarda la realizzazione dell'applicazione, essa è stata possibile in breve tempo, nonostante l'utilizzo di tecnologie e linguaggi non conosciuti, grazie a un duro lavoro di apprendimento che però ha portato i risultati sperati, accomodando tutte le richieste avanzate. Il progetto non solo è terminato con successo, ma anche il modo in cui è stato svolto ha raggiunto dei buoni requisiti in materia di qualità e usabilità. Inoltre il lavoro presentato fornisce la possibilità di imparare ad implementare una web app full-stack con diverse tecnologie combinate tra loro, sicuramente può essere preso come fonte di ispirazione per altri sviluppi.

La parte finale relativa lo studio dell'introduzione del data mining invece ha dato modo di applicare brevemente alcune nozioni di teoria apprese durante il percorso accademico in corsi facenti strettamente parte della specializzazione scelta e che hanno scaturito una passione a livello personale.

Oltre ad aspetti tecnici questo progetto di tesi ha anche fornito la possibilità di integrarsi in un ambiente di lavoro, acquisendo esperienza. Durante la sua intera durata è stato possibile dialogare con altre persone che si sono rese disponibili a condividere le proprie conoscenze, mentre in altre occasioni si ha avuto modo di mostrare le competenze personali, favorendo sempre uno spirito di condivisione di informazioni.

Il progetto nel complesso ha fornito l'opportunità di mettere in pratica argomenti appresi in moltissimi dei corsi effettuati da docenti dell'ateneo. Per questo motivo grazie a questa tesi è stato possibile mettere in pratica parecchi concetti che fino a questo punto erano solo stati visti teoricamente.

Oltre che ad un risultato didattico va sottolineata la crescita professionale che è maturata nei vari mesi. Da una parte l'opportunità di imparare tecnologie, framework e linguaggi nuovi ha aiutato a completare perfettamente il percorso di studi intrapreso, in modo da avere una versatilità nel campo dell'informatica in moltissimi settori. Dall'altra l'integrazione in un ambiente di lavoro, l'utilizzo di metodologie agili e il dialogo costante ha rappresentato una ricchezza anche dal punto di vista umanitario, massimizzando il comfort e aumentando di conseguenza la produttività, caratteristica da non sottovalutare in uno sviluppo di un progetto.

Il completamento della tesi è anche stato possibile grazie a una capacità di adattamento logico appresa durante lo studio in questi anni, che ha conseguito un rapido apprendimento di tecnologie e linguaggi nuovi in poco tempo.

Riferimenti bibliografici

[1] Hector Sunol. *6 Primary Warehouse Processes & How to Optimize Them*. 2019. URL: https://articles.cyzerg.com/warehouse-processes-how-to-optimize-them.

[2] Hector Sunol. *Warehouse Operations: Optimizing the Receiving Process*. 2019. URL: https://articles.cyzerg.com/receiving-process-optimization-warehouse-operations.

[3] Hector Sunol. *Warehouse Operations: Optimizing the Putaway Process*. 2019. URL: https://articles.cyzerg.com/putaway-process-optimization-warehouse-operations.

[4] Jorge Jimeno Bernal. *ABC Analysis – How to optimize warehouses and inventory*. 2013. URL: https://pdcahome.com/english/151/abc-analysis-how-to-optimize-warehouses-and-inventory/.

[5] Hector Sunol. *Warehouse Operations: Optimizing the Storage Process*. 2019. URL: https://articles.cyzerg.com/warehouse-operations-optimizing-the-storage-process.

[6] Shelly Stazzone. *6 Types of Warehouse Storage Systems*. 2020. URL: https://www.camcode.com/asset-tags/types-of-warehouse-storage-systems/.

[7] Derrick Weiss. *15 Warehouse Management KPIs You Need to Track*. 2018. URL: https://www.skunexus.com/blog/warehouse-management-kpis.

[8] Hector Sunol. *Warehouse Operations: Optimizing the Picking Process*. 2019. URL: https://articles.cyzerg.com/picking-process-optimization-warehouse-operations.

[9] Martin Murray. *Efficient Packaging Methods in a Warehouse*. 2019. URL: https://www.thebalancesmb.com/packaging-in-the-warehouse-2221182.

[10] Hector Sunol. *Warehouse Operations: Optimizing the Shipping Process*. 2019. URL: https://articles.cyzerg.com/warehouse-operations-optimizing-the-shipping-process.

[11] ClarusWMS. *What's the Difference Between Inventory and Warehouse Management?* 2018. URL: https://www.claruswms.co.uk/inventory-and-warehouse-management/.

[12] Joannès Vermorel. *STOCK KEEPING UNIT (SKU)*. 2013. URL: https://www.lokad.com/it/definizione-stock-keeping-unit-(sku).

[13] Brian Barry. *Top 6 Features to Look For In An Inventory Management System*. 2020. URL: https://www.fcbco.com/blog/top-6-features-to-look-for-in-an-inventory-management-system.

[14] Wikipedia. *Enterprise resource planning — Wikipedia, L'enciclopedia libera*. 2020. URL: https://it.wikipedia.org/w/index.php?title=Enterprise_resource_planning&oldid=112969246.

[15] SOFTWARE TESTING HELP. *Top 10 Best Warehouse Management Software Systems (WMS) In 2020*. 2020. URL: https://www.softwaretestinghelp.com/warehouse-management-software/.

[16] Wikipedia. *Software development kit — Wikipedia, L'enciclopedia libera*. 2019. URL: https://it.wikipedia.org/w/index.php?title=Software_development_kit&oldid=108041678.

[17] Tom Greenhalgh. *NATIVE APPS VS. WEB APPS: STRENGTHS AND WEAKNESSES YOU NEED TO KNOW*. 2018. URL: https://www.apppartner.com/native-apps-vs-web-apps-strengths-and-weaknesses-you-need-to-know/.

[18] StatCounter. *Mobile Operating System Market Share Worldwide*. 2020. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[19] The App Solutions Inc. USA. *IOS VS ANDROID APP DEVELOPMENT*. 2019. URL: https://theappsolutions.com/blog/development/ios-vs-android/#contents_0.

[20] Android. *Android Jetpack*. 2020. URL: https://developer.android.com/jetpack/.

[21] tutorialspoint. *Android - Architecture*. 2020. URL: https://www.tutorialspoint.com/android/android_architecture.htm.

[22] Chris Griffith. *What is Hybrid App Development?* 2020. URL: https://ionicframework.com/resources/articles/what-is-hybrid-app-development.

[23] Vitaly K. *Native vs. Hybrid App Development: Which Approach to Choose?* 2019. URL: https://www.cleveroad.com/blog/native-or-hybrid-app-development-what-to-choose.

[24] Microsoft. *What is Xamarin?* 2019. URL: https://docs.microsoft.com/it-it/xamarin/get-started/what-is-xamarin.

[25] Microsoft. *Cross-platform mobile development in Visual Studio*. 2019. URL: https://docs.microsoft.com/it-it/visualstudio/cross-platform/cross-platform-mobile-development-in-visual-studio?view=vs-2019.

[26] Facebook. *React Native*. 2020. URL: https://reactnative.dev/.

[27] Facebook. *Setting up the development environment*. 2020. URL: https://reactnative.dev/docs/environment-setup.

[28] Flutter. *Technical overview*. 2020. URL: https://flutter.dev/docs/resources/technical-overview.

[29] Ionic. *Ionic Framework 5*. 2020. URL: https://ionicframework.com/.

[30] Nick Hyatt. *Introducing Capacitor Support for Ionic Appflow*. 2020. URL: https://ionicframework.com/blog/introducing-capacitor-support-for-ionic-appflow/.

[31] Naiya Sharma. *React Native vs. Xamarin vs. Ionic vs. Flutter: Which is best for Cross-Platform Mobile App Development?* 2018. URL: https://www.apptunix.com/blog/frameworks-cross-platform-mobile-app-development/.

[32] Yeeply. *Sviluppo web app: vantaggi e svantaggi*. 2020. URL: https://it.yeeply.com/blog/vantaggi-e-svantaggi-web-app/.

[33] digimktg@blockchainsimplified.com. *Progressive Web App Development — Technical Components of PWA*. 2020. URL: https://medium.com/

@digimktg/progressive-web-app-development-technical-components-of-pwa-139ba8e0f0a6.

[34] Microsoft. *BlazorModelli di hosting ASP.NET Core*. 2020. URL: https://docs.microsoft.com/it-it/aspnet/core/blazor/hosting-models?view=aspnetcore-3.1.

[35] Wikipedia. *Angular*. 2020. URL: https://it.wikipedia.org/wiki/Angular.

[36] Wikipedia. *React (web framework)*. 2020. URL: https://it.wikipedia.org/wiki/React_(web_framework).

[37] Wikipedia. *Vue.js*. 2020. URL: https://it.wikipedia.org/wiki/Vue.js.

[38] Wikipedia. *Laravel*. 2020. URL: https://it.wikipedia.org/wiki/Laravel.

[39] Wikipedia. *Ruby on Rails*. 2020. URL: https://it.wikipedia.org/wiki/Ruby_on_Rails.

[40] Wikipedia. *Django (informatica)*. 2020. URL: https://it.wikipedia.org/wiki/Django_(informatica).

[41] Wikipedia. *CodeIgniter*. 2020. URL: https://it.wikipedia.org/wiki/CodeIgniter.

[42] Wikipedia. *jQuery*. 2020. URL: https://it.wikipedia.org/wiki/JQuery.

[43] Wikipedia. *Bootstrap (informatica)*. 2020. URL: https://it.wikipedia.org/wiki/Bootstrap_(informatica).

[44] Npm. *ReactN*. 2020. URL: https://www.npmjs.com/package/reactn.

[45] Npm. *React Loading Overlay*. 2018. URL: https://www.npmjs.com/package/react-loading-overlay.

[46] Npm. *React Webcam*. 2020. URL: https://www.npmjs.com/package/react-webcam.

[47] Npm. *React Bootstrap*. 2020. URL: https://www.npmjs.com/package/react-bootstrap.

[48] Reactstap. *Reactstrap*. 2020. URL: https://reactstrap.github.io/.

[49] Npm. *React Router Dom*. 2020. URL: https://www.npmjs.com/package/react-router-dom.

[50] Npm. *React Icons*. 2020. URL: https://www.npmjs.com/package/react-icons.

[51] Matt Gaunt. *Service Workers: an Introduction*. 2020. URL: https://developers.google.com/web/fundamentals/primers/service-workers/.

[52] Npm. *react-axios*. 2018. URL: https://www.npmjs.com/package/react-axios.

[53] Microsoft. *Driver SQL per Python*. 2017. URL: https://docs.microsoft.com/it-it/sql/connect/python/python-driver-for-sql-server?view=sql-server-ver15.

[54] Zhang Zhe. *Spark-Apriori*. 2018. URL: https://github.com/zhang943/Spark-Apriori.

[55] Hui Yang Xiujin Shi Shaozong Chen. *DFPS: Distributed FP-growth algorithm based on Spark*. 2017. URL: https://ieeexplore.ieee.org/document/8054308.

[56] Apache Spark. *Frequent Pattern Mining*. 2020. URL: https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html#fp-growth.