UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Progettazione e sviluppo di una web app per l'asset management in Oracle APEX

Design and development of a web app for asset management in Oracle APEX

Relatore:

Prof. **Emanuele Storti**

Tesi di Laurea di:

Alessandro Pieragostini

Anno Accademico 2022/23

INDICE

INDICE DELLE FIGURE

Introduzione

Questa tesi è basata su quanto svolto durante il tirocinio inter-curriculare presso l'azienda Magazzini Gabrielli, nello specifico nel reparto DSI (Direzione Sistemi Informativi). L'esigenza della DSI era avere uno strumento unico e centralizzato in grado di tenere traccia degli asset presenti all'interno del magazzino interno al reparto e di gestire le loro movimentazioni verso determinate locazioni, quindi altri magazzini, negozi (punti di vendita) o direzioni (reparti specifici). Fino ad ora, ogniqualvolta si voleva registrare un immagazzinamento o un'uscita di un asset da un magazzino verso una locazione o viceversa, era necessario che l'operatore incaricato appuntasse i vari movimenti con strumenti come ad esempio fogli Excel, rendendo precaria la storicizzazione dei dati e perdendo così anche la possibilità di analizzare e sfruttare molte informazioni invece utili al miglioramento della qualità del lavoro, alla gestione del capitale riferito al materiale allocato nelle varie posizioni fisiche e ad avere un'informazione sulle quantità in magazzino. Il progetto in questione ha lo scopo di creare un applicativo di gestione che possa essere utile al soddisfacimento delle esigenze appena presentate. La web app è stata realizzata tramite l'utilizzo di Oracle Application Express (APEX), uno strumento basato sul low-code che permette di creare delle web app con la possibilità di renderla Progressive (PWA) così da poterla utilizzare in modo veloce e reattivo da dispositivi mobile.

Nel primo capitolo inizialmente verrà fatta una breve introduzione all'azienda così da poter dare un'idea del campo in cui si sta lavorando. Successivamente si tratterà la fase di analisi, argomento principale del capitolo, ossia una panoramica degli obiettivi che la

web app dovrà raggiungere per far sì che rispecchi a pieno la volontà dell'azienda e una valutazione sui requisiti che il prodotto finale dovrà rispettare.

Nel secondo capitolo viene fatta un'introduzione allo strumento utilizzato, con le motivazioni che hanno portato alla scelta di esso, spiegando il perché l'utilizzo di Oracle APEX può tornare utile in alcune situazioni che solitamente in aziende non di tipo puramente informatico possono capitare, fornendo un'ottima alternativa alla programmazione vera e propria

Il terzo capitolo riguarda la progettazione, sia del database che dell'applicazione, quindi lo sviluppo dello schema E-R e uno studio di come deve essere suddivisa la web app soprattutto a livello di front-end per dar si che sia comoda ed intuitiva da utilizzare.

Nel quarto capitolo viene descritta la fase di implementazione, descrivendo come sono state sviluppate le varie funzionalità, parlando anche delle funzionalità utili che APEX mette a disposizione, le quali hanno giocato un ruolo fondamentale per la buona riuscita di alcune parti della web app.

Capitolo 1: Analisi degli obiettivi e dei requisiti

Prima di affrontare lo sviluppo della web app è stato necessario avviare una fase di analisi in cui si è discusso di come l'applicazione potesse andare a migliorare e velocizzare la questione dell'asset management del magazzino interno agli uffici del reparto DSI. Per capire come si struttura l'azienda e quindi capire meglio di cosa si parla è stato fatta una breve introduzione dell'azienda.

1.1 Introduzione all'azienda

La Magazzini Gabrielli Spa vede le sue origini nel 1953 quando Pietro Gabrielli e sua moglie Celsira iniziano l'attività commerciale in un emporio ad Ascoli Piceno, città sede attuale dell'azienda. Il successo della nuova attività spinge la famiglia all'apertura di sei nuovi punti vendita e nel 1967 viene introdotto il reparto alimentare. Negli anni Settanta l'attività diventa una vera e propria realtà imprenditoriale attraverso l'importante contributo dei tre fratelli Michele, Luciano e Giancarlo che creano la Magazzini Gabrielli Spa. La crescita dell'azienda è vertiginosa e tra il 1978 e il 2000 si assiste alla genesi delle due insegne commerciali che caratterizzano poi tutti i punti vendita successivi: Tigre per i supermercati e Oasi per le grandi superfici. L'ereditarietà carismatica dei Gabrieli passa attraverso le generazioni e oggi l'azienda è una solida realtà della Grande Distribuzione Organizzata.

La Magazzini Gabrielli Spa gestisce negozi (superstore, supermercati e superette) a vocazione prettamente alimentare. Organizza attività di acquisto e di consumo da fornitori (sia nazionali che locali). Razionalizzando le risorse in complessi processi aziendali, la merce è offerta alla clientela per mezzo di una rete di punti di vendita sia a gestione diretta

che tramite imprenditori affiliati. In un settore dominato dall'omologazione del servizio offerto, la pluriennale presenza nei territori di appartenenza e la multicanalità dell'azienda, data dalla capacità di offrire superfici di grandi e piccole dimensioni, permette di conoscere a fondo la clientela di riferimento e dunque di modulare un'offerta commerciale aderente alle sue esigenze.

1.2 Obiettivi

Questo progetto nasce quindi con l'intenzione di centralizzare, informatizzare e velocizzare tutto ciò che riguarda la movimentazione degli asset di carattere IT presenti e gestiti a livello di magazzino interno. Si riuscirà così a sfruttare le informazioni derivanti dai trasferimenti delle merci, potendo quindi:

- tenere traccia delle quantità in magazzino ma anche nelle varie locazioni;
- tenere traccia delle movimentazioni di ciascun asset tra le locazioni;
- quantificare in valore le rimanenze in magazzino, riuscendo così ad avere un confronto con la giacenza contabile comunicata periodicamente dall'amministrazione, calcolata tramite registrazioni contabili;
- velocizzare i tempi di immagazzinamento delle informazioni all'interno dello stesso strumento e in un unico database;
- sorvegliare le scorte, così da rendersi conto quando si è sotto scorta e quindi effettuare ordini per non rimanere senza nel caso di urgenze;

- semplificare l'attività di inventario, abbattendo i tempi e riducendo così i costi aggiuntivi legati allo svolgimento di quest'ultima.

1.3 Requisiti

Dopo aver chiarito quali sono gli obiettivi principali, è stato possibile proseguire l'attività di analisi procedendo con l'analisi dei requisiti, che andranno a definire più nel dettaglio ciò che è emerso dall'analisi degli obiettivi, cercando di stilare una lista requisiti (funzionali e non) così da avere sempre una sequenza di punti in modo tale da avere in futuro la possibilità di poter fare un confronto con ciò che era stato fatto e ciò che invece era ancora da implementare, riuscendo velocemente ad avere un punto della situazione mano a mano che si andava avanti con lo sviluppo.

1.3.1 Requisiti funzionali

I requisiti fondamentali che si andranno successivamente a sviluppare sono:

- *Visualizzazione report del magazzino*: la possibilità di avere in modo rapido e reattivo di consultare una tabella che indicasse la giacenza fisica in magazzino, in modo da tener traccia di un inventario e tenere sott'occhio la disponibilità in base a ciascuna categoria di asset in tempo reale;

- *Visualizzazione di asset sotto scorta*: avere a disposizione un qualsiasi tool in grado di mostrare in tempo reale gli asset che sono numericamente sotto una certa soglia minima prestabilita per ciascun asset, così da tenere sotto controllo le scorte e capire quando è il momento di effettuare nuovi ordini;

- *Visualizzazione report della giacenza contabile in magazzino*: essere in condizione di venire a conoscenza della giacenza contabile tutte le volte in cui venga richiesto in ambito amministrativo per poter poi effettuare un controllo incrociato tra i dati riscontrando o meno anomalie a livello di magazzino e movimentazioni;

- *Visualizzazione report della giacenza contabile per ogni locazione*: poter essere a conoscenza del valore complessivo degli asset assegnati ad una certa locazione come un punto di vendita o una direzione (uffici);

- *Monitoraggio del ciclo di vita di un asset*: avere a disposizione un confronto tra la data di inizio e di fine ciclo di vita per monitorare l'affidabilità di un asset rispetto ad un altro inerente alla stessa categoria, oppure capire se all'interno di una locazione avviene o meno il corretto trattamento di essi;

- *Tracciamento delle movimentazioni*: avere uno storico esaustivo delle movimentazioni degli asset, quindi la possibilità di sapere principalmente le informazioni relative a sorgente, destinazione, data in cui si effettua e in che stato sono gli asset al momento della movimentazione.

1.3.2 Requisiti non funzionali

Oltre ai requisiti funzionali, per una applicazione di questo genere si è reso necessario tener conto anche degli aspetti non funzionali, ovvero di cosa si necessitasse affinché un qualsiasi operatore la utilizzasse riuscisse ad usarla con tempi ridotti. I requisiti principali che sono emersi sono:

- *Velocità di esecuzione:* la reattività della web app in generale non può mai essere un aspetto da non sottovalutare, soprattutto quando il suo obiettivo è quello di ridurre i tempi;

- *Interfacciamento con database Oracle:* tenere in considerazione gli strumenti già presenti in azienda e far comunicare la web app con essi per garantire la centralizzazione dell'informazione;

- *Espandibilità:* tener conto della possibilità di poter implementare successivamente nuove funzionalità nel caso in cui si voglia tenere a disposizione informazioni diverse da quelle sviluppate nella prima versione;

- *Genericità:* questa web app è applicata soltanto alla gestione degli asset IT del magazzino interno alla DSI ma uno sviluppo non fossilizzato negli specifici asset darà modo di valutare un possibile sviluppo di una nuova web app (o espandere la stessa) dando la possibilità di poter tracciare ogni singolo asset materiale dell'intera azienda;

- *Responsive*: deve essere progettata per adattarsi a diverse dimensioni di schermi, che la renda adatta per dispositivi desktop, tablet e smartphone;

- *Utilizzo per smartphone:* questo è dato dal fatto che l'applicazione dovesse essere fruibile dai dispositivi dati in dotazione ai dipendenti i quali sono provvisti di sistema operativo Android e posseggono un lettore di barcode integrato, in modo tale da entrare in magazzino con un dispositivo mobile e sfruttare il lettore per

velocizzare i tempi di immagazzinamento, riuscendo a captare il seriale dell'asset che si vuole gestire e riducendo le eventuali tempistiche a cui si sarebbe andati incontro nel caso si fosse dovuto inserire dati manualmente;

- *Intuitività:* la possibilità di avere dei report facili ed intuitivi utili alla consultazione dei dati è fondamentale per soddisfare tutte le esigenze dell'azienda presentate in precedenza.

Capitolo 2: Strumenti utilizzati

In questo capitolo verranno descritte le caratteristiche e i punti di forza dello strumento scelto. Come già anticipato, la scelta dello strumento da utilizzare è ricaduta su Oracle APEX per vari motivi, sia per la presenza di database Oracle all'interno della Magazzini Gabrielli, sia per l'efficienza, la versatilità e rapidità di sviluppo dello strumento, che utilizza un approccio low-code. In sintesi, l'utilizzo di esso permette di rispettare tutti i requisiti descritti nel capitolo precedente.

2.1 Oracle APEX

Oracle APEX (Oracle Application Express) fornisce gli strumenti necessari per creare le applicazioni in un'unica piattaforma estensibile, eseguita nell'ambito del database Oracle. Lo scopo principale di questo software consiste nel consentire agli sviluppatori di creare facilmente applicazioni con funzionalità, prestazioni ed esperienza dell'utente finale di alto livello anche non avendo competenze sufficienti nell'ambito dell'informatica e soprattutto, nello specifico, senza la necessità di conoscenze avanzate di programmazione front-end, con la tendenza quindi di eliminare le caratteristiche di complessità legate allo sviluppo e alla distribuzione di applicazioni aziendali. Di seguito alcuni motivi per cui la scelta di Oracle APEX si è rivelata oculata [1].

2.1.1 Eliminazione della complessità

Oracle APEX ha semplificato notevolmente il processo di sviluppo e distribuzione dell'applicazione a tutti i livelli, fornendo una piattaforma low-code che è intuitiva, rapida da utilizzare e richiede meno risorse, soprattutto tempo, rispetto ad altre opzioni

disponibili. Tuttavia, va sottolineato che lo sviluppo di applicazioni low-code con Oracle APEX va oltre la semplice riduzione della quantità di codice necessaria per creare applicazioni. L'utilizzo di questo strumento ha aiutato a gestire e ridurre la complessità a tutti i livelli:

- Eliminazione del mapping tra gli oggetti relazionali e gli oggetti applicazione;
- Eliminazione della complessità delle chiamate di procedura remota;
- Eliminazione della proliferazione delle connessioni al database;
- Eliminazione della logica di applicazione di livello intermedio: tutto è nel database, non è quindi necessario progettare, sviluppare o gestire livelli intermedi complessi;
- Eliminazione della complessità delle funzioni di High Availability[1] e disaster recovery[2] [1].

2.1.2 Sviluppo semplice e veloce

Questo strumento si è distinto rispetto ad altre opzioni per la sua facilità d'uso e apprendimento. L' Application Builder fornisce allo sviluppatore un'interfaccia browser completa e intuitiva che lo guida attraverso tutte le fasi della creazione delle applicazioni.

[1] *High Availability*: capacità di un sistema o di un componente di assicurare una performance alta senza interruzioni in un dato periodo di tempo [5].

[2] *Disaster Recovery:* l'insieme delle misure tecnologiche e logistico/organizzative atte a ripristinare sistemi, dati e infrastrutture a fronte di gravi emergenze che ne intacchino la regolare attività [6].

Grazie ai componenti di Oracle APEX, è possibile arricchire le applicazioni con funzionalità avanzate, limitando al minimo la necessità di scrivere codice personalizzato. Il processo di sviluppo inizia solitamente con un foglio di calcolo o un modello dati, e le applicazioni vengono create in un ambiente basato su browser a bassa codifica, in cui è possibile eliminare la quasi totalità della codifica manuale ma consentendo comunque una personalizzazione a livello di front-end tramite file CSS. Inoltre, Oracle APEX opera direttamente sui dati presenti nelle tabelle del database, eliminando la necessità di definire strutture dati complesse per gestire i flussi di dati tra l'applicazione e il database [1].

2.1.3 Sicurezza

Oracle APEX è pensato per sviluppare applicazioni web dotate di un robusto sistema di sicurezza integrato. In un contesto in cui gli standard web sono in continua evoluzione, le norme di sicurezza si rafforzano e le minacce degli hacker crescono costantemente, concentrarsi sulla sicurezza significa garantire che le applicazioni siano protette e rimangano all'avanguardia [1].

2.1.4 Portabilità

Oracle APEX può essere eseguito ovunque: nel cloud Oracle, in locale e ovunque sia disponibile un database Oracle. Le applicazioni possono essere distribuite agevolmente in qualsiasi ambiente, per questo motivo si era avvantaggiati avendo già a disposizione le infrastrutture necessarie all’interno dell’azienda. La distribuzione dell'applicazione e dei dati nel database è notevolmente più semplice a quella delle applicazioni di livello intermedio [1].

2.1.5 Progressive Web App

Una delle necessità dell'azienda era il poter utilizzare la web app comodamente tramite dispositivi mobile. In Oracle APEX, quando si crea un'applicazione, nelle Features basta selezionare la voce "Install Progressive Web App" per abilitare la possibilità di installare la propria web app sul dispositivo mobile, e quindi renderla progressive. Il termine Progressive Web App (PWA) si riferisce ad una applicazione web che viene sviluppata e caricata come una normale pagina web, ma che si comporta in modo simile alle applicazioni native quando utilizzata su un dispositivo mobile [2].

2.2 Low-Code

Il paradigma del low-code rappresenta un approccio visuale alla creazione di software che accelera notevolmente la distribuzione delle applicazioni, richiedendo una quantità minima di programmazione manuale. Le interfacce grafiche e le funzionalità di trascinamento e rilascio offerte dalle piattaforme low-code automatizzano diversi aspetti del processo di sviluppo, riducendo al minimo la necessità di seguire i tradizionali approcci di programmazione. Queste piattaforme low-code democratizzano l'attività di sviluppo delle applicazioni, soprattutto per i "citizen" developer, cioè gli utenti aziendali che non possiedono obbligatoriamente una formazione approfondita in programmazione, come ad esempio gli analisti aziendali, i project manager, i commercialisti o i ragionieri.

Questi strumenti permettono ai dipendenti meno tecnici di avere più impatto sul business in molti modi, ad esempio alleggerendo i backlog[3] del dipartimento IT, riducendo lo

[3] *Backlog*: lista ordinata di tutte le funzionalità/item, poco importa la forma o il formato dell'item, che può essere user story, specifiche, brief, un modello con powerpoint o gherkin [7].

shadow IT[4] e appropriandosi dei flussi di lavoro della gestione del processo di business (BPM - Business Process Management). Tuttavia, le piattaforme low-code sono vantaggiose anche per i programmatori con maggiore esperienza. Poiché richiedono poca o nessuna esperienza di codifica, permettono una maggiore flessibilità di background di codifica di uno sviluppatore. Ad esempio, capita spesso che all'interno di una azienda alcune applicazioni già presenti richiedano conoscenze di diversi e specifici linguaggi di programmazione, andando così a limitare le scelte degli sviluppatori. Abolendo questa barriera, le piattaforme low-code accorciano il ciclo di vita di sviluppo delle applicazioni, permettendo agli sviluppatori di fare di più in meno tempo, andando così incontro all' economia dell'azienda.

La pandemia di COVID-19 ha aumentato la necessità di automatizzare i processi e dare priorità alle iniziative di trasformazione digitale. Le piattaforme low-code rispondono a tale esigenza, aiutando a semplificare i flussi di lavoro e ad accelerare i progetti di automazione.

Un approccio low-code favorisce la creazione veloce di applicazioni, migliorando l'esperienza dell'utente. Sviluppatori professionisti e non traggono vantaggio da funzionalità di base come un ambiente di sviluppo integrato (IDE - Integrated Development Environment) visivo, connettori di dati integrati e/o API e modelli di codice [3]. Queste funzionalità avanzate delle piattaforme low-code contribuiscono a ottimizzare il processo DevOps[5], consentendo di risparmiare tempo prezioso e concentrarsi

[4] *Shadow IT*: descrive sistemi e soluzioni IT implementati e usati all'interno delle organizzazioni senza l'approvazione esplicita dell'organizzazione stessa [8].
[5] *DevOps*: metodologia di sviluppo del software utilizzata in informatica che punta alla comunicazione, collaborazione e integrazione tra sviluppatori e addetti alle operazioni della information technology (IT).

maggiormente sull'innovazione, tutto ciò a beneficio dell'azienda che, abbattendo i tempi, riuscirà a trarre un vantaggio economico tenendo intatta la qualità del prodotto finale e non sacrificandola, come solitamente succede ad un qualsiasi progetto secondo il project management triangle. Questo fa intuire che la scelta degli strumenti è un'analisi fondamentale che l'azienda non può trascurare se si vogliono ottimizzare qualità, tempi e costi. Per progetti come questo, di carattere fortemente gestionale, basato su un database con molti dati e con poca logica di elevata complessità, strumenti low-code miglioreranno la gestione del progetto.

2.2.1 Quick SQL

Il Quick SQL fornito da Oracle APEX è sicuramente uno strumento di low-code utile allo sviluppatore per la creazione della base dati, è progettato per ridurre il tempo e l'impegno necessari per creare tabelle, trigger e strutture di indici SQL. Il Quick SQL non è progettato per sostituire la modellazione dei dati, è semplicemente un modo rapido per sviluppare uno script per semplici tabelle e viste, ovvero generare DDL (Data Definition Language) ed una volta generato, l'SQL può essere ottimizzato ed espanso. Fornisce un modo rapido per generare l'SQL necessario a creare un modello di dati relazionale da un documento di testo indentato, seguendo alcune tecniche definite nelle varie documentazioni. Ad esempio, come è possibile notare nell'esempio riportato in *figura 1,* una tabella che viene scritta più indentata rispetto ad un'altra erediterà automaticamente la chiave esterna della tabella al di sopra di essa. È possibile infine generare facilmente dati casuali da inserire all'interno del database. Nella *figura 1* viene mostrato un esempio, preso direttamente dai Samples all'interno della pagina di Help del Quick SQL:

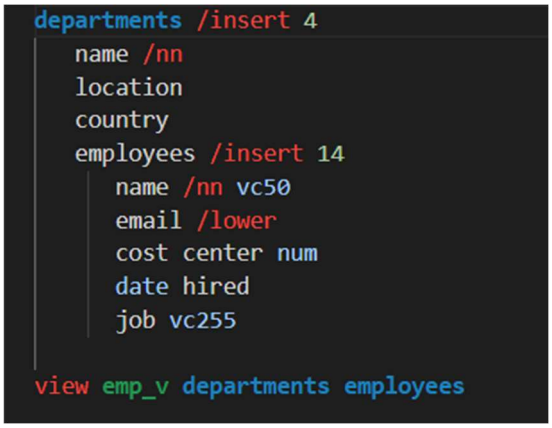


Figura 1: Quick SQL

In questo esempio, vengono generate le tabelle Departments e Employees, dove l'ultima erediterà la chiave esterna della prima essendo scritta più rientrante rispetto a Departments. È possibile notare inoltre come utilizzando il simbolo "/" si possano indicare diversi comandi, elencati in diverse sezioni della pagina di Help. Nel codice in *figura 1* vengono riportati alcuni esempi di come vengono dichiarate alcune proprietà o alcuni constraint: "/insert NN" per indicare di generare casualmente NN record, "/nn" per rendere not null quel campo, "/lower" per indicare che i dati di quella colonna devono essere obbligatoriamente lower case, quindi genera un constraint. Infine, si può vedere come è possibile assegnare o meno i tipi di dato (vc255 e vc50 per specificare un varchar2 di rispettivamente 255 e 50 caratteri, num invece sta per number) e nel caso non vengano specificati verranno generati come varchar2 con 4000 caratteri. Nel capitolo sull'implementazione del database verrà mostrato nello specifico il codice usato per generare la base di dati per questa web app, insieme al risultato del Quick SQL.

Capitolo 3: Progettazione

3.1 Progettazione Database

Identificazione delle entità e relazioni fondamentali

Dopo aver affrontato l'analisi dei requisiti è stato possibile constatare quali sono le entità fondamentali per la base di dati, in modo tale da poter costruire uno schema E-R. Le entità fondamentali saranno: Asset, Movimentazione, Fornitore e Locazione.

3.1.1 Schema Asset

Per asset si intende la definizione di uno specifico asset ogni volta che ne viene acquistato uno e di conseguenza entra in magazzino, come fosse quindi la riga di una bolla di entrata. Sarà quindi definito da una classificazione, un seriale (se ne dispone, quindi se è o meno un materiale di consumo), il prezzo di acquisto unitario, la quantità acquistata (che sarà "1" per gli asset con seriale poiché unici e "n" per gli asset di consumo), la data di inizio e fine ciclo di vita per monitorare la durata di un asset (nello specifico utile soltanto per gli asset che dispongono di un seriale).

Si vanno quindi ad implementare le altre entità: Classificazione e le sue componenti Categoria e Marca. Per quanto riguarda la classificazione, essa avrà quindi una categoria (es. PC, mouse, tastiera, cassa), una marca ed un modello. Possiede inoltre un campo per indicare se il genere di asset, riferito alla specifica categoria, dispone o meno di un seriale. L'entità Categoria sarà composta principalmente da un campo per descrivere il tipo di categoria e un campo che servirà da riferimento per mantenere l'informazione del numero

minimo di asset di quella specifica categoria che si vorrà avere a disposizione in magazzino, ovvero la soglia minima. La composizione dell'entità Marca riguarderà soltanto l'informazione riguardante la descrizione della marca.

Lo schema asset complessivamente sarà quindi il seguente (*figura 2)*:

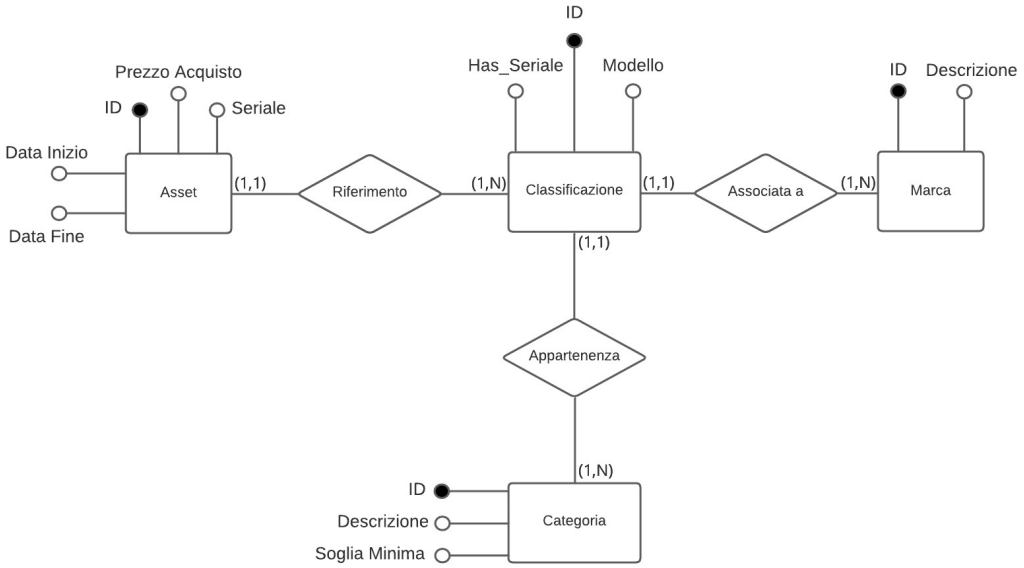


Figura 2: Schema Asset

3.1.2 Schema Movimentazione

Con movimentazione si andrà a descrivere ogni trasferimento e movimento di un asset. Sarà sicuramente necessario tener conto della sorgente e della destinazione (che saranno chiavi esterne) di un asset, la causale del trasferimento, la quantità di materiale movimentata e la data in cui viene effettuata la movimentazione.

Per normalizzare lo schema avremo bisogno di avere le informazioni riguardanti le causali in un'altra tabella a sé. Le causali, da quanto emerso in un'analisi successiva, possono essere di cinque tipi diversi:

- *In stock*, indica che la movimentazione è verso un magazzino, ciò vuol dire che, a meno di rotture, l'asset trasferito tornerà appunto in stock;
- *Assegnato,* indica che un asset è stato assegnato ad una locazione;
- *Rotto,* questa causale verrà utilizzata quando, nel caso generale, un asset rientra in magazzino perché rotto, e quindi sarà poi da inviare in manutenzione;
- *In manutenzione,* descrive la movimentazione quando l'asset è stato inviato per svolgere la manutenzione;
- *Dismesso,* una volta che un asset viene dismesso si crea una movimentazione con questa causale.

Lo schema generale per le movimentazioni sarà il seguente (*figura 3*):

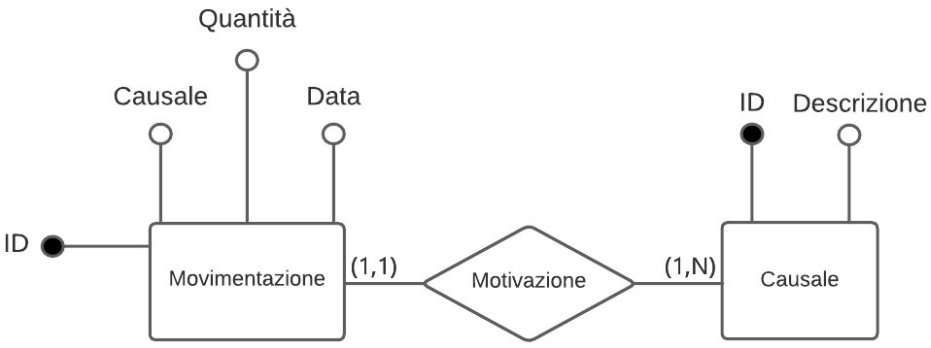


Figura 3: Schema Movimentazione

3.1.3 Schema Locazione

Le locazioni rappresenteranno quindi i vari luoghi, tra i quali gli asset si muoveranno, e saranno di tre tipi: direzione (insieme di uffici che svolgono le stesse mansioni, come ad esempio la DSI), negozio (punti di vendita) e magazzino. Si avrà necessità di conoscere la ragione sociale di ciascun luogo e l'indirizzo. Questa tabella non verrà implementata con il Quick SQL poiché per questa entità si è scelto di creare una "maschera", quindi creare una tabella con determinati campi, descritti successivamente, e poi fare l'unione con i dati già presenti in una tabella con molti più campi, selezionando soltanto le colonne di interesse tramite una vista. Così facendo, ogni volta che la tabella madre viene aggiornata con nuovi dati verrà fatto un merge automatico anche nella tabella di questo schema poiché attinge direttamente dalla tabella principale. I campi di cui si avrà bisogno sono le informazioni riguardanti la città della locazione, il CAP, l'indirizzo, la ragione sociale e il tipo, che indica se si tratta di una direzione, di un negozio (o punto di vendita) o di un magazzino, come già anticipato. Lo schema è rappresentato in *figura 4.*

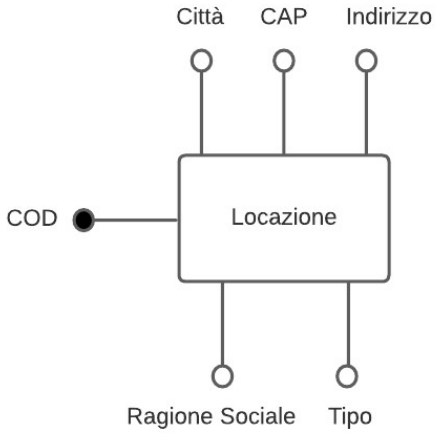


Figura 4: Schema Locazione

3.1.4 Schema Fornitore

I fornitori sono tutti coloro dai quali si acquistano i vari asset IT. Per questa tabella è stato scelto di crearla autonomamente senza l'aiuto del Quick SQL poiché poi è stata fatta una importazione dei dati da delle tabelle Excel, esportate a loro volta dal database principale dell'azienda, grazie alla funzionalità di importazione di Oracle APEX, che permette di importare dati da tabelle esterne soltanto dalle colonne selezionate dallo sviluppatore, facendo corrispondere quindi le informazioni con i campi inerenti. L'entità Fornitore si compone della locazione, ovvero la città dove ha sede legale il fornitore, la ragione sociale e la partita IVA riferita ad esso. Lo schema è visibile in *figura 5*.

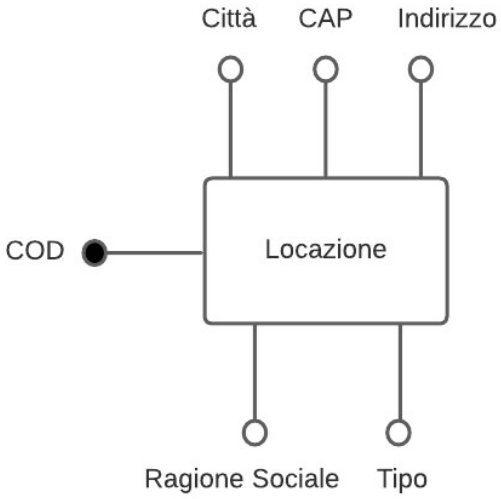


Figura 5: Schema Fornitore

3.1.5 Schema E-R finale

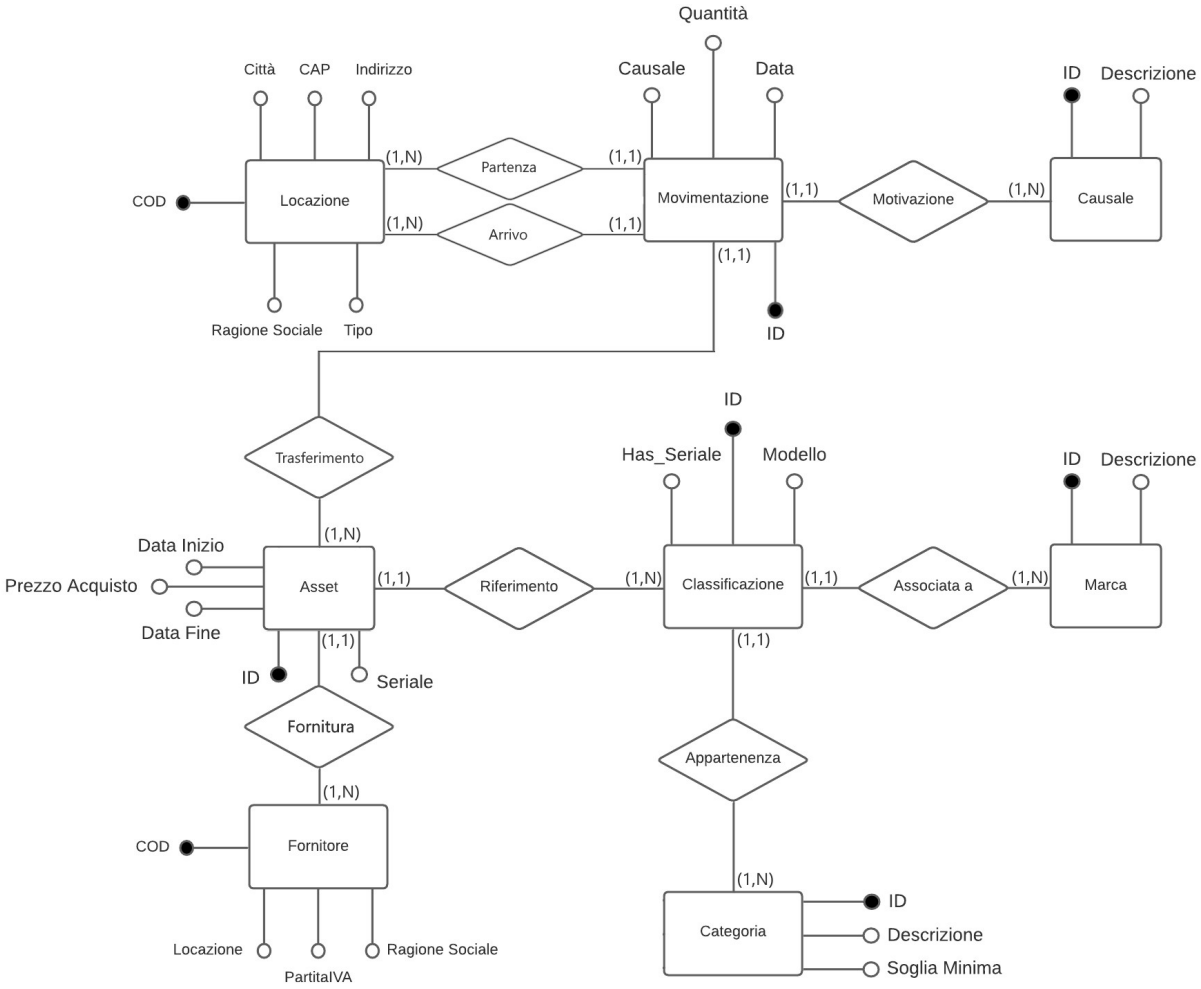


Figura 6: Schema E-R totale

3.2 Progettazione Applicazione

La progettazione dell'app non ha richiesto un lungo studio poiché utilizzando Oracle APEX come strumento non ci si è occupati della struttura dell'app. Questa parte ha rappresentato fondamentalmente la progettazione dei mockup soltanto a livello di bozze cartacee, così da poter capire come strutturare le pagine seguendo anche la precedente analisi dei requisiti, sia funzionali che non, in modo da poterli soddisfare al meglio, il tutto con la consapevolezza di poter sfruttare il designer visuale di Oracle APEX.

Le pagine che andranno a comporre l'applicazione sono:

- Login Page (autogenerata);
- Home Page;
- Creazione Asset;
- Movimentazioni;
- Magazzino;
- Configurazioni, all'interno della quale è possibile inizializzare e gestire le varie classificazioni, categorie e marche degli asset, o anche gestire le locazioni e i fornitori tramite report e form.

Nella *figura 7* è possibile vedere il Navigation Menu, dove sono presenti le pagine create.

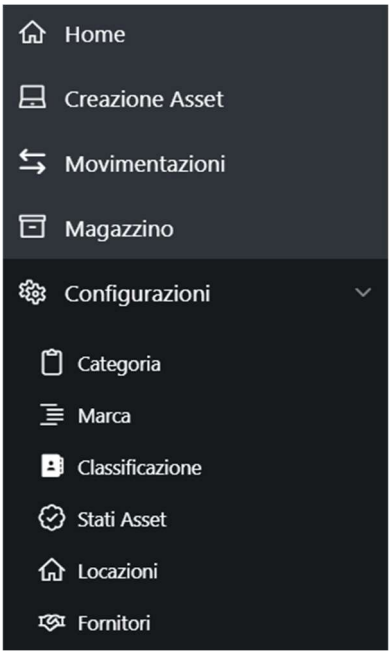


Figura 7: Navigation Menu

L'ultima considerazione è stata che per quasi la totalità delle pagine era necessario implementare anche una pagina di form per l'inserimento dei dati poiché quasi tutte le pagine contengono dei report, nei quali è possibile modificare i record di ognuno ma anche ovviamente inserirne di nuovi.

Capitolo 4: Implementazione

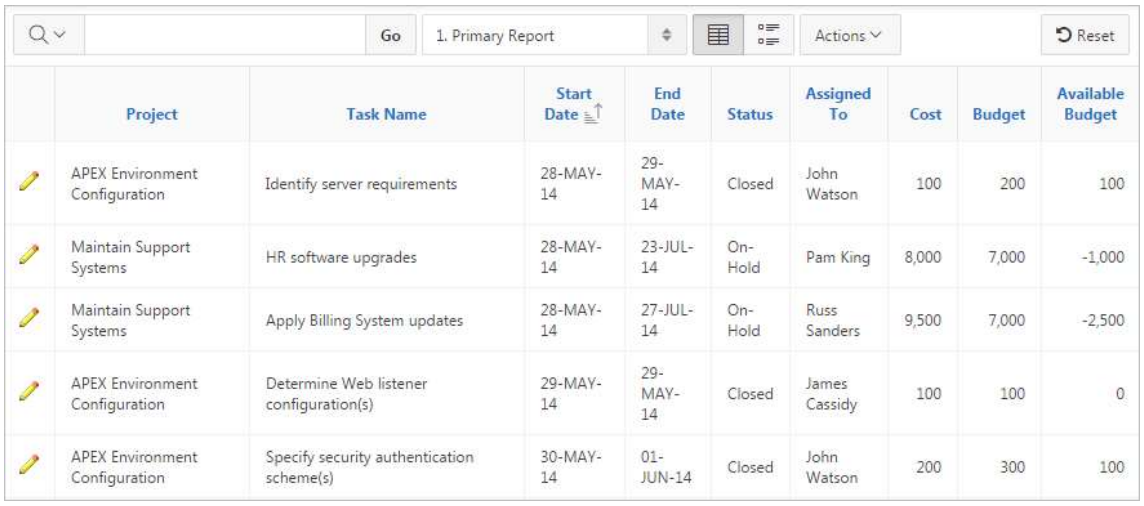
In questa sezione viene spiegato come sono state effettivamente implementate ognuna delle pagine che compongono la web app tramite Oracle APEX, entrando nel dettaglio delle query che sono state utilizzate per mostrare dati e descrivendo aspetti che hanno migliorato l'esperienza dell'utente finale.

4.1 Funzionalità utili

In questo progetto vengono utilizzate diverse funzionalità utili che hanno migliorato il prodotto finale.

4.1.1 Interactive Reports

Partendo da una sorgente iniziale (Source), che può essere una tabella esistente o il risultato di una query, un report di questo tipo permette di personalizzare il suo layout nascondendo o mostrando colonne specifiche grazie all' utilizzo di search bar (implementata in automatico) o di bottoni che permettono di filtrare, ordinare ed evidenziare righe o celle. Questo dà un'impostazione della pagina che consente all'utente finale di cercare ciò che vuole effettuando query direttamente sul report e in modo diretto, senza utilizzare linguaggio SQL. Nella *figura 8* viene riportato un esempio preso dalla documentazione Oracle di come si presenta un Interactive Report.



	Project	Task Name	Start Date	End Date	Status	Assigned To	Cost	Budget	Available Budget
	APEX Environment Configuration	Identify server requirements	28-MAY-14	29-MAY-14	Closed	John Watson	100	200	100
	Maintain Support Systems	HR software upgrades	28-MAY-14	23-JUL-14	On-Hold	Pam King	8,000	7,000	-1,000
	Maintain Support Systems	Apply Billing System updates	28-MAY-14	27-JUL-14	On-Hold	Russ Sanders	9,500	7,000	-2,500
	APEX Environment Configuration	Determine Web listener configuration(s)	29-MAY-14	29-MAY-14	Closed	James Cassidy	100	100	0
	APEX Environment Configuration	Specify security authentication scheme(s)	30-MAY-14	01-JUN-14	Closed	John Watson	200	300	100

Figura 8: Esempio di Interactive Report

È possibile notare, infatti, oltre al report, diversi elementi della pagina. I principali sono i bottoni presenti nella prima colonna, tramite i quali è possibile modificare un record utilizzando un form generato dallo strumento (il più delle volte viene modificato a piacere dello sviluppatore per rendere più chiaro l'inserimento dei dati), la barra di ricerca, tramite la quale si possono mostrare soltanto i record che contengono le parole chiave della ricerca, e il menu Actions, tramite il quale è possibile effettuare diverse azioni (vedi *figura 9*). Una volta personalizzato il report è possibile salvarlo cliccando nel menu Actions poi su Report ed infine Save Report così da poterlo riutilizzare in altre sessioni successive.

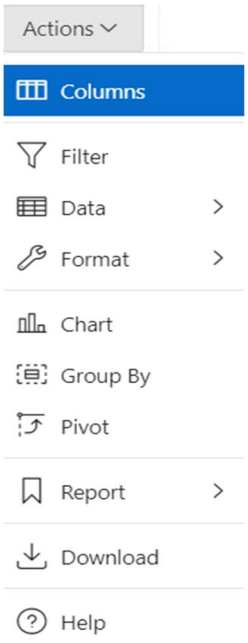


Figura 9: Actions

4.1.2 LOV (List Of Values)

Le LOV (List Of Values) è una definizione statica o dinamica utilizzata per visualizzare un elemento di pagina di tipo specifico. Una LOV può essere statica, ovvero basata sui valori immessi dall'utente o dinamica, quindi basata su una Source che può essere di tipo Local, REST Enable SQL o Web. Una LOV è uno Shared Component dell'applicazione, ciò vuol dire che una volta creata essa può essere applicata in diverse pagine della web app. Vengono definite tutte in un'unica posizione, e questo rende facile utilizzarle o modificarle [4]. Solitamente vengono usate quando in un menù di selezione compaiono le chiavi primarie poiché ovviamente identificano un record, ma l'utente finale potrebbe non essere al corrente del record a cui corrisponde l'identificatore; quindi, tramite le LOV si vanno a mostrare a video le informazioni che fanno capire di quale record si parla, ma una volta effettuata la selezione, la variabile conterrà il valore della chiave, così da poter

essere utilizzabile anche in campo di chiavi esterne. Ad esempio, se in un menù a tendina bisogna selezionare un tipo di asset, senza le LOV verranno visualizzati gli ID corrispondenti all'asset, ma un operatore non sarà a conoscenza di quale asset si tratta. Grazie alle LOV è possibile mostrare a video informazioni utili, come categoria, marca, modello e seriale, così da distinguerli facilmente.

I Popup LOV sono un tipo di componente che consente agli sviluppatori di creare una finestra di dialogo pop-up che permette agli utenti di selezionare un valore da un elenco predefinito sfruttando le LOV. Questo è utile quando si desidera fornire all' utente finale un modo rapido e intuitivo per selezionare un valore da un elenco piuttosto che digitare manualmente i dati, ma soprattutto consentono all'utente di visualizzare dati consistenti invece di dover selezionare degli identificatori senza sapere di cosa si tratta. Per la quasi totalità dei casi in questa applicazione le Popup LOV sono di tipo dinamico ed hanno una Source che deriva o da tabelle o da query SQL.

4.1.3 Chart

I chart sono rappresentazioni grafiche dei dati che consentono di visualizzare informazioni in modo più comprensibile e interattivo. Con Oracle APEX c'è la possibilità di creare grafici di diversa natura, tra cui barre, linee, torte, scatter, radar, e molti altri, a seconda delle proprie esigenze. È possibile popolare i grafici con dati, statici o dinamici, provenienti da diverse fonti, tra cui tabelle del database, query SQL e servizi web REST. Per la creazione dei grafici si può utilizzare il Page Builder per aggiungere un componente grafico alla pagina e quindi configurarlo utilizzando le opzioni disponibili, ad esempio personalizzare gli assi X e Y, i colori, le etichette ed altri aspetti. Una delle funzionalità più importanti, che verranno sfruttate nell'implementazione, è che i grafici possono essere

aggiornati in modo dinamico in risposta a eventi o filtri applicati dall'operatore, ciò significa che i grafici possono riflettere immediatamente le modifiche nei dati o nelle selezioni fatte da chi utilizza l'app senza dover ricaricare l'intera pagina [4].

4.2 Implementazione Database

Dopo aver svolto l'attività di progettazione, si è passati all'implementazione del database. Come anticipato durante la progettazione del database, sono state create prima le tabelle Locazione e Fornitore per i motivi precedentemente discussi. Successivamente però si è usato lo strumento del Quick SQL che ha permesso di risparmiare tempo prezioso. Di seguito, nella *figura 10,* il codice Quick SQL utilizzato per la creazione della base di dati.

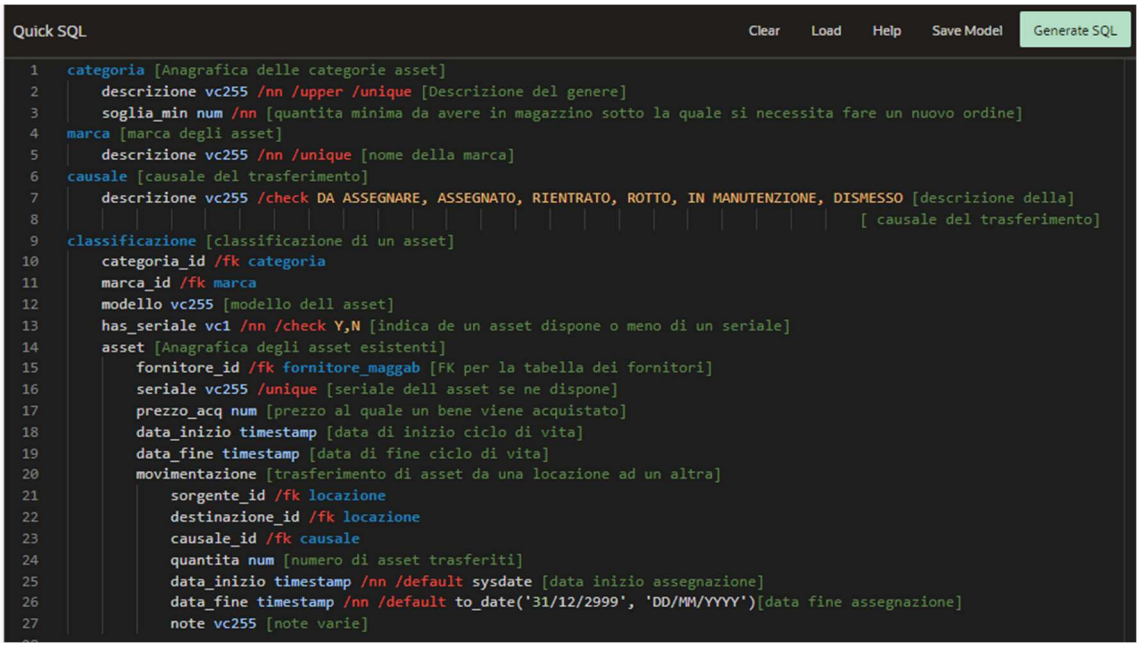


Figura 10: Quick SQL per la base di dati

Dopo aver impostato nei Settings la generazione automatica degli audit per il *created by* e per il *updated by,* con le relative date, per ogni tabella, cliccando il tasto "Generate SQL" il risultato sarà quello mostrato nella *figura 11,* nella *figura 12* e nella *figura 13* (nella *figura 13* viene mostrato soltanto un esempio inerente alla tabella categoria, per le altre tabelle i trigger sono identici).

```sql
-- create tables
create table asset.am_categoria (
    id                          number generated by default on null as identity
                                constraint am_asset_am_categoria_id_pk primary key,
    descrizione                 varchar2(255 char)
                                constraint am_asset_am_catego_descriz_unq unique not null,
    soglia_min                  number not null,
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- comments
comment on table asset.am_categoria is 'Anagrafica delle categorie asset';
comment on column asset.am_categoria.descrizione is 'Descrizione del genere';
comment on column asset.am_categoria.soglia_min is 'quantita minima da avere in magazzino sotto la quale si necessita fare un nuovo ordine';

create table asset.am_marca (
    id                          number generated by default on null as identity
                                constraint am_asset_am_marca_id_pk primary key,
    descrizione                 varchar2(255 char)
                                constraint am_asset_am_marca_descrizi_unq unique not null,
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- comments
comment on table asset.am_marca is 'marca degli asset';
comment on column asset.am_marca.descrizione is 'nome della marca';

create table asset.am_causale (
    id                          number generated by default on null as identity
                                constraint am_asset_am_causale_id_pk primary key,
    descrizione                 varchar2(15 char) constraint am_causale_descrizione_ck
                                check (descrizione in ('DA ASSEGNARE','ASSEGNATO','RIENTRATO','ROTTO','IN MANUTENZIONE','DISMESSO')),
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- comments
comment on table asset.am_causale is 'causale del trasferimento';
comment on column asset.am_causale.descrizione is 'descrizione della causale del trasferimento';

create table asset.am_classificazione (
    id                          number generated by default on null as identity
                                constraint am_asset_am_classific_id_pk primary key,
    categoria_id                number
                                constraint am_asset_am_classi_categori_fk
                                references asset.am_categoria,
    marca_id                    number
                                constraint am_asset_am_classific_marca_fk
                                references asset.am_marca,
    modello                     varchar2(255 char),
    has_seriale                 varchar2(1 char) constraint am_classificazi_has_seriale_ck
                                check (has_seriale in ('Y','N')) not null,
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- table index
create index am_asset_am_classific_i1 on asset.am_classificazione (categoria_id);
create index am_asset_am_classific_i112 on asset.am_classificazione (marca_id);
```

Figura 11: Parte I del codice SQL

```
-- comments
comment on table asset.am_classificazione is 'classificazione di un asset';
comment on column asset.am_classificazione.has_seriale is 'indica de un asset dispone o meno di un seriale';
comment on column asset.am_classificazione.modello is 'modello dell asset';

create table asset.am_asset (
    id                          number generated by default on null as identity
                                constraint am_asset_am_asset_id_pk primary key,
    classificazione_id          number
                                constraint am_asset_am_asse_classifica_fk
                                references asset.am_classificazione,
    fornitore_id                number
                                constraint am_asset_am_asset_fornitore_fk
                                references asset.am_fornitore_maggab,
    seriale                     varchar2(255 char)
                                constraint am_asset_am_asset_seriale_unq unique,
    quantita                    number,
    prezzo_acq                  number,
    data_inizio                 timestamp,
    data_fine                   timestamp,
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- table index
create index am_asset_am_asset_i1 on asset.am_asset (classificazione_id);
create index am_asset_am_asset_i172 on asset.am_asset (fornitore_id);

-- comments
comment on table asset.am_asset is 'Anagrafica degli asset esistenti';
comment on column asset.am_asset.data_fine is 'data di fine ciclo di vita';
comment on column asset.am_asset.data_inizio is 'data di inizio ciclo di vita';
comment on column asset.am_asset.fornitore_id is 'FK per la tabella dei fornitori';
comment on column asset.am_asset.prezzo_acq is 'prezzo al quale un bene viene acquistato';
comment on column asset.am_asset.seriale is 'seriale dell asset se ne dispone';

create table asset.am_movimentazione (
    id                          number generated by default on null as identity
                                constraint am_asset_am_movimenta_id_pk primary key,
    asset_id                    number
                                constraint am_asset_am_movimenta_asset_fk
                                references asset.am_asset,
    causale_id                  number
                                constraint am_asset_am_movime_causale_fk
                                references asset.am_causale,
    sorgente_id                 number
                                constraint am_asset_am_movime_sorgente_fk
                                references asset.am_locazione,
    destinazione_id             number
                                constraint am_asset_am_movi_destinazio_fk
                                references asset.am_locazione,
    quantita                    number,
    data_inizio                 timestamp default SYSDATE not null,
    data_fine                   timestamp default to_timestamp('31/12/2999', 'DD/MM/YYYY') not null,
    note                        varchar2(255 char),
    created                     timestamp not null,
    created_by                  varchar2(255 char) not null,
    updated                     timestamp not null,
    updated_by                  varchar2(255 char) not null
)
;

-- table index
create index am_asset_am_movimenta_i1 on asset.am_movimentazione (asset_id);
create index am_asset_am_movimenta_i242 on asset.am_movimentazione (causale_id);
create index am_asset_am_movimenta_i253 on asset.am_movimentazione (destinazione_id);
create index am_asset_am_movimenta_i264 on asset.am_movimentazione (sorgente_id);

-- comments
comment on table asset.am_movimentazione is 'trasferimento di asset da una locazione ad un altra';
comment on column asset.am_movimentazione.data_fine is 'data fine assegnazione';
comment on column asset.am_movimentazione.data_inizio is 'data inizio assegnazione';
comment on column asset.am_movimentazione.note is 'note varie';
comment on column asset.am_movimentazione.quantita is 'numero di asset trasferiti';
```

Figura 12: Parte II del codice SQL

```
-- triggers
create or replace trigger asset.am_categoria_biu
    before insert or update
    on asset.am_categoria
    for each row
begin
    if inserting then
        :new.created := systimestamp;
        :new.created_by := coalesce(sys_context('APEX$SESSION','APP_USER'),user);
    end if;
    :new.updated := systimestamp;
    :new.updated_by := coalesce(sys_context('APEX$SESSION','APP_USER'),user);
    :new.descrizione := upper(:new.descrizione);
end am_categoria_biu;
/
```

Figura 13: Esempio di trigger per gli audit sulla tabella Categoria

4.3 Struttura delle pagine di report

4.3.1 Configurazioni

Per questioni di semplicità e utilità, il primo approccio con l'App Builder di APEX è iniziato con la creazione delle pagine di configurazione precedentemente anticipate. Le pagine che compongono il sottomenù "configurazioni" del Navigation Menu sono Categoria, Marca, Classificazione, Stati Asset, Locazioni e Fornitori. Ognuna di queste è una pagina di report generata indicando nella Source del Page Designer la tabella a cui il report fa riferimento. L'unica pagina tra queste che non prende la tabella originale è la pagina Locazioni che, come anticipato nel capitolo 3.1.3, si appoggia su una query, mostrata in *figura 14* (per motivi di privacy alcuni dati sono stati mascherati). Quest'ultima sfruttando la UNION unisce i dati della tabella creata appositamente per la base di dati della web app a quelli presenti nella vista creata su una tabella già esistente all'interno dell'azienda, così da tenere sempre aggiornati i dati delle locazioni ogni volta

che la tabella originale viene aggiornata. La tabella originale però non conteneva i dati inerenti alle direzioni e ai magazzini. Questo giustifica la scelta di creare una tabella apposita per l'app e non usare direttamente quella già presente, così da poter aggiungere nuove locazioni all'occorrenza, senza modificare obbligatoriamente la tabella principale.

```
SELECT COD,
       CITTA,
       CAP,
       INDIRIZZO,
       RAG_SOCIALE,
       TIPO
  FROM AM_LOCAZIONE
UNION
SELECT [redacted],
       [redacted],
       [redacted],
       [redacted],
       [redacted],
       'NEGOZIO'
  FROM [redacted] A
 WHERE TRIM (A.[redacted]) IS NULL AND A.[redacted] = 0;
```

Figura 14: Codice SQL per la Source della pagina Locazioni

4.3.2 Creazione Asset

Questa pagina è dedicata all'immissione nel sistema dei nuovi asset acquistati. Come già anticipato, questa pagina è stata concepita con l'idea che ogni riga di questo report è una riga di una bolla di entrata dei nuovi asset acquistati, così da inizializzare ogni nuovo materiale. Il Source del report di questa pagina è dato non da una tabella ma da un codice SQL poiché si voleva mostrare tutte le informazioni riguardo quell'asset;

quindi, c'era bisogno di far visualizzare tutto il dato complessivo riguardante la classificazione di esso, quindi la categoria, la marca e il modello. Il codice è mostrato in *figura 15.*

```
select ass.ID,
       cla.categoria_id,
       cla.marca_id,
       cla.modello,
       ass.FORNITORE_ID,
       ass.QUANTITA,
       ass.SERIALE,
       ass.PREZZO_ACQ,
       ass.DATA_INIZIO,
       ass.DATA_FINE,
       ass.CREATED,
       ass.CREATED_BY,
       ass.UPDATED,
       ass.UPDATED_BY
  from AM_ASSET ass
  join am_classificazione cla on ass.classificazione_id = cla.id
order by ass.DATA_INIZIO desc
```

Figura 15: Codice SQL del report Creazione Asset

Come è possibile vedere la tabella finale è data dal JOIN tra le tabelle AM_ASSET e AM_CLASSIFICAZIONE.

4.3.3 Movimentazioni

Questa pagina è concepita per soddisfare il bisogno principale che ha spinto l'azienda a sviluppare la web app, ovvero tener traccia di ogni spostamento di ciascun asset dall'inizio del suo ciclo di vita fino alla fine. La soluzione più ovvia era creare un report che mostrasse tutti i dati necessari a rendere l'informazione utile in cui si potessero effettuare ricerche per diversi scopi, ad esempio monitorare il ciclo di vita di un asset, quindi avere un tracciamento di tutte le movimentazioni, oppure monitorare, ricercando per locazioni, il traffico di asset tra un magazzino e un punto di vendita o una direzione. Queste informazioni che serviranno a capire come gli asset vengono gestiti e mantenuti nelle varie locazioni, riuscendo a capire se le eventuali rotture sono causate da malfunzionamenti derivanti da difetti di fabbrica di alcuni modelli (e quindi ricorrenti) o se sono causate dalla pessima gestione del materiale dato in affido. Ad esempio, se uno specifico modello di un PC risulta avere un ciclo di vita medio di 8 anni e un punto di vendita riconsegna lo stesso PC all'azienda ogni 2 anni significa che il PC non è stato utilizzato nel modo corretto e quindi avere la capacità di intervenire per capire quali cattive usanze causano le rotture degli asset in quella locazione, istruendo in modo specifico gli operatori per far si che non si ripetano gli stessi errori. Questa possibilità, avendo molto materiale distribuito in tutti i punti vendita della nazione, consentirà di ridurre le manutenzioni straordinarie portando l'azienda a risparmiare sui costi.

Il Source per il report di questa pagina è dato da un JOIN tra le tabelle AM_MOVIMENTAZIONE e AM_ASSET come è possibile vedere in *figura 16.*

```sql
select mov.ID,
       mov.ASSET_ID,
       ass.classificazione_id,
       mov.CAUSALE_ID,
       mov.SORGENTE_ID,
       mov.DESTINAZIONE_ID,
       mov.QUANTITA,
       mov.DATA_INIZIO,
       mov.DATA_FINE,
       mov.NOTE,
       mov.CREATED,
       mov.CREATED_BY,
       mov.UPDATED,
       mov.UPDATED_BY
  from AM_MOVIMENTAZIONE mov
  join am_asset ass on ass.id = mov.asset_id
```

Figura 16: Codice SQL del report Movimentazioni

4.3.4 Magazzino

Questa pagina è stata sviluppata poiché era un altro dei requisiti fondamentali che la web app dovesse avere, ovvero un rapido collegamento ad un report che mostrasse in tempo reale la disponibilità dei materiali in magazzino. Un report anche qui era una soluzione facile ed intuitiva per la velocizzazione del lavoro. Dato che al momento i magazzini che contengono asset IT sono 2, il report deve visualizzare i dati o di uno o dell'altro. Per questo motivo è stato aggiunto un menù a tendina che consente la selezione tra i magazzini, sfruttando la funzionalità dei Popup LOV e mostrando quindi i nomi dei magazzini ma ritornando il valore della chiave primaria COD.

In questo caso la query che è stata inserita nel Source per la generazione di questo report è leggermente più strutturata, vedi *figura 17,* poiché vengono effettuate operazioni principalmente nella tabella delle movimentazioni non essendoci una tabella vera e propria che contenesse la disponibilità nel magazzino.

```
select tab_qta_in.descrizione, tab_qta_in.marca, tab_qta_in.modello,
        sum(coalesce(tab_qta_in.qta_in - tab_qta_out.qta_out, tab_qta_in.qta_in)) as Quantità
from
(select ass.id, cat.descrizione, m.descrizione as Marca, cla.modello, sum(mov.quantita) as qta_in
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
inner join am_marca m on m.id = cla.marca_id
where mov.causale_id = 1 and mov.destinazione_id = :SELECT_MAGAZZINO
group by ass.id, cat.descrizione, m.descrizione, cla.modello) tab_qta_in
left join
(select ass.id, cat.descrizione, sum(mov.quantita) as qta_out
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id != 1 and mov.sorgente_id = :SELECT_MAGAZZINO
group by ass.id, cat.descrizione) tab_qta_out
on tab_qta_in.id = tab_qta_out.id
group by tab_qta_in.descrizione, tab_qta_in.marca, tab_qta_in.modello
```

Figura 17: Codice SQL del report Magazzino

La logica dietro questa query consiste nel sommare tutte le quantità entranti nel magazzino dall'inizio del suo ciclo di vita fino alla sua ultima movimentazione e sottrarre a questa quantità la somma di tutte le quantità che risultano essere uscite dal magazzino nello stesso arco temporale. Così facendo è possibile ricavare il numero di asset presenti attualmente nel magazzino. Viene effettuato quindi un LEFT JOIN tra la tabella nominata tab_qta_in e la tabella tab_qta_out nelle quali è stata effettuata la somma delle quantità rispettivamente in entrata e in uscita, e da questa tabella si ricava la sottrazione tra le due quantità facendo un GROUP BY per categoria, marca e modello. Nella clausola WHERE di entrambe le tabelle tab_qta_in e tab_qta_out compare la variabile SELECT_MAGAZZINO, che avrà il valore corrispondente alla selezione del Popup LOV presente nella pagina.

4.3.5 Home Page

Essendo la home page di una web app di livello gestionale, non deve mostrare informazioni e panoramiche di livello pubblicitario, ma è stato invece scelto un approccio basato sull'utilità. I primi componenti che si possono incontrare sono infatti due bottoni che fungono da shortcuts per arrivare alla pagina degli asset e alla pagina delle movimentazioni, utili a velocizzare il lavoro degli operatori. Successivamente si possono trovare due sezioni.

La prima sezione è dedicata ai grafici per monitorare gli asset presenti nei magazzini, con un Popup LOV che permette di scegliere di quale magazzino mostrare i dati e poi tre grafici, uno che mostra la giacenza contabile del magazzino selezionato (vedi *figura 18*), e quindi che mostra il valore economico disponibile in magazzino diviso per categoria, il secondo che espone la giacenza fisica sempre dello stesso magazzino (vedi *figura 19*) e l'ultimo che mostra gli asset sotto scorta (vedi *figura 21)*.

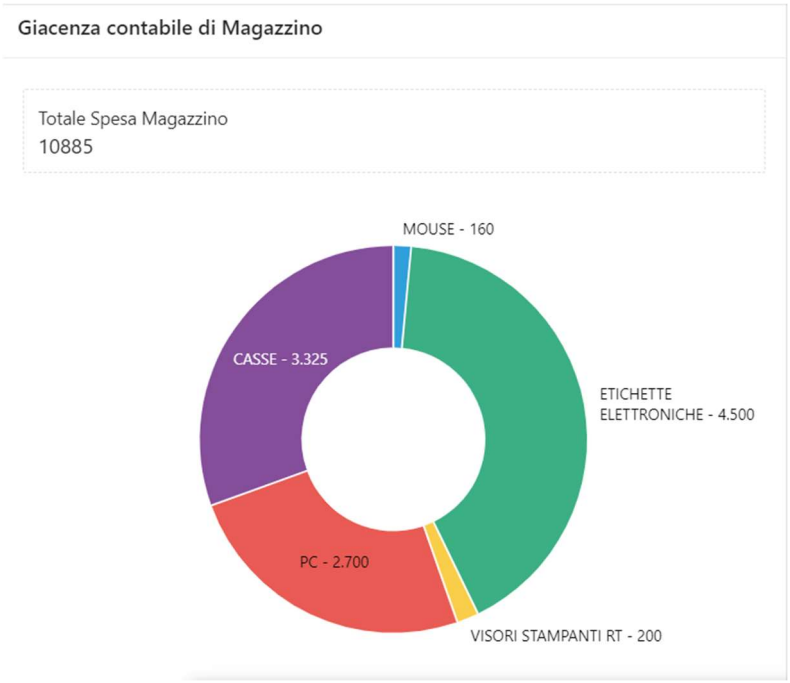


Figura 18: Grafico giacenza contabile di un magazzino

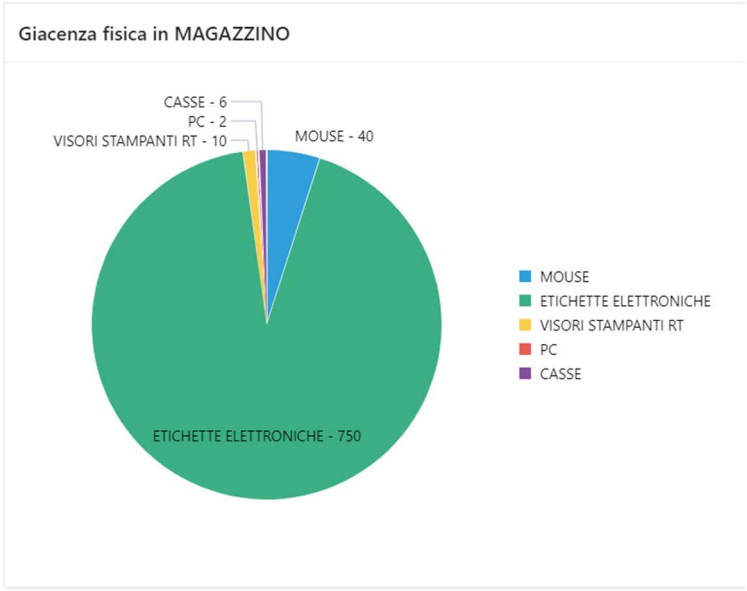


Figura 19: Grafico giacenza fisica di un magazzino

I grafici mostrati in *figura 18* e *figura 19* vanno di pari passo poiché il primo grafico prende le quantità che compaiono nel secondo essendo le quantità degli asset divisi per categoria, ma nel primo questi dati vengono moltiplicati per il prezzo unitario di ogni asset. Per fare una prova, si prendano in considerazione i PC. Nella *figura 20* che segue, viene effettuata una ricerca dei PC nel report delle movimentazioni; quindi, avremo come risultato tutte le movimentazioni che coinvolgono i PC:

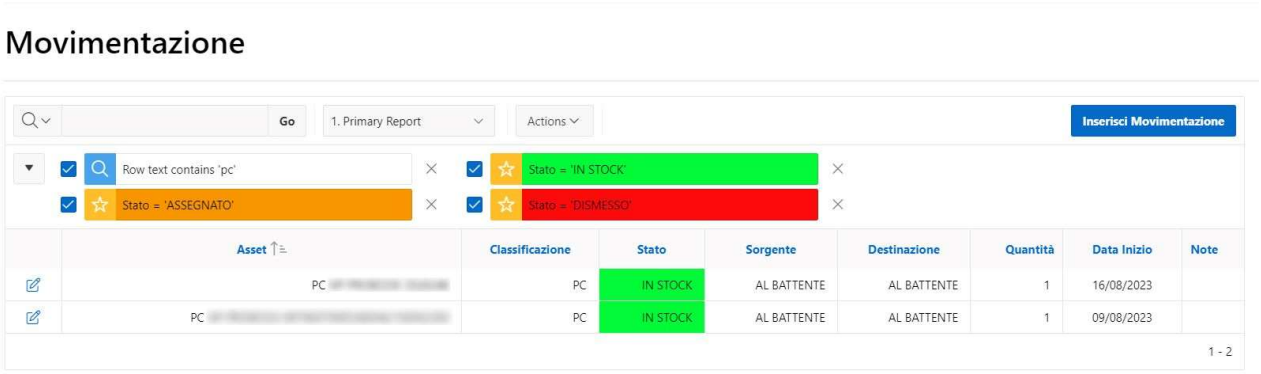


Figura 20: Esempio report Movimentazione con ricerca "PC"

È possibile notare che entrambi i PC sono stati soltanto immagazzinati (quando sia la sorgente che la destinazione hanno come informazione lo stesso magazzino significa che è stato immagazzinato per la prima volta) e non c'è nessuna movimentazione in uscita. Evidentemente i PC nella locazione Al Battente, ovvero in nome assegnato al magazzino interno al reparto DSI, saranno due, come mostrato nel grafico in *figura 19*. Quando i due asset sono stati definiti è stato inserito il costo unitario di ognuno dei due PC, rispettivamente 1500€ e 1200€. Effettuando la somma dei due il risultato sarà di 2700€, ovvero la cifra riportata nello spicchio inerente ai PC nel primo grafico in *figura 18.* Sempre per quanto riguarda i PC, avendo istanziato che la soglia minima per la categoria

PC è 15, e ricordando che i PC in magazzino sono due è intuitivo che per quanto riguarda questa categoria si è sotto scorta; infatti, come è possibile vedere nella seguente *figura 21* compare il dato inerente ad essi, mostrando il numero effettivo di PC attualmente in magazzino:

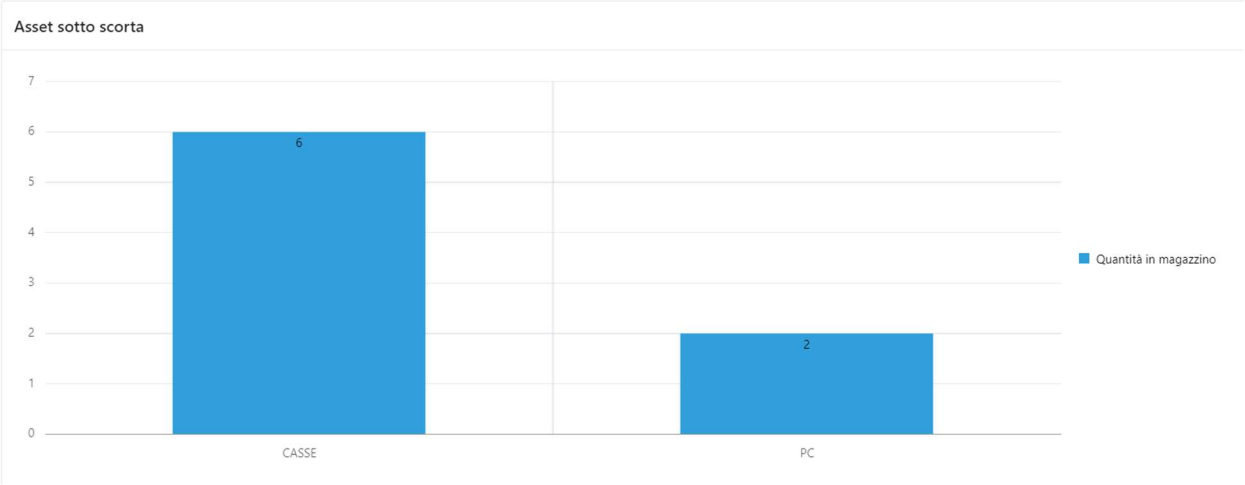


Figura 21: Grafico degli asset sotto scorta in un magazzino

Per quanto riguarda la logica dietro a questa sezione riguardante i dati di un magazzino, quando viene selezionato un magazzino di una Select List (un componente simile ad un Popup LOV ma appare come una semplice lista, di solito utilizzato se gli elementi della lista sono meno di 10), vengono innescate tre Dinamic Action (azioni dinamiche che da un evento fanno scaturire un'azione), una per grafico, che vanno ad aggiornare tramite un refresh soltanto le zone della pagina che contengono i grafici, evitando di fare il refresh totale della pagina.

Il grafico in *figura 18* prende i dati da una query simile a quella utilizzata per il report del magazzino tranne per il fatto che invece di sommare soltanto le quantità viene effettuata

la somma del prodotto tra quantità e costo unitario, così da avere una quantità che rappresenta il valore economico di ciò che si ha nel magazzino (vedi *figura 22*). Immediatamente sopra al grafico è stato messo un campo in cui è possibile vedere il valore totale della giacenza in magazzino facendo semplicemente la somma dei valori di tutte le categorie.

```
select  tot_spesa_in_stock.descrizione, sum(coalesce(tot_spesa_in_stock.spesa_in - tot_spesa_out.spesa_out,
        tot_spesa_in_stock.spesa_in)) as Spesa_del_Magazzino

from

(select ass.id, cat.descrizione, sum(ass.prezzo_acq * mov.quantita) as Spesa_in
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id = 1 and mov.destinazione_id = :SELECT_MAGAZZINO_GEN
group by ass.id, cat.descrizione) tot_spesa_in_stock

left join

(select ass.id, cat.descrizione, sum(ass.prezzo_acq * mov.quantita) as Spesa_out
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id in (2,3,4,5) and mov.sorgente_id = :SELECT_MAGAZZINO_GEN
group by ass.id, cat.descrizione) tot_spesa_out

on tot_spesa_in_stock.id = tot_spesa_out.id

group by tot_spesa_in_stock.descrizione
```

Figura 22: Codice SQL usato come Source del grafico per la giacenza contabile in un magazzino

Il grafico rappresentato in *figura 19* invece, dovendo mostrare il numero di asset divisi per categoria, prende i dati da una query identica a quella utilizzata per generare il report del magazzino (vedi *figura 17*).

Il grafico in *figura 21*, l'ultimo di questa sezione dedicata ai magazzini, serve a visualizzare gli asset che sono sotto scorta e in che numero sono presenti nel magazzino. La logica dietro questa query è la stessa usata per ricavare le quantità del magazzino soltanto che vengono presi solo i record in cui questa quantità è minore della soglia minima definita a livello di categoria (*figura 23*)

```
select x.descr as Descrizione, x.soglia as Soglia_Minima, x.qta_mag as Quantità_in_Magazzino

from

(select  tab_qta_in.descrizione as descr, tab_qta_in.soglia_min as soglia,
        sum(coalesce(tab_qta_in.qta_in - tab_qta_out.qta_out, tab_qta_in.qta_in)) as qta_mag

from

(select ass.id, cat.descrizione, cat.soglia_min , sum(mov.quantita) as qta_in
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id = 1 and mov.destinazione_id = :SELECT_MAGAZZINO_GEN
group by ass.id, cat.descrizione, cat.soglia_min) tab_qta_in

left join

(select ass.id, cat.descrizione, sum(mov.quantita) as qta_out
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id in (2,3,4,5) and mov.sorgente_id = :SELECT_MAGAZZINO_GEN
group by ass.id, cat.descrizione) tab_qta_out

on tab_qta_in.id = tab_qta_out.id

group by tab_qta_in.descrizione, tab_qta_in.soglia_min) x

where qta_mag < soglia
```

Figura 23: Codice SQL usato come Source del grafico degli asset sotto scorta

Nella seconda sezione sono stati posti tre elementi: un Popup LOV dove vengono mostrate tutte le locazioni tranne i magazzini, visualizzando la loro ragione sociale così da poterle distinguere e cercare facilmente (*figura 24*), un campo dove è possibile

visualizzare l'ammontare del valore economico degli asset in possesso della specifica locazione e un grafico, che mostra sempre in valore economico come è distribuita la merce a livello di categoria, come già fatto per i magazzini.

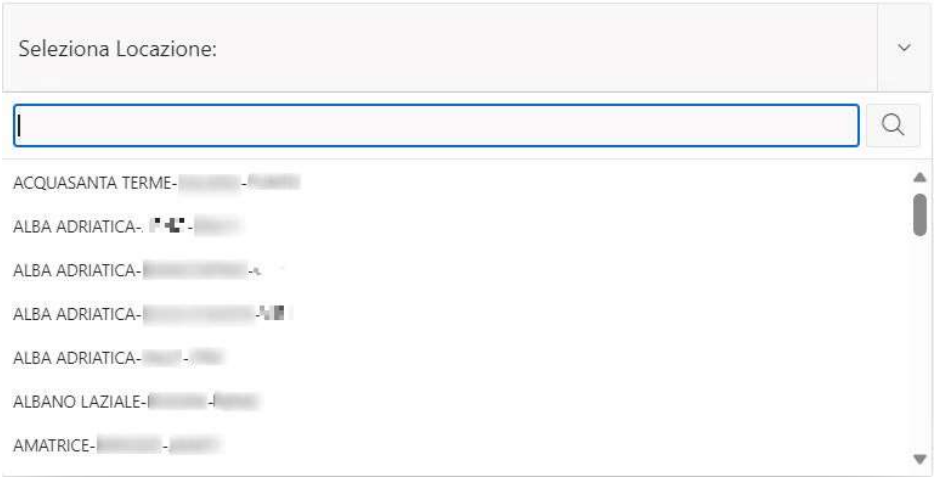


Figura 24: Popup LOV per la selezione della locazione

Una volta selezionato un elemento, il suo valore, ovvero l'identificativo, verrà immesso nella variabile SELECT_LOCAZIONE, che verrà poi utilizzata nella query usata nel Source del grafico per la giacenza contabile nella locazione selezionata. La query è pressoché identica a quella utilizzata per il grafico per la giacenza contabile di un magazzino, eccezion fatta per la variabile a cui fa riferimento (vedi *figura 25*), mentre il grafico è generato dallo stesso modo.

```sql
select  tot_spesa_in_stock.descrizione, coalesce(tot_spesa_in_stock.spesa_in - tot_spesa_out.spesa_out,
        tot_spesa_in_stock.spesa_in) as Spesa_del_Magazzino

from

(select ass.id, cat.descrizione, sum(ass.prezzo_acq * mov.quantita) as Spesa_in
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id = 2 and mov.destinazione_id = :SELECT_LOCAZIONE
group by ass.id, cat.descrizione, mov.sorgente_id) tot_spesa_in_stock

left join

(select ass.id, cat.descrizione, sum(ass.prezzo_acq * mov.quantita) as Spesa_out
from am_movimentazione mov
inner join am_asset ass on mov.asset_id = ass.id
inner join am_classificazione cla on cla.id = ass.classificazione_id
inner join am_categoria cat on cat.id = cla.categoria_id
where mov.causale_id in (1,3,4,5) and mov.sorgente_id = :SELECT_LOCAZIONE
group by ass.id, cat.descrizione) tot_spesa_out

on tot_spesa_in_stock.id = tot_spesa_out.id
```

Figura 25: Codice SQL usato come Source del grafico della giacenza contabile in una locazione

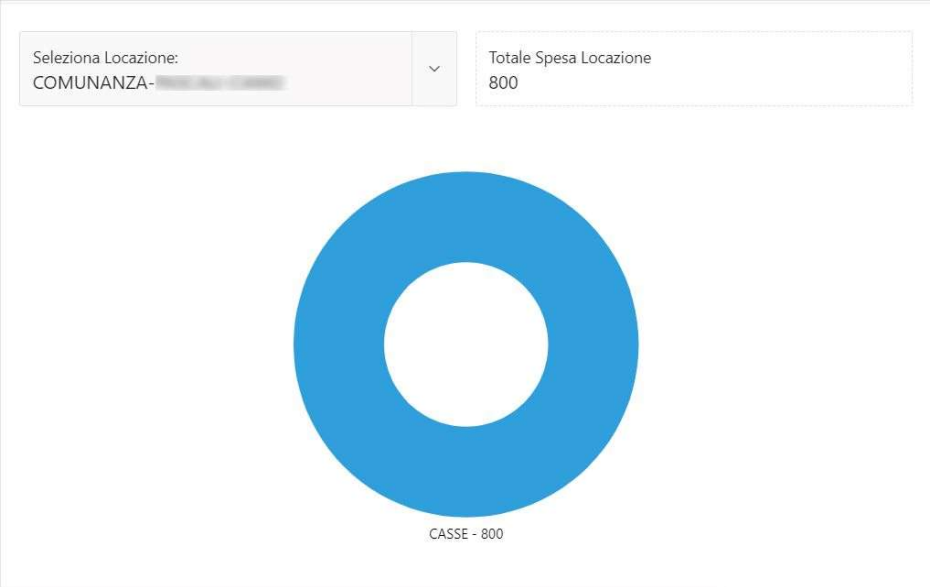


Figura 26: Sezione dedicata alla giacenza contabile in una locazione

Nel semplice esempio in *figura 26* è riportato il grafico inerente ad una specifica locazione. Si può notare che il totale del valore economico degli asset in quel punto di vendita ammonta a 800€ poiché dalle movimentazioni è possibile vedere come effettivamente l'unica movimentazione che la coinvolge riguarda un'entrata di una cassa dal valore di 800€.

Conclusioni

In questo elaborato sono stati presentati tutti i passaggi che occorrono per passare da un'esigenza ad uno strumento in grado di soddisfarla, che in questo caso è stata una web app. Si è partiti con una prima fase di analisi dei requisiti, che è stata utile per avere sempre a portata di mano un punto della situazione ed una scaletta da seguire e rispettare. Prima di passare alle fasi di progettazione e implementazione si è fatto il punto su cosa fosse Oracle APEX, motivando il perché la scelta è ricaduta sull'utilizzo di questo strumento per lo sviluppo della web app in questione e accennando il low-code spiegando l'utilità di un approccio di questo tipo in alcuni ambiti. A questo punto è iniziata la fase di progettazione, sia del database che della web app. Sicuramente la progettazione del database è stata più complessa rispetto a quella della web app date le caratteristiche dello strumento utilizzato. Infine, come ultima fase è stata sviluppata l'implementazione della web app, e quindi anche qui sia la parte del database sia la parte dell'app, spiegando il più possibile come sono state sfruttate le funzionalità di Oracle APEX, che hanno garantito semplicità e velocità nello sviluppo di essa.

A conclusione del lavoro, si può dire che il prodotto finale soddisfa tutti i requisiti, sia funzionali che non funzionali. Sicuramente sarà possibile effettuare qualche miglioramento in futuro, d'altronde ogni prodotto sul mercato viene aggiornato a nuove versioni in continuazione per migliorare sempre di più le performance che derivano dal suo utilizzo, ed anche questo non sarà da meno, andando ad esempio a velocizzare il tempo di configurazione degli asset, risparmiando ulteriore tempo. Per quanto riguarda la tecnologia utilizzata bisogna dire che, come detto anche in altri punti dell'elaborato, il low-code non deve essere visto come un qualcosa che vada contro gli sviluppatori ma

come uno strumento utile, un'arma in più per riuscire a diminuire i tempi di sviluppo e quindi produrre di più, traendone un vantaggio. A livello personale l'esperienza che mi è stata offerta ha arricchito il mio bagaglio personale, non solo perché mi è stata data l'opportunità lavorare con uno strumento di cui non ero a conoscenza, ma anche perché mi ha aiutato ad entrare in diversi meccanismi aziendali che all'interno delle facoltà universitarie non è facile cogliere.

SITOGRAFIA

[1] «Oracle Apex» [Online]. Available: https://apex.oracle.com/it/

[2] «Wikipedia» [Online]. Available: https://it.wikipedia.org/wiki/Progressive_Web_App

[3] «IBM» [Online]. Available: https://www.ibm.com/it-it/topics/low-code

[4] «Oracle APEX Documentation» [Online]. Available: https://docs.oracle.com/en/database/oracle/apex

[5] «Nexsys» [Online]. Available: https://www.nexsys.it/high-availability-vs-sistemi-tradizionali/

[6] «Wikipedia» [Online]. Available: https://it.wikipedia.org/wiki/Disaster_recoverytopics/low-code

[7] «QRP International» [Online]. Available: https://www.qrpinternational.it/blog/faq/che-cose-un-backlog/

[8] «Wikipedia» [Online]. Available: https://it.wikipedia.org/wiki/Shadow_IT