# Progetto Ingegneria Cauzzi Marco

# Del Software

# Magazzini Catena Negozi Sportivi

# VR376899

Descrizione Progetto

Il progetto consiste nello sviluppo di un prototipo per la gestione dei magazzini di una catena sportiva.

Sono stati individuati tre attori principali:

* **Il Magazziniere:** Ha il compito di gestire le entrate ed uscite dal magazzino e di muovere gli articoli ricevuti in modo da ottimizzare l’occupazione di spazio.
* **Il Gestore del negozio:** ordina gli articoli necessari allo svolgimento dell’attività del proprio negozio.
* **La Segreteria amministrativa:** Inserisce i tipi di articoli sportivi venduti dalla catena e può visualizzare i movimenti di essi tra negozio e magazzino.

Vi sono poi due entità principali:

* **L’Applicazione:** Permette ai tre attori individuati d’interagire col database in maniera corretta.
* **Il Database:** Che contiene tutti i dati necessari alla corretta gestione dei magazzini.

Per lo sviluppo dell’applicazione si è scelto di ricorrere al linguaggio di programmazione Java e di usare le sue rispettive librerie per l’UI Swing.

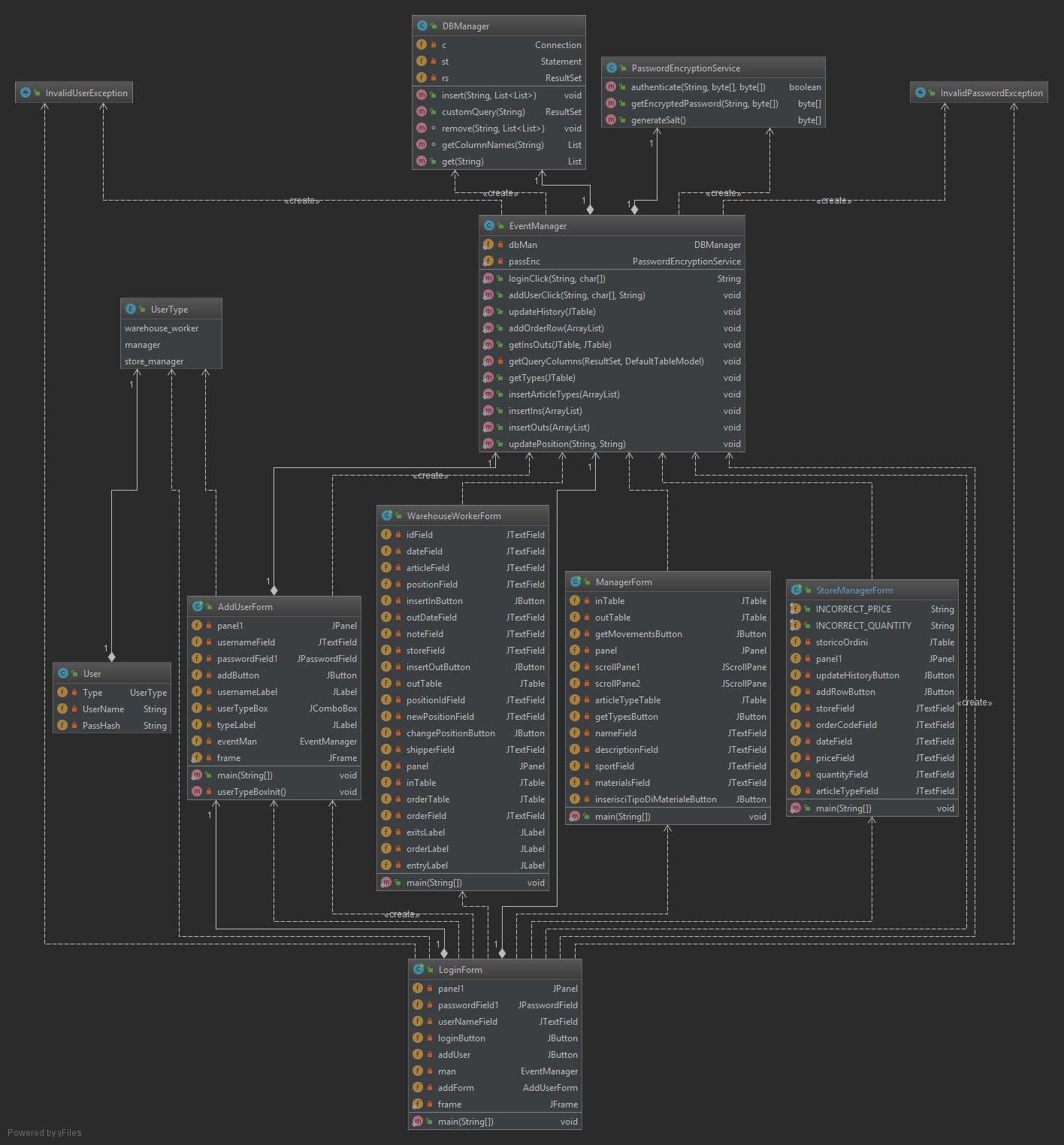
Per la scrittura del codice si è scelto di usare l’IDE IntelliJ Idea, in quanto si sono voluti provare IDE alternativi ad Eclipse.

La scelta è stata confermata dopo aver notato le notevoli migliorie nel autocompletamento d’IntelliJ rispetto ad Eclipse e notevole diminuzione dei tempi di risposta generali.

Come DBMS si è optato per PostgreSQL abbinato alla piattaforma di gestione pgAdmin, in quanto gia testato in UNI, e quindi la sua sintassi è familiare, e perché è gratis.

Per testare il programma si è scelto un approccio iterativo, ovvero, dopo la scrittura di ogni singola form, essa veniva provata per individuarne eventuali mancanze rispetto al documento di definizione e per rilevare eventuali bachi nelle funzionalità.

Struttura Applicazione



Abbiamo un totale di 8 classi cosi suddivise:

* 5 Form + le loro relative classi di gestione, corrispondenti ai tre tipi di utente + una di login e una di creazione utente, non richiesta nel documento di design.
* 1 classe di gestione degli eventi, che s’interfaccia col gestore del DB e il servizio di gestione del login
* 1 classe che gestisce il DB, che permette all’applicazione di inviare query al Database
* 1 classe di gestione del login, che verifica i dati di login e, nel caso di creazione di un nuovo utente, aggiunge il salt alla pass e crea l’hash che verrà poi inserito nel DB

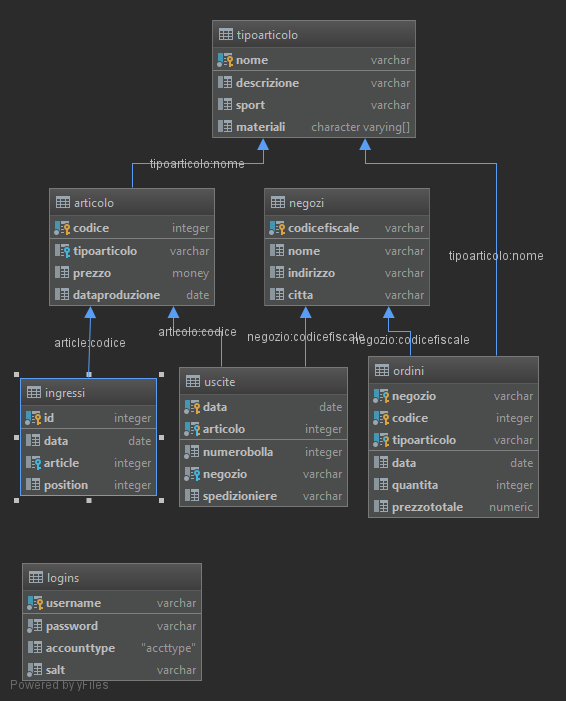
## Form e relative classi di gestione degli eventi

Le 5 form sono definite nel designer d’IntelliJ trascinando i vari controlli in un’interfaccia molto simile a quella presente in altri IDE come Eclipse o Visual studio, e come questi la rappresentazione visiva verrà trasformata in file .from che è praticamente un file -xml che contiene tutti i controlli definiti e le loro proprietà.

A questi file sono associati dei file sorgente Java i quali contengono la funzione d’inizializzazione della form

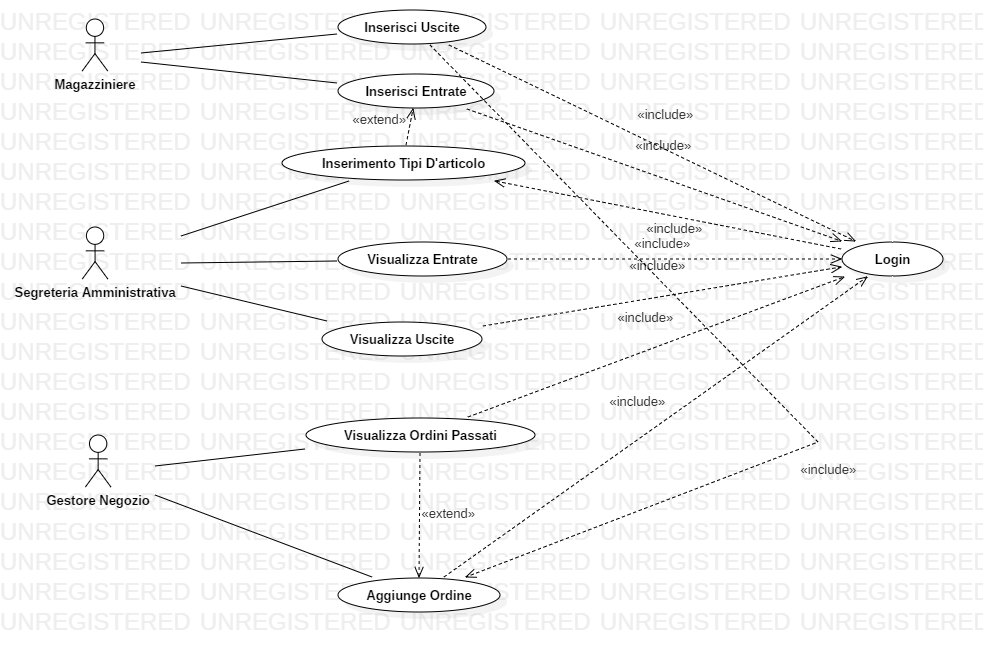
(dove vengono settate alcune proprietà di essa come la grandezza e viene caricata) e delle funzioni che sono “iscritte” ad alcuni eventi (come per esempio il click del mouse) associati ai controlli della form.

Struttura Database



Vi sono 7 tabelle nel DB, 6 relative alla gestione dei dati del magazzino(+ una vista) e una per gestire i login all’applicazione.

* **TipoArticolo**: Contiene informazioni per ogni tipo di articolo venduto dalla catena
* **Articolo**: Contiene informazioni sul singolo articolo presente in magazzino
* **Ingressi**: Contiene informazioni sugli articoli arrivati in magazzino.
* **Negozi**: Contiene informazioni sui singoli articoli della catena
* **Ordini**: Contiene informazioni sui singoli ordini effettuati dai negozi della catena
* **Uscite**: Contiene informazioni sulle uscite dal magazzino ai negozi

CASI D’USO

**Caso d’uso**: Inserisci Entrata

**Precondizioni:** L’utente deve aver fatto il Login da Magazziniere

**Sequenza degli eventi**:

1. Il magazziniere deve inserire i dettagli dell’Entrata(ID, Data, Articolo ed posizione in magazzino)
2. Il Magazziniere preme sul pulsante “Inserisci Entrata”

**Postcondizioni:**

1. Viene inserita un’entrata nel Database

**Caso d’uso**: Inserisci Uscite

**Precondizioni:** L’utente deve aver fatto il Login da Magazziniere

**Sequenza degli eventi**:

1. Il magazziniere deve inserire i dettagli dell’uscita (Data, Bolla, Negozio, spedizioniere ed ordine)
2. Il Magazziniere preme sul pulsante “Inserisci Uscita”

**Postcondizioni:**

1. Viene inserita un’uscita nel Database

**Caso d’uso**: Inserisci Tipo Articolo

**Precondizioni:** L’utente deve aver fatto il Login da Segreteria Amministrativa

**Sequenza degli eventi**:

1. Vengono inseriti i dettagli dell’articolo(Nome, Descrizione, Sport e lista materiali)
2. Il Magazziniere preme sul pulsante “Inserisci Articolo”

**Postcondizioni:**

1. Viene inserita il tipo di articolo nel Database

**Caso d’uso**: Visualizza entrate

**Precondizioni:** L’utente deve aver fatto il Login da Segreteria Amministrativa

**Sequenza degli eventi**:

1. Viene premuto il pulsante visualizza entrate

**Postcondizioni:**

1. Viene visualizzata una lista delle entrate

**Caso d’uso**: Visualizza uscite

**Precondizioni:** L’utente deve aver fatto il Login da Segreteria Amministrativa

**Sequenza degli eventi**:

1. Viene premuto il pulsante visualizza uscite

**Postcondizioni:**

1. Viene visualizzata una lista delle uscite

**Caso d’uso**: Visualizza Ordini Passati

**Precondizioni:** L’utente deve aver fatto il Login da Gestore del negozio

**Sequenza degli eventi**:

1. Viene premuto il pulsante visualizza storico

**Postcondizioni:**

1. Viene visualizzata una lista degli ordini passati

**Caso d’uso**: Aggiungi Ordine

**Precondizioni:** L’utente deve aver fatto il Login da Gestore del negozio

**Sequenza degli eventi**:

1. Vengono Inseriti i dettagli dell’ordine(Codice, data, Tipo articolo, quantità, prezzo unitario)

**Postcondizioni:**

1. Viene inserito l’ordine nel database

Pattern Adottati

**Facade**: EventManager è un interfaccia front-end per la classe DBManager, difatti nessuna delle classi di gestione dell’interfaccia interagisce direttamente con il DB ma lo fa attraverso delle funzioni contenute in EventManager il quale poi va a chiamare delle funzioni contenute in DBManager

**Singleton**: Tutte le classi in questo progetto sono dei singleton in quanto alcune di esse forniscono solo delle funzioni di servizio (EventManager, DBManager e PasswordAuthenticationService) oppure sono associate alla form correntemente visualizzata, per cui ha senso che vi sia una sola istanza di esse.

**Proxy**: La classe DBManager fa da proxy per il Database in quanto gestisce il suo uso e la creazione della connessione ad esso.

**Observer:** Le classi di gestione dell’interfaccia implementano la parte observr, in quanto s’”iscrivono” ad alcuni eventi legati all’interfaccia e reagiscono di conseguenza.

Come pattern architetturale si è scelto il Model-View-Controller.

Il model è dato dal Database e dall’EventManager in quanto il DB gestisce i dati che vengono poi suati dal programma, mentre l’EventManager gestisce l’accesso ai dati secondo la logica di chi sta usando l’applicazione.

La view è data dalla definizione dell’interfaccia e dalle librerie associate ad essa in quanto regolano come vengono presentati i dati all’utente finale e forniscono gli eventi generati dall’utente al controller.

Il controller è dato dalle classi che ricevono gli eventi dell’interfaccia, dal DBmanager e dal servizio di autenticazione in quanto esse gestiscono la logica interna dell’applicazione