Progetto Ingegneria del Software: Gestionale “Sweet & Sour”

Matteo Casarotto Sant’Ana, Alvin Preidt, Enrico Zasso

DOCUMENTO DI ARCHITETTURA

1. SCOPO DEL DOCUMENTO

Il presente documento contiene la definizione dell’architettura di “Sweet & Sour”, attraverso l’utilizzo di diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL).
Tenendo conto della progettazione definita nel documento precedente attraverso il diagramma degli use case, il diagramma di contesto, e quello dei componenti, viene definita l’architettura del sistema dettagliando le classi che dovranno essere implementate a livello software, e la logica che regola il suo comportamento.
Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML, invece la logica viene descritta in OCL poichè tali concetti non sono rappresentabili in nessun altro modo attraverso il linguaggio UML.

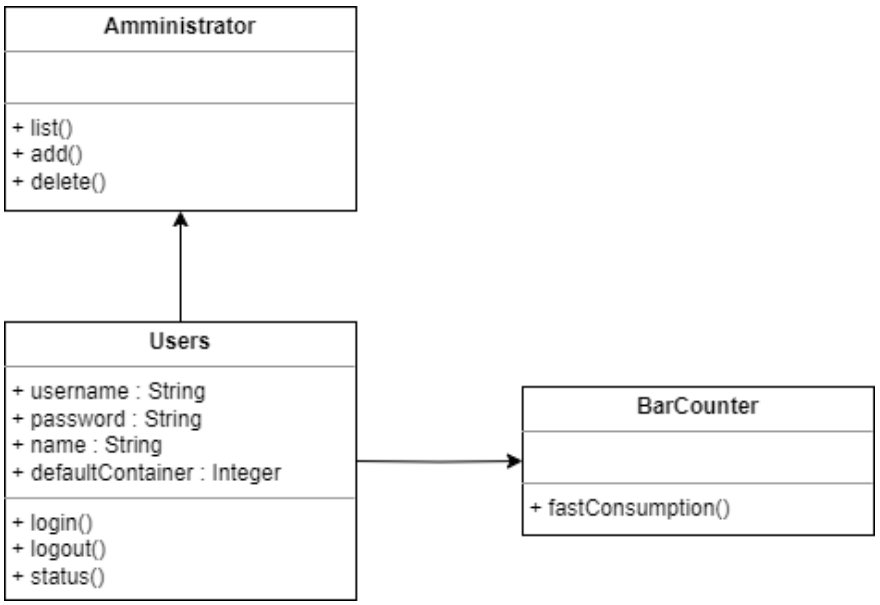
2. DIAGRAMMA DELLE CLASSI

Nel presente capitolo vengono presentate le classi previste per il sistema “Sweet & Sour”.
I componenti del diagramma di contesto, presentati nel D3, vengono trasformati in classi con i rispettivi attributi (dati gestiti dalla classe) e metodi (operazioni eseguibili dalla classe).
Ogni classe può essere associata ad altre classi per definire relazioni tra esse.

2.1. UTENTI

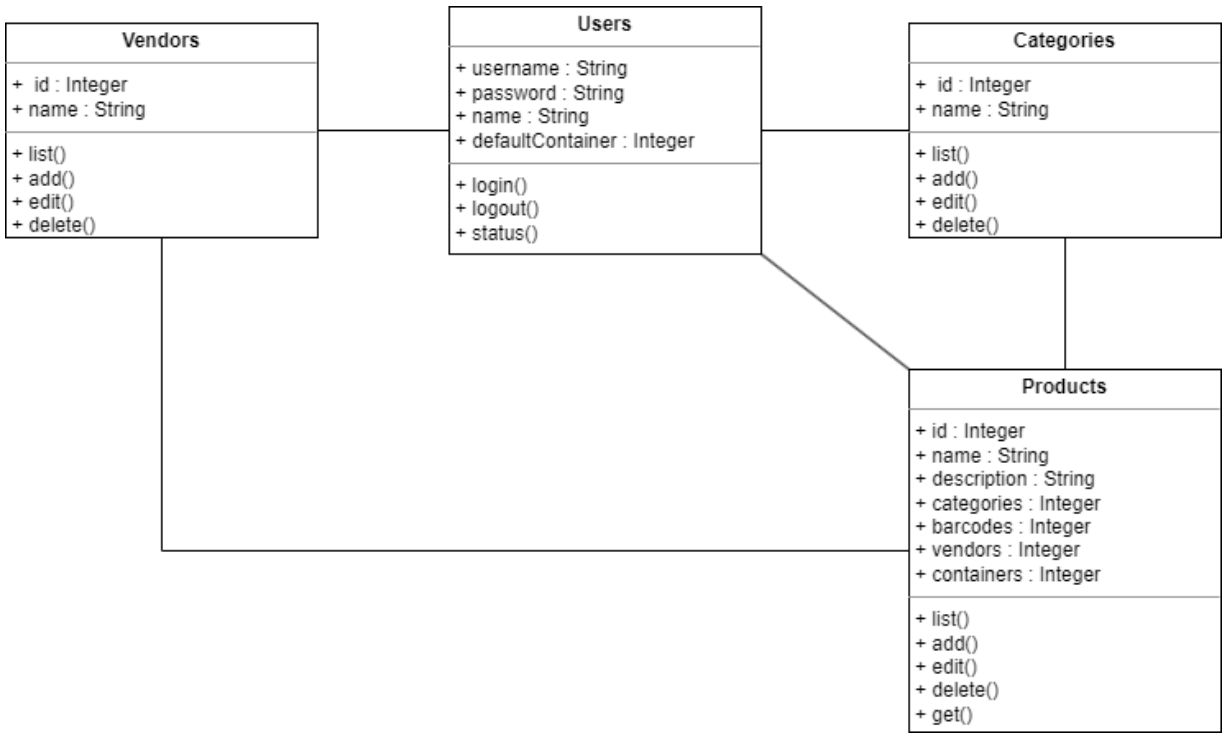
Il diagramma di contesto del documento precedente include la presenza di tre attori: “User”, “Amministratore” e “Banco”.

- User è il principale utilizzatore dell’applicazione e ha accesso alla maggior parte dei dati memorizzati all’interno del sistema. Questo attore dispone delle funzionalità CRUD per prodotti, container, movimenti, fatture, categorie e fornitori, può eseguire spostamenti di prodotti tra i container e inserire nel sistema e analizzare le fatture.
- Amministratore eredita ogni attributo e metodo di “User” e quindi può svolgere tutte le azioni di quest’ultimo. Egli, inoltre, può visualizzare e creare statistiche, gestire gli utenti, cioè può modificarli, aggiungerli e rimuoverli.
- Banco, oltre a ereditare tutte le funzioni di “User” dispone del metodo *fastConsumption()* che gli permette, attraverso la scannerizzazione di un codice a barre, di rimuovere un prodotto dal container associato all’utente.



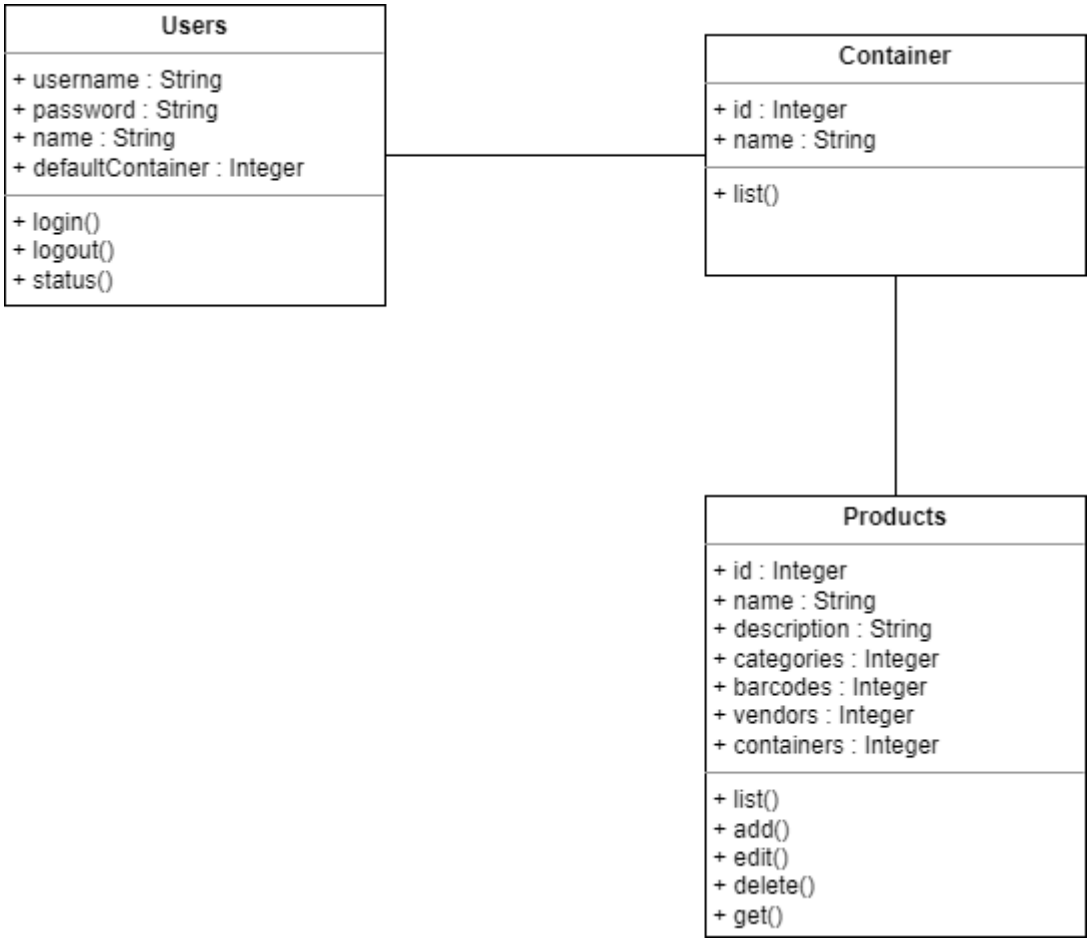
2.2. GESTIONE PRODOTTI, FORNITORI E CATEGORIE

Per i componenti “Ricerca prodotto”, “Gestione prodotto”, “Fornitori” e “Categorie”, che sono strettamente connesse, abbiamo identificato tre classi attraverso le quali l’utente può accedere alle funzionalità CRUD per i prodotti, i fornitori e le categorie.



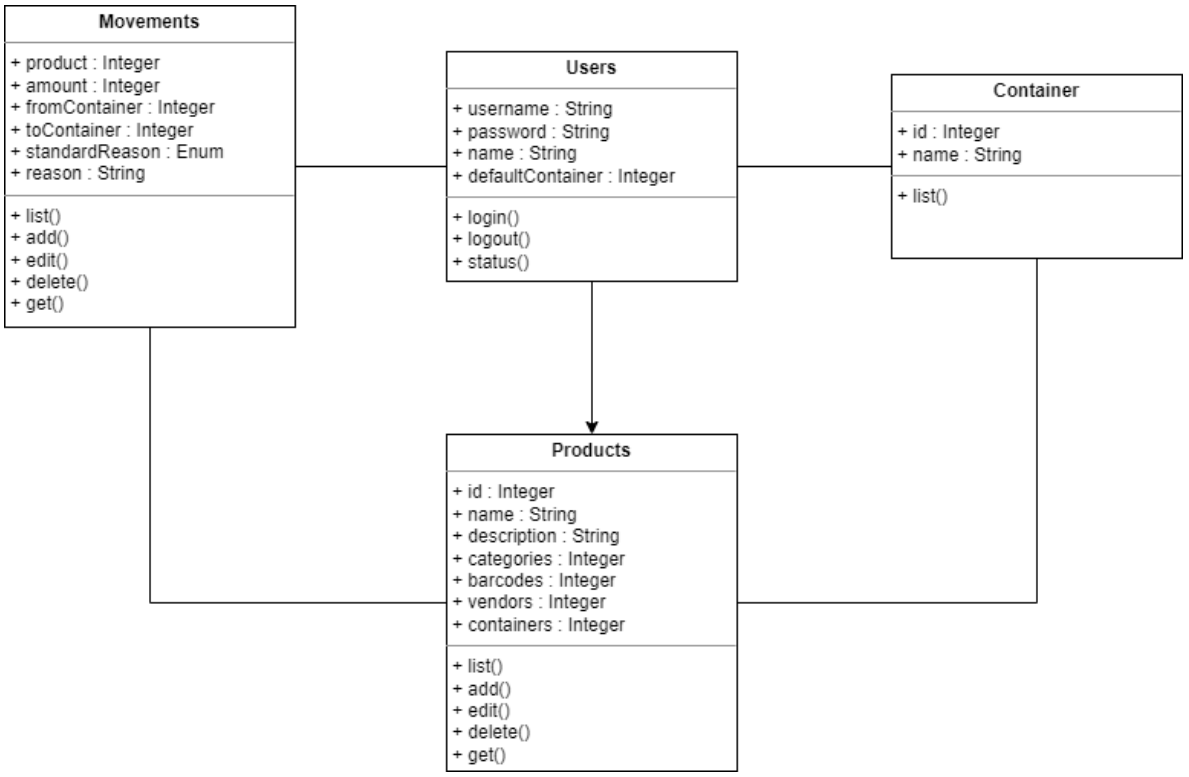
2.3. GESTIONE CONTAINERS

Per la gestione del componente “Gestione Container” abbiamo identificato una classe che permette di visualizzare i containers registrati nel sistema dai quali poter accedere a varie funzionalità. Questa classe deve essere opportunamente relazionata al componente di gestione prodotti.



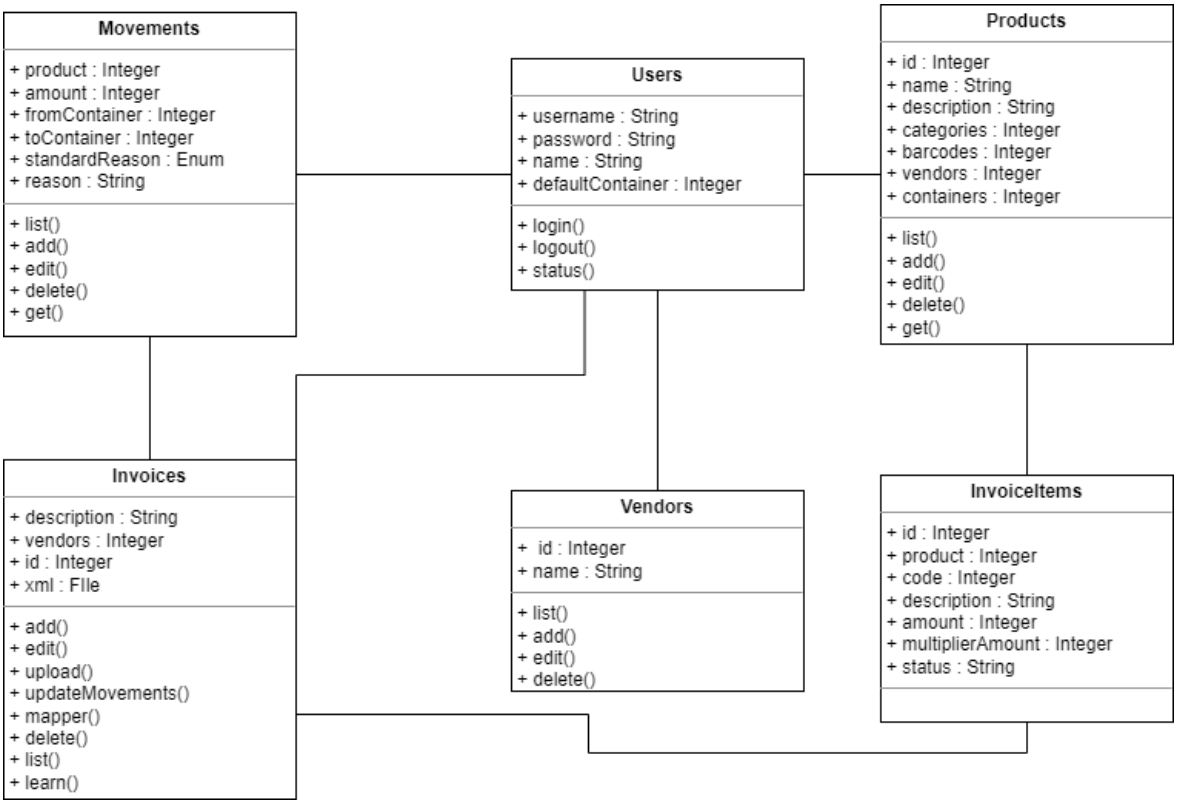
2.4. GESTIONE MOVIMENTI

I componenti “Movimenti” e “Ricerca Movimenti” permettono all’utente di effettuare movimenti di prodotti tra container, di effettuare carichi dal magazzino e di accedere alle funzionalità CRUD per i movimenti. Abbiamo identificato una classe, opportunamente relazionata alle componenti di gestione prodotti e gestione container, per adempiere a questo scopo.



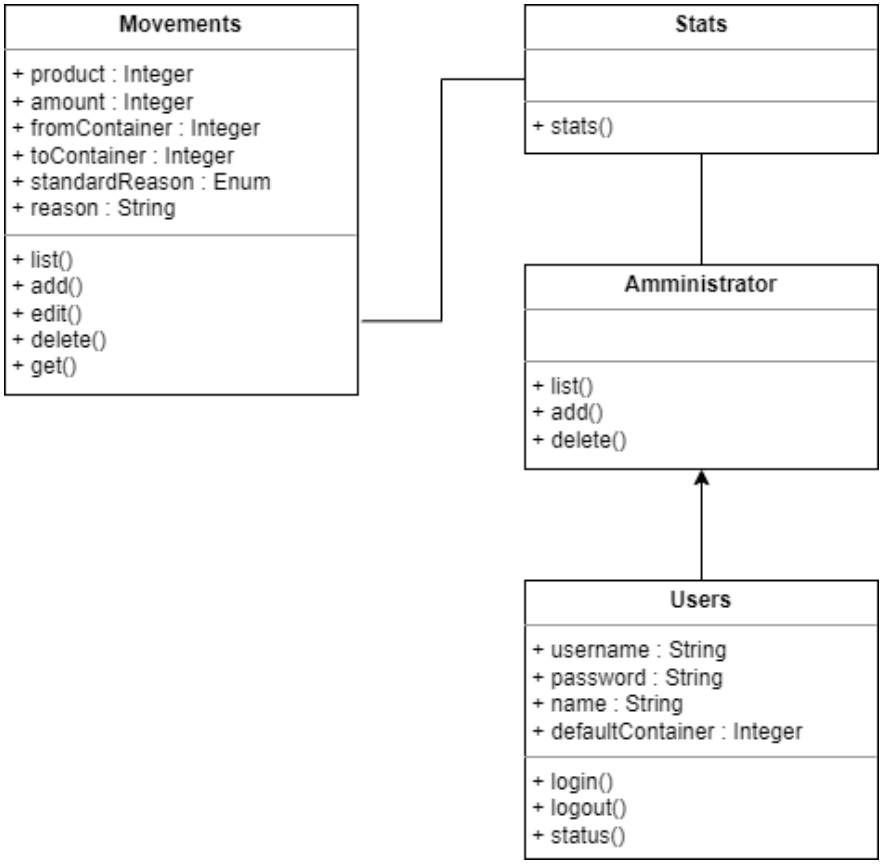
2.5. GESTIONE FATTURE

Per la componente “Lettura Fatture” identifichiamo due classi opportunamente relazionate alla componente di gestione prodotti, quella di gestione movimenti e quella dei fornitori. Questa classe permette all’utente di caricare a sistema una fattura in formato .xml e caricare il magazzino con i prodotti rilevati in essa oltre che ad accedere, anche in questo caso, alle funzionalità CRUD.



2.6. GESTIONE STATISTICHE

Il componente “Statistiche” permette all’utente Amministratore di visualizzare delle opportune statistiche realizzate sulla base dei movimenti effettuati. Per permettere questa funzionalità abbiamo identificato una classe adeguatamente relazionata con la componente di gestione movimenti.



3. CODICE IN OBJECT CONSTRAINT LANGUAGE

In questo capitolo viene descritta la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché questi concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

3.1. Unicità dei fornitori

Non è possibile per un utente creare un fornitore con lo stesso nome di uno già esistente.

```
context Vendors inv: Vendors.allInstances->forAll(v |
v.name != self.name)
```

3.2. Unicità delle categorie

Non è possibile per un utente creare una categoria con lo stesso nome di una già esistente.

```
context Categories inv:
Categories.allInstances->forAll(c |
c.name != self.name)
```

3.3. Limiti per i dati del prodotto

- Non è possibile per un utente creare un prodotto con lo stesso nome di uno già esistente.
- Non è possibile per un utente associare a un prodotto un codice a barre già associato ad un altro prodotto.
- Le quantità minime di un prodotto (per ogni container) sono sempre maggiori o uguali a 0.

```
context Products inv: Products.allInstances->forAll(p |
p.name != self.name)

context Products inv: self.barcodes->forAll(b1 |
! Products.allInstances->exists(p |
p.barcodes->forAll(b2 | b1 == b2)))

context Products inv: self.containers->forAll(c |
c.min_amount >= 0)
```

4. DIAGRAMMA DELLE CLASSI CON CODICE OCL

Viene riportato il diagramma delle classi con tutte le classi presentate fino ad ora, completo di OCL.