****

**Master Document**

**Versione <0.1>**

# 1. Sommario

# 2. Autori

|  |  |  |
| --- | --- | --- |
| **Nome** | **Cognome** | **Matricola** |
| Diego | Avella | 05225 00464 |
| Domenico Antonio | Tropeano | 05225 00513 |

# 3. Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 21/11/2017 | 0.1 | Init del template | Tutti |

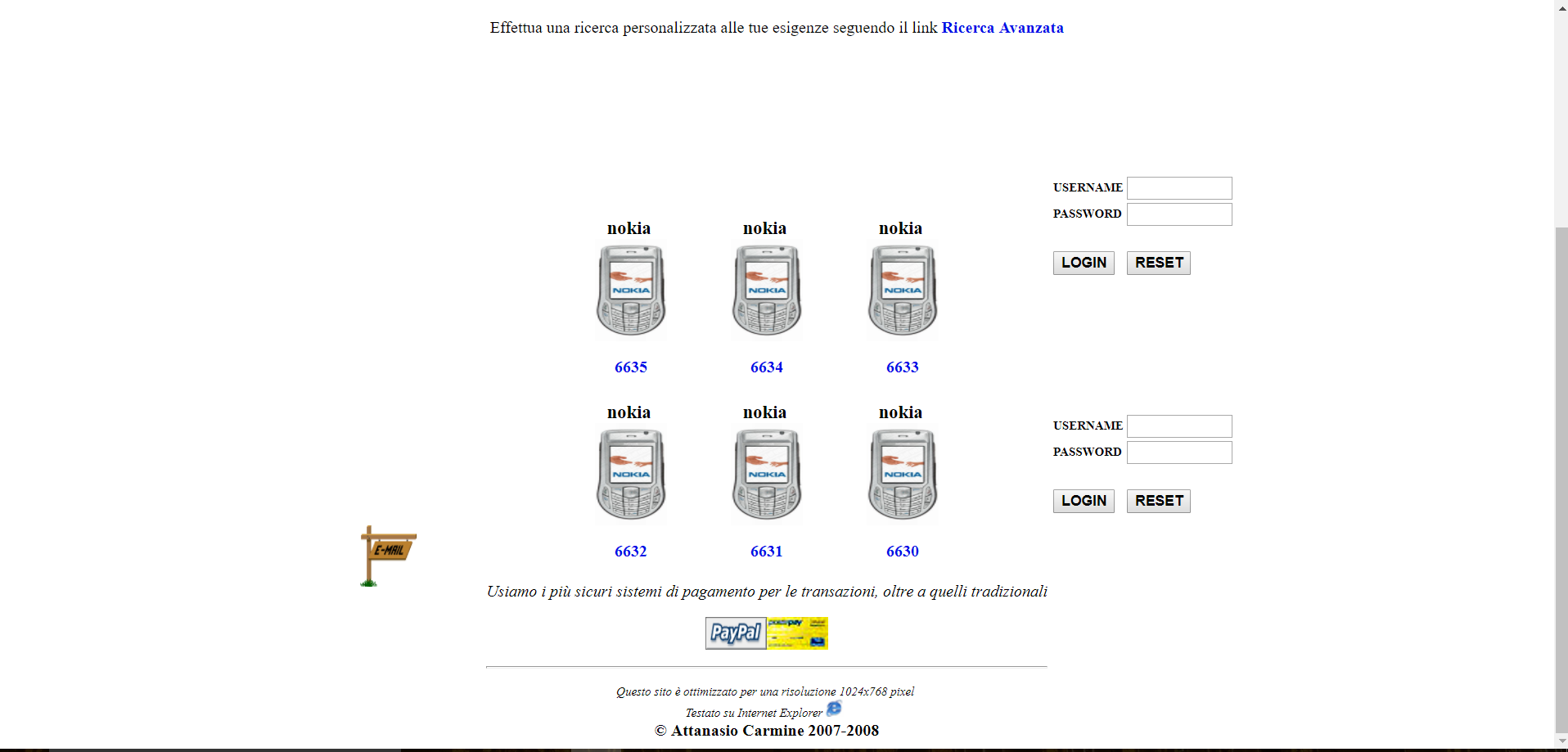
# 4. Contesto del sistema

## 4.1. Introduzione

Il sistema sul quale si vuole applicare una change request è un e-commerce dedicato alla vendita di smartphone. È evidente che si è cercato di implementare le funzioni principali di un generico servizio di vendita online. Il contesto in cui è stato sviluppato il software è quello accademico,ciò fa comprendere che sono stati tralasciati molti aspetti fondamentali di uno sviluppo da software web professionale. Il software è stato sviluppato nel 2008, basandosi sulla tecnologia di Java Servlet e JSP per il back-end , e tecnologie HTML, CSS e JavaScript per il lato front-end.

L’applicativo sviluppato non presenta alcuna documentazione,né di tipo formale, né di tipo informale. Ciò implicherà la presenza di una massiccia attività di reverse engineering.

Eseguendo normalmente l’applicativo si viene accolti dalla seguente pagina principale



Con una prima analisi superficiale si può notare che le funzionalità principali hanno un funzionamento corretto. Di seguito saranno descritti le varie caratteristiche del sistema.

## 4.2. Panoramica delle funzionalità

**Login**

Tutte le operazione (tranne la ricerca), richiedono come prerequisito l’autenticazione dell’utente, che avviene tramite l’inserimento di un username ed una password. Si distinguono due attori principali: l’utente semplice ed il gestore.

**Navigazione & Ricerca**

La funzionalità di utilizzo più immediato è la ricerca e la navigazione del sito. Vi sono due tipi di ricerca :

* semplice: si può effettuare ricerca per marca e per modello.
* avanzata: può essere effettuata una ricerca considerando varie tipologie di parametri.

La navigazione può avvenire tramite ipertesto, inoltre il permette una navigazione coerente tramite i comandi “UNDO” e “REDO”.

**Ordinazione**

Quando si è loggati come cliente,è possibile inserire vari prodotti nel carrello, ogni aggiunta viene gestita come un oggetto “Ordine”. Una volta riempito il carrello è possibile effettuare l’acquisto dei prodotti o l’eliminazione di essi.

**Modifica del profilo**

L’utente semplice può effettuare varie modifiche sul proprio account, in particolare gli è permesso modificare quasi tutti i campi delle info personali. L’utente può eliminare definitivamente il proprio account.

**Vendita**

Quando si vuole effettuare l’acquisto del contenuto del carrello si viene indirizzati in un processo di acquisto, durante il quale è possibile scegliere le modalità di pagamento e di spedizione. Ogni oggetto “Ordine” viene trasformato in un oggetto vendita.

**Amministrazione**

Sono presenti alcuni account che godono privilegi di amministratore, potendo così effettuare modifiche sui prodotti presenti nel magazzino (Eliminazione, Aggiunta, Modifica). Sono presenti due funzionalità di visualizzazione: una permette di vedere i clienti e l’altra permette la visualizzazione dei prodotti in via di esaurimento. È possibile inoltre effettuare importazione del “magazzino”.

## 4.3. Tecnologie utilizzate

Il progetto è stato sviluppato tramite Java Servlet e JSP, per il lato front-end è stato utilizzato HTML,CSS,JavaScript. Il database è stata sviluppato tramite una struttura a file basata su XML. Dato che il software è stato sviluppato nel 2008 le librerie utilizzate risultano ora obsolete.

## 4.4. Principi di architettura

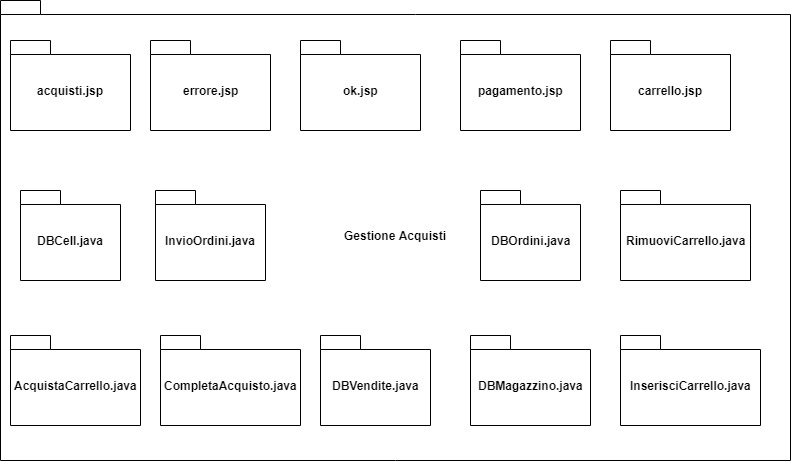
Il progetto è stato sviluppato cercando di seguire una struttura Two-Tier fisico, utilizzando poche direttive MVC.

# 5. Attività di Reverse Engineering

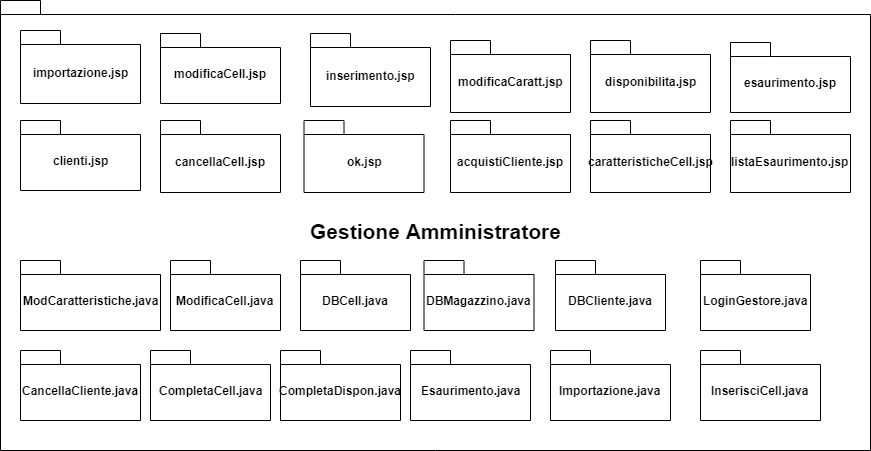
Per lavorare sulla change request si è innanzitutto eseguita una pre-fase di reverse engineering sul sistema in modo da acquisirne le conoscenze strutturali e funzionali necessarie per intervenire dove è opportuno. L'approccio utilizzato è stato di tipo bottom up usando la notazione **UML.** Innanzitutto sono stati estratti i Package Diagram per avere una comprensione delle classi coinvolte all’interno di ogni funzionalità, infine sono stati estratti i Sequence Diagram per ottenere una panoramica su come il sistema esegue ogni sua singola funzionalità. L’estrazione è stata semi-automatica grazie al supporto di alcuni tool specifici.

## 5.1. Package Diagram

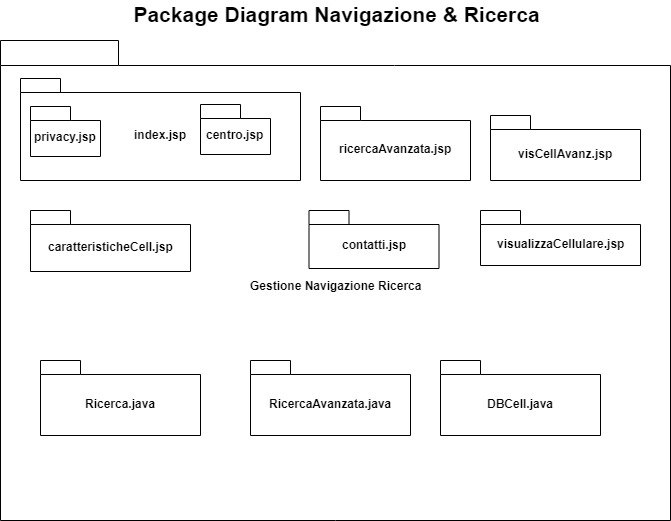
### 5.1.1. Gestione Acquisto:



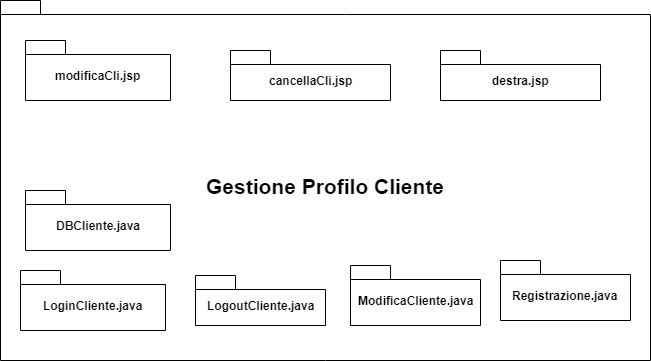
### 5.1.2. Gestione Amministratore:



### 5.1.3. Gestione Navigazione e Ricerca

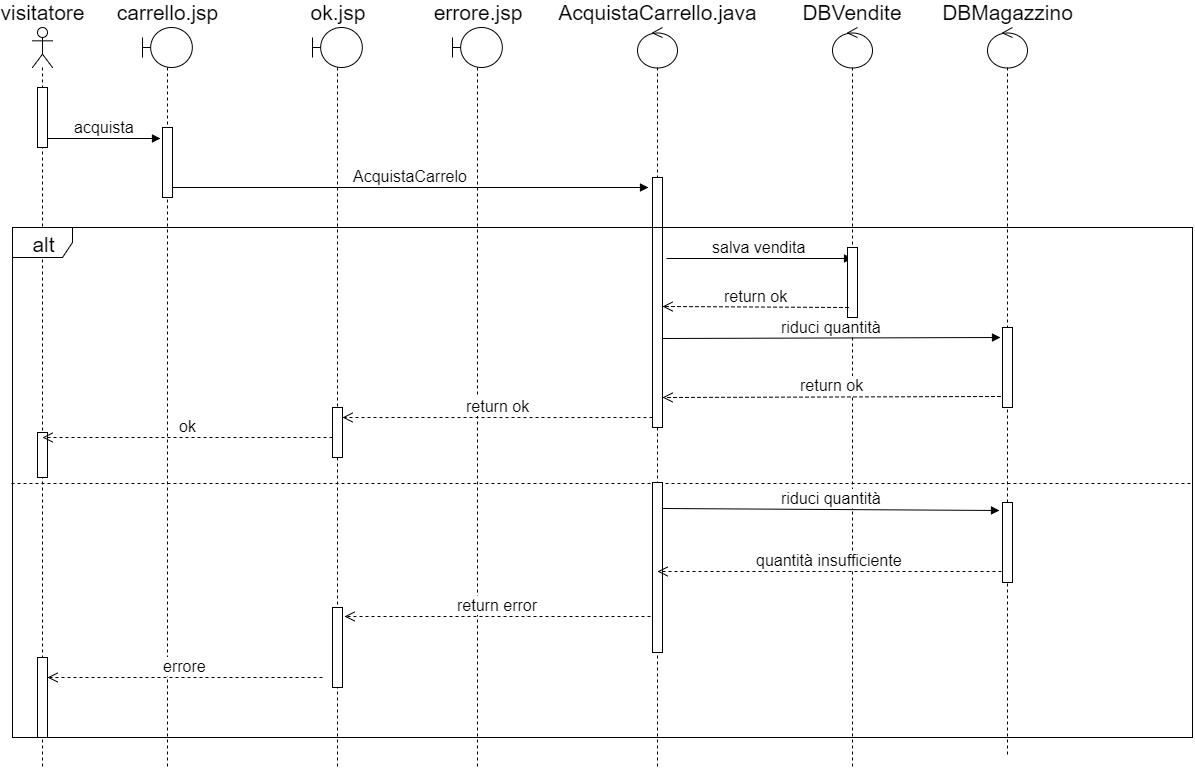


### 5.1.4. Gestione Profilo

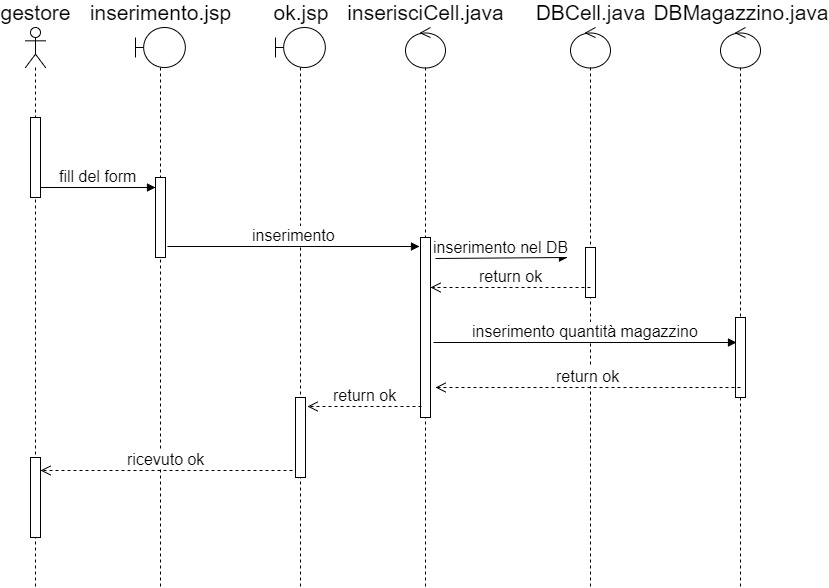


## 5.2. Sequence Diagram:

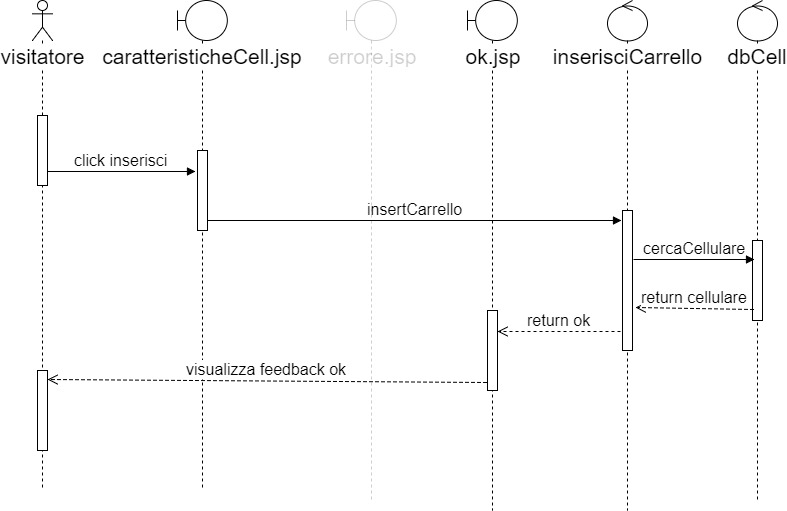
### 5.2.1. Acquista



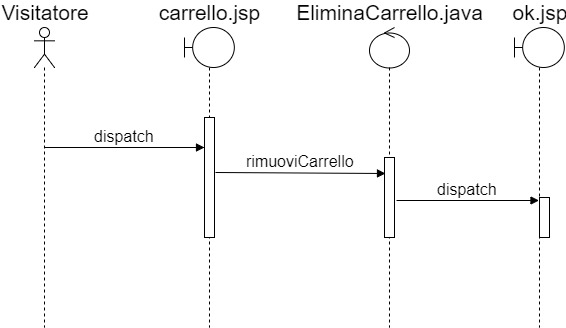
### 5.2.2. Add Cellulare:



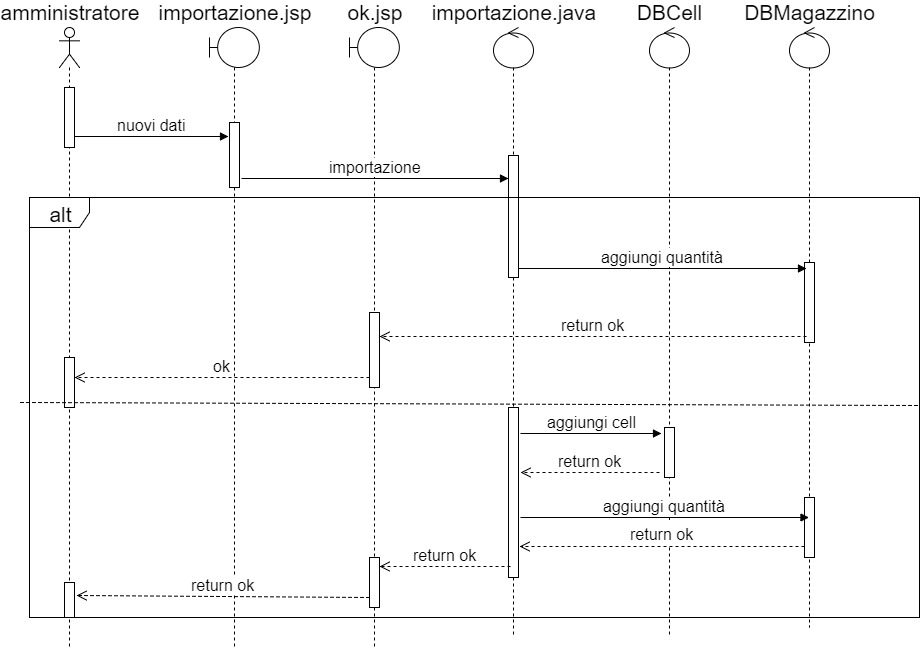
### 5.2.3. Aggiungi al Carrello:



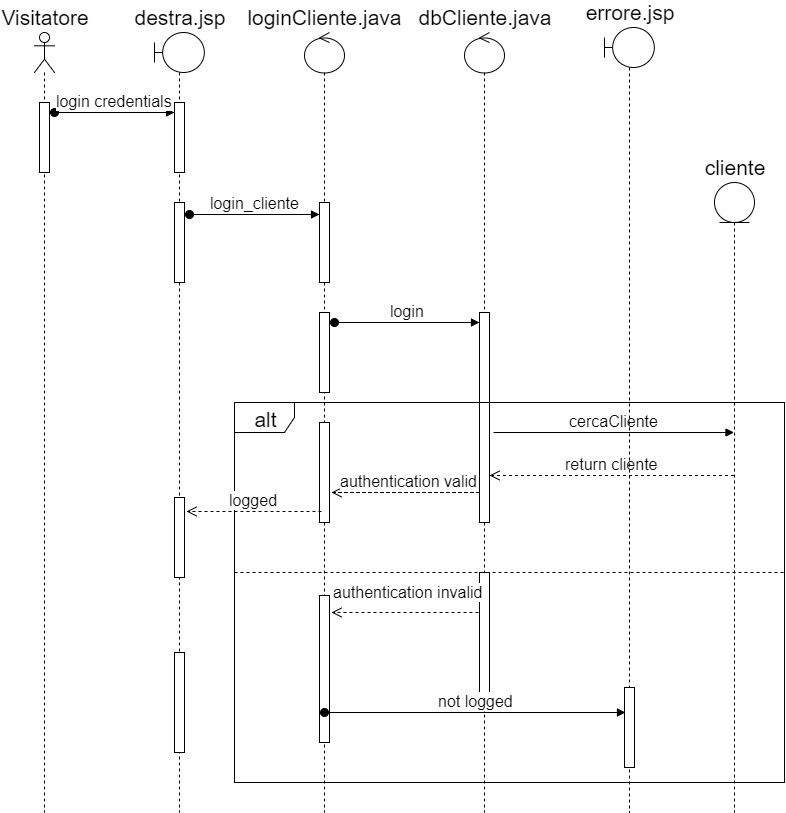
### 5.2.3.1. Elimina Carrello:



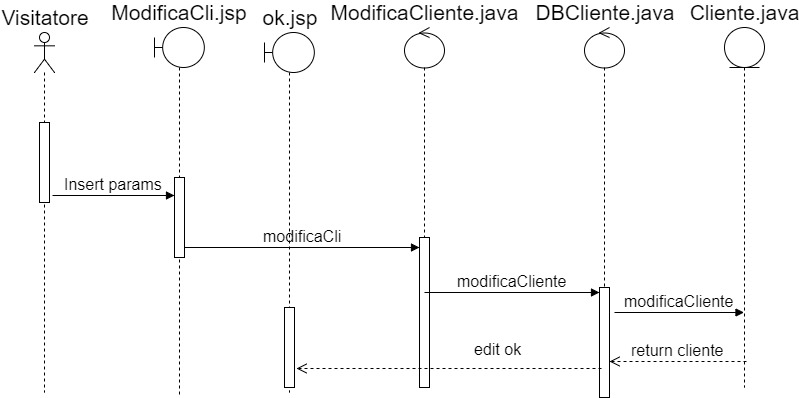
### 5.2.4. Importazione XML:



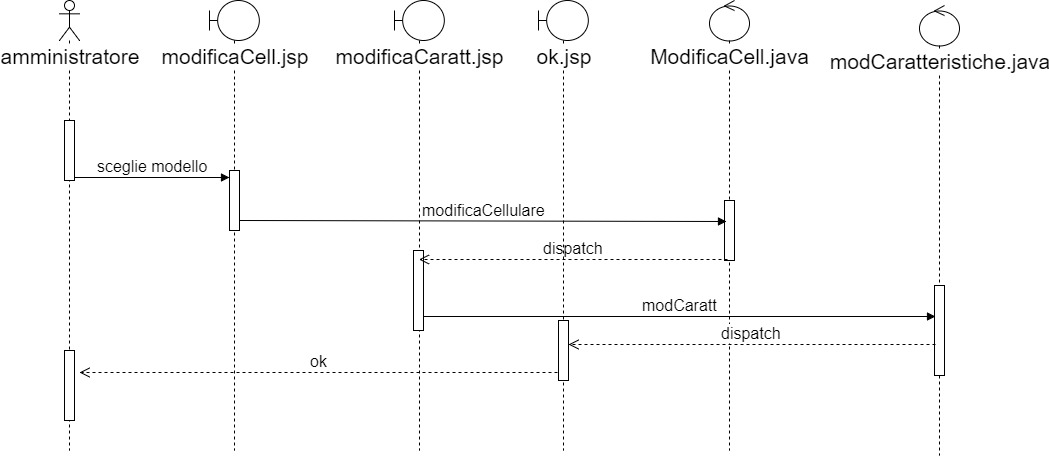
### 5.2.5. Login:



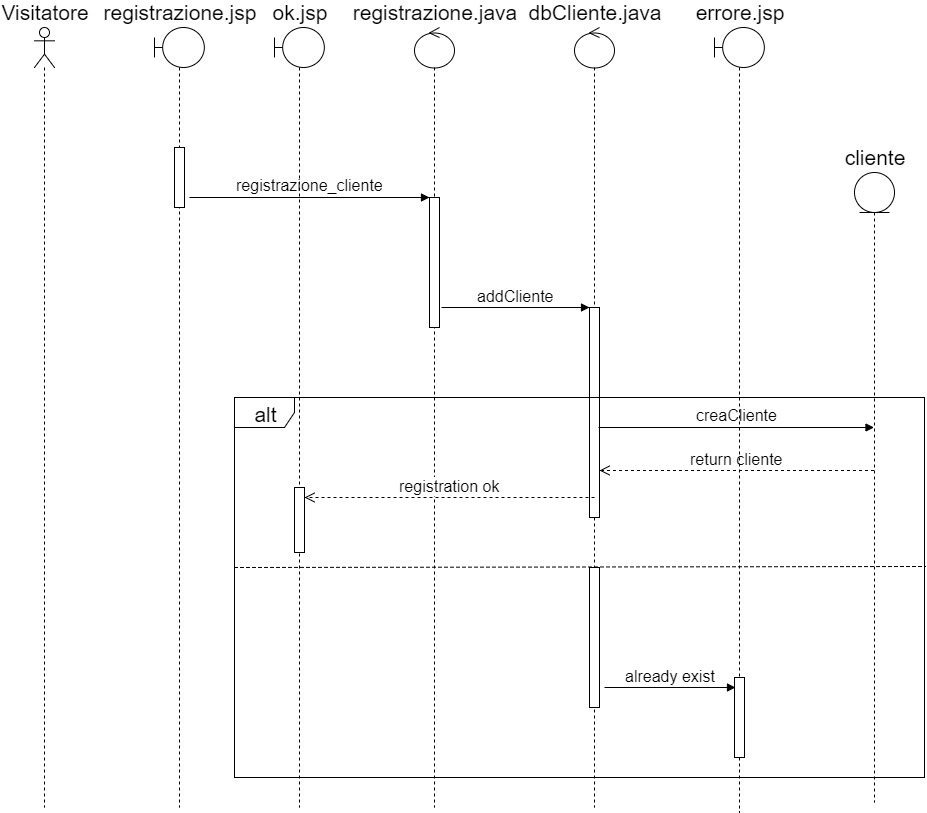
### 5.2.6. Modifica Account:



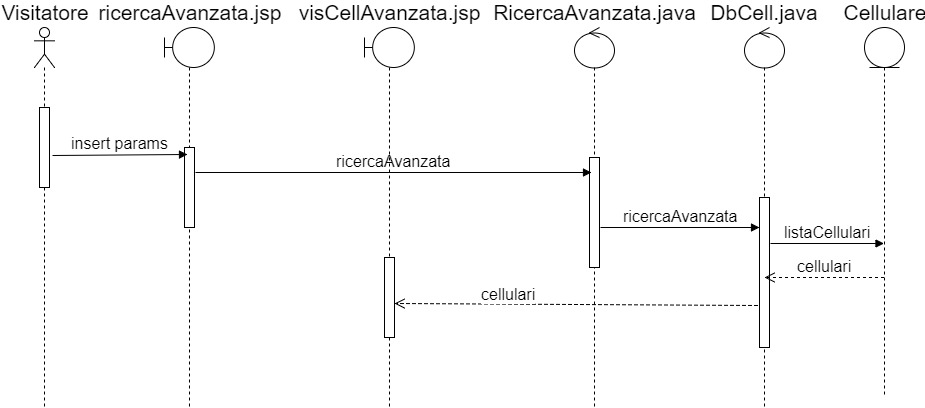
### 5.2.7. Modifica Cellulare:



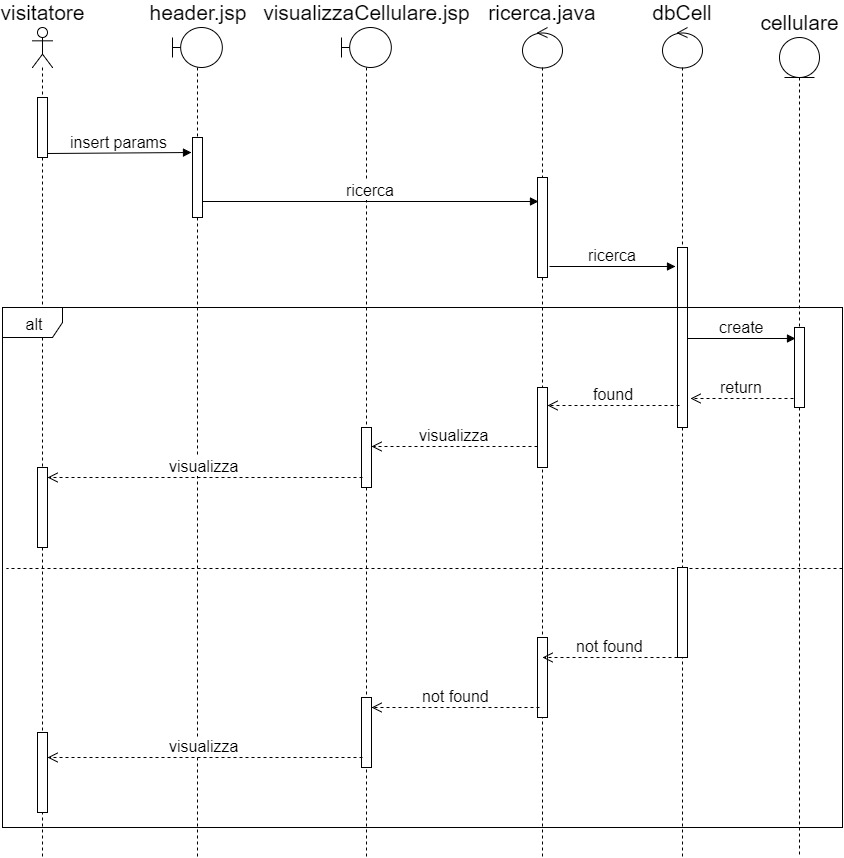
### 5.2.8. Registrazione:



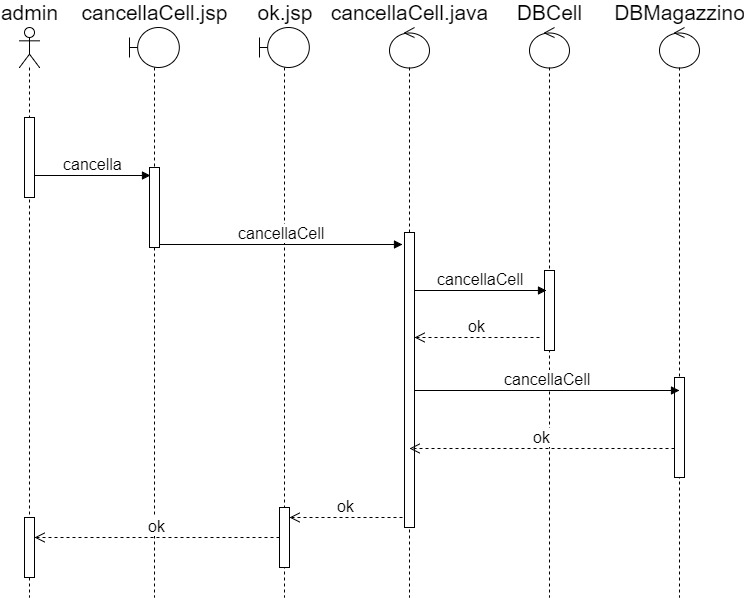
### 5.2.9. Ricerca Avanzata:



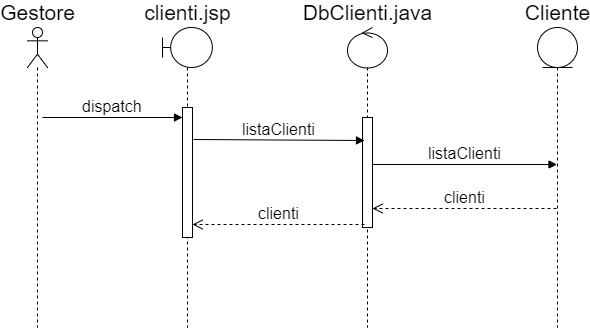
### 5.2.9.1. Ricerca:



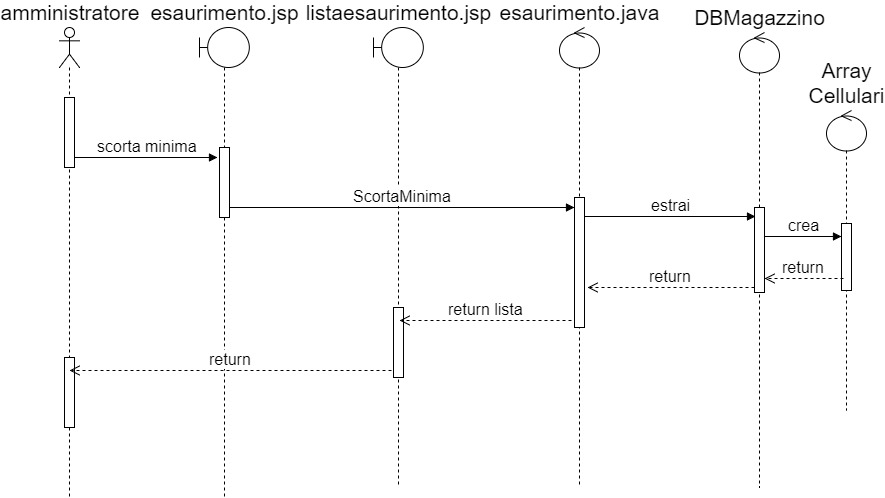
### 5.2.9.2. Rimuovi Cellulare:



### 5.2.9.3. Visualizza Clienti:



### 5.2.10. Visualizza Prodotti in Esaurimento:



# 6. Impact Analysis

## 6.1. Descrizione della change request

### 6.1.1. Layer Database per DB relazionale

Nella change request si fa richiesta di sostituire lo storage layer sottostante, rappresentato attualmente tramite file xml, con una database relazionale utilizzando come DBMS *MySQL* .

Tale cambiamento è opportuno in quanto la nuova struttura , oltre a delineare in modo netto le entità, permette una scalabilità e manutenibilità maggiore.

### 6.1.2. Modifiche estetiche front-end(HTML5,CSS3,JQUERY)

Verranno apportare dei cambiamenti alle interfacce dell’applicativo, rendendolo utilizzabile anche su dispositivi mobili. In particolare sarà adottato il Framework CSS Foundation.

### 6.1.3. Trasformazione cellulare a smartphone

Dato che l’applicativo è stato progettato circa 10 anni fa si rende necessario sostituire l’entità CELLULARE con una più moderna entità SMARTPHONE.

### 6.1.4. Attivazione della funzionalità di registrazione.

Deve essere permessa la possibilità di aggiungere nuovi utenti, implementando quindi una funzionalità di registrazione.

## 6.2. Implicazione dei cambiamenti proposti

### 6.2.1. Layer di storage

La sostituzione del layer di storage impatterà orizzontalmente tutto il sistema, ciò richiede quindi un cambiamento strutturale dei Model Object e dei Control Object. Questo cambiamento ha forti impatti su varie caratteristiche del sistema:

* Sicurezza: Implementando il layer di storage tramite un database relazionale, sarà necessario inserire alcuni controlli per preservare l’integrità dei dati.
* Prestazioni: I database relazionali sono notoriamente più lenti di una archiviazione basata su xml. Per preservare dei tempi di risposta accettabili, sarà necessario progettare efficientemente le chiamate a sistema.
* Affidabilità: Costruendo una struttura basata su un database gestionale è possibile gestire in modo ottimale gli errori.

### 6.2.2. Modifiche Estetiche

Le modifiche estetiche hanno un impatto molto leggero sul sistema ,essendo l'architettura basata su un layer MVC è possibile modificare solo il layer di presentazione. Il sistema presenta un utilizzo massiccio di scriptlet all’interno delle pagine JSP ma per quanto sia un approccio semplice e di facile utilizzo presenta notevoli problemi:

1. **Bassa leggibilità:** Nel momento in cui la pagina viene sovraccaricata di codice java ed HTML insieme diventa difficile, anche per un esperto, poter identificare subito di quale funzionalità si occupi la pagina JSP.
2. **Pessima manutenibilità:** Quando si interviene su un metodo o una classe utilizzata da una JSP che utilizza scriplet oltre alla classe da modificare si è costretti ad intervenire anche sulla pagina stessa rendendo il processo di manutenzione tedioso e frustrante oltre ad infrangere la regola base dell’ architettura MVC ovvero la separazione indipendente del layer di presentazione da quello di controllo.

Consapevoli di queste problematiche e considerando lo sforzo richiesto per effettuare con successo la change request, si è deciso di utilizzare la JSTL (java standard template library) in modo da risolvere i problemi sopracitati e irrobustendo l’architettura del sistema.

### 6.2.3. Trasformazione cellulare a smartphone

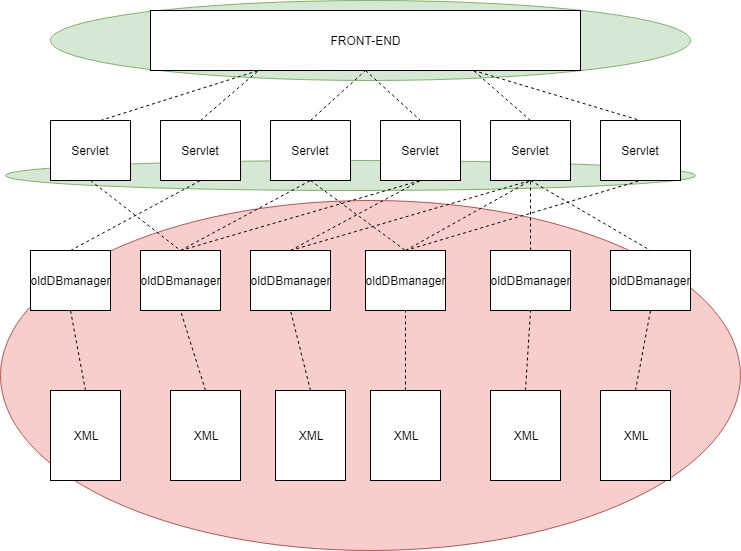
È la modifica che ha impatto verticalmente su tutto il sistema, ovviamente modificando il layer di storage implementare questa modifica sarà effettuata insieme alla ristrutturazione del layer. Ciò che invece chiederà un intervento a parte sarà la modifica del Control Object e la modifica del layer di presentazione. Successivamente verrà implementata la modifica dell’entity Smartphone negli altri layer. Ed infine sarà abilitata la funzionalità di registrazione.

### 6.2.4. Attivazione della funzionalità di registrazione.

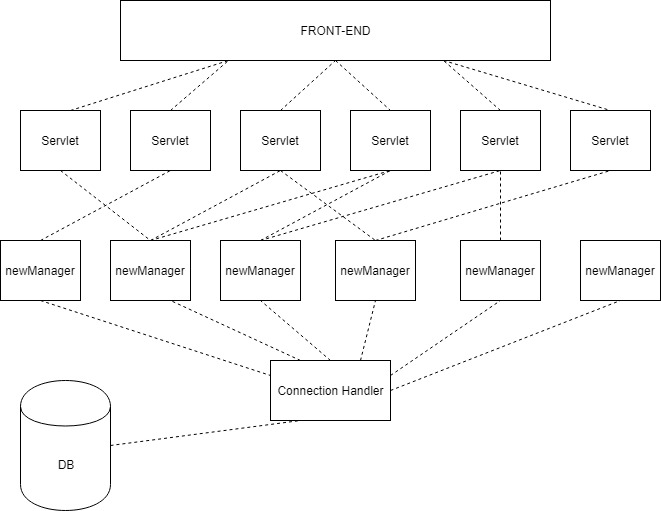
È una modifica che impatta leggermente sul sistema, in quanto la funzionalità è già implementata ma è disabilitata.

### Sunto

La change request richiede delle modifiche profonde nell’area circolare rossa e delle modifiche leggere nell’area circolare verde.



La struttura a grandi linee del software post modifica sarà la seguente:



## Valutazione alternative

Dall Impact Analysis si evince che tale modifica ha dei costi elevati, in quanto le modifiche impattato su tutte le classi e tutte le jsp. L’alternativa migliore è quella di rifiutare la change request e magari procedere con una nuova implementazione dell e-commerce.

# 7. Strategia Adottata

Per applicare le varie modifiche vengono definiti alcuni processi di implementazione di cui alcuni possono procedere in parallelo,mentre altri necessitano di essere eseguiti un tempo successivo in quanto hanno delle dipendenze con altre modifiche.

Il primo task da eseguire è la sostituzione del layer di storage, in contemporanea si può effettuare un primo step di manutenzione del front end dal punto di vista estetico tramite l’ausilio del framework CSS per dargli una “rinfrescata” e renderlo responsive. Successivamente sarà necessario aggiornare i Manager delle rispettive entity in modo che interagiscano con le tabelle del database. Questo passo richiede un enorme sforzo poiché sarà necessario superare l’incompatibilità che c’è tra il modello OOP e quello relazionale del DBMS utilizzato. Una soluzione rapida sarebbe stata quella di utilizzare una libreria che facesse da ORM a scapito di un overhead di performance ma essendo le tabelle coinvolte di numero discreto abbiamo deciso di scegliere una soluzione handmade usando tecniche comuni in ambito di sviluppo software come il Template Pattern e il DAO (Database Object Access Pattern). L’ultima fase consisterà nella ristrutturazione delle Servlet e nel secondo step della ristrutturazione del front end utilizzando i tag JSTL al posto degli scriplet.

Segue ora in dettaglio la descrizione della strategia adottata.

## 

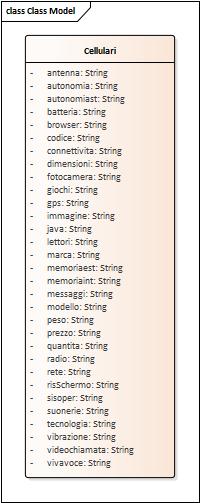
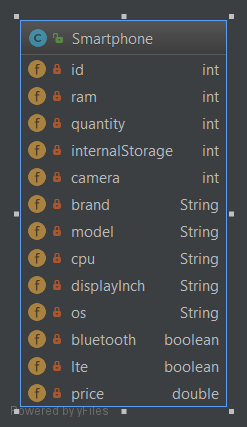
## 

## 

## 7.1. Layer Database per DB relazionale

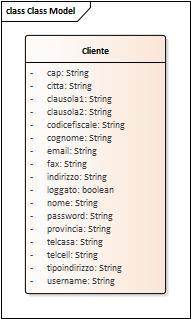
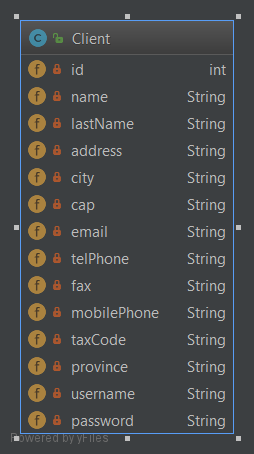
Modificare questo layer significa andare a modificare tutti i file del package **mobilesolutions.beans** contenente:

*Cellulari*



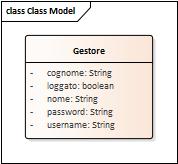
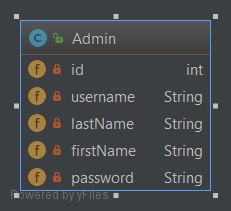
L’entità Cellulare verrà sostituita con una più moderna entità Smartphone. La lista di attributi è stata modificata, e sono state specificati anche i campi per ogni attributi. Maggiori dettagli possono trovarsi in figura….

*Cliente*



Verrà effettuato un refactoring dell’entità Cliente, in modo da tradurre tutti gli attributi in inglese. Alcuni degli attributi saranno eliminati o modificati. Per ognuno di essi sarà fornita una specifica formale. Maggiore dettagli possono trovarsi in figura.

*Gestore*

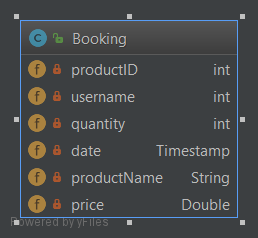
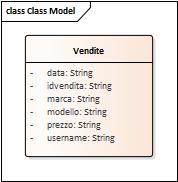


Verrà effettuato un refactoring sull’entità gestore, in modo da utilizzare la lingua inglese. Per ogni attributo è stata formalizzata una specifica, maggiori dettagli possono trovarsi in “Descrizioni funzioni servlet e controlli”

*Magazzino*

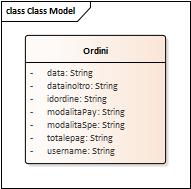
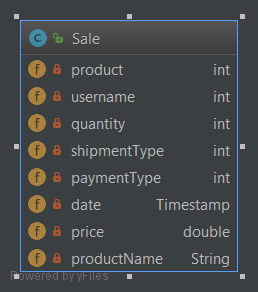
L’entità magazzino verrà completamente eliminata, in quanto è un’entità che può essere sostituita aggiungendo l’attributo “quantità” all’Entità Prodotto.

*Vendite*



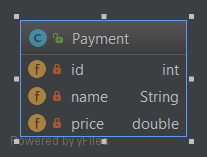
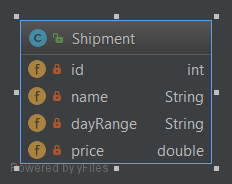
L’entità Vendite verrà sostituita con l’entità Booking,eliminando buona parte degli attributi, ciò è possibile aggiungendo un link al prodotto che sarà aggiunto nel carrello. Si è deciso di cambiare il nome per evitare ambiguità.

Ordini



L’entità Ordini verrà sostituita con l’entità Sale. Sono stati definiti le specifiche degli attributi, ed alcuni attributi stringa verranno sostituiti con dei link.

Sono stati inoltre aggiunge due entità :



Tali entità incapsuleranno dei dati che prima erano fissati nel codice, si è deciso che il caso di renderli gestibili.

Ovviamente sarà necessario sostituire completamente il layer che interagisce con lo storage.Il layer di storage sarà implementato tramite delle interfacce e si baserà sul tecnologia JDBC, ponendo particolare attenzione sui controlli sui dati.

La modifica impatta sulle classi presenti nel package **mobilesolution.gestioneDB** contenente:

* DBCell
* DBCliente
* DBGestore
* DBMagazzino
* DBOrdini
* DBVendite

Sarà necessario includere un layer sottostante che si occupi della gestione della connessione al database, al momento tale componente è completamente inesistente in quanto il salvataggio dei dati avviene tramite file xml.

Sarà necessario andare a ritoccare le servlet , in modo strettamente contenuto, in quanto la modifica riguarderà la sostituzione delle chiamate ai gestori dei database, dato che non esiste al momento una convenzione.

# 8. Design della soluzione

## Descrizione del layer di storage

Connection Handler: Database.java

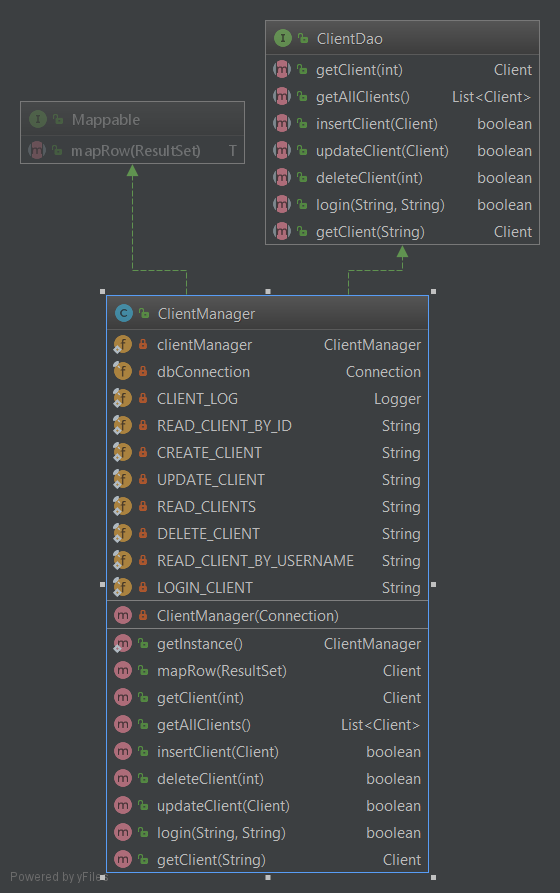
È l’oggetto statico che si occupa di inizializzare la connessione e di gestirne il ciclo di vita. Sarà implementato tramite una connection pool , ed è possibile richiamare i seguenti metodi:

* getConnection() : è un metodo che permette di ottenere una connessione al database.
* releaseConnection() : è un metodo che permette di rilasciare la connessione quando si è finito di utilizzarla.

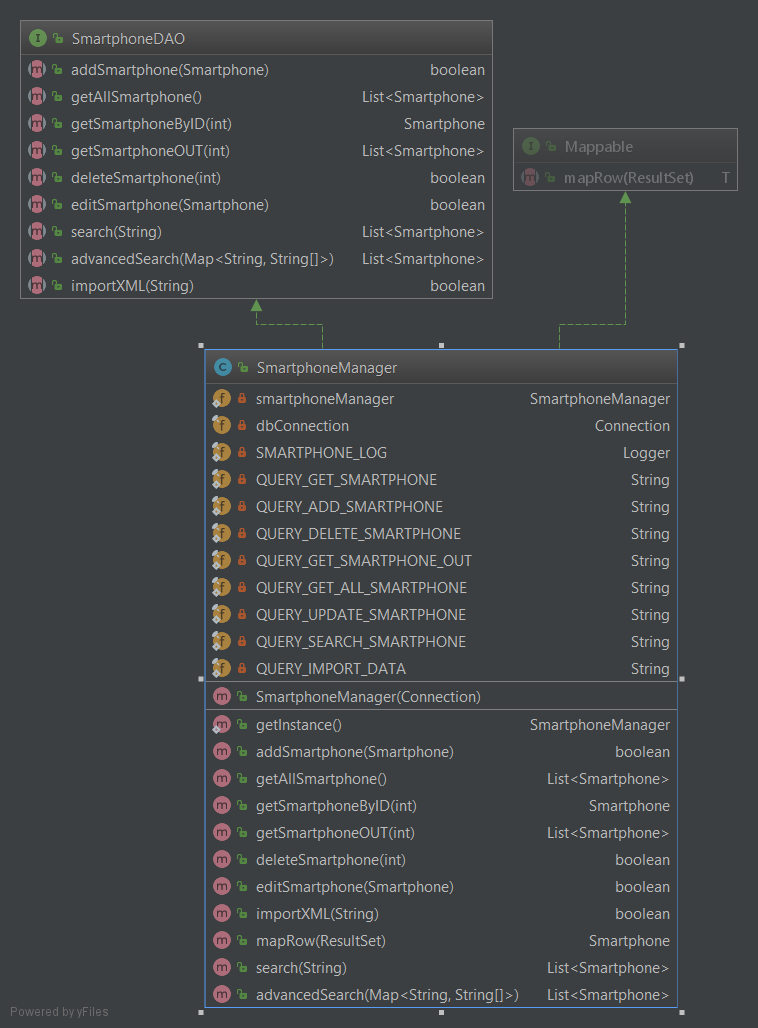
### Manager:

Vengono implementati alcuni manager, uno per ogni tabella nel database:

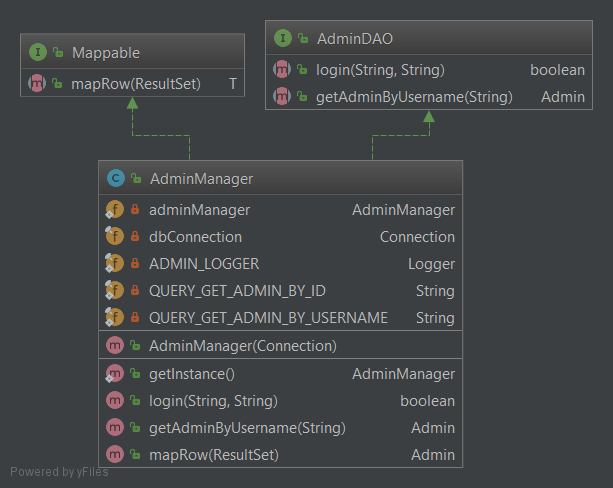
**ClientManager**

****

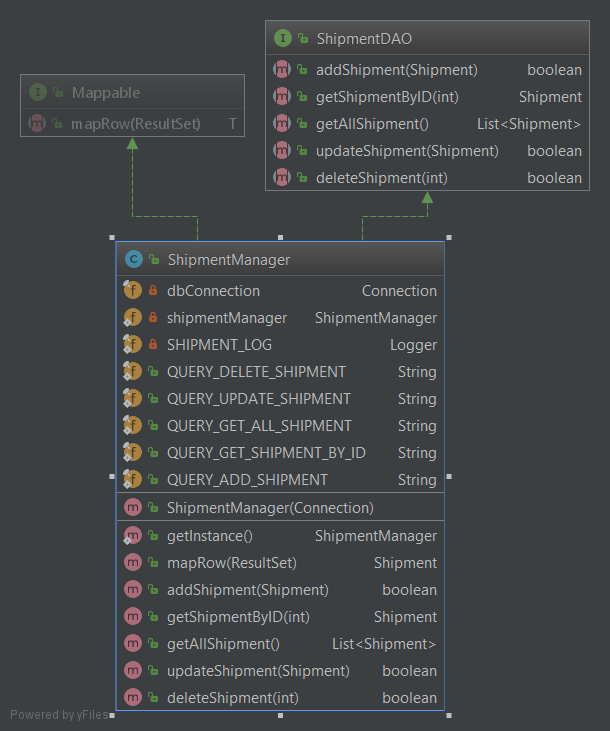
**SmartphoneManager**

****

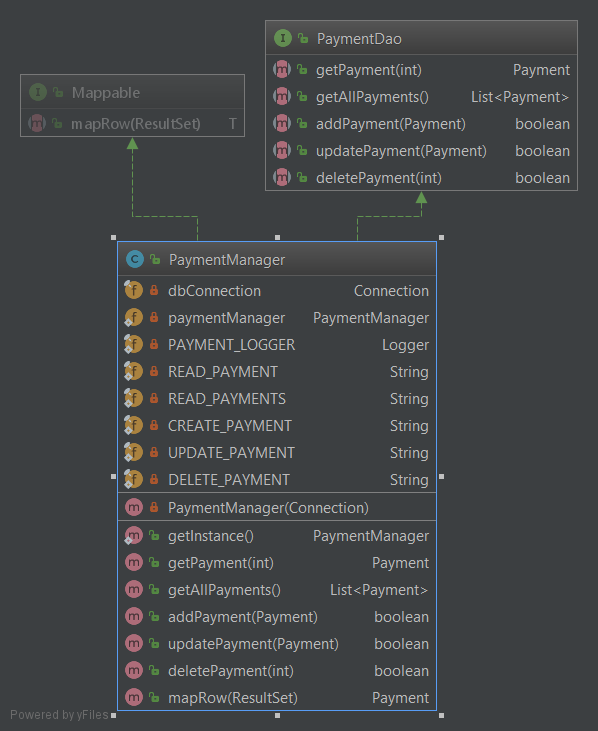
**AdminManager**



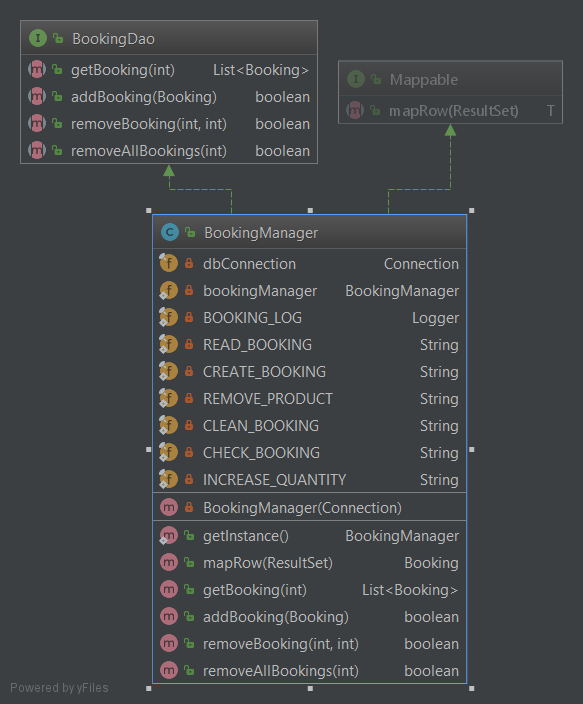
**ShipmentManager**

****

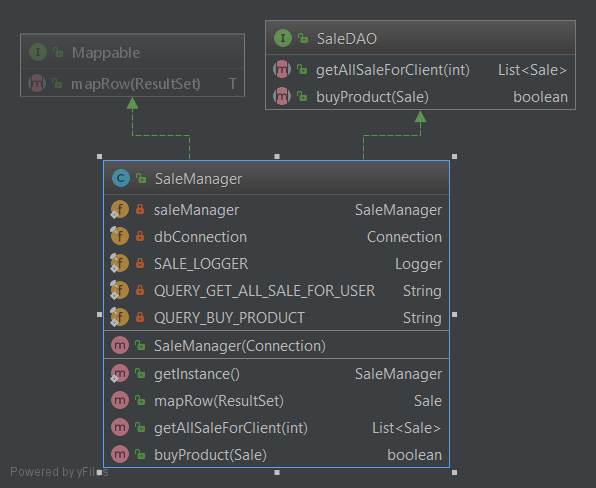
**PaymentManager**



**BookingManager**

****

**SalesManager**



Ogni manager implementa le principali operazioni CRUD, oltre ad altri metodi ad-hoc.

Il tutto sarà gestito tramite Statement parametrizzati onde evitare errori e prevenire attacchi del tipo SQL-Injection. I manager sono implementati tramite il design pattern Singleton.

### Descrizioni con schemi di servlet e controlli

Le servlet saranno ritoccate parzialmente, in quanto sarà necessario modificare le chiamate ai manager. Dal Reverse Engineering effettuato sul software si è dedotto che non solo i controlli sugli input sono quasi inesistenti,ma anche le specifiche sono scarne. Si è deciso quindi di definire delle specifiche per le varie entità. I controlli saranno sia effettuati lato front-end, sia nelle Servlet. Per il front end si è voluto fare affidamento sulla direttive imposte dallo standard HTML5 per la validazione dei dati annotando i campi delle form con i relativi attributi di validazione (pattern ,minlength, maxlength, required) mentre per il back end, si è scelto di utilizzare la specifica **JSR 303 Bean Validation** promossa da Oracle per la validazione dei Java Bean e utilizzando come implementazione di riferimento la libreria **Hibernate-Validator**. Questa scelta è stata approvata perchè porta con sé molti vantaggi:

1. Una dichiarazione della regole di validazione semplice grazie all’ausilio delle Java Annotation definite nella specifica.
2. Un’ alta manutenibilità in quanto è possibile modificare queste regole impattando solo sulle annotazioni poste all’interno dei rispettivi POJO.

**Legenda Controlli**

*Alpha = /^[a-zA-Z]\s+$/*

*AlphaNumber = /^[a-zA-Z0-9]+$/*

*Number = /^[-+]?\d\*(?:[\.\,]\d+)?$/*

*Integer = /^[-+]?\d+$/*

*Email = /^[a-zA-Z0-9.!#$%&'\*+\/=?^\_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)+$/*

**Client**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Length | Required |
| ID | Integer | autoincrement | 11 | YES |
| First Name | String | Alpha | 15 | YES |
| Last Name | String | Alpha | 15 | YES |
| CIF | String | AlphaNumber | 15 | YES |
| Address | String | AlphaNumber & Blank | 100 | YES |
| CAP | Numerical | Integer | 5 | YES |
| City | String | Alpha | 20 | YES |
| Province | String | Alpha | 2 | YES |
| Telephone Number | String | Integer | 12 | NO |
| Mobile Number | String | Integer | 12 | YES |
| Fax | String | Integer | 12 | NO |
| Username | String | AlphaNumber | 15 | YES |
| Password | String | AlphaNumber | min 8 max 20 | YES |
| Email | String | Email | 25 | YES |

**Admin**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| ID | integer | autoincrement | 11 | YES |
| Username | String | AlphaNumber | 20 | YES |
| Password | String | AlphaNumber | 20 | YES |
| FirstName | String | Alpha | 20 | YES |
| Last Name | String | Alpha | 20 | YES |

**Smartphone**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| ID | integer | autoincrement |  | YES |
| Brand | String | Alpha | 20 | YES |
| Model | String | AlphaNumber | 20 | YES |
| Display Inch | Double | Integer | 2.1 | YES |
| OS | String | Alpha | 8 | YES |
| CPU | String | AlphaNumber | 20 | YES |
| RAM | Integer | Integer | 2 | YES |
| Internal Storage | Integer | Integer | 3 | YES |
| Bluetooth | Bool | Boolean |  | \*default |
| LTE | Bool | Boolean |  | \*default |
| Camera | Integer | Integer | 1.1 | YES |
| Price | Double | Number | 6 | YES |
| Quantity | Integer | Integer | 4 | YES |

**Payment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| ID | integer | autoincrement |  | YES |
| Name | String | AlphaNumber | 20 | YES |
| Price | Double | Number | 6 | YES |

**Shipment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| ID | integer | autoincrement |  | YES |
| Name | String | AlphaNumber | 20 | YES |
| Price | Double | Number | 6 | YES |
| Day range | String | Integer+”-” | 2 | YES |

**Booking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| Product | link | link |  | YES |
| Username | link | link |  | YES |
| Datetime | Date | Date | x | YES |
| Quantity | Integer | Integer | 4 | YES |

**Sale**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Specification | Lenght | Required |
| Product | Link | link |  | YES |
| Username | Link | link |  | YES |
| Datetime | Data | Data | x | YES |
| Quantity | Integer | Integer | 4 | YES |
| Price | Double | Number | 8 | YES |
| Shipment Type | link | link |  |  |
| Payment Type | link | link |  |  |

# Schema ER

Lo schema sottostante è il risultato ottenuto durante la fase di ingegnerizzazione del data layer. Come si può notare si è riusciti ad ottenere un compromesso adeguato fra ridondanza delle informazioni ed efficienza di accesso ai dati.

