



**DD**

**Design Document**

**Partecipanti**:

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Valenti Andrea | 0522500403 |
| Papaleo Giuseppe | 0522500465 |
| Tammaro Ruggero |  |

|  |  |
| --- | --- |
| **Scritto da:** | Andrea Valenti, Giuseppe Papaleo |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Sommario

[1. Introduzione 4](#_Toc504567295)

[*1.1 Descrizione del sistema corrente* 4](#_Toc504567296)

[*2. Sistema proposto* 6](#_Toc504567297)

[*2.1 Descrizione* 6](#_Toc504567298)

[*2.2 Design Goal* 7](#_Toc504567299)

[*2.2.1 Performance* 7](#_Toc504567300)

[*2.2.1 Performance* 7](#_Toc504567301)

[*2.2.3 Maintenance* 8](#_Toc504567302)

[*2.2.4 Linee guida implementazione, Packaging* 8](#_Toc504567303)

[Immagine 2.2.4.1 Package diagram del sistema corrente 9](#_Toc504567304)

[Immagine 2.2.4.2 Dettaglio dei package 10](#_Toc504567305)

[*2.3 Architettura* 10](#_Toc504567306)

[Immagine 2.3.1 MVC 11](#_Toc504567307)

[*2.4 Deployment Diagram* 11](#_Toc504567308)

[3. Lista delle Modifiche 12](#_Toc504567309)

[*4. Data Design* 14](#_Toc504567310)

# 

# 1. Introduzione

Lo scopo di questo documento è quello di revisionare il sistema esistente e, ove occorre, progettare quelle che sono le modifiche necessarie allo stesso in base alle informazioni raccolte durante la fase di analisi dei requisiti, nonché a ciò che viene richiesto dalle *Change Requests*.

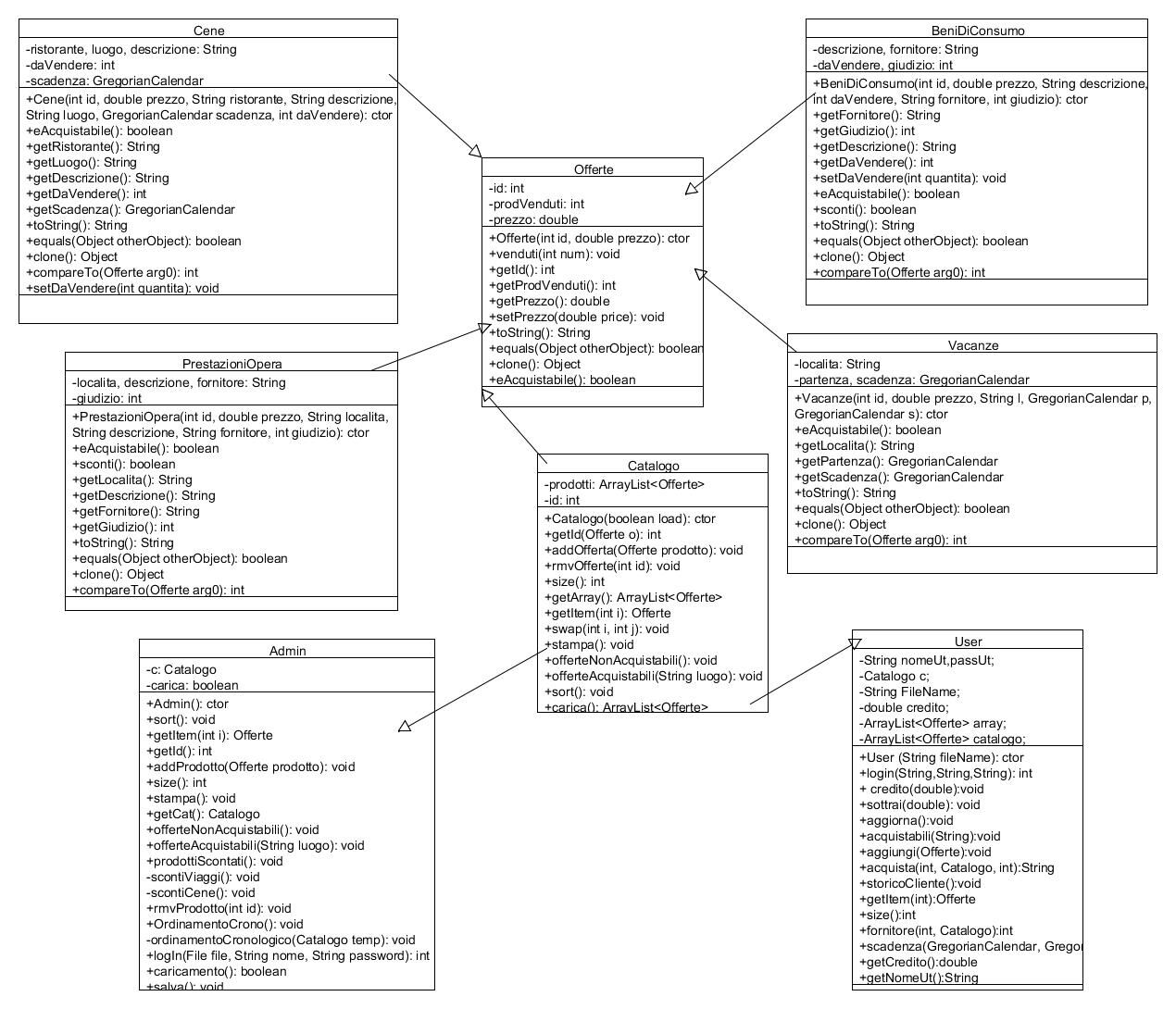
Dopo aver vagliato diverse soluzioni (vedi doc. *Descrizione Ptogetto IGES-v-0-2*) che andavano ad esaminare i documenti in possesso e le possibili tecnologie con cui affrontare la migrazione, è stata scelta la seguente strategia:

* creazione di una interfaccia web ex-novo che va a sostituire quella esistente;
* sostituzione della struttura di file con un DBMS;
* creazione di un layer di comunicazione tra interfaccia e sistema esistente.

## *1.1 Descrizione del sistema corrente*

Lo scopo di base del progetto è quello di creare un’applicazione che sia un ibrido tra Grupon e un e-commerce, poiché prevede che ci sia un admin che inserisce delle offerte di diversa natura, che vanno dalle scarpe fino a vacanze e viaggi. A tutte le offerte si possono applicare determinati sconti, che dipendono dalla quantità di prodotti acquistati oppure da altri fattori che possono essere scelti nel momento dell’inserimento dello sconto. Oltre questa parte di back-end il progetto prevede anche una parte di front-end dove un utente effettua la login e può fare acquisti, gestire il proprio profilo e effettuare delle ricariche di credito utili per le spese. Per l’esattezza il nuovo progetto migrerà la parte di Font-End completamente, e lascerà per uno sviluppo futuro la parte di UI per l’admin.

Dopo una fase di analisi dei documenti a disposizione, sono state analizzate le classi del vecchio progetto per capire quanta della logica originale poteva essere ereditata. Fortunatamente l’intera logica applicativa era presente nelle classi JAVA, e non nell’interfaccia e ciò ha facilitato molto il nostro lavoro. Questo perché le classi JAVA standard, possono essere usate come modelli per un progetto JAVA EE. Così facendo siamo riusciti a portarci dietro molto logica, sostituendo solo la parte che si interfacciava con i file, andando ad inserire un middleware che assicura la consistenza. Il Class model originario era il seguente:



### Immagine 1.1.1 Class diagram del sistema corrente

# *2. Sistema proposto*

## *2.1 Descrizione*

Il sistema da sviluppare avrà come obbiettivo finale quello di creare un prototipo di un ibrido tra un sito di e-commerce e grupon. Tale sistema, in particolare ricostruisce l’intero processo di acquisto, a partire dalla funzione di autenticazione fino al vero e proprio acquisto. Inoltre il sistema offre altre funzioni di contorno come la visione del catalogo, oppure il riepilogo degli acquisti o ancora i propri dati. Tutto ciò sarà possibile sempre previa connessione ad Internet.

Al primo accesso verrà chiesto di inserire un username e una password univoche per cadauna persona, premendo l'apposito pulsante di login; qualora si fosse sprovvisti di queste credenziali, bisognerà richiedere la registrazione dei propri dati all’amministratore di sistema, unico a poter effettuare la fase di registrazione manualmente sul BE. L’utilizzo di un web server con supporto JavaEE è previsto per fornire le funzionalità del sistema attraverso un front-end basato su browser web. Si prevede, inoltre, di realizzare le nuove interfacce utente migliorandole quanto più possibile, al fine di non pregiudicare l’usabilità dell’applicativo e consentire agli utenti un’agevole transizione verso l’utilizzo di interfacce su browser web.

Il processo di implementazione prevederà l’esecuzione in parallelo dello sviluppo dell’ interfaccia utente e del back-end, che dovrà interfacciarsi con la vecchia logica ereditata. Si cercherà, per quanto possibile, di non utilizzare tecnologie differenti rispetto a quelle già in esercizio.

E' stato necessario aggiornare i design goal per il nuovo sistema: la soluzione adottata riguarda l'utilizzo della tecnologia *Java Servlet,* che fornisce agli sviluppatori un linguaggio lato server facile da utilizzare e compatibile con la tecnologia Java ( linguaggio con il quale è stato implementato Prezzi Pazzi) per la creazione di moduli in grado di comunicare con il sistema ereditato. Il design pattern pensato per descrivere ed implementare il nuovo sistema può essere considerato un MVC (Model-View-Controller). Più precisamente, il sistema esistente (privato del suo layer di UI e di Model) comporrà il controller che sarà rappresentato da uno strato intermedio che conterrà i moduli di comunicazione tra la nuova interfaccia utente e il nuovo model di cui discuteremo a breve, accoppiato con delle classi sviluppate ex-novo.

Il livello View verrà implementato utilizzando HTML5,Javascript e pagine JSP utili per sfruttare quanto più possibile la logica ereditata. La scelta di utilizzare lo schema MVC permetterà di minimizzare l’accoppiamento tra i layer del sistema, in modo da permettere che future modifiche non comportino la modifica dei sorgenti che generano i contenuti. Oltre alle tecnologie web di cui si è discusso, è stato scelto di utilizzare un framework JAVAScript-based per velocizzare la creazione degli elementi dell’UI: ReactJS.

## *2.2 Design Goal*

Obiettivo principale del sistema PrezziPazzi è di garantire un modo pratico e veloce per fare acquisti, soprattutto verso chi non possiede un elevato grado di esperienza nell’uso del computer in genere. Esso dovrà fornire semplici interfacce che permetteranno ai clienti di sentirsi a proprio agio con il sistema, garantendo un basso tempo di training.

Segue la descrizione dei design goal previsti per il sistema da realizzare.

## *2.2.1 Performance*

Il sistema dovrà rispondere abbastanza velocemente agli input esterni, mantenendo i tempi di necessari ad eseguire le varie operazioni ridotti il più possibile, considerando anche che alcuni fattori che determinano il rallentamento non sono gestibili e/o migliorabili solo tramite la progettazione del software: un esempio sono i tempi di sistema influenzati dall’interazione con il database e dalla velocità con cui il server soddisfa le richieste; oltretutto i tempi di risposta sono fortemente relazionati alla potenza del computer e alla connessione al Web disponibile per il singolo terminale.

## *2.2.1 Performance*

Il sistema dovrà essere in grado di gestire in maniera efficace tutti gli imprevisti che potrebbero presentarsi al momento del suo utilizzo. La risposta del sistema a questi eventi inaspettati dovrà essere immediata e significativa, in modo tale da garantire il corretto funzionamento in futuro.

Robustezza: il sistema dovrà essere in grado di impedire all’utente di alterare, in maniera accidentale o volontaria, il corretto funzionamento del sistema con, ad esempio, l’inserimento di dati non validi. Tutte le possibili interazioni tra l’utente e il sistema dovranno essere opportunamente controllate e gestite in modo tale da impedire il verificarsi di situazioni anomale.

Disponibilità: l’applicazione prodotta dovrà essere disponibile 24 ore su 24, ad eccezione di una specifica fascia oraria, o una data precisa, che saranno utilizzate per apportare aggiornamenti ed eventuali modifiche o correzioni. Si dovrà garantire all’utente di poter effettuare determinate operazioni all’interno del sistema quando ha la volontà di farle.

Fault tollerance: il sistema dovrà essere in grado di rispondere in maniera adeguata ad eventuali comportamenti errati, impedendo la prosecuzione delle operazioni. Prezzi Pazzi dovrà guidare l’utente nella compilazione dei campi, segnalando eventuali discordanze tra quanto richiesto e quanto fornito in input.

## *2.2.3 Maintenance*

Il sistema dovrà avere tutte le caratteristiche che permettono una semplice ed immediata manutenzione, ovvero la possibilità di apportare cambiamenti di qualsiasi genere senza la necessità di modificare l' organizzazione interna e l' implementazione stessa.

Estendibilità: il sistema dovrà presentare una struttura totalmente modularizzata per poter consentire l'aggiunta di nuove funzionalità, che siano esse richieste dal cliente o dovute alla naturale evoluzione delle regole di business.

Modificabilità: la struttura interna del sistema dovrà essere concepita in modo da consentire di apportare velocemente una o più modifiche, sia che siano esse relative a componenti che hanno generato comportamenti anomali, sia che rispondano alle necessità di qualche miglioramento.

## *2.2.4 Linee guida implementazione, Packaging*

Di seguito vengono riportate le linee guida da utilizzare nel disegno del servlet set:

* i nomi dei packages dovranno iniziare con la lettera minuscola, a differenza dei subpackages, che inizieranno con la maiuscola;
* i nomi delle classi dovranno iniziare tutti con lettera minuscola e continuare con lettere minuscole;
* i metodi per l’accesso e la modifica delle variabili dovranno avere rispettivamente il suffisso get (accesso) e set (modifica, inserimento, cancellazione);
* i commenti alle classi, ai metodi e alle variabili di classe dovranno seguire lo standard JavaDoc, quindi iniziare con /\*\* e terminare con \*\*/.

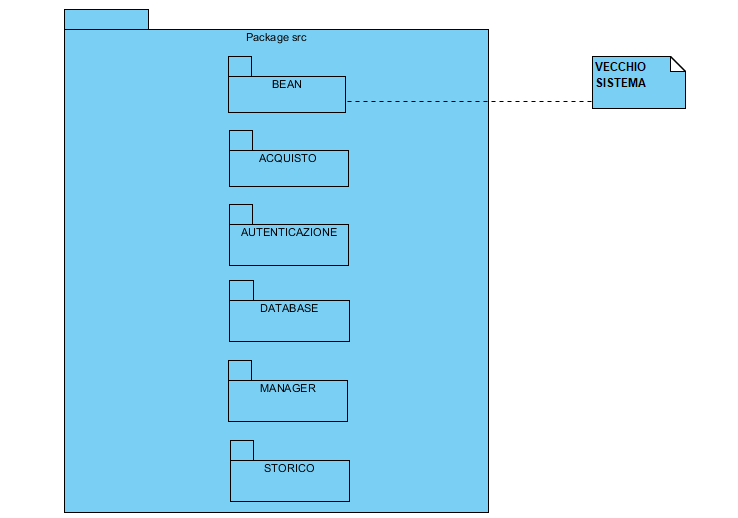
Inoltre, i seguenti tag verranno aggiunti agli artefatti che costituiranno il codice:

@author nome dell’autore (nelle Classi);

@param tipo nome del parametro (nella dichiarazione dei metodi).

2.2.4.1 Packages

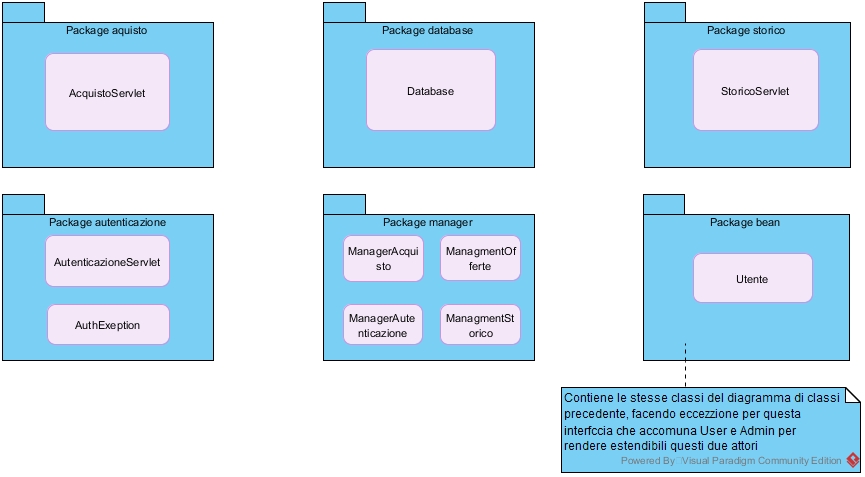
Il progetto sarà suddiviso il packages come segue:



### Immagine 2.2.4.1 Package diagram del sistema corrente

Come è possibile osservare dalla figura Immagine 2.2.4.1, il sistema conterrà diveri packages:

* *Bean*: conterrà il sistema ereditato
* *Acquisto,Autenticazione,Storico*: conterranno le classi (servlet) che andranno a comporrre il Control dell’MVC e sono classi scritte ex-novo. Tali classi si basano con le vecchie classi ereditate contenute in Bean
* *Manager* : conterrà vari strati di middlware che si interfacciano con le relative classi concrete presenti nel model (database)
* *Database*: conterrà la classe per interfacciarsi al DB ed ottenere la connessione in ogni punto del progetto.



### Immagine 2.2.4.2 Dettaglio dei package

## *2.3 Architettura*

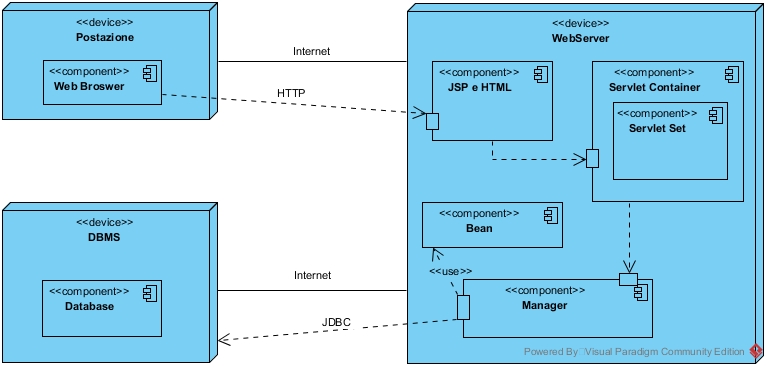
Con la migrazione delle funzionalità ereditate sul web, si è deciso di seguire i paradigmi dell’MVC design pattern, organizzando le risorse come segue: verrà creata una nuova UI che potrà essere visualizzata su un qualunque web browser (**View**). L’UI come già accennato, sarà scritta con JSP e linguaggi web. Attraverso le JSP, si riesce ad utilizzare le entità del sistema ereditato, migrandole verso componenti web. Il sistema esistente, privato della UI originaria e della parte di interfacciamento con i file, costituirà il **Model**. Il **Controller**, che permette la comunicazione tra i due layer precedentemente descritti, è costituito da un modulo contente le servlet, anch’esse modularizzate a seconda dell’operazione che andranno a svolgere. I vantaggi apportati da tale architettura sono quelli di rendere indipendente ogni modulo: tale aspetto fornisce all’intero sistema una facile e semplice manutenibilità e aggiornabilità del software, in particolare per quel che riguarda la parte del sistema che si occupa dell’ accesso all’archivio dei dati.



### Immagine 2.3.1 MVC

## *2.4 Deployment Diagram*

Il seguente Deployment Diagram mostra la struttura del nuovo sistema. Il nodo *Web Server* rappresenta l’omonimo server web nel quale è in esecuzione un servlet container per la gestione e il controllo delle Servlet. Il server in questione ha quindi il compito di gestire le richieste inoltrate dagli utenti attraverso l’interfaccia presentata loro dai browser, contenendo il set delle servlet. L’Application Server contiene il sistema esistente, le cui funzionalità sono chiamate dal servlet set, a seconda delle requests che giungono dal client; sarà l’Application Server stesso a comunicare con il DBMS ed effettuare le operazioni richieste.



# 3. Lista delle Modifiche

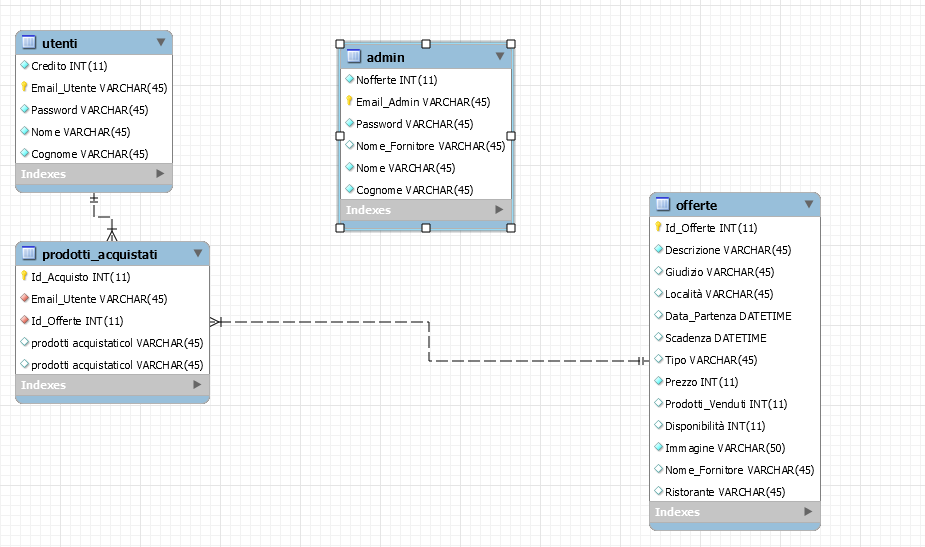
Di seguito è riportata la lista delle modifiche apportate a PrezziPazzi.

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNZIONALITA’** | **Artefatto** | **Nuovi Artefatti** | **Note** |
| AUTENTICAZIONE | Admin.java,  User.java | index.html,  AutenticazioneServlet.java,  AuthException.java  ManagerAutenticazione.java  User.java  Tipoutente.java (semplice enumerazione per individuare utente ADMIN o USER) | Prima di tutto abbiamo apportato una modifica alle classi ereditate Admin e User, facendo si che ereditassero la classe astratta Utente creata ex-novo. Ciò permette di estendere meglio in futuro le gerarchie e accumunare funzioni uguali eliminando ridondanza di codice. Poi abbiamo creato un index.html, in modo tale che questa fosse la prima pagina caricata dall’applicazione e l’utente la potesse usare come prima pagina su cui eseguire l’autenticazione. Inoltre è stata creata una servlet per offrire il servizio di login usato dalla form presente nella pagina index.html. Infine poi abbiamo creato una classe manager che permette di interfacciarsi con il DB per effettuare le query relative all’autenticazione. Per corredare il tutto è stata creata una classe per individuare le eccezioni lanciate dalle classi di autenticazione |
| CATALOGO/ACQIUSTO | Offerte.java  Vacanze.java  Cene.java  PrestazioniOpera.java  BeniDiConsumo.java  Catalogo.java | AcquistoServlet.java  ManagerAcquisto.java  ManagmentOfferte.java  head.html (funzione accessoria per fare un import delle librerie necessarie in tutte le pagine html)  purchase.jsp  home.jsp | Per la parte di acquisto e visualizzazione catalogo innanzitutto, abbiamo modificato la super classe Offerte inserendo altri campi utili per il web, come le immagini ad esempio. Dopo queste modifiche alla vecchia struttura abbiamo eliminato tutti i riferimenti che esistevano tra il catalogo.dat (il vecchio file) e la classe Catalogo. Una volta eseguiti questi interventi preliminari abbiamo scritto una pagina home.JSP che come prima azione, carica dalla sessione il catalogo e lo passa alla parte html. Per gestire la parte di DB che mantiene le informazioni sul catalogo nella tabella offerte, abbiamo scritto un manager chiamato ManagmentOfferte.java. Per la parte di acquisto invece è stato necessario scrivere una servlet AcquistoServlet.java, che offre il servizio che viene richiamato quando l’utente clicca acquista su un prodotto. Anche in questo caso per gestire la parte di DB abbiamo usato un manager chiamato ManagerAcquisto.java . Per offrire un feedback all’utente infine, abbiamo creato la pagina purchase.jsp che riceve in input l’elaborazione della servlet AcqistoServlet, e mostra a video o un messagio di conferma, o la causa dell’errore. |
| STORICO | Catalogo.java | StoricoServlet.java  ManagmentStorico.java | Per la parte di storico acquisti è stata creata una servlet, StoricoServlet, e un manager, ManagmentStorico. La servlet offre il servizio BE che viene invocato quando l’untente preme su storico acquisti, ed essta restituisce le offerte presenti nella tabella offerte Acquistate, attraverso l’uso del manager. L’UI di questa featurse è stata inserita nella pagina home.jsp . |

# 

# *4. Data Design*

Il progetto ereditato per gestire la persistenza dei dati utilizzava dei file salvati sul hard disk, del computer su cui l’applicazione veniva lanciata. Tale scelta è stata completamente abbandonata, poiché tale logica in alcun modo può adottata per un progetto JAVA EE. In tale envoirorment, è necessario l’utilizzo di un Database per rendere i dati fruibili a tutti i dispositivi che li richiedono via internet. Quindi il gruppo si è trovato a dover affrontare tutto il flusso per lo sviluppo di un Database. Quindi il primo step è stato quello di mappare in tabelle le varie entità del sistema creando delle relazioni laddove necessario. Dopo questa fase quindi abbiamo elaborato un diagramma delle tabelle del DB che poi siamo andati ad implementare.



La scelta ce ha creato molti punti di riflessione è stata quella relativa alla tabella Offerte. Questa tabella rappresenta nel vecchio progetto la classe astratta Offerte, tale classe viene poi ereditata ed estesa nelle implementazioni concrete Vacanze,Cene,Prestazioni d’Opera e Beni di Consumo. Questa struttura gerarchica, funziona molto bene all’interno sia di JAVA standard che EE, ma non può essere mappata 1 a 1 nel DB, come invece è stato possibile per il concetto di Utente e Admin. Dopo aver esaminato le possibili scelte (vedi *Descrizione Ptogetto IGES-v-0-2* ) il gruppo ha deciso di accorpare le classi figlie nel padre e avere un’unica tabella offerte con un campo Tipo per identificare più velocemente la natura concreta della riga i-esima. Questo ha portato si alla presenza di più valori null, ma dall’altro lato ha reso la logica dell’applicazione più snella e lineare diminuendo il numero di join necessarie per ottenere una certa informazione. Ciò si tramuta anche in una velocità maggiore nell’esecuzione delle query relative al catalogo. Per tenere traccia dei prodotti acquistati inoltre è stata creata una relazione M a N tra offerte e utente. Questo perché un utente può acquistare N prodotti e un prodotto può essere acquistato da più utenti. Tale relazione viene mappata in una nuova tabella che abbiamo chiamato prodotti acquistati. Finita questa fase di progettazione, per implementare interfacciarci con il DB abbiamo scelto tra le possibili tecnologie JDBC. L’utilizzo di JDBC è presente nella classe Database.java, che prima di tutto se non esiste già una connessione col DB la istanzia, e poi la restituisce a chi la richiede. Inoltre per gestire in maniera efficiente le connessioni istanziate e ridurre i tempi di acceso al DB è stata utilizzata una tecnica di Connection Polling. Infine è stato scelto per gestire il Database il DBMS MySQL.