



UNIVERSITÀ DEGLI STUDI DI SALERNO

Ingegneria del Software

OBJECT DESIGN DOCUMENT

IL MERCANTE



DI LIBRI

Anno Accademico 2019/2020

PROGETTO: “Il Mercante di Libri”	Versione 1.0
DOCUMENTO: OBJECT DESIGN DOCUMENT	Data: 10/10/19

Partecipanti

Nome	Matricola
Abbate Giorgia	0512104544
Stanescu Tania	0512104493

Revisione History

Data	Versione	Descrizione	Autore
10/10/19	1.0	Prima revisione del documento	Abbate/Stanesco

INDICE

1.Introduzione

1.1 Object Design Trade-offs

1.2 Linee Guida per la Documentazione delle Interfacce

1.3 Definizioni, acronimi e abbreviazioni

1.4 Riferimenti

2. Design pattern

2.1. Packages

3.Class Interfaces

4. Glossario

1. Introduzione

1.1 Object Design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:

Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

Prestazioni vs Costi:

Essendo il progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, per alcune funzionalità verranno utilizzati dei template open source esterni in particolare Bootstrap.

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa, fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:

La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

Naming Convention

- E' buona norma utilizzare nomi:

1. Descrittivi
2. Pronunciabili
3. Di uso comune
4. Lunghezza medio-corta
5. Non abbreviati
6. Evitando la notazione ungherese
7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.
- E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore “_”, ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Metodi:

- I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo `getNomeVariabile()` e `setNomeVariabile()`. Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitare la leggibilità. Esistono però casi particolari come ad esempio nell'implementazione dei model, dove viene utilizzata l'interfaccia CRUD.

Esempio: `getUsername()`, `setUsername()`

- I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

Classi e pagine :

- I nomi delle classi e delle pagine devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo.
- La dichiarazione di classe caratterizzata da:
 1. Dichiarazione della classe pubblica
 2. Dichiarazioni di costanti
 3. Dichiarazioni di variabili di classe
 4. Dichiarazioni di variabili d'istanza
 5. Costruttore
 6. Commento e dichiarazione metodi.

1.3 Definizioni, acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- CRUD: Create Read Update Delete

Abbreviazioni:

- DB: DataBase

1.4 Riferimenti

- Documento SDD del progetto Il Mercante Di Libri
- Documento RAD del progetto Il Mercante Di Libri
- Documento DatiPersistenti_ Il Mercante Di Libri

2. Design pattern

2.1. Packages

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

- *Interface layer*
- *Application Logic layer*
- *Storage layer*

Il package Il Mercante Di Libri contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Interface layer

Rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati.

Application Logic layer

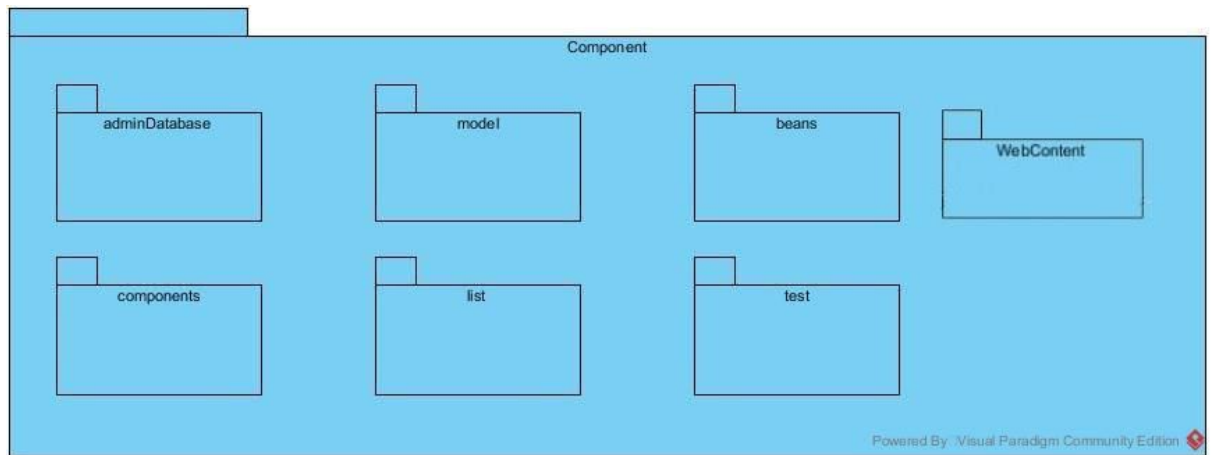
Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali:

- **Gestione Autenticazione**
- **Gestione Registrazioni**
- **Gestione Libri**
- **Gestione Utenti**
- **Gestione Ordini**

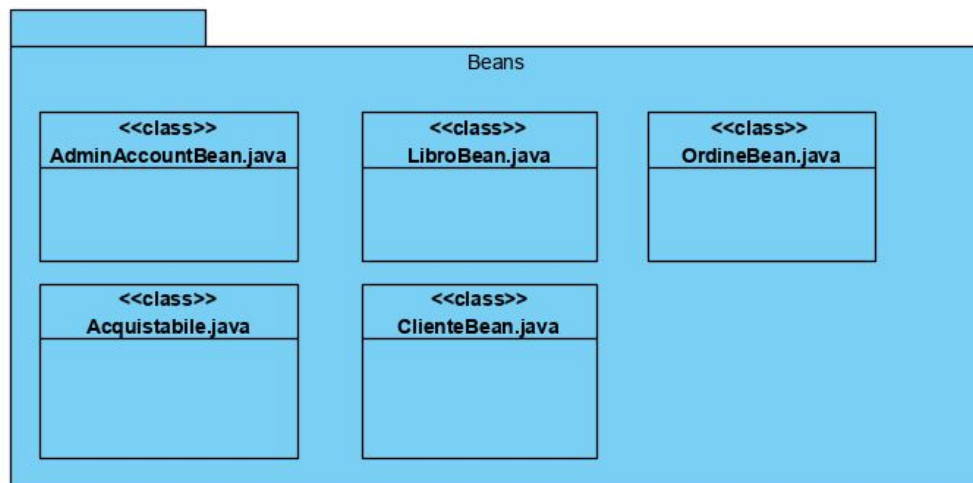
Storage layer

Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dall' Application Logic layer inoltrandole al DBMS e restituendo i dati richiesti.

2.1.0 Package Component- Application Layer

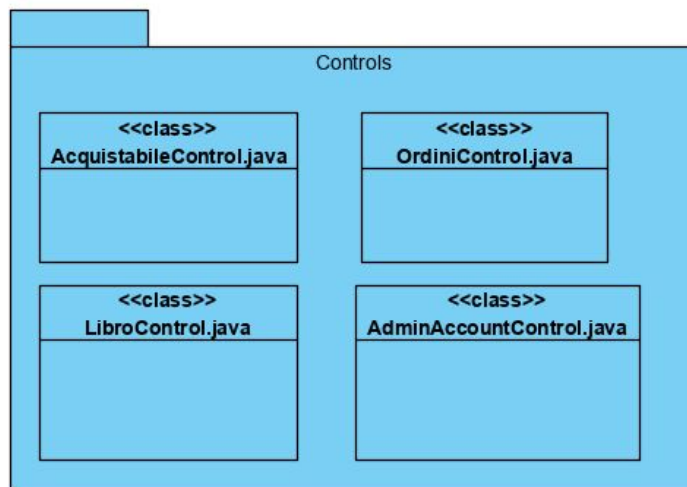


2.1.1 Package beans



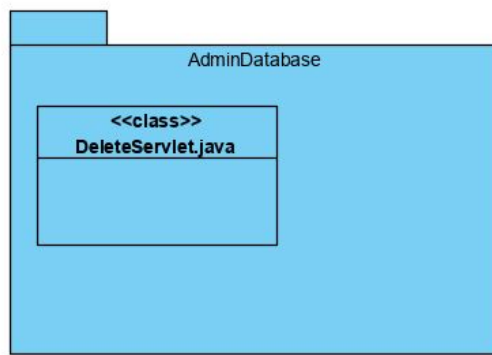
Classe	Descrizione
AdminAccountBean.java	Descrive un amministratore registrato al sistema.
OrdineBean.java	Descrive un ordine effettuato.
Acquistabile.java	Descrive un'interfaccia che permette l'utilizzo dei metodi della classe LibroBean.java
LibroBean.java	Descrive il prodotto presente nel sistema
ClienteBean.java	Descrive le credenziali del cliente presente nel sistema

2.1.2 Package controls



Classe	Descrizione
AcquistabileControl.java	Controller che consente di effettuare l'ordine
OrdineControl.java	Controller che permette l'acquisto dell'ordine o la sua eliminazione
AdminAccountControl.java	Controller che consente di verificare le credenziali dell'admin
LibroControl.java	Controller che consente l'aggiunta del prodotto al carrello

2.1.3 Package AdminDatabase



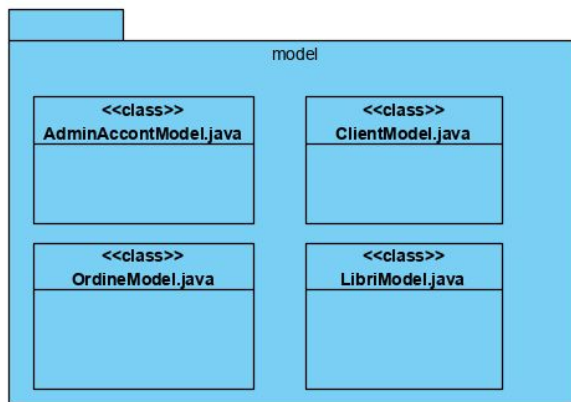
Classe	Descrizione
DeleteServlet.java	Consente di eliminare elementi dalle tabelle

2.1.4 Package List



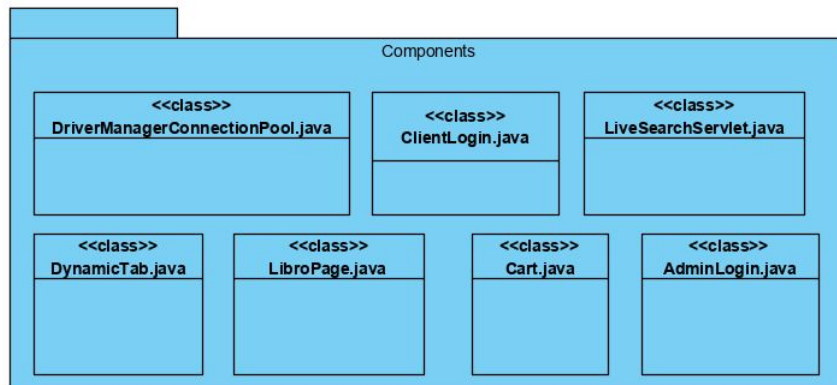
Classe	Descrizione
AllLibriList.java	Permette di generare la lista di libri, consentendone la ricerca

2.1.5 Package Model(Manager)



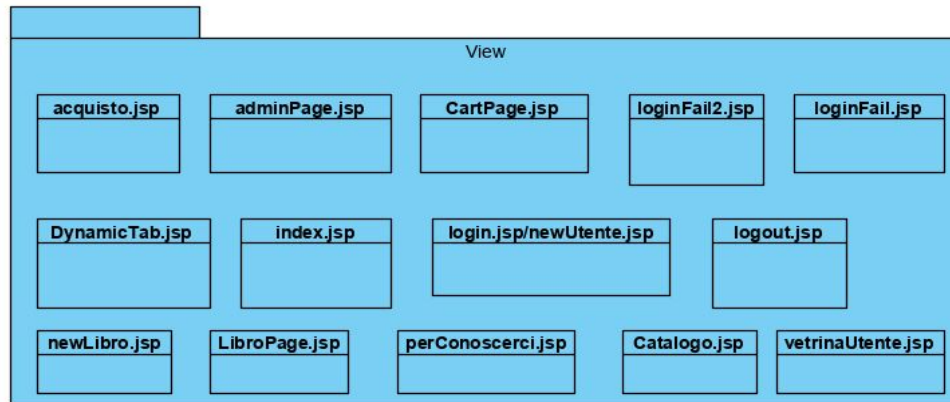
Classe	Descrizione
AdminAccountModel.java	Il model che effettua tutte le query interfacciandosi con il database a cui è connesso, riguardanti le funzionalità degli admin
ClientModel.java	Il model che effettua tutte le query interfacciandosi con il database a cui è connesso, riguardanti le funzionalità del cliente
OrdineModel.java	Il model che effettua tutte le query interfacciandosi con il database a cui è connesso, riguardanti le funzionalità dell'ordine
LibriModel.java	Il model che effettua tutte le query interfacciandosi con il database a cui è connesso, riguardanti le funzionalità del prodotto

2.1.6 Package Components



Classe	Descrizione
DriverManagerConnectionPool.java	La classe che permette la creazione di una connessione con il database se non istanziata precedentemente oppure restituisce la connessione esistente
ClientLogin.java	Gestisce il login dei clienti.
LiveSearchServlet.java	La classe che consente di cercare il libro all'interno del sistema tramite la parola completa o una lettera presente nel nome del libro
DynamicTab.java	La classe che consente l'aggiunta dinamica di righe alle tabelle presenti nel sistema
Cart.java	La classe che consente l'aggiunta di libri al carrello
LibroPage.java	La classe che consente l'inserimento delle informazioni del prodotto nel database
AdminLogin.java	Gestisce il login da parte di un admin.

2.1.7 Package WebContent-Presentation Layer



Classe	Descrizione
Acquisto.jsp	View che mostra un messaggio che ti notifica che l'ordine è avvenuto con successo.
adminPage.jsp	View che mostra la Zona Riservata dell'admin.
CartPage.jsp	View che mostra il carrello del sistema.
DynamicTable.jsp	View che mostra le tabelle che contengono le informazioni delle varie gestioni del sistema.
index.jsp	View che mostra la Home del sistema.
login.jsp/newUtente.jsp	View che consente all'admin o al cliente di loggarsi al sistema e/o a un nuovo utente di registrarsi
logout.jsp	View che mostra il corretto logout dal sistema.
loginFail.jsp	View che mostra un messaggio di errore se i dati inseriti non sono corretti.
loginFail2.jsp	View che mostra un messaggio di errore se l'username inserita per un nuovo utente è già presente nel sistema e rimanda alla pagina di ogin.jsp/newUtente.jsp
newLibro.jsp	View che consente di inserire un nuovo libro al sistema.
libroPage.jsp	View che mostra le informazioni sul libro
perConoscerci.jsp	View che mostra ulteriori informazioni sul sito
catalogo.jsp	View che mostra la lista del libri
vetrinaUtente.jsp	View che mostra i libri da vendere contenuti nella vetrina degli utenti.

3.Class Interfaces

Package beans

AdminAccountBean

- function AdminAccountBean()**:costruttore della classe;
- function getUsername()**: questo metodo ritorna l'username dell'admin;
- function setUsername(String username)**: questo metodo consente di modificare l'username dell'admin;
- function getPassword()**: questo metodo ritorna la password dell'admin;
- function setPassword(String psw)**: questo metodo consente di modificare la password dell'admin;
- function toString()**: questo metodo restituisce una stringa formata dal nome della classe, username e password dell'admin

Acquistabile(Interfaccia)

Eredita tutti i metodi della classe LibroBean

- function getCodice()**:questo metodo restituisce il codice del libro;
- function setCodice(int codice)**:questo metodo consente di modificare il codice del libro;
- function getNome()**:questo metodo restituisce il nome del libro;
- function setNome(String nome)**: questo metodo consente di modificare il nome del libro;
- function getEditore()**: questo metodo restituisce l'editore del libro;
- function setEditore(String editore)**: questo metodo consente di modificare l'editore del libro;
- function getPrezzo()**: questo metodo restituisce il prezzo del libro;
- function setPrezzo(double prezzo)**: questo metodo consente di modificare il prezzo del libro;
- function toString()**: questo metodo restituisce una stringa formata dal nome della classe, codice, nome,editore e prezzo;

ClientBean

- function ClienteBean()**:costruttore della classe;
- function getEmail()**: questo metodo ritorna l'username del cliente;
- function setEmail(String email)**: questo metodo consente di modificare l'email del cliente;
- function getNome()**: questo metodo ritorna il nome del cliente;
- function setNome(String nome)**: questo metodo consente di modificare il nome del cliente;
- function getCognome()**: questo metodo ritorna il cognome del cliente;
- function setCognome(String cognome)**:questo metodo consente di modificare il cognome del cliente;
- function getPassword()**: questo metodo ritorna la password;
- function setPassword(String psw)**: questo metodo consente di modificare la password dell'admin;
- **function getIndirizzo()** restituisce l'indirizzo completo dell'utente;
- **function getCAP()**: restituisce il CAP;
- **function setCAP()**: consente di modificare il CAP;
- **function getVia()**: restituisce la Via;
- **function setVia()**: consente di modificare il campo Via;
- **function getCittà()**: restituisce la città;
- **function setCittà()**: consente di modificare la città;
- **function setNrCivico()**: consente di modificare il campo numero civico;
- **function getNrCivico()**: restituisce il numero civico;
- function toString()**: questo metodo restituisce una stringa formata dal nome della classe, email, nome, cognome, password, indirizzo;

LibroBean

- funcion LibroBean()**:costruttore della classe;
- function getCodice()**:questo metodo restituisce il codice del libro;
- function setCodice(int codice)**:questo metodo consente di modificare il codice del libro;
- function getNome()**:questo metodo restituisce il nome del libro;
- function setNome(String nome)**: questo metodo consente di modificare il nome del libro;
- function getEditore()**: questo metodo restituisce l'editore del libro;
- function setEditore(String editore)**: questo metodo consente di modificare l'editore del libro;
- function getPrezzo()**: questo metodo restituisce il prezzo del libro;
- function setPrezzo(double prezzo)**: questo metodo consente di modificare il prezzo del libro;
- function toString()**: questo metodo restituisce una stringa formata dal nome della classe, codice, nome,editore e prezzo;

OrdineBean

- function OrdineBean()**:costruttore della classe;
- function getCodice()**: questo metodo ritorna il codice dell'ordine;
- function setCodice(int codice)**: questo metodo setta il codice dell'ordine;
- function assegnaCodice()**: questo metodo assegna il codice dell'ordine ad uno specifico utente;
- function getEmailCliente()**: questo metodo ritorna l'username del cliente che ha effettuato l'ordine;
- function setEmailCliente(String emailCliente)**: questo metodo consente di modificare l'username del cliente che ha effettuato l'ordine;
- function getPrezzo()**:questo metodo ritorna il prezzo dell'ordine;
- function getLibriOrdine()**:questo metodo ritorna una lista di prodotti che possono essere acquistati;
- function addLibro(Acquistabile Libro)**:questo metodo consente di aggiungere uno o più libri ad un ordine;
- function removeAll()**:questo metodo consente di eliminare tutti i libri presenti in un ordine;
- function toString()**: questo metodo restituisce una stringa formata dal nome della classe, codice, username cliente, prezzo, e la lista di prodotti inseriti nell'ordine, con tutte le caratteristiche;
- function deleteProduct(Acquistabile libro)**:questo metodo consente di eliminare un libro da un ordine.

Package model (Manager)

AdminAccountModel

- function doSave(AdminAccountBean account)**: genera la query INSERT per salvare un nuovo elemento admin all'interno del db;
- function doDeleteString(String code)**:genera la query DELETE per eliminare la riga identificata dall'username all'interno del db;
- function doRetrieveAccountByName(String userIn)**: genera una query SELECT * per ricevere i dati in base ad un determinato username;
- function doRetrieveAllAccount(String order)**: genera la query SELECT * per cercare utenti all'interno del db mediante username o psw;

ClientModel

- function doSave(ClienteBean cliente)**: questo metodo consente di salvare un utente;
- function doDeleteString(String code)**:questo metodo consente di eliminare un cliente mediante l'email corrispondente;
- function doRetrieveClientByEmail(String user)**:questo metodo consente di ricercare un cliente mediante email;
- function doRetrieveAllClient()**:questo metodo consente di ricercare tutti i clienti;

OrdineModel

-function doSave(OrdineBean ordine):consente di salvare un ordine effettuato da un utente, in seguito all'aggiunta di uno o più prodotti;

-function doDeleteInt(int code):consente di eliminare l'ordine effettuato da un cliente, tramite il codice corrispondente;

-function assegnaCodice():consente di assegnare il codice ad un ordine corrispondente ad un determinato cliente.

LibroModel

-function doAddProduct(LibroBean product):consente di aggiungere un libro al sistema;

-function doSave(LibroBean product):consente di salvare un prodotto all'interno del sistema o nell'ordine;

-function doDeleteInt(int code):consente di eliminare un prodotto tramite il suo codice;

-function doRetrieveBookByName(String nome):consente di cercare i libri tramite il nome;

-function doRetrieveAllBooks():consente di cercare tutti i libri presenti nel sistema;

-function doRetrieveBooksByAutor(String tipo): consente di cercare i prodotti tramite l'autore a cui appartengono;

-function doRetrieveBookByKey(int isbn):consente di cercare i libri tramite l'isbn;

-function doRetrieveBookBySearch(String nome):consente di cercare un determinato libro, tramite nome, all'interno del sistema.

4. Glossario

Il Mercante di Libri: Nome del sistema

Query: Termine utilizzato per indicare l'interrogazione da parte di un utente di un database.

Cookie: File di informazioni che i siti web memorizzano sul computer dell'utente di Internet durante la navigazione.