Object Design Document
Dante’s Library

Cognome	*Nome*	*Matricola*
Buongusto	Andrea	0512105272
Salierno	Marco	0512105332

Indice

1. Introduzione

1.1 Object design trade-offs

I compromessi di progettazione sorti durante questa fase sono i seguenti:

Comprensibilità vs Tempo
Il codice dev'essere comprensibile e rispettare le convenzioni concordate, commentando le porzioni di codice che risultano essere meno chiare. In questo modo i futuri sviluppatori esterni al progetto potranno comunque comprendere quanto implementato finora.

Usabilità vs Funzionalità
Il sistema è progettato per essere semplice ed user-friendly, rinunciando all'aggiunta di funzionalità più complesse e meno necessarie rispetto a quelle già presenti.

Robustezza vs Costi
Essendo il sistema una piattaforma online, e che potrebbe potenzialmente essere utilizzata da una larga utenza, si preferisce una maggiore affidabilità rispetto ad un sistema economico e meno sicuro.

1.2 Componenti off-the-shelf

Durante l'implementazione verranno utilizzate solo piccole componenti off-the-shelf. Per ottenere alcuni semplici effetti visivi come lo "sliding" utilizzeremo JQuery, una libreria scritta in Javascript.

Inoltre per facilitare l'implementazione della funzionalità della selezione di date (per la prenotazione) utilizzeremo il selettore di date (datepicker) offerto dalla libreria JQuery-UI.

Per la criptazione delle password verrà utilizzata una funzione di hashing chiamata B-crypt scritta in linguaggio Java. In questo modo non verranno salvate password in chiaro all'interno del database.

1.3 Linee guida per la documentazione dell'interfaccia

Per l'implementazione verranno tenute in considerazione molteplici convenzioni a seconda del tipo di file e linguaggio con il quale si sta lavorando. Di seguito vengono illustrate le convenzioni nel dettaglio:

1.3.1 Java

```
public String getDescription() {
    return description;
}
```

Esempio di un metodo

- La parentesi graffa di apertura del corpo di un metodo si trova sulla stessa riga della signature.
- La parentesi graffa di chiusura si trova invece allo stesso livello di indentazione dell'inizio della dichiarazione del metodo (nell'esempio, sulla stessa linea verticale del carattere 'p' di public).
- Le istruzioni racchiuse all'interno di un blocco di istruzioni (delimitato dalle parentesi graffe { e }), devono avere almeno una tabulazione.
- Per quanto riguarda i nomi delle classi e dei metodi, viene rispettata la "notazione a cammello" o più comunemente conosciuta come CamelCase.
- In caso un nome di una variabile è composta da due o più sostantivi, questi saranno separati da un carattere "_".
- Devono esserci commenti per la documentazione Javadoc.

1.3.2 JSP

```
<%@ include file="./jsp/layout/navbar.jsp" %>

<% if(session.getAttribute("user") == null) {
    response.sendRedirect("login.jsp"); /*Se l'utente non è autenticato viene
                                        reindirizzato alla pagina di login*/
    return;
}
%>
```

- Il tag di apertura (<%) si trova all'inizio di una riga;
- Il tag di chiusura (%>) è seguito immediatamente dalla fine della propria riga;

1.3.3 HTML

Il codice HTML definito all'interno dei file JSP deve rispettare lo standard HTML5 ed essere indentato come mostrato nel seguente esempio:

```
<form id="sign-form" class="box" method="post">
    <label for="email">Email</label>
    <input id="email" type="text" name="email"/>
    <label for="password">Password</label>
    <input id="password" type="password" name="password"/>
    <a href="./reset_password" style="font-size: 13px; color: #0099ff">Password dimenticata?</a>
    <button type="submit" formaction="login">Accedi</button>
</form>
```

1.3.4 Javascript

```
<script>
function aFunction(data) {
    var x = 10;
    x += data;
    console.log(x);
}
</script>
```

- Il codice scritto in Javascript rispetta le stesse convenzioni del codice Java.

1.3.4 CSS

```
/*******************************************
* Sezione per form di login e registrazione *
*******************************************/
#form-container {
    padding: 10px;
    max-width: 400px;
    margin: 0 auto;
}
#form-container h2 {
    margin: 0;
}
.box {
    width: 100%;
    margin: 5px 0;
    padding: 15px;
    background-color: #ffffff;
    box-shadow: 0px 0px 2px 1px #ccc;
    border-radius: 4px;
    display: inline-block
}
```

Le convenzioni adottate per il foglio di stile CSS sono le seguenti:

- Ciascun selettore deve iniziare dalla colonna 0 del file.
- Nel caso in cui si sta definendo una media query, allora il selettore deve essere preceduto da un carattere di tabulazione.
- La parentesi graffa di apertura del corpo di una regola si trova sulla stessa riga del selettore.
  La parentesi graffa di chiusura si trova invece allo stesso livello di indentazione dell'inizio della dichiarazione della regola.

- All’interno di una regola, ogni proprietà che la costituisce va messa in una nuova riga, indentata rispetto al suo selettore.
- I selettori che applicano regole a livello globale (senza specificare id o classi) vanno posizionati nella parte iniziale del documento.
- Vi deve essere un commento al di sopra di ogni gruppo di regole che definiscono lo stile di una funzionalità del sito.

1.4 Design Patterns

MVC
Il sistema Dante’s Library utilizza come pattern architetturale MVC.
Il Model-View-Controller (MVC) è un modello che separa un'applicazione in tre componenti logici principali: il Model, la View ed il Controller. Ognuno di questi componenti è creato per gestire specifici aspetti di sviluppo di un'applicazione: il Model corrisponde a tutta la logica relativa ai dati con cui l'utente lavora, la View viene utilizzata per tutto ciò che riguarda il livello di presentazione mentre il Controller agisce da interfaccia tra i componenti Model e View per: elaborare tutta la logica di business; gestire le richieste da parte degli utenti; manipolare i dati utilizzando il componente Model per produrre le risposte degli utenti. Nel progetto queste classi verranno divise in package. Ognuno conterrà quindi delle classi che implementano funzionalità relative al package di appartenenza.

1.5 Acronimi e abbreviazioni

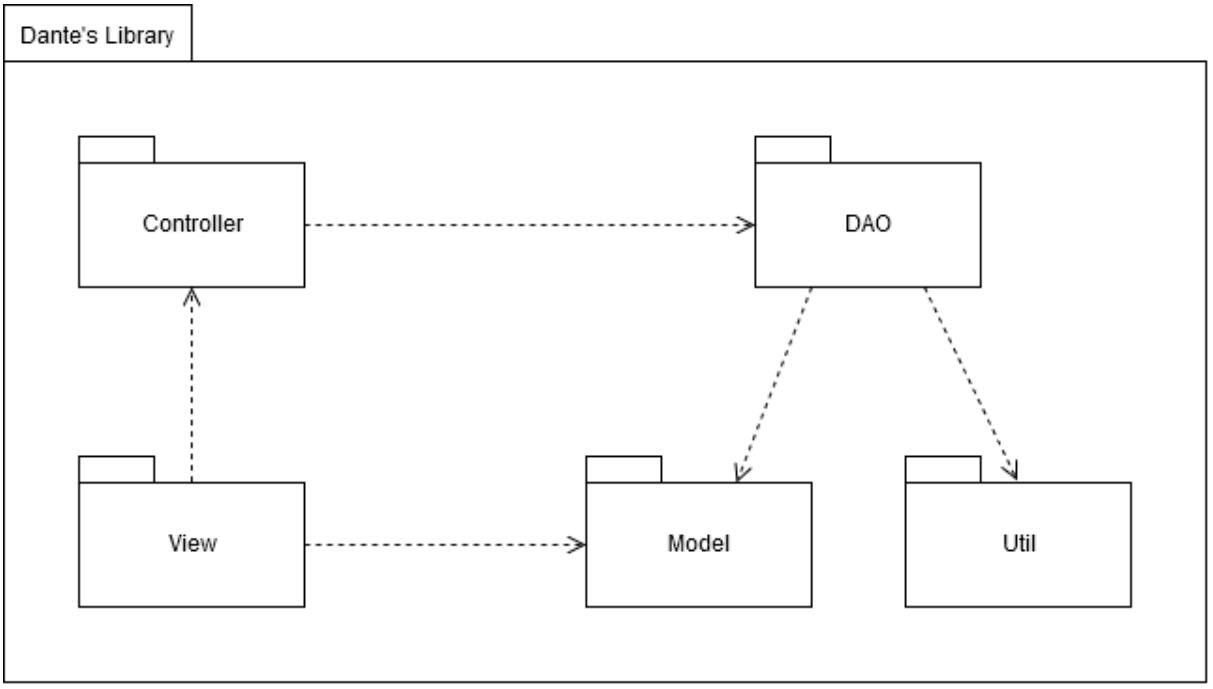
JSP : JavaServer Pages
CSS: Cascading Style Sheets
HTML: HyperText Markup Language
MVC: Model-View-Controller

1.6 Riferimenti

- Problem Statement
- Use Case And Requirements
- RAD (Requirements Analysis Document)
- SDD (System Design Document)

2. Packages

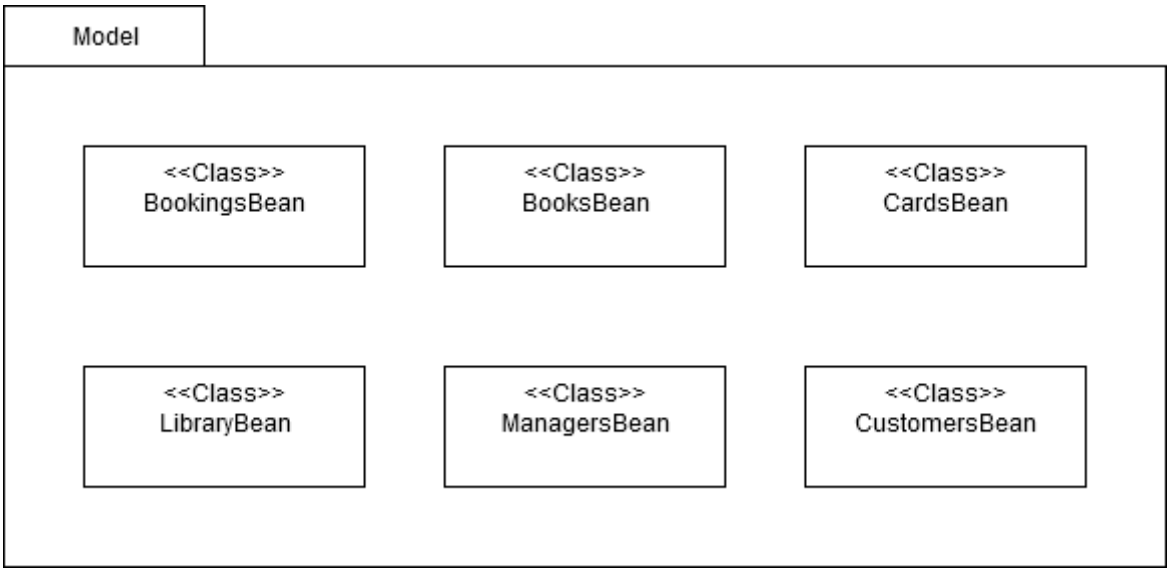
2.1 Panoramica del sistema



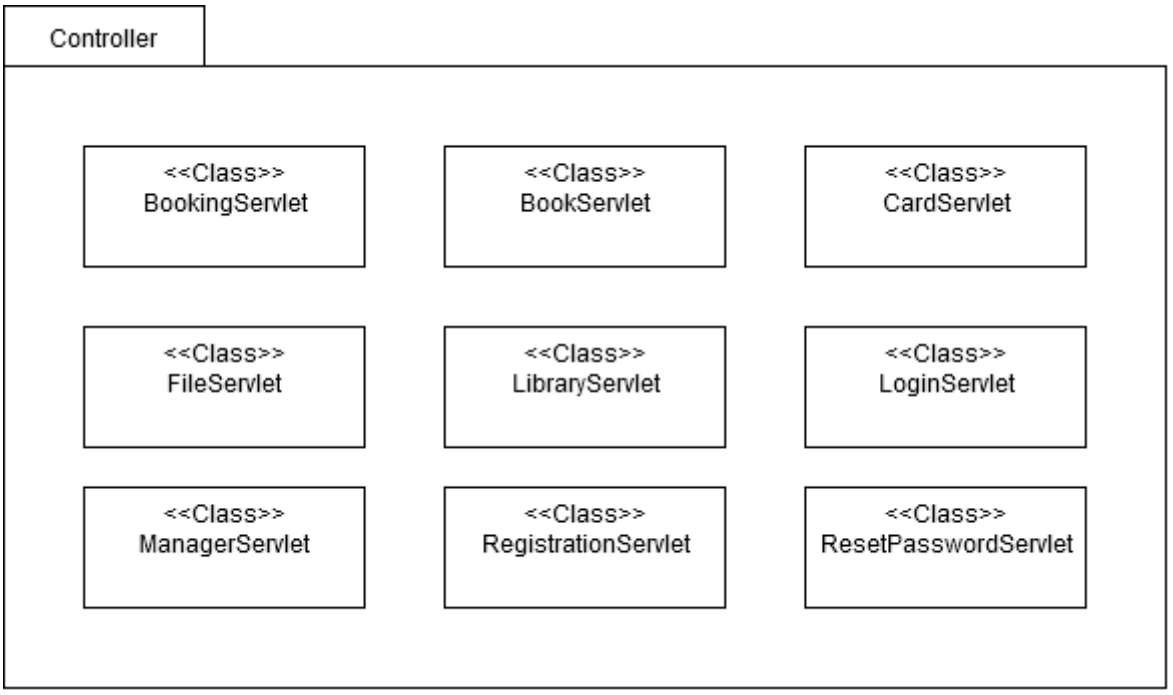
Il sistema comprende i package “Model”, “Controller”, “View”, “DAO” e “Util”.

- Model: comprende le classi Bean, ossia semplici classi Java utilizzate per contenere delle informazioni.
- Controller: comprende Servlet Java per gestire richieste, elaborare informazioni e fornire risposte.
- View: comprende le pagine JSP che realizzano il livello di presentazione, ossia ciò con cui l’utente può interagire.
- DAO: contiene le classi Java responsabili dell’interfacciamento con il Data Layer.
- Util: contiene una classe che specifica le informazioni per la connessione al Database, una classe contenente i parametri di configurazione per l’invio di email, una classe metodi per il controllo del formato degli input e una classe che offre metodi per la cifratura di stringhe (algoritmo di hashing BCrypt).

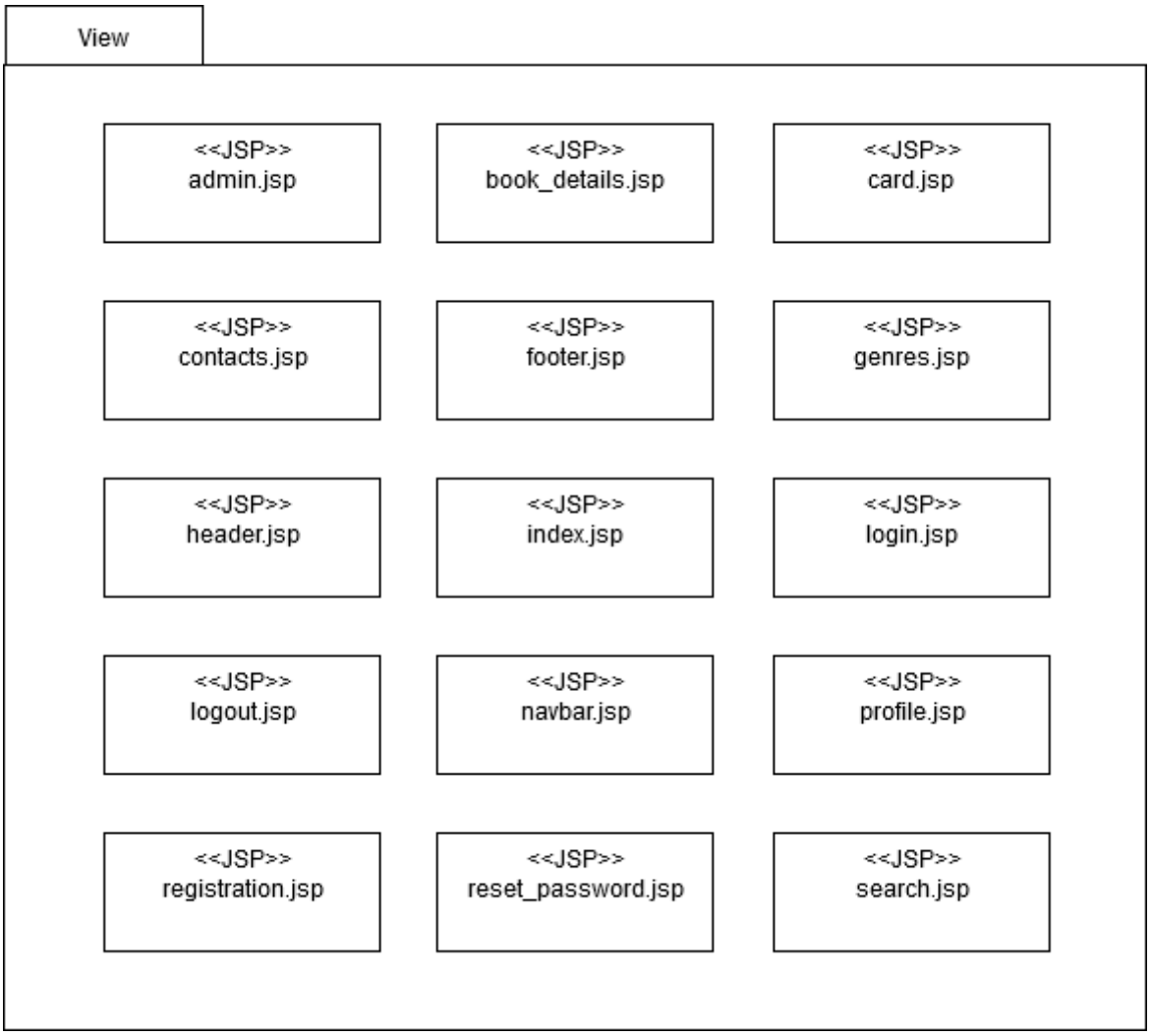
2.2 Model



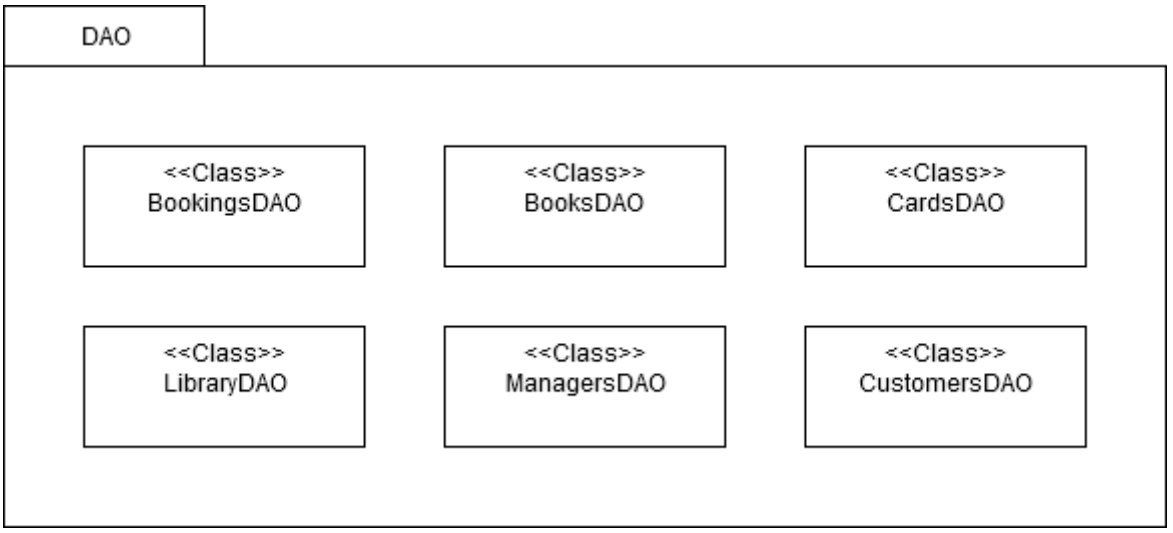
2.3 Controller



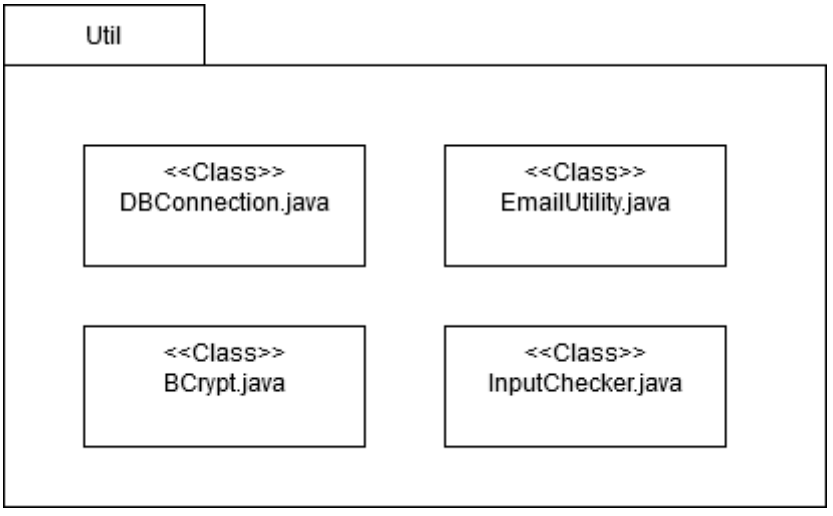
2.4 View


View
<<JSP>>
admin.jsp
<<JSP>>
book_details.jsp
<<JSP>>
card.jsp
<<JSP>>
contacts.jsp
<<JSP>>
footer.jsp
<<JSP>>
genres.jsp
<<JSP>>
header.jsp
<<JSP>>
index.jsp
<<JSP>>
login.jsp
<<JSP>>
logout.jsp
<<JSP>>
navbar.jsp
<<JSP>>
profile.jsp
<<JSP>>
registration.jsp
<<JSP>>
reset_password.jsp
<<JSP>>
search.jsp

2.5 DAO



2.6 Util



3. Class Interfaces

Per maggiori approfondimenti consultare la documentazione Javadoc allegata al progetto.

3.1 Model

Nome classe	BookingsBean.java
Descrizione	Classe che definisce l'entità Prenotazione.
Attributi e Metodi	+ booking_id : int + start_date : LocalDate + end_date : LocalDate + state_name : String + email : String + card_id : int + book_id : int + codice_fiscale : String + title : String + getBooking_id() : int + setBooking_id(booking_id : int) : void + getStart_date() : LocalDate + setStart_date(start_date : LocalDate) : void + getEnd_date() : LocalDate + setEnd_date(end_date : LocalDate) : void + getState_name() : String + setState_name(state_name : String) : void + getEmail() : String + setEmail(email : String) : void + getCard_id() : int + setCard_id(card_id : int) : void + getCodice_fiscale() : String + setCodice_fiscale(codice_fiscale : String) : void + getTitle(): String + setTitle(title : String) : void
Precondizioni	
Postcondizioni	
Invarianti	

Nome classe	BooksBean.java
Descrizione	Classe che definisce l'entità Libro.
Attributi e Metodi	+ book_id : int + title : String + description : String + publisher : String + quantity : int + cover : String + authors : ArrayList<String> + genres : ArrayList<String> + getBook_id() : int + setBook_id(book_id : int) :void + getTitle() : String + setTitle(title : String) : void + getDescription() : String + setDescription(description : String) : void + getPublisher() : String + setPublisher(publisher : String) : void + getQuantity() : int + setQuantity(quantity : int) : void + getCover() : String + setCover(cover : String) : void + getAuthors() : ArrayList<String> + setAuthors(authors : ArrayList<String>) : void + getGenres() : ArrayList<String> + setGenres(genres : ArrayList<String>) : void
Precondizioni	
Postcondizioni	
Invarianti	**context** BooksBean inv:self.getQuantity() >= 0

Nome classe	CardsBean.java
Descrizione	Classe che definisce l'entità Tessera.
Attributi e Metodi	+ card_id : int + codice_fiscale : String + associated : boolean + name : String + surname : String + email : String + getCard_id() : int + setCard_id(card_id : int) : void + getCodice_fiscale() : String + setCodice_fiscale(codice_fiscale : String) : void + isAssociated() : boolean + setAssociated(associated : boolean) : void + getName() : String + setName(name : String) : void + getSurname() : String + setSurname(surname : String) : void + getEmail() : String + setEmail(email : String) : void
Precondizioni	
Postcondizioni	
Invarianti	

Nome classe	LibraryBean.java
Descrizione	Classe che definisce l’entità Biblioteca.
Attributi e Metodi	+ name : String + logo : String + contacts : String + getName() : String + setName(name : String) : void + getLogo() : String + setLogo(logo : String) : void + getContacts() : String + setContacts(contacts : String) : void
Precondizioni	
Postcondizioni	
Invarianti	

Nome classe	ManagersBean.java
Descrizione	Classe che definisce l'entità Gestore.
Attributi e Metodi	+ email : String + password : String + name : String + surname : String + address : String + phone : String + roles : ArrayList<String> + getEmail() : String + setEmail(email : String) : void + getPassword() : String + setPassword(password : String) : void + getName() : String + setName(name : String) : void + getSurname() : String + setSurname(surname : String) : void + getAddress() : String + setAddress(address : String) : void + getPhone() : String + setPhone(phone : String) : void + getRoles() : ArrayList<String> + setRoles(roles : ArrayList<String>) : void
Precondizioni	
Postcondizioni	
Invarianti	

Nome classe	CustomersBean.java
Descrizione	Classe che definisce l'entità Cliente.
Attributi e Metodi	+ name : String + surname : String + email : String + password : String + codice_fiscale : String + address : String + getName() : String + setName(name : String) : void + getSurname() : String + setSurname(surname : String) : void + getEmail() : String + setEmail(email : String) : void + getPassword() : String + setPassword(password : String) : void + getCodice_fiscale() : String + setCodice_fiscale(codice_fiscale : String) : void + getAddress() : String + setAddress(address : String) : void
Precondizioni	
Postcondizioni	
Invarianti	

3.2 Controller

Nome classe	BookingServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti le Prenotazioni e che produce output da inviare come risposta.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** BookingServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Pre** : CustomersBean cutomer = session.getAttribute("customer") <> null **Pre:** IF request.getParameter("cancel_booking") <> null THEN booking_id : request.getParameter("booking_id") <> null BookingsDAO.getBookingById(booking_id).getState_name() <> "Annullata" ELSE request.getParameter("book_id") <> null and request.getParameter("start_date") <> null and request.getParameter("end_date") <> null and CardsDAO.getCardByCodice_fiscale(customer.getCodice_fiscale()) .isAssociated() = true
Postcondizioni	**context** BookingServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Post** : IF @pre request.getParameter("cancel_booking") <> null THEN BookingsDAO.updateBooking(booking_id, "Annullata") <> 0 ELSE BookingsDAO.newBooking(email, start_date, end_date, card_id, book_id) <> 0
Invarianti	

Nome classe	BookServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti le ricerche di Libri e che produce output da inviare come risposta.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** BookServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Pre** : IF request.getParameter("search") <> null THEN request.getParameter("q") <> null and request.getParameter("filter") <> null ELSE request.getParameter("random") <> null
Postcondizioni	**context** BookServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Post** : IF @pre request.getParameter("search") <> null THEN BooksDAO.getBooksByFilter(q, filter) <> 0 ELSE BooksDAO.getBookList() <> 0
Invarianti	

Nome classe	CardServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti il secondo passo della Registrazione di un nuovo Cliente. Qui il cliente può scegliere se richiedere una nuova tessera oppure registrarne una inserendone il codice.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** CardServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Pre** : session.getAttribute("customer_incomplete") <> null IF request.getParameter("new_card") <> null THEN CardsDAO.getCardByCodiceFiscale(codice_fiscale) = null ELSE card_id : request.getParameter("card_id") <> null and card_id.match(/^\d{5}$/) CardsDAO.getCardByCodiceFiscale(codice_fiscale) <> null and CardsDAO.getCardByCodiceFiscale(codice_fiscale).getCard_id() = card_id
Postcondizioni	**context** CardServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) **Post** : IF @pre request.getParameter("new_card") <> null THEN CardsDAO.newCard(codice_fiscale) <> 0 ELSE CardsDAO.associateCard(card_id) <> 0 **Post :** CustomersDAO.register(customer) <> 0
Invarianti	

Nome classe	FileServlet.java
Descrizione	Classe che riceve richieste GET riguardanti la localizzazione di file nel File System e risponde dando in output il file richiesto se presente.
Attributi e Metodi	# doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	
Postcondizioni	
Invarianti	

Nome classe	LibraryServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti la Biblioteca e che produce output da inviare come risposta.
Attributi e Metodi	+ init(config : ServletConfig) : void
Precondizioni	
Postcondizioni	**context** LibraryServlet::init(config : ServletConfig) : void **Post :** config.getServletContext().getAttribute("library") <> null
Invarianti	

Nome classe	LoginServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti l'autenticazione degli Clienti e che produce output da inviare come risposta.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** LoginServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Pre :** session.getAttribute("customer") == null **Pre :** email : request.getParameter("email") <> null && password : request.getParameter("password") <> null
Postcondizioni	**context** LoginServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Post :** IF CustomersDAO.login(email, password) <> 0 THEN session.getAttribute("customer") <> null
Invarianti	

Nome classe	ManagerServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti l'autenticazione dei Gestori e le operazioni del pannello di amministrazione.
Attributi e Metodi	# doPost(request : HttpServletRequest, response :HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** ManagerServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Pre :** IF session.getAttribute("admin") = null THEN email : request.getParameter("email") <> null and password : request.getParameter("password") <> null
Postcondizioni	**context** ManagerServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Post :** IF @pre session.getAttribute("admin") = null THEN ManagersDAO.login(email, password) <> 0 and session.getAttribute("admin") <> null
Invarianti	

Nome classe	RegistrationServlet.java
Descrizione	Classe che riceve richieste GET e POST riguardanti il primo passo della registrazione. Al termine reindirizza su CardServlet per il completamento della registrazione.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** RegistrationServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Pre :** session.getAttribute("customer") == null **Pre :** email : request.getParameter("email") <> null and email.match(/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/) and CustomersDAO.checkExistingEmail(email) = false **Pre :** codice_fiscale : request.getParameter("codice_fiscale") <> null and codice_fiscale.match(/[a-zA-Z]{6}\d\d[a-zA-Z]\d\d[a-zA-Z]\d\d\d[a-zA-Z]/) and CustomersDAO.checkExistingCodice_fiscale(codice_fiscale) = false; **Pre :** password : request.getParameter("password") <> null and password.match(/^\w{6,20}$/) and name : request.getParameter("name") <> null and name.match(/^[a-zA-Z]{1,30}$/) and surname : request.getParameter("surname") <> null and surname.match(/^[a-zA-Z]{1,30}$/) and address: request.getParameter("address") <> null and address.match(/^\w{1,100}$/)
Postcondizioni	**context** RegistrationServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Post :** session.getAttribute("customer_incomplete") <> null
Invarianti	

Nome classe	ResetPasswordServlet.java
Descrizione	Classe che riceve richieste GET e POST per reimpostare la password di un Cliente.
Attributi e Metodi	# doPost(request : HttpServletRequest, response : HttpServletResponse) : void # doGet(request : HttpServletRequest, response : HttpServletResponse) : void
Precondizioni	**context** ResetPasswordServlet::doGet(request : HttpServletRequest, response : HttpServletResponse) : void **Pre :** tmp_link : request.getParameter("link") <> null and CustomersDAO.getEmailByTemporaryLink(tmp_link) <> null **context** ResetPasswordServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Pre :** customer_email : session.getAttribute("customer_email") <> null and new_password : request.getParameter("new_password") <> null and new_password.match(/^\w{6,20}$/)
Postcondizioni	**context** ResetPasswordServlet::doGet(request : HttpServletRequest, response : HttpServletResponse) : void **Post :** session.getAttribute("customer_email") <> null **context** ResetPasswordServlet::doPost(request : HttpServletRequest, response : HttpServletResponse) : void **Post :** CustomersDAO.updateCustomerPassword(email, new_password) <> 0
Invarianti	

3.3 DAO

Nome classe	BookingsDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Prenotazione.
Attributi e Metodi	+ newBooking(email : String, start_date : String, end_date : String, state_name : String, card_id : int, book_id : int) : int + getBookingsByFilter(filter : int, keyword : String) : ArrayList <BookingsBean> + getAllBookings() : ArrayList <BookingsBean> + removeBooking(booking_id : int) : int + getCustomerBookings(email : String) : ArrayList <BookingsBean> + getBookingById(booking_id : int) : BookingsBean + updateBooking(booking_id : int, state : String) : int
Precondizioni	**context** BookingsDAO::newBooking(email, start_date, end_date, state_name, card_id, book_id) : int **Pre :** email <> null and start_date <> null and end_date <> null and state_name <> null and card_id <> null and book_id <> null **context** BookingsDAO::getBookingsByFilter(filter, keyword) : ArrayList<BookingsBean> **Pre :** filter <> null and keyword <> null **context** BookingsDAO::removeBooking(booking_id) : int **Pre :** booking_id <> null **context** BookingsDAO::getCustomerBookings(email) : ArrayList<BookingsBean> **Pre :** email <> null **context** BookingsDAO::getBookingById(booking_id) : BookingsBean **Pre :** booking_id <> null **context** BookingsDAO::updateBooking(booking_id, state) : int **Pre :** booking_id <> null and state <> null
Postcondizioni	
Invarianti	

Nome classe	BooksDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Libri.
Attributi e Metodi	+ getAllBooks() : ArrayList<BooksBean> + getBookList() : LinkedHashMap<String, ArrayList<BooksBean>> + getBookById(book_id : int) : BooksBean + getBookGenres(book_id : int) : ArrayList<String> + getBookAuthors(book_id : int) : ArrayList<String> + getJSONAllGenres() : JSONArray + getAllGenres() : ArrayList<String> + getBooksByFilter(filter : int, keyword : String) : ArrayList<BooksBean> + getRandomBookId() : int + removeBook(book_id : int) : int + updateBook(book : BooksBean) : int + newBook(book : BooksBean) : int + getBookCoverById(book_id : int) : String + newGenre(genre_name : String) : int + removeGenre(genre_name : String) : int
Precondizioni	**context** BooksDAO::getBookById(book_id) : BooksBean **Pre :** book_id <> null **context** BooksDAO::getBookGenres(book_id) : ArrayList<String> **Pre :** book_id <> null **context** BooksDAO::getBookById(book_id) : ArrayList<String> **Pre :** book_id <> null **context** BooksDAO::getBooksByFilter(filter, keyword) : ArrayList<BooksBean> **Pre :** filter <> null and keyword <> null **context** BooksDAO::removeBook(book_id) : int **Pre :** book_id <> null **context** BooksDAO::updateBook(book) : int **Pre :** book <> null **context** BooksDAO::newBook(book) : int **Pre :** book <> null **context** BooksDAO::getBookCoverById(book_id) : String **Pre :** book_id <> null

	context BooksDAO::newGenre(genre_name) : int **Pre :** genre_name <> null **context** BooksDAO::removeGenre(genre_name) : int **Pre :** genre_name <> null
Postcondizioni	
Invarianti	

Nome classe	CardsDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Tessere.
Attributi e Metodi	+ getCardsByFilter(card_id : int, keyword : String) : ArrayList<CardsBean> + getAllCards() : ArrayList<CardsBean> + removeCard(card_id : int) : int + getCardByEmail(email : String) : CardsBean + getCardByCodice_fiscale(codice_fiscale : String) : CardsBean + getCardById(card_id : int) : CardsBean + associateCard(card_id : int) : int + newCard(codice_fiscale : String, associated : boolean) : int + newCardAdmin(card : CardsBean) : int
Precondizioni	**context** CardsDAO::getCardsByFilter(card_id, keyword) : ArrayList<CardsBean> **Pre :** card_id <> null and keyword <> null **context** CardsDAO::removeCard(card_id) : int **Pre :** card_id <> null **context** CardsDAO::getCardByEmail(email) : CardsBean **Pre :** email <> null **context** CardsDAO::getCardByCodice_fiscale(codice_fiscale) : CardsBean **Pre :** codice_fiscale <> null **context** CardsDAO::getCardById(card_id) : CardsBean **Pre :** card_id <> null **context** CardsDAO::associateCard(card_id) : int **Pre :** card_id <> null

	context CardsDAO::newCard(codice_fiscale, associated) : int **Pre :** codice_fiscale <> null and associated <> null **context** CardsDAO::newCardAdmin(card) : int **Pre :** card <> null
Postcondizioni	
Invarianti	

Nome classe	LibraryDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Biblioteca.
Attributi e Metodi	+ getLibraryInfo() : LibraryBean + updateLibraryInfo(library : LibraryBean) : int
Precondizioni	**context** LibraryDAO::updateLibraryInfo(library) : int **Pre :** library <> null
Postcondizioni	
Invarianti	

Nome classe	ManagersDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Gestori.
Attributi e Metodi	+ login(email : String, password : String) : ManagersBean + getManagersByFilter(filter : int, keyword : String) : ArrayList<ManagersBean> + getManagerRoles(email : String) : ArrayList<String> + getAllManagers() : ArrayList<ManagersBean> + removeManager(email : String) : int + newManager(manager : ManagersBean) : int + getManagerByEmail(email : String) : ManagersBean + updateManager(manager : ManagersBean, email : String) : int
Precondizioni	**context** ManagersDAO::login(email, password) : ManagersBean **Pre :** email <> null and password <> null **context** ManagersDAO:: getManagersByFilter(filter, keyword) : ArrayList<ManagersBean>

	Pre : filter<> null and keyword <> null **context** ManagersDAO::getManagerRoles(email) : ArrayList<String> **Pre :** email <> null **context** ManagersDAO::removeManager(email) : int **Pre :** email <> null **context** ManagersDAO::newManager(manager) : int **Pre :** manager<> null **context** ManagersDAO::getManagerByEmail(email): ManagersBean **Pre :** email<> null **context** ManagersDAO::updateManager(manager, email) : int **Pre :** manager<> null and email <> null
Postcondizioni	
Invarianti	

Nome classe	CustomersDAO.java
Descrizione	Classe che si occupa dell'interfacciamento con il database per l'esecuzione di query riguardanti oggetti Clienti.
Attributi e Metodi	+ login(email : String, password : String) : CustomersBean + register(customer : CustomersBean) : boolean + checkExistingEmail(email : String) : boolean + checkExistingCodiceFiscale(codice_fiscale : String) : boolean + getCustomerByEmail(customer_email : String) : CustomersBean + getCustomersByFilter(filter : int, keyword : String) : ArrayList<CustomersBean> + getAllCustomers() : ArrayList<CustomersBean> + removeCustomer(email : String) : int + setTemporaryLink(email : String, tmp_link : String) : int + getEmailByTemporaryLink(tmp_link : String) : String + updateCustomerPassword(email : String, new_password : String) : int + deleteTemporaryLink(email : String) : int + updateCustomer(customer : CustomersBean, old_email : String) : int
Precondizioni	**context** CustomersDAO::login(email, password) : CustomersBean **Pre :** email <> null and password<> null **context** CustomersDAO::register(customer) : boolean

	Pre : customer<> null **context** CustomersDAO::checkExistingEmail(email) : boolean **Pre :** email <> null **context** CustomersDAO::checkExistingCodiceFiscale(codice_fiscale) : boolean **Pre :** codice_fiscale <> null **context** CustomersDAO::getCustomerByEmail(customer_email) : CustomersBean **Pre :** customer_email <> null **context** CustomersDAO::getCustomersByFilter(filter, keyword) : ArrayList<CustomersBean> **Pre :** filter <> null and keyword <> null **context** CustomersDAO::removeCustomer(email) : int **Pre :** email <> null **context** CustomersDAO::setTemporaryLink(email, tmp_link) : int **Pre :** email <> null and tmp_link <> null **context** CustomersDAO::getEmailByTemporaryLink(tmp_link) : String **Pre :** tmp_link <> null **context** CustomersDAO::updateCustomerPassword(email, new_password) : int **Pre :** email <> null and new_password <> null **context** CustomersDAO::deleteTemporaryLink(email) : int **Pre :** email <> null **context** CustomersDAO::updateCustomer(customer, old_email) : int **Pre :** customer<> null and old_email <> null
Postcondizioni	
Invarianti	

3.4 View

Nome classe	admin.jsp
Descrizione	Pagina che si occupa dell'autenticazione dei Gestori e della visualizzazione del Pannello di Amministrazione dal quale essi possono svolgere diverse funzionalità.

Nome classe	book_details.jsp
Descrizione	Pagina che mostra nel dettaglio le informazioni contenute in un oggetto Libro. Da qui è inoltre possibile effettuare la prenotazione del libro che si sta consultando.

Nome classe	card.jsp
Descrizione	Pagina visibile solo al secondo step della registrazione di un nuovo Cliente. Si occupa di far scegliere se richiedere una nuova tessera oppure registrarne una già esistente.

Nome classe	contacts.jsp
Descrizione	Pagina che visualizza i contatti della biblioteca, prelevati appunto dall'oggetto Biblioteca.

Nome classe	footer.jsp
Descrizione	Parte inferiore del sito. Viene incluso in ciascuna pagina JSP.

Nome classe	genres.jsp
Descrizione	Pagina che si occupa della visualizzazione del catalogo libri (UC:7 del RAD).

Nome classe	header.jsp
Descrizione	Contiene tutti i link ai vari script e fogli di stile utilizzati per la corretta visualizzazione di ciascuna pagina JSP. È pertanto incluso in ognuna di essa.

Nome classe	index.jsp
Descrizione	Pagina che si occupa della visualizzazione dell'homepage.

Nome classe	login.jsp
Descrizione	Pagina che si occupa della visualizzazione del form per l'autenticazione di un Cliente.

Nome classe	logout.jsp
Descrizione	Pagina che si occupa della disconnessione di un Utente dal sistema.

Nome classe	navbar.jsp
Descrizione	Parte superiore del sito. Si occupa di fornire un menu orizzontale per facilitare la navigazione all'interno del sito. Viene incluso in ciascuna pagina JSP.

Nome classe	profile.jsp
Descrizione	Pagina che si occupa della visualizzazione del profilo prelevando le informazioni da un oggetto Cliente.

Nome classe	registration.jsp
Descrizione	Pagina che si occupa di visualizzare il form per la registrazione di un nuovo Cliente.

Nome classe	reset_password.jsp
Descrizione	Pagina che si occupa dell'invio del link di reset password e del cambio della stessa.

Nome classe	search.jsp
Descrizione	Pagina che si occupa di visualizzare i risultati di ricerca in base al filtro selezionato.

3.5 Util

Nome classe	DBConnection.java
Descrizione	Classe che si occupa di specificare i parametri di configurazione per la connessione con il DBMS utilizzando il driver JDBC e della gestione delle connessioni.
Attributi e metodi	+ freeDbConnections : List<Connection> + createDBConnection() : Connection + getConnection() : Connection + releaseConnection(conn : Connection) : void
Precondizioni	**context** DBConnection::releaseConnection(conn) : void **Pre :** conn <> null
Postcondizioni	
Invarianti	

Nome classe	EmailUtility.java
Descrizione	Classe che si occupa dell'invio di email in base ai parametri passati. (es. host, porta, username, password, destinatario, oggetto, messaggio).
Attributi e metodi	+ sendEmail(host : String, port : String, username : String, password : String, toAddress : String, subject: String, message : String) : void
Precondizioni	**context** EmailUtility::sendEmail(host, port, username, password, toAddress, subject, message) : void

	Pre : host <> null and port <> null and username <> null and password <> null and toAddress <> null and subject <> null and message <> null
Postcondizioni	
Invarianti	

Nome classe	InputChecker.java
Descrizione	Classe contenente metodi per il controllo del formato degli input utilizzando espressioni regolari.
Attributi e metodi	+ checkEmail(email : String) : boolean + checkPassword(password : String) : boolean + checkName(name : String) : boolean + checkSurname(surname : String) : boolean + checkAddress(address : String) : boolean + checkPhone(phone : String) : boolean + checkCodice_fiscale(codice_fiscale : String) : boolean + checkCard_id(card_id : String) : boolean + checkBooking_id(booking_id : String) : boolean + checkBook_id(book_id : String) : boolean + checkTitle(title : String) : boolean + checkAuthor(author : String) : boolean + checkPublisher(publisher : String) : boolean + checkGenre(genre : String) : boolean + checkDescription(description : String) : boolean + checkQuantity(quantity : String) : boolean
Precondizioni	**context** InputChecker::checkEmail(email) : boolean **Pre :** email <> null **context** InputChecker::checkPassword(password) : boolean **Pre :** password <> null **context** InputChecker::checkName(name) : boolean **Pre :** name <> null **context** InputChecker::checkSurname(surname) : boolean **Pre :** surname <> null **context** InputChecker::checkAddress(address) : boolean **Pre :** address <> null **context** InputChecker::checkPhone(phone) : boolean **Pre :** phone <> null

	context InputChecker::checkCodice_fiscale(codice_fiscale) : boolean **Pre :** codice_fiscale <> null **context** InputChecker::checkCard_id(card_id) : boolean **Pre :** card_id <> null **context** InputChecker::checkBooking_id(booking_id) : boolean **Pre :** booking_id <> null **context** InputChecker::checkBook_id(book_id) : boolean **Pre :** book_id <> null **context** InputChecker::checkTitle(title) : boolean **Pre :** title <> null **context** InputChecker::checkAuthor(author) : boolean **Pre :** author <> null **context** InputChecker::checkPublisher(publisher) : boolean **Pre :** publisher <> null **context** InputChecker::checkGenre(genre) : boolean **Pre :** genre <> null **context** InputChecker::checkDescription(description) : boolean **Pre :** description <> null **context** InputChecker::checkQuantity(quantity) : boolean **Pre :** quantity <> null
Postcondizioni	
Invarianti	

Glossario

Framework: piattaforma che funge da strato intermedio tra un sistema operativo e il software che lo utilizza.

Hashing: con hashing si intende il processo che genera un output di dimensione fissa partendo da un input di dimensioni variabili.

Off-the-shelf: componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.

Singleton: è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Sliding: effetto visivo che simula lo scorrimento in maniera crescente o decrescente di un oggetto. L’oggetto è inizialmente nascosto, mostrandosi del tutto a fine animazione o viceversa.

Javascript: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client.