ODD Object Design Document

Bookster

Versione	0.6
Data	17/02/2023
Destinatario	Esame di Ingegneria del Software 2022/23
Presentato da	Francesco Alfonso Barlotti (0512110169) Teodoro Grauso (0512111084) Giovanni Manfredi (0512112926)

Revision History

Data	Versione	Descrizione	Autori
09/01/2023	0.1	Prima stesura	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso
10/1/2023	0.2	Definizione dei package	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso
24/1/2023	0.3	Definizione e stesura dei design pattern	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso
8/2/2023	0.4	Implementazione delle interfacce delle classi e del Class Diagram Ristrutturato	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso
9/2/2023	0.5	Revisione finale	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso
17/02/2033	0.6	Revisione	Francesco Alfonso Barlotti Giovanni Manfredi Teodoro Grauso

Team members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Giovanni Manfredi	Team Member	GM	g.manfredi5@studenti.unisa.it
Teodoro Grauso	Team Member	TG	t.grauso@studenti.unisa.it
Francesco Alfonso Barlotti	Team Member	FAB	f.barlotti1@studenti.unisa.it

Sommario

1. Introduzione 5
1.1 Linee guida per la scrittura del codice 5
1.2 Definizione, acronimi e abbreviazioni 5
1.3 Riferimenti e link utili 5
2. Packages 6
3. Class Interfaces 8
4. Class Diagram Ristrutturato 17
5. Elementi di Riuso 17
5.1 Design Pattern Usati 17
6. Glossario 19

1. Introduzione

L'obiettivo prefissato da Bookster è di sensibilizzare e di invogliare quanto più possibile alla lettura la popolazione, considerando come la percentuale dei lettori in Italia risulta essere pari solamente al 56%, come riportato nel progetto di ricerca nel 2021 sulla lettura frutto della collaborazione tra il Centro per il libro e la lettura (CEPELL) e l'Associazione Italiana Editori (AIE) e presentato al Salone internazionale del Libro di Torino nel convegno ***Leggere in pandemia #1 – Nuovi percorsi di lettura degli italiani.***

Mediante un percorso ed un processo innovativo Bookster intende rivoluzionare il modo tradizionale della lettura, creando delle basi per uno strumento di comunicazione e di sana competitività, affinché la lettura possa diventare una sana abitudine quotidiana per gli utenti.

1.1 Linee guida per la scrittura del codice

Le linee guida prevedono una serie convenzioni e di canoni che gli sviluppatori dovrebbero procedere a rispettare nella progettazione delle interfacce del sistema. Per la definizione e costituzione delle linee guida si è provveduto a fare riferimento alla convenzione Java nota come **Sun Java Coding Conventions.**

Link a documentazione ufficiale sulle convenzioni

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **Java Sun:** https://checkstyle.sourceforge.io/sun_style.html
- **HTML:** https://www.w3schools.com/html/html5_syntax.asp

1.2 Definizione, acronimi e abbreviazioni

Di seguito, sono riportate alcune definizioni presenti nel documento:

- **Package:** raggruppamento di classi, interfacce o file correlati;
- **Design pattern:** template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità;
- **Interfaccia:** insieme di signature delle operazioni offerte dalla classe;
- **Javadoc:** sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile;

1.3 Riferimenti e link utili

Di seguito una lista di riferimenti e link ad altri documenti utili durante la lettura:

- Requirements Analysis Document
- System Design Document
- Test Plan
- Javadoc di Bookster

2. Packages

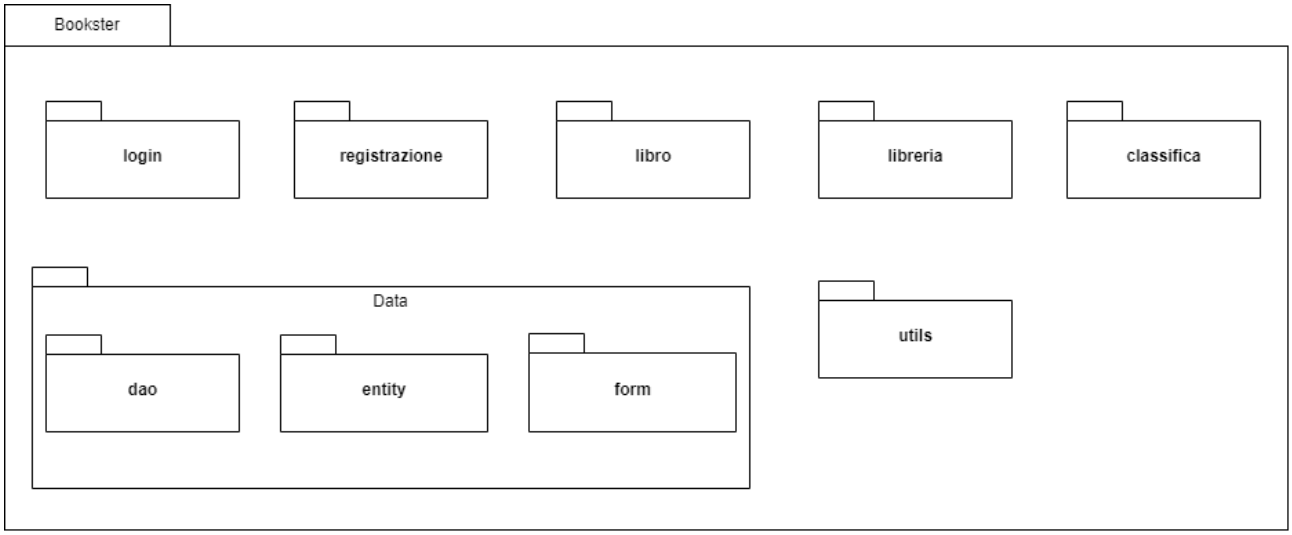
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

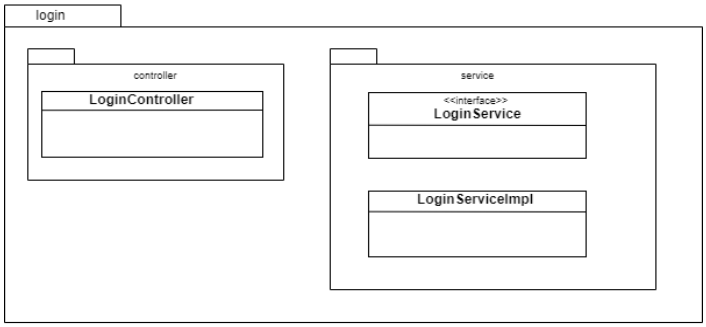
- **.idea**
- **.mvn,** contiene tutti i file di configurazione per Maven
- **src,** contiene i file sorgenti
    - **main**
        - **java,** contiene le classi Java relativamente alle componenti Logic e Data
        - **webapp,** contiene i fogli di stile CSS, pagine HTML, script JS e le immagini necessarie
        - **WEB-INF,** contiene i file JSP relativi alle componenti del Presentation
    - **test,** contiene tutto il necessario per il testing
        - **java,** contiene le classi Java necessarie per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven

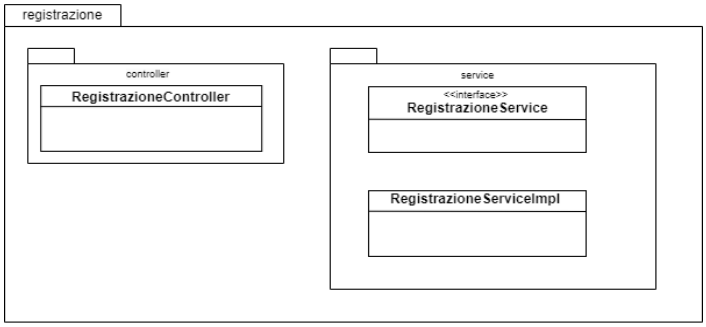
Package di Bookster

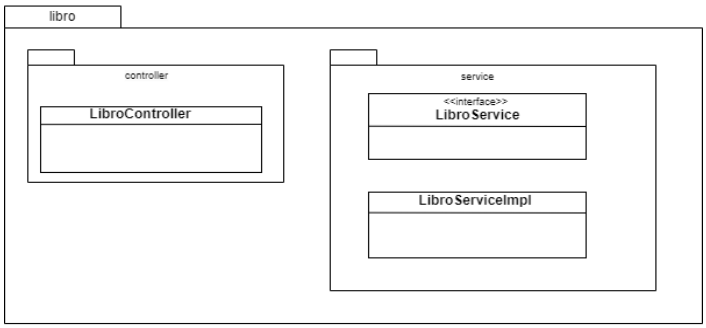
Nella seguente sezione si procede a mostrare e ad evidenziare la struttura del package principale di Bookster. La struttura prevede tre package ottenuta da tre principali scelte:

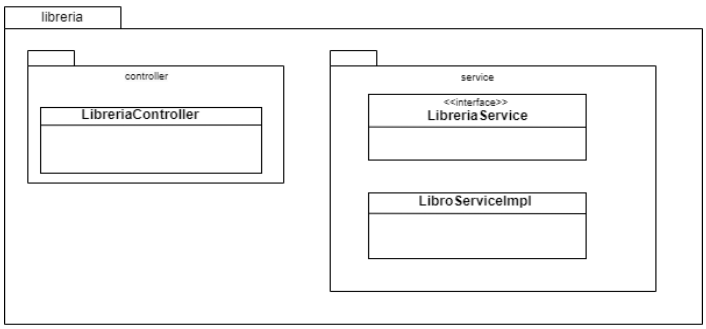
1. **package Logic:** contiene i sottosistemi per gestire l'accesso, la registrazione, la classifica, la gestione della libreria personale del lettore e le funzionalità di ricerca di un contenuto;
2. **package Data:** contiene le classi Entity ed i DAO per l'accesso al Database
3. **package utils:** il seguente pacchetto provvederà a contenere eventuali classi software, che possono essere utilizzate dagli altri componenti del sistema;

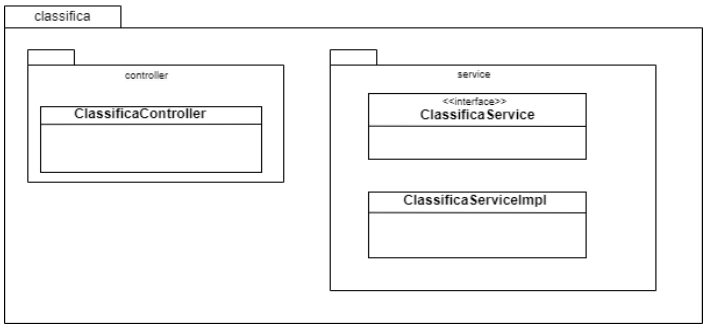



login
controller
LoginController
service
<<interface>>
LoginService
LoginServiceImpl


registrazione
controller
RegistrazioneController
service
<<interface>>
RegistrazioneService
RegistrazioneServiceImpl


libro
controller
LibroController
service
<<interface>>
LibroService
LibroServiceImpl


libreria
controller
LibreriaController
service
<<interface>>
LibreriaService
LibroServiceImpl


classifica
controller
ClassificaController
service
<<interface>>
ClassificaService
ClassificaServiceImpl

3. Class Interfaces

Di seguito si presentano le interfacce di ogni classe.

Javadoc di Bookster

Per motivi di leggibilità si è scelto di creare un sito, hostato tramite GitHub pages, contenente la JavaDoc di Bookster. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero sistema.

Di seguito, il link al sito in questione: https://grauso-t.github.io/Bookster_Classe03/

1. **Package Login**
   it.unisa.Bookster.login.Login

Nome classe	**LoginServlet**
Descrizione	La seguente classe consente di gestire le operazioni relativamente l'accesso dell'utente alla piattaforma.
Metodi	+doPost(HttpServletRequest request, HttpServletResponse response): void +doLogin(String email, String password): Lettore
Invariante di classe	/

Nome Metodo	**+doPost(HttpServletRequest request, HttpServletResponse response)**
Descrizione	Il seguente metodo gestisce la richiesta del lettore di accesso alla piattaforma
Pre-condizione	**context**: LoginServlet:: +doPost(HttpServletRequest request, HttpServletResponse response) **pre:** request.getParameter("email") !=null and request.getParameter("password") !=null
Post-condizione	**context**: LoginServlet:: +doPost(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+doLogin(String email, String password)
Descrizione	Il seguente metodo provvede a verificare l'esistenza dell'account del lettore all'interno del database.
Pre-condizione	**context:** LoginServlet:: doLogin(String email, String password) **pre**: email!=null && password!=null
Post-condizione	**context:** LoginServlet:: doLogin(email,password) **post:** LettoreDAO.doLogin(email, password)!=null

2. Package Registrazione

it.unisa.Bookster.registrazione.Registrazione

Nome classe	RegistrazioneServlet
Descrizione	La seguente classe consente di gestire le operazioni relativamente la registrazione del lettore alla piattaforma.
Metodi	+ doPost(HttpServletRequest request, HttpServletResponse response): void +doRegistrazione(Lettore lettore): void +controlloEmail(String email): boolean +doRetriveUtente(): List<Lettore>
Invarianti di classe	/

Nome Metodo	doPost(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo gestisce la richiesta di registrazione dell'account del lettore alla piattaforma.
Pre-condizione	**context:** RegistrazioneServlet:: doPost(HttpServletRequest request, HttpServletResponse response) **pre**: request.getParameter("username") !=null and request.getParameter("firstName")!=null and request.getParameter("lastName")!=null and Date.valueOf(request.getParameter("birthday"))!=null and request.getParameter("email")!=null and request.getParameter("password")!=null and request.getParameter("confirmPassword")!=null and request.getParameter("phone")!=null
Post-condizione	**context:** RegistrazioneServlet:: doPost(HttpServletRequest request, HttpServletResponse response) **post**: Lettore l !=null
Nome Metodo	+doRegistrazione(Lettore lettore)
Descrizione	Questo metodo consente la registrazione di un nuovo lettore
Pre-condizione	**context:** RegistrazioneService: controlloEmail(email) **post:** controlloEmail(email)==false
Post-condizione	**context:** RegistrazioneService:: doRegistrazione(Lettore lettore) **post:** /

Nome Metodo	+controlloEmail(String email): boolean
Descrizione	Questo metodo consente di verificare che l'indirizzo email inserito al momento della registrazione non sia già presente.
Pre-condizione	**context:** RegistrazioneService:: controlloEmail(String email) **pre:** email!=null
Post-condizione	**context:** RegistrazioneService:: controlloEmail(String email) **post:** controlloEmail(email)==false
Nome Metodo	+doRetriveUtente()
Descrizione	Questo metodo consente di ottenere tutti gli utenti registrati.
Pre-condizione	**context:** RegistrazioneServlet:: doRetriveUtente() **pre:** /
Post-condizione	**context:** RegistrazioneServlet:: doRetriveUtente() **post:** /

3. Package Libro

it.unisa.Bookster.libro.Libro

Nome classe	**SearchServlet**
Descrizione	La seguente servlet consente di gestire le operazioni di ricerca dei libri mediante titolo dell'opera, autore ed ISBN
Metodi	+ doGet(HttpServletRequest request, HttpServletResponse response): void
Invarianti di classe	/

Nome Metodo	**+doGet(HttpServletRequest request, HttpServletResponse response)**
Descrizione	Il seguente metodo procede ad effettuare l'opportuna ricerca immessa sulla base dei valori sottomessi dal lettore, procedendo ad interfacciarsi con le API di Google Books.
Pre-condizione	**context:** SearchServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getParameter("selectionInput")!=null && request.getParameter("searchBar")!=null
Post-condizione	**context:** SearchServlet:: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /

Nome classe	InsertBookServlet
Descrizione	La seguente servlet consente di inserire manualmente un libro, qualora non fosse presente tra i risultati proposti mediante la ricerca effettuata precedentemente.
Metodi	+ doGet(HttpServletRequest request, HttpServletResponse response) :void + doSave(Book book): void
Invarianti di classe	/

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo procede a recuperare i parametri immessi dal lettore per la memorizzazione del libro all'interno del database.
Pre-condizione	**context:** InsertBookServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getParameter("isbn")!=null or request.getParameter("title")!=null or request.getParameter("author")!=null or request.getParameter("numberPage")!=null
Post-condizione	**context:** InsertBookServlet:: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+ doSave(Book book)
Descrizione	Il seguente metodo provvede al salvataggio del libro inserito manualmente dal lettore all'interno del database.
Pre-condizione	**context:** InsertBookServlet:: doGet(HttpServletRequest request, HttpServletResponse response), **pre:** book! =null
Post-condizione	**post:** /

Nome classe	BookInformationServlet
Descrizione	La seguente servlet consente di visualizzare la scheda informativa dei libri selezionati dal lettore
Metodi	BookInformationServlet: +doGet(HttpServletRequest request, HttpServletResponse response): void
Invarianti di classe	/

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo procede a mostrare la scheda informativa del libro selezionato dal lettore
Pre-condizione	**context:** BookInformationServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("bookList")!=null and int index!=null
Post-condizione	**context:** BookInformationServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** Book book!=null

4. Package Libreria

it.unisa.Bookster.libreria.Libreria

Nome classe	LibraryServlet
Descrizione	La seguente servlet consente di aggiungere i libri alla libreria del lettore sulla base della/e ricerca/ricerche effettuata/e.
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +doGetAddToLib(HttpServletRequest request, HttpServletResponse response): void + doGetAddFavourite(HttpServletRequest request, HttpServletResponse response): void + isExisting(String isbn,List<Book> libraryList): int
Invarianti di classe	/

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo procede a selezionare il metodo appropriato per l'aggiunta dei libri alla sezione "*Libreria*" o "*Preferiti*", a seconda dell'azione immessa dal lettore.
Pre-condizione	**context:** LibraryServlet:: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("lettore")!=null and action !=null

Nome Metodo	+doGetAddToLib(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo procede ad aggiungere alla *"libreria"* del lettore i libri selezionati.
Pre-condizione	**context:** LibraryServlet:: doGetAddToLib(HttpServletRequest request, HttpServletResponse response) **pre:** session.getAttribute("bookList")!= null
Post-condizione	**context:** LibraryServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+doGetAddFavourite (HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo provvede ad aggiungere i libri preferiti selezionati alla sezione dei *"preferiti"*.
Pre-condizione	**context:** LibraryServlet: +doGetAddToFavourite(HttpServletRequest request, HttpServletResponse response) **pre:** session.getAttribute("bookList")!= null
Post-condizione	**context:** LibraryServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+ isExisting(String isbn,List<Book> libraryList): int
Descrizione	Il seguente metodo provvede a verificare sulla base dell'ISBN se un libro risulti essere già presente nella libreria del lettore.
Pre-condizione	**context:** LibraryServlet: +isExisting(String isbn,List<Book> libraryList): int **pre:** librayList!=null and isbn!=null
Post-condizione	**context:** LibraryServlet: +isExisting(String isbn,List<Book> libraryList): int **post:** int index==i or index==-1

Nome classe	LibraryBookInfoServlet
Descrizione	La seguente servlet consente di aggiungere i libri alla libreria/preferiti del lettore a partire dalla scheda informativa del libro visualizzato
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +doGetAddToLib(HttpServletRequest request, HttpServletResponse response): void + doGetAddFavourite(HttpServletRequest request, HttpServletResponse response): void + isExisting(String isbn,List<Book> libraryList): int
Invarianti di classe	/

Nome Metodo	**+doGet(HttpServletRequest request, HttpServletResponse response)**
Descrizione	Il seguente metodo procede a selezionare il metodo appropriato per l'aggiunta dei libri alla sezione "*Libreria*" o "*Preferiti*", a seconda dell'azione immessa dal lettore.
Pre-condizione	**context:** LibraryBookInfoServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("lettore")!=null and action !=null

Nome Metodo	**+doGetAddToLib(HttpServletRequest request, HttpServletResponse response)**
Descrizione	Il seguente metodo procede ad aggiungere alla *"libreria"* del lettore i libri selezionati.
Pre-condizione	**context:** LibraryBookInfoServlet: +doGetAddToLib(HttpServletRequest request, HttpServletResponse response) **pre:** session.getAttribute("bookList")!= null
Post-condizione	**context:** LibraryBookInfoServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+doGetAddFavourite (HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo provvede ad aggiungere i libri preferiti selezionati alla sezione dei *"preferiti"*.
Pre-condizione	**context:** LibraryBookInfoServlet: +doGetAddToFavourite(HttpServletRequest request, HttpServletResponse response) **pre:** session.getAttribute("bookList")!= null
Post-condizione	**context:** LibraryServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** /
Nome Metodo	+ isExisting(String isbn,List<Book> libraryList): int
Descrizione	Il seguente metodo provvede a verificare sulla base dell'ISBN se un libro risulti essere già presente nella libreria del lettore.
Pre-condizione	**context:** LibraryBookInfoServlet: +isExisting(String isbn,List<Book> libraryList): int **pre:** librayList!=null and isbn!=null
Post-condizione	**context:** LibraryBookInfoServlet: +isExisting(String isbn,List<Book> libraryList): int **post:** index==i or index==-1

5. Package Classifica

it.unisa.Bookster.classifica.Classifica

Nome classe	**ScoreServlet**
Descrizione	La seguente servlet provvede a calcolare il punteggio dei lettori per procedere ad elaborare la relativa classifica.
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void +upgradePunteggio(String username, int punteggio): void
Invarianti di classe	/

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo procede a calcolare il punteggio del lettore.
Pre-condizione	**context:** ScoreServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** int pages = Integer.parseInt(request.getParameter("pages"))!=null and int time = Integer.parseInt(request.getParameter("seconds"))!=null
Post-condizione	**Context:** ScoreServlet: +doGet(HttpServletRequest request, HttpServletResponse response): void **post:** /
Nome Metodo	+upgradePunteggio(String username, int punteggio): void
Descrizione	Il seguente metodo provvede a memorizzare nel database il punteggio associato al lettore calcolato precedentemente.
Pre-condizione	**context:** ScoreServlet: upgradePunteggio(String username, int punteggio) **pre:** request.getSession().getAttribute("lettore")!=null and punteggio>=0
Post-condizione	**Context:** ScoreServlet: +doGet(HttpServletRequest request, HttpServletResponse response): void **post:** /

Nome classe	**RankingServlet**
Descrizione	La seguente servlet provvede ad elaborare la classifica sulla base dei punteggi calcolati.
Metodi	+doGet(HttpServletRequest request, HttpServletResponse response): void
Invarianti di classe	/

Nome Metodo	+doGet(HttpServletRequest request, HttpServletResponse response)
Descrizione	Il seguente metodo provvede a stilare la classifica sul relativo punteggio precedentemente calcolato
Pre-condizione	**context:** RankingServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** LettoreDAO. doRetriveUtente() !=null
Post- condizione	**context:** RankingServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** List<Lettore> rakingList!= null

4. Class Diagram Ristrutturato

Per questione di visibilità si consiglia la visione del Class Diagram Ristrutturato al seguente link: https://ibb.co/8YtCKFp

5. Elementi di Riuso

5.1 Design Pattern Usati

Nella seguente sezione si procede a descrivere e a dettagliare i design patterns utilizzati per procedere allo sviluppo e alla realizzazione di Bookster. Per i design pattern utilizzati si procede a definire:

- Breve descrizione del design pattern utilizzato;
- Problematica da risolvere per il contesto applicativo di Bookster;
- Breve spiegazione della risoluzione della problematica presentata nello sviluppo applicativo di Bookster;
- Un grafico della struttura delle classi, che le implementano il pattern;

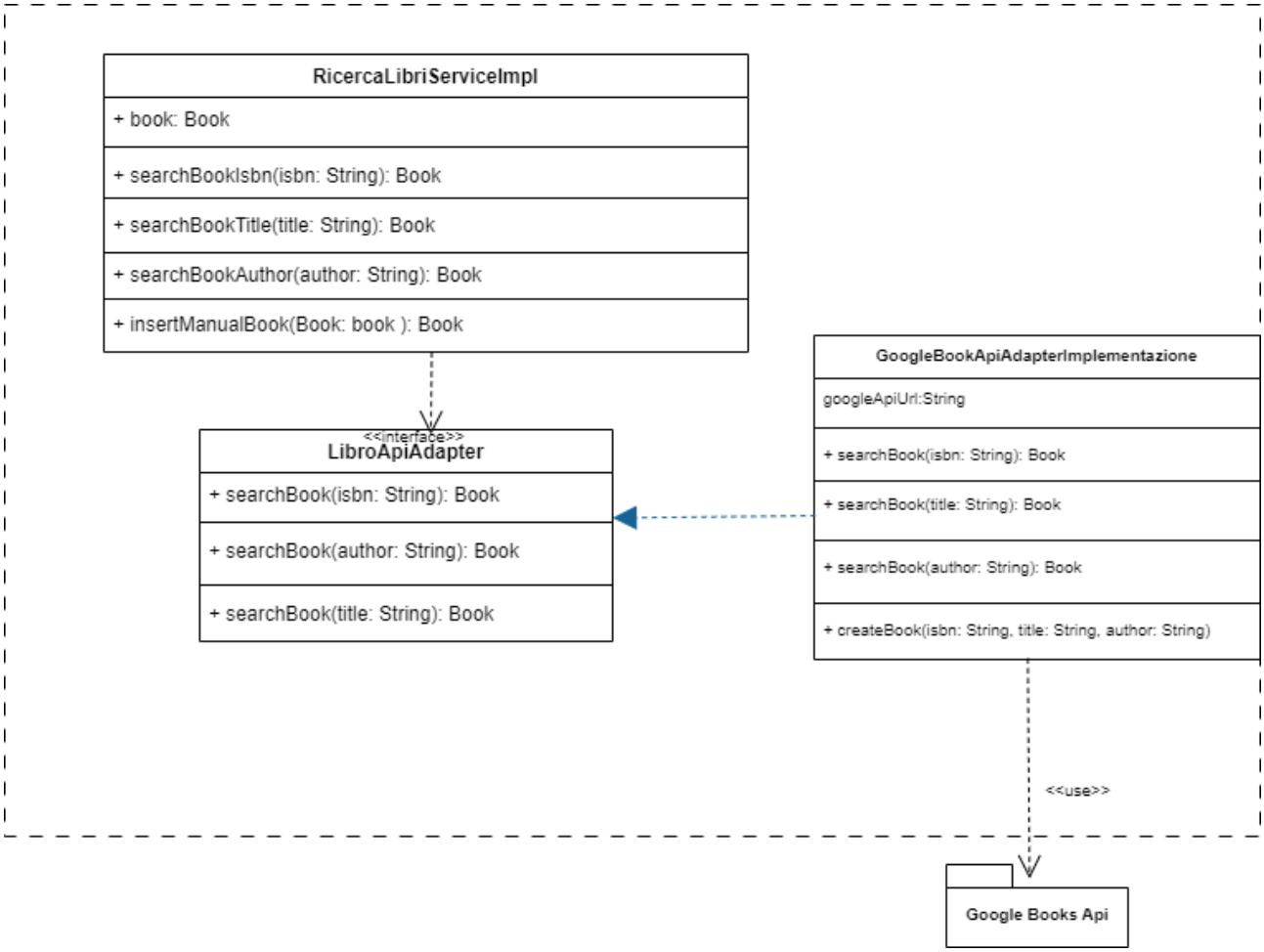
Adapter

Il pattern Adapter è un pattern di progettazione software che consente di adattare un'interfaccia di una classe per renderla compatibile con un'altra interfaccia utilizzata dalle classi client. Il pattern Adapter converte l'interfaccia di una classe in un'altra interfaccia che le classi client aspettano di trovare. In questo modo le classi esistenti possono lavorare insieme, anche se le loro interfacce sono incompatibili**.**

Bookster si pone l'obiettivo principale di sensibilizzare la popolazione alla lettura, cercando di renderla come principale attività quotidiana. Ai lettori Bookster offre la possibilità di generare una libreria personale, potendo così tenere traccia dell'insieme delle letture in esecuzione, in programma e da leggere. L'aggiunta alla libreria del lettore risulta essere possibile grazie alla ricerca del contenuto mediante l'apposito ISBN, titolo oppure mediante l'autore del libro. La possibilità di ricerca dei seguenti componenti risulta essere possibile mediante le API di Google Books, dove risulta essere possibile procedere accedere ai contenuti offerti dai database di Google Books. I risultati della ricerca restituiti dalle API di Google Books risultano essere nel formato JSON.

Il design pattern Adapter in questa fase si occupa di effettuare la conversione dei dati JSON nel corrispettivo oggetto della classe Libro, affinché si possa provvedere a garantire la conversione dei dati in

maniera accessibile al sistema. Il funzionamento del design pattern Adapter prevede il trasferimento della query alle API mediante il contenuto inserito dal lettore e la relativa conversione del risultato mediante il design pattern Adapter.



Strategy

Il pattern Strategy è un pattern di progettazione software che consente di selezionare e utilizzare le diverse azioni tra diverse alternative in modo dinamico. Il pattern Strategy definisce una serie di azioni che possono essere utilizzati per risolvere un problema specifico e consente di scegliere quale utilizzare in un determinato contesto. Ciò permette di separare la logica di alto livello dalla logica di basso livello, rendendo il codice più facile da modificare e mantenere.

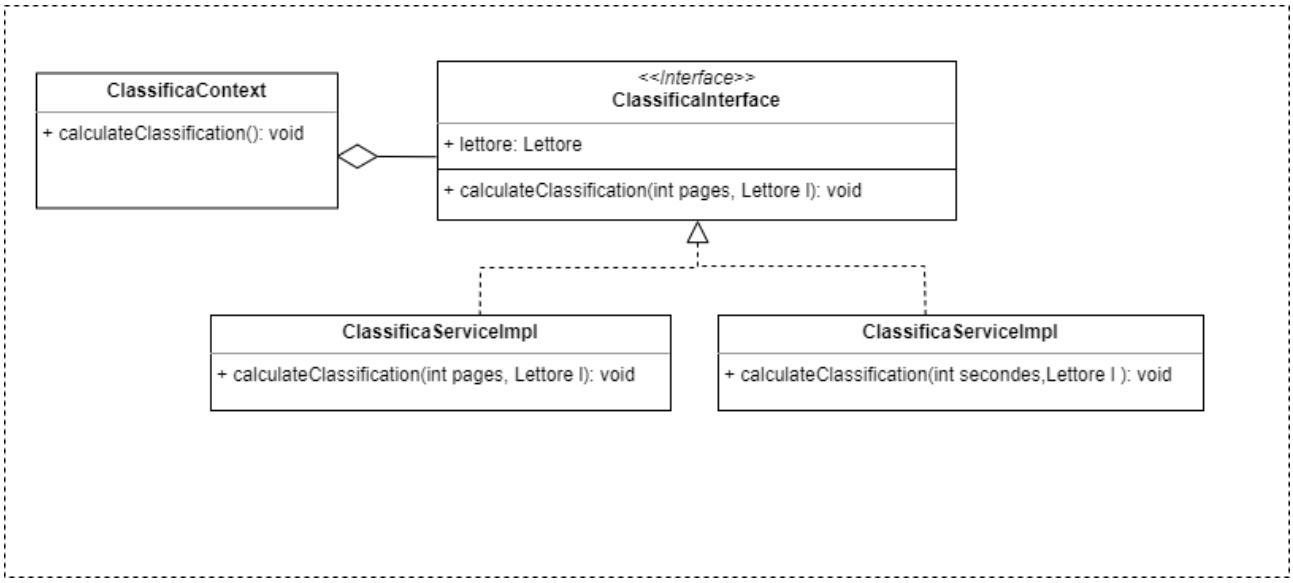
Bookster siccome vuol rendere la lettura un momento di piacevolezza e allo stesso tempo un momento gratificante mediante dei meccanismi competitivi. Bookster prevede la definizione di una graduatoria per i lettori più attivi, andando ad associare il relativo punteggio e premiando i lettori più assidui in un determinato arco temporale. La generazione della classifica dovrebbe invogliare i lettori a svolgere le sessioni di lettura col giusto brio di **competitività e di motivazione.** La generazione della classifica è possibile definirla secondo diverse possibilità.

La classifica può essere stilata secondo **diverse possibilità ed opportunità** al fine di perseguire l'obiettivo di premiare i lettori più produttivi ed efficienti. Per il seguente pattern di Strategy si procederà ad effettuare

una suddivisione dei modi possibili della determinazione della classifica, una a seconda della strategia da adottare:

1. la classifica può essere definita sulla base sessione di lettura impostata dai lettori, andando a calcolare il relativo punteggio sulla base di tempo dedicato alla lettura;
2. la classifica può essere definita sull'insieme delle pagine lette dai lettori, andando ad associare per ogni pagina letta dal lettore il relativo punteggio;

Successivamente all'attività di calcolo del punteggio a seconda delle modalità evidenziate, si provvederà ad effettuare la generazione della classifica secondo un ordinamento di tipo crescente per i lettori partecipanti alla competizione, salvando successivamente il relativo punteggio associato.



6. Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di classi ed interfacce.
Applicazione Web	Applicazione accessibile attraverso web per mezzo di una rete come ad esempio Internet.
Adapter pattern	L'Adapter Design Pattern è un modello di progettazione che consente di adattare un'interfaccia di una classe a un'altra interfaccia, attesa dal client. L'Adapter Design Pattern fornisce un modo per convertire le interfacce di classi diverse in un'interfaccia comune, che possa essere utilizzata dal client. Questo è utile quando si vuole utilizzare una classe esistente in un nuovo contesto dove l'interfaccia non è compatibile. Questo pattern crea un'interfaccia di adattamento tra due interfacce, che può essere utilizzato per garantire la compatibilità tra le due interfacce.
Strategy pattern	Il Strategy Design Pattern è un modello di progettazione che consente di selezionare un algoritmo da eseguire a runtime, tra un insieme di algoritmi disponibili, in base alle esigenze del contesto. In altre parole, questo pattern separa l'algoritmo dall'oggetto che lo utilizza e lo rende intercambiabile. Il client può scegliere quale algoritmo utilizzare e cambiarlo a runtime, senza dover modificare il codice sorgente dell'oggetto che utilizza l'algoritmo. Questo rende

	il codice più flessibile e facilmente adattabile a nuove esigenze, poiché l'algoritmo può essere modificato senza influire sulla struttura dell'oggetto che lo utilizza.