Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software*- Prof. C. Gravino

Object Design Document

Riferimento	
Versione	1.2
Data	17/01/2022
Destinatario	Prof. C. Gravino
Presentato da	Sabato Celentano Daniele Donia Gennaro Rascato Pasquale Somma
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
16/12/2021	1.0	Prima stesura	Sabato Celentano Daniele Donia Gennaro Rascato Pasquale Somma
13/01/2022	1.1	Modifica package	Daniele Donia Gennaro Rascato Pasquale Somma
17/01/2022	1.2	Modifica design pattern	Sabato Celentano Pasquale Somma

Sommario

1. Introduzione

1.1 Object Design Trade-off

1.1.1 Comprensibilità vs Costo di sviluppo

L'obiettivo di avere un codice comprensibile e con una struttura ben definita è in contrasto con quello di rispettare i costi di sviluppo, ovvero le 50 ore di lavoro per team member. Si intende favorire la comprensibilità, anche se questo comporta un aumento delle ore di lavoro per team member.

1.1.2 Usabilità vs Funzionalità

Nella progettazione, l'obiettivo di avere un'usabilità dei servizi maggiore è in contrasto con quello di fornire molteplici funzionalità, in quanto un numero elevato di servizi può rendere difficoltoso l'utilizzo delle interfacce da parte dell'utente. Si intende quindi favorire l'usabilità, anche se questo comporta un numero di servizi minore.

1.1.3 Efficienza vs Portabilità

L'obiettivo di avere un sistema efficiente e quello di avere un sistema portabile sono contrastanti, in quanto un sistema portabile adattandosi a più le piattaforme perde di efficienza. Si intende favorire l'efficienza, anche se questo comporta una portabilità minore. Limiteremo quindi la portabilità all'ambiente Android, dalla versione 5.0 in poi.

1.2 Components Off The Shelf (COTS)

Il Sistema utilizzerà i seguenti componenti off the shelf:

- **LightFM,** libreria che consiste nell'implementazione in Python di una serie di algoritmi di raccomandazione popolari per feedback sia implicito che esplicito;
- **Apache Tomcat**, un server web (nella forma di contenitore servlet) open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java.
- **Gradle**, uno strumento di automazione della compilazione (build automation tool) che fornisce un linguaggio specifico di dominio (DSL) basato su Groovy. Viene eseguito su JVM, oltre a consentire il supporto per la creazione di progetti nativi. Gradle consente di utilizzare repository Maven o altri definiti dall'utente per la gestione delle dipendenze, supportando build poliglotte;
- **MySQL**, un RDBMS, ovvero un sistema open source di gestione di database relazionali SQL. SQL è la lingua più popolare per aggiungere, accedere e gestire del contenuto in un database.
- **Gson**, libreria Java che consente di convertire in maniera semplice e veloce oggetti Java nella loro rappresentazione JSON, e viceversa, nello sviluppo di un'applicazione Android.
- **AsyncHttpClient**, libreria che permette alle applicazioni Java di inviare facilmente richieste Http e ricevere asincronamente risposte Http.
- **Android,** libreria che si occupa di contenere tutti i file supportati in un progetto android, compreso il codice sorgente, risorse e file di manifesto comprese le configurazioni. La build risultante è un archivio android chiamato AAR, un file che puoi aggiungere come dipendenza ai moduli dell'applicazione android.

1.3 Interfaces, Documentation, Guidelines

Di seguito sono riportate tutte le linee guida che i programmatori dovranno seguire nella fase di implementazione:

1.3.1 Classi e Interfacce Java

Nella scrittura del codice Java facciamo riferimento alla guida fornita da Google, consultabile al sito https://google.github.io/styleguide/javaguide.html.
In particolare, nel nostro codice facciamo riferimento alle seguenti regole:

- Le parentesi graffe per blocchi non-vuoti o maggiori di una riga;
- nessuna interruzione di riga (line break) prima di aprire la parentesi;
- un line break dopo l'apertura della parentesi;
- un line break prima di chiudere la parentesi;
- un line break dopo la chiusura della parentesi solo se la parentesi chiude uno statement o termina il corpo di un metodo, costruttore o classe;
- I package sono tutti in lowercase, con parole consecutive semplicemente concatenate senza underscore;
- I nomi delle classi sono scritti in UpperCamelCase. Essi sono tipicamente sostantivi che fanno riferimento al dominio applicativo del sistema. I metodi sono scritti in lowerCamelCase, i cui nomi sono tipicamente verbi, come anche i campi (che non sono final) sono scritti in lowerCamelCase con nomi che sono tipicamente sostantivi;
- I campi final, che rappresentano delle costanti, vengono scritti in maiuscolo;
- Le classi di test hanno nomi che iniziano con il nome della classe testata e terminano con "Test".

Tuttavia, nel nostro codice applichiamo alcune variazioni:

- Tutte le variabili dovrebbero essere private
- Per gli if, else, while, for…il cui corpo è vuoto o contiene una riga di codice non usiamo parentesi graffe, ma l'indentazione è necessaria

- All'interno degli switch, i case che contengono più di una riga di codice devono avere il corpo racchiuso tra parentesi graffe, per favorire la leggibilità
- All'inizio di ogni classe devono essere presenti alcune informazioni contenute in commenti che rispettano lo stile Javadoc (capitolo 7 del sito indicato)

1.3.2 File XML

Nella stesura dei file XML, per quanto riguarda la struttura facciamo riferimento alle regole documentate da Google al sito https://google.github.io/styleguide/xmlstyle.html
Ogni documento XML deve essere ben formato, vale a dire:

- Deve avere una sola radice
- L'annidamento dei tag deve essere corretto
- Tutti i tag aperti devono essere chiusi
- I valori degli attributi devono essere specificati tra virgolette

All'interno dell'applicazione utilizzeremo diversi file XML per diversi obiettivi (definire le componenti dell'applicazione, il layout delle interfacce utenti, dimensioni e modifiche delle View…), scritti prendendo come riferimento il vocabolario XML di Android, che mette a disposizione tag e attribute definiti.

1.3.3 DB SQL

Le tabelle del database dovrebbero essere in terza forma normale di Codd (3NF).
I nomi delle tabelle devono seguire le seguenti regole:

- essere sostantivi singolari;
- essere in maiuscolo, con gli underscore (_) al posto degli spazi;

I nomi dei campi devono seguire le seguenti regole:

- essere in snake_case;

- essere sostantivi singolari che fanno riferimento al dominio applicativo del problema;
- gli id che costituiscono la chiave primaria dovrebbero avere come prefisso ciò che identificano, cioè il nome della tabella;
- le chiavi esterne devono essere indicate col nome della tabella che referenziano seguito da _fk;
- i campi booleani dovrebbero iniziare con is_, has_ oppure does_

1.3.4 Script Python

Nella scrittura degli script in Python utilizziamo lo stile descritto nella documentazione fornita da Google al sito https://google.github.io/styleguide/pyguide.html.
Le regole base che utilizzeremo sono:

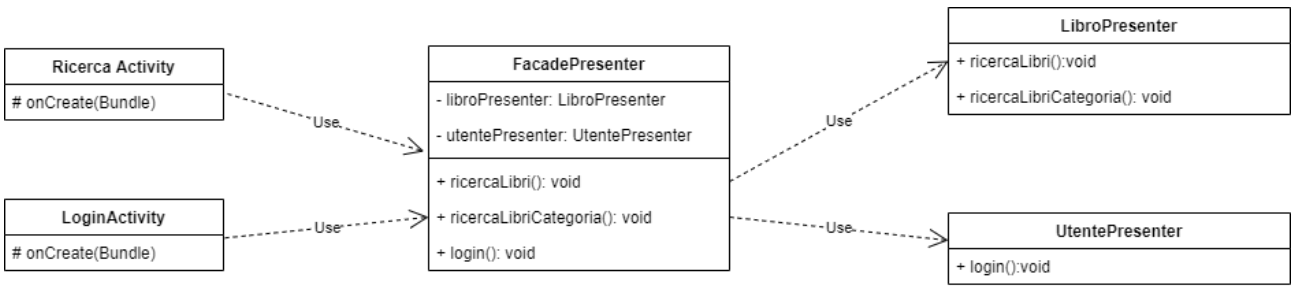
- Non terminare una linea di codice con ";" e non usarlo per inserire due comandi sulla stessa linea
- Indentare il codice con 4 spazi
- Due righe vuote tra le definizioni top-level, una riga vuota tra le definizioni dei metodi
- Gli import devono essere su linee separate
- Un solo statement per ogni riga
- Il nome dei package, dei moduli, dei metodi e delle variabili deve essere in snake-case
- Il nome delle classi e delle eccezioni deve essere in UpperCamelCase
- Il nome delle costanti deve essere scritto in maiuscolo, con le parole concatenate dall'underscore

1.4 Design Pattern e Architectural Pattern

In questa sezione vengono presentati i design pattern individuati per lo sviluppo del sistema UnisaLIB. La presentazione consiste in un'introduzione del design pattern e una descrizione del problema che si intende risolvere e della soluzione attuata, correlata dalla rappresentazione delle componenti coinvolte.

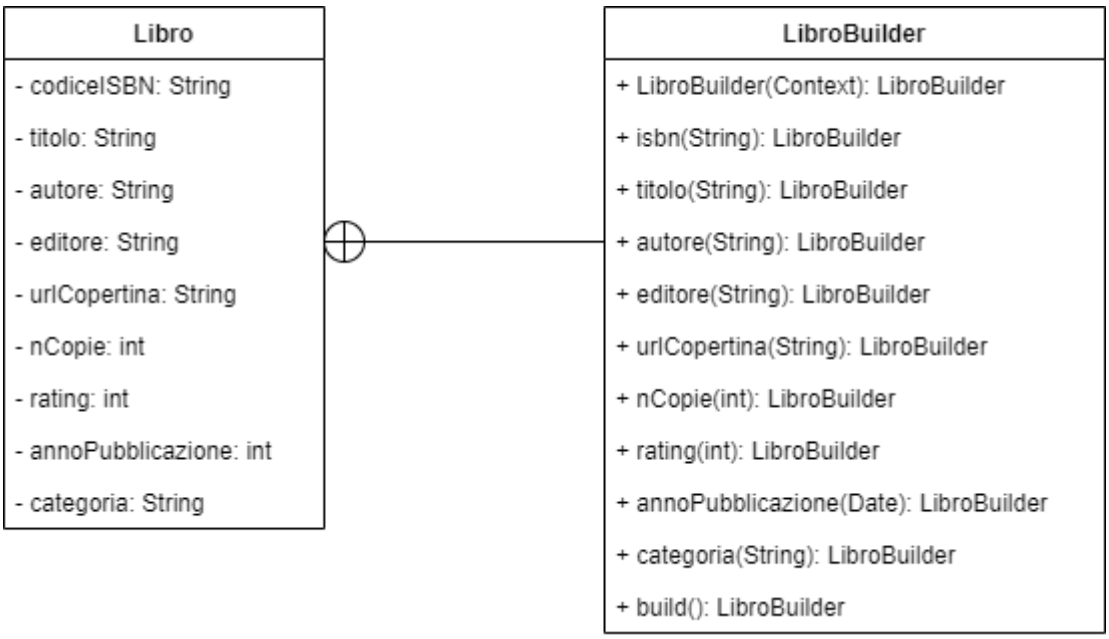
1.4.1 Facade

L'utilizzo del design pattern Facade ha come obiettivo quello di incapsulare un sistema complesso tramite un'interfaccia di facile utilizzo. In UnisaLIB viene applicato ai sottosistemi del layer Presenter, in modo da nascondere la complessità delle classi e dei metodi utilizzati e garantire quindi un basso accoppiamento. Per fare ciò si individua una classe che espone le principali operazioni offerte dai vari sottosistemi. Di seguito viene mostrato il grafico di tale applicazione:



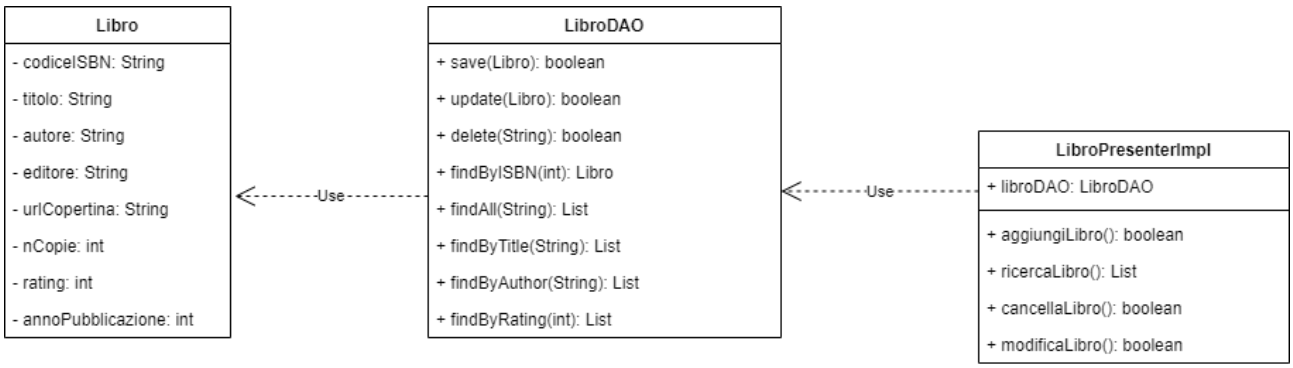
1.4.2 Builder

Il design pattern Builder viene utilizzato per astrarre il processo di istanziare un oggetto di dominio complesso. In UnisaLIB si intende utilizzare il Builder per costruire passo per passo oggetti complessi, come Libro e Prestito, un approccio utile con un'interfaccia interattiva. Ciò si traduce in una classe Builder interna alla classe dell'oggetto da istanziare, che offrirà i metodi per la costruzione dell'oggetto. Di seguito è allegato lo schema per la classe Libro:



1.4.3 DAO

Il pattern Data Access Object viene utilizzato per gestire la persistenza, in particolare ha l'obiettivo di separare la logica di business dall'accesso ai dati. In UnisaLIB intendiamo definire una classe DAO per ogni entità che permetta di effettuare le operazioni CRUD e le query. Ciò garantisce un sistema più facile da manutenere e con un maggiore livello di astrazione. Per fare ciò si individua un'interfaccia, la quale espone i metodi per operazioni query. Di seguito viene mostrata l'applicazione del DAO per l'entità Libro:



1.5 Definizioni, Acronimi e Abbreviazioni

- Framework: è un insieme di API che svolgono compiti onerosi per lo sviluppatore permettendo uno sviluppo facile e veloce del software;
- snake_case: è la pratica di scrivere gli identificatori separando le parole che li compongono tramite trattino basso (o underscore: _), solitamente con le prime lettere delle singole parole in minuscolo;
- UpperCamelCase: è la pratica di scrivere parole composte o frasi unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole. La prima lettera della prima parola è in maiuscolo;
- lowerCamelCase: è la pratica di scrivere parole composte o frasi unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole. La prima lettera della prima parola è in minuscolo;
- XML: acronimo di "Extensible Markup Language", è un metalinguaggio per la definizione di linguaggi di markup;
- DB: acronimo di "DataBase";
- SQL: acronimo di "Structured Query Language", è un linguaggio dichiarativo utilizzato per interagire con DataBase di tipo relazionale;
- RDBMS: acronimo di Relational Database Management System;
- JVM: acronimo di Java Virtual Machine
- DSL: acronimo di Domain Specific Language
- API: Application Programming Interface

2. Packages

Di seguito viene presentata la suddivisione in package del Sistema, scelta sulla base della progettazione mostrata nel SDD.
Il Sistema si compone di due nodi principali: l'applicazione mobile e l'application server.
La suddivisione in package dell'applicazione è così definita:

- **.gradle**
- **.idea**
- **app**
    - **build**, contenente i build output
    - **libs**, contenente librerie private
    - **src**, contenente tutto il codice e le risorse
        - **androidTest**
        - **main**
            - **java**, contenente il codice sorgente Java
                - **model**
                - **presenter**
                - **view**
            - **res**, contenente le risorse dell'applicazione come i file drawable e I file di layout
                - **drawable**
                - **layout**
                - **mipmap**
                - **values**
        - **test**
- **gradle**

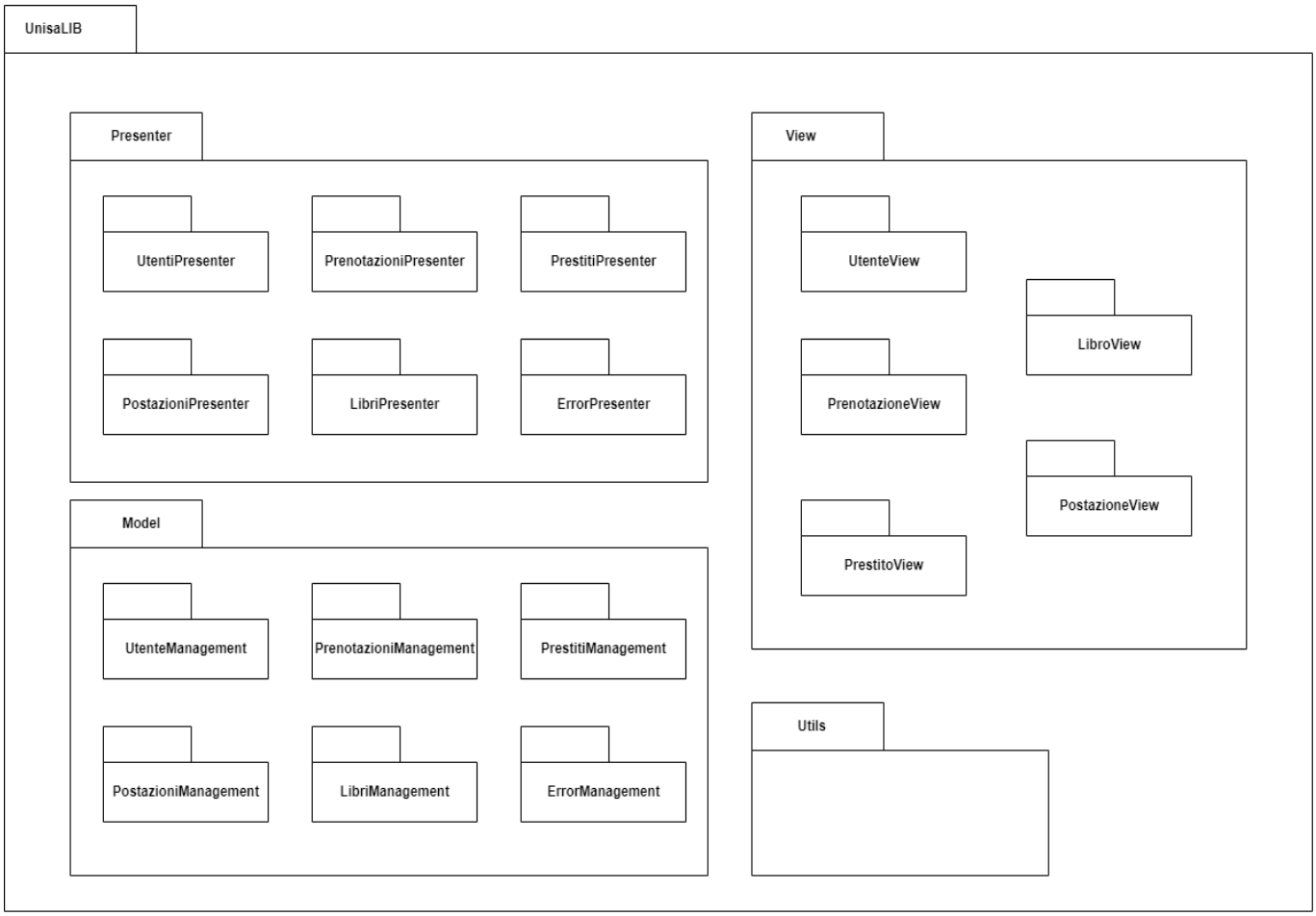
Sia all'interno dell'applicazione mobile che dell'application server la suddivisione in pacchetti è frutto della seguente scelta: si individua un pacchetto per ogni sottosistema, in modo da dividere le classi controller dalle classi model. Quindi si individuano poi due ulteriori pacchetti: controller, che racchiude tutte le classi controller, e model, che racchiude tutte le classi model.
I layout che costituiscono la View, invece, sono distinti in due soli pacchetti: uno per UtenteUnisa e uno per l'amministratore. Un ulteriore pacchetto è definito per racchiudere quelle classi utilizzate da tutti i sottosistemi.

La suddivisione in package dell'application server è così definita:

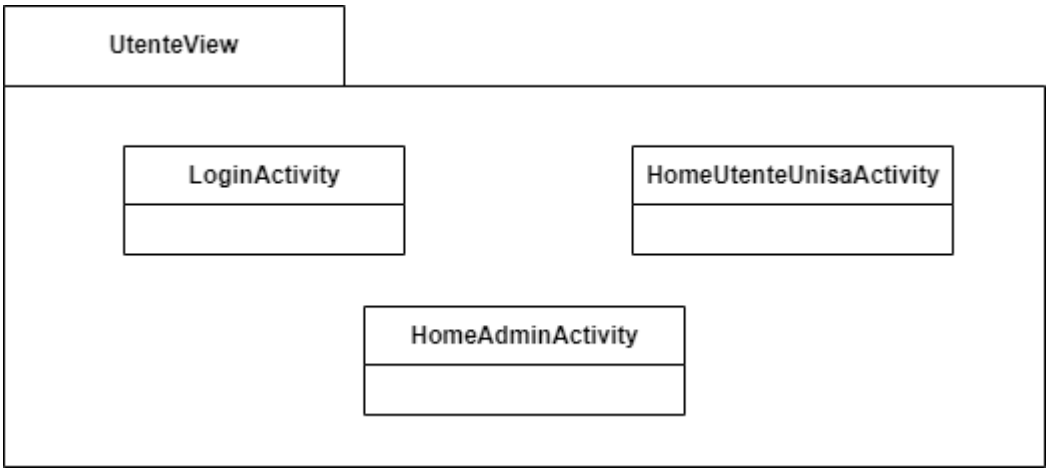
- **.idea**
- **src**, contenente il codice
    - **main**
        - **python**
        - **java**, contenente il codice sorgente java
            - **model**
            - **presenter**
    - **test**
- **target**

2.1 Package UnisaLib

In questa sezione viene mostrata la struttura del package principale dell'applicazione mobile.

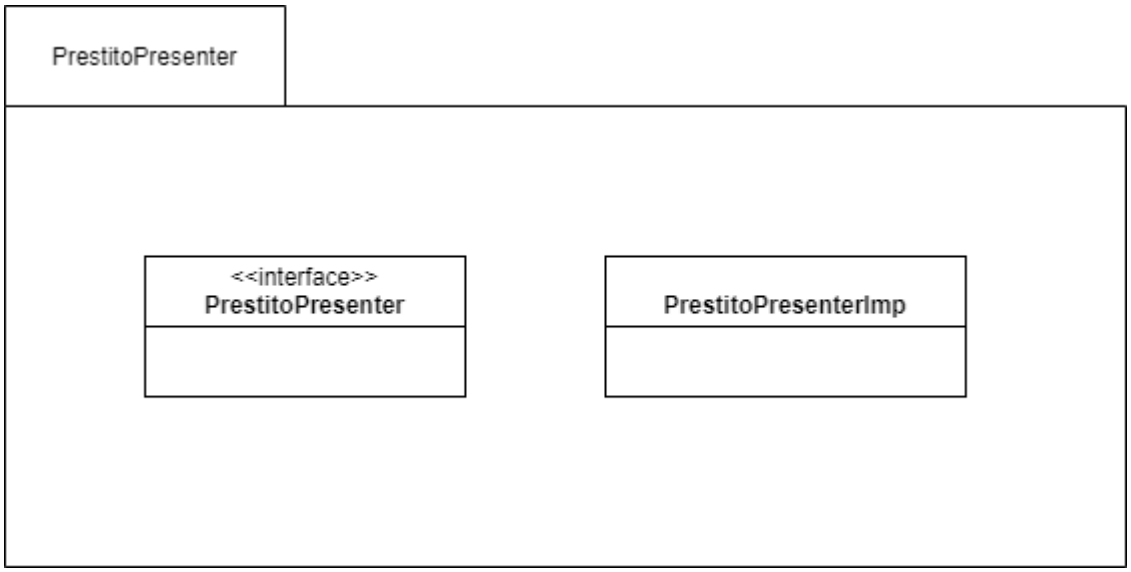


2.1.1 Package UtenteView



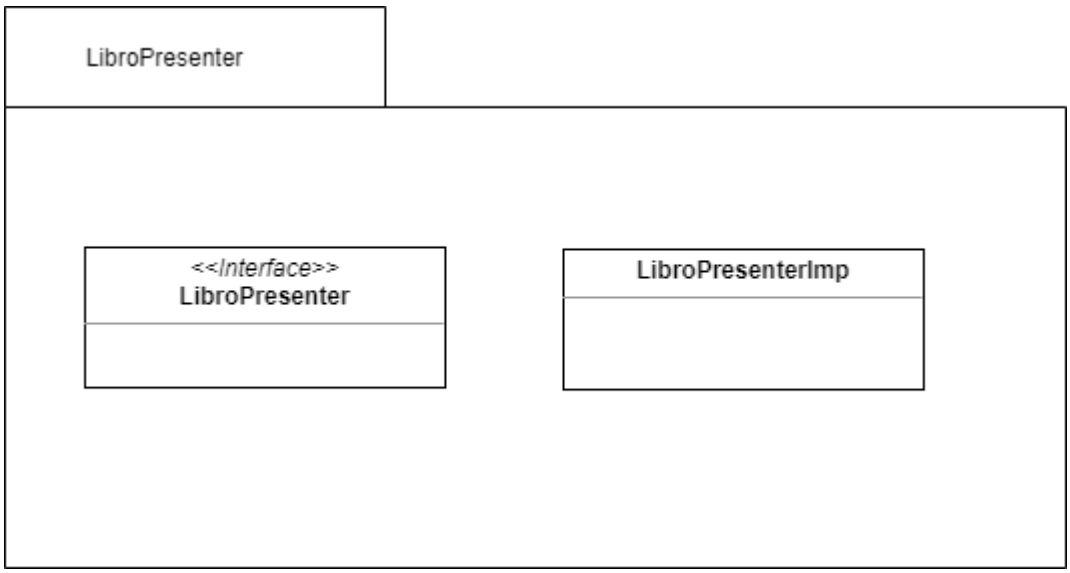
Autore: Sabato Celentano

2.1.2 Package PrestitoPresenter



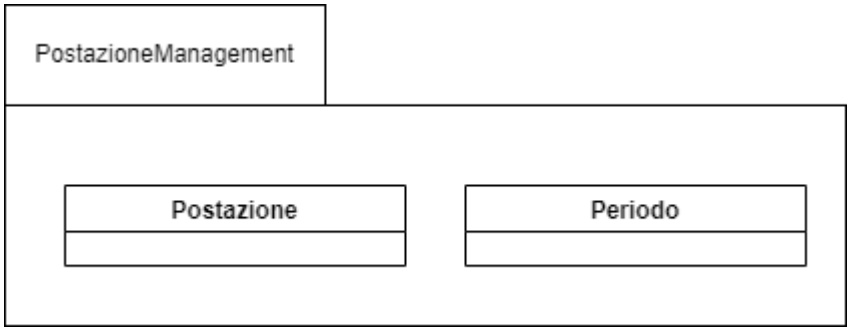
Autore: Daniele Donia

2.1.3 Package LibroPresenter



Autore: Pasquale Somma

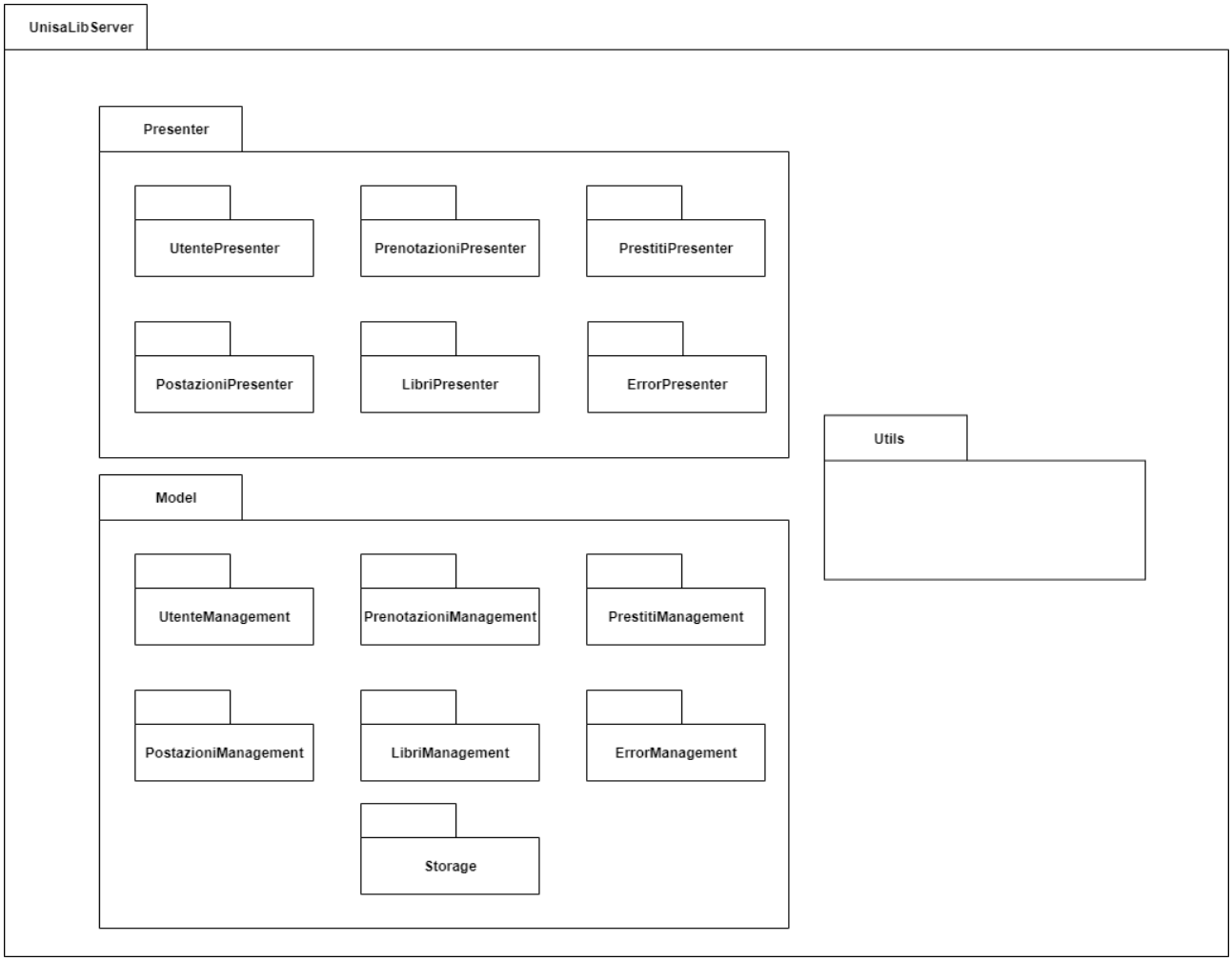
2.1.4 Package PostazioneManagement



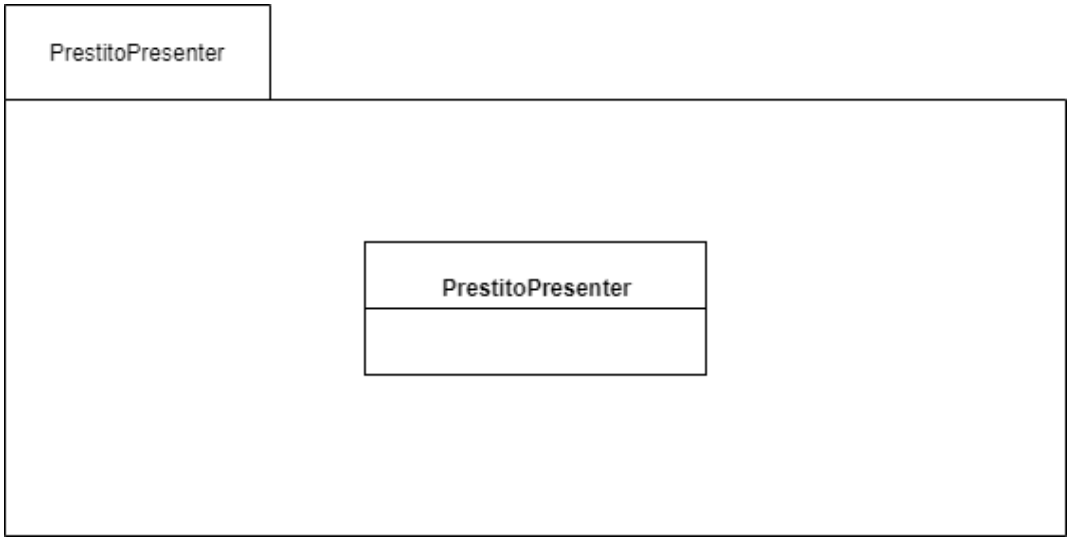
Autore: Gennaro Rascato

2.2 Package UnisaLibServer

In questa sezione viene mostrata la struttura del package principale dell'application server.

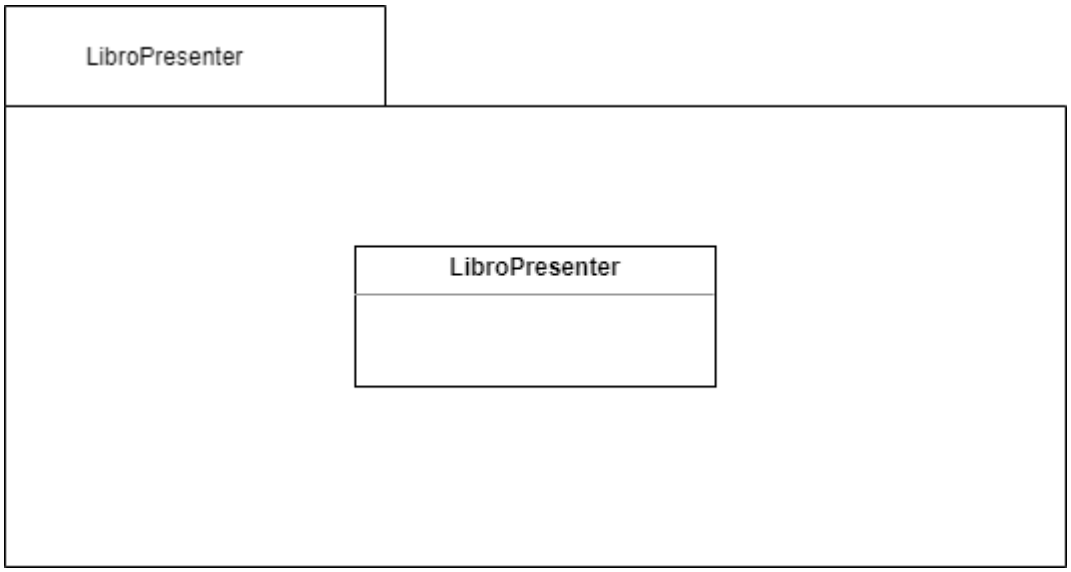


2.2.1 Package PrestitoPresenter



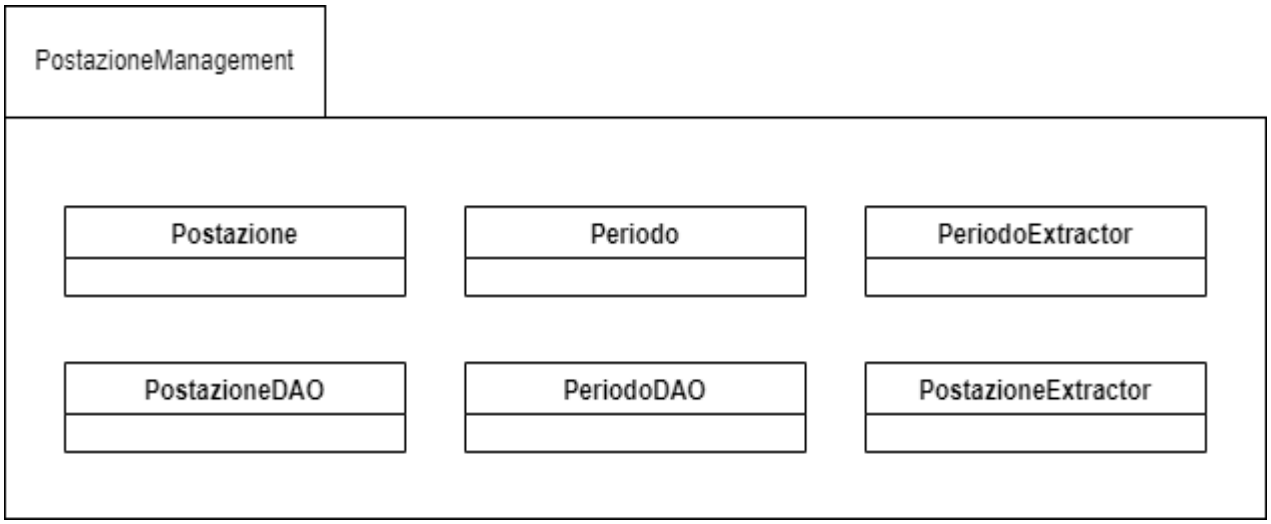
Autore: Daniele Donia

2.2.2 Package LibroPresenter



Autore: Sabato Celentano

2.2.3 Package PostazioneManagement



Autori: Gennaro Rascato, Pasquale Somma

3. Class Interfaces

3.1 Package UtenteView

UnisaLib.app.src.main.java.view.interfacciageneral.LoginActivity

Nome Classe	LoginActivity
Descrizione	Per ogni azione eseguita su general_login.xml, questa classe richiama un metodo di UtentePresenter.
Metodi	+onCreate(Bundle savedInstanceState): void +getAppContext(): Context
Invariante di classe	/

Nome Metodo	+onCreate(Bundle savedInstanceState)
Descrizione	Questo metodo viene invocato quando la main activity viene avviata e viene utilizzato per inizializzare l'activity stessa.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getAppContext( )
Descrizione	Questo metodo viene invocato all'esterno della LoginActivity per poter invocare un'altra Activity
Pre-condizione	/
Post-condizione	

3.2 Package PrestitoPresenter

UnisaLib.app.src.main.java.presenter.prestitopresenter.PrestitoPresenter

Nome Classe	PrestitiPresenter
Descrizione	Questa classe permette di gestire le operazioni sui prestiti.
Metodi	+creaPrestito(Prestito p): void +attivaPrestito(Prestito p): void +concludiPrestito(Prestito p, GregorianCalendar dConsegna): void +valutaPrestito(Prestito p, int voto, String commento): void +mostraPrestitiLibro(String isbn): void +mostraMieiPrestiti(Context c): void

Invariante di classe	/

Nome Metodo	+creaPrestito(Prestito p)
Descrizione	Questo metodo viene invocato per creare un nuovo prestito.
Pre-condizione	/
Post-condizione	**context** PrestitoPresenter::creaPrestito(p) **post:** findAll().contains(p) and p.dataFine == p.dataInizio.add(GregorianCalendar.DATE, +31)
Nome Metodo	+attivaPrestito(Prestito p)
Descrizione	Questo metodo viene invocato per attivare un prestito.
Pre-condizione	**context** PrestitoPresenter::attivaPrestito(p) **pre:** p.dataInizio – Calendar.getInstance() <=Calendar.WEEK & p.dataConsegna==null
Post-condizione	**context** PrestitoPresenter::attivaPrestito(p) **post:** p.attivo == true
Nome Metodo	+concludiPrestito(Prestito p, GregorianCalendar dConsegna)
Descrizione	Questo metodo viene invocato per concludere un prestito.
Pre-condizione	**context** PrestitoPresenter::concludiPrestito(p, dConsegna) **pre:** p.attivo== true & dConsegna!=null
Post-condizione	**context** PrestitoPresenter::concludiPrestito(p, dConsegna) **post:** p.dataConsegna == dConsegna
Nome Metodo	+valutaPrestito(Prestito p, int voto, String commento)

Descrizione	Questo metodo viene invocato per valutare il libro relativo ad un prestito.
Pre-condizione	**context** PrestitoPresenter:: valutaPrestito(p, voto, commento) **pre:** p.dataConsegna!=null & p.voto==0
Post-condizione	**context** PrestitoPresenter:: valutaPrestito(p, voto, commenti) **post:** p.voto == voto and p.commento == commento
Nome Metodo	+mostraPrestitiLibro(String isbn)
Descrizione	Questo metodo restituisce tutti i prestiti non terminati relativi ad un libro.
Pre-condizione	/
Post-condizione	**context** PrestitoPresenter::mostraPrestitiLibro(isbn) **post**: prestiti->forAll(p:Prestito | p.dataConsegna==null)
Nome Metodo	+mostraMieiPrestiti(Context c)
Descrizione	Questo metodo restituisce tutti i prestiti relativi all'utente.
Pre-condizione	/
Post-condizione	/

3.3 Package PostazionePresenter

UnisaLib.app.src.main.java.presenter.postazionepresenter.PostazionePresenter

Nome Classe	PostazioniPresenter
Descrizione	Questa classe permette di gestire le operazioni sulle postazioni.

Metodi	+bloccoIndeterminato(String pID): void +bloccoDeterminato(Postazione p, GregorianCalendar d, int i, int f): void +sbloccaPostazione (String pID): void +sbloccaPostazione (String pID, Periodo p): void + mostraRicercaPostazioni(boolean isAdmin, Context c): void + mostraElencoPostazioni(Posizione po, GregorianCalendar d): void + mostraElencoPostazioni(Context c, Posizione po): void + mostraOrariDisponibili(List<Postazione> p, String idPos, List<Prenotazione> pren, GregorianCalendar gc): List<String> + cercaBlocchiPostazione(String pID):void
Invariante di classe	/

Nome Metodo	+bloccoIndeterminato(String pID):void
Descrizione	Questo metodo viene invocato per rendere una postazione non disponibile per un periodo di tempo indeterminato.
Pre-condizione	/
Post-condizione	**context** PostazionePresenter::bloccaPostazione(pID) **post:** cercaPostazione(pID).disponibile == false
Nome Metodo	+bloccoDeterminato(Postazione p, GregorianCalendar d, int i, int f): void
Descrizione	Questo metodo viene invocato per rendere una postazione non disponibile per un periodo di tempo determinato.
Pre-condizione	**context** PostazionePresenter::bloccaPostazione(pID, p) **pre:** p.data!= null and p.oraInizio!=null and p.oraFine != null and p.oraFine>p.oraInizio
Post-condizione	**context** PostazionePresenter::bloccaPostazione(pID, p)

	post: cercaBlocchiPostazione(pID).size == @pre.cercaBlocchi().size + 1
Nome Metodo	+sbloccaPostazione(String pID):void
Descrizione	Questo metodo viene invocato per rendere una postazione nuovamente disponibile dopo un blocco indeterminato.
Pre-condizione	**context** PostazionePresenter::sbloccaPostazione(pID) **pre:** cercaPostazione(pID).disponibile == false
Post-condizione	**context** PostazionePresenter::sbloccaPostazione(pID) **post:** cercaPostazione(pID).disponibile == true
Nome Metodo	+sbloccaPostazione(String pID, Periodo p):void
Descrizione	Questo metodo viene invocato per rimuovere il blocco di una postazione in uno specifico periodo.
Pre-condizione	**context** PostazionePresenter::sbloccaPostazione(pID, p) **pre:** p.data != null and p.oraInizio !=null and p.oraFine != null and p and p.oraFine>p.oraInizio and cercaBlocchiPostazione(pID).size != 0
Post-condizione	**context** PostazionePresenter::sbloccaPostazione(pID, p) **post:** cercaBlocchiPostazione(pID).size == @pre.cercaBlocchiPostazione(pID) - 1
Nome Metodo	+ mostraRicercaPostazioni(boolean isAdmin, Context c):void
Descrizione	Questo metodo mostra le posizioni per ricercare le postazioni in base alla tipologia di utente.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+ mostraElencoPostazioni(Posizione po, GregorianCalendar d)

Descrizione	Questo metodo mostra le postazioni prenotabili dall'utenteUNISA nella data inserita.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+ mostraElencoPostazioni(Context c, Posizione po):void
Descrizione	Questo metodo mostra le postazioni relative ad una posizione dal punto di vista dell'amministratore.
Pre-condizione	/
Post-condizione	/
Nome Metodo	mostraOrariDisponibili(List<Postazione> p, String idPos, List<Prenotazione> pren, GregorianCalendar gc):List<String>
Descrizione	Mostra le fasce orarie disponibili per prenotare una postazione.
Pre-condizione	/
Post-condizione	/
Nome Metodo	cercaBlocchiPostazione(String pID):void
Descrizione	Mostra tutti i blocchi relativi ad una posizione.
Pre-condizione	/
Post-condizione	/

3.4 Package PrestitiManagement

UnisaLib.app.src.main.java.model.prestitimanagement.Prestito

Nome Classe	Prestito
Descrizione	Questa classe permette di gestire un prestito.
Metodi	+getDataInizio(): GregorianCalendar +getDataFine(): GregorianCalendar +getDataConsegna(): GregorianCalendar +getVoto(): int +getCommento(): String +isAttivo(): boolean +getLibro(): Libro +getUtente():Utente +toJSON(Prestito p): String +toJSON(List<Prestito> prestiti): String +fromJSON(JSONObject json): Prestito +fromJSON(String json): List<Prestiti>
Invariante di classe	**context** Prestito **inv**: dataFine – dataInizio == Calendar.MONTH and if attivo==true than dataConsegna==null

Nome Metodo	+getDataInizio(): GregorianCalendar
Descrizione	Questo metodo restituisce la data di inizio di un prestito.
Pre-condizione	/

Post-condizione	/
Nome Metodo	+getDataConsegna(): GregorianCalendar
Descrizione	Questo metodo restituisce la data di inizio di un prestito.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+toJSON(Prestito p): String
Descrizione	Questo metodo viene invocato per convertire un prestito in una stringa in formato JSON
Pre-condizione	/
Post-condizione	/
Nome Metodo	+toJSON(List<Prestito> prestiti): String
Descrizione	Questo metodo viene invocato per convertire una lista di prestiti in una stringa in formato JSON
Pre-condizione	/
Post-condizione	/
Nome Metodo	+fromJSON(Prestito json): Prestito
Descrizione	Questo metodo viene invocato per convertire un oggetto JSON in un prestito
Pre-condizione	/
Post-condizione	/

Nome Metodo	+fromJSON(String json): List<Prestito>
Descrizione	Questo metodo viene invocato per convertire una stringa in formato JSON in una lista di prestiti
Pre-condizione	/
Post-condizione	/

3.5 Package PostazioniManagement

UnisaLib.app.src.main.java.model.postazionimanagement.Postazioni

Nome Classe	Postazione
Descrizione	Questa classe permette di gestire una postazione.
Metodi	+getID(): int +isDisponibile(): boolean +getPosizione(): Posizione +getBlocchi(): List<Periodo> +setID(int id): void +setDisponibile(int disponibile): void +setPosizione(Posizione pos): void +toJSON(Postazione p): String +toJSON(List<Postazione> postazioni): String +fromJSONArray(JSONArray response): List<Postazione> +fromJSONArray(JSONObject json): Postazione +fromJSONArray(String json): List<Postazione> +fromJSON(String json): Postazione
Invariante di classe	/

Nome Metodo	+getID(): int
Descrizione	Questo metodo restituisce l'id della postazione.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+isDisponibile (): boolean
Descrizione	Questo metodo viene invocato per sapere se una postazione è disponibile o meno.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getBlocchi(): List<Periodo>
Descrizione	Questo metodo restituisce la lista di periodi in cui la postazione presenta un blocco
Pre-condizione	/
Post-condizione	/
Nome Metodo	+setID(int id): void
Descrizione	Questo metodo viene invocato per settare l'id di una postazione
Pre-condizione	/
Post-condizione	/

Nome Metodo	+setDisponibile(int id): void
Descrizione	Questo metodo viene invocato per settare la disponibilità di una postazione
Pre-condizione	/
Post-condizione	/
Nome Metodo	+toJSON(Postazione p): String
Descrizione	Questo metodo viene invocato per trasformare una postazione in una stringa in formato JSON
Pre-condizione	/
Post-condizione	/
Nome Metodo	+toJSON(List<Postazione> p): String
Descrizione	Questo metodo viene invocato per trasformare una lista di postazioni in una stringa in formato JSON
Pre-condizione	/
Post-condizione	/
Nome Metodo	+fromJSONArray(String json): List<Postazione>
Descrizione	Questo metodo viene invocato per trasformare una stringa formato JSON in una lista di postazioni
Pre-condizione	/
Post-condizione	/

Nome Metodo	+fromJSON(String json): Postazione
Descrizione	Questo metodo viene invocato per trasformare una stringa formato JSON in una postazione
Pre-condizione	/
Post-condizione	/

3.6 Package PostazioniManagement

UnisaLibServer.src.main.java.model.postazionimanagement.PostazioneDAO

Nome Classe	PostazioneDAO
Descrizione	Questa classe permette di gestire l'accesso ai dati persistenti delle postazioni.
Metodi	+insert(Postazione p): boolean +doRetrieveByID(String id): Postazione +doRetrieveByPosizione(String b, String z): List<Postazione> +doRetrieveDisponibiliByPosizione(String b, String z): List<Postazione> +isDisponibile(String id): int +bloccaPostazione(String id): boolean +bloccoDeterminato(Periodo pd, Postazione p): String +sbloccaPostazione(String id): boolean +sbloccaPostazione(String id, Periodo pd): boolean
Invariante di classe	/

Nome Metodo	+insert(Postazione p): boolean
Descrizione	Questo metodo viene invocato per inserire una nuova postazione nel database.
Pre-condizione	/
Post-condizione	**context** PostazioneDAO::insert(p) **post:** doRetrieveByID(p.id)!=null
Nome Metodo	+doRetrieveByID(int id): Postazione
Descrizione	Questo metodo restituisce la postazione relativa ad un id, se è presente nel database.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+doRetrieveByPosizione(Posizione p): List<Postazione>
Descrizione	Questo metodo restituisce l'elenco di postazioni relativa ad una posizione.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+doRetrieveDisponibiliByPosizione: List<Postazione>
Descrizione	Questo metodo restituisce l'elenco di postazioni disponibili relative ad una posizione.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+isDisponibile(String id): int

Descrizione	Questo metodo viene invocato per sapere se una postazione relativa ad un id è disponibile.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+bloccaPostazione(String id): boolean
Descrizione	Questo metodo viene invocato per bloccare una postazione in modo indeterminato.
Pre-condizione	/
Post-condizione	**context** PostazioneDAO::bloccaPostazione(id) **post:** doRetrieveByID(id).disponibile == false
Nome Metodo	+bloccoDeterminato(Periodo pd, Postazione p): boolean
Descrizione	Questo metodo viene invocato per bloccare una postazione in modo determinato.
Pre-condizione	/
Post-condizione	**context** PostazioneDAO::bloccoDeterminato(id) **post:** doRetrieveByID(id).blocchi.contains(pd)
Nome Metodo	+sbloccaPostazione(String id): boolean
Descrizione	Questo metodo viene invocato per rimuovere un blocco determinato sulla postazione relativi all'id
Pre-condizione	/
Post-condizione	**context** PostazioneDAO::sbloccaPostazione(id) **post:** doRetrieveByID(id).disponibile == true
Nome Metodo	+sbloccaPostazione(String id, Periodo pd): boolean

Descrizione	Questo metodo viene invocato per rimuovere un blocco determinato sulla postazione relativi all'id
Pre-condizione	/
Post-condizione	**context** PostazioneDAO::sbloccaPostazione(id) **post:** not doRetrieveByID(id).blocco.contains(pd)

3.7 Package PrestitoPresenter (Server)

UnisaLibServer.src.main.java.presenter.prestitopresenter.PrestitoPresenter

Nome Classe	PrestitoPresenter
Descrizione	Questa classe permette di gestire le operazioni sui prestiti.
Metodi	+creaPrestito(Prestito p): void +attivaPrestito(Prestito p): void +concludiPrestito(Prestito p): void +valutaPrestito(Prestito p): void +cercaPrestitiValidiPerLibro (String codiceISBN): List<Prestito> +cercaPrestitiPerUtente (String email) : void
Invariante di classe	/

Nome Metodo	+creaPrestito(Prestito p):void
Descrizione	Questo metodo viene invocato per creare un nuovo prestito.
Pre-condizione	**context** PrestitoPresenter::creaPrestito(p) **pre:** p.dataInizio <= Calendar.getIstance()
Post-condizione	**context** PrestitoPresenter::creaPrestito(p) **post:** cercaPrestitiPerUtente(p.utente.email).contains(p)

Nome Metodo	+attivaPrestito(Prestito p):void
Descrizione	Questo metodo viene invocato per attivare un prestito.
Pre-condizione	**context** PrestitoPresenter::attivaPrestito(p) **pre:** p.dataInizio – Calendar.getInstance() <=Calendar.WEEK
Post-condizione	**context** PrestitoPresenter::attivaPrestito(p) **post:** cercaPrestitiValidiPerLibro(p.libro.isbn).contains(p) and p.attivo==true
Nome Metodo	+concludiPrestito(Prestito p):void
Descrizione	Questo metodo viene invocato per concludere un prestito.
Pre-condizione	**context** PrestitoPresenter::concludiPrestito(p, dConsegna) **pre:** p.dataInizio < p.dataConsegna and p.attivo==true
Post-condizione	**context** PrestitoPresenter::concludiPrestito(p) **post:** p.dataConsegna !=null
Post-condizione	**context** PrestitoPresenter::cancellaPrestito(p) **post:** not findAll().contains(p)
Nome Metodo	+valutaPrestito(Prestito p)
Descrizione	Questo metodo viene invocato per valutare un prestito
Pre-condizione	**context** PrestitoPresenter::valutaPrestito(p) **pre:** p.getDataConsegna()!=null and (p.voto>=1 and p.voto<=5)
Post-condizione	**context** PrestitoPresenter::aggiungiVoto(p) **post:** p.getVoto()!=0
Nome Metodo	+cercaPrestitiValidiPerLibro(int codiceISBN)
Descrizione	Questo metodo restituisce tutti i prestiti attivi relativi ad un libro.

Pre-condizione	/
Post-condizione	/
Nome Metodo	+cercaPrestitiPerUtente(String email)
Descrizione	Questo metodo restituisce tutti i prestiti attivi relativi ad un utente.
Pre-condizione	/
Post-condizione	/

3.8 Package LibroPresenter (Server)

UnisaLibServer.src.main.java.presenter.libropresenter.LibroPresenter

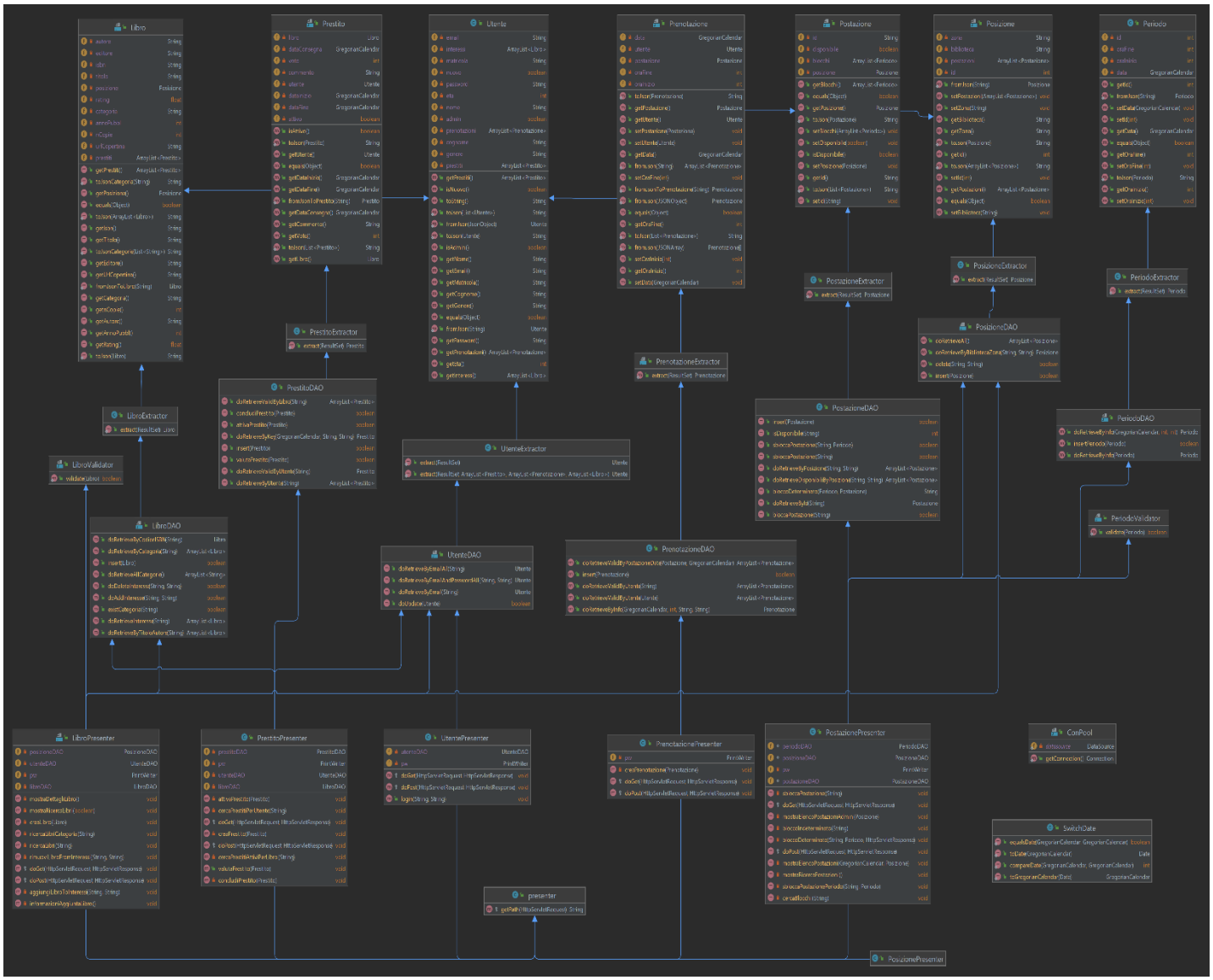
Nome Classe	LibroPresenter
Descrizione	Questa classe permette di gestire le operazioni sui libri.
Metodi	+creaLibro(Libro l): void +rimuoviLibroFromInteressi(String isbn, String email):void +aggiungiLibroToInteressi(String isbn, String email):void +ricercaLibri (String ricerca): void +ricercaLibriCategoria (String categoria): void +mostraRicercaLibri(boolean admin): void +informazioniLibro():void
Invariante di classe	/

Nome Metodo	+creaLibro(Libro l):void
Descrizione	Questo metodo viene invocato per creare un nuovo libro.
Pre-condizione	**context** LibriPresenter::modificaLibro(l) **pre:** doRetrieveByISBN(l)==null
Post-condizione	**context** LibroPresenter::creaLibro(l) **post:** doRetrieveByISBN(l)!=null
Nome Metodo	+rimuoviLibroFromInteressi(String isbn, String email):void
Descrizione	Questo metodo viene invocato per rimuovere un libro dagli interessi di un utente.
Pre-condizione	**context** LibriPresenter::rimuoviLibroFromInteressi(isbn, email) **pre:** doRetrieveUtenteByEmail(email).interessi.contains (doRetrieveLibroByCodiceISBN(isbn))
Post-condizione	**context** LibriPresenter::rimuoviLibroFromInteressi(isbn, email) **post:** not doRetrieveUtenteByEmail(email).interessi.contains (doRetrieveByCodiceISBN(isbn))
Nome Metodo	+aggiungiLibroFromInteressi(String isbn, String email):void
Descrizione	Questo metodo viene invocato per aggiungere un libro agli interessi di un utente.
Pre-condizione	**context** LibriPresenter:: aggiungiLibroToInteressi(isbn, email) **pre:** doRetrieveUtenteByEmail(email)!=null and doRetrieveByCodiceISBN(isbn)!=null
Post-condizione	**context** LibroPresenter::aggiungiLibroToInteressi(isbn, email) **post:** not doRetrieveUtenteByEmail(email).interessi.contains (doRetrieveByCodiceISBN(isbn))
Nome Metodo	+ricercaLibri(String ricerca):void

Descrizione	Questo metodo restituisce un libro in base alla ricerca specificata dall'utente.
Pre-condizione	**context** LibriPresenter::modificaLibro(l) **pre:** ricerca != null
Post-condizione	/
Nome Metodo	+ricercaLibriCategoria(String categoria):void
Descrizione	Questo metodo restituisce tutte i libri appartenenti ad una categoria.
Pre-condizione	**context** LibriPresenter::modificaLibro(l) **pre:** categoria != null
Post-condizione	/
Nome Metodo	+mostraRicercaLibri(boolean admin):void
Descrizione	Questo metodo restituisce le categorie da visualizzare nella ricerca.
Pre-condizione	/
Post-condizione	/
Nome Metodo	+informazioniLibro()
Descrizione	Questo metodo restituisce le categorie e le posizioni relative ai libri.
Pre-condizione	/
Post-condizione	/

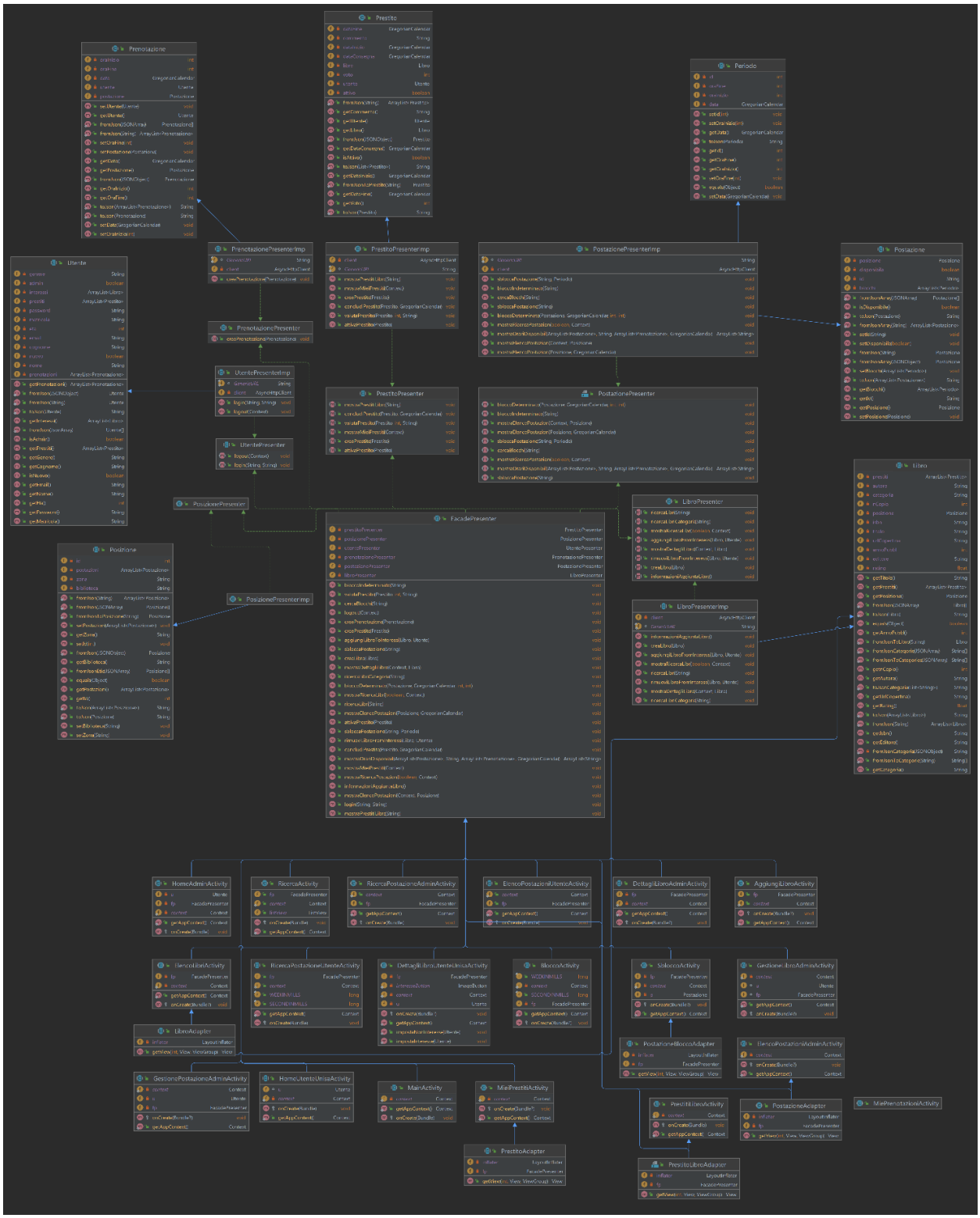
4. Class Diagram

4.1 Classi lato Server



Autore: Sabato Celentano

4.1 Classi lato Client



Autore: Gennaro Rascato

5. Glossario

Linebreak: ritorno a capo

Statement: nella programmazione, uno statement è un'unità sintattica di un linguaggio di programmazione che esprime alcune azioni da eseguire.

Database relazionale: una raccolta di elementi dati tra i quali sussistono relazioni predefinite. Questi elementi sono organizzati sotto forma di set di tabelle con righe e colonne.

Feedback implicito: è un feedback che viene acquisito automaticamente senza uno sforzo dell'utente.

Feedback esplicito: è un feedback che viene dato coscientemente ed esplicitamente dall'utente.

Maven: uno strumento di gestione di progetti software basati su Java e
build automation. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo.

Groovy: un linguaggio di programmazione ad oggetti per la Piattaforma Java alternativo al linguaggio Java. Può essere visto come linguaggio di scripting

Server web: un'applicazione software che, in esecuzione su un server, è in grado di gestire le richieste di un client; la comunicazione tra server e client avviene tramite il protocollo HTTP.

Servlet: oggetti scritti in linguaggio Java che operano all'interno di un server web oppure un server per applicazioni permettendo la creazione di applicazioni web.

Applicazione web: indica genericamente tutte le applicazioni distribuite ovvero applicazioni accessibili/fruibili via web per mezzo di un network.