UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Informatica, a.a. 2022-23
Progetto del corso di Ingegneria del Software
prof. A. De Lucia, prof. M. De Stefano
Repository GitHub: https://github.com/giocolella/is-bookbearer-22-23

- Object Design Document |

Versione 1.0

Coordinatore del progetto:

Nome	**Matricola**
De Lucia Andrea	
De Stefano Manuel	
Colella Giorgio	0512105946

Partecipanti:

Nome	**Matricola**
Colella Giorgio	0512105946

Scritto da:	Colella Giorgio

Revision History

Data	Versione	Descrizione	Autore

Indice

1.Introduzione

In questo documento si delimita l'aspetto implementativo del sistema. Si cerca di integrare ed espandere le funzionalità individuate nei documenti precedenti definendo le operazioni delle classi, i trade-off, l'impacchettamento, ecc…

1.1Trade-off

Comprensibilità vs Tempo:
Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice avrà quindi una nomenclatura comprensibile. Inoltre, si cercherà di accompagnarlo con commenti che ne semplifichino la comprensione. Nel caso in cui questo non venga fatto in tempo per la release, si provvederà successivamente.

Prestazioni vs Costi:
Essendo il progetto sprovvisto di un minimo budget, si utilizzeranno componenti gratuite. Tuttavia, Firebase è a pagamento ma comunque molto economico. Si spera che i profitti permettano di pagare Firebase che comunque è inizialmente gratuito.

RESPONSE TIME vs HARDWARE:
Il sistema garantisce una certa reattività alle richieste, e quindi è in grado di poter comunque offrire una contemporaneità di servizi agli utenti. Ovviamente questa caratteristica sarà limitata dall'hardware del sistema.

Sicurezza vs Efficienza:
La sicurezza è importante nel sistema ma non fondamentale non dovendo gestire dati molto sensibili come carte di credito ed indirizzi abitativi. Tuttavia va detto che Firebase offre un'ottima sicurezza autogestita.

Interfaccia vs Usabilità:

L’interfaccia grafica è stata realizzata in modo tale da essere molto intuitiva. Si fa uso di form e pulsanti disposti in maniera da rendere semplice l’utilizzo del sistema da parte dell’utente finale.

1.2Linee guida per la documentazione delle interfacce

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

Naming Convention
È buona norma utilizzare nomi:
- Descrittivi;
- Di uso comune;
- Lunghezza medio-corta;
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9);
- È possibile utilizzare il carattere underscore “_”, nel caso di variabili costanti o proprietà statiche.

Variabili:
I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Devono essere tutte allineate per facilitare la leggibilità.

Esempio: numeroPagine Esempio: IMAGE_CODE

Metodi:
I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica una azione, seguito dal nome di un oggetto. I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile().

Esempio: getEmail(), setEmail(…)
La descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

Classi:
I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di quest'ultime devono fornire informazioni sul loro scopo.

Esempio: ReviewActivity.java

Layout:
I layout dovranno essere ben indentati e i loro nomi saranno scritti in minuscolo con un occasionale underscore per separare le parole.

1.3 Definizioni, acronimi e abbreviazioni

- RAD: Requirements Analysis Document;
- SDD: System Design Document;
- ODD: Object Design Document.

1.4 Riferimenti

Documenti SDD e RAD del progetto Book Bearer.

2.Packaging

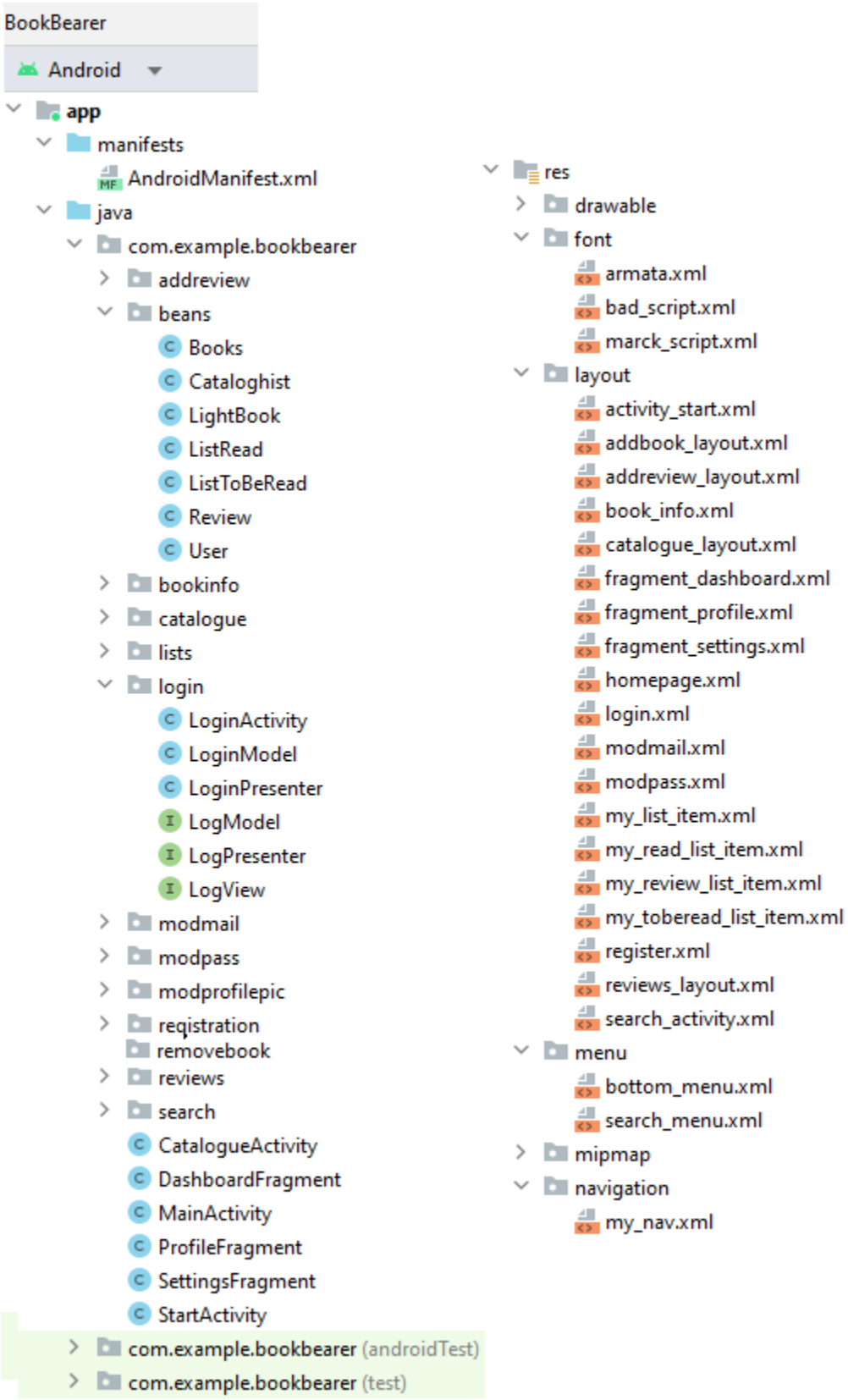
Seguendo lo stile MVP ogni funzione avrà un determinato pacchetto contenuto a sua volta in un unico pacchetto principale. Ogni pacchetto conterrà un'activity (View), un presenter ed un model. Ci saranno inoltre le tre interfacce che gli permetteranno di comunicare tra di loro. Nel caso di ListView ci saranno anche dei CustomAdapter.

Ci sono alcune activity prive di presenter e model poiché tutto quello che fanno è ridirezionare verso altre activity. Una di questa è StartActivity che non fa altro che ridirezionare alle funzioni di registrazione o login in base alle scelte dell'utente. Lo stesso vale per CatalogueActivity che serve per ridirezionare verso AddCatalogueActivity e MainActivity che serve alla ridirezione della view verso i frammenti DashboardFragment,

ProfileFragment e SettingsFragment relativi alla barra di navigazione sottostante. Questi ultimi hanno diverse funzioni relative a più pacchetti da cui sono estratti per semplicità d'uso.

La cartella layout contiene le diverse view statiche. La cartella navigation contiene la barra di navigazione sottostante. La cartella menu contiene la view statica della ricerca e alcune impostazioni della barra di navigazione sottostante. La cartella font contiene alcuni font specifici dell'app. La cartella drawable contiene alcune immagini ed icone.

Gli elementi del pacchetto "beans" vengono utilizzati per comodità all'interno di altri pacchetti. Questo viene fatto a discrezione dello sviluppatore. I pacchetti restanti sono indipendenti gli uni dagli altri.


BookBearer
Android
app
manifests
AndroidManifest.xml
java
com.example.bookbearer
addreview
beans
Books
Cataloghist
LightBook
ListRead
ListToBeRead
Review
User
bookinfo
catalogue
lists
login
LoginActivity
LoginModel
LoginPresenter
LogModel
LogPresenter
LogView
modmail
modpass
modprofilepic
registration
removebook
reviews
search
CatalogueActivity
DashboardFragment
MainActivity
ProfileFragment
SettingsFragment
StartActivity
com.example.bookbearer (androidTest)
com.example.bookbearer (test)
res
drawable
font
armata.xml
bad_script.xml
marck_script.xml
layout
activity_start.xml
addbook_layout.xml
addreview_layout.xml
book_info.xml
catalogue_layout.xml
fragment_dashboard.xml
fragment_profile.xml
fragment_settings.xml
homepage.xml
login.xml
modmail.xml
modpass.xml
my_list_item.xml
my_read_list_item.xml
my_review_list_item.xml
my_toberead_list_item.xml
register.xml
reviews_layout.xml
search_activity.xml
menu
bottom_menu.xml
search_menu.xml
mipmap
navigation
my_nav.xml

2.1Packaging back-end

addreview	
Classe	Descrizione
AddReviewActivity	La view che mostra l'aggiunta di una recensione.
AddReviewPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
AddReviewModel	Il model che prima controlla se l'utente non ha già aggiunto una recensione al libro scelto e, in caso negativo, inserisce la recensione e, in caso positivo, chiede al presenter di informare la view che la recensione non può essere aggiunta.

beans	
Classe	Descrizione
Books	Contiene tutte le informazioni di un libro.
LightBook	Contiene solo alcune informazioni su un libro. Esiste per semplicità ed ottimizzazione.
ListRead	Contiene informazioni sulla lista dei libri già letti.
ListToBeRead	Contiene informazioni sulla lista dei libri da leggere.
Review	Contiene informazioni sulle recensioni.
User	Contiene informazioni sull'utente.

bookinfo	
Classe	Descrizione

BookInfoActivity	La view che mostra la pagina delle informazioni del libro.
BookInfoPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
BookInfoModel	Il model che permette di ottenere le informazioni di un libro selezionato e di inserirlo nella lista dei libri da leggere nel dopo aver controllato che non sia già presente in tale lista o in quella dei libri già letti.

catalogue	
Classe	Descrizione
AddCatalogueActivity	La view che si occupa di mostrare al cataloghista i campi per l'aggiunta di un libro al catalogo.
CataloguePresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
CatalogueModel	Il model che si occupa di aggiungere un libro nel database dopo aver controllato che non sia già presente.

lists	
Classe	Descrizione
ListReadAdapter	CustomAdapter per la ListView usato per mostrare la lista dei libri già letti.
ListToBeReadAdapter	CustomAdapter per la ListView usato per mostrare la lista dei libri da leggere.
ListsPresenter	Il presenter che gestisce la comunicazione con il model facendo, se

	serve, alcuni controlli dell'input dell'utente.
ListsModel	Il model che aggiunge un libro ad una delle due liste oppure lo cancella da una delle due liste.

login	
Classe	Descrizione
LoginActivity	La view che mostra il form per il login.
LoginPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
LoginModel	Il model che permette il login distinguendo tra normale iscritto e cataloghista.

modmail	
Classe	Descrizione
ModMailActivity	La view che permette all'iscritto di modificare l'email.
ModMailPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
ModMailModel	Il model che permette di modificare l'email.

modpass	
Classe	Descrizione
ModPassActivity	La view che permette all'iscritto di modificare la password.
ModPassPresenter	Il presenter che gestisce la comunicazione con il model facendo, se

	serve, alcuni controlli dell'input dell'utente.
ModPassModel	Il model che permette di modificare la password.

modprofilepic	
Classe	Descrizione
ModProfilePicPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
ModProfilePicModel	Il model che permette di modificare l'immagine di profilo.

registration	
Classe	Descrizione
RegistrationActivity	La view che permette all'ospite di iscriversi.
RegistrationPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
RegistrationModel	Il model che permette all'ospite di registrarsi.

removebook	
Classe	Descrizione
RemoveBookActivity	La view che permette la rimozione del libro dal catalogo.
RemoveBookPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.

RemoveBookModel	Il model che permette la rimozione del libro dal catalogo.
RemoveBookAdapter	CustomAdapter per la ListView usato per mostrare i libri del catalogo nella ricerca del libro da eliminare.

reviews	
Classe	Descrizione
ReviewsActivity	La view che permette all'iscritto di leggere le recensioni di un determinato libro.
ReviewsPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
ReviewsModel	Il model che permette di recuperare le recensioni dal database.

search	
Classe	Descrizione
ListAdapter	La view del CustomAdapter della ricerca del libro.
SearchActivity	La view che permette all'iscritto di cercare uno specifico libro.
SearchPresenter	Il presenter che gestisce la comunicazione con il model facendo, se serve, alcuni controlli dell'input dell'utente.
SearchModel	Il model che permette di cercare un libro nel database in base al titolo.

com.example.bookbearer	
Classe	Descrizione

CatalogueActivity	La view che permette al cataloghista di essere ridirezionato verso la view relativa all'aggiunta di un libro al catalogo.
DashboardFragment	La view che permette all'utente di vedere l'home page con le liste dei libri già letti e dei libri da leggere.
MainActivity	La view che contiene i frammenti per la barra di navigazione sottostante.
ProfileFragment	La view che permette all'utente di vedere il proprio profilo e modificare l'immagine di profilo.
SettingsFragment	La view che permette all'iscritto di uscire dall'account o eliminarlo e di essere ridirezionato verso le view per la modifica dell'email e della password.
StartActivity	La view di partenza che permette di ridirezionare l'ospite verso le view per la registrazione e login.

2.2Packaging front-end

Il drawable contiene icone fornite da Android Studio ed immagini provenienti da fonti esterne.

Il font contiene dei font specifici provenienti da fonti esterne.

La cartella navigation contiene la barra di navigazione sottostante.

La cartella menu contiene la view statica della ricerca e alcune impostazioni della barra di navigazione sottostante.

layout	
XML	Descrizione
activity_start.xml	La view statica di StartingActivity.
addbook_layout.xml	La view statica di AddBookActivity.
addreview_layout.xml	La view statica di AddReviewActivity.

book_info.xml	La view statica di BookInfoActivity.
catalogue_layout.xml	La view statica di CatalogueActivity.
fragment_dashboard.xml	La view statica di DashboardFragment.
fragment_settings.xml	La view statica di SettingsFragment.
homepage.xml	La view statica di MainActvity
login.xml	La view statica di LoginActivity.
modmail.xml	La view statica di ModMailActivity
modpass.xml	La view statica di ModPassActivity.
my_list_item.xml	La view statica relativa alla ListView di SearchActivity.
my_read_list_item.xml	La view statica relativa alla ListView della lista dei libri già letti di DashboardFragment.
my_review_list_item.xml	La view statica relativa alla ListView della lettura delle recensioni di ReviewActivity.
my_toberead_item_list.xml	La view statica relativa alla ListView della lista dei libri da leggere di DashboardFragment.
register.xml	La view statica di RegistrationActivity.
reviews_layout.xml	La view statica di ReviewsActivity.
search_activity.xml	La view statica di SearchActivity.

3.Class interface

Facendo riferimento al RAD è chiaro che tutte le classi che utilizzano Firebase Authentication possiedono l'invariante per cui l'utente deve essere autenticato e cioè:

context (nome classe) inv: FirebaseUser.getInstance().getCurrentUser() <> null

Inoltre tutte le classi che usano il database possiedono l'invariante per cui l'istanza del database è diversa da null e cioè:

context (nome class) inv: FirebaseFirestore.getInstance() <> null

Considerando la struttura MVP ogni View avrà l'invariante per cui l'istanza del Presenter dovrà essere diversa da null e cioè:

context (view classe) inv: self.presenter <> null

Ogni Presenter avrà l'invariante per cui l'istanza della View e quella del Model dovranno essere diverse da null e cioè:

context (presenter classe) inv: self.view <> null

context (presenter classe) inv: self.model <> null

Infine ogni Model avrà l'invariante per cui l'istanza del presenter dovrà essere diversa da null e cioè:

context (model classe) inv: self.presenter <> null

Molti dei metodi del Presenter servono per la comunicazione tra View e Model. Questi sono metodi void molto semplici e derivabili dal funzionamento della struttura MVP quindi verranno documentati solo quelli effettivamente interessanti.

Và ricordato infine che il database Firestore trasferisce i dati in oggetti Map<String,Object>.

3.1Metodi standard utilizzati

-Ogni Activity possiede questo metodo

Nome metodo	onCreate
Descrizione	Primo metodo chiamato nel ciclo di vita di ogni activity.
Signature	void onCreate(Bundle savedInstanceState)
Pre-condizione	L'activity è nello stato "distrutta".
Post-condizione	L'activity è nello stato "creata" e le singole view sono state inizializzate.

-StartActivity fa l'override di questo metodo

Nome metodo	onStart
Descrizione	Metodo chiamato durante il ciclo di vita di ogni activity.
Signature	void onStart()
Pre-condizione	L'activity è nello stato "create" o "reinizializzata"
Post-condizione	L'activity è nello stato "inizializzata" e se l'utente è autenticato lo spedisce alla MainActivity

-Ogni Activity possiede questo metodo

Nome metodo	…Message
Descrizione	Metodo chiamato per visualizzare un Toast.
Signature	void …Message(String msg)
Pre-condizione	Nessun Toast è visualizzato.
Post-condizione	Un Toast con la stringa msg viene visualizzato.

-Ogni CustomAdapter possiede questo metodo

Nome metodo	getView
Descrizione	Metodo chiamato per definire la struttura di un ListView customizzato.
Signature	View getView(int position, View convertView, ViewGroup parent)
Pre-condizione	Esiste un ListView statico.
Post-condizione	Il ListView statico è stato dinamicamente popolato.

-Ogni Frammento possiede questo metodo

Nome metodo	onCreateView
Descrizione	Metodo chiamato durante il ciclo di vita di un frammento.

Signature	View onCreateView(LayoutInflater inflater,ViewGroup container,Bundle savedInstanceState)
Pre-condizione	Il frammento è in stato “creato”
Post-condizione	Il frammento è in stato inizializzato e le singole view sono state inizializzate.

3.2addreview

Classe	AddReviewActivity

Nome metodo	reviewAdded
Descrizione	Metodo chiamato per visualizzare un Toast e terminare l’activity.
Signature	void reviewAdded()
Pre-condizione	Nessun Toast è visualizzato e l’activity non è in stato “distrutta”
Post-condizione	context AddReviewActivity::reviewAdded() post: self.addReviewMessage("Recensione aggiunta") and self.isFinishing() = true

Classe	AddReviewModel

Nome metodo	addReview
Descrizione	Metodo chiamato per aggiungere una recensione ad un libro nel database.
Signature	void addReview(String ISBN,Review review)
Pre-condizione	La recensione aggiunta è la prima da parte dell’utente in un determinato libro. context AddReviewModel::addReview(ISBN: String, review: Review) pre:

	ISBN <> null and review <> null
Post-condizione	context AddReviewModel:: addReview(ISBN: String, review: Review) post: db.collection("Books").document(ISBN).collection("Reviews").document(user.getUid()).data() = m

3.3bookinfo

Classe	BookInfoActivity

Nome metodo	useBookResult
Descrizione	Metodo chiamato per visualizzare le info del libro.
Signature	void useBookResult(Map<String,Object> mp)
Pre-condizione	context BookInfoActivity::useBookResult(mp:Map<String,Object>) pre: mp <> null and mp->notEmpty() and title <> and author <> null and genere <> null and annoUscita <> null and numPagine <> null and iv <> null
Post-condizione	Context BookInfoActivity:: useBookResult(mp:Map<String,Object>) post: self.title = 'Titolo: ' + mp->at('titolo') and self.author.getText() = 'Autore: ' + mp->at('autore') and self.genere.getText() = 'Genere: ' + mp->at('genere') and self.annoUscita.getText() = 'Anno d'uscita: ' + mp->at('annoUscita') and self.numPagine.getText() = 'Numero di pagine: ' + mp->at('NumPagine') and self.iv.getImageUrl() = mp->at('bookImg')

Nome metodo	sendToAddReview

Descrizione	Metodo chiamato per passare all'activity AddReviewActivity.
Signature	void sendToAddReview()
Pre-condizione	Ci si trova in BookInfoActivity
Post-condizione	Ci si trova in AddReviewActivity e BookInfoActivity è stata aggiunta nello stack.

Classe	BookInfoModel

Nome metodo	getSelectedBook
Descrizione	Metodo chiamato per trovare le info del libro nel database.
Signature	void getSelectedBook(String ISBN)
Pre-condizione	context: BookInfoModel::getSelectedBook(ISBN:String) pre: ISBN <> null
Post-condizione	context: BookInfoModel:: getSelectedBook(ISBN:String) post: self.mp <> null and self.mp->size() = 6 and self.mp->includesAll(['titolo', 'autore', 'genere', 'annoUscita', 'NumPagine','bookImg'])

Nome metodo	addToList
Descrizione	Metodo chiamato per controllare se un libro è nella lista dei libri da leggere di un determinato utente.
Signature	void addToList(String ISBN)
Pre-condizione	context BookInfoModel::addToList(ISBN: String) pre: ISBN <> null

Post-condizione	context BookInfoModel::addToList(ISBN: String)
	post:
	if (docRef.get().getResult().exists() and docRef.get().getResult().getData().keys().includes(ISBN))
	then
	bPresen.sendMessage("Libro già in lista da leggere")
	else

	checkRead(ISBN) endif

Nome metodo	checkRead
Descrizione	Metodo chiamato per controllare se un libro è nella lista dei libri letti di un determinato utente.
Signature	void checkRead(String ISBN)
Pre-condizione	context BookInfoModel::checkRead(ISBN: String) pre: ISBN <> null

Post-condizione	context BookInfoModel::checkRead(ISBN: String)
	post:
	if (docRef.get().getResult().exists() and
	docRef.get().getResult().getData().keys().includes(ISBN))
	then
	bPresen.sendMessage("Libro già in lista letti")
	else
	addBookToList(ISBN)
	endif

Nome metodo	addBookToList
Descrizione	Metodo chiamato per aggiungere un libro alla lista dei libri da leggere di un determinato utente.
Signature	void addBookToList(String ISBN)
Pre-condizione	ISBN <> null, il libro non è nella lista dei libri da leggere e non è nella lista dei libri letti.
Post-condizione	Il libro è presente nella lista dei libri da leggere di un determinato utente.

Nome metodo	checkIfSecond

Descrizione	Metodo chiamato per controllare se la recensione che l'utente vuole scrivere ad un determinato libro è la seconda da parte sua.
Signature	void checkIfSecond(String ISBN)
Pre-condizione	context BookInfoModel::checkIfSecond(ISBN: String) pre: ISBN <> null

Post-condizione	context BookInfoModel::checkIfSecond(ISBN: String)
	post:
	if (docRef.get().getResult().exists())
	then
	bPresen.sendMessage("Recensione già aggiunta")
	else
	bPresen.sendToAR()
	endif

Nome metodo	deleteReviewDocument(String ISBN)
Descrizione	Metodo chiamato per eliminare la recensione di un utente ad un determinato libro.
Signature	void deleteReviewDocument(String ISBN)
Pre-condizione	context BookInfoModel::deleteReviewDocument(ISBN: String) pre: ISBN <> null
Post-condizione	La recensione dell'utente relativa al libro con isbn ISBN è stata cancellata.

3.4catalogue

Classe	AddCatalogueActivity

Nome metodo	bookAdded
Descrizione	Metodo chiamato per terminare l'activity.
Signature	void bookAdded()
Pre-condizione	L'activity non è in stato "distrutta"

Post-condizione	L’activity è in stato “distrutta” e ci si trova in CatalogueActivity.

Classe	CataloguePresenter
Invariante	

Nome metodo	validateBook
Descrizione	Metodo chiamato per controllare che i campi compilati nella view rispettino alcune regole.
Signature	Boolean validateBook(String ISBN,String titolo,String autore,String genre,String annoUscita,String bookImg,String numPagine)
Pre-condizione	context CataloguePresenter: pre: ISBN <> null and titolo <> null and autore <> null and genere <> null and annoUscita <> null and bookImg <> null and numPagine <> null
Post-condizione	context CataloguePresenter: post: (ISBN.size() >= 14 and annoUscita.size() >= 4 and numPagine.matches('^[0-9]*$') and annoUscita.matches('^[0-9]*$')) implies (result = true) (ISBN.size() < 14) implies (result = false) (annoUscita.size() < 4 or not annoUscita.matches('^[0-9]*$')) implies (result = false) (not numPagine.matches('^[0-9]*$')) implies (result = false)

Classe	CatalogueModel

Nome metodo	addCatalogueBook
Descrizione	Metodo chiamato per controllare se un libro è già presente nel database.
Signature	void addCatalogueBook(Books book)
Pre-condizione	context CatalogueModel: addCatalogueBook(book: Books) pre: books.* <> null

Post-condizione	context CatalogueModel: addCatalogueBook(book: Books)
	post:
	if (docRef.get().getResult().exists())
	then
	presenter.sendMessage("Libro già esistente")
	else
	addBook(book)
	endif

Nome metodo	addBook
Descrizione	Metodo chiamato per aggiungere un libro nel database.
Signature	void addBook(Map<String,Object> m,Books book)
Pre-condizione	context CatalogueModel: addBook(m: Map<String,Object>, book : Books) m.* <> null and book.* <> null
Post-condizione	Il libro viene aggiunto al database

3.5lists

Classe	ListsPresenter

Nome metodo	giveReadResults

Descrizione	Metodo chiamato per fornire a ListReadAdapter i libri della lista letti con cui popolare il ListView.
Signature	void giveReadResults(List<Map<String,Object>> l)
Pre-condizione	context ListsPresenter: giveReadResults(l : List<Map<String,Object>>) pre: l.* <> null
Post-condizione	Aggiunge i libri, se presenti, al ListView di DashboardFragment.

Nome metodo	giveToBeReadResults
Descrizione	Metodo chiamato per fornire a ListToBeReadAdapter i libri della lista da leggere con cui popolare il ListView.
Signature	void giveToBeReadResults(List<Map<String,Object>> l)
Pre-condizione	context ListsPresenter: giveToBeReadResults(l : List<Map<String,Object>>) pre: l.* <> null
Post-condizione	Aggiunge i libri, se presenti, al ListView di DashboardFragment.

Classe	ListsModel

Nome metodo	getRead
Descrizione	Metodo chiamato per ottenere i libri dalla lista dei libri già letti di un determinato utente.
Signature	void getRead()
Pre-condizione	
Post-condizione	context ListsModel::getRead() post: if (docRef.get().getResult().exists()) then presenter.giveReadResults(...)

	endif

Nome metodo	getToBeRead
Descrizione	Metodo chiamato per ottenere i libri dalla lista dei libri da leggere di un determinato utente.
Signature	void getToBeRead()
Pre-condizione	

Post-condizione	context ListsModel::getRead()
	post:
	if (docRef.get().getResult().exists())
	then
	presenter.giveToBeReadResults(…)
	endif

Nome metodo	addRead
Descrizione	Metodo chiamato per aggiungere un libro alla lista dei libri già letti di un determinato utente e allo stesso tempo cancellarlo da quella dei libri da leggere.
Signature	void addRead(Map<String,Object> m)
Pre-condizione	context ListsModel: addRead(m : Map<String,Object) m.* <> null and il libro è nella lista dei libri da leggere
Post-condizione	Aggiunge un libro alla lista dei libri già letti di un utente e lo cancella dalla lista dei libri da leggere.

Nome metodo	deleteRead
Descrizione	Metodo chiamato per cancellare un libro dalla lista dei libri già letti.
Signature	void deleteRead(Map<String,Object> m)
Pre-condizione	context ListsModel: deleteRead(m : Map<String,Object) m.* <> null and il libro è nella lista dei libri già letti.
Post-condizione	Cancella un libro dalla lista dei libri già letti.

Nome metodo	deleteToBeRead

Descrizione	Metodo chiamato per cancellare un libro dalla lista dei libri da leggere.
Signature	void deleteToBeRead(Map<String,Object> m)
Pre-condizione	context ListsModel: deleteToBeRead(m : Map<String,Object) m.* <> null and il libro è nella lista dei libri da leggere
Post-condizione	Cancella un libro dalla lista dei libri da leggere.

3.6login

Classe	LoginPresenter

Nome metodo	validateLogin
Descrizione	Metodo chiamato per controllare i campi del form di login.
Signature	boolean validateLogin(String uMail,String uPass)
Pre-condizione	context LoginPresenter: validateLogin(uMail:String,uPass:String) pre: uMail <> null and uPass <> null
Post-condizione	context LoginPresenter: validateLogin(uMail:String,uPass:String) post: (uMail.size() > 0 and uPass.size() > 0) implies (result = true) (uMail.size() = 0 or uPass.size() = 0) implies (result = false)

Classe	LoginModel

Nome metodo	loginUser
Descrizione	Metodo chiamato per effettuare l'autenticazione di un ospite.
Signature	void loginUser(String uMail,String uPass)
Pre-condizione	context LoginModel:loginUser(uMail:String,uPass:String) pre: uMail <> null and uPass <> null

Post-condizione	L’utente è autenticato

Nome metodo	checkList
Descrizione	Metodo chiamato per creare nel database le liste dei libri letti e già letti, se non esistono, relative ad un utente. Le liste vengono create solo al primo login.
Signature	void checkList(String userId)
Pre-condizione	context LoginModel: loginUser(uMail:String,uPass:String) pre: uMail <> null and uPass <> null
Post-condizione	context LoginModel: loginUser(uMail:String,uPass:String) post: (authResult <> null and auth.getCurrentUser() <> null) implies (presenter.sendMessage() = "Login effettuato!" and self.checkList()) (authResult = null or auth.getCurrentUser() = null) implies (presenter.sendMessage() = "Credenziali errate!")

Nome metodo	isCataloghist
Descrizione	Metodo chiamato per controllare se l’ospite che sta cercando di autenticarsi è un cataloghista.
Signature	void isCataloghist(String uMail,String uPass)
Pre-condizione	context LoginModel:: isCataloghist(uMail:String,uPass:String) pre: uMail <> null and uPass <> null
Post-condizione	context LoginModel:: isCataloghist(uMail:String,uPass:String) post: if (docRef.get().getResult().exists()) then rimanda a CatalogueActivity

	else
	self.loginUser(uMail,uPass)
	endif

3.7modmail

Classe	ModMailPresenter

Nome metodo	validateFields
Descrizione	Metodo chiamato per controllare i campi del form di modifica email.
Signature	boolean validateFields(String uMail,String uPass)
Pre-condizione	context ModMailPresenter:: validateFields(uMail:String,uPass:String) pre: uMail <> null and uPass <> null
Post-condizione	context ModMailPresenter:: validateFields(uMail:String,uPass:String) post: (uMail.size() > 0 and uPass.size() > 7 and uMail.contains(“@”)) implies (result = true) (uMail.size() = 0 or uPass.size() = 0) implies (result = false) (uPass.size() <8 0) implies (result = false) (not uMail.contains(“@”)) implies (result = false)

Classe	ModMailModel

Nome metodo	reAutenticateAndChangeMail

Descrizione	Metodo chiamato per re-autenticare un utente (Firebase richiede questo controllo per cambio email o cambio password) e poi modificarne l'email.
Signature	void reAutenticateAndChangeMail(String uMail,String uPass)
Pre-condizione	context ModMailModel:: reAutenticateAndChangeMail(uMail:String,uPass:String) pre: uMail<>null and uPass<>null

Post-condizione	context ModMailModel:: reAutenticateAndChangeMail(uMail:String,uPass:String)
	post:
	if(task.isSuccessful)
	cambia l'email e presenter.sendMessage() ="Email modificata"
	else
	presenter.sendMessage() ="Email già esistente"

3.8modpass

Classe	ModPassPresenter

Nome metodo	validateFields
Descrizione	Metodo chiamato per controllare i campi del form di modifica password.
Signature	boolean validateFields(String uNPass,String uOPass)
Pre-condizione	context ModPassPresenter:: validateFields(uNPass:String,uOPass:String) pre: uNPass <> null and uOPass <> null
Post-condizione	context ModPassPresenter:: validateFields(uNPass:String,uOPass:String) post: (uNPass.size() > 7 and uOPass.size() > 7 implies (result = true)

	(uNPass.size() =0 or uOPass.size() =0) implies (result = false) (uNPass.size() < 8 or uOPass.size() < 8) implies (result = false)

Classe	ModPassModel

Nome metodo	reAutenticateAndChangePass
Descrizione	Metodo chiamato per re-autenticare un utente (Firebase richiede questo controllo per cambio email o cambio password) e poi modificarne l'email.
Signature	void reAutenticateAndChangePass(String uNPass,String uOPass)
Pre-condizione	context ModPassModel:: reAutenticateAndChangePass (uNPass:String,uOPass:String) pre: uNPass<> null and uOPass<>null

Post-condizione	context ModPassModel:: reAutenticateAndChangePass (uNPass:String,uOPass:String)
	post:
	if(task.isSuccessful)
	cambia la password e presenter.sendMessage() =”Password modificata”
	else
	presenter.sendMessage() =”Password errata”

3.9modprofilepic

Classe	ModProfilePicModel

Nome metodo	uploadImage

Descrizione	Metodo chiamato per fare l'upload dell'immagine di profilo di un utente nel Firebase Storage e settarla nell'informazioni di Firebase Authentication relative all'utente.
Signature	void uploadImage(Uri imageUri)
Pre-condizione	context ModProfilePicModel:: uploadImage(imageUri:Uri) pre: imageUri <> null
Post-condizione	context ModProfilePicModel:: uploadImage(imageUri:Uri) post: l'immagine viene aggiunta a FirebaseStorage nella cartella "/profileImages"

3.10registration

Classe	RegistrationPresenter

Nome metodo	validateRegistration
Descrizione	Metodo chiamato per controllare che i campi del form di registrazione siano corretti.
Signature	boolean validateRegistration(String uName,String uMail,String uPass,String uConfPass)
Pre-condizione	context RegistrationPresenter:: validateRegistration (uName:String,uMail:String,uPass:String,uConfPass:String) pre: uName <> null and uMail <> null and uPass <> null and uConfPass <> null
Post-condizione	context RegistrationPresenter:: validateRegistration (uName:String,uMail:String,uPass:String,uConfPass:String) post: (uName.size() > 0 and uMail.size() > 0 and uPass.size() > 7 and uConfPass.size() > 0 and uMail.contains("@") and uPass.equals(uConfPass)) implies (result = true)

	(uName.size() = 0 or uMail.size() = 0 or uPass.size() = 0 or uConfPass.size() = 0) implies (result = false) (not uMail.contains("@")) implies (result = false) (uPass <> uConfPass) implies (result = false) (uPass.size() < 8) implies (result = false)

Classe	RegistrationModel

Nome metodo	registerUser
Descrizione	Metodo chiamato per registrare un utente ed inserire il suo nome utente in Firebase Autentication.
Signature	void registerUser(String uName,String uMail,String uPass)
Pre-condizione	context RegistrationModel:: registerUser(uName:String,uMail:String,uPass:String) pre: uName <> null and uMail <> null and uPass <> null

Post-condizione	context RegistrationModel::
	registerUser(uName:String,uMail:String,uPass:String)
	post:
	if (task.isSuccessful)
	then
	FirebaseUser.getInstance() <> null and
	FirebaseUser.getInstance().getDisplayName() = uName and
	FirebaseUser.getInstance().getEmail() = uMail
	else
	presenter.sendMessage() = “Email già esistente”

	endif

3.11removebook

Classe	RemoveBookModel

Nome metodo	removeBook()
Descrizione	Metodo chiamato per rimuovere un libro dal database.
Signature	void removeBook(String ISBN)
Pre-condizione	context RemoveBookModel:: removeBook(ISBN:String) pre: ISBN <> null
Post-condizione	context RemoveBookModel:: removeBook(ISBN:String) post: il libro con isbn uguale ad “ISBN” viene rimosso dal database

3.12reviews

Classe	ReviewActivity

Nome metodo	userResults
Descrizione	Metodo chiamato per settare il CustomAdapter della ListView che mostra le recensioni utilizzando dei risultati ottenuti dal database.
Signature	void userResults(ArrayList<Review> results)
Pre-condizione	context ReviewActivity::userResults(results: ArrayList<Review>) pre: results <> null
Post-condizione	context ReviewActivity::userResults(results: ArrayList<Review>) post: il ListView di ReviewActivity viene popolato con i risultati, se ci sono, provenienti da “results”

Classe	ReviewModel

Nome metodo	readReviews
Descrizione	Metodo chiamato per trovare le recensioni relative ad un libro nel database.
Signature	ArrayList<Map<String,Object>> readReviews(String rISBN)
Pre-condizione	context ReviewModel::readReviews(rISBN:String) pre: rISBN <> null

Post-condizione	context ReviewModel::readReviews(rISBN:String)
	post:
	if (task.isSuccessful())
	then
	presenter.userResult(…)
	endif

3.12search

Classe	SearchActivity

Nome metodo	useResult
Descrizione	Metodo chiamato per settare il CustomAdapter della ListView che mostra i risultati di una ricerca di un libro.
Signature	void useResult(ArrayList<LightBook> arrayLb)
Pre-condizione	context SearchActivity::useResult(arrayLb: ArrayList<LightBook>) pre: arrayLb <> null
Post-condizione	context SearchActivity::useResult(arrayLb: ArrayList<LightBook>) post: il ListView di SearchActivity viene popolato con i risultati provenienti da “arrayLb”

Classe	SearchPresenter

Nome metodo	resultArrived
Descrizione	Metodo chiamato per mandare un Toast all'utente nel caso in cui la ricerca di libri nel databse non abbia prodotto alcun risultato oppure chiamare il èrecedente metodo useResult(…) nel caso in cui ci siano risultati.
Signature	void resultArrived(ArrayList<LightBook> lb)
Pre-condizione	context SearchPresenter::resultArrived(lb: ArrayList<LightBook>) pre: lb <> null

Post-condizione	context SearchPresenter::resultArrived(lb: ArrayList<LightBook>)
	post:
	if (lb.size = 0)
	then
	view.sendMessage("Nessun risultato")
	else
	view.useResult(lb)
	endif

Classe	SearchModel

Nome metodo	giveSearched
Descrizione	Metodo chiamato per cercare uno o più libri nel database in base al titolo scritto per intero.
Signature	ArrayList<LightBook> giveSearched(String title)
Pre-condizione	context SearchModel::giveSearched(title:String) pre: title <> null
Post-condizione	context SearchModel::giveSearched(title:String) post: if(task.isSuccessful()) then

	presenter.resultArrived(...) endif

3.13com.example.bookbearer

Classe	DashboardFragment

Nome metodo	userReadResults
Descrizione	Metodo chiamato per settare il CustomAdapter della ListView che mostra i libri già letti.
Signature	void userReadResults(ArrayList<ListRead> l)
Pre-condizione	context DashboardFragment::userReadResults(l: ArrayList<ListRead>) pre: l <> null
Post-condizione	context DashboardFragment::userReadResults(l: ArrayList<ListRead>) post: il ListView di DashboardFragment viene popolato con i risultati di “l”

Nome metodo	userToBeReadResults
Descrizione	Metodo chiamato per settare il CustomAdapter della ListView che mostra i libri da leggere.
Signature	void userToBeReadResults(ArrayList<ListToBeRead> l)
Pre-condizione	context DashboardFragment::userToBeReadResults(l: ArrayList<ListToBeRead>) pre: l <> null
Post-condizione	context DashboardFragment::userToBeReadResults(l: ArrayList<ListToBeRead>) post:

	il ListView di DashboardFragment viene popolato con i risultati di “l”

Classe	ProfileFragment

Nome metodo	onActivityResult
Descrizione	Metodo chiamato per ottenere l’uri dell’immagine selezionata dall’utente attraverso un servizio fornito dal sistema Android e non da Book Bearer. Questo metodo poi chiama uploadImage(…) di ModProfilePicPresenter passandogli l’uri.
Signature	void onActivityResult(int requestCode,int resultCode,Intent data)
Pre-condizione	context ProfileFragment:: onActivityResult(requestCode:int,resultCode:int,data:Intent) pre: requestCode <> -1 and data <> null

Post-condizione	context ProfileFragment::
	onActivityResult(requestCode:int,resultCode:int,data:Intent)
	post:
	if(requestCode = 2)
	then
	presenter.uploadImage(data.getData())
	endif

Nome metodo	getExtension
Descrizione	Metodo chiamato per ottenere il tipo dell’immagine di profilo scelta dall’utente. Android si occupa di selezionare solo i tipi immagine (jpg,png,ecc…).
Signature	String getExtension(Uri uri)
Pre-condizione	context ProfileFragment::getExtension(uri:Uri) pre: uri <> null
Post-condizione	context ProfileFragment::getExtension(uri:Uri) post:

	il metodo ritorna il tipo di “uri” sotto forma di String

Nome metodo	setProfileImg
Descrizione	Metodo chiamato che usa Glide per settare nell’ImageView l’immagine di profilo scelta dall’utente.
Signature	void setProfileImg(String uri)
Pre-condizione	context ProfileFragment::setProfileImg(uri:String) pre: uri <> null
Post-condizione	context ProfileFragment::setProfileImg(uri:String) post: l’ImageView “self.iv” viene popolato con l’immagine avente Uri “uri” prelevata usando Glide da FirebaseStorage

Classe	SettingsFragment

Nome metodo	signingOut
Descrizione	Metodo chiamato per permettere all’utente di de-autenticarsi.
Signature	void signingOut()
Pre-condizione	context SettingsFragment::signingOut() pre: FirebaseUser.getInstance() <> null
Post-condizione	context SettingsFragment::signingOut() post: FirebaseUser.getInstance() = null

Nome metodo	deleteAccount
Descrizione	Metodo chiamato per cancellare l’account di un utente da Firebase Authentication e per cancellare i suoi documenti nel database come le liste dei libri già letti e dei libri da leggere.
Signature	void deleteAccount()
Pre-condizione	context SettingsFragment::deleteAccount() pre:

	FirebaseUser.getInstance() <> null and le liste dell'utente sono presenti nel database and l'utente è presente nella collezione Users del database
Post-condizione	context SettingsFragment::deleteAccount() post: FirebaseUser.getInstance() = null and le liste dell'utente non sono presenti nel database and l'utente non è presente nella collezione Users del database