SCHOOLIB – Progetto B – Applicazione Java Client/Server

Membri del progetto:

-Marco Di Capua (matricola: 727673)

-Mattia Lo Schiavo (matricola: 726690)

-Filippo Pelosi (matricola: 726602)

-Riccardo Zorzi (matricola: 726748)

INDICE

1. Descrizione del progetto

Il problema proposto consiste nel realizzare un'applicazione client/server in Java e il relativo database, al fine di amministrare una libreria.

Nello specifico l'obiettivo è la progettazione e implementazione di due applicazioni client distinte chiamate AppReader e AppLibrarian e di un modulo server denominato serverSchoolLib.
Particolare attenzione deve essere posta nello sviluppo di quest'ultima componente del sitema, che deve essere di grado di interagire con più istanze di AppReader e AppLibrarian in parallelo e mantenendo la consistenza dei dati.

I moduli client invece devono permettere agli utenti di effettuare operazioni standard quali la creazione di un account, il reset della password nel caso venga dimenticata, e il login attraverso il codice fiscale, ustilizzato come userid, e la password associata al profilo.

Le differenze principali tra i due client consistono nella tipologia di utenti a cui sono rivolte e nelle funzionalità che offrono.

AppReader infatti è destinata alla categoria di utenti denominati lettori, e offre funzionaità relative alla visualizzazione dei libri del catalogo, registrazione e cancellazione di una prenotazione, storico di prestiti e prenotazioni, modifiche dei dati personali e cancellazione del profilo.

AppLibrarian invece è destinata agli utenti bibliotecari e offre la possibilità di effettuare operazioni di gestione della libreria, come inserimento e rimozione di nuovi libri nel sistema, e servizi di analisi quali cronologia dei prestiti, visualizzazione di prenotazioni e prestiti attivi e dei prestiti sconfinanti, cioè non restituiti oltre la data massima di riconsegna.
In aggiunta questo modulo client consente di modificare le informazioni relative al proprio account e visualizzare la cassifica dei libri più letti assoluta, per categoria e inquadramento.
Come per AppReader è consentita anche la canellazione dell'account dal sitema.

Una delle caratteristiche più distintive di AppLibrarian è quella di contenere al suo interno gran parte delle funzionalità contenute da AppReader.
Specificando il codice fiscale di un utente lettore è infatti possibile autenticarsi attraverso l'account associato, e compiere varie operazioni tra le quali: prenotazione e prestito di libri, cancellazione e annullamento di prenotazioni e prestiti, visualizzazione dello storico dei prestiti erogati all'utente e cancellazione del profilo.
Tramite AppLibrarian si possono quindi effettuare operazioni per conto di un altro utente e assistere utenti lettori nella creazione di un account.

Ciascun profilo prima di poter essere utilizzato necessita del completamento di una procedura di attivazione che consiste nell'inserimento, dopo il login, di un codice di attivazione generato casualmente e inviato all'indirizzo email specificato nella fase di registrazione. L'utente ha cinque tentativi per inserire il codice corretto, dopodichè l'account associato verrà eliminato.

Il sistema deve inoltre essere provvisto di un componente che si occupi dell'invio di email che avvisino gli utenti di modifiche relative alle loro informazioni personali, della disponibilità di un libro prenotato e dell'irriperibilità di un testo con conseguente annullamento della prenotazione.

Tutti gli utenti autenticati nel sistema hanno la possibilità di effettuare ricerche parametriche per titolo, nome/cognome dell'autore e categoria per visualizzare solo determinati libri del catalogo.

Gli utenti lettori dispongono, come accennato in precenza, delle funzionaità per registrare prestiti e prenotazioni, a condizione che siano soddisfatti determinati vincoli.
Nello specifico i vicoli sono:

- Si possono avere al massimo 10 prenotazioni pendenti;
- Ogni utente può ricevere in prestito al più 5 libri;
- La durata massima dei prestiti è di 30 giorni;
- L'erogazione del prestito non comporta il superamento del numero massimo di prestiti in carico all'utente;
- L'utente non ha in carico prestiti sconfinanti, cioè non possiede libri ricevuti da oltre 30 giorni;
- la coda di prenotazioni pendenti sul libro è vuota, oppure l'identificativo dell'utente figura in cima alla coda di prenotazioni nel libro.

2. Definizione dello spazio del problema

Al fine di creare un'applicazione che rispetti i requisiti, la fase che gioca il ruolo più importante è l'analisi in dettaglio di ciò che l'applicazione deve offrire identificabile come analisi dello spazio del problema.

Analisi dei requisiti funzionali

La prima fase affrontata è stata una attenta lettura dei requisiti richiesti nei quali viene specificato che il sistema è quindi dotato di un modulo server (serSchooLib) in grado di fornire servizi back end, analisi dei dati estratti dal DBMS relazionale, moduli client (appReader&appLibrarian) in grado di fornire servizi per gli utenti.

Successivamente abbiamo realizzato la seguente raccolta delle macro operazioni

1. Gestione Utenti: Registrazione, modifica ed eliminazione profilo utente e reset password;
2. Catalogo libri: Ricerca libri tramite categoria, autori e titolo o parti del titolo, prenotazione con servizio email per la disponibilità;
3. Gestione delle prenotazioni&prestiti: Visualizzazione e cancellazione prenotazioni;
4. Gestione Utenti: tutte le funzionalità sopra citate e in aggiunta creazione e cancellazione profili utenti di terzi;
5. Gestione catalogo libri;
6. Gestione delle prenotazioni&prestiti: avviso e riconsegna prestiti;
7. Analisi di utilizzo del sistema: analisi dati dal database (storico prestiti, disponibilità libri, indicizzazione libri, prestiti in corso..).

Aspetti strutturali

Successivamente abbiamo iniziato a delineare il sitema, insteso come unione delle quattro macro-componenti individuate: server, AppLibrarian, AppReader e database.
Ciascun componente è costituisco da componenti più piccoli legati tra di loro che verrano analizzati nel dettaglio in seguito.

L'obiettivo primario è stato quello di identificarei diversi tipi di utente che possono interagire con l'applicazione, nello specifico:

- Utente Bibliotecario (o librarian)
- Utente Lettore (studente, docente, tecnico, amministrativo)

Successivamente abbiamo messo in relazione le operazioni che l'applicazione offre, identificate nella fase precedente dello studio del problema, con il tipo di utente al quale ne è consentito l'uso.

Ciò ci ha portato a individuare i diversi tipi di utente dell'applicazione accompagnati dalle relative principali operazioni fornite:

- Utente Bibliotecario: gestione del catalogo dei libri amministrati, prestiti e prenotazioni, analisi del proprio profilo;
- Utente Lettore: visualizza da remoto il catalogo dei libri presenti con l'annessa possibilità di prenotare libri e analizzare il proprio profilo lettore.

In questa fase abbiamo potuto notare come quasi la totalità delle funzionalità disponibili per l'utente lettore possono essere sfruttate anche dall'utente bibliotecario quando agisce per conto di altri utenti.

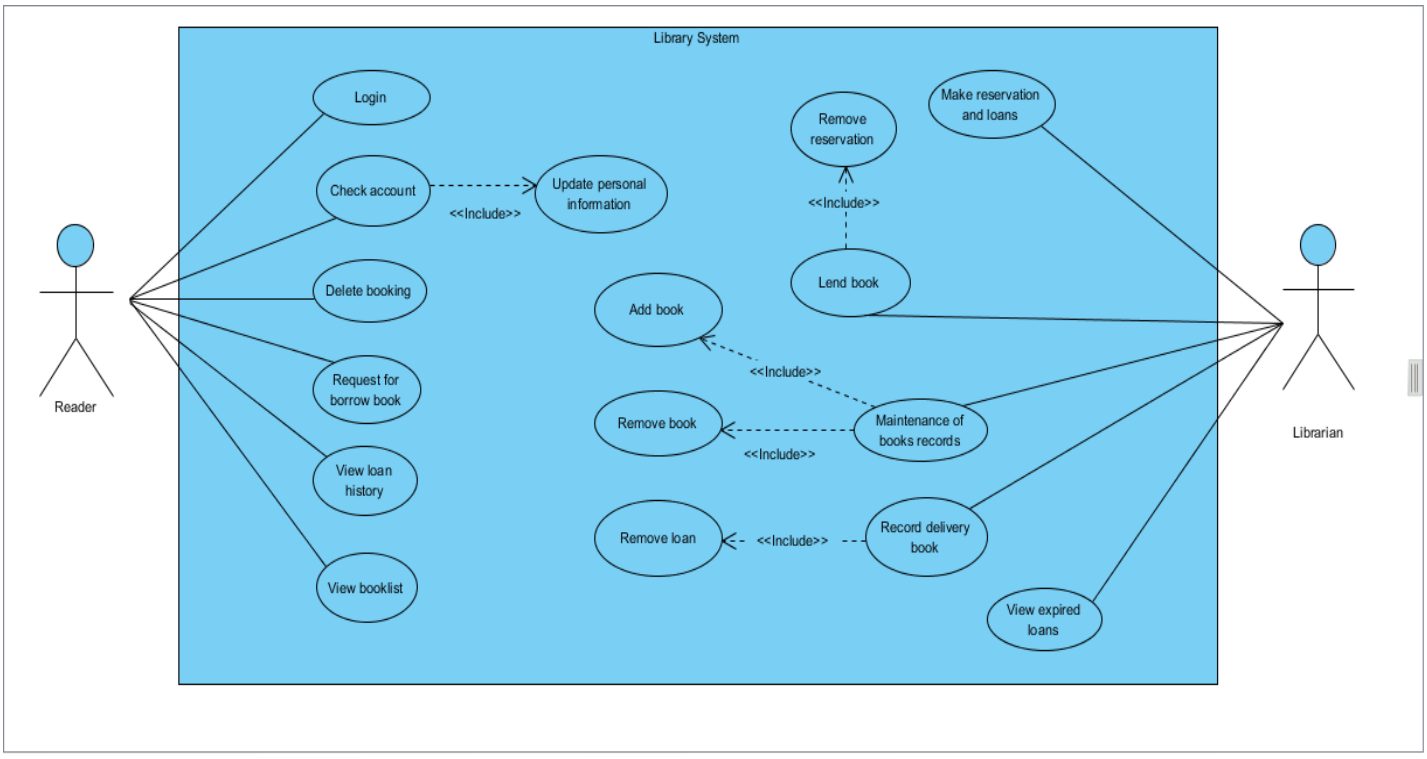
UML

Lo studio del problema ha in seguito guidato alla stesura di una documentazione formalmente descritta in linguaggio UML. Vengono prese in considerazione quindi tutte le funzionalità che l'applicazione deve implementare sotto forma di dipendenze e struttura attraverso una raccolta dati dettagliata al fine di descriverla in maniera univoca e indipendente dal linguaggio di programmazione.

La documentazione provvede quindi a definire l'architettura di quella che verrà implementata come soluzione del problema.

Use case diagram

Per prima cosa, con lo scopo di dare una vista alternativa alla descrizione del problema abbiamo deciso di realizzare lo use case diagram riportato in seguito:



Nella costruzione di questo diagramma sono stati individuati i due "attori" principali del sistema cioè l'utente lettore (indicato con reader) e quello bibliotecario (librarian). Sono state poi definite le possibili azioni di entrambi, che vengono qui sotto elencate.

Utente Reader:

- Login;
- Controlla profilo, che include aggiornare le informazioni personali;
- Cancellare una prenotazione;
- Richiedere una prenotazione;
- Visionare lo storico dei prestiti;
- Visionare a lista dei libri della libreria.

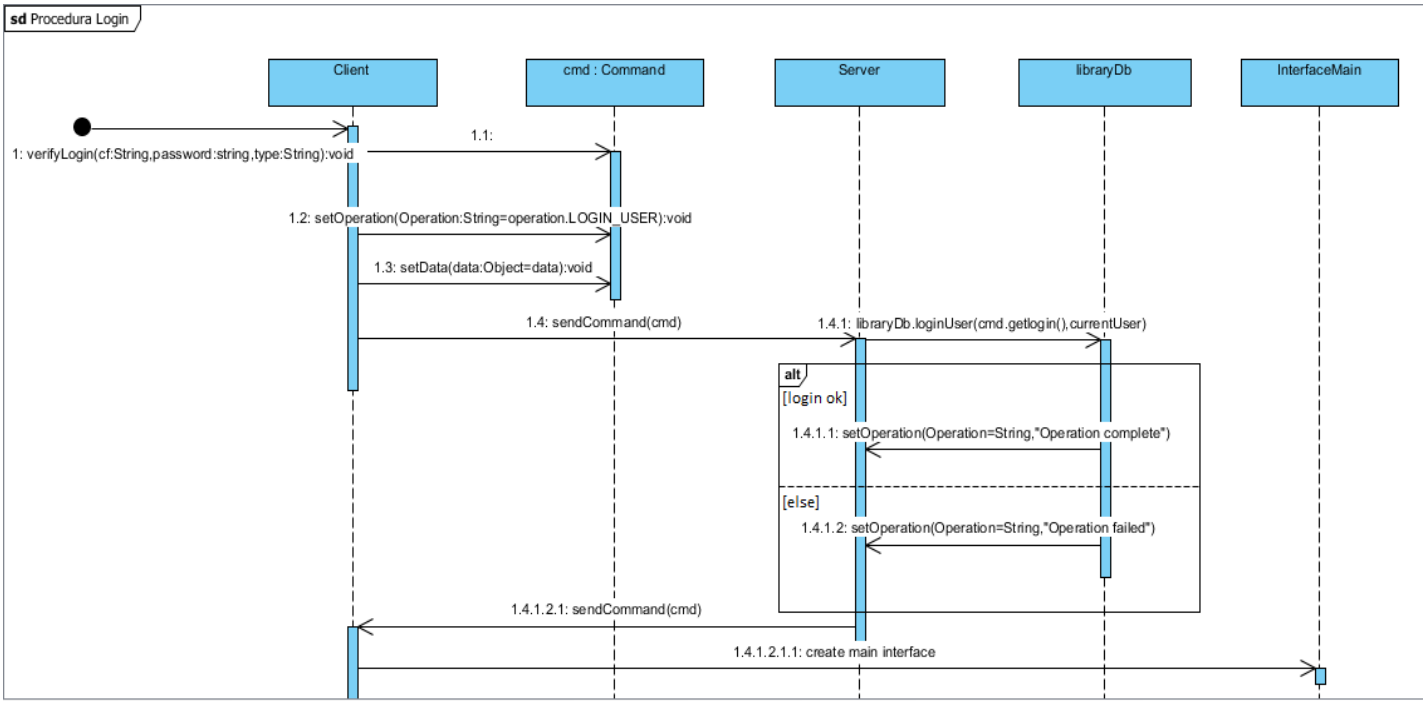
Utente Librarian:

- Registrare prenotazioni e prestiti;
- Concedere prestiti, che include rimuovere la prenotazione associata;
- Manutenzione della lista dei libri, che include aggiungere e rimuovere un libro
- Visionare i libri in prestito, che include annulare un prestito;
- Visualizzare i prestiti sconfinanti.

Sequence diagram

Successivamente sono stati realizzati i sequence diagram, relativi alle funzionalità dell'utente, che permettono di vedere quali componenti del sistema sono coinvolti nelle diverse operazione e si seguire il flusso degli eventi.

Di seguito viene riportato il sequence diagram relativo all'operazione di login:



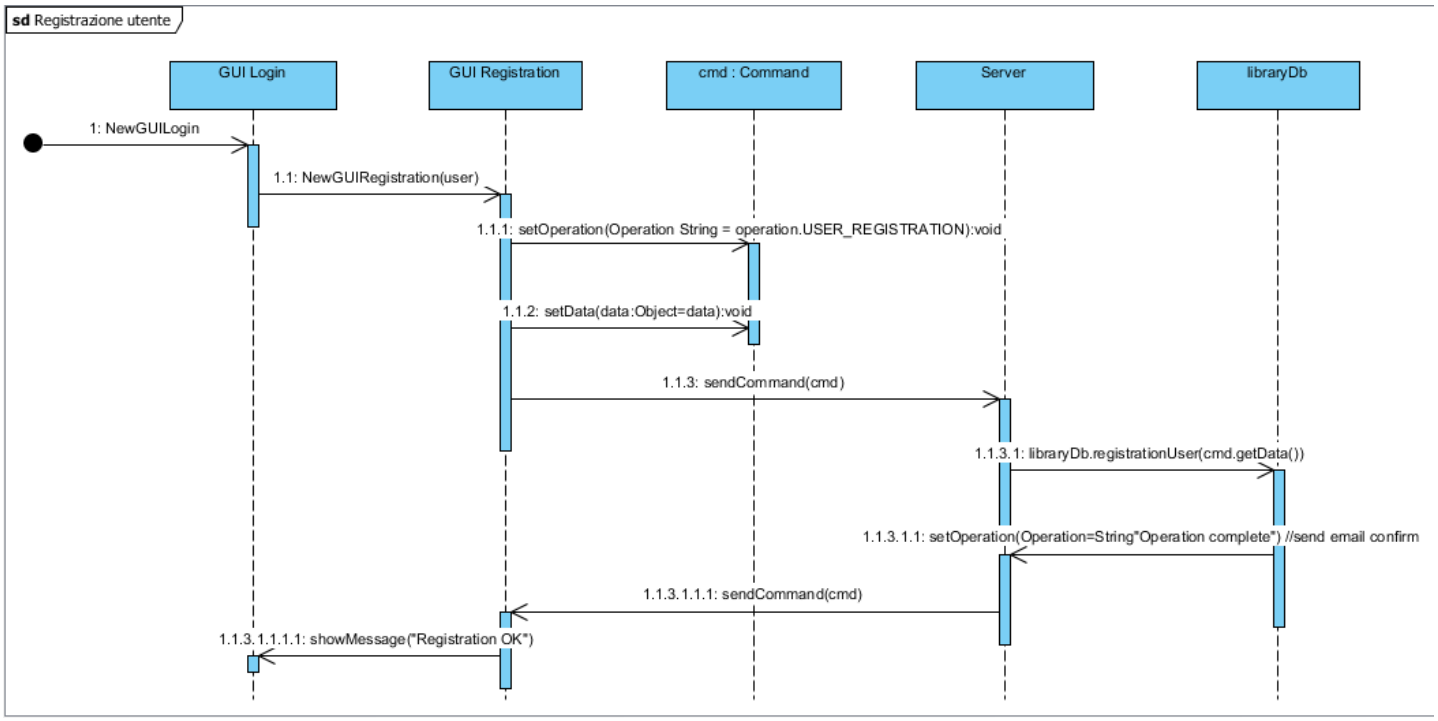
Come si può notare dal diagramma le componenti coinvolte nell'operazione di login sono:

- Client: applicazione Appreader o AppLibrarian (la procedura viene gestita nello stesso modo, con la sola differenza che in AppReader si possono loggare solo utenti di tipo lettore mentre in AppLibrarian solo bibliotecari) che, ricevuti in input i dati dell'utente, inoltra la richiesta al server sfruttando la creazione di un oggetto Command che viene inizializzato con i dati necessari per la procedura;
- Command: oggetto che contiene le informazioni necessarie per permettere al server di identificare la richiesta ed eseguirla. In questo specifico caso le informazioni inserite sono Login-User per identificare l'operazione di login e l'oggetto Login con i dati inseriti dall'utente;
- Server: serverSchoolLib che identifica la richiesta grazie al contenuto dell'operazione, processa la richiesta costruendo la query per la ricerca dell'utente e inviandola al database, e restituisce il risultato dell'interrogazione al client attraverso un altro oggetto Command;

- LibraryDb: database della libreria che ricerca una corrispondenza tra i dati inseriti in input dall'utente e le tuple della tabella users, e restituisce al server il risultato della query;
- Interface Main: interfaccia principale che viene aperta conclusa l'operazione di login.

Si può notare quindi come il flusso di esecuzione preveda che il client si connetta al server e mandi un oggetto Command con i dati identificativi della richiesta, e sia poi il server che si occupi di comunicare con il database e inviare i dati elaborati al client.

Successivamente è stato realizzato il sequence diagram per la registrazione di un nuovo utente:



In questo diagramma le entità coinvolte sono:

- GUILogin: interfaccia iniziale del programma dalla quale è possibile selezionare l'opzione per la registrazione;
- GUIRegistration: interfaccia per la registrazione con i campi che devono essere compilati dall'utente;
- Command: oggetto che contiene le informazioni necessarie per permettere al server di identificare la richiesta ed eseguirla. In questo specifico caso le informazioni inserite sono User-Registration per identificare l'operazione di login e l'oggetto Login con i dati inseriti dall'utente;

- serverSchoolLib che identifica la richiesta grazie al contenuto dell'operazione, processa la richiesta costruendo la query per l'inserimento dell'utente e inviandola al database, invia l'email con i dati della registrazione e restituisce il risultato dell'interrogazione al client attraverso un altro oggetto Command;
- LibraryDb: database della libreria che inserisce tra i dati inseriti in input registrando un nuovo profilo nel sistema;

Anche in questo caso dallo schema si può notare come il flusso di esecuzione preveda che il client, attraverso le GUI, si connetta al server e mandi un oggetto Command con i dati identificativi della richiesta, e sia poi il server che si occupi di comunicare con il database e inviare i dati elaborati al client.

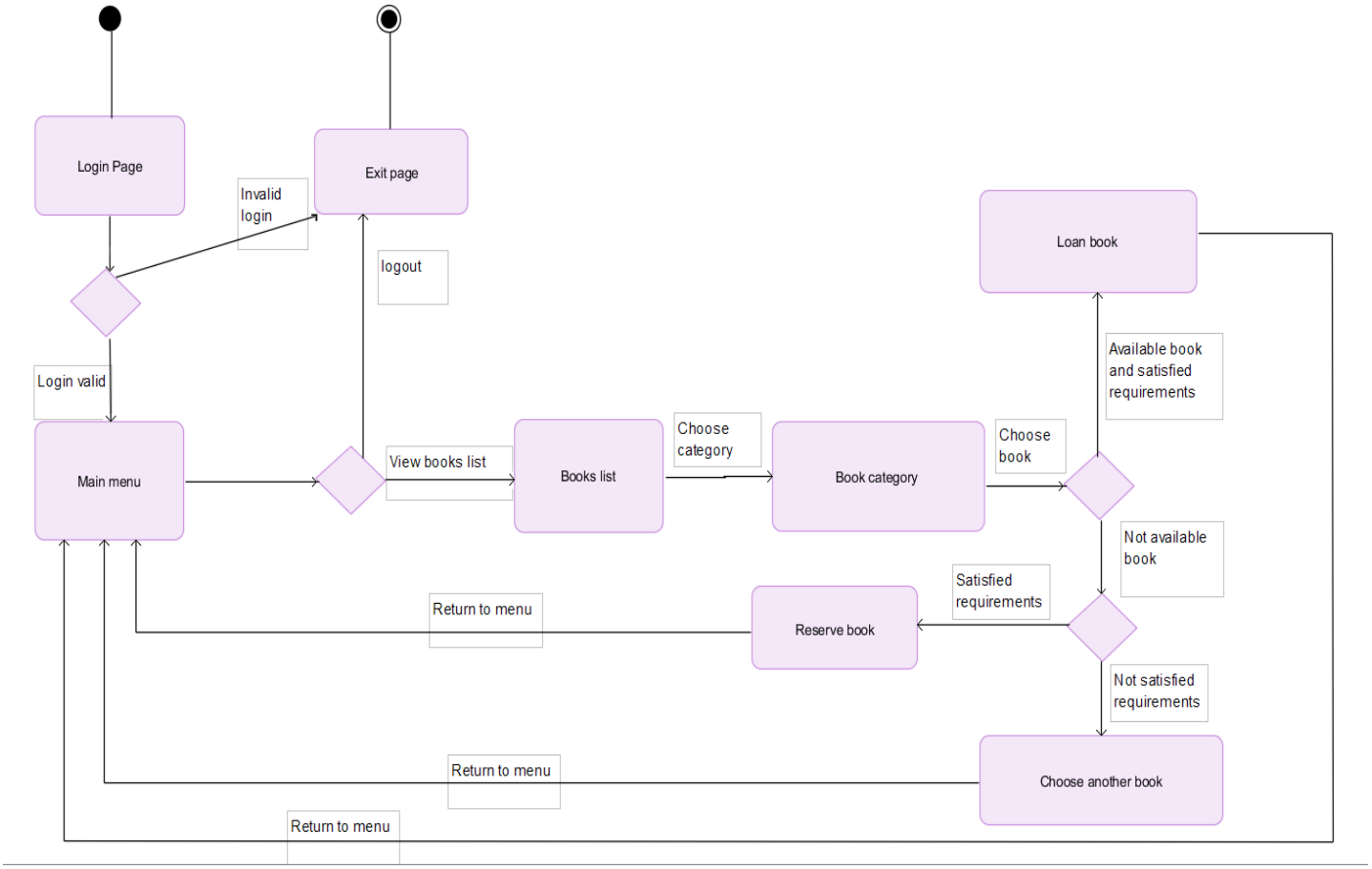
State diagram

La fase successiva è stata la creazione di state diagram che permettessero di individuare facilmente i vari stati assunti dal sistema durante l'esecuzione di una determinata operazione. Gli elementi rappresentati da uno state diagram sono lo stato (distinguendo tra iniziale, intermedio e finale), l'evento, l'azione e la guardia.

Lo stato descrive una qualità dell'entità o classe che si sta rappresentando (pratica aperta, in lavorazione, sospesa, chiusa); l'evento è la descrizione dell'azione che comporta il cambiamento di stato, l'azione è l'evento che ne consegue, la guardia è l'eventuale condizione che si deve verificare perché si possa compiere l'azione.

Gli state diagram che abbiamo realizzato sono quelli relativi alle funzionalità principali offerte dal sistema, cioè prenotazione e prestito di libri tramite AppLibrarian, prenotazioni tramite AppReader e modifica del profilo utente.

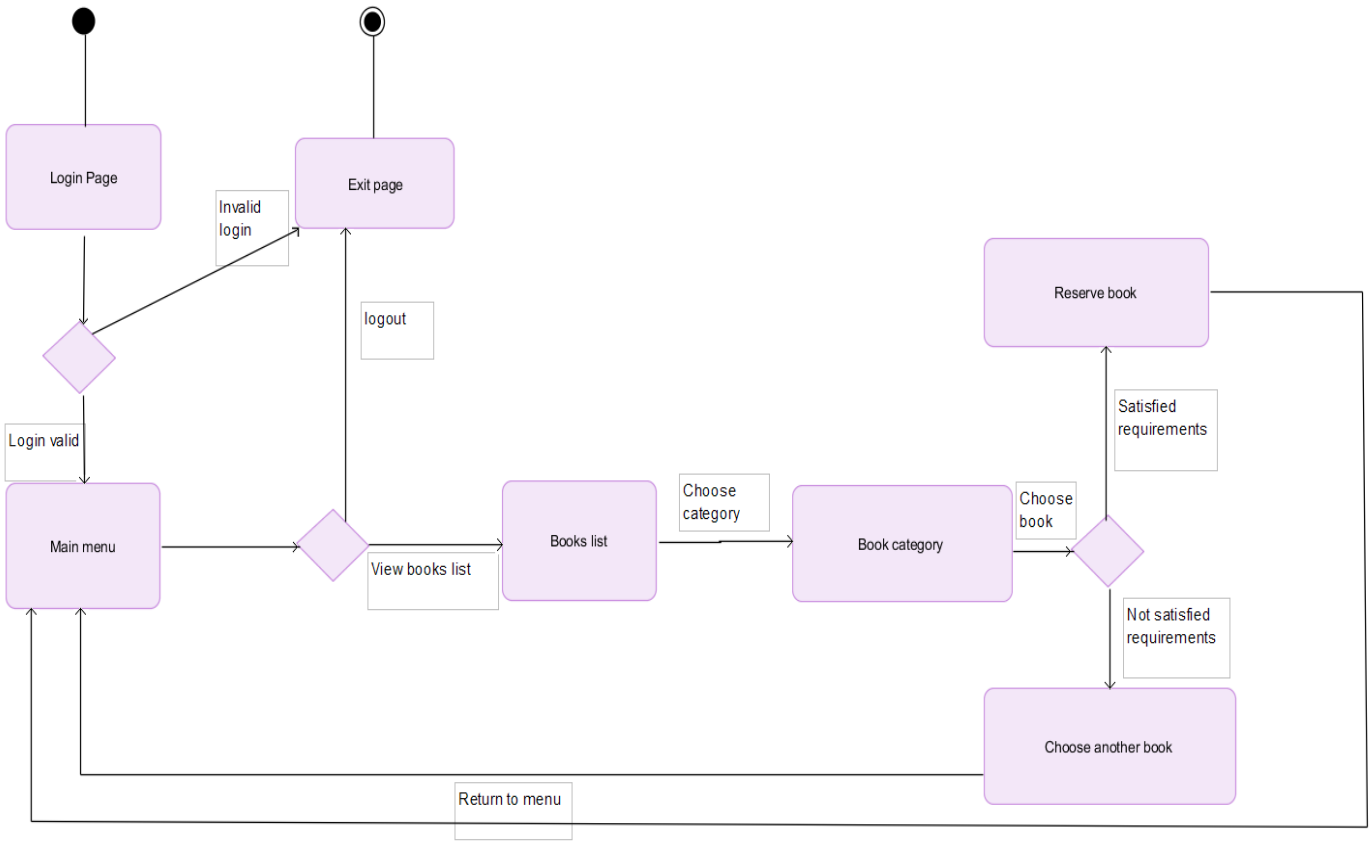
Di seguito viene riportato lo state diagram relativo alla prenotazione\prestito di libri tramite AppLibrarian:



Nel diagramma si possono riconoscere i vari stati dell'applicazione:

- Login page: pagina iniziale del programma dalla quale è possibile effettuare il login;
- Exit page: pagina di uscita dal programma in caso di login fallito o log out;
- Main menu: menu principale dell'applicazione dal quale è possibile selezionare diverse opzioni, tra cui "log out" che porta alla pagina di uscita e "lista dei libri" che visualizza l'interfaccia associata;
- Books list: interfaccia con l'elenco dei libri;
- Books category: selezione del filtro tramite categoria, esistono diverse possibilità per filtrare i libri a seconda del contenuto, hanno tutte le stesse modalità e nello schema è stata riportata solo questa poiché le altre usano la medesima metodologia;
- Loan book: stato in cui si arriva dopo aver scelto un libro per il quale l'utente rispetta tutti i vincoli. Nello specifico i vincoli necessari per la concessione di un prestito sono: non avere cinque attualmente in prestito, non avere prestiti sconfinanti, non avere già quel libro in prestito al momento della consegna e essere i primi nella coda della prenotazione del libro. Dopo la consegna del prestito il programma ritorna nello stato di main menu per permettere all'utente di proseguire con altre operazioni;
- Reserve book: stato raggiunto se il libro richiesto non è disponibile e l'utente soddisfa tutti i vincoli necessari per procedere con la registrazione di una prenotazione. In questo caso i vincoli da rispettare sono: non avere dieci prenotazioni e non essere già presenti nella coda delle prenotazioni di quel determinato libro. Dopo la registrazione della prenotazione il programma ritorna nello stato di main menu per permettere all'utente di proseguire con altre operazioni;
- Choose another book: stato che viene raggiunto nel caso in cui il libro scelto non è disponibile e l'utente non soddisfa uno o più requisiti necessari per la registrazione del prestito. In questo scenario l'utente viene invitato a scegliere un altro testo e il programma torna nello stato di main menu.

Successivamente è stato realizzato lo state diagram relativo alla prenotazione di un libro tramite AppReader:



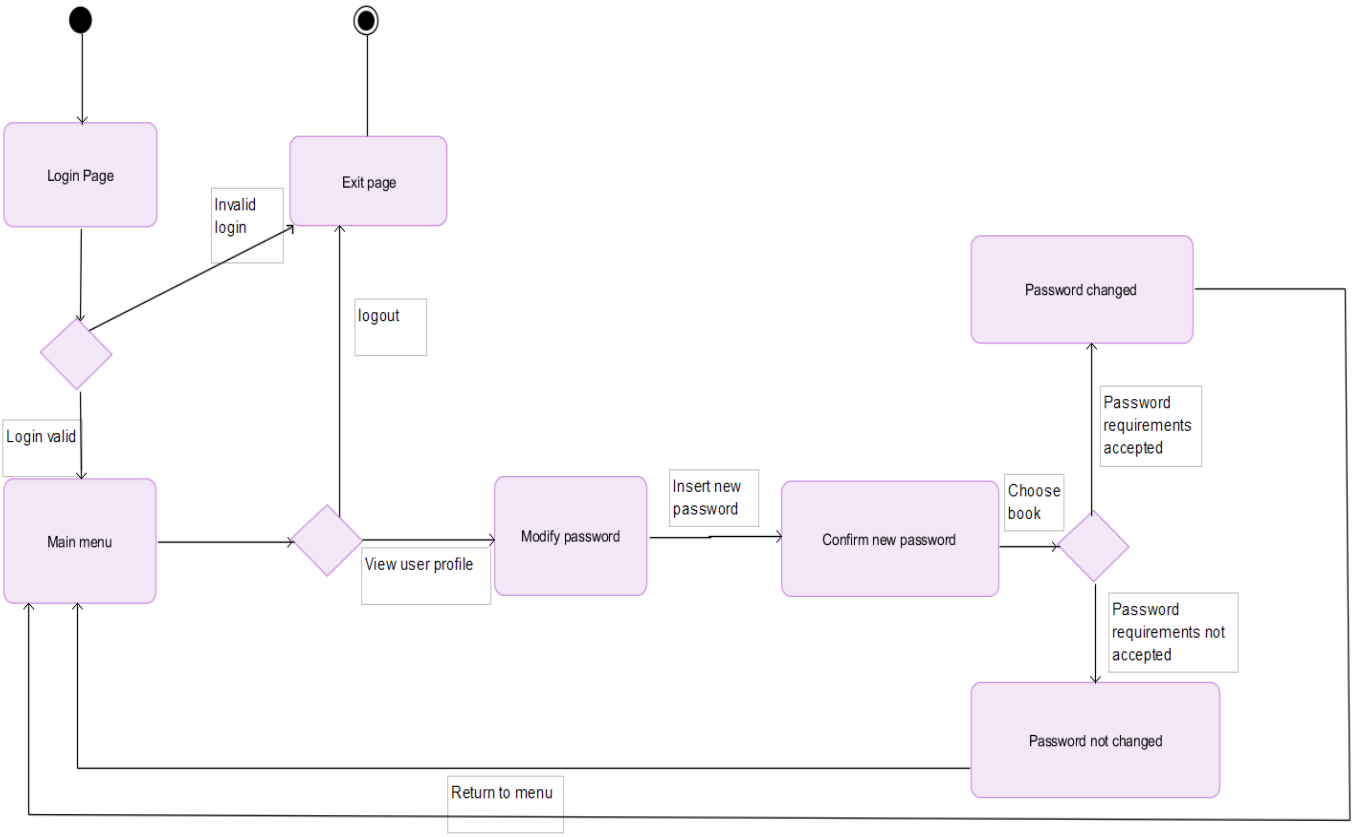
Gli stati che l'applicazione può assumere in questa operazione sono:

- Login page: pagina iniziale del programma dalla quale è possibile effettuare il login;
- Exit page: pagina di uscita dal programma in caso di login fallito o log out;
- Main menu: menu principale dell'applicazione dal quale è possibile selezionare diverse opzioni, tra cui "log out" che porta alla pagina di uscita e "lista dei libri" che visualizza l'interfaccia associata;
- Books list: interfaccia con l'elenco dei libri;
- Books category: selezione del filtro tramite categoria, esistono diverse possibilità per filtrare i libri a seconda del contenuto, hanno tutte le stesse modalità e nello schema è stata riportata solo questa poiché le altre usano la medesima metodologia;
- Reserve book: stato raggiunto se l'utente soddisfa tutti i vincoli necessari per procedere con la registrazione di una prenotazione. In questo caso i vincoli da rispettare sono: non avere dieci prenotazioni e non essere già presenti nella coda delle prenotazioni di quel

determinato libro. Dopo la registrazione della prenotazione il programma ritorna nello stato di main menu per permettere all'utente di proseguire con altre operazioni;

- Choose another book: stato che viene raggiunto nel caso in cui il libro scelto non è disponibile e l'utente non soddisfa uno o più requisiti necessari per la registrazione del prestito. In questo scenario l'utente viene invitato a scegliere un altro testo e il programma torna nello stato di main menu.

Terminata la stesura di questi due state diagram, il passo successivo è stato quello di definire il diagramma degli stati dell'operazione di modifica del profilo, di seguito viene riportato quello relativo alla modifica della password:



In questo scenario gli stati possibili sono:

- Login page: pagina iniziale del programma dalla quale è possibile effettuare il login;
- Exit page: pagina di uscita dal programma in caso di login fallito o log out;

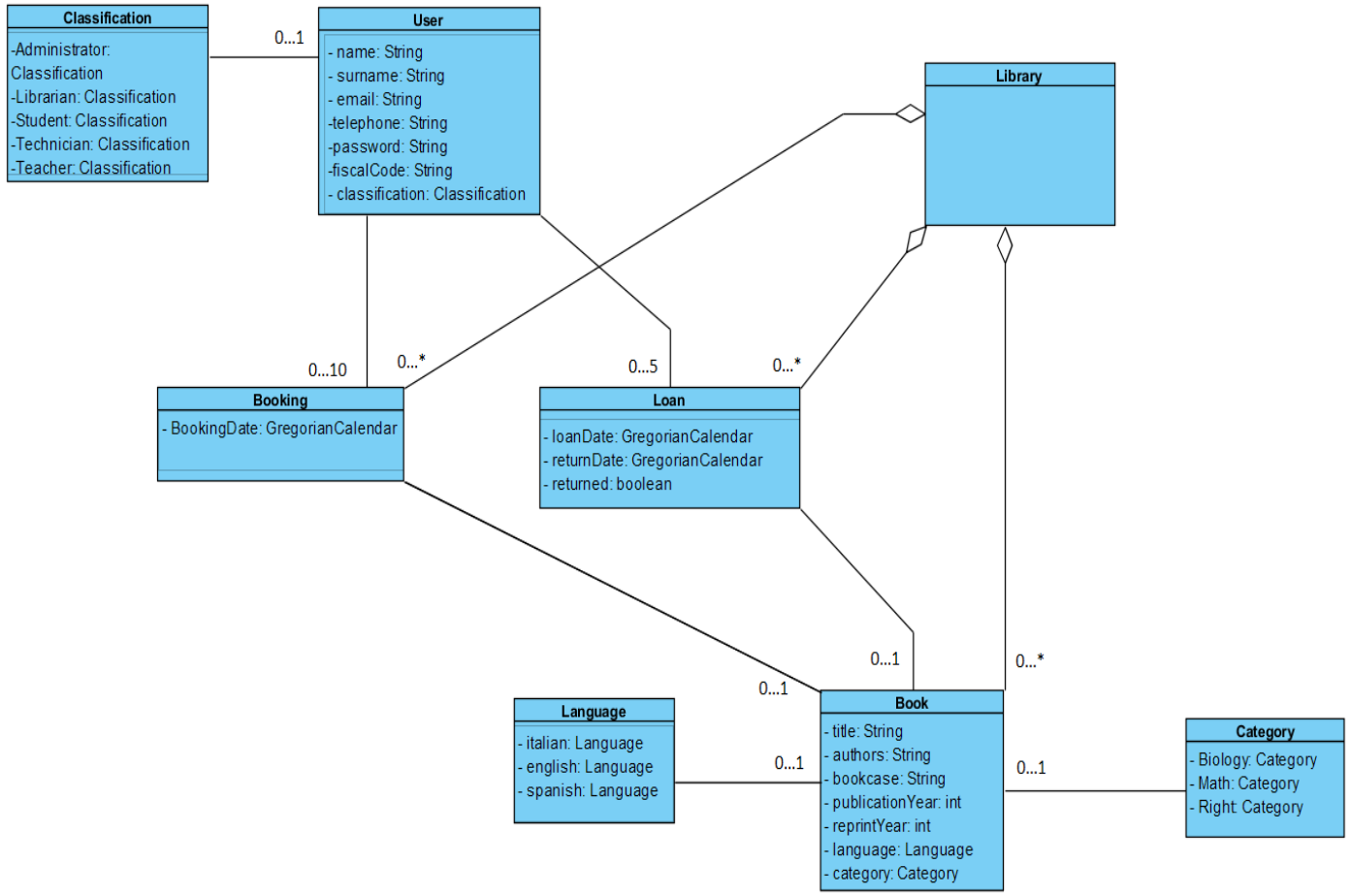
- Main menu: menu principale dell'applicazione dal quale è possibile selezionare diverse opzioni, tra cui "log out" che porta alla pagina di uscita e "impostazioni profilo" che visualizza l'interfaccia associata;
- Modify password: interfaccia per modificare la password associata all'account;
- Confirm new password: conferma della nuova password inserita;
- Password changed: caso in cui la nuova password inserita rispetta i requisiti richiesti e viene modificata;
- Password not changed: caso in cui la nuova password inserita non rispetta i requisiti richiesti e non viene accettata.

La scelta di realizzare solo il diagramma relativo alla modifica della password deriva dal fatto che le altre operazioni di modifica del profilo (modifica di email, inquadramento e numero di telefono) sono gestite nello stesso modo e quindi gli schemi sarebbero risultati ripetitivi.

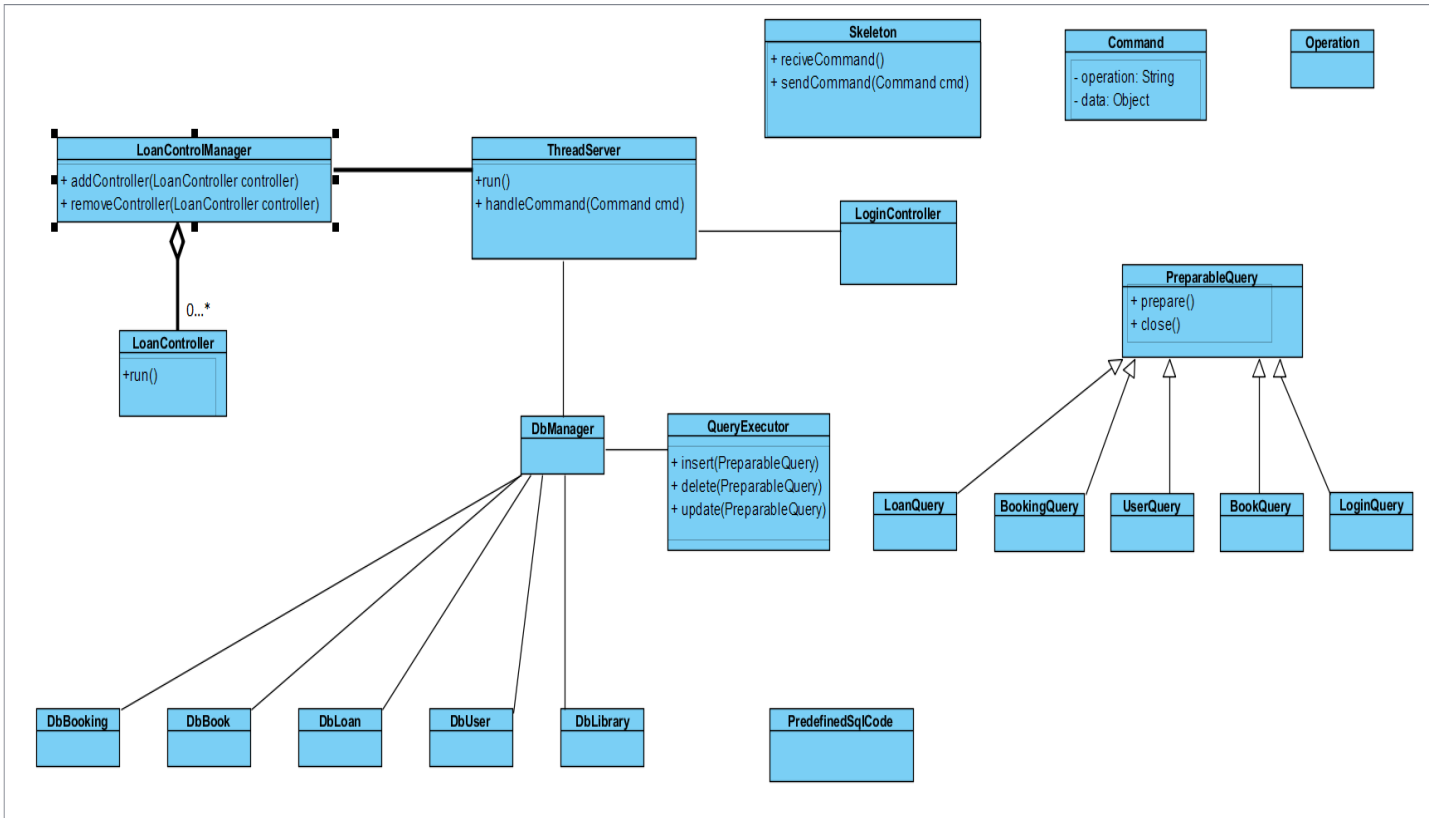
Class diagram

L'ultima fase è stata la creazione dei class diagram, che in termini generali, consentono di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi tipi.

Di seguito viene riportato il diagramma relativo al modello dei dati utilizzato:



Abbiamo inoltre realizzato il class diagram relativo a quella che poi sarebbe diventata, con qualche modifica, l'architettura del componente server:



Insieme alla relazione è presente una cartella contente il class diagram della versione finale dell'intero sistema implementato.

3. Definizione dello spazio della soluzione

Conclusa la fase di definizione dello spazio del problema ci siamo concentrati sulla definizione dello spazio della soluzione, cioè su come realizzare un'applicazione che risolva il problema descritto in precedenza, focalizzando la nostra attenzione sulle scelte di progettazione, la ricerca e l'utilizzo di design patterns utili per risolvere i problemi di progettazione incontrati e sulla creazione di una base di dati che soddisfi i requisiti richiesti.

Scelte di progettazione

In questa sezione vengono illustrate e descritte le scelte di progettazione effettuate, ponendo particolare attenzione sulla scelta del protocollo di comunicazione, le strutture dati utilizzate e l'utilizzo della libraria Swing.

Protocollo di comunicazione

Per raggiungere l'obiettivo della progettazione di un'architettura client – server abbiamo dovuto implementare un opportuno protocollo che permettesse alle diverse applicazioni di comunicare tra di loro in maniera efficiente. Dopo una analisi delle varie possibilità offerte dal linguaggio Java, la nostra scelta è ricaduta sull'utilizzo delle classi Socket – ServerSocket che forniscono metodi per creare connessioni tra client e server utilizzando il protocollo di rete TCP/IP.

Altre possibilità di implementazione di sistema, quali ad esempio il protocollo di comunicazione UPD o la tecnologia RMI, non sono state scelte perché, a nostro parere, non idonee alla soluzione del problema proposto.

Stabilito il protocollo di rete, ci siamo concentrati sulla scrittura in Java di un protocollo che permettesse lo scambio di informazioni tra client e server. Per implementare in modo efficiente il protocollo ci siamo basati sull'utilizzo del design pattern Command, descritto approfonditamente nella sezione relativa ai design pattern utilizzati. L'incapsulamento dei dati e delle operazioni in un solo oggetto ci ha permesso di inviare le richieste e i dati elaborati con un'unica istruzione rendendo il codice più leggibile ed efficiente. Tutte le classi che definiscono oggetti inviati attraverso i socket implementano l'interfaccia Serializable che ne permette la serializzazione.

Strutture dati utilizzate

Per la memorizzazione temporanea dei dati abbiamo deciso di utilizzare due strutture dati in particolare: ArrayList e HashMap.

ArrayList è una struttura dati dinamica che implementa l'interfaccia List in un array ridimensionabile. Ogni volta che viene aggiunto un elemento, se l'array è pieno, esso viene automaticamente riallocato con una dimensione maggiore del 50%, i dati vengono copiati nel nuovo array e il nuovo elemento viene aggiunto nella prima posizione disponibile. Questa operazione avviene in modo trasparente per l'utente.
Dato che implementa l'interfaccia List è possibile utilizzare i metodi forniti da questa interfaccia per manipolare i dati nell'array.

Il vantaggio principale di questa struttura dati è che ogni operazione di accesso posizionale richiede tempo costante O(1), mentre l'operazione di rimozione/aggiunta di un elemento ha complessità O(1) nel caso migliore (rimozione/aggiunta nell'ultima posizione) e O(n) nel caso peggiore (rimozione/aggiunta nella prima posizione).

La scelta per la memorizzazione di libri, prestiti e prenotazioni è ricaduta su questa struttura dati a causa dei numerosi accessi posizionali che vengono effettuati negli aggiornamenti delle interfacce grafiche quando i dati sono visualizzati all'interno delle tabelle.

HashMap è una struttura dati utilizzata per mettere in corrispondenza una data chiave con un dato valore (che può anche essere null); nell'implementazione che abbiamo usato ogni valore della tabella (un qualsiasi Object) è indicato da una combinazione di due chiavi. L'hash map per le sue proprietà è molto utilizzata nei metodi di ricerca. Difatti l'hashing è completamente diverso da una struttura basata su confronti: invece di muoversi nella struttura data in funzione dell'esito dei confronti tra chiavi, si accede direttamente ai dati ricercati all'interno della tabella e difatti il costo medio di ricerca di ogni elemento è indipendente dal numero di elementi.

Questa struttura è stata utilizzata per memorizzare gli utenti loggati nel sistema in modo da evitare login multipli da parte dello stesso account così da ridurre il problema della concorrenza che si può verificare in un server che è in grado di gestire più connessioni in parallelo.
Nel nostro caso la chiave generata relativa alle righe della tabella è rappresentata dalla stringa contente il codice fiscale dell'utente loggato e quella relativa alle colonne è rappresentata dall'oggetto Skeleton che gestisce la comunicazione e che viene utilizzato per informare il client di effettuare il log out forzato dell'utente in caso di un nuovo login con lo stesso account.

Graphical User Interface(GUI)

Per migliorare l'esperienza dell'utente abbiamo deciso di creare un'interfaccia grafica 2d sfruttando le librerie awt e swing di Java.

Nella libreria Swing la realizzazione delle interfacce viene svolta dall'Event Dispatch Thread (EDT), thread che esegue il drawing dei componenti grafici e il codice per la gestione degli eventi generati dall'interazione tra i componenti grafici e l'utente.

Nello sviuppo di interfacce grafiche basate su questa libreria bisogna rispettare la seguente regola: ogni operazione che consiste nella visualizzazione, modifica o aggiornamento di un componente Swing, o che accede allo stato del componente stesso, deve essere eseguita nell'Event Dispatch Thread.
Questo perché gran parte dei metodi che operano su oggetti della libreria Swing non sono threadsafe: se invocati da più thread concorrenti possono portare a deadlock o ad errori di consistenza della memoria, non rispettare la regola può quindi portare ad errori imprevedibili e non sempre riproducibili.

Se l'EDT è impegnato in attività lunghe e complesse o nell'esecuzione di codice con istruzioni bloccanti, l'applicazione appare "congelata" (freezed) e sembra non rispondere ai comandi dell'utente.

La classe SwingWorker consente di gestire ed eseguire in background attività lunghe e complesse o bloccanti che necessitano di interagire con la GUI Swing al termine della computazione e/o durante il processamento.

Nei clients abbiamo deciso di implementare questa classe per gestire la ricezione dei dati da visualizzare, una attività potenzialmente lunga e complessa. Definendo una sottoclasse di SwingWorker, chiamata TableWorker, che si occupa della gestione della comunicazione con il server e della visualizzazione nelle tabelle dei dati ricevuti, siamo stati quindi in grado di evitare "congelamenti" dell'interfaccia che rimane sempre reattiva agli input dell'utente.

Gestione delle concorrenza

Per limitare le problematiche dovute alla concorrenza abbiamo deciso di implementare un componente dal lato server che impedisce login multipli tramite uno stesso account.

In questo modo siamo stati in grado di impedire agli utenti di effettuare modifiche concorrenti di un account, intese sia come modifiche ai dati personali che alla gestione dei libri.

Nell'implementazione di questo componente abbiamo deciso di utilizzare una HashMap in cui tenere traccia dei codici fiscali degli utenti, attualmente loggati nel Sistema, e degli oggetti Skeleton che si occupano della comunicazione con il client relativo all'account loggato.
Al momento del login di un utente la coppia <userId, skeleton> viene memorizzata nella struttura dati, e nel caso la chiave relativa all'userId sia già presente, il riferimento precedente ad essa viene rimosso dalla mappa e viene sfruttato lo skeleton relativo a quella connessione per informare il client di forzare il logout dell'utente.

Tramite questo semplice ma efficace meccanismo si ha la certezza che in ogni istante ci sia al più un solo account loggato relativo a ogni utente.

L'inserimento dei dati nella hashmap viene effettuato da un metodo con il costrutto synchronized per evitare il verificarsi di situazioni di concorrenza che potrebbero portare a uno stato inconsistente dei dati nella struttura utilizzata.

Design patterns

In questa fase abbiamo cercato tra i vari design patterns esistenti quelli che erano più utili per risolvere il nostro problema, e abbiamo individuato questi cinque pattern:

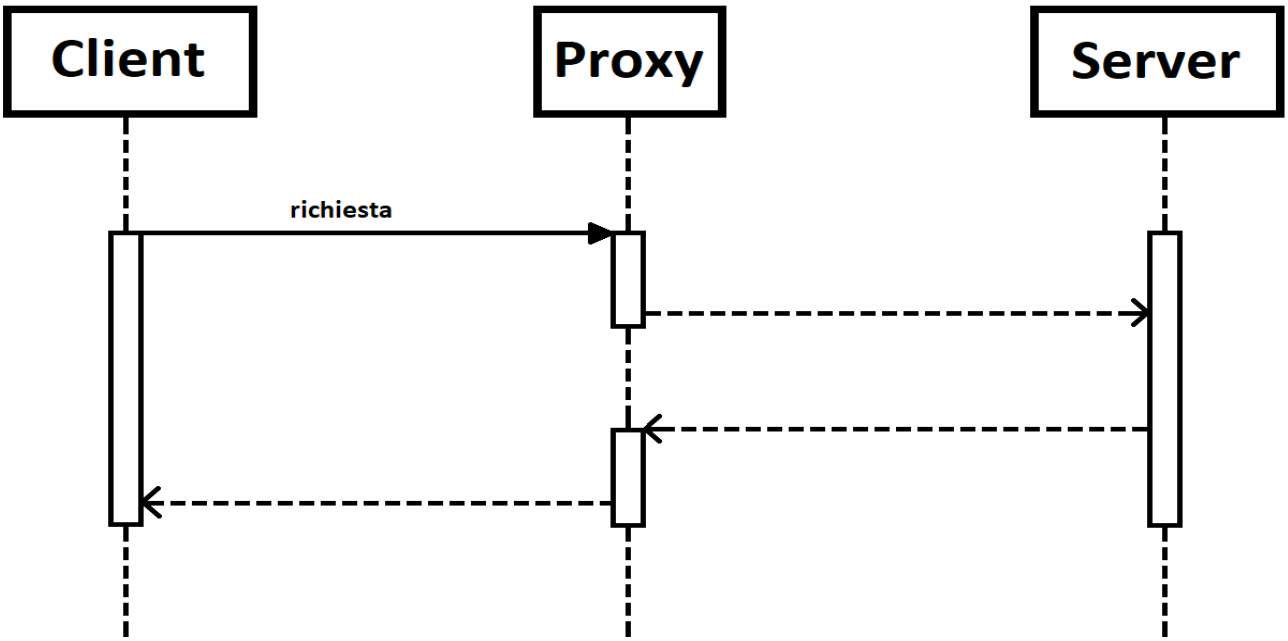
Pattern proxy:

pattern che, nella sua forma più generale, indica una classe che funziona come interfaccia per qualcos'altro, ad esempio una connessione di rete, un grosso oggetto in memoria, un file e altre risorse che sono costose o impossibili da duplicare. Nel nostro caso il design pattern proxy è stato implementato nell'applicazione dal lato client con lo scopo di sollevare il client dalla gestione della comunicazione con il server.

L'obiettivo principale è quello di evitare che nel client il codice applicativo sia mischiato con quello per la comunicazione, in questo modo infatti i meccanismi necessari per l'instaurazione

della comunicazione, e lo scambio dei dati, sono implementati ed incapsulati solo ed esclusivamente all'interno del proxy.

Di seguito è riportato il sequence diagram di questo pattern con lo scopo di chiarire ulteriormente il suo ruolo chiave all'interno dell'applicazione, fornendo una visione alternativa.



Come si può notare dal diagramma la classe Proxy viene idealmente posizionata tra il client e il server, in questo modo il client si limita a invocare i metodi della classe e attende una risposta senza preoccuparsi dell'implementazione del protocollo di comunicazione e senza conoscere nulla sulla posizione e l'implementazione del componente che processa le richieste.

Pattern skeleton:

Equivalente del design pattern proxy con la differenza che viene implementato dal lato del server. L'obiettivo resta sempre quello di separare in modo netto la gestione della comunicazione dalla logica applicativa che costituisce il server.

Pattern command:

questo pattern, noto anche come action o transaction, permette di inoltrare richieste ad oggetti senza conoscere nulla dell'operazione da eseguire o del destinatario della richiesta. Questo è possibile per il fatto che il pattern in questione tratta la richiesta come un oggetto differente rispetto sia al richiedente che all'oggetto destinatario. Questo oggetto specifica l'azione da svolgere sul destinatario, sfruttandone i comportamenti in modo tale da poter portare a termine la richiesta.

Il pattern command permette quindi di incapsulare una richiesta in un oggetto permettendo al client di inoltrare richieste di varia natura, il vantaggio più significativo nell'applicazione di questo pattern è il fatto di avere un disaccoppiamento tra l'oggetto che invoca il comando e il destinatario, cioè colui che conosce il modo per portare a termine l'operazione.

Nel nostro progetto il design pattern command è stato sfruttato nella costruzione di un protocollo di comunicazione client-server.

Un oggetto Command è costituito da due campi:

- operation: campo di tipo String che contiene l'operazione da eseguire scelta tra una delle costanti della classe Operation;
- data: campo di tipo Object che contiene le informazioni necessarie per eseguire la richiesta.

Sia client che server utilizzano oggetti di questo tipo per inviare richieste e ricevere i dati elaborati. Generalmente il client creerà oggetti Command inserendo nel campo operation la richiesta da eseguire e nel campo data i dati necessari per l'esecuzione, mentre il server riporterà nel campo operation l'esito dell'operazione e nel campo data gli eventuali dati elaborati se presenti.

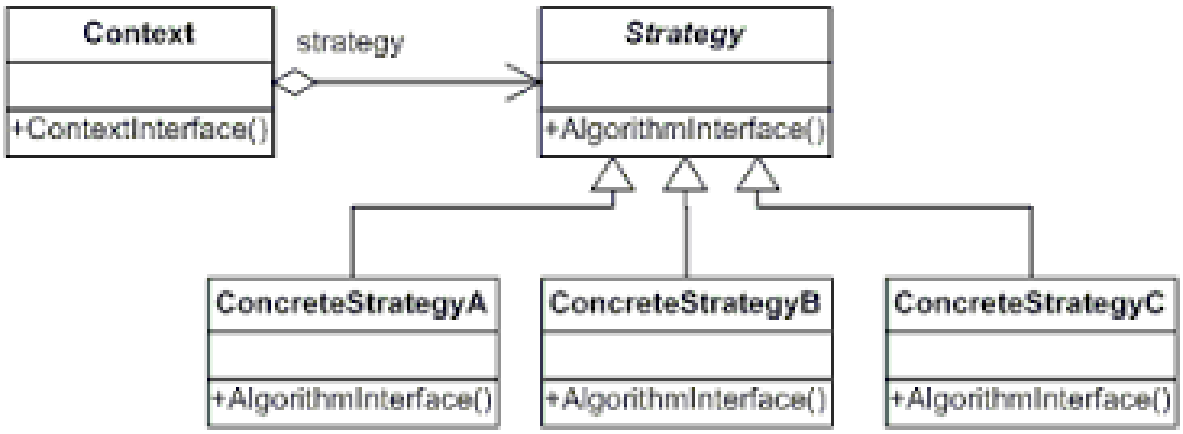
Pattern strategy:

Il design pattern strategy è un pattern comportamentale che permette di definire una famiglia di algoritmi, di incapsularli e renderli intercambiabili tra di loro in base ad una specifica condizione, in modalità trasparente al client che ne fa uso.

Questo pattern consente agli algoritmi di variare in modo indipendente rispetto al loro contesto di utilizzo, fornendo un basso accoppiamento tra le classi partecipanti del pattern e una alta coesione funzionale delle diverse strategie di implementazione.

L'obiettivo di questa architettura è isolare un algoritmo all'interno di un oggetto, in maniera tale da risultare utile in quelle situazioni dove sia necessario modificare dinamicamente gli algoritmi utilizzati da un'applicazione.

Il class diagram generico del pattern strategy è il seguente:



Nel progetto questo pattern è stato utilizzato nelle seguenti classi:

- QueryExecutor: assume il ruolo equivalente a quello della classe Context nello schema riportato sopra. Si occupa di preparare, eseguire e chiudere le query contenute negli oggetti che implementano l'interfaccia PreparabeQuery;
- PreparableQuery: assume il ruolo equivalente a quello dell'interfaccia Strategy nello schema riportato sopra. Definisce al suo interno i metodi prepareStatement e close per preparare e chiudere uno Statement;
- BookQuery, BookingQuery, Libraryuery, LoanQuery, LoginQuery e UserQuery: assumono il ruolo equivalente a quello delle classi ConcreteStrategyA, ConcreteStrategyB e ConcreteStrategyC nello schema riportato sopra. Implementano l'interfaccia PreparableQuery e ridefiniscono i metodi per preparare e chiudere uno statement.

Pattern singleton:

Il singleton è un design pattern creazionale usato per assicurare che una classe abbia una sola istanza ed un unico punto di accesso globale. L'implementazione di questo pattern consiste nell'affidare alla classe stessa la responsabilità di creare le proprie istanze. In questo modo è quest'ultima che assicura che nessun'altra istanza possa essere creata, gestendo in modo centralizzato le richieste di creazione di nuove istanze. Questo pattern si può rivelare utile nel caso in cui si abbia la necessità di centralizzare informazioni e comportamenti in un'unica entità condivisa da tutti i suoi utilizzatori.

Per implementare il pattern singleton è necessario che la classe venga progettata con i costruttori privati per evitare la possibilità di istanziare un numero arbitrario di oggetti della stessa. La classe fornisce inoltre un metodo statico (instance())che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico.

Nel nostro progetto questo design pattern è stato implementato nelle seguenti classi: Proxy, Library, LoanControllerManager, DbManager.

Nell'implementazione abbiamo utilizzato un approccio basato sul principio della lazy initialization ("inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa necessario, cioè al primo tentativo di uso.

Base di dati

Analisi dei requisiti

Inizialmente abbiamo fatto una'nalisi approfondita della traccia che ci ha portato ad avere una visione globale delle entità, con i relativi attributi, che compongono il sistema.

La richiesta prevede l'implementazione di un database relazionale scritto in linguaggio SQL in grado di:

- Gestire le informazioni relative agli utenti registrati;
- Gestire il catalogo dei libri della libreria;
- Memorizzare le prenotazioni degli utenti;
- Tenere traccia dei libri prestati dalla libreria.

Entrando più nel dettaglio le informazioni delle quali è richiesta la memorizzazione sono:

- nome, cognome, codice fiscale, email, inquadramento (specificante la funzione dell'utente), numero di telefono, password, codice di attivazione. L'inquadramento deve essere specificato tra: bibliotecario, studente-classe, docente, tecnico, amministrativo.
- codice identificativo ISBN, titolo, autori, casa editrice, anno di pubblicazione, anno di ristampa, categoria del libro, lingua in cui è scritto, scaffale della biblioteca in cui è riposto. Anche la categoria così come l'inquadramento si riferisce a elementi estratti da una lista predefinita;
- codice identificativo dell'utente che ha effettuato la prenotazione, il codice del libro che si intende prenotare, data e ora della prenotazione.
- codice del libro, il codice dell'utente a cui viene erogato il prestito, la data di rilascio del prestito, la data di consegna.

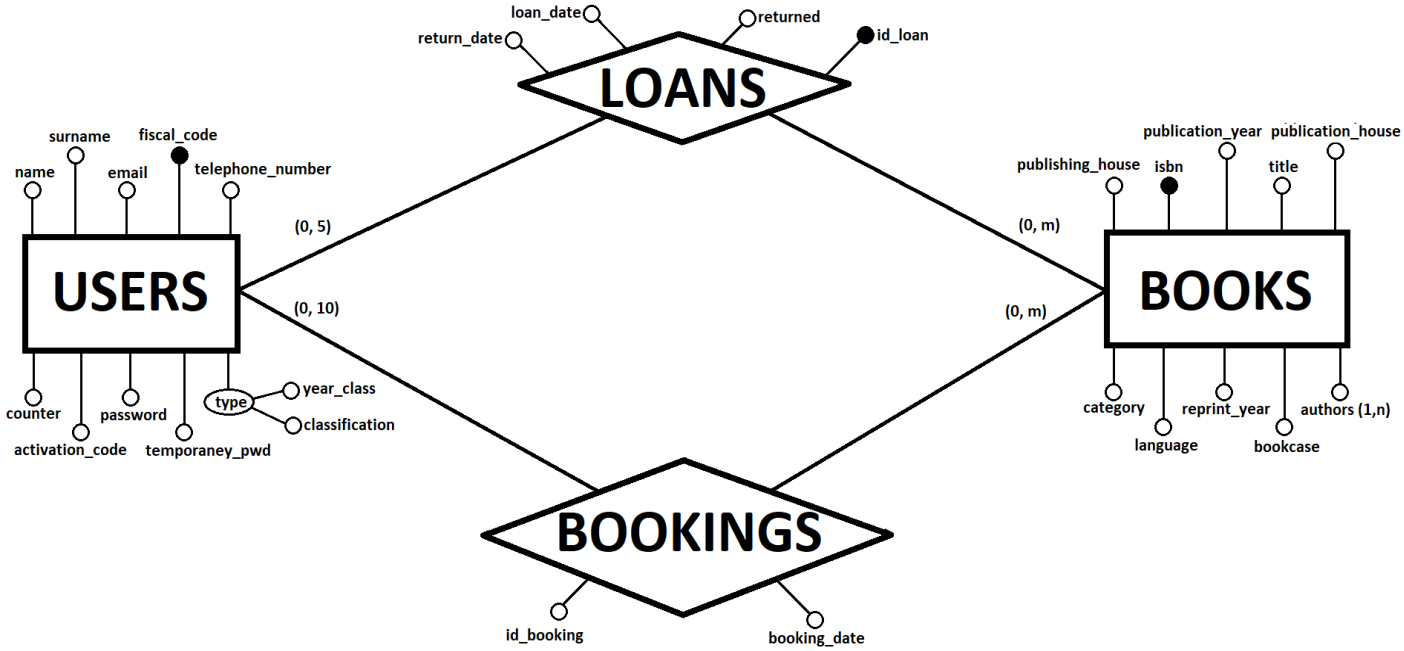
In particolare è richiesto il rispetto dei seguenti vincoli:

- Un utente può avere prenotazioni attive per massimo 10 libri;
- Ogni utente non può ricevere più di 5 libri in prestito;
- Ogni libro prestato deve essere riconsegnato alla biblioteca entro 30 giorni.

Le entità che abbiamo ricavato da questa fase sono: users, books, bookings e loans.

Schema ER

Dalla prima fase di analisi è quindi stato prodotto il seguente schema concettuale:



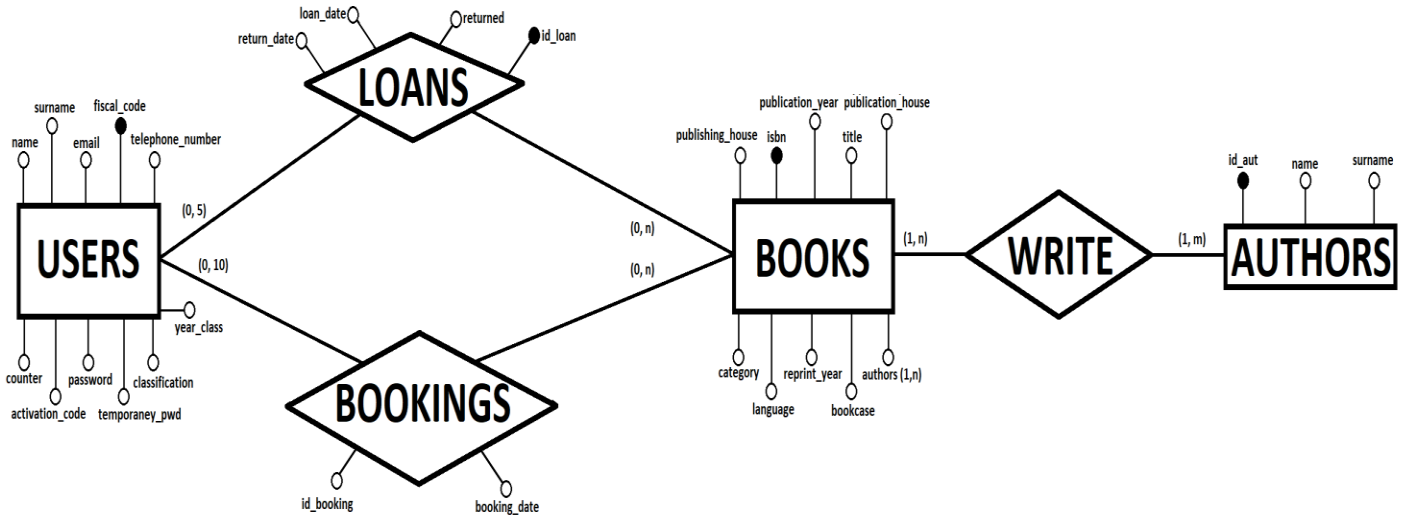
Come si può notare lo schema è formato dalle due entità users e books che contengono gli attributi relativi agli utenti e ai libri e le due associazioni bookings e loans relative rispettivamente alle prenotazioni e ai prestiti.

In questa prima fase abbiamo preso due scelte importanti:

- rappresentare la tipologia degli utenti come attributo composto formato dall'inquadramento dell'utente e dall'attributo anno e classe disponibile per gli studenti;
- inserire nell'entità books l'attributo multi-valore authors con cardinalità 1-n perché ogni libro viene scritto da almeno un autore.

Schema ER ristrutturato

Siamo poi passati alla successiva fase di ristrutturazione dello schema er che ha portato al seguente schema ristrutturato dove sono stati eliminati l'attributo composto type e l'attributo multi-valore authors.



Le scelte effettuate in questa fase di ristrutturazione sono le seguenti:

- abbiamo eliminato l'attributo composto type trasformando i suoi sotto-attributi classification e year_class in attributi dell'entità users. Adottando questa soluzione si perde la relazione che si crea tra i sotto-attributi, ma allo stesso tempo ciò permette di accedere separatamente a ciascun attributo, fatto molto importante che ottimizza il nostro programma dove gli inquadramenti sono gestiti come enumerativi e l'attributo year_class è necessario solo per l'inquadramento di tipo studente mentre è superfluo per tutti gli altri.
- Abbiamo eliminato l'attributo multi-valore authors definendo una nuova entità (authors) collegata al'entità books attraverso una nuova relazione (write) che ci permette di tenere traccia degli autori che hanno scritto i vari libri. Nell'entità authors sono stati aggiunti tre attributi: name e surname che contengono il nome e il cognome dell'autore e id_aut che contiene un codice identificativo dell'autore.

Schema logico

Lo schema ER ristruttrato è stato poi tradotto nel seguente schema logico utilizzando la metodologia standard:

Users

Questa entità contiene le informazioni relative agli utenti.

I suoi attributi sono:

-name (nome dell'utente)

-surname (cognome dell'utente)

-fiscal_code (codice fiscale dell'utente)

-telephone number (numero di telefono dell'utente)

-email (indirizzo di posta elettronica dell'utente)

-classification (classificazione dell'utente)

-year_class (anno e classe dello studente)

-activation_code (codice di attivazione dell'utente)

-password (password dell'utente)

-temporaney_pwd (valore booleano che indica se la password è temporanea)

-counter (numero di tentativi rimasti per l'attivazione del profilo)

La chiave primaria di users è fiscal_code, in quanto attraverso questo attributo possiamo identificare in modo univoco un utente.

Books

Questa entità contiene le informazioni relative ai libri.

I suoi attributi sono:

-isbn (codice ISBN del libro)

-title (titolo del libro)

-publication_year (anno di pubblicazione del libro)

-reprint_year (anno di ristampa del libro)

-bookcase (scaffale in cui è riposto libro)

-language (lingua del libro)

-category (categoria del libro)

-publishing_house (casa editrice del libro)

La chiave primaria di books è isbn, in quanto attraverso questo attributo possiamo identificare in modo univoco un libro.

Authors

Questa entità contiene le informazioni relative agli autori dei libri.

I suoi attributi sono:

-name (nome dell'autore)

-surname (cognome dell'autore)

-id_aut (id dell'autore)

La chiave primaria di authors è id_aut, in quanto attraverso questo attributo possiamo identificare in modo univoco un autore.

Bookings

Questa entità contiene le informazioni relative alle prenotazioni.

I suoi attributi sono:

-id_user (codice fiscale dell'utente che ha effettuato la prenotazione)

-id_book (codice ISBN del libro prenotato)

-booking_date (data di prenotazione del libro)

-id_booking (codice identificativo della prenotazione)

La chiave primaria di bookings è formata dagli attributi id_user e id_book, perché nei requisiti del programma viene specificato che un utente può avere al più una prenotazione riferita a un determinato libro.

Le chiavi esterne di bookings sono id_user (si riferisce all'utente che effettua la prenotazione contenuto nella tabella users) e id_book (si riferisce al libro prenotato contenuto nella tabella books).

Loans

Questa entità contiene le informazioni relative ai prestiti

I suoi attributi sono:

-id_book (codice ISBN del libro prestato)

-id_user (codice fiscale dell'utente a cui è stato prestato il libro)

-loan_date (data di inizio del prestito)

-return_data (data termine del prestito)

-returned (valore booleano che indica se il libro è stato restituito)

-id_loan (codice identificativo del prestito)

La chiave primaria di loans è id_loan, un numero progressivo che identifica in modo univoco un prestito.

Le chiavi esterne di loans sono id_user (si riferisce all'utente che effettua il prestito contenuto nella tabella users) e id_book (si riferisce al libro preso in prestito contenuto nella tabella books).

Write

Questa entità contiene le informazioni relative a quali libri sono stati scritti da un autore.

I suoi attributi sono:

-id_book (codice ISBN del libro)

-id_aut (codice identificativo dell'autore che ha scritto il libro)

Le chiavi esterne di WRITE sono id_aut (si riferisce all'autore che ha scritto il libro contenuto nella tabella authors) e id_book (si riferisce al libro contenuto nella tabella books).

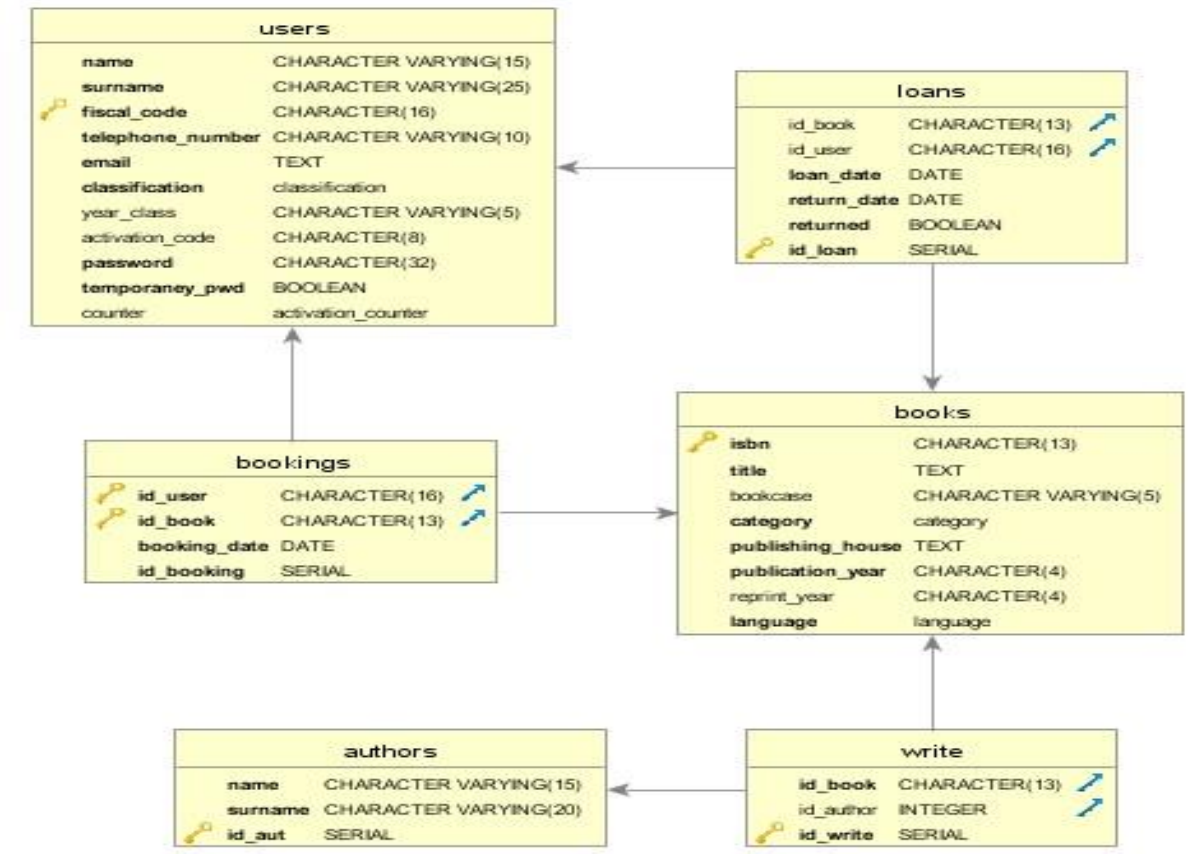
Implementazione con PostreSQL

Terminata la costruzione dello schema logico abbiamo fatto una ricerca approfondita delle funzionalità offerte dal programma PostrgeSQL, e ciò ci ha portato alla decisione di sfruttare la possibilità di creare dei tipi di dato personalizzati.

I tipi di dato personalizzati che abbiamo implementato sono i seguenti:

- activation_counter: dato il cui tipo base è integer e che contiene un vincolo check che verifica che il valore del contatore sia compreso tra zero e cinque estremi inclusi;
- category: dato il cui tipo base è text e che contiene un vincolo check che verifica tramite l'operatore SQL "IN" che il valore della categoria sia uno tra quelli disponibili;
- classification: dato il cui tipo base è text e che contiene un vincolo check che verifica tramite l'operatore SQL "IN" che il valore dell'inquadramento sia uno tra quelli disponibili;
- language: dato il cui tipo base è text e che contiene un vincolo check che verifica tramite l'operatore SQL "IN" che il valore della lingua sia uno tra quelli disponibili;

Abbiamo successivamente implementato lo schema utilizzando il programma PostgreSQL, ottenendo il seguente risultato:



Come si può notare dallo schema per i dati contenenti stringhe sono stati utilizzati i tipi character varying, che permette di inserire una stringa avente un numero di caratteri compreso tra zero e il numero specificato tra parentesi, character, che permette di inserire una stringa avente un numero di caratteri pari al numero specificato tra parentesi e text che permette di inserire una stringa di lunghezza indefinita. Per i dati di tipo booleano e per le date sono stati utilizzati i corrispondenti tipi boolean e date.

Per gli attributi id_aut, id_write, id_booking e id_loan abbiamo usato il tipo serial che consente di creare indici auto-incrementanti, particolarmente utile per la definizione di codici univoci da utilizzare come chiavi primarie o per caratterizzare le tuple di una relazione.
La caratteristica più importante di questo tipo di dato è il fatto che il DBMS gestisce in automatico l'incremento agevolando il lavoro del programmatore liberandolo da questo compito.

Per l'attributo password invece abbiamo usato character(32) perché, come illustrato nella parte relativa alle funzionalità aggiuntive, abbiamo deciso di criptare le password degli utenti grazie a una funzione che restituisce come output una stringa di 32 caratteri, e l'utilizzo del tipo character dava la garanzia che le password salvate nel database fossero criptate e non in chiaro.

Descrizione chiavi e vincoli

Le chiavi esterne implementate sono le seguenti:

- book_isbn_fk: mette in relazione l'attributo id_book della tabella bookings con l'attributo isbn di books, per creare un collegamento tra la tabella delle prenotazioni e quella dei libri in modo da tenere traccia dei libri prenotati. In caso di cancellazione di una tupla dalla tabella books è stata utilizzato il comando "cascade" che si occupa della cancellazione di tutte le tuple riferite nella tabella bookings con quella chiave;

- user_fk: mette in relazione l'attributo id_user della tabella bookings con l'attributo fiscal_code di users, per creare un collegamento tra la tabella delle prenotazioni e quella degli utenti in modo da tenere traccia dell'utente che ha effettuato una prenotazione. In caso di cancellazione di una tupla dalla tabella users è stata utilizzato il comando "cascade" che si occupa della cancellazione di tutte le tuple riferite nella tabella bookings con quella chiave;

- id_books_fk: mette in relazione l'attributo id_book della tabella loans con l'attributo isbn di books, con lo scopo di creare un collegamento tra la tabella dei prestiti e quella

dei libri in modo da tenere traccia dei libri prestati. In caso di cancellazione di una tupla dalla tabella books è stata utilizzato il comando “cascade” che si occupa della cancellazione di tutte le tuple riferite nella tabella loans con quella chiave;

- id_users_fk: mette in relazione l’attributo id_user della tabella loans con l’attributo fiscal_code di users, per creare un collegamento tra la tabella dei prestiti e quella degli utenti in modo da tenere traccia dell’utente che ha ricevuto un prestito. In caso di cancellazione di una tupla dalla tabella users è stata utilizzato il comando “cascade” che si occupa della cancellazione di tutte le tuple riferite nella tabella loans con quella chiave;

- authors_fk: mette in relazione l’attributo id_author della tabella write con l’attributo id_aut di authors, con lo scopo di creare un collegamento tra l’id dell’autore memorizzato nella tabela write e i dati dell’autore conservati nella tabela authors. In caso di cancellazione di una tupla dalla tabella authors è stata utilizzato il comando “cascade” che si occupa della cancellazione di tutte le tuple riferite nella tabella write con quella chiave;

- books_fk: mette in relazione l’attributo id_book della tabella write con l’attributo id_book di books, per creare un collegamento tra il codice ISBN memorizzato nella tabella write e I dati del libro nella relazione books. In caso di cancellazione di una tupla dalla tabella books è stata utilizzato il comando “no action” perché gli autori che hanno scritto un libro potrebbero averne scritti altri, è il programma java che attraverso una opportuna procedura si occupa di rimuovere gli autori contenuti nella tabella authors che non hanno nessuna ricorrenza nella tabella write;

Per quanto riguarda invece la modalità “on update” in tutte le chiavi esterne è stato definito il comando “cascade”.

Abbiamo deciso quindi di implementare alcuni vincoli check per controllare la consistenza dei dati immessi nel database:

- ALTER TABLE public.books ADD CONSTRAINT check_year CHECK (reprint_year >= publication_year);
  controlla che l’anno di ristampa del libro sia successivo o uguale all’anno di pubblicazione;
- ALTER TABLE public.loans ADD CONSTRAINT check_date CHECK (return_date >= loan_date);

controlla che la data di restituzione di un prestito sia successiva o uguale alla data di inizio prestito

Vista

Terminata l'implementazione della base di dati abbiamo deciso di creare una vista contente le informazioni degli utenti bibliotecari, dopo aver considerato che tramite AppLibrarian possono loggarsi solo utenti bibliotecari.
Questa scelta deriva dalla necessità di velocizzare il tempo impiegato dalla query per individuare nel database l'utente corrispondente ai dati inseriti nella fase di login.

Il codice sql della vista implementata è il seguente:

```
CREATE OR REPLACE VIEW public.view_librarian WITH (check_option=local) AS

SELECT users.name, users.surname, users.fiscal_code, users.telephone_number, users.email,
users.classification, users.year_class, users.activation_code, users.password,
users.temporaney_pwd, users.counter
FROM users
WHERE users.classification::text = 'LIBRARIAN'::text;
```

Per eseguire le query sulla base di dati attraverso il codice Java ci siamo basati sulle funzionalità offerte dalla libreria "postgresql-42.1.4" fornita ufficialmente dal supporto di PostgreSQL. Per una descrizione dettagliata delle query si consulti il file "Query.pdf" allegato alla relazione.

Codice

Nella stesura del codice sono state implementate le scelte di progettazione e i design patterns descritti in precedenza.

Particolare attenzione è stata posta sull'utiizzo di nomi, per variabili e metodi, che fossero chiari e spiegassero il ruolo dell'entità.

Per gestire i problemi relativi alla concorrenza che possono presentarsi nell'esecuzione di una applicazione client – server è stato utilizzato il costrutto synchronized e sono state utiizzate le tecniche ilustrate nella parte relativa alle scelte progettuali.

Al fine di evitare un'eccessivo utilizzo delle risorse, intese sia come spazio occupato in memoria che utilizzo della connesione, abbiamo deciso di inviare i libri richiesti dal client, per la visualizzazione nelle interfacce, a bocchi di massimo venticinque elementi.
Per realizzare ciò ci siamo affidati alle istruzioni SQL "LIMIT" che permette di impostare il numero di elementi massimo da prelevare dal database e "OFFSET" che specifica quanti elementi ignorare prima di iniziare a prelevare dati.

In questo modo abbiamo potenzialmente aumentato il numero di richieste di visualizzazione del client nei confronti del server, ma allo stesso tempo siamo stati in grado di diminuire il carico di lavoro che avrebbero avuto entrambi i componenti se avessimo deciso di inviare direttamente l'intero catalogo dei libri, soprattutto in caso di catologhi di notevoli dimensioni.

Inoltre per rispettare la richiesta di non memorizzare le credenziali su memoria di massa, le abbiamo richieste nell'interfaccia iniziale all'avvio del server e ne abbiamo tenuto traccia all'interno di variabili private nel codice.

È stata infine presa la decisione di memorizzare indirizzo IP e porta di ascolto del server in un file di testo chiamato "config.txt", e di leggere queste informazioni durante la fase di avvio di entrambi i clients per configurarli correttamente.
La scelta di questa strategia è dovuta al fatto che in questo modo nel caso in cui i dati della connessione del server cambino, sarebbe sufficiente cambiare le informazioni in questo file, al contrario il salvataggio di indirizzo IP e porta nel codice avrebbero costretto ad una operazione ben più complessa.

4. Informazioni aggiuntive

Ant

Ant è una libreria JAVA sviluppata da Apache che permette di automatizzare il processo di sviluppo di applicazioni Java.

Grazie ad Ant, infatti, è possibile creare un progetto che compila, genera la documentazione e realizza i file jar di un'applicazione.

La realizzazione dello script Ant permette a tutte le macchine con Ant installato di generare facilmente e velocemente tutti i compilati del progetto senza dover dipendere da tool quali Eclipse.

I comandi che Ant esegue sono letti da un file XML, chiamato di default build.xml. In questo file sono definite le operazioni disponibili dette "target" e, per ciascuna di esse, i comandi da eseguire detti ''task''.

Ciascun target può avere target dipendenti, ciò vuol dire che, se viene richiesta l'esecuzione di un target che possiede dipendenze da altri target, Ant eseguirà prima i suoi target dipendenti.

Il file build.xml è composto da un tag <project> che contiene a sua volta tutti i comandi disponibili, denominati target. Nel tag <project> è anche definito quale target eseguire per primo, questo sarà posto dopo "default=".

Il file build.xml di questo progetto può essere analizzato suddividendolo in blocchi composti da "target", ognuno con funzioni diverse:

- Il primo target chiamato "compile'' ha il compito di compilare i file presenti nella cartella "bin".
- Il secondo target chiamato ''jar'' ha il compito di generare i file .jar dei file presenti nella cartella ''bin''. Questo target ha dipendenza dal target ''compile''
- Il terzo target chiamato ''doc'' ha il compito di generare la documentazione del programma.

Alcuni "task" utilizzati nel file build.xml sono:

- <mkdir> task che permette la creazione di una cartella.

- <javac> task che permette la compilazione di uno o più file.
- <copy> task che permette di copiare dei file.
- <delete> task che permette di cancellare dei file.
- <jar> task che permette la generazione dei file di tipo .jar.
- <manifest> task che rende un file di tipo .jar eseguibile.
- <fileset> task che permette di agire su più file in contemporanea.
- <zipfileset> uno speciale tipo di <fileset> che, a seconda degli attributi passati quali "src" o ''bin'', agisce in modo differente.

Il sistema è stato testato su sistema operativo Windows 10, i comandi ant eseguibili nella directory contente il file di build sono:

- Generazione dei file .jar: Eseguire comando "ant";
- Generazione dei file .class: Eseguire comando "ant compile";
- Generazione JavaDoc: Eseguire comando "ant doc"

Allegati aggiuntivi

Insieme a questa relazione vengono allegati oltre ai file richiesti nella traccia del progetto, anche i seguenti file:

- "script_db.sql": script SQL contente tuto il codice che costituisce il database "libraryDb". Questo script può essere eseguito per la creazione immediata di tutta la base di dati necessaria per il funzionamento del programma;
- "backup schema.backup": file di backup dello schema della base di dati generato grazie a PostreSQL che permette attraverso il programma di ripristinare lo schema del database;
- "backup database.backup": file di backup della base di dati generato grazie a PostreSQL che permette attraverso il programma di ripristinare lo schema e i dati del database. I dati contenuti in questo file sono stati inseriti manualmente nelle tabelle ed utilizzati durante la fase di testing, e consistono in circa 80 libri, circa 45 prenotazioni, circa 75 prestiti attivi o conclusi e circa 45 utenti divisi tra i vari inquadramenti;
- "descrizione query.pdf": file pdf contenente tutte le query utilizzate nel progetto seguite da una breve descrizione del loro scopo e utilizzo;
- - "manuale utente.pdf": file che contiene il manuale utente con la spiegazione delle funzionalità implementate nell'applicazione.

Funzionalità aggiuntive implementate

Oltre alle funzionalità richieste nella traccia del progetto, abbiamo deciso di implementarne altre per migliorare l'esperienza dell'utente con l'applicazione:

- Abbiamo realizzato la possibilità di effettuare ricerche parametriche per autore, categoria e titolo non solamente per i libri nel catalogo ma anche per facilitare gli utenti nella ricerca di prenotazioni o prestiti;
- Possibilità per l'utente bibliotecario di inserire nel sistema più libri uguali senza dover ripetere le informazioni per ogni inserimento. In questo scenario il bibliotecario può compilare tutte le informazioni relative al libro e successivamente inserire in uno spinner la quantità di libri da aggiungere alla libreria, il sistema provvederà automaticamente a registrare la quantità di libri inserita aumentando in modo progressivo il codice ISBN dei libri a partire dal primo inserito.
  Questo processo avviene in modo completamente trasparente rispetto all'utente.
  L'idea di questa funzionalità nasce dal fatto che solitamente le librerie dispongono di più copie dello stesso libro che non vengono ordinate singolarmente;

Abbiamo deciso di criptare la password degli utenti con l'algoritmo MD5 prima di salvarla nel database per migliorare la sicurezza dell'applicazione. MD5 è una funzione di hash crittografica unidirezionale e irreversibile, ciò significa che data in input una stringa di lunghezza arbitraria la funzione ne restituisce un'altra a 128 bit dalla quale è impossibile risalire alla stringa di input.
Il processo di generazione è molto veloce e assicura che è altamente improbabile ottenere con due diverse stringhe in input uno stesso valore hash in output.

Possibili miglioramenti

Di seguito vengono elencati alcuni aspetti dall'applicazione che a nostro parere possono essere migliorati, seguiti da una breve descrizione di una possibile soluzione:

- Criptare le password in modo più efficiente e sicuro: migliorare la funzione per criptare le password degli account. Si potrebbe per esempio generare un salt randomico, associato al profilo, da concatenare alla password prima di criptarla oppure utilizzare un algoritmo più sicuro ed efficiente di MD5, ad esempio bcrypt o scrypt;

- Codici di attivazione temporanei: sempre nell'ottica di miglioramenti relativi alla sicurezza si potrebbe implementare un metodologia per rendere temporanei i codici di attivazione in modo da accettarne uno solo se inserito entro un intervallo di tempo predefinito, per esempio quindici minuti;

- Migliorare l'interfaccia grafica: rendere l'interfaccia grafica più user-friendly e più gradevole esteticamente per migliorare ulteriormente l'esperienza dell'utente con l'applicazione.

5. Riferimenti esterni

Componenti utilizzati

- IDE utilizzato per la progettazione e la stesura del corpo del programma:

- Javadoc: strumento che permette di generare la documentazione di un programma attraverso l'inserimento di tag specifici nel codice stesso.

- WindowBuilder: estensione di Eclipse che permette di sviluppare interfacce grafiche tramite semplici operazioni di drag-and-drop.

- Ant: Apache Ant è un software per l'automazione del processo di build. È scritto in Java ed è principalmente orientato allo sviluppo in Java. Ant è un progetto Apache, open source, ed è distribuito sotto licenza Apache.

- Postgresql: software che mette a disposizione un sistema di database relazionale avanzato, utilizzato appunto per sviluppare il database.

- DbVisualizer: software per sviluppatori e amministratori di database, che aiuta sia nello sviluppo che nella manutenzione del database stesso.

- Visual Paradigm Community Edition : strumento di modellazione UML, utilizzato per i sequence diagram e per lo use case diagram.

Sitografia

1. http://www.stackoverflow.com : sito con domande e risposte relative alla programmazione java;

2. http://forum.html.it/forum : sito con guide relative alla programmazione java;

6. Suddivisione del carico di lavoro

I primi passi nello sviluppo del progetto sono stati una attenta lettura della traccia, l'analisi dei requisiti e la divisione iniziale del lavoro tra i componenti del gruppo.

Successivamente abbiamo proseguito con la realizzazione degli artefatti UML per descrivere il sitema e degli schemi ER e logico della base di dati.
Contemporaneamente sono state fatte numerose e approfondite ricerche sulle strutture dati da utilizzare e sui design patterns che erano più adatti alla soluzione del problema proposto.

La fase seguente è stata quella di procedere con lo sviluppo in parallelo delle varie componenti del sistema seguite dai test per verificarne il corretto funzionamento.

Terminata l'implementazione di tutti i componenti del programma, sono stati effettuati numerosi test del sitema finale che hanno portato alla correzione di diversi malfunzionamenti e a rendere il codice più leggero e più leggibile.

Divisione dei compiti:

Marco di Capua si è occupato della progettazione e implementazione di tutte interfacce grafiche e dell'utilizzo di ant.

Mattia Lo Schiavo ha progettato e sviluppato il database su cui si appoggia il server e ha realizzato i sequence diagram.

Filippo Pelosi si è occupato dello sviluppo del codice per quanto riguarda la logica delle tre applicazioni e la loro struttura.

Riccardo Zorzi ha scritto questa relazione e si è occupato di tutti gli allegati del progetto e della realizzazione dei sequence diagram e del class diagram.

Tutti i componenti del gruppo hanno collaborato alla fase di analisi dei requisiti, progettazione del programma e alle fasi di test.

La javadoc è stata scritta in parallelo allo sviluppo dei componenti.

7. Conclusioni

È stato un progetto impegnativo ma allo stesso tempo molto avvincente che ha sicuramente contribuito ad un aumento delle conoscenze sia nell'ambito della progettazione che della programmazione.

Durante lo sviluppo del progetto sono emerse varie difficoltà e imprevisti, dovuti anche al fatto che un componente del gruppo è un lavoratore full-time, che sono stati brilantemente risolti grazie alle numerose riunioni e al costante contatto tramite Whatsapp.