UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Sviluppo di una Web Application di una biblioteca digitale e integrazione in una rete locale

Candidato: *Calogero Giudice*

Relatore: *Prof.ssa Simona Turbanti*

Correlatore: *Prof. Claudio Gallicchio*

Anno Accademico 2020-2021

Tutto in me è cambiato
e questo gioco è appena cominciato
Spingendo fino in fondo
il sogno si realizzerà

Indice

1. Introduzione

È stata svolta un'attività di tirocinio con l'Istituto Comprensivo "*Toniolo*" di Pisa, con il quale si è pensato di creare un sistema bibliotecario interno accessibile al suo bacino di utenti, dagli alunni ai docenti.

L'idea è sorta dal fatto che essi, per poter consultare il catalogo, disponevano di un elenco PDF statico: scorrere a mano cataloghi statici di libri alla ricerca della voce desiderata può risultare molto difficoltoso, specie se il numero di notizie bibliografiche ammonti all'ordine delle migliaia.

L'obiettivo principale di questo elaborato è quello di raccogliere le informazioni principali del complesso scolastico (come è organizzato, il tipo di utente che può interagire, la struttura degli elenchi, il sistema dei prestiti e così via), analizzarle, elaborare un modello che soddisfi le esigenze dell'istituto e sviluppare così una Web Application apposita.

Durante lo sviluppo è stato seguito un metodo semplice e lineare: da un'elaborazione concettuale si è passati a una schematica riuscendo ad ottenere, a sua volta da essa, il prodotto commissionato.

Nel capitolo 2 verranno analizzati i requisiti dai quali verrà estrapolato il modello utilizzato per il database. Lo sviluppo vero e proprio di questo, con tutta l'interfaccia e le funzionalità necessarie al funzionamento, sono trattate nel successivo (capitolo 3). Esiste una rete bibliotecaria chiamata "Bibliolandia" che interconnette diversi sistemi bibliotecari nell'area pisana. Nel capitolo 4 si analizzerà la struttura di tale rete e si faranno delle valutazioni riguardo una possibile integrazione del sistema dell'istituto all'interno di Bibliolandia.

Adesso si procederà con l'inizio della trattazione, a cominciare dallo studio dei requisiti dell'istituto scolastico.

2. Analisi dei requisiti

Prima di procedere alla progettazione vera e propria del sistema bibliotecario è bene introdurre i concetti fondamentali che illustrano la progettazione della stessa. La progettazione di una Web Application si articola principalmente in sei fasi:

1. *Analisi dei requisiti*
   È una fase quasi formale, un primo approccio tra il progettista e il committente. Vengono illustrati i requisiti fondamentali della piattaforma richiesta, i meccanismi e le operazioni attuabili. In seguito, è compito del progettista analizzare tali requisiti cercando di soddisfare il più possibile le esigenze dell'ente committente. Questa fase è articolata nel § 2.1, in cui è stato esposto il lavoro richiesto dall'istituto.

2. *Progettazione concettuale*
   Consiste nella "traduzione" dei requisiti in elementi grafici per chiarire quali sono le entità coinvolte e le relazioni che intercorrono tra di esse. In questa fase ci si avvale dello schema Entità-Relazione (qui schema E-R), i cui elementi fondamentali verranno esposti in questo paragrafo.

3. *Progettazione logica*
   Si trasforma lo schema generato nella fase precedente in vere e proprie tabelle, tenendo conto di tutti i vincoli richiesti. Qualora si tratti di uno schema ridondante, è possibile farne anche una normalizzazione. Per motivi pratici è bene valutare se lo schema generato sia performante.

4. *Progettazione fisica*
   Si scelgono le strutture di memorizzazione del database, si valuta l'efficienza dello schema integrando eventuali strutture d'accesso. Si trasformano le tabelle in uno schema fisico in formato SQL e si gestiscono le transazioni. Per semplicità questa fase di progettazione è stata saltata.

5. *Programmazione dell'interfaccia*
   Su base del database creato si sviluppa l'interfaccia più fedelmente possibile alle esigenze del committente. Prima si lavora lato client e poi si integra ove possibile l'interazione con il server.

6. *Verifica*
   Si verifica il progetto finito: si fa un test delle performances, si risolvono eventuali bug presenti e si fa una valutazione finale del progetto. Se il lavoro soddisfa i requisiti richiesti, si valuta il caricamento della piattaforma su server.

Esposte le fasi principali della progettazione della Web Application è il momento di vedere nel dettaglio i requisiti della biblioteca e le fasi successive dell'analisi e della progettazione del database.

2.1. Sommario dei requisiti

Si è richiesto di sviluppare una Web Application in grado di gestire l'insieme delle biblioteche dell'Istituto Comprensivo "Toniolo" di Pisa.

Esso è strutturato in diversi plessi. Ciascun plesso è identificato da un codice univoco del tipo *PIXX00000X*[1], da un nome inteso come *alias* e da un tipo, poiché un plesso può essere una scuola dell'infanzia, primaria oppure secondaria. Informazioni aggiuntive ed accessorie sono l'indirizzo, il numero di telefono e l'e-mail.

Dopo averne introdotto il concetto, bisogna capire come è strutturata la singola biblioteca di ciascun plesso. Nell'azione del prestito figurano tre entità fondamentali: l'oggetto, il fautore e l'agente del prestito. Risulta evidente che in una biblioteca si identifichino rispettivamente le entità di libro, utente e operatore.

[1] Con *X* si intende una lettera maiuscola, mentre con *0* una cifra.

Analizzando l'elenco dei libri della biblioteca di un plesso, la registrazione bibliografica di un libro consta di un titolo, di un autore, di un editore, di un anno, di una collocazione e del plesso in cui è disponibile. Inoltre, un libro ha uno stato in quanto può essere disponibile al prestito oppure già in prestito. Tuttavia, esistono delle categorie di libri, come i vocabolari, che spesso sono di sola consultazione all'interno di una biblioteca. Dunque, è necessario anche distinguere i testi disponibili al prestito da quelli di sola consultazione.

L'utente di una biblioteca è iscritto ad un plesso ed è caratterizzato dai suoi dati anagrafici, ovvero il nome, il cognome e la data di nascita. Possiede anche un tipo, giacché l'utente può trattarsi di un alunno, di un alunno che ha conseguito il diploma oppure un docente. Qualora si tratti di un alunno, quest'ultimo è iscritto a una sezione di una determinata classe; in caso contrario, questi due parametri non sono definiti. Oltre a questi dati, un utente deve fornire un numero di telefono al fine di mantenere un contatto con l'amministrazione della biblioteca. A ciascun utente viene assegnato un codice univoco di sei caratteri alfanumerici con il quale è possibile prenotare un determinato libro da remoto, se disponibile.

L'operatore di una biblioteca è un suo componente amministrativo. L'operatore è l'unico in grado di accedere alla piattaforma e gestire i libri, gli utenti e i prestiti. Un operatore può essere di due tipi: l'*admin* semplice, limitato alla gestione di libri, utenti e prestiti, e il *gestore*, in grado di aggiungere e gestire gli operatori della biblioteca e di accedere ad alcune funzionalità aggiuntive. Un operatore ha un nome, un cognome e un tipo (admin o gestore), oltre al plesso di appartenenza. Per poter accedere, l'operatore necessita di uno username e di una password. Per eventuali comunicazioni da parte dell'istituto è necessario fornire anche un indirizzo e-mail.

Il prestito è una relazione che intercorre tra le quattro entità esposte poco sopra. Infatti, come detto in precedenza, un prestito è dato dal libro in oggetto, dall'utente che intende prenderlo e dall'operatore che deve ufficializzarlo. Oltre a questi parametri fondamentali, sono necessari anche la data di prestito, la data di

restituzione, il plesso dove è stato effettuato e lo stato del prestito. Quest'ultimo può infatti assumere tre stati: uno stato di *attesa*, nel caso di una prenotazione di un libro da remoto, uno stato di prestito *effettivo* e uno di prestito *concluso*, ovvero che il libro è stato restituito.

Oltre a questo, può essere importante monitorare le azioni che si susseguono, per cui è possibile anche consultare uno storico delle azioni compiute, qui di seguito *log*. Una voce del *log* consiste nella collocazione storica di un'azione compiuta all'interno del database. Comprende il tipo di azione (un inserimento, un aggiornamento, una cancellazione o una promozione, nel caso di un alunno), il tipo di entità coinvolta, l'entità manipolata, l'operatore che ha compiuto tale operazione e data ed ora effettive di azione.

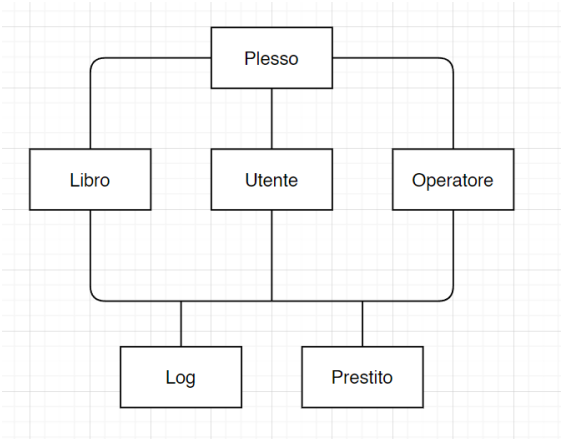

Figura 1 *- Gerarchia degli elementi principali del database della biblioteca*

La ***Figura 1*** mostra uno schema iniziale che mette in relazione i singoli elementi costituenti il database descritto in questo paragrafo. Come si può ben notare, il plesso è l'elemento *genitore*: senza di esso, gli altri elementi non hanno alcun significato. I figli diretti di un plesso sono i libri, gli utenti e gli operatori poiché tutte queste entità devono necessariamente dipendere dal plesso di appartenenza. Il prestito rappresenta un figlio di tre elementi appena delineati, in quanto un prestito non è altro che una relazione tra il libro in prestito, l'utente che intende prenderlo in prestito e l'operatore che lo accetta. Infine, anche il log rappresenta un figlio del libro, dell'utente e dell'operatore poiché vengono registrate le cosiddette –operazioni *CUD* (*Create, Update, Delete*).

Modellare un semplice catalogo, tuttavia, non è una condizione sufficiente per poter sviluppare una Web Application di questo tipo. Le informazioni bibliografiche di un sistema bibliotecario devono attenersi a dei requisiti funzionali, ovvero i *Functional Requirements for Bibliographic Records* (FRBR)[2]. Affinché un catalogo bibliografico possa essere ritenuto tale, oltre ai requisiti funzionali è necessario verificare se le informazioni bibliografiche soddisfino le *Regole Italiane di Catalogazione* (REICAT)[3]. Di tutto questo si discuterà in seguito, nel § 4.1.1.

Fatto il punto sulla situazione, è bene procedere alla progettazione concettuale del database (§ 2.2), per poi proseguire verso quella logica (§ 2.3) e infine arrivare allo sviluppo vero e proprio della piattaforma (§ 3).

2.2. Progettazione concettuale

Preso in esame il sommario dei requisiti (§ 2.1), si trasforma in uno schema che riassume i concetti presi in esame.

2.2.1. Schema Entità-Relazione (E-R)

Lo schema E-R permette di descrivere graficamente le entità e le relazioni del database in oggetto. Questo tipo di schema è tipico nell'ambito della descrizione di un modello relazionale, così come quello dei database, in cui vengono definite determinate relazioni tra elementi appartenenti a un medesimo schema. In questo progetto, lo schema è basato su FRBR di cui accennato nel § 2.1.

[2] International Federation of Library Associations and Institutions, Study Group on the Functional Requirements for Bibliographic Records, *Functional requirements for bibliographic records: final report*, approved by the Standing Committee of the IFLA Section on Cataloguing, München, Saur, 1998.

[3] *Regole italiane di catalogazione: REICAT*, a cura della Commissione permanente per la revisione delle regole italiane di catalogazione, Roma, ICCU, 2009.

Per costruire uno schema E-R è necessario delineare gli elementi fondamentali che lo costituiscono. In questa relazione si prendono in considerazione solo gli elementi utilizzati. Bisogna sottolineare che la ***Figura 1*** non rappresenta uno schema E-R in quanto mette in evidenza gli elementi del database, ma solo a un livello basilare.

Un'*entità* rappresenta un insieme di elementi che possiedono tratti e caratteristiche comuni, la cui esistenza è autonoma. Identificata da un rettangolo, un esempio di entità è il libro, in quanto tutti i libri di una singola biblioteca possiedono un titolo, un autore, un editore, etc. Su base delle analisi dei requisiti, le entità individuate nel database sono quattro: il *plesso*, il *libro*, l'*utente* e l'*operatore*.

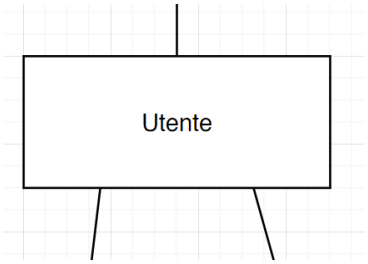

Figura 2 *- Raffigurazione dell'entità Utente*

Una *relazione* è un legame che, appunto, mette in relazione due o più entità. Così come quest'ultime, possono includere degli attributi. Solitamente una relazione collega due entità distinte, ma anche più di una, oppure una singola entità con sé stessa. In questa situazione, come verrà esposto in seguito, vi sono due relazioni che collegano tre entità e tre relazioni che ne collegano due. Si rappresenta con un rombo a cui sono associate le entità coinvolte.

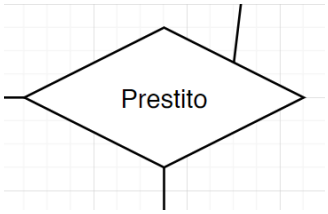

Figura 3 *- Raffigurazione della relazione Prestito*

Un *attributo* è una singola caratteristica di un'entità o di una relazione. Ad esempio, un attributo di un utente può essere il nome, oppure la sezione qualora si tratti di un alunno. Per rappresentare un attributo si indica un cerchio vuoto (○) collegato all'entità o alla relazione corrispondente. Supponendo di inserire un attributo "*Cognome*" all'entità di ***Figura 2***, si ottiene quanto in ***Figura 4***.

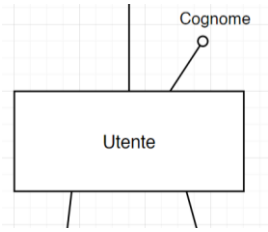

Figura 4 *- Entità Utente con l'attributo Cognome*

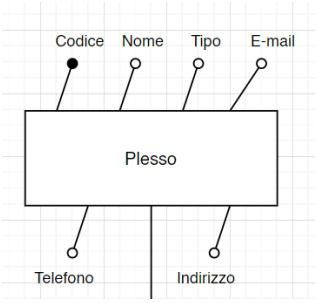

***Figura 5** - Entità Plesso completo di attributi e identificatore interno Codice*

Un *identificatore* è un attributo, o un insieme di attributi, che identificano univocamente un'entità o una relazione. Esso può essere interno alla tabella di riferimento (una chiave primaria[4]), oppure esterno (recapitato da un'altra entità o relazione). Si indica esattamente come l'attributo, ma l'indicatore è pieno (●): si veda ***Figura 5***.

La *cardinalità* `(X,Y)` indica il tipo di relazione che insiste tra una entità e una relazione. All'interno di uno schema, un elemento può essere facoltativo oppure obbligatorio oppure presente in più istanze. La ***Tabella 1*** illustra le singole combinazioni possibili a livello di cardinalità. Essa è interpretabile come l'espressione "*a X elementi si possono associare Y elementi*":

		Y	
		1	**N**
X	**0**	**(0,1)** – Zero a uno	**(0,N)** – Zero a molti
	1	**(1,1)** – Biunivocità	**(1,N)** – Uno a molti
	N		**(N,N)** – Molti a molti

***Tabella 1** - Combinazioni possibili di cardinalità*

[4] Per *chiave primaria* si intende un campo o un insieme di campi che identificano un record all'interno di una tabella. I campi di chiave non possono mai contenere valori *NULL* (non esistenti oppure sconosciuti).

Definiti gli elementi utilizzati, si definisce lo schema corrispondente all'analisi dei requisiti presentati nel § 2.1: lo schema completo è in ***Figura 6***:

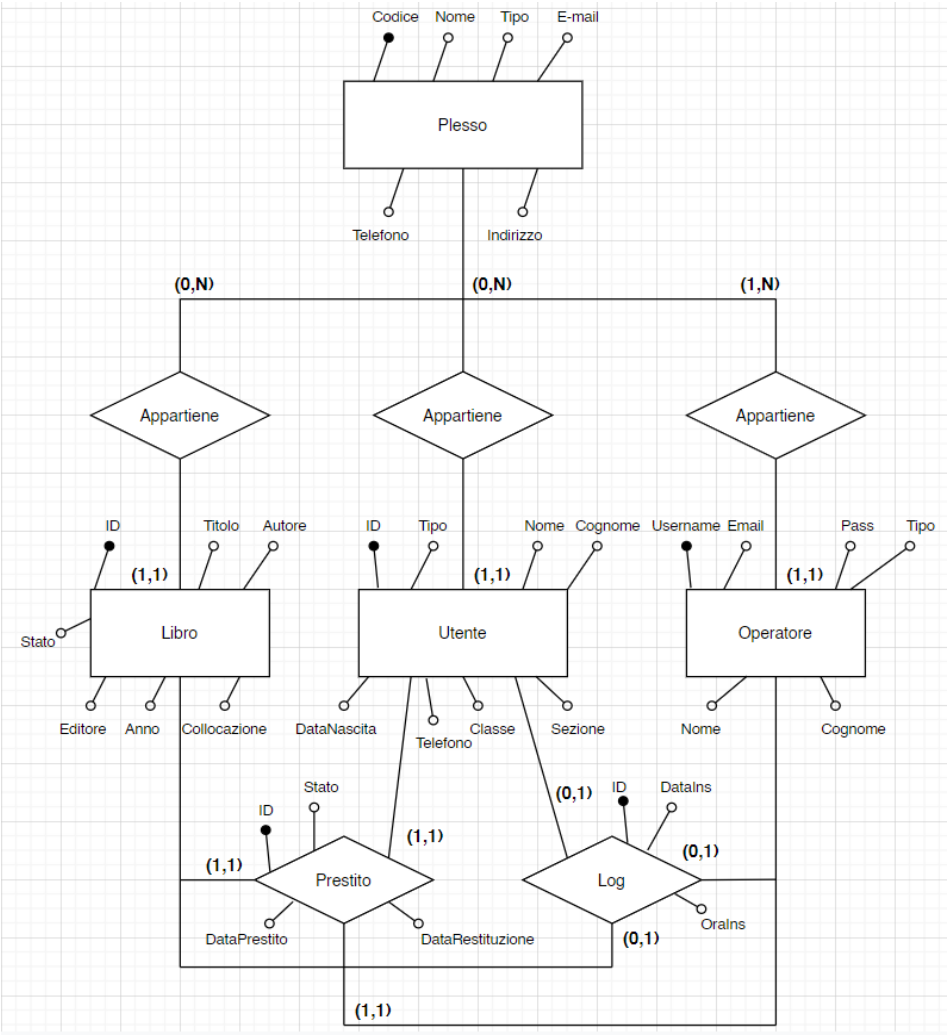

Figura 6 *- Schema E-R della Web Application del sistema bibliotecario dell'Istituto*

Si noti come il modello in ***Figura 6*** è in formato rigorosamente più completo e ricco di informazioni rispetto allo schema gerarchico presentato in ***Figura 1***.

Completata questa fase di progettazione della Web Application, quindi dopo aver fatto chiarezza sulle entità individuate e le relazioni che intercorrono tra esse, è necessario tradurre il modello in qualcosa di più schematico e quindi più semplice da interpretare in seguito.

2.3. Progettazione logica

La progettazione logica di una base di dati consiste nell'organizzare tutte le informazioni fornite dal modello E-R (§ 2.2) in tabelle, effettuare eventuali normalizzazioni (§ 2.3.2), e fare una stima delle prestazioni (§ 2.3.3).

2.3.1. Definizione delle tabelle e dei vincoli

Dal modello E-R bisogna estrarre le singole informazioni e ordinarle nel modo più schematico possibile: per una trattazione più completa è bene prendere in considerazione il nome, una breve descrizione, gli attributi e gli identificatori di ogni entità e relazione individuati in precedenza.

Analizzando le entità presenti in ***Figura 6*** si descrivono le seguenti entità, qui mostrate in ***Tabella 2***:

Entità	Descrizione	Attributi	Identificatore
Plesso	Luogo in cui è presente una biblioteca	Nome, Tipo, E-mail, Indirizzo, Telefono	Codice
Libro	Oggetto di un prestito nella biblioteca di un plesso	Titolo, Autore, Editore, Anno, Collocazione, Stato	ID
Utente	Persona all'interno di un plesso che intende prendere in prestito un libro	Tipo, Nome, Cognome, Data di nascita, Telefono, Classe, Sezione	Codice
Operatore	Agente che effettua un prestito e gestisce la biblioteca	E-mail, Password, Tipo, Nome, Cognome	Username

Tabella 2 *- Riepilogo delle entità presenti*

Per quanto riguarda le relazioni, si ottiene lo schema seguente, illustrato in ***Tabella 3***:

Relazione	Descrizione	Attributi	Entità coinvolte	Identificatori
Appartiene	Relazione di appartenenza di un libro/utente/operatore a un plesso	—	Libro	—
			Utente	
			Operatore	
			Plesso	
Prestito	Voce di un prestito	Data prestito, Data restituzione, Stato	Libro, utente, operatore	Codice
Log	Voce con cui si tiene traccia delle azioni compiute	Data, Ora, Tipo	Una delle entità presenti	ID

Tabella 3 *- Riepilogo delle relazioni presenti*

Tramite le tabelle ***Tabella 2*** e ***Tabella 3***, il modello E-R è stato frammentato in modo da renderlo più comprensibile. Non considerando la relazione *Appartiene* in ***Tabella 3***, inizialmente si individuano sei tabelle all'interno dello schema del database, qui di seguito descritte. Nota che in grassetto si indicano i nomi delle singole tabelle mentre si sottolineano gli identificatori:

- **Plesso** (<u>Codice</u>, Tipo, Nome, Indirizzo, Telefono, Email)
- **Libro** (<u>ID</u>, Titolo, Autore, Editore, Anno, Collocazione, Plesso, Stato)
- **Utente** (<u>ID</u>, Tipo, Cognome, Nome, DataNascita, Telefono, Classe, Sezione, Plesso)
- **Operatore** (<u>Username</u>, Email, Password, Nome, Cognome, Tipo, Plesso)
- **Prestito** (<u>ID</u>, Libro, Utente, Operatore, DataPrestito, DataRestituzione, Plesso, Stato)
- **Log** (<u>ID</u>, DataIns, OraIns, Azione, Tipo, Elemento)

Affinché i dati presenti nel database risultino consistenti, è necessario introdurre determinate restrizioni, in gergo *vincoli*[5], che ne garantiscano sia il senso logico che la consistenza, per l'appunto, dei dati stessi. Esistono due categorie principali di vincoli: quelli *intrarelazionali* che sussistono all'interno di una medesima tabella e quelli *interrelazionali*, ovvero tra più di una tabella.

Questi sono riuniti nella prima tabella di ***Tabella 4***. Invece nella seconda della medesima figura sono specificati i *vincoli di integrità referenziale*: stabiliscono le relazioni tra un attributo di una tabella con la chiave di un'altra dello stesso database.

[5] Un *vincolo* è una condizione che deve essere soddisfatta affinché una determinata relazione sia corretta e consistente.

Prendendo in esame i requisiti richiesti nel § 2.1, nello studio si considerino i seguenti vincoli opportunamente catalogati per il loro tipo qui in ***Tabella 4***:

Vincoli di record			
#	**Tabella**	**Attributo**	**Descrizione**
1	Plesso	Codice	Il codice del plesso deve essere nel formato *PIXX00000X*
2	Libro	Anno	L'anno del libro deve essere compreso tra il 1900 e l'anno corrente inclusi
3		Collocazione	La collocazione di un libro deve essere nel formato *X/000y*
4	Utente	ID	L'ID di un utente è una stringa alfanumerica di sei caratteri
5		DataNascita	La data di nascita di un utente varia in base al tipo dell'utente stesso e al tipo di plesso in cui è iscritto (infanzia, primaria o secondaria)
6		Classe	La classe di un utente è definita solo se quest'ultimo è un alunno. Partendo da 1, in base al plesso in cui è iscritto il valore massimo varia da 3 a 5
7		Sezione	La classe di un utente è definita solo se quest'ultimo è un alunno. Si tratta di una lettera maiuscola
8	Operatore	Username	Lo username di un operatore è formato da 6 a 20 caratteri alfanumerici
9		Pass	La password di un operatore deve avere almeno 6 caratteri di cui almeno una maiuscola, una cifra e un carattere speciale
10	Prestito	DataPrestito	Un prestito *in attesa* non ha note le date di prestito e restituzione, uno *in prestito* ha nota solo quella di prestito, uno *concluso* ha entrambe le date note
		DataRestituzione	
		Stato	

Vincoli di integrità referenziale		
#	**Attributo di partenza**	**Attributo di riferimento**
1	Libro (Plesso)	Plesso (Codice)
2	Utente (Plesso)	
3	Operatore (Plesso)	
4	Prestito (Plesso)	
5	Prestito (Libro)	Libro (ID)
6	Prestito (Utente)	Utente (ID)
7	Prestito (Operatore)	Operatore (Username)
8	Log (Elemento)	Libro (ID)
9		Utente (ID)
10		Plesso (Codice)
11		Operatore (Username)
12	Log (Admin)	

Tabella 4 *- Riepilogo dei vincoli individuati*

2.3.2. Normalizzazione

Come illustrato nel § 2.3.1, a partire dai requisiti richiesti è stata fatta una prima trasformazione in singole tabelle. Dopo un riepilogo concettuale è stata fatta una prima organizzazione delle idee e una successiva traduzione in tabelle. Per una piattaforma più performante, in questa fase è necessario svolgere un'operazione di normalizzazione, sia per controllare se è stata svolta una buona traduzione, sia per analizzare eventuali ridondanze e rimuovere quelle non necessarie.

La normalizzazione è un procedimento che verifica il modello in modo che soddisfi le condizioni di una forma normale. Quest'ultima "certifica" la qualità dello schema formato: infatti se un modello non si presenta in forma normale presenta ridondanze e anomalie, le quali possono compromettere le performances della Web Application.

A questo punto è necessario introdurre due concetti che permettono la trattazione di questa fase:

- La *ridondanza* è un'informazione affine a un'altra che si ripete all'interno di una relazione: un esempio potrebbe essere la visualizzazione di tutte le informazioni bibliografiche di uno stesso libro all'interno di una relazione di prestito. Si mostri un esempio di ridondanza in ***Tabella 5***: si noti come per uno stesso *Libro* 1 si ripetano ogni volta le medesime informazioni nei campi *TitoloLibro* e *AutoreLibro*:

IDPrestito	Utente	Data	Libro	TitoloLibro	AutoreLibro
1	AA0021	03/04	1	*Divina Commedia*	Dante Alighieri
2	BB2112	06/05	1	*Divina Commedia*	Dante Alighieri
3	CC3563	07/09	1	*Divina Commedia*	Dante Alighieri

Tabella 5 *- Esempio di ridondanza*

- L'*anomalia* è un inconveniente che può presentarsi nell'ambito dell'*inserimento*, dell'*aggiornamento* o dell'*eliminazione* di un dato. Ad esempio, quando si elimina un utente potrebbe essere molto difficoltoso identificare e rimuovere in seguito tutti i prestiti che egli ha effettuato in biblioteca.

Una verifica strutturale del database è attuabile tramite lo strumento di *dipendenza funzionale*. Essa rappresenta un vincolo di integrità che descrive legami funzionali tra gli attributi di una relazione.

Si prendano in considerazione le tabelle formate nel § 2.3.1 e se ne analizzino le dipendenze funzionali. Nel caso in oggetto, gli attributi di una tabella dipendono tutti dalla propria chiave primaria. Nello schema seguente per evitare eventuali ambiguità con gli attributi omonimi si pospone agli stessi la tabella di riferimento: ad esempio per indicare il tipo del plesso si scrive *TipoPl*. Per fare maggiore chiarezza si indicano le seguenti dipendenze:

- CodicePl → TipoPl, NomePl, IndirizzoPl, TelefonoPl, EmailPl
- IDLib → Titolo, Autore, Editore, Anno, Collocazione, PlessoLib, StatoLib
- IDUt → TipoUt, CognomeUt, NomeUt, DataNascita, TelefonoUt, Classe, Sezione, PlessoUt
- Username → Email, Password, NomeOp, CognomeOp, TipoOp, PlessoOp
- IDPr → IDLib, IDUt, Username, DataPrestito, DataRestituzione, CodicePl, StatoPr
- IDLog → DataIns, OraIns, Azione, TipoLog, Elemento

Anche grazie all'introduzione degli ID, nel caso in esame non si notano particolari ridondanze o anomalie. Si nota che in ogni dipendenza funzionale tutti gli elementi che non sono chiave dipendono univocamente dalla chiave stessa: si dice che lo schema proposto sia in *Terza forma normale* (3NF). Ciò implica che si tratta di un modello le cui ridondanze e anomalie sono ridotte al minimo.

2.3.3. Analisi delle prestazioni

Altra fase importante della progettazione logica è quella di analisi delle prestazioni. Serve a valutare l'efficienza del sistema in termini di quantità e peso delle operazioni che vengono compiute.

Per tali analisi ci si serve di due parametri, ovvero il *costo dell'operazione* (il numero medio di occorrenze visitate per eseguirne una) e l'*occupazione di memoria* (lo spazio necessario a memorizzarla). Per studiare tali parametri occorrono due variabili fondamentali: il *volume dei dati* (il numero di occorrenze per operazione e le dimensioni di ciascun attributo) e le *caratteristiche delle operazioni* (il tipo dell'operazione, la frequenza e i dati coinvolti).

In primis è bene fare una stima dei volumi delle entità e delle relazioni che formano l'intero schema. In ***Tabella 6*** si fa una stima del numero di record, e quindi della dimensione, che popolano il database.

Per stimare la dimensione minima in ***Tabella 6***, per ogni tabella del database si sono presi in considerazione i tipi dei singoli attributi[6], in quanto ogni tipo di dato ha una dimensione specifica. Le dimensioni in Byte di tutti i tipi di dato utilizzati nel progetto sono le seguenti:

- `VARCHAR(n)` | $(n + 2)$ B
- `CHAR` | 1 B
- `TINYINT` | 1 B
- `INT` | 4 B
- `DATE` | 3 B
- `TIME` | 5 B

Si prenda in considerazione, ad esempio, l'entità Plesso la cui struttura, non considerando attributi aggiuntivi in modo da non appesantire la lettura, è la seguente:

```
Codice VARCHAR(10),
Tipo TINYINT,
Nome VARCHAR(30),
Indirizzo VARCHAR(80),
Telefono VARCHAR(10),
Email VARCHAR(50)
```

A questo punto si sommano le dimensioni sopra. Quindi il peso di un'entità Plesso vale

$$[(10 + 2) + 1 + (30 + 2) + (80 + 2) + (10 + 2) + (50 + 2)]\ \text{B} = 191\ \text{B}$$

[6] Tali tipi di dato verranno spiegati in maniera più approfondita nel paragrafo § 3.2 in cui si illustra lo script di creazione del database.

Moltiplicando la dimensione per il volume stimato in ***Tabella 6*** si trova la dimensione totale. Lo stesso calcolo viene applicato a tutte le tabelle, in modo da ottenere la ***Tabella 6*** nella sua completezza. Indicando con *E* un'entità e con *R* una relazione si ha:

Elemento	Tipo	Volume	Dimensione	
			Unitaria	Totale
Plesso	E	7	191 B	1.31 KB
Libro	E	3600	217 B	762.9 KB
Utente	E	1200	102 B	119.53 KB
Operatore	E	15	378 B	5.54 KB
Prestito	R	2500	63 B	153.81 KB
Log	R	6000	58 B	339.84 KB
			Totale	**~ 1.35 MB**

Tabella 6 *– Tavola dei volumi*

Dalla tavola dei volumi in ***Tabella 6*** si stima che in totale l'intero database dell'istituto peserà almeno 1.35 MB.

Considerando il numero totale di libri e quello degli utenti, si ritiene si tratti di una dimensione di partenza alquanto ragionevole. Se si provasse a frammentare un determinato elemento ci sarebbe il rischio di aumentare in modo considerevole la portata dell'intero database e quindi appesantire lo spazio offerto dal server.

In ***Tabella 7a*** è illustrata una tavola delle operazioni. Si stima il numero delle operazioni che vengono effettuate all'interno del database entro un fissato intervallo di tempo. Per ogni riga si indicano l'operazione stessa, il suo tipo e la sua frequenza. Il tipo di un'operazione può essere di tipo interattivo, eseguita dall'utente (*I*) oppure di batch, automaticamente dal sistema, cioè in background (*B*).

Nota che ove possibile si moltiplicano le frequenze per sette perché si tiene conto del numero totale di operazioni in tutti i plessi. Si specifica che per *CRUD* viene inteso l'insieme delle operazioni di creazione (*Create*), lettura (*Read*), aggiornamento (*Update*) e cancellazione (*Delete*):

Invece nella ***Tabella 7b*** sono illustrati gli accessi delle singole operazioni. Per ogni operazione si esplicitano tutti i concetti in gioco e si indica il numero di accessi per ogni operazione. Il tipo di un accesso può essere di lettura (*R*) o di scrittura (*W*): quest'ultimo è di solito utilizzato nell'immissione dei dati all'interno del sistema bibliotecario.

Operazione	Descrizione	Tipo	Frequenza
1	Operazione *CRUD* su un plesso (*Read*, op. **1a**, *Create*, *Update* e *Delete* op. **1b**)	I	Almeno 7 volte
2	Inserimento di una lista di libri da file	B	
3	Operazione *CUD* su un libro	I	10×7 / mese
4	Lettura di un record di un libro	I	20×7 / giorno
5	Operazione *CUD* su un utente	I	8×7 / giorno
6	Lettura di un record di un utente	I	20×7 / giorno
7	Operazione *CUD* su un operatore	I	3×7 / anno
8	Lettura di un record di un operatore	I	10×7 / anno
9	Registrazione evento nel log	B	10×7 / giorno
10	Prenotazione libro da remoto	I	10×7 / giorno
11	Gestione prestito	I	20×7 / giorno

Tabella 7a – *Tavola delle operazioni*

Si noti come si presentano lineari gli accessi di ogni singola operazione. Ad esempio, le operazioni **2** e **3** constano solo di una scrittura di una voce *libro* e di una scrittura di una voce *log*:

Operazione	Concetto	Costrutto	Accessi	Tipo
1a	Plesso	E	1	R
1b	Plesso	E	1	W
	Log	R	1	W
2, 3	Libro	E	1	W
	Log	R	1	W
4	Libro	E	1	R
5	Utente	E	1	W
	Log	R	1	W
6	Utente	E	1	R
7	Operatore	E	1	W
	Log	R	1	W
8	Operatore	E	1	R
9	Plesso/Libro/ Utente/Operatore	E	1	R
	Log	R	1	W
10	Libro	E	1	R
	Utente	E	1	R
	Prestito	R	1	W
11	Prestito	R	1	R
	Prestito	R	1	W

Tabella 7b *– Tavola degli accessi*

Grazie alla tavola degli accessi in ***Tabella 7b*** è possibile svolgere un'analisi accurata delle prestazioni. Per valutare il singolo costo di operazione $C(O_T)$ bisogna eseguire la seguente operazione:

$$C(O_T) = f(O_T)\, w_P\, (\alpha C_W + C_R)$$

In particolare, il costo totale di ogni operazione $C(O_T)$ è data dal prodotto della frequenza della medesima $f(O_T)$, del suo peso w_P e della somma del numero di accessi in lettura C_R e in scrittura C_W. Poiché l'operazione di scrittura è più costosa di una in lettura, essa è moltiplicata per una costante moltiplicativa $\alpha > 1$.

Questo perché la scrittura nel database, per la maggiore, prevede una lettura intrinseca dei dati (ad esempio nell'aggiornamento o nella cancellazione dei dati). Invece, il peso dell'operazione w_P dipende dal suo tipo, poiché un'operazione interattiva I può avere un peso maggiore di un'operazione batch B.

Infine, per il costo complessivo delle operazioni si fa una semplice sommatoria

$$C_{tot} = \sum_{i=1}^{n} C_i(O_T)$$

dove con n si indica il numero totale di operazioni. Per il calcolo delle prestazioni in termini di tempo si stabilisce il valore dei singoli parametri che sono i seguenti:

- $w_I = 0.5$;
- $w_B = \frac{w_I}{2} = 0.25$;
- $a = 2$.

Tenendo conto della tavola degli accessi in ***Tabella 7b***, per ciascuna operazione si applica l'operazione definita in precedenza. La ***Tabella 7c*** riassume il calcolo di tutti gli accessi:

Operazione	$f(O_T)$	C_R	C_W	$C(O_T)$
1a	7 una tantum	1	—	**0**
1b		—	2	**0**
2		—	2	**0**
3	2,3 / giorno	—	2	**46**
4	140 / giorno	1	—	**70**
5	56 / giorno	—	2	**112**
6	140 / giorno	1	—	**70**
7	0,058 / giorno	—	2	**0,116**
8	0,19 / giorno	1	—	**0,095**
9	70 /giorno	1	1	**52,5**
10	70 / giorno	2	1	**140**
11	140 / giorno	1	1	**280**
Totale				**770,711**

***Tabella 7c** - Riepilogo dei costi delle operazioni principali*

Le operazioni **1** e **2** non vengono eseguite determinate volte nel tempo ma solo in un numero limitato, poi non vengono più effettuate. Per questo motivo non vengono considerate e si stimano uguali a 0. Invece, la frequenza delle operazioni **3**, **7** e **8** è calcolata rispetto a mesi o anni. Perciò si sono convertite le frequenze dividendo rispettivamente per 30 (nel caso dei mesi, operazione **3**) e 365 (per gli anni, operazioni **7** e **8**).

Questa fase di analisi conclude la progettazione logica dello schema del sistema bibliotecario. Tenendo conto di tutte le informazioni ricavate in questo capitolo è possibile procedere allo sviluppo vero e proprio della Web Application, di cui si tratterà in maniera molto approfondita nel capitolo successivo.

3. Sviluppo della Web Application

In questo capitolo si tratta la fase successiva alle progettazioni concettuale e logica, ovvero lo sviluppo della Web Application dell'Istituto "Toniolo": si trasformano elementi astratti in un'elaborazione di dati concreti.

Prima di poter andare nel dettaglio, si illustrano i linguaggi utilizzati nel corso dello sviluppo della Web Application. È un tipo di lavoro basato su due livelli: bisogna fare una distinzione fondamentale tra programmazione Front-end e Back-end:

- Il *Front-end* è la parte di programma in cui l'utente è in grado di interagire direttamente con la piattaforma, ovvero tutto ciò che è direttamente visibile. Per il Front-end sono stati utilizzati i linguaggi HTML, CSS e JavaScript:
    - *HTML* descrive il contenuto di un sito web;
    - *CSS* descrive il suo stile;
    - *JavaScript* (*JS*) si occupa dell'interattività dell'applicazione. In particolare, si è utilizzata la versione 3.6.0 della libreria *jQuery* in grado di semplificare la stesura del codice JS.

- Il *Back-end* è tutto ciò che non è direttamente visibile dall'utente, bensì processato in background. Questa parte è sempre presente quando di base vi è la manipolazione di dati forniti da un database. I linguaggi utilizzati sono JS e PHP:
    - Qui JS funge da tramite tra l'applicazione e il server. Con la tecnica *AJAX* è possibile uno scambio di richieste in maniera asincrona senza necessità di ricaricare la pagina al termine;
    - *PHP* processa le informazioni ottenute lato client per poter restituire un risultato dal database: è stata utilizzata la versione 7.2.34.

3.1. Descrizione dell'interfaccia

Il sito si compone di tre pagine principali, ossia quella di benvenuto *index.php*, quella dei risultati di ricerca *result.php* e la console di gestione del sistema bibliotecario *manageBoard.php*. Lo stile di tutti gli elementi del sito è definito dal file *style.css*, presente nella cartella *CSS*.

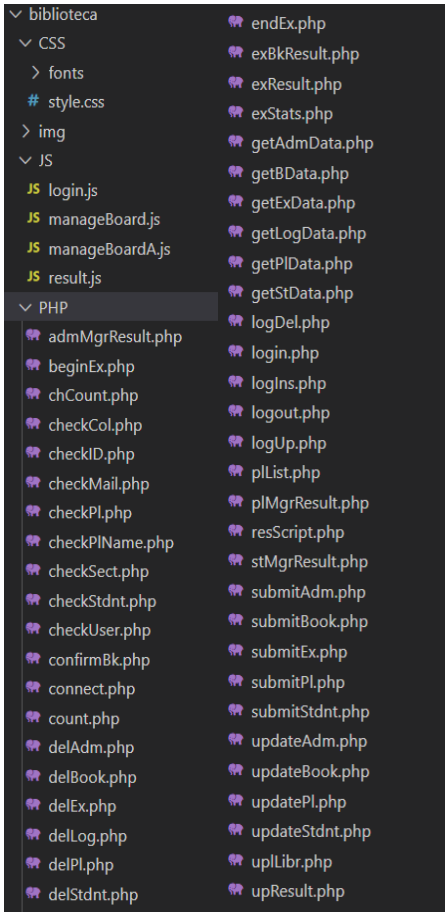

***Figura 7** - Contenuto della cartella del progetto considerando solo gli script creati*

Gli elementi interattivi sono gestiti dai file presenti nella cartella *JS*. È stato scritto un singolo file per ciascuna pagina, ad eccezione di *login.js* presente in tutte le pagine (poiché il login nel sito è possibile in ogni pagina) e di *manageBoardA.js*, chiamato solo quando un operatore di tipo gestore effettua l'accesso all'interno della piattaforma.
Nella cartella *img* vi sono alcune immagini che vengono utilizzate a livello grafico.

Infine, tutti gli script PHP nella cartella omonima, in totale cinquanta, sono di piccola dimensione (da 1 a 11 KB) e vengono chiamati soltanto quando vengono richiesti da una determinata chiamata AJAX, tranne *connect.php* perché è uno script particolare, di cui verrà trattato a breve.

In ***Figura 7*** si illustra l'elenco dei file che consentono il funzionamento dell'applicazione.

Tornando alla discussione delle pagine del sistema bibliotecario, tutte dispongono di un *header* che contiene i collegamenti principali. In alto a sinistra vi è il logo con il nome dell'istituto. A destra vi è il menu con cui è possibile navigare verso il sito della scuola e il *padlet* del progetto "*Biblioteca viva*", in cui la componente bibliotecaria dell'istituto pubblica regolarmente informazioni o nuovi progetti inerenti la lettura destinati ai propri alunni. Infine, vi è una voce riservata agli operatori del sistema bibliotecario in cui gli operatori possono effettuare l'accesso. Quando un operatore ha effettuato l'accesso, al posto del pulsante di login vengono visualizzati quello di rinvio alla console di gestione e quello che permette il logout.

In basso, la pagina dispone di un *footer* in cui vi sono tutti i contatti del plesso principale. L'interfaccia è stata progettata per una consultazione su dispositivi desktop, tablet e/o TV: questo significa che la visualizzazione in un dispositivo mobile (ad esempio uno smartphone) non è ottimale, ma sempre coerente con l'interfaccia. La ***Figura 8*** mostra l'header e il footer delle pagine:

Figura 8 *- Header e footer delle pagine del sistema bibliotecario*

Gli elementi appena descritti sono comuni a tutte le pagine dell'interfaccia. Si è pensato a uno stile che potrebbe rispecchiare quello di un sistema bibliotecario di un istituto scolastico, tenendo conto del target sia degli utenti a cui è destinato il servizio, sia di quello degli utenti che gestiscono la piattaforma. Per questo motivo lo stile è semplice e gli elementi interattivi di semplice intuizione, favorendo in questo modo l'usabilità da parte degli utenti.

3.2. Script di creazione del database

Finora si è discusso nelle linee generali sia della struttura dell'applicazione in termini di file, sia dell'impostazione dell'interfaccia principale.

Oltre alle pagine effettivamente visibili e agli script accennati nel § 3.1, tra questi ultimi ne esiste uno in particolare che funge da punto di riferimento all'intero sistema bibliotecario. Esso permette il collegamento del server al database e, qualora non esista, la creazione di esso. Si tratta dell'unico script presente in tutte le pagine della Web Application poiché permette l'interazione diretta con la base di dati definita nel capitolo precedente. Il file in questione è *connect.php* e si trova all'interno della cartella PHP.

Lo script si avvale di tre componenti fondamentali che lo caratterizzano, ovvero la creazione della connessione, la query di creazione del database e infine la creazione di un account di amministrazione di default.

All'inizio del file vengono dichiarate le variabili che contengono il nome del server `$servername`[7], le credenziali di accesso allo stesso `$username` e `$password` e il nome del database `$db_name`. Dato che si è lavorato in un ambiente locale, il nome del server è *localhost*, lo username è *root*, la password è vuota e il nome del database *biblioteca*. Quindi all'inizio di *connect.php* si dichiara quanto segue:

```
$servername = 'localhost';
$username = 'root';
$password = '';
$db_name = 'biblioteca';
```

Dopo tali dichiarazioni, il file tenta di creare una connessione con il server in uso. Così come in altri linguaggi di programmazione, in PHP le eccezioni si gestiscono tramite il costrutto `try/catch`.

[7] Nel linguaggio PHP il nome delle variabili è sempre preceduto dal carattere "$".

All'interno di `try` vengono immesse le istruzioni da "provare". In caso di rilevazione di un'eccezione viene chiamato il contenuto di `catch`, in cui solitamente viene gestita la stampa di messaggi di errore.

All'inizio del blocco `try` viene creata la connessione al server. Si dichiara una variabile `$db` che contiene una nuova connessione di tipo *PDO*[8] (*PHP Data Object*) con le credenziali riportate nella pagina precedente. Per una maggiore verifica di eventuali errori da parte del database, si permette la visualizzazione delle eccezioni.

In seguito alla creazione della connessione al server, viene eseguita la prima query del sistema bibliotecario, ovvero la creazione del database. Per motivi di compatibilità è molto importante sottolineare che l'istruzione utilizzata `CREATE DATABASE IF NON EXISTS` non è supportata in PHP 8. Infatti, in quest'ultimo caso verrebbe catturato un errore in cui la prima tabella esiste già (dunque il costrutto `IF NOT EXISTS` non è supportato).

Per creare il database bisogna tenere conto di tutte le considerazioni svolte nel capitolo precedente, in particolare del § 2.3.2. Per ogni tabella individuata all'interno del database viene eseguita un'istruzione `CREATE TABLE`, al cui interno si dichiarano tutti gli attributi, tutte le eventuali chiavi esterne e tutti i possibili vincoli.

[8] È stato adottato il modello PDO in quanto è maggiormente orientato sulla sicurezza dei dati, anche grazie all'introduzione dei *prepared statements*. Esistono altre API (Application Programming Interface) di PHP in grado di gestire e manipolare basi di dati, come ad esempio MySQLi, ma quest'ultima si concentra principalmente sulla velocità di esecuzione delle operazioni.

Ad esempio, nel caso della registrazione bibliografica di un libro si ottiene il seguente codice:

```
CREATE TABLE libro (
      ID INT AUTO_INCREMENT PRIMARY KEY,
      Titolo VARCHAR(60) NOT NULL,
      Autore VARCHAR(60) NOT NULL,
      Editore VARCHAR(60),
      Anno VARCHAR(4),
      Collocazione VARCHAR(6) NOT NULL,
      Plesso VARCHAR(10) DEFAULT '-',
      Stato CHAR DEFAULT 'D',

      UNIQUE(Collocazione, Plesso),

      FOREIGN KEY (Plesso)
            REFERENCES plesso(Codice)
            ON UPDATE CASCADE
            ON DELETE CASCADE
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Il codice sopra si descrive come segue:

- `ID` è un campo di tipo `INT`, un intero. È un valore che non ha bisogno di essere impostato in quanto funge da indice e quindi si incrementa autonomamente. Si tratta di una chiave primaria[4], ovvero identifica univocamente un libro della tabella.

- Tutti i campi che descrivono un libro sono di tipo `VARCHAR`, cioè una stringa di lunghezza variabile. Tra parentesi è riportato il numero massimo di caratteri consentito (ad esempio, il titolo, l'autore e l'editore non possono sforare i 60 caratteri). Gli attributi con la dicitura `NOT NULL` sono quelli ritenuti obbligatori, ovvero che non possono essere vuoti.

- Lo stato del libro viene interpretato come un carattere. Come default è impostato il valore `'D'`, ovvero che il libro è disponibile e quindi pronto al prestito. Gli altri stati sono `'T'`, libro in prestito e `'C'`, di sola consultazione.

- La coppia collocazione-plesso è sempre unica (`UNIQUE`) dato che in un plesso non possono esistere due record catalografici nella stessa collocazione.

- Il campo `Plesso` è una chiave esterna, il cui valore corrisponde a uno dei codici memorizzati nell'omonima tabella. Quando si aggiorna o si elimina il plesso di riferimento, le modifiche si effettuano automaticamente a cascata. All'aggiornamento/eliminazione di un plesso dalla console, tutte le entità e le relazioni associate ad esso vengono a sua volta aggiornati/eliminati.

- L'*engine* utilizzato è `InnoDB`. In particolare, un *engine* determina il modo in cui i dati vengono salvati nel database. Con `InnoDB` si indica un salvataggio di default.

- La codifica di caratteri `CHARSET` utilizzata è UTF-8, in cui un carattere può essere codificato anche da più di un byte. Questo permette una maggiore flessibilità e un maggiore supporto a più caratteri (come gli hiragana e i katakana giapponesi oppure i caratteri dell'alfabeto thai). La versione 8.0 del linguaggio SQL ha anche introdotto il supporto alle emoji sfruttando delle varianti della codifica UTF-8, ovvero `utf8mb3` e `utf8mb4`. Poiché sono caratteri ritenuti superflui si è ritenuto mantenere la codifica UTF-8.

Al termine della dichiarazione di tutto il DDL (Data Definition Language) del database, si esegue l'istruzione `$db->exec($database)`. In PDO, le istruzioni di esecuzione delle query maggiormente utilizzate sono `execute` ed `exec`. Quest'ultima si applica quando nella query non sono previsti valori di input, né di output (come verrà approfondito in seguito). In più le query chiamate con `exec` vengono eseguite una volta sola.

In seguito, si definisce cosa dovrebbe restituire il server in caso di errore all'interno del blocco `catch`, ovvero un messaggio di errore e la chiusura immediata della connessione. Fatte queste considerazioni si ottiene lo *snippet* di codice nel § 7.1 già presente nel file *connect.php*.

Si è preferito integrare la creazione del database all'interno di uno script PHP piuttosto che uno script esterno in formato .sql poiché in questo modo non si necessita l'importazione separata del database, rendendo così l'inizializzazione più immediata.

Alla creazione del database non si dispone di un modo per poter accedere al sistema bibliotecario per la prima volta: dunque è necessario creare un operatore di tipo gestore in modo automatico. Tuttavia, secondo quanto riportato in ***Figura 1*** - Gerarchia degli elementi principali del database della biblioteca, non è sufficiente creare un singolo gestore poiché prima di esso è necessario introdurre un plesso di riferimento. Il ragionamento da fare è il seguente: si controlla se esistono nel database operatori di tipo gestore:

- Se non sono presenti viene inserito un primo plesso di default;
- In seguito, si aggiunge un operatore di tipo gestore di default;
- Se esiste almeno un gestore l'operazione viene saltata.

Adesso si illustra nel dettaglio il ragionamento appena svolto. La query di controllo dell'esistenza di un gestore è la seguente:

```
SELECT COUNT(*)
FROM operatore
WHERE Tipo = 'G'
```

Qui nella tabella `operatore` si visualizza il numero (`COUNT(*)`) di quelli il cui tipo è proprio `'G'`, ovvero gestore. Se tale valore è uguale a zero, si inizializzano le informazioni del plesso di default tenendo conto dei formati stabiliti durante l'analisi dei requisiti; quindi, viene eseguita la query di inserimento del plesso.

Creato il plesso di default si inizializzano a sua volta le informazioni del plesso di default e la query associata viene eseguita.

In un qualsiasi script PHP le query vengono eseguite, tramite l'API PDO, nel seguente modo:

- Si inizializzano gli input se provenienti da un form, altrimenti vengono inizializzate delle variabili di default;
- Si inizializza una stringa con la query di `SELECT`, `UPDATE` o `DELETE`;
- Si prepara la query con il database in uso tramite l'istruzione `prepare()`;
- Si effettua il binding dei parametri di input, ovvero si dichiara il contenuto di essi con la funzione `bindParam()`. Nota che si utilizza solamente se nella query vi sono degli input dichiarati sotto dei *prepared statements*;
- Si esegue la query con `execute()`. Nel caso di query di selezione (`SELECT`), dopo tale istruzione si può operare con il risultato proiettato tramite istruzioni *ad hoc* (`fetchAll()`, `fetchColumn()`, `rowCount()`, …).

Dunque, seguendo l'algoritmo sovrastante, se non vi sono gestori si inizializzano le variabili il cui contenuto è obbligatorio:

```
$pl = "PAXX00000X";
$pType = "2";
$pName = "Centrale";
```

Dopo si scrive la query che inserisce un plesso nel database:

```
$sql = "INSERT INTO plesso VALUES(:pl, :pType, :pName, '-', '-
', '-')";
```

Nota che le variabili che iniziano con ":" fungono da segnaposto per il binding successivo.

A questo punto si procede con la preparazione della query, il binding dei parametri e l'esecuzione della query:

```
$stmt = $db->prepare($sql);
$stmt->bindParam(':pl', $pl);
$stmt->bindParam(':pType', $pType);
$stmt->bindParam(':pName', $pName);
$stmt->execute();
```

Al termine di queste istruzioni la query è conclusa e il plesso di default è stato correttamente inserito.

In maniera pressoché simile, subito dopo si inserisce un operatore di default di tipo gestore. Qui, tuttavia, è bene introdurre una funzione molto importante che presta attenzione alla sicurezza dei dati.

Nell'ambito dello *storing* dei dati personali è altamente sconsigliato salvare una password in un database così com'è: bisogna criptare la password con una opportuna funzione. Nel caso preso in esame è stato adottato il costrutto `password_hash()` che, prendendo in input il valore della password e il tipo di crittografia, trasforma una stringa nella sua chiave in base al tipo di crittografia scelto: qui è stata utilizzata la chiave `BCRYPT`, che genera una stringa di 60 caratteri con la stringa da codificare. Dunque, la password verrà salvata in questo modo:

```
password_hash($pass, PASSWORD_BCRYPT)
```

Dopo aver descritto come è strutturato lo script che consente la creazione e l'inizializzazione del database, nel paragrafo successivo viene illustrato nel dettaglio il funzionamento vero e proprio di tutte le componenti.

3.3. Funzionamento della Web Application

In questo paragrafo viene descritto in maniera approfondita il funzionamento di tutte le componenti del sistema bibliotecario, sia in lato Front-end che in Back-end[9].

3.3.1. Login/logout

Per poter accedere al sistema, l'operatore clicca sulla voce del menu apposita. Di seguito compare un piccolo form in cui egli immette le proprie credenziali di accesso: in caso di esito positivo, viene visualizzato un messaggio di benvenuto all'operatore e quest'ultimo viene rimandato direttamente alla console di gestione (di cui si tratterà nel § 3.3.2). In maniera speculare, cliccando sull'opzione di logout, l'admin è in grado di uscire dalla piattaforma e quindi ritrovarsi nella pagina principale. In ***Figura 9*** si illustra la finestra di login del sistema.

Figura 9 *- Finestra di login*

I meccanismi di login e di logout sono gestiti tramite lo script *login.js*, presente all'interno della cartella JS. In jQuery, la sintassi di un generico evento è la seguente:

```
$("#selettore").evento(function(){…});
```

Rispetto alla sintassi JavaScript, jQuery (§ 3.1) è in grado di svolgere le stesse funzioni ma sfruttando meno righe di codice e quindi rendere lo sviluppo più rapido lato client.

[9] Le definizioni di *Front-end* e di *Back-end* sono illustrate nel § 3.1.

Vengono dichiarati gli elementi HTML con la stessa sintassi CSS: ciò migliora anche la comprensione del codice che viene scritto. Inoltre, è importante nell'ambito di AJAX in quanto semplifica l'interazione asincrona con il database.

Solitamente non vengono dichiarati parametri formali[10] all'interno delle funzioni. Tuttavia, quando si gestiscono gli eventi di tipo *click* e/o *submit*, è necessario gestire l'evento al fine di bypassare il comportamento di default del browser.

La maggior parte delle funzioni jQuery scritte per questo sistema hanno la seguente struttura:

- Prevenzione del comportamento di default, se esistente;
- Definizione delle variabili;
- Definizione di eventuali *pattern*[11] di controllo;
- Controllo degli input con annesso risultato dell'operazione.

Dunque, al click della voce di login la prima operazione svolta è la prevenzione dell'evento di default. Subito dopo, all'interno del box `.loginBox`[12], viene generato il form di login. Successivamente si definiscono tutti gli eventi associati agli elementi HTML. Ad esempio, si prenda in considerazione l'evento associato al pulsante di tentativo di login `#loginAtt`[12].

[10] In programmazione i *parametri formali* sono quelle variabili che vengono utilizzate all'interno di una funzione. I *parametri attuali* sostituiscono quelli formali al momento della chiamata della funzione stessa.

[11] Per *pattern* si intendono le cosiddette *espressioni regolari*. Sono delle stringhe che permettono il controllo di una determinata stringa affinché soddisfi determinate condizioni strutturali (ad esempio i caratteri consentiti).

[12] Nel linguaggio CSS le classi vengono definite con il nome della classe preposto dal carattere ".", mentre gli identificatori con "#". La differenza sostanziale tra ID e classi è che i primi sono univoci all'interno della pagina o di un box, mentre le classi possono ripetersi. Quest'ultime sono utili nel definire un *set* di elementi HTML con le medesime proprietà.

Poiché si sta gestendo un evento di tipo click si previene l'azione di default tramite `e.preventDefault()`. Successivamente si dichiarano i parametri di riferimento: in questa funzione entrano in gioco `id` che contiene il valore del campo Username, `pw` con la password e infine due pattern che controllano il formato dei due parametri. Si prenda in considerazione la variabile contenente il pattern di controllo dello username `uPtrn` e se ne analizzi il contenuto:

```
var uPtrn = /^([A-Za-z0-9]{6,20})$/;
```

A differenza delle semplici stringhe, in JavaScript le espressioni regolari sono poste tra due caratteri "/" a differenza delle virgolette "‟". A sua volta, agli estremi della *regex*[11], i caratteri "^" e "$" indicano rispettivamente l'inizio e la fine di una stringa o di un blocco a inizio/fine di una riga. All'interno si legge una parentesi tonda che contiene un insieme di singoli caratteri e il numero di questi. Dunque, la parentesi quadra `[A-Za-z-0-9]` indica un singolo carattere dell'alfabeto maiuscolo o minuscolo, oppure una cifra da 0 a 9. Invece la graffa immediatamente accanto {6,20} indica che il carattere precedente può ripetersi da 6 a 20 volte. Quindi, `uPtrn` controlla se una stringa possiede da 6 a 20 caratteri alfanumerici.

Dichiarate le variabili si svolge un loro primo controllo di esistenza: Se almeno uno dei campi è vuoto, lo script restituisce un messaggio di errore. In maniera pressoché simile si comportano i due controlli successivi, in cui vi è il controllo degli input in ingresso attraverso i pattern specifici per via del metodo `nomePattern.test(input)`. Infine, se nessuno di questi controlli è soddisfatto (poiché si verifica la condizione opposta, ovvero il non match), viene invocata una chiamata asincrona AJAX al database di tipo POST[13].

[13] Esistono diversi metodi di richiesta nel Web Programming: quelli principali sono GET e POST. La differenza tra i due è che mentre con il metodo GET le informazioni sono reperibili all'interno dell'URL, con POST offre maggiore discrezione e riduce sensibilmente la tracciabilità degli input. Quindi GET è ideale nell'implementazione di sistemi di ricerca, mentre POST nella manipolazione dei dati nel database.

Ci sono diversi modi per poter invocare una funzione di tipo AJAX. In questo sistema bibliotecario ne sono stati applicati due, in base alla manipolazione o meno di file (di questo verrà trattato nel § 3.3.2.2). La sintassi di quella più utilizzata è illustrata qui di seguito:

```
$.metodo("script.php", {dati in ingresso}, function(data){…},
"formato output");
```

Dopo aver dichiarato il metodo della chiamata AJAX (GET/POST) se ne definiscono i parametri fondamentali all'interno di essa: all'inizio si specifica lo script PHP di rimando in cui vengono processati i dati, quest'ultimi sotto forma di dizionario. In seguito, si indica la funzione che definisce l'esito dell'operazione al variare degli input e infine, opzionalmente, il tipo di dato in output (solitamente JSON[14]). La chiamata AJAX utilizzata nel login è la seguente:

```
$.post('PHP/login.php', {user: id, pass: pw}, function(data) {
      $("#loginStatus").html(data);
});
```

A questo punto si descrive lo sviluppo Back-end. In quasi tutti gli script PHP che interagiscono con l'interfaccia, le prime istruzioni sono l'avvio della sessione[15] per mezzo di `session_start()` e l'inclusione del file che inizializza la connessione (§ 3.2). Dopodiché si recuperano le variabili di input presenti all'interno del dizionario dichiarato nella chiamata AJAX, tenendo conto che l'argomento della variabile superglobale `$_POST` corrisponde alla chiave all'interno del dizionario. A queste viene applicata un'operazione di `trim()` che rimuove eventuali spazi vuoti all'inizio e alla fine della stringa.

[14] JSON sta per *JavaScript Object Notation* ed è un formato di interscambio di dati tra i più utilizzati nell'interazione client-server.

[15] Una *sessione* è un metodo di salvataggio di informazioni che possono essere utilizzate in più pagine. Essa si autodistrugge alla chiusura del browser. A differenza delle sessioni, i *cookie* sono dei dati che vengono salvati su disco per poter essere riutilizzati in seguito. Qui non ne vengono utilizzati.

In seguito, verificata l'esistenza degli input trasmessi, viene eseguita una query che controlla l'esistenza di quell'operatore. In caso di non esistenza viene restituito un messaggio di errore che poi viene visualizzato come risposta da AJAX. Altrimenti viene eseguita una *fetch*, un recupero, delle informazioni ottenute: in particolare è necessaria per poter recuperare la password dell'operatore che sta entrando nella console.

Tale controllo viene eseguito tramite la funzione speculare a `password_hash()`, ovvero `password_verify()`. Tale funzione trova l'hash della stringa password di input e la confronta con quella memorizzata nel database: se il metodo restituisce `true`, allora vengono inizializzate le variabili di sessione `$_SESSION` in cui si memorizzano lo username dell'operatore, il suo tipo (admin o gestore) e il plesso di riferimento. Fatto questo, lo script restituisce un messaggio di successo e infine, tramite un breve script JS, il *redirect* alla console di gestione, di cui si tratterà ampiamente il funzionamento nel paragrafo successivo.

Qualora almeno uno dei campi non fosse corretto, lo script invierà un messaggio di errore e lo visualizza.

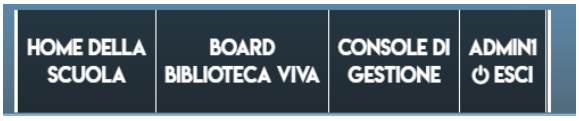

Figura 10 *– Il menu con pulsante di logout*

Per quanto riguarda il logout, a livello interattivo la procedura è molto simile. Al click del pulsante di logout (***Figura 10***), viene chiamato lo script *logout.php*: esso subito dopo l'avvio della sessione distrugge tutte le variabili `$_SESSION` e chiude immediatamente lo script. Al termine, l'operatore viene rimandato alla pagina *index.php*.

Dopo aver ampiamente descritto i meccanismi di login e logout all'interno del sistema bibliotecario, ci si accinge a trattare il funzionamento della sua console di gestione.

3.3.2. Console di gestione

La console di gestione rappresenta il cuore di tutta la Web Application poiché permette la gestione di tutti i dati all'interno del database della biblioteca. È una sezione accessibile solamente agli operatori che hanno effettuato correttamente l'accesso (§ 3.3.1).

Come specificato nel § 2.1, esistono due tipi di operatore che usufruiscono della console, ovvero l'admin semplice (che per praticità verrà chiamato admin) e il gestore: gli admin possono solo registrare l'inserimento manuale della voce di un libro, della registrazione di un utente e della gestione dei prestiti. Oltre a queste azioni, un gestore può amministrare gli operatori della piattaforma e i plessi dell'istituto scolastico.

Figura 11 *- Menu accessibile da un admin e quello visualizzabile da un gestore*

La pagina *manageBoard.php* è relativa proprio alla console di gestione. Se un visitatore accede alla pagina, viene visualizzato un messaggio in cui avvisa che bisogna effettuare l'accesso per poter entrare nella console. In caso un admin effettui l'accesso, vedrà una schermata pressoché simile, ma con il menu della console a lato. Se invece è un gestore ad accedere visualizzerà delle schede aggiuntive oltre a quelle degli admin. La ***Figura 11*** mostra il menu visualizzato dagli admin e quello dai gestori.

Queste diverse visualizzazioni sono gestite grazie all'esistenza e del contenuto della variabile `$_SESSION["type"]` all'interno della manipolazione PHP in *manageBoard.php*. Infatti, se tale variabile è inizializzata con il carattere "`A`" si ricade nella situazione a sinistra in ***Figura 11***. Altrimenti, se il tipo è uguale a "`G`", il menu visualizzato sarà quello a destra. Infine, se `$_SESSION["type"]` non è definita, logicamente non verrà visualizzato.

Dunque, le variabili di sessione si dimostrano fondamentali nella gestione degli utenti che si occupano del sistema bibliotecario, diversificando le azioni che possono compiere in base al tipo di utente registrato nel database. Bisogna sempre tenere conto delle tabelle ottenute durante tutte le progettazioni concettuale e logica (in particolare i risultati nel § 2.3.1).

A ciascun pulsante del menu corrisponde una differente sezione del sistema, in cui gli operatori possono gestire le entità presenti all'interno del database. Più precisamente ogni voce mostra due schede: in una è possibile la gestione di un elemento all'interno del sistema bibliotecario, mentre nell'altra l'inserimento di una voce.

La maggior parte delle animazioni, tutte le schede generate e l'interazione con esse si trovano all'interno di uno script apposito, ovvero *manageBoard.js*, presente all'interno della cartella *JS*. Per una maggiore flessibilità e sicurezza si è preferito non includere anche le schede riservate ai gestori nello stesso script.

Infatti, quando un gestore effettua l'accesso, viene caricato anche lo script *manageBoardA.js*, che contiene le funzionalità aggiuntive riservate a questa tipologia di operatori.

All'interno della pagina sono nascosti molti box vuoti che in base all'interazione desiderata ne viene alterato il contenuto. Sicuramente ci saranno stati molti modi più rapidi ed efficaci: tuttavia, poiché si tratta di un primo tentativo di sviluppo di

questo calibro, l'impostazione risulta pressoché sufficiente e performante. Con dei futuri aggiornamenti sarà possibile migliorare ulteriormente l'interfaccia puntando sulla sua efficienza.

Delineata l'interfaccia principale della console, si mostra la gestione delle singole entità presenti all'interno del sistema bibliotecario a cominciare dalla gestione dei plessi, cardine di tutto il database.

3.3.2.1. Gestione dei plessi

Come schematizzato in ***Figura 6***, il plesso rappresenta l'entità di riferimento all'interno di tutto il sistema bibliotecario poiché in ciascuno di essi sono incluse le altre, ovvero i libri, gli utenti e gli operatori. Questo frammento di console è gestito dallo script *manageBoardA.js*, accessibile solamente agli operatori di tipo gestore.

Figura 12 *- Box di gestione dei plessi*

Nel § 3.3.2 è stato illustrato il contenuto di ciascuna sezione all'interno della console: poiché gli algoritmi sono pressoché gli stessi, in questo sotto paragrafo si riporta nel dettaglio il procedimento; negli altri invece verranno trattati eventualmente solo gli elementi più distintivi. Si cominci a descrivere la scheda di inserimento di un plesso.

La ***Figura 12*** - Box di gestione dei plessi illustra un generico menu di gestione di un elemento all'interno del sistema bibliotecario; in questo caso si mostra la gestione del plesso. Dunque, al click del pulsante "*Plessi*" presente all'interno del menu, si mostra il contenuto in ***Figura 12***: inizialmente vuoto, dentro il box *#plSection*[12] vengono mostrati i pulsanti di visualizzazione e di inserimento.

Cliccando sul pulsante "*Aggiungi plesso*" viene mostrato *#addPBox* contenente il form di inserimento del plesso (***Figura 13***). Esso comprende tutti i campi da riempire con tutte le informazioni dell'entità: per semplificare il lavoro alcuni di essi, come il codice, l'indirizzo e il recapito telefonico sono già parzialmente precompilati di default. Il pulsante "*Reset*" permette la cancellazione dei dati del form per una nuova compilazione. Invece, il pulsante di invio del form è disabilitato poiché la sua attivazione richiede come condizione la validità di tutti i valori inseriti. Il meccanismo di gestione della validità verrà discussa a breve.

Durante la digitazione del codice del plesso, ovvero all'evento jQuery *keyup()*, lo script *checkPl.php* chiamato tramite AJAX controlla in tempo reale se quel codice è presente all'interno del database. Qualora fosse già esistente lo script restituisce un messaggio di errore, altrimenti notifica il successo.

Figura 13 *- Form di inserimento di un plesso*

È stato applicato similmente il meccanismo già descritto nel login (§ 3.3.1). Viene catturato il contenuto dell'input e dichiarato il pattern di riferimento: se il codice esiste e la stringa fa *match* con l'espressione regolare

```
var ptrn = /^PI[A-Z]{2}\d{5}[A-Z]{1}$/;
```

allora viene chiamato via AJAX *checkPl.php*.

Questo script prende in ingresso il contenuto dell'input e diventa argomento della query

```
SELECT COUNT(*)
FROM plesso
WHERE Codice = :code
```

Essa restituisce un intero che verifica l'esistenza di quel codice all'interno della tabella. Infatti, se `$check == 0`, non esistono plessi con quel codice specifico e quindi è disponibile; altrimenti per `$check > 0` (in questo caso può assumere solo un valore tra 0 e 1 in quanto chiave primaria) restituisce un errore.

Gli errori presenti in un form sono gestiti tramite jQuery. A ogni form è associato un dizionario in cui una chiave rappresenta un campo, mentre il valore è un booleano (`true` o `false`) che indica se il valore immesso è conforme a quanto richiesto. In questo caso, prima del form è stato definito il seguente dizionario:

```
var plOk = {"code": false, "name": false, "type": true,
"addr": true, "ph": true, "mail": true};
```

Tutti i campi che richiedono un valore obbligatorio di default sono impostati a `false` mentre quelli facoltativi, oppure i campi di tipo `<select>` (le opzioni a tendina), a `true`. Quando un campo soddisfa le condizioni che lo caratterizzano, allora il suo valore corrispondente viene settato a `true`, altrimenti è `false`.

Questo dizionario è un modo efficiente per poter controllare in un colpo solo la validità di un form. Infatti, il sistema permette l'invio solo quando tutti i valori all'interno del dizionario sono `true`. Per controllare ciò è stata scritta una funzione che, prendendo in input un dizionario, ne esamina tutti i valori. Se almeno uno di essi è `false` restituisce `false`, altrimenti dà `true`.

In quest'ultimo caso il pulsante di inserimento a fine form si abilita. La funzione è la seguente:

```
function allTrue(ob) {
    for (var x in ob) {
        if (!ob[x]) {
            return false;
        }
    }
    return true;
}
```

Verificata la validità di tutti gli inserimenti si procede al tentativo di inserimento lato server. Nel caso dei plessi, viene chiamato tramite AJAX il file *submitPl.php*. Tramite il metodo jQuery `serialize()` applicato al form, vengono catturati automaticamente tutti gli ingressi e poi salvati come dizionario all'interno della variabile `input`. Le chiavi di quest'ultimo hanno come nome l'ID dei campi del form (ovvero il contenuto dell'attributo `id`).

A questo punto si procede con la chiamata lato server in PHP. Dopo l'inizializzazione delle variabili e il controllo dell'esistenza della variabile di sessione dell'operatore, si procede alla verifica nel database del codice inserito. Se esso esiste, allora viene restituito un messaggio di errore. In seguito, per riempire i campi *non* obbligatori, si assegna a ciascuno di essi il carattere "–" per indicare proprio un campo vuoto.

Dopodiché, se il codice non esiste e rispetta il formato stabilito (§ 2.1) si procede all'inserimento del plesso nel database. La query che viene eseguita è la seguente:

```
$sql = "INSERT INTO plesso VALUES(:code, :ty, :name,
:addr, :ph, :mail)";
```

Dopo il binding dei parametri di ingresso (§ 3.2) viene effettuato un secondo controllo degli input, questa volta lato server: se tutte le condizioni sono rispettate allora si procede all'inserimento effettivo nel database. In questo caso, se almeno una condizione è vera, allora viene restituito il codice `-1` (che qui indica un errore), che permette la visualizzazione a video dell'esito dell'operazione ed esce dallo script. Se invece la condizione dell'istruzione `if` non è verificata, si inserisce il record nel database.

***Figura 14** - Esito di successo nell'inserimento di un plesso*

A questo punto bisogna registrare l'inserimento del plesso all'interno della tabella *log*. In maniera pressoché simile viene inserita questa voce all'interno della tabella *log*:

```
$query = "INSERT INTO log VALUES(NULL, CURDATE(),
CURTIME(), 'I', 'P', :code, :u)";
```

Il primo campo è `NULL` poiché l'ID di una voce nel log è `AUTO_INCREMENT` e quindi non è necessario immettere dei valori. `CURDATE()` e `CURTIME()` memorizzano rispettivamente la data e l'ora correnti, mentre i caratteri '`I`' e '`P`' indicano l'azione di inserimento e tipo di entità, il Plesso.

Se anche questa query ha successo, il processo è andato a buon fine e quindi è possibile uscire dallo script. Altrimenti viene visualizzato un messaggio di errore. In ogni caso, poiché si lavora con AJAX e quindi chiamate asincrone di script PHP, non è necessario un *refresh* della pagina al termine in quanto le modifiche al database sono in tempo reale. La ***Figura 14*** mostra un caso di successo di inserimento di un plesso all'interno del sistema bibliotecario.

Dopo l'inserimento di un elemento all'interno del sistema, può sorgere la necessità di poter modificare o eliminare quell'elemento. Magari ci si accorge di errori di battitura, oppure ci si rende conto che quell'elemento non è più necessario e quindi bisogna rimuoverlo. Tutte queste funzioni, nel caso dei plessi, sono gestite dalla voce "*Elenco plessi*" (***Figura 12***). Al click del pulsante viene visualizzato l'elenco dei plessi registrati in ordine alfabetico. Si mostra un esempio di elenco in ***Figura 15***:

CODICE	NOME	TIPO	INDIRIZZO	TELEFONO	EMAIL	AZIONI
PIIC83400P	CENTRALE	SECONDARIA	-	-	-	
PIAA23989E	CENTRALE2	PRIMARIA	*Via dei Tigli 5*	-	-	

Figura 15 *- Esempio di tabella dei plessi*

Ciascuna voce contiene tutte le informazioni del plesso di riferimento. In fondo è stata inserita una colonna *Azioni* in cui si possono controllare le operazioni di modifica e cancellazione di un plesso dal database. Nel caso di tabelle molto ricche di informazioni, ovvero i libri, gli utenti e i prestiti, si dispone di un sistema di ricerca integrato con cui si possono cercare direttamente le risorse bibliografiche interessate senza dover scorrere la tabella all'infinito (§ 3.3.2.2). Tutte le tabelle sono generate da degli script PHP appositi che consentono la visualizzazione dei dati desiderati.

Il primo pulsante nella colonna *Azioni* permette l'aggiornamento dei dati del plesso da modificare. Ciascuno di questi pulsanti genera un *modal*[16] che permette di procedere con l'operazione scelta. Il sistema riconosce quale plesso sta per essere manipolato perché in ciascuna riga del corpo HTML della tabella sono stati inseriti i codici delle chiavi all'interno dell'ID di una riga della tabella.

[16] *Modal* è un modo alternativo per indicare una finestra di dialogo con cui l'utente può eseguire una determinata operazione.

In questo esempio la riga del plesso *Centrale2* viene rappresentata nel seguente modo:

```
<tr class="rRow tr3" id="rPIAA23989E">
   <td class="rCol rCode lemon">PIAA23989E</td>
   <td class="rCol rName lemon">Centrale2</td>
   <td class="rCol rType lemon">Primaria</td>
   <td class="rCol rAddr courgette">Via dei Tigli 5</td>
   <td class="rCol rPh lemon">-</td>
   <td class="rCol rMail courgette">-</td>
   <td class="rCol lemon"><img class="mIcon upIcon"
id="upPIAA23989E" src="img/up.png" alt="upPIAA23989E"
width="24px"></td>
   <td class="rCol lemon"><img class="mIcon delIcon"
id="delPIAA23989E" src="img/del.png" alt="delPIAA23989E"
width="24px"></td>
   </tr>
```

Da notare come gli ID della riga `<tr>` e quelli corrispondenti ai pulsanti di azione (le ultime due colonne `<td>`) comprendono il codice del plesso da manipolare, qui evidenziato in grassetto. In questo modo risulta più semplice reperire la chiave e quindi tutte le informazioni relative a quell'entità.

Cliccando sul pulsante di modifica del plesso viene generato il modal illustrato in ***Figura 16***:

Figura 16 *- Modal di aggiornamento del plesso*

Si visualizzano le informazioni del plesso attualmente salvate e in basso un form precompilato con tali informazioni eventualmente da aggiornare.

Nell'esempio in ***Figura 16*** sono state modificate alcune informazioni: sono stati adottati esattamente gli stessi metodi utilizzati nella fase di inserimento del plesso, in quanto il procedimento è per l'appunto il medesimo.

La *submit* dell'aggiornamento del plesso viene gestita dallo script *updatePl.php*. Dopo aver inizializzato i nuovi valori viene eseguita una query che controlla l'esistenza di quel codice plesso, a condizione che il nuovo sia diverso da quello vecchio. Se presente nel database allora è possibile modificare quella voce:

```
$check = "SELECT COUNT(*)
      FROM plesso
      WHERE (Codice = :code
      AND :nCode <> :code)";
```

Se il codice esiste si inizializzano i nuovi valori. Verificate tutte le condizioni specifiche degli attributi del plesso, si "riempiono" i campi vuoti con un "-" e in seguito viene eseguita la query di aggiornamento:

```
$query = "UPDATE plesso
      SET Codice = :nCode, Tipo = :nTy, Nome = :nName,
            Indirizzo = :nAddr, Telefono = :nPh, Email =
            :nMail
      WHERE Codice = :code";
```

Come per l'inserimento, se non ci sono problemi di aggiornamento del plesso allora viene registrata la modifica nel *log*: l'unica differenza è che nel campo "*Azione*" il carattere è "`U`" per *Update*. Al termine dello script si visualizzerà il plesso con gli attributi scelti modificati.

In caso di modifica di una chiave, tutte quelle relative ai plessi vengono automaticamente aggiornate, anche perché in quanto chiave esterna. Questa è una caratteristica del meccanismo di `CASCADE` che in questo sistema si applica per le *Update* e le *Delete* (§ 7.1).

Il *modal* di cancellazione di un plesso dal database ha una struttura più semplice di quello di aggiornamento: esso consente la rimozione di un plesso e di tutte le entità, ovvero tutti i libri, gli utenti, i prestiti e gli operatori ad esso registrati. La ***Figura 17*** mostra un esempio di *modal* di cancellazione di un plesso.

La finestra mostra il plesso da cancellare e subito in basso un *warning* che avvisa l'operatore che eliminando il plesso tutti i dati relativi ad esso vengono a sua volta eliminati. Cliccando sull'eliminazione effettiva viene eseguito lo script *delPl.php*. Prende in input il codice del plesso e il gestore collegato e quindi semplicemente si esegue la query di cancellazione

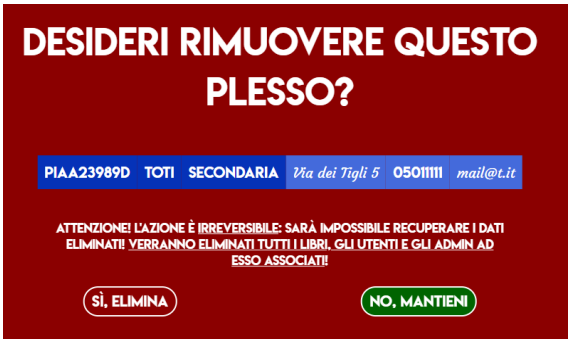

Figura 17 *- Modal di cancellazione del plesso*

```
$query = "DELETE FROM plesso WHERE Codice = :code";
```

e infine si registra l'avvenuta cancellazione nel *log*. In questo caso il campo "*Azione*" della tabella *log* conterrà il carattere "`D`" che sta per *Delete*.

Dopo aver illustrato nel dettaglio la gestione dei plessi si procede nell'analisi di quella dei libri nel database. Poiché diversi metodi sono simili se non uguali tra loro dal prossimo paragrafo, salvo casi particolari, essi non verranno approfonditi.

3.3.2.2. Gestione dei libri

Per la descrizione della gestione delle altre entità presenti all'interno del sistema bibliotecario si prenda come modello quello della gestione dei plessi nel § 3.3.2.1.

In questo sistema bibliotecario sono stati implementati due metodi di inserimento di record bibliografici all'interno del catalogo: un metodo *manuale* con cui è possibile aggiungere un libro così come con i plessi e uno *automatico*, riservato ai gestori, con il quale si possono caricare liste intere di libri attraverso il caricamento di file esterni senza la necessità di battere a mano ogni singola voce.

Cliccando sul menu *Libri* e successivamente su "*Aggiungi libro*" si presenta il form di inserimento. Non è presente un campo ID in quanto è calcolato automaticamente: nello script di inserimento *submitBook.php* viene eseguita una query per il calcolo dell'ID, in questo modo viene inserito il primo ID possibile. Nel form vi sono tutti i campi compilabili con i valori desiderati; in fondo è disponibile un pulsante "*Sola consultazione*": esso indica che quel libro non è disponibile per il prestito ma solo per la consultazione nella biblioteca di quel plesso.

In questo form bisogna tener conto che la collocazione del libro è unica in ogni plesso e deve essere del formato *X/0(y)*[17], mentre l'anno di pubblicazione è compreso tra il 1900 e l'anno corrente.

[17] Con *X* ed *y* si indicano delle lettere dell'alfabeto rispettivamente maiuscole e minuscole; con *0* invece un numero. Nota che *y* è facoltativo.

Il controllo della collocazione libera all'interno del plesso è verificato per mezzo della query

```
$sql_col = "SELECT COUNT(*)
            FROM libro
            WHERE (Collocazione = :col
                  AND Plesso = :pl)";
```

inclusa nel file *checkCol.php*.

Dopo la corretta compilazione del form si procede all'inserimento della voce all'interno del database. Viene eseguito lo script *submitBook.php*: al momento dell'inizializzazione, come stato di default è impostato quello di Disponibile al prestito "`D`". Tuttavia, in caso di sola consultazione viene assegnato lo stato apposito "`C`". In seguito, si procede al controllo degli input. Prima si esegue un controllo sulla validità della collocazione sfruttando la query poco sopra; poi viene calcolato il primo ID disponibile all'interno dell'elenco tramite la query

```
$sql = "SELECT MIN(L1.ID) + 1
      FROM libro L1
            LEFT OUTER JOIN libro L2
            ON L2.ID = L1.ID + 1
      WHERE L2.ID IS NULL";
```

Dopodiché, esattamente come con i plessi, si inizializza la query di inserimento, si controlla il formato di tutti gli input e, in caso di successo, si aggiunge il record nel database e infine si registra nel *log* (come nel § 3.3.2.1).

L’inserimento automatico dei libri da un elenco all’interno del database si dimostra uno strumento molto utile per velocizzare il lavoro di popolazione del catalogo dell’istituto. Questa funzione di carattere avanzato è utilizzabile solo dagli operatori di grado gestore ed è disponibile nel pannello Strumenti della console (***Figura 18***).

Figura 18 *- Sezione Strumenti e pannello di Upload libri*

Al momento dell’upload, il gestore seleziona il plesso a cui caricare l’elenco. In seguito, cliccando sul pulsante “*Carica file*”, si seleziona l’elenco da aggiungere al database. È molto importante sottolineare che:

- Il file deve essere in formato CSV[18];
- Non deve contenere intestazioni particolari, ma i nomi delle colonne da occupare (come stabilito nel § 3.2);
- I record vengono aggiunti solo se tutti i formati sono rispettati (saltano le righe in cui la collocazione non è corretta ad esempio).

[18] CSV sta per *Comma Separated Values* ed è un semplice sistema di rappresentazione di dati sotto forma di testo. Si chiama in questo modo perché le colonne dei record sono interrotte proprio da una virgola.

	A	B	C	D	E
1	Titolo	Autore	Editore	Anno	Collocazione
2	Zanna Bianca	London Jack	Del Drago	1989	A/67b
3	Incompreso	Montogonery Florence	Del Drago	1990	A/138
4	Il giovane Holden	Salinger J. D.	Einaudi	2014	C/178
5	Tom Jones Storia di un trovatello	Fielding Henry	L'Unità/Garzanti	1997	C/129
6	Il ragazzo e la tempesta	Ferrara Antonio	Rizzoli	2014	I /202
7	Il volo dell'asso di picche	Christian Hill	Einaudi ragazzi	2014	R/206
8	Piccolo mondo antico	Fogazzaro Antonio	Bur Rizzoli	2011	B/87
9	Enciclopedia degli asini, dei ribelli e di altri beni	Pouy Jean-Bernard/	Rizzoli	2011	T/41
10	Ritorno alla casa saracena	Merenda Adriana	Piemme	2008	R/205
11	I giardini degli altri	Barone Marta	Rizzoli	2011	D/61
12	Fratelli	Moeyaert Bart	Rizzoli	2011	I /203
13	Manuale dell'uomo normale	Beppe Severgnini	Bur	2008	T/37
14	Coraline	Gaiman Neil	Mondadori	2013	D/62
15	Sono troppo un genio del male	Lieb Josh	Rizzoli	2010	I/205
16	Che pasticcio, Clementina!	Bernardi Lorenza	EL	2013	R/207
17	I 10 mesi che mi hanno cambiato	Sonnenblick Jordan	Giunti	2013	I/206
18	La vita	John Boyne	Bur	2007	L/68
19	Il bambino con il pigiama a righe				
20	L'estate di Giacomo	Randazzo Luca	Rizzoli	2019	L73
21	La stagione delle bombe	Judith Kerr	Rizzoli	2010	L/67
22	L'eroe invisibile	L. Cognolato-S. Del Fran	Einaudi	2014	L/69
23	Il fantasma di Canterville	Oscar Wilde	Einaudi	2014	H/51
24	Ultima fermata Auschwitz	Sessi Frediano	Einaudi	1995	L/70

Figura 19 *- Esempio di file .csv con un elenco di libri della biblioteca di un plesso. In verde sono evidenziate alcuni record corretti, in rosso altri non corretti.*

La ***Figura 19*** mostra un esempio di file esterno caricabile all'interno del sistema. Si preferisce utilizzare degli elenchi in formato *.csv* perché sono più facilmente manipolabili in PHP. In verde chiaro è evidenziato un esempio di record valido, mentre in rosso uno non valido all'inserimento. In quest'ultimo caso si noti ad esempio come la riga 19 sia sprovvista di collocazione, mentre nella riga successiva essa non è nel formato adeguato.

Al momento del caricamento, lo script *manageBoardA.js* controlla se il file sia proprio nel formato *.csv*. In caso affermativo viene mostrato il nome del file caricato, altrimenti un messaggio di errore.

A questo punto è possibile procedere con il caricamento del file e la sua processazione lato server. Bisogna prestare particolare attenzione a come viene invocata la chiamata AJAX. Qui, a differenza di altre funzioni, si invia un file per cui una semplice chiamata via `$.get` o `$.post` (§ 3.3.1) può rivelarsi intricata. Quindi è stato adottato un metodo preciso per effettuare chiamate asincrone in cui sono previsti dei file in input: esiste un oggetto JavaScript di nome `FormData()` il quale, oltre alla semplice raccolta di dati, permette anche la trasmissione di file. Lo si immagini come una sorta di pacchetto comprendente tutte le informazioni da inviare al server.

Dunque, vengono reperiti il nome del plesso selezionato e il file caricato. In seguito viene creato un oggetto `FormData()` a cui si allegano tali dati tramite il metodo `.append("input in PHP", input)`. Dopodiché viene creata una nuova connessione

```
var xhr = new XMLHttpRequest();
```

e all'evento di cambio di stato `onreadystatechange` della richiesta, se non sono stati riscontrati errori, viene recuperato il valore di output dallo script di gestione del file *uplLibr.php*. Se esso è zero, non sono state aggiunti record nel sistema, altrimenti visualizza un messaggio di successo con il numero di record inseriti e quello totale. Infine, sempre lato client, si inizializza lo script di processazione e i dati di ingresso rispettivamente con i metodi `xhr.open("nomescript.php")` e `xhr.send(FormData)`. Illustrato il processo lato client, si mostri il funzionamento lato server analizzando il file di processazione *uplLibr.php*.

Inizializzate le variabili e controllata la loro esistenza, viene aperto un handler di apertura del file caricato in lettura. Se l'handler fallisce oppure il file non esiste allora restituisce un errore, altrimenti si procede all'analisi del file. Viene invocata la funzione `fgetcsv($handler, 0, ";")` che suddivide le singole righe del documento. A questo punto si scorrono tutte e si salvano in un array, ottenendo così una matrice. Fatto ciò, si conta il numero di righe e si inizializza un contatore di righe effettivamente memorizzate. Infine, per ciascuna riga, si esegue l'inserimento così come illustrato per quello manuale. Lo script ritornerà il numero di righe totali e quelle inserite nell'`onreadystatechange` dell'AJAX.

L’aggiornamento e la cancellazione di una voce all’interno della biblioteca sono pressoché simili a quello approfondito per i plessi (§ 3.3.2.1); tuttavia vi sono degli accorgimenti di cui bisogna tener conto:

- La tabella con l’elenco dei record bibliografici può risultare molto lunga e complicata da consultare. Così si è pensato di implementare una barra di ricerca che permette di effettuare una ricerca mirata con qualunque parametro desiderato (***Figura 20***).

Figura 20 *- Ricerca di un libro in console*

- Nel form di aggiornamento del libro vi è un’opzione che modifica tutte le copie dello stesso libro (ovvero tutte quelle con stesso titolo, autore, editore ed anno): in questo modo si possono correggere eventuali errori di battitura in una volta. Nota che quando si modifica la collocazione l’opzione si disattiva (***Figura 21***).

Figura 21 *- Particolare del form di aggiornamento di un libro*

3.3.2.3. Gestione degli utenti

L'inserimento di un utente all'interno del sistema avviene grazie al pulsante "*Aggiungi utente*" presente nella sezione apposita.

Il form di inserimento prevede la compilazione di tutti i campi, qui obbligatori: a differenza dei form illustrati in precedenza, qui è presente un pulsante che consente la generazione di un nuovo ID, unico campo di sola lettura e non modificabile manualmente. Inoltre, se l'utente da registrare nel sistema è un alunno, l'operatore deve immettere la sua classe e sezione; negli altri casi quella sezione di form non viene visualizzata in quanto un docente e un alunno che ha conseguito il diploma non dispone di tali dati. In questo caso verranno interpretati come vuoti e quindi in PHP riempiti con "–".

Sempre al variare del tipo di utente, cambia anche il range della data di nascita: questo è necessario per evitare casi di inconsistenza. Ad esempio, è impossibile che una persona di 13 anni possa essere abilitato all'insegnamento.

L'ID dell'utente è univoco ed è necessario sia all'identificazione dell'utente di una biblioteca dell'istituto, sia alla prenotazione dei libri (da remoto e non). Come verrà illustrato nel § 3.3.3, non è necessario immettere delle credenziali di login per poter usufruire del servizio. Si è optato per una combinazione di $k = 6$ caratteri alfanumerici sia per una maggiore sicurezza, sia per il numero di combinazioni possibili. Infatti, secondo il calcolo combinatorio, con $n = 36$ caratteri si possono ottenere

$$C'_{n,k} = \binom{n+k-1}{k} = \binom{36+6-1}{6} = \binom{41}{6} = 4496388$$

combinazioni: dunque c'è una probabilità molto bassa che due utenti abbiano lo stesso codice.

Accertata la correttezza di tutti i campi lato client, si procede all'inserimento dell'utente nel database tramite lo script *submitStdnt.php*[19]. Vengono svolti gli stessi procedimenti illustrati con l'accorgimento qui di seguito spiegato. Dopo il controllo dell'ID si verifica, in caso di inserimento di un alunno, la correttezza della classe: un plesso di scuola primaria arriva fino alla quinta classe, gli altri fino alla terza. Quindi se si seleziona un plesso di scuola *non* primaria e una classe superiore alla terza, il sistema dà errore.

Figura 22 *- Finestra di aggiornamento di un utente nel database*

L'aggiornamento e la cancellazione di un utente dal sistema si comporta esattamente come con le altre entità. Riguardo l'aggiornamento è stata introdotta una funzione utile, specie durante il periodo di scrutini, in cui bisogna promuovere gli alunni alla classe successiva. Nella finestra di aggiornamento è presente una funzione che permette la promozione di tutti gli alunni alla classe successiva di quel plesso (***Figura 22***). La cancellazione si comporta come con gli altri elementi.

[19] Gli elementi relativi agli utenti sono attribuiti agli alunni (dicitura *Stdnt* e simili) perché inizialmente il sistema di prenotazione da remoto era pensato solo per gli alunni. In seguito, si è esteso anche ad altri tipi di utente.

3.3.2.4. Gestione degli operatori

L’amministrazione degli operatori (qui admin) è riservata ai gestori del sistema, così come dei plessi (§ 3.3.2.1). L’opzione di inserimento di un operatore è disponibile all’interno della sezione *Admin* gestita da *manageBoardA.js*.

Nel form di inserimento di un admin vengono effettuati dei controlli sull’esistenza dello username e dell’indirizzo e-mail, sempre applicando l’algoritmo mostrato nel § 3.3.1. Se i campi Username, E-Mail e Password rispettano il formato richiesto e i primi due valori non esistono all’interno del database (controllo gestito dagli script *checkUser.php* e *checkMail.php*), viene chiesto di riscrivere tali parametri come conferma. Accanto il campo password è presente un pulsante che permette di mostrare la password: essa deve contenere almeno una lettera maiuscola, una cifra e un carattere speciale.

Compilato il form con tutti i campi corretti, si procede all’inserimento lato server via lo script *submitAdm.php*. Dichiarate le variabili di input e verificata l’esistenza di tali valori, si controlla se esiste un account che ha quello username o quell’indirizzo e-mail: in caso positivo, viene restituito un errore. Infine, si procede all’inserimento dell’operatore nel database, esattamente come illustrato nel § 3.2 in cui viene aggiunto l’operatore di default.

L’aggiornamento e la cancellazione di un operatore sono strutturati come illustrato nei paragrafi precedenti.

3.3.2.5. Gestione dei prestiti

Descritta la gestione delle entità principali del sistema bibliotecario, adesso si illustra il meccanismo di gestione dei prestiti nella console. In questo paragrafo verrà mostrato sia come viene avviato un prestito all'interno della sede bibliotecaria, quindi in presenza, sia come si possono gestire i prestiti in corso.

***Figura 23** - Esempio di box di ricerca nel pannello dei prestiti*

Aperto il box *Nuovo prestito* all'interno della sezione *Prestiti* viene generato il form apposito. Così come con gli utenti (§ 3.3.2.3), è presente un campo di sola lettura in cui è visibile il codice del prestito, una sequenza di 8 cifre. I due input successivi sono con completamento automatico, in cui è possibile cercare l'utente e il libro di riferimento.

Le opzioni di ricerca come in ***Figura 23*** sono generate grazie agli script *stMgrResult.php* e *exBkResult.php*. Essi, tramite chiamata AJAX con il metodo GET[13], recuperano in tempo reale le prime dieci opzioni che includono la stringa immessa dall'operatore. Nota che per quanto riguarda i libri, vengono presi in considerazione solo quelli disponibili, non quelli in prestito, né quelli di sola consultazione.

Al click della voce scelta, il campo si riempie con l'ID dell'entità corrispondente: per un maggiore controllo, al momento della selezione vengono visualizzati in basso dei box contenenti le informazioni dell'utente e del libro selezionati (un esempio in secondo piano, ***Figura 23***).

In caso di eventuali problemi da parte dell'utente e/o dell'operatore, è stato inserito un campo *Data* in cui è possibile eventualmente posticipare l'avvio del prestito.

Definite le entità del prestito, si può avviarlo e quindi registrarlo all'interno del database: l'avvio del prestito è gestito dal file *submitEx.php*. Recuperati i valori in input, si svolge un primo controllo del loro formato: se i requisiti sono rispettati, si controlla l'esistenza dell'utente e del libro all'interno del database. Se almeno uno non esiste o non è valido, oppure se quella voce bibliotecaria non è disponibile al prestito il sistema invia un messaggio di errore e lo script viene chiuso.

In caso contrario, si procede all'esecuzione di due query:

- Inserimento del prestito nella tabella apposita;
- Aggiornamento dello stato del libro da *Disponibile* a *In prestito*.

Se tutti i campi sono validi, le query vengono eseguite e l'inserimento va a buon fine. Bisogna tener conto che, ovviamente, quando un prestito in corso la data di prestito è definita, ma non quella di restituzione.

A questo punto, il prestito inserito è disponibile all'interno dell'*Elenco prestiti* all'interno della scheda apposita. Essa è, di consueto, provvista di una barra di ricerca con cui è possibile trovare la voce desiderata.

Sopra la barra è presente una piccola finestra informativa che visualizza in numero di prestiti in attesa, in corso e conclusi.

Nella digitazione non è necessario conoscere i codici precisi in quanto è un tipo di ricerca mirato: ad esempio, cercando "*Il giardino segreto*", oppure la sua collocazione, oppure il cognome dell'utente che ha preso in prestito il libro, rimanda direttamente all'insieme di valori richiesti. In ***Figura 24*** si illustra un esempio.

Un limite di questo sistema è il fatto che nell'ambito della visualizzazione dei risultati vi sono soltanto i codici di riferimento, non gli altri dati (che sono sempre ricercabili nelle altre tabelle). Nei prossimi aggiornamenti della piattaforma, se richiesti dall'istituto, sarebbe utile creare dei *tooltips*[20] contenenti le informazioni di quella specifica entità, che sia un libro, un utente o un operatore (cosa diversa per il plesso perché il suo nome è direttamente visualizzabile).

Tutti i prestiti sono ordinati per stato (quelli in attesa di conferma sono messi sempre in alto, poi quelli in corso e infine quelli conclusi), poi per data di prestito, data di restituzione e infine per ID del libro. Un prestito in corso ha due azioni principali: la conclusione di esso e la rimozione diretta. Quest'ultima opzione è utile specialmente in caso di errore o ripensamento da parte dell'operatore.

Figura 24 *- Esempio di ricerca di un prestito*

Cliccando sul pulsante di eliminazione compare un modal[16] in cui si chiede la rimozione del prestito. In caso positivo, lo script *delEx.php* lo elimina dalla tabella.

[20] Un *tooltip* è un messaggio che compare sopra (o sotto) il testo contenente brevi informazioni aggiuntive.

Invece, selezionando la conclusione del prestito e confermando la scelta, lo stato del libro passa a *Concluso* e viene inserita la data di restituzione con la data corrente: il tutto è gestito dal file *endEx.php*. In entrambi i casi, vi è anche un passaggio di stato del libro da *In prestito* a *Disponibile*.

I prestiti conclusi invece hanno solo una semplice opzione, quella della rimozione dall'elenco. Nella finestra di conferma di eliminazione di un semplice prestito concluso è stato introdotto un pulsante in grado di svuotare l'elenco dei prestiti conclusi, mantenendo quelli in attesa e in corso.

La gestione dei prestiti in attesa verrà illustrato nel § 3.3.3, in concomitanza alla descrizione del meccanismo di prenotazione da remoto da parte dell'utente che intende prendere in prestito un libro a distanza.

3.3.3. Prenotazione da remoto

Oltre ai consueti prestiti in presenza, è sorta la necessità di poter dare la possibilità agli utenti registrati nelle sedi bibliotecarie di poter prenotare un libro a distanza.

Prima di ciò è bene illustrare il funzionamento della ricerca di una voce all'interno del database. La pagina principale *index.php* consta di una barra di ricerca e poco sotto dei filtri che permettono la sua affinazione, come mostrato in ***Figura 25***: nell'esempio riportato si sta provando a trovare un libro disponibile al prestito uscito nel 2011.
In basso, a fini estetici, sono riportati il numero

Figura 25 *- Esempio di ricerca di una voce nella pagina iniziale*

di libri, quello di autori e il numero di record disponibili al prestito all'interno del catalogo.

Il processo di ricerca *non* è asincrono, quindi *non* è stato utilizzato AJAX. Come mostrato nel codice HTML, nel form della barra di ricerca in *index.php* si dichiara il metodo di passaggio della query e l'action, ovvero in questo caso il reindirizzamento alla pagina dei risultati *result.php* (***Figura 26***) in cui si passano i parametri dichiarati:

```
<form class="search" action="result.php" method="GET">
```

Come esposto in precedenza, sotto la barra c'è una serie di filtri che mirano la ricerca. Essi sono degli input di tipo *checkbox* (i box che si spuntano) e passano come input dei valori booleani (`0` o `1`).

All'avvio della ricerca, l'utente viene rimandato alla pagina *result.php* con il suo esito. Se non ci sono risultati o la stringa è vuota allora viene mostrato un messaggio di errore, altrimenti si visualizza una schermata simile a quella in ***Figura 26***.

Figura 26 *- Esempio di risultato di ricerca con i parametri in* ***Figura 25****.*

I risultati in oggetto vengono generati da una query dinamica ottenuta dallo script *resScript.php*. Al variare della combinazione dei valori delle *checkbox* presenti in *index.php*, viene generata una query diversa in cui il corpo delle istruzioni `SELECT` e `FROM` è lo stesso, ma cambia quello del `WHERE`. Questo perché ad un campo selezionato corrisponde una condizione all'interno di esso: tutte saranno incatenate dall'operatore logico `OR` poiché la selezione di più filtri indica un

insieme di alternative, un'unione. Caso a parte è la selezione dei volumi disponibili al prestito, è preceduto da un `AND`, cioè un'intersezione.

Adesso si presti attenzione alla pagina dei risultati di ricerca in ***Figura 26***: accanto allo stato della voce visualizzata è presente un pulsante *Prenota* che permette, appunto, la prenotazione di un libro in uno dei plessi dell'istituto. Nel tentativo di ricerca, gli operatori che hanno effettuato l'accesso non vedranno tali pulsanti: lo stesso vale per i record bibliografici in prestito (non è possibile prenotare un libro già in prestito da qualcuno, bisogna aspettare che lo restituisca) e di sola consultazione.

Scelto il libro desiderato, l'utente clicca sul pulsante "*Prenota*" corrispondente ad esso. Comparirà una finestra in cui vengono richieste le generalità dell'utente (***Figura 27***): questo non è altro che un primo controllo di esistenza all'interno del sistema. Infatti, se la persona in oggetto non è registrata viene visualizzato un messaggio di errore. La gestione delle prenotazioni da remoto avviene tramite lo script *result.js*: la struttura del form ricorda quello di modifica di un utente nella console, solo che qui è l'utente a immettere le proprie generalità. Inoltre, non è necessario immettere il tipo di utente perché grazie al proprio codice egli verrà riconosciuto automaticamente, per cui tale dato è superfluo.

Figura 27 *- Form di prenotazione di un libro da parte dell'utente*

Questo controllo è efficiente nell'ambito della verifica dei dati, ma può rivelarsi macchinoso perché nel caso di una prenotazione di più libri di testo bisognerebbe compilare ogni volta tutti i dati. Quindi per accorciare i tempi di compilazione sarebbe interessante creare un *cookie*[21], magari in aggiornamenti futuri.

Dopo aver compilato i campi e verificata l'esistenza dell'utente all'interno del database, viene chiamato lo script *checkStdnt.php* che controlla l'effettiva esistenza dell'utente. Se l'esito è positivo, il form con le sue generalità dà spazio a quello di inserimento del proprio codice (***Figura 28***).

Alla conferma viene effettuata sia la verifica del codice utente che gli opportuni inserimenti nel database. Lo script *confirmBk.php* prende come input non solo il codice inserito, ma anche tutti i dati inseriti prima: si tratta di un caso di chiamata AJAX annidata, in cui una funzione `$.post` si trova all'interno di un'altra funzione `$.post`. Questo non è l'unico caso riscontrato in questo progetto: basti pensare, ad esempio, alle finestre di aggiornamento e cancellazione nella console di gestione (§ 3.3.2).

Figura 28 *- Box di immissione del codice utente*

[21] Un *cookie* è un file che memorizza particolari informazioni nella macchina dell'utente fino alla distruzione del cookie stesso. Sono utili, ad esempio, nel salvataggio di preferenze in un sito, oppure per memorizzare articoli nel carrello durante la navigazione.

Prima dell'inserimento del prestito nel database con i dati inseriti, il file *confirmBk.php* controlla:

- l'esistenza dell'utente;
- l'effettiva disponibilità del libro (ovvero che non sia già in prestito);
- che il libro non sia già stato messo in attesa da un altro utente.

Verificate queste tre condizioni, viene inserita nella tabella *Prestito* un record con il codice del prestito, il libro, l'utente, il plesso di riferimento e lo stato impostato come `'Q'` (per *Queue*, ovvero *in coda*). In questo modo la richiesta viene inviata e il prestito è in attesa di essere approvato da un operatore.

Quando arrivano prestiti in attesa, l'operatore collegato alla console vedrà una notifica sul pulsante del menu *Prestiti* ad indicare che un utente intende prendere un libro in sede e quindi necessita approvazione. Quindi ci si sposta nell'elenco dei prestiti e si visualizza la coda. Come specificato nel § 3.3.2.5, i prestiti in attesa vengono mostrati sempre *prima* di tutti gli altri perché hanno una maggiore priorità.

Per praticità sono state introdotte delle *hot keys*[22] che permettono una navigazione più rapida. Digitando il singolo carattere '`q`' nella barra di ricerca si visualizzano tutti i prestiti in attesa. Questo è possibile perché quando si cerca un prestito all'interno della tabella, nella query apposita è inclusa anche la ricerca dello stato del prestito stesso. Passando il mouse sull'etichetta con il numero di prestiti in corso viene mostrata la *hot key* da digitare.

[22] Una *hot key* è una "chiave" che permette un rapido accesso a una determinata funzione. Ad esempio, quando al computer si digita `CTRL+C` si copiano degli elementi in memoria per poi incollarli con `CTRL+V`.

Così come il prestito in corso, anche quello in coda consta di due azioni: la sua conferma oppure la sua eliminazione. Dato che, al momento, non sono stati implementati sistemi di notifica automatici che avvisano in tempo reale l'inoltro di un prestito in attesa, oppure la conferma o meno del prestito stesso, è dovere dell'operatore comunicare l'esito direttamente all'utente prima dell'effettiva convalida: ecco perché al momento della registrazione l'utente fornisce un numero di cellulare.

In caso di rifiuto da parte dell'operatore, cliccando sul pulsante apposito viene generato il modal di riferimento: alla conferma della rimozione viene invocato, come visto in § 3.3.2.5, *delEx.php*.

Al contrario, in caso di accettazione del prestito, lo script *beginEx.php* controlla in primo luogo se il libro è già stato preso in prestito: in caso negativo vengono effettuati due aggiornamenti, ovvero quello del libro (il cui stato passa da *Disponibile* a *In prestito*), e quello del prestito in attesa (si definiscono la data di inizio, l'operatore che l'ha accettato e lo stato passa da *In attesa* a *In prestito*). Tutte le altre funzioni relative ai prestiti sono illustrate nel § 3.3.2.5.

4. Integrazione della Web Application in una rete locale

Sviluppata la piattaforma è possibile fare un passo avanti nella discussione. Ci si chiede se il sistema bibliotecario sviluppato finora possa avere i requisiti necessari per aderire a una rete bibliotecaria locale, ovvero Bibliolandia.

In questo capitolo si studieranno le generalità di un catalogo bibliotecario, la struttura dei REICAT, della rete Bibliolandia e del software associato. In seguito, si valuterà se il database ottenuto sia conforme a quello della rete e le possibili modifiche da attuare per l'integrazione.

4.1. Nozioni principali

Prima di introdurre la rete Bibliolandia (§ 4.2) è bene illustrare, come accennato nel § 2.1, quelli che sono i concetti principali di questo settore, a cominciare dal catalogo digitale (§ 4.1.1) per poi passare a REICAT (§ 4.1.2), a rete bibliotecaria (§ 4.1.3) e infine a OPAC (§ 4.1.4). Successivamente, in base alla rete illustrata si faranno opportune valutazioni su una possibile integrazione del sistema bibliotecario sviluppato a Bibliolandia (§ 4.3).

4.1.1. Catalogo

La biblioteconomia è la disciplina che studia i criteri e le modalità di raccolta e ordinamento, oltre ai problemi relativi alla descrizione formale di risorse bibliografiche[23].

[23] *Guida alla biblioteconomia* / a cura di Mauro Guerrini; con Gianfranco Grupi e Stefano Gambari; collaborazione di Vincenzo Fugaldi, Milano, Editrice Bibliografica, 2008, p. 22-23.

Il termine più preciso per indicare un insieme di risorse bibliografiche che possiedono tutte le medesime caratteristiche è *catalogo*. Quando si tratta di catalogazione si intende l'atto di registrare dei record bibliografici rispettando dei criteri standard, validi per tutti. Riprendendo il database creato (§ 2.3.1), la tabella *Libro* è considerabile il catalogo elettronico che gli utenti sono in grado di consultare.

È importante sottolineare che, a differenza di una semplice tabella, le registrazioni catalografiche di un catalogo forniscono un determinato servizio all'utente[24]: nello schema illustrato egli è in grado di consultare una risorsa oppure prenderla in prestito; inoltre, il catalogo deve mostrarsi uniforme e coerente (ad esempio, il formato della collocazione del record catalografico, la formattazione degli autori e così via). Si tratta di sistemi informativi, che sono in grado di prestare un certo servizio all'utente; quindi, un catalogo non è un semplice insieme di notizie catalografiche.

Con l'avvento della catalogazione digitale la gestione bibliotecaria è diventata più semplice e più efficiente, sempre se il sistema sia sviluppato bene. L'utente è in grado di cercare con molta più facilità la risorsa desiderata, senza dover scorrere intere tabelle con tutte le notizie bibliografiche disponibili, magari senza sapere se la pubblicazione desiderata è già in prestito.

[24] Petrucciani A., Turbanti S., *Manuale di catalogazione: principi, casi e problemi*, Milano, Editrice Bibliografica, Associazione Italiana Biblioteche, 2021, p. 11.

La funzione principale di un catalogo bibliotecario è il controllo della disponibilità di una certa risorsa e la sua relativa posizione. Nel 1875, Charles A. Cutter[25] definì nel suo saggio *Rules for a printed dictionary catalogue* le funzioni principali di un catalogo, i cui intenti principali sono i seguenti:

- L'utente deve essere in grado di trovare una risorsa cercando:
  - l'autore,
  - il titolo,
  - il soggetto
- La biblioteca deve avere i requisiti per mostrare cosa possiede:
  - di un autore,
  - su un argomento,
  - in un genere
- Suggerire nella scelta di una risorsa:
  - riguardo la sua edizione bibliografica,
  - riguardo il suo carattere, di tipo letterario o saggistico.

In qualche modo, queste funzioni sono state riprese in seguito per poter definire il modello FRBR[26], basato su quello E-R (§ 2.2.1), per poter rappresentare un record catalografico. FRBR ha l'obiettivo di sintetizzare in maniera precisa le informazioni bibliografiche di una risorsa all'interno di un catalogo, identificandone i parametri ritenuti essenziali, a partire dai bisogni sia dell'utente che usufruisce del servizio, sia di quello che lo gestisce (l'operatore). A differenza dei modelli precedenti, che erano incentrati sulla descrizione della pubblicazione, FRBR si concentra maggiormente sui bisogni dell'utente.

[25] Charles Ammi Cutter è noto per aver introdotto i quadri espansivi di classificazione delle risorse bibliografiche e per le tavole di collocazione delle opere in ordine alfabetico.
Cutter, Charles Ammi. "Treccani", consultato il 10 gennaio 2022, https://www.treccani.it/enciclopedia/charles-ammi-cutter/.

[26] International Federation of Library Associations and Institutions, Study Group on the Functional Requirements for Bibliographic Records, *Functional requirements for bibliographic records: final report*, approved by the Standing Committee of the IFLA Section on Cataloguing, München, Saur, 1998.

Secondo il modello FRBR, i target principali di ricerca di un catalogo bibliografico possono essere raggruppati in:

- *Opere*, ovvero risorse create dall'attività individuale
    - *Opera*, il contenuto intellettuale
    - *Espressione*, una realizzazione di un'opera
    - *Edizione*, la materializzazione di un'espressione
    - *Item*, un esemplare di una manifestazione;
- *Persone o enti*, responsabili della produzione, della tutela o della distribuzione della risorsa;
- *Soggetti* contenuti nell'opera: concetti, oggetti, eventi e luoghi.

Quindi, a partire dal titolo dell'opera o da eventuali indicazioni di responsabilità, l'utente è in grado di trovare la pubblicazione desiderata. Questo ad oggi è possibile, poiché la biblioteca non è più intesa solamente come lo spazio fisico in cui consultare delle risorse bibliografiche[27].

4.1.2. REICAT

Dal 1979 al 2009 le regole di catalogazione bibliografica erano dettate dalle *Regole Italiane di Catalogazione per Autori* (RICA). Si trattava di un codice contenente i principi generali di intestazione, sintassi e formattazione delle singole informazioni di una risorsa bibliografica.

In seguito, con l'avvento dei mezzi informatici, quindi dei cataloghi elettronici, dei cataloghi collettivi, in contemporanea all'introduzione di nuovi tipi di materiali e di nuove esigenze di biblioteche ed utenti, è sorta la necessità di rivedere e aggiornare le RICA perché divenute obsolete.

[27] *Guida alla biblioteconomia* / a cura di Mauro Guerrini; con Gianfranco Grupi e Stefano Gambari; collaborazione di Vincenzo Fugaldi, Milano, Editrice Bibliografica, 2008, p. 38-39.

Così, nel 1996 venne convocata una commissione permanente con il compito di rivedere le RICA: al termine del lavoro, nel 2009 esse vennero soppresse a favore delle *Regole Italiane di Catalogazione*[28] (REICAT).

Le REICAT, grazie a determinati formalismi adottati, permettono una maggiore leggibilità e uniformità della descrizione delle notizie bibliografiche: infatti, aiutano molto il catalogatore nell'identificazione di tutte le informazioni dell'esemplare, a partire dal titolo e dalle indicazioni di responsabilità con tutte le informazioni di edizione e di pubblicazione[29].

Nel caso della piattaforma implementata per l'istituto scolastico, le informazioni non seguono il modello REICAT: l'elenco in ***Figura 19*** mostra una catalogazione poco curata.

Basandosi sul modello FRBR (§ 4.1.1), le REICAT incentrano la catalogazione sulla pubblicazione in sé. Il manuale delle REICAT[3] è suddiviso in tre parti principali:

- Descrizione bibliografica e informazioni sull'esemplare;
- Opere ed espressioni in relazione a delle pubblicazioni;
- Responsabilità persone/enti in relazione alle loro opere.

Nella trattazione in corso si terrà conto delle REICAT come riferimento del formato delle informazioni bibliografiche inserite all'interno del sistema, ovvero se ad esempio i titoli o gli autori di un libro rispettano i criteri stabiliti.

[28] *Regole italiane di catalogazione: REICAT*, a cura della Commissione permanente per la revisione delle regole italiane di catalogazione, Roma, ICCU, 2009.

[29] Simona Turbanti, *REICAT*, Roma, Associazione Italiana Biblioteche, 2016.

4.1.3. Rete bibliotecaria

Una rete bibliotecaria è un insieme di biblioteche interconnesse tra loro grazie a degli accordi di cooperazione e collaborazione, diventando una sorta di grande biblioteca con un unico catalogo condiviso dalle singole sedi. Tutte le biblioteche che afferiscono a una stessa rete hanno un peso distribuito sul catalogo perché condiviso, quindi tutte godono degli stessi benefici.

La più importante rete bibliotecaria nazionale è il Servizio Bibliotecario Nazionale (SBN)[30], promosso dal Ministero dei Beni e delle Attività Culturali (MIBAC). Ad esso aderiscono oltre 6.000 sedi bibliotecarie di diverso settore, come quelle pubbliche o quelle universitarie[31].

Vi sono inoltre delle reti bibliotecarie di ampiezza minore, a livello locale, che si comportano esattamente come quelle del calibro di SBN, ovvero le reti locali. Tra queste, nell'area toscana, una delle più importanti è Bibliolandia (§ 4.2).

Caratteristica peculiare delle reti bibliotecarie è quella della *catalogazione partecipata*: è una forma di arricchimento collettivo del catalogo di una rete. Quando una biblioteca acquisisce una determinata risorsa, prima si controlla se essa sia già stata catalogata da altre sedi all'interno della rete: se è già presente all'interno del catalogo, allora si dovranno inserire soltanto i dati relativi alla sua collocazione. Altrimenti vengono inseriti tutti i metadati della nuova risorsa. Tuttavia, è molto frequente, almeno nel caso di SBN, che un record all'interno del sistema si ripeta più volte perché spesso non viene eseguito il controllo dell'esistenza di quella pubblicazione all'interno del sistema.

[30] *OPAC SBN. Catalogo del Servizio Bibliotecario Nazionale*, a cura dell'Istituto Centrale per il Catalogo Unico, Roma, ICCU, consultato l'11 gennaio 2022. https://nuovo-opac.sbn.it/.

[31] *Informazioni catalogo SBN*, a cura dell'Istituto Centrale per il Catalogo Unico, Roma, ICCU, consultato l'11 gennaio 2022.
https://www.iccu.sbn.it/export/sites/iccu/documenti/OPAC/OPAC_SBN_Informazioni.pdf.

4.1.4. Definizione di OPAC

Con l'avvento dei mezzi informatici sono cambiati anche i mezzi di gestione delle biblioteche: quindi esse diventano *elettroniche* e sono in grado di organizzare meglio il lavoro che fino a qualche tempo prima spettava agli schedari (§ 4.1.1). Una delle funzioni principali di una biblioteca elettronica è la disponibilità di un catalogo consultabile online dagli utenti: questo è detto OPAC (*Online Public Access Catalog*, cioè *Catalogo Online di Accesso Pubblico*).

Quindi un OPAC non è altro che un catalogo digitale, eventualmente condiviso da una rete bibliotecaria, interrogabile e consultabile gratuitamente. Ciascun OPAC ha dei metodi di interrogazione diversi l'uno dall'altro perché, in base alle esigenze della biblioteca, o della rete bibliotecaria, viene sviluppato in modo diverso.

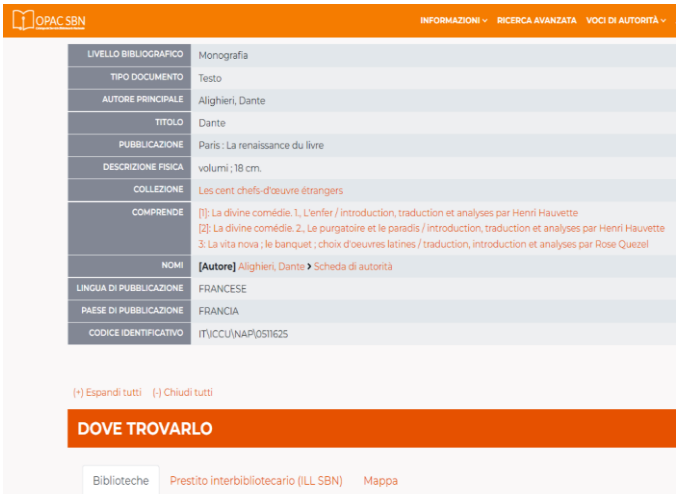

Figura 29 *- Esempio di voce all'interno dell'OPAC del SBN.*

La consultazione di un OPAC offre diversi vantaggi, come il controllo della disponibilità al prestito, la possibilità di una ricerca mirata, e così via; è un catalogo che si basa su FRBR (§ 4.1.1) e che assume come punto di riferimento l'esemplare in sé.

Si prenda in considerazione una voce all'interno del catalogo SBN e si analizzi il contenuto (***Figura 29***). A differenza del database creato, qui vengono inseriti anche il livello bibliografico e il tipo di documento; inoltre, il titolo e altri campi presentano una formattazione diversa rispetto a quelle presenti all'interno degli elenchi forniti dall'istituto scolastico perché le risorse bibliografiche all'interno dell'OPAC rispettano le REICAT (§ 4.1.2).

Altro dettaglio da non trascurare è che le diverse edizioni della pubblicazione sono collegate alla prima e che viene citato se quella pubblicazione fa parte di una certa collana di volumi. Poco sotto vi è l'elenco di tutte le ubicazioni in cui quella pubblicazione è disponibile alla consultazione e/o al prestito.

4.2. Una rete bibliotecaria locale: Bibliolandia

Dopo aver trattato i caratteri principali di una rete bibliotecaria e di un OPAC si illustri Bibliolandia e il suo funzionamento.

Ci si chiede per quale motivo il database bibliotecario dell'istituto scolastico possa più afferire ad una rete di ampiezza più ristretta come Bibliolandia e non direttamente ad una di portata maggiore come SBN. Esistono due modi per poter siglare un'adesione con SBN:

- Collegamento diretto con SBN grazie a degli accordi con ICCU[32];
- Collegamento con SBN per mezzo di sottoreti già esistenti, proprio come nel caso di Bibliolandia[33].

[32] *Istituto Centrale per il Catalogo Unico*. Si tratta dell'ente che si occupa della gestione e dell'organizzazione di SBN.

[33] *Polo pisano della Rete documentaria Bibliolandia (PBI)*. a cura dell'Istituto Centrale per il Catalogo Unico, Roma, ICCU, consultato il 13 gennaio 2022. https://www.iccu.sbn.it/it/SBN/poli-e-biblioteche/polo/PBI-Polo-pisano-della-Rete-documentaria-Bibliolandia/.

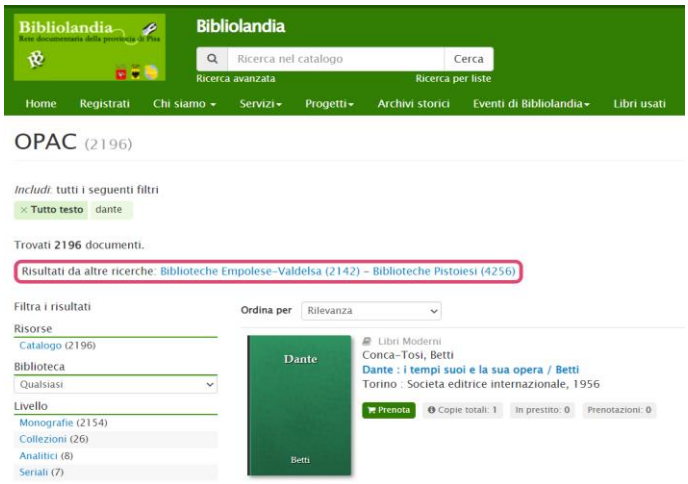

Figura 30 *- Pagina di ricerca nell'OPAC di Bibliolandia.*

Per quanto riguarda la prima opzione, essa si rivela dispendiosa: infatti, l'adesione di una singola biblioteca può essere approvata solo in casi eccezionali. Inoltre, sono previsti dei costi, che possono essere anche ingenti, proporzionati alle modalità stabilite nella convenzione[34]. Dunque, per poter in qualche modo ridurre i costi e soprattutto semplificare l'*iter* di adesione a SBN, conviene aggregarsi a una rete locale per poter comunque vantare di vantaggi simili, se non maggiori, al collegamento diretto a SBN. Detto questo, si descrive qui di seguito Bibliolandia e il suo funzionamento.

La rete Bibliolandia[35], nata nel 1999, ad oggi si snoda in 67 sedi bibliotecarie della provincia di Pisa: la maggior parte di esse sono dislocate nei comuni di Pontedera, nodo principale della rete (15 biblioteche), Pisa (11) e San Miniato (5). A Bibliolandia afferiscono, oltre a diverse biblioteche comunali della provincia, anche delle sedi bibliotecarie di diversi enti, associazioni e istituti scolastici, sia di istruzione superiore che comprensivi. Inoltre, si tratta di una rete che conta più di 500.000 volumi e si riscontrano tutte le caratteristiche citate nel § 4.1. La ***Figura 30*** mostra un esempio di risultati di ricerca all'interno dell'OPAC di Bibliolandia. Nota che il comportamento è pressoché simile a quello riscontrato con SBN (***Figura 29***).

[34] *FAQ: Domande generali frequenti sulla circolare 138/2002. Come si fa ad aderire a SBN?*, a cura della Direzione generale Biblioteche e diritto d'autore, 2002, consultato il 13 gennaio 2022. https://www.librari.beniculturali.it/it/documenti/FAQpremiamo_i_risultati.pdf.

[35] *Bibliolandia. Rete documentaria della provincia di Pisa*, Bibliolandia. Consultato il 13 gennaio 2022. https://bibliolandia.comperio.it/.

Così come in SBN, i metadati di una scheda bibliografica dell'OPAC di Bibliolandia rispettano le REICAT e sono tutte ben collegate tra di loro: ad esempio, cliccando sul nome dell'autore di una pubblicazione si rimanda ai risultati di ricerca di quell'autore, oppure è possibile navigare tra diverse edizioni di una medesima opera.

Si illustri il funzionamento tecnico di Bibliolandia. La rete si appoggia al sistema Comperio[36], specializzato nell'organizzazione e nella gestione di sistemi bibliotecari.

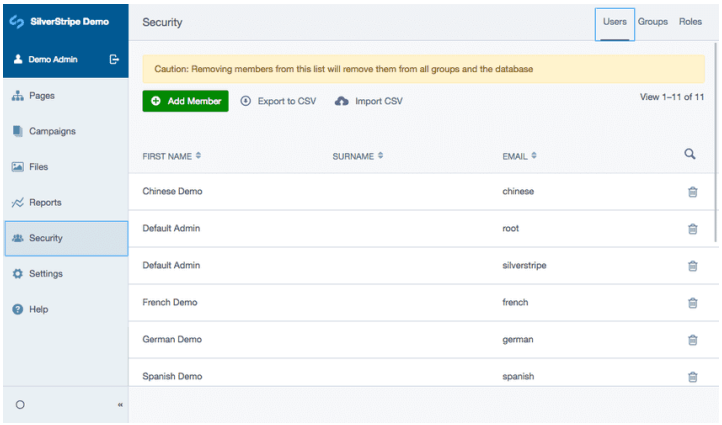

Figura 31 *– Schermata di gestione degli utenti dell'interfaccia del framework SilverStripe adottato da Comperio per lo sviluppo di DiscoveryNG.*

In particolare sono stati utilizzati diverse componenti esterne nello sviluppo e nella realizzazione di Bibliolandia, a cominciare da alcune librerie già utilizzate nel sito sviluppato (come ad esempio jQuery, § 3.1).

Il fulcro della gestione catalografica del sistema della rete è DiscoveryNG, software sviluppato da Comperio: a sua volta, DiscoveryNG è basato su SilverStripe[37], CMS[38] per il linguaggio PHP (§ 3.1).

Il framework si comporta simile alla console di gestione implementata (§ 3.3.2) con delle funzionalità in più: ad esempio, a livello di login si possono salvare le credenziali di accesso (questo per mezzo di qualche cookie[21]), oppure si possono creare dei ruoli personalizzati a cui assegnare determinati permessi (***Figura 31***).

[36] *Credits*. Bibliolandia, consultato il 13 gennaio 2022. https://bibliolandia.comperio.it/credits.

[37] *Silverstripe CMS User Help guide*, SilverStripe, consultato il 13 gennaio 2022. https://userhelp.silverstripe.org/en/4/.

[38] Un CMS (*Content Management System*) è uno strumento che permette di semplificare la gestione dei contenuti di un sito web, evitando di programmare dinamicamente lato server.

Quest'ultima funzionalità non è stata implementata nel progetto poiché sono stati definiti *a priori* due ruoli distinti con privilegi diversi.

Nel § 4.3 verrà svolta un'analisi sia relazionale che catalografica del sistema bibliotecario creato e si effettueranno delle valutazioni in merito a migliorie che possono permettere una possibile integrazione alla rete Bibliolandia.

4.3. Integrazione dell'applicazione in Bibliolandia

Definiti i concetti principali, dopo aver illustrato Bibliolandia e il suo funzionamento, è opportuno fare delle osservazioni in merito alla compatibilità della Web Application sviluppata per l'istituto scolastico a Bibliolandia e relativi miglioramenti da implementare sia dal punto di vista tecnico che catalografico.

Tecnicamente, gli strumenti utilizzati nella realizzazione del lavoro (§ 3.1) possono considerarsi soddisfacenti: questi linguaggi sono ampiamente utilizzati e sono necessari nella gestione di informazioni di questo tipo.

Tuttavia ci sono molte ridondanze e ripetizioni di codice che sarebbe preferibile evitare al fine di migliorare ancora le prestazioni del sito: queste sono facilmente gestibili grazie all'introduzione sia di funzioni apposite (specialmente lato client), sia della programmazione a oggetti (nell'ambito della stabilità, della sicurezza e dell'organicità dei dati). Riguardo quest'ultima caratteristica, poiché non è stata affrontata nel corso del percorso accademico non è stata presa in considerazione, ma potrà essere implementata in futuro. Si potrebbe in qualche modo fare riferimento al framework SilverStripe[37] adottato da Comperio e prenderlo a modello negli aggiornamenti successivi.

Considerando il modello creato in ***Figura 6*** (§ 2.2.1), a livello tecnico il database sembra un buon punto di partenza. Per adattarsi a Bibliolandia bisogna arricchire l'entità *Libro* di informazioni nello schema ottenuto nel capitolo 2.

Questo perché il database non è bibliocentrico: la risorsa non è ricca di informazioni come negli altri OPAC e non è vista come punto di riferimento all'interno della piattaforma.

Attualmente, l'elenco dei record catalografici si rivela sterile e con carente contenuto bibliografico. Tra i metadati di una risorsa (***Figura 29***) mancano ad esempio l'immagine di copertina, le dimensioni, la descrizione fisica, la lingua, il paese, il genere della risorsa e altri dati descrittivi. L'interfaccia (§ 3.1) non è quindi provvista di una pagina dedicata alla pubblicazione in cui sono visualizzabili tutte queste informazioni.

Inoltre, l'OPAC attuale del sistema bibliotecario non soddisfa in alcun modo le REICAT: non esiste una formattazione standard dei titoli e degli autori della pubblicazione. Ecco che è necessario lavorare anche in questa direzione per migliorare sempre più la qualità del sistema.

Altra caratteristica da prendere in considerazione è il criterio con cui sono state definite le collocazioni. Analizzando uno degli elenchi forniti dall'istituto scolastico (***Figura 19***), sembra che sia stata adottata una linea più generale. Il codice della collocazione è relativo a un macro-genere, a cui viene assegnato un indice all'interno di tale categoria. Ad esempio, la collocazione di codice I/25 indica una pubblicazione di categoria "*Libri in inglese*" al posto numero 25.

Quindi sarebbe opportuno adottare una classificazione più analitica delle pubblicazioni, come ad esempio l'applicazione della classificazione Dewey. Tuttavia, poiché si tratta di una micro-rete destinata principalmente a un'utenza scolastica, per semplicità i metodi di notazione adottati possono considerarsi accettabili. In ogni caso, l'adozione della classificazione Dewey migliorerebbe sensibilmente la categorizzazione e la ricerca all'interno della Web Application.

Riguardo la gestione degli utenti (§ 3.3.2.3), la sua registrazione *in loco* e il rilascio di un codice utente univoco si dimostra un mezzo efficiente in termini di praticità e sicurezza in quanto, oltre all'immissione delle proprie generalità, non sorge la necessità di effettuare login. Questo meccanismo è stato pensato per incentivare una certa praticità e sicurezza dei dati perché la compilazione del form è volatile: eppure può risultare noioso e dispendioso a livello temporale, per cui sarebbe utile, come specificato nel § 3.3.3, implementare dei *cookie*[21] che salvano temporaneamente i dati in locale.

Si faccia un appunto anche sui prestiti (§ 3.3.2.5): se la dimensione diventasse molto grande, il sistema potrebbe a sua volta rallentarsi perché i dati da processare sarebbero molti di più. Sarebbe utile trovare un modo per rendere la tabella *Prestiti* più efficiente.

Nel complesso, sebbene si tratti del primo tentativo di sviluppo di una Web Application di questo tipo, senza conoscenze pregresse nell'ambito dello sviluppo Back-end (§ 3.1), si tratta di un software di un livello abbastanza buono. Tuttavia, affinché il database sia compatibile con quello di Bibliolandia, è necessario prestare maggiore attenzione alla pubblicazione in quanto punto di riferimento, recuperarne sia informazioni tecniche che di contenuto, trovare un modo per trasformare una semplice tabella in un vero e proprio catalogo. Si tratta di un lavoro accettabile, ma bisogna fare un *upgrade* in questa direzione.

5. Conclusioni

In questo progetto è stata sviluppata una Web Application in grado di gestire la biblioteca di un istituto scolastico. Dopo un'attenta analisi delle specifiche è seguita una progettazione concettuale da cui è stato possibile ricavare lo schema del sito: su esso sono state fatte opportune valutazioni di carattere tecnico e un'analisi delle ridondanze e delle prestazioni del database. Dopodiché si sono illustrate le fasi di sviluppo della Web Application, descrivendo sia le funzionalità principali che in modo più dettagliato i meccanismi di fondo, sia lato client che lato server.

Infine, si è svolta una valutazione in ambito biblioteconomico per stabilire se il database si possa definire un catalogo e quali sono le opportune osservazioni e migliorie da adottare affinché possa considerarsi tale. In relazione a ciò si è dimostrato se fosse possibile o meno una integrazione alla rete Bibliolandia effettuando opportune valutazioni in merito.

Il sistema sembra fluido e stabile, sebbene si possa investire di più sulle prestazioni e sulle funzionalità offerte. Lo stile rispecchia in gran parte il contesto di riferimento e il target di utenti coinvolti: si mostra colorato e allo stesso tempo semplice nella consultazione e nell'utilizzo.

Per poter ovviare tutte le problematiche riscontrate in caso di verifica verranno rilasciati di volta in volta degli aggiornamenti di *bugfix*, di ottimizzazione delle prestazioni, di maggiore flessibilità del codice e il rilascio di nuove *features* richieste dall'istituto scolastico.

Come è stato trattato nel § 4.3, a livello catalografico bisogna mettere la voce bibliografica al centro di tutto il database e arricchirlo con più metadati rendendolo maggiormente consultabile da parte dell'utente.

Apportando questo tipo di miglioramenti, il database creato può considerarsi un OPAC *completo* e quindi compatibile con una possibile integrazione a Bibliolandia.

A titolo personale, sono dovuti sentiti ringraziamenti a tutti i miei familiari, i miei amici, ma soprattutto alla mia famiglia per tutti i sacrifici affrontati, nell'avermi supportato fino in fondo in tutto il percorso.

6. Bibliografia

- Mehdi Achour [*et al.*], *PHP Manual*, edited by Peter Cowburn, "PHP", 2022, https://www.php.net/manual/en/index.php.

- Paolo Atzeni [*et al.*], *Basi di dati: modelli e linguaggi di interrogazione – Quarta edizione*, Milano, McGraw-Hill, 2013.

- Galluzzi A., *Biblioteche e cooperazione. Modelli, strumenti, esperienze in Italia*, Milano, Editrice Bibliografica, 2004.

- Gambari S., Guerrini M., *Definire e catalogare le risorse elettroniche. Un'introduzione a ISBD(ER), AACR2 e metadati*, Milano, Editrice Bibliografica, 2002.

- *Guida alla biblioteconomia* / a cura di Mauro Guerrini; con Gianfranco Grupi e Stefano Gambari; collaborazione di Vincenzo Fugaldi, Milano, Editrice Bibliografica, 2008, p. 22-23, 38-39.

- International Federation of Library Associations and Institutions, Study Group on the Functional Requirements for Bibliographic Records, *Functional requirements for bibliographic records: final report*, approved by the Standing Committee of the IFLA Section on Cataloguing, München, Saur, 1998.

- *jQuery API*, a cura della OpenJS Foundation, "jQuery", 2021, https://api.jquery.com/.

- Petrucciani A., Turbanti S., *Manuale di catalogazione: principi, casi e problemi*, Milano, Editrice Bibliografica, Associazione Italiana Biblioteche, 2021, p. 11-27, 34-54, 64-67.

- *Regole italiane di catalogazione: REICAT*, a cura della Commissione permanente per la revisione delle regole italiane di catalogazione, Roma, ICCU, 2009.

- Simona Turbanti, *REICAT*, Roma, Associazione Italiana Biblioteche, 2016.

7. Appendice

7.1. Contenuto del `try/catch` nel file di creazione del database (§ 3.2)

```
try {
    $db = new PDO("mysql:host=$servername", $username, $password);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $database = "CREATE DATABASE IF NOT EXISTS $db_name;
                 USE $db_name;

                 CREATE TABLE plesso (
                     Codice VARCHAR(10) PRIMARY KEY,
                     Tipo TINYINT NOT NULL DEFAULT '1',
                     Nome VARCHAR(30),
                     Indirizzo VARCHAR(80),
                     Telefono VARCHAR(10),
                     Email VARCHAR(50)

                     ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

                 CREATE TABLE libro (
                     ID INT AUTO_INCREMENT PRIMARY KEY,
                     Titolo VARCHAR(60) NOT NULL,
                     Autore VARCHAR(60) NOT NULL,
                     Editore VARCHAR(60),
                     Anno VARCHAR(4),
                     Collocazione VARCHAR(6) NOT NULL,
                     Plesso VARCHAR(10) DEFAULT '-',
                     Stato CHAR DEFAULT 'D',

                     UNIQUE(Collocazione, Plesso),

                     FOREIGN KEY (Plesso)
                         REFERENCES plesso(Codice)
                         ON UPDATE CASCADE
                         ON DELETE CASCADE

                     ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

                 ALTER TABLE libro AUTO_INCREMENT = 0;

                 CREATE TABLE utente (
                     ID VARCHAR(6) PRIMARY KEY,
                     Tipo CHAR NOT NULL DEFAULT 'S',
                     Cognome VARCHAR(30) NOT NULL,
                     Nome VARCHAR(30) NOT NULL,
                     DataNascita DATE NOT NULL,
                     Telefono VARCHAR(10) NOT NULL,
                     Classe CHAR DEFAULT NULL,
                     Sezione CHAR DEFAULT NULL,
                     Plesso VARCHAR(10),
```

```
    FOREIGN KEY (Plesso)
        REFERENCES plesso(Codice)
        ON UPDATE CASCADE
        ON DELETE CASCADE

    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE operatore (
    Username VARCHAR(20) PRIMARY KEY,
    Email VARCHAR(40) NOT NULL UNIQUE,
    Pass VARCHAR(255) NOT NULL,
    Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
    Tipo CHAR NOT NULL DEFAULT 'A',
    Plesso VARCHAR(10),

    FOREIGN KEY (Plesso)
        REFERENCES plesso(Codice)
        ON UPDATE CASCADE
        ON DELETE CASCADE

    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE prestito (
    ID VARCHAR(8) PRIMARY KEY,
    Libro INT,
    Utente VARCHAR(6),
    Operatore VARCHAR(20),
    DataPrestito DATE DEFAULT NULL,
    DataRestituzione DATE DEFAULT NULL,
    Plesso VARCHAR(10),
    Stato CHAR NOT NULL DEFAULT '-',

    FOREIGN KEY (Libro)
        REFERENCES libro(ID)
        ON UPDATE CASCADE
        ON DELETE SET NULL,

    FOREIGN KEY (Utente)
        REFERENCES utente(ID)
        ON UPDATE CASCADE
        ON DELETE SET NULL,

    FOREIGN KEY (Operatore)
        REFERENCES operatore(Username)
        ON UPDATE CASCADE
        ON DELETE SET NULL,

    FOREIGN KEY (Plesso)
        REFERENCES plesso(Codice)
        ON UPDATE CASCADE
        ON DELETE CASCADE

    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
                        CREATE TABLE log (
                            ID INT AUTO_INCREMENT,
                            DataIns DATE NOT NULL,
                            OraIns TIME NOT NULL,
                            Azione CHAR NOT NULL,
                            Tipo CHAR NOT NULL,
                            Elemento VARCHAR(20) NOT NULL,
                            Admin VARCHAR(20) NOT NULL,

                            PRIMARY KEY(ID, Elemento),

                            FOREIGN KEY (Admin)
                                REFERENCES operatore(Username)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE

                            ) ENGINE=InnoDB DEFAULT CHARSET=utf8; \n\n;";

    $db->exec($database);

}

catch (PDOException $e) {
    echo "Errore in $database: <br />" . $e->getMessage();
    die();
}
```