Description

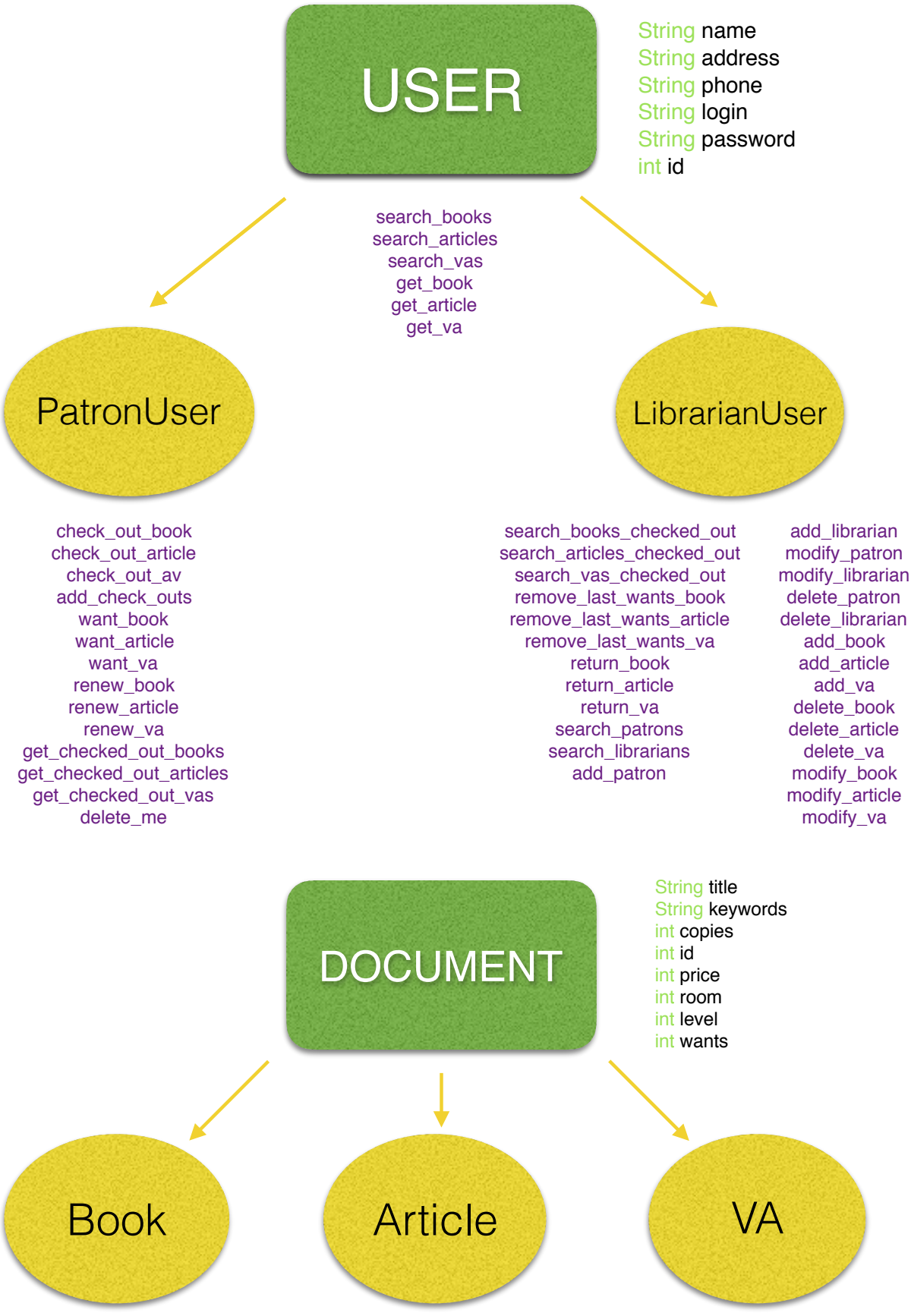
Our team

Nikolai Mikriukov

Nikola Novarlic

Ilnur Garipov

Kamil Shakirov


USER
String name
String address
String phone
String login
String password
int id
search_books
search_articles
search_vas
get_book
get_article
get_va
PatronUser
LibrarianUser
check_out_book
check_out_article
check_out_av
add_check_outs
want_book
want_article
want_va
renew_book
renew_article
renew_va
get_checked_out_books
get_checked_out_articles
get_checked_out_vas
delete_me
search_books_checked_out
search_articles_checked_out
search_vas_checked_out
remove_last_wants_book
remove_last_wants_article
remove_last_wants_va
return_book
return_article
return_va
search_patrons
search_librarians
add_patron
add_librarian
modify_patron
modify_librarian
delete_patron
delete_librarian
add_book
add_article
add_va
delete_book
delete_article
delete_va
modify_book
modify_article
modify_va
DOCUMENT
String title
String keywords
int copies
int id
int price
int room
int level
int wants
Book
Article
VA

Class User:

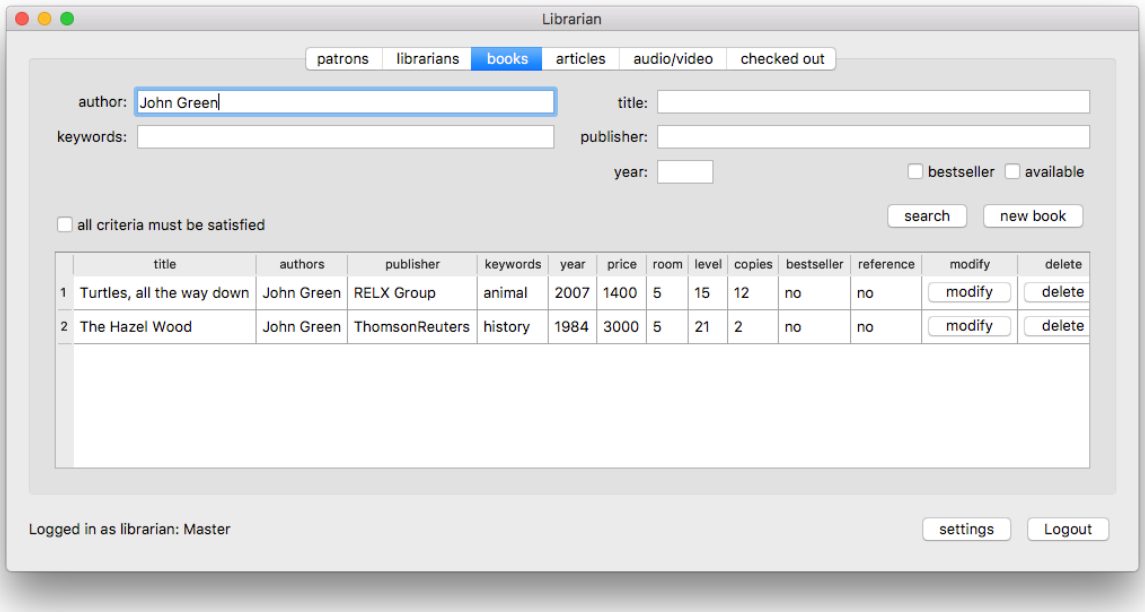
- search_books, search_articles, search_vas

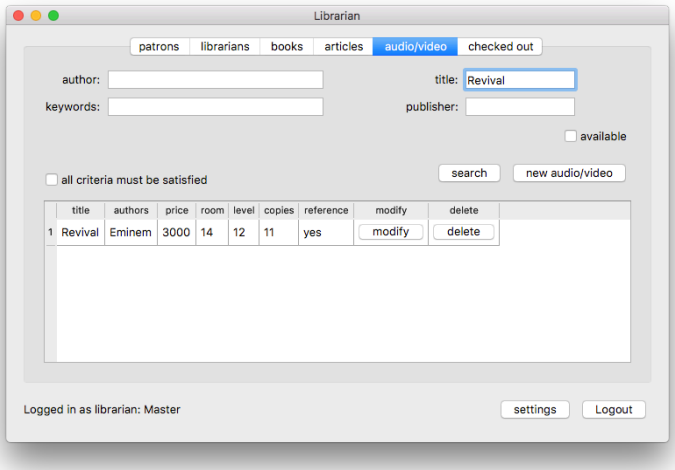
These three methods work both in LibrarianUser and in PatronUser. They search documents/vas in the system for various parameters. Here is an example of the code of one of the methods

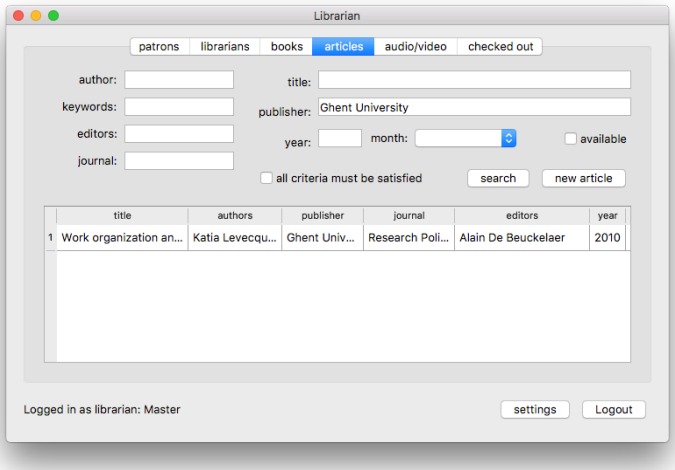
```
QVector<Book> search_books(QString authors, QString title, QString keywords, QString publisher, int year, bool bestseller, bool available, bool or_and){
    QSqlQuery query;
    QString ins = or_and ? " AND " : " OR ";
    authors = authors.toLower();
    title = title.toLower();
    keywords = keywords.toLower();
    publisher = publisher.toLower();
    QString req = "SELECT * FROM books WHERE ";
    if (authors != "") req += "instr(lower(authors), '"+authors+"') > 0" + ins;
    if (title != "") req += "instr(lower(title), '"+title+"') > 0" + ins;
    if (keywords != "") req += "instr(lower(keywords), '"+keywords+"') > 0" + ins;
    if (publisher != "") req += "instr(lower(publisher), '"+publisher+"') > 0" + ins;
    if (year != 0) req += "instr(year, '"+QString::number(year)+"') > 0" + ins;
    if (bestseller) req += "bestseller = 1" + ins;
    if (available) req += "copies > 0" + ins;
    req += "1 = " + QString(or_and ? "1" : "0");//nice hack to finish statement correctly
    if (req.length() == 31)//no parameters given
        req = "SELECT * FROM books";
    query.exec(req);
    QVector<Book> ans;
    while (query.next()) {
        int id = query.value(0).toInt();
        QString title = query.value(1).toString();
        QString authors = query.value(2).toString();
        QString publisher = query.value(3).toString();
        QString keywords = query.value(4).toString();
        int year = query.value(5).toInt();
        int price = query.value(6).toInt();
        int room = query.value(7).toInt();
        int level = query.value(8).toInt();
        int copies = query.value(9).toInt();
        bool bestseller = query.value(10).toInt();
        bool reference = query.value(11).toInt();
        QString wants_str = query.value(12).toString();
        ans.push_back(Book(authors, title, keywords, publisher, id, year, copies, price, room ,level, bestseller, reference, wants_str));
    }
    return ans;
}
```

Class User:

- search_books, search_articles, search_vas







Class User:

- get_book, get_article, get_va

These three methods return the document/va by its identifier. Here is an example of the code of one of the methods

```
Book get_book(int book_id){
    QSqlQuery query;
    query.exec("SELECT * FROM books WHERE id = " + QString::number(book_id));
    query.next();
    QString title = query.value(1).toString();
    QString authors = query.value(2).toString();
    QString publisher = query.value(3).toString();
    QString keywords = query.value(4).toString();
    int year = query.value(5).toInt();
    int price = query.value(6).toInt();
    int room = query.value(7).toInt();
    int level = query.value(8).toInt();
    int copies = query.value(9).toInt();
    bool bestseller = query.value(10).toInt();
    bool reference = query.value(11).toInt();
    QString wants_str = query.value(12).toString();
    return Book(authors, title, keywords, publisher, book_id, year, copies, price, room ,level, bestseller, reference, wants_str);
}
```

Class PatronUser:

- check_out_book, check_out_article, check_out_va

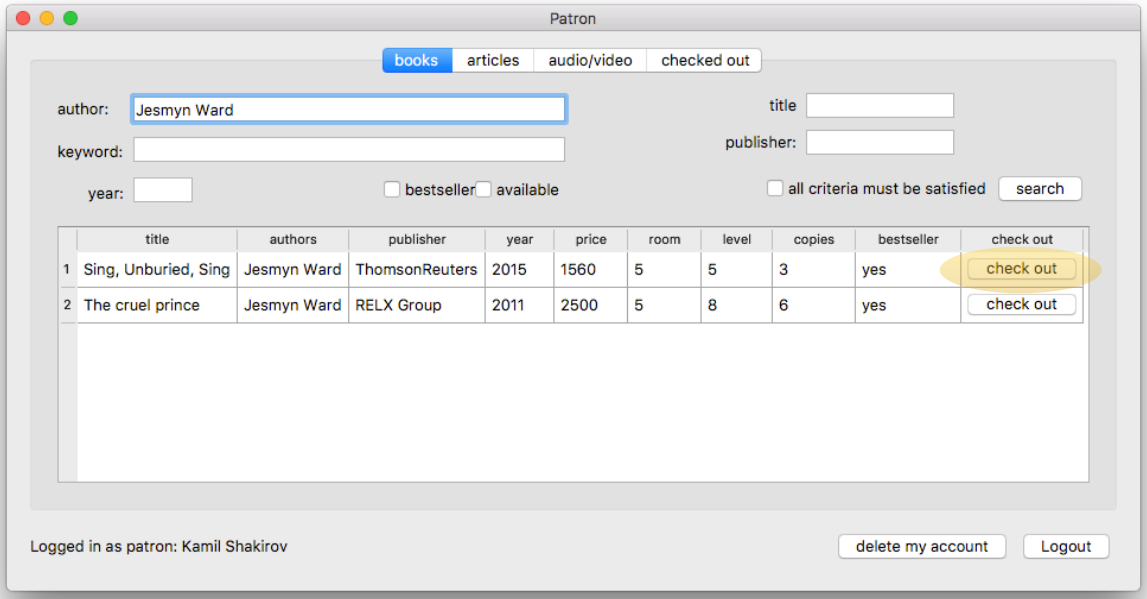
These three methods realize the booking of documents/va. Here is an example of the code of one of the methods

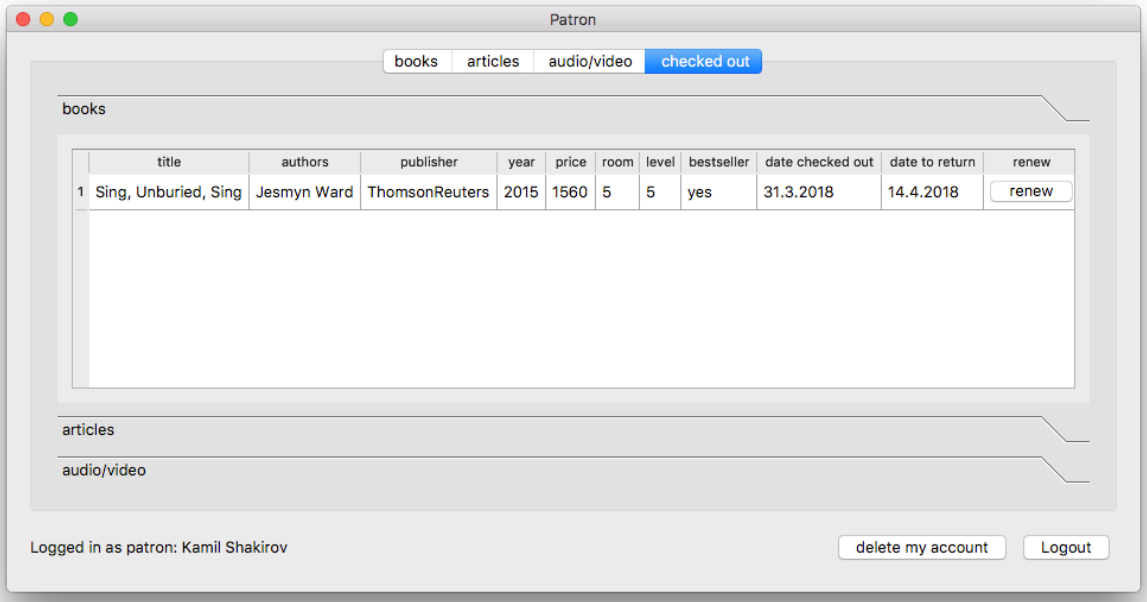
```
void check_out_book(int document_id, QDate *gdate = NULL){
    QSqlQuery query;
    query.prepare("SELECT COUNT(*) FROM check_outs WHERE user_id = :user_id AND document_type = 1 AND document_id = :document_id AND year_end IS NULL");
    query.bindValue(":user_id", id);
    query.bindValue(":document_id", document_id);
    query.exec();
    query.next();
    bool already = query.value(0).toInt();
    if(already) return; //cant check out twice

    query.exec("SELECT copies, reference from books WHERE id = " + QString::number(document_id));
    if (!query.next()) return;//book doesnt exist
    int copies = query.value(0).toInt();
    bool reference = query.value(1).toInt() ;
    if (copies <= 0 || reference) return;

    QDate date;
    if(gdate == NULL)
        date = QDate::currentDate();
    else
        date = *gdate;
    int year_start = date.year();
    int month_start = date.month();
    int day_start = date.day();

    query.prepare("UPDATE books SET copies = copies - 1 WHERE id = :document_id");
    query.bindValue(":document_id", document_id);
    query.exec();

    query.prepare("INSERT INTO check_outs (user_id,document_type,document_id,year_start,month_start,day_start) VALUES(:user_id,:document_type,:document_id,:year_start,:month_start,:day_start)");
    query.bindValue(":user_id", id);
    query.bindValue(":document_type", BOOK);
    query.bindValue(":document_id", document_id);
    query.bindValue(":year_start", year_start);
    query.bindValue(":month_start", month_start);
    query.bindValue(":day_start", day_start);
    query.exec();

    //get last check out id
    query.exec("SELECT check_out_id FROM check_outs ORDER BY check_out_id DESC LIMIT 1");
    query.next();
    int check_out_id = query.value(0).toInt();

    query.prepare("UPDATE patrons SET check_outs = check_outs || :check_out_id WHERE id = :user_id");
    query.bindValue(":check_out_id", QString::number(check_out_id)+";");
    query.bindValue(":user_id", id);
    query.exec();
}
```

Class PatronUser:

- check_out_book, check_out_article, check_out_va

If user wants to reserve the document/va, and it is available, he clicks on the «check out" button and then this document/va is added the list «checked out»



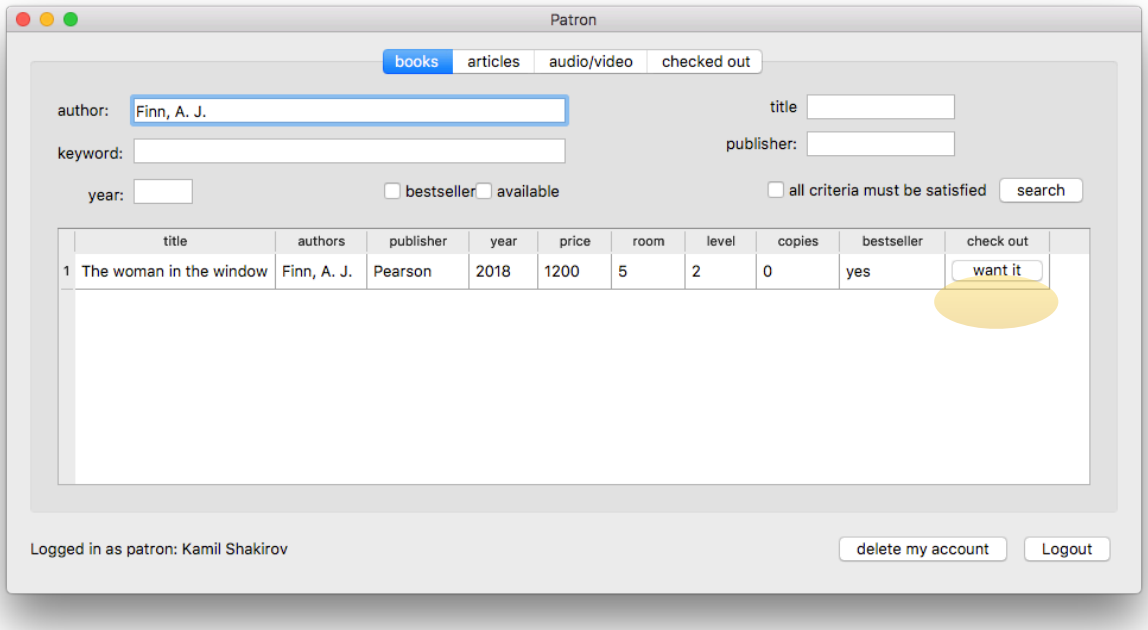


Class PatronUser:

- want_book, want_article, want_va

If user wants to reserve the document/va, but it is unavailable he clicks on the «want it» button and then this user is added to the queue to get this document/va. Here is an example of the code of one of the methods

```
void want_book(int document_id){
    //set renew_state to 1
    QSqlQuery query;
    query.prepare("UPDATE check_outs SET renew_state = 1 WHERE document_type = 1 AND renew_state = 0 AND year_end IS NULL AND document_id = :document_id");
    query.bindValue(":document_id", document_id);
    query.exec();

    Book book = get_book(document_id);
    if (!book.wants.contains(id)){
        query.prepare("UPDATE books SET wants = wants || :user_id WHERE id = :document_id");
        query.bindValue(":user_id", QString::number(id)+";");
        query.bindValue(":document_id", document_id);
        query.exec();
    }
}
```

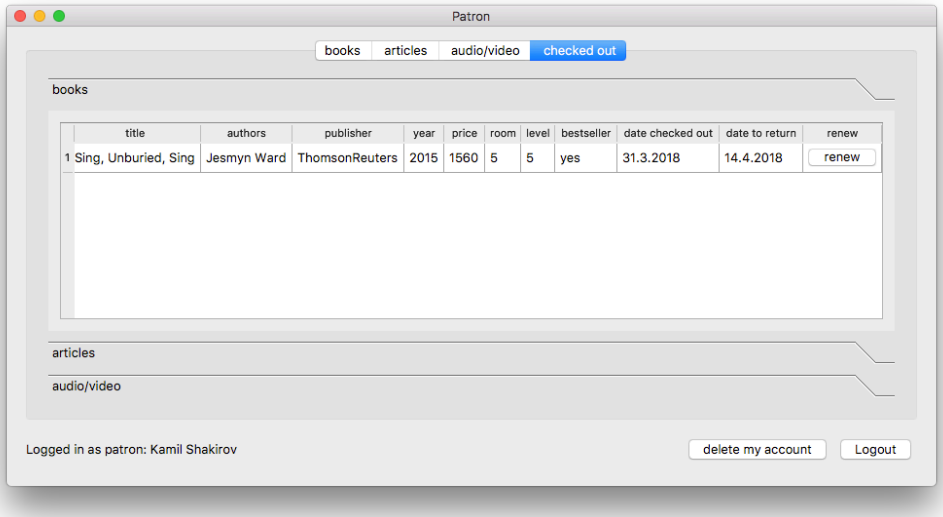


Class PatronUser:

- renew_book, renew_article, renew_va

If user wants to extend the deadline for the document/va, and there are no other users in the booking queue for this document/va, user can click on the "renew" button and extend the deadline for the document/va for a certain period,

```
int renew_book(int check_out_id){
    QSqlQuery query;
    QDate today = QDate::currentDate();

    query.prepare("SELECT * FROM check_outs WHERE check_out_id = :check_out_id");
    query.bindValue(":check_out_id", check_out_id);
    query.exec();
    if (!query.next()) return 0; //no such check out

    int book_id = query.value(3).toInt();
    int year_start = query.value(4).toInt();
    int month_start = query.value(5).toInt();
    int day_start = query.value(6).toInt();
    int renew_state = query.value(10).toInt();

    if (renew_state != 0) return 1; //someone wants

    Book book = get_book(book_id);
    std::pair<QDate, int> end = calculate_check_out(1, year_start, month_start, day_start, faculty, book.bestseller, book.price, renew_state);

    if (today.daysTo(end.first) != 0 && today.daysTo(end.first) != 1)
        return 2; //too late or early

    query.prepare("UPDATE check_outs SET renew_state = 2 WHERE check_out_id = :check_out_id");
    query.bindValue(":check_out_id", check_out_id);
    query.exec();
    return 3;
}
```

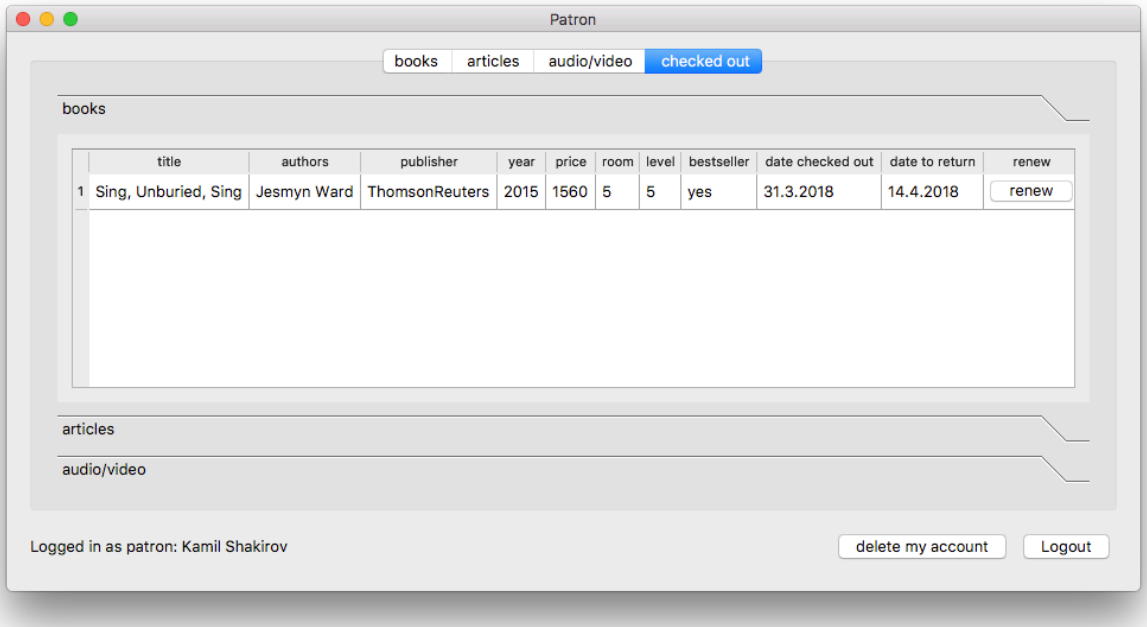


Class PatronUser:

- get_check_out_book, get_check_out_article, get_check_out_va

This method shows your checked out documents/vas

```
QVector<std::pair<Check_out, Book> > get_checked_out_books(){
    QSqlQuery query;
    QVector<std::pair<Check_out, Book> > ans;
    query.exec("SELECT * FROM check_outs WHERE document_type = 1 AND year_end IS NULL AND user_id = " + QString::number(id));
    while (query.next()) {
        int check_out_id = query.value(0).toInt();
        int book_id = query.value(3).toInt();
        int year_start = query.value(4).toInt();
        int month_start = query.value(5).toInt();
        int day_start = query.value(6).toInt();
        int state_renew = query.value(10).toInt();
        Book book = get_book(book_id);
        std::pair<QDate, int> end = calculate_check_out(1, year_start, month_start, day_start, faculty, book.bestseller, book.price, state_renew);
        ans.push_back(make_pair(Check_out(id, 1, book_id, check_out_id, year_start, month_start, day_start, end.first.year(), end.first.month(), end.first.day(), end.second), book));
    }
    return ans;
}
```

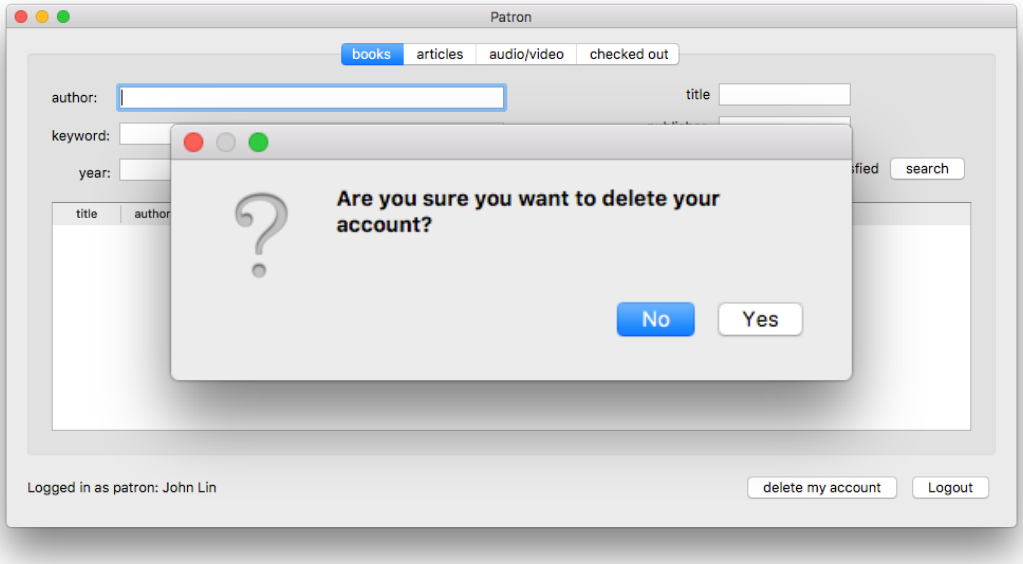


Class PatronUser:

- delete_me

Account deleting

```
bool delete_me(){
    QSqlQuery query;
    if (check_outs.size() > 0)
        return 0; //cant delete if has check outs
    query.exec("DELETE FROM patrons WHERE id = " + QString::number(id));
    return 1;
}
```

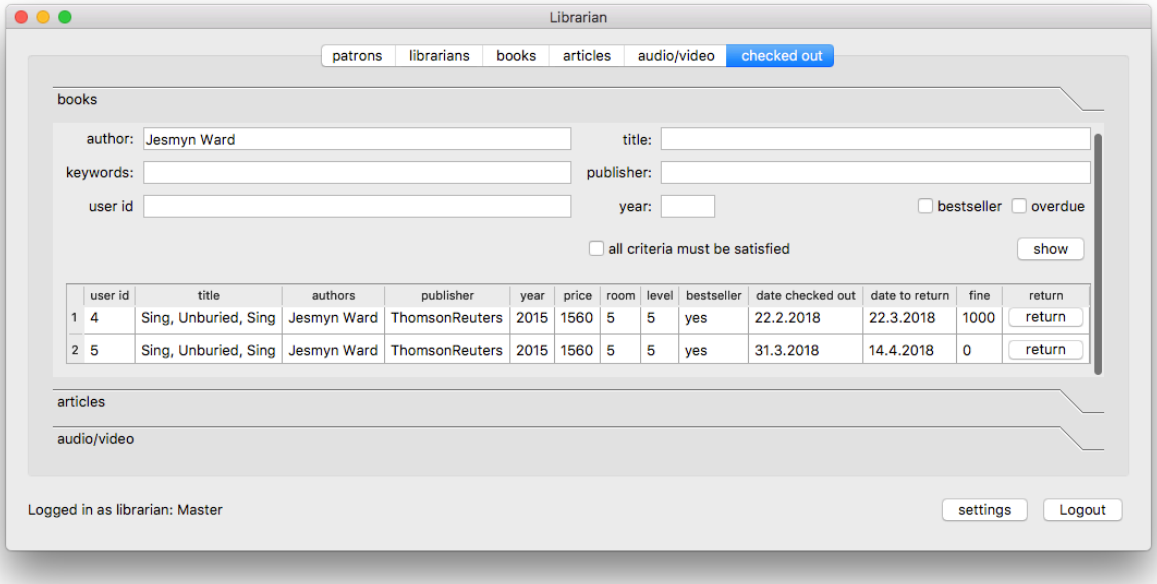


Class LibrarianUser:

- search_books_checked_out, search_articles_checked_out, search_vas_checked_out

Methods for searching checked out documents/vas

```
QVector<std::pair<Check_out, Book> > search_books_checked_out(int user_id, QString authors, QString title, QString keywords, QString publisher, int year, bool bestseller, bool overdue, bool or_and){
    QSqlQuery query;
    QVector<Book> books = search_books(authors, title, keywords, publisher, year, bestseller, 0, or_and);
    QVector<std::pair<Check_out, Book> > ans;
    for (int i = 0; i < books.size(); i++){
        QString req = "SELECT * FROM check_outs WHERE document_type = 1 AND year_end IS NULL AND document_id = " + QString::number(books[i].id);
        if (user_id != 0) req += QString(or_and ? " AND" : " OR") + " user_id = " + QString::number(user_id);
        query.exec(req);
        while (query.next()) {
            int check_out_id = query.value(0).toInt();
            int current_user_id = query.value(1).toInt();
            int book_id = query.value(3).toInt();
            int year_start = query.value(4).toInt();
            int month_start = query.value(5).toInt();
            int day_start = query.value(6).toInt();
            int state_renew = query.value(10).toInt();
            Book book = get_book(book_id);
            PatronUser patron = get_patron(current_user_id);
            std::pair<QDate, int> end = calculate_check_out(1, year_start, month_start, day_start, patron.faculty, book.bestseller, book.price, state_renew);
            QDate return_date;
            return_date.setDate(end.first.year(), end.first.month(), end.first.day());
            if (overdue && QDate::currentDate().daysTo(return_date) >= 0) continue;
            ans.push_back(make_pair(Check_out(current_user_id, 1, book_id, check_out_id, year_start, month_start, day_start, end.first.year(), end.first.month(), end.first.day(), end.second), book));
        }
    }
    return ans;
}
```

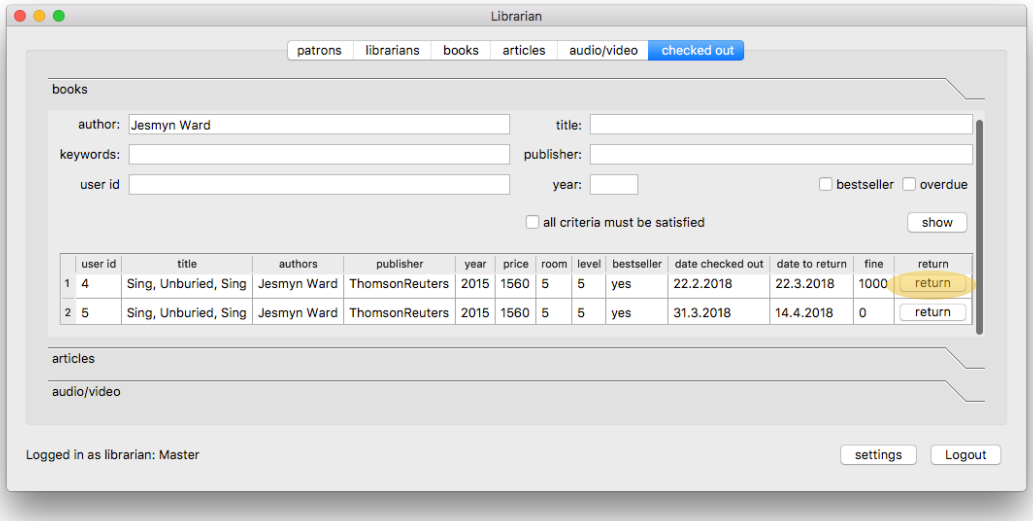


Class LibrarianUser:

- return_book, return_article, return_va

Returning system

```
std::pair<int, int> return_book(int check_out_id){
    QSqlQuery query;
    query.exec("SELECT * FROM check_outs WHERE document_type = 1 AND check_out_id = " + QString::number(check_out_id));
    if (!query.next()) return make_pair(-1, -1);
    int user_id = query.value(1).toInt();
    int book_id = query.value(3).toInt();
    int year_start = query.value(4).toInt();
    int month_start = query.value(5).toInt();
    int day_start = query.value(6).toInt();
    int renew_state = query.value(10).toInt();

    PatronUser patron = get_patron(user_id);
    Book book = get_book(book_id);
    std::pair<QDate, int> end = calculate_check_out(1, year_start, month_start, day_start, patron.faculty, book.bestseller, book.price, renew_state);
    int fine = end.second;

    query.prepare("UPDATE check_outs SET year_end = :year_end, month_end = :month_end, day_end = :day_end WHERE check_out_id = :check_out_id");
    query.bindValue(":check_out_id", check_out_id);
    query.bindValue(":year_end", end.first.year());
    query.bindValue(":month_end", end.first.month());
    query.bindValue(":day_end", end.first.day());
    query.exec();

    query.prepare("UPDATE patrons SET check_outs = replace(check_outs, :check_out_id_str, '') WHERE id = :user_id");
    query.bindValue(":check_out_id_str", ";" + QString::number(check_out_id));
    query.bindValue(":user_id", user_id);
    query.exec();

    query.prepare("UPDATE books SET copies = copies + 1 WHERE id = :document_id");
    query.bindValue(":document_id", book.id);
    query.exec();

    query.prepare("DELETE FROM check_outs WHERE check_out_id = :check_out_id");
    query.bindValue(":check_out_id", check_out_id);
    query.exec();

    int wants_id = remove_last_wants_book(book.id);
    if (wants_id != -1)
        return make_pair(fine, wants_id);
    return make_pair(fine, -1);
}
```

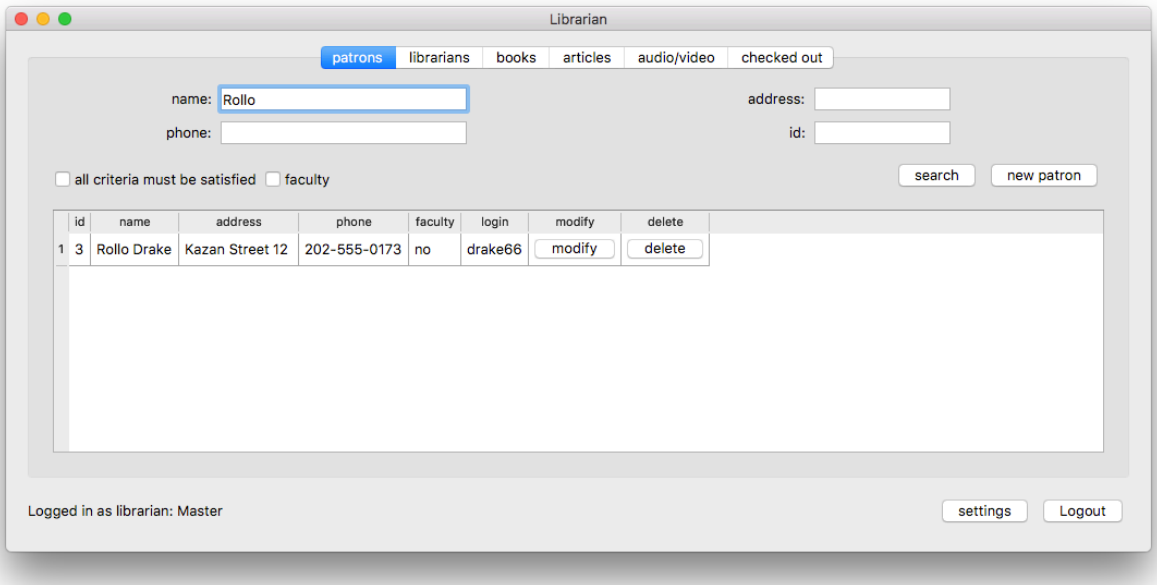


Class LibrarianUser:

- search_patrons, search_librarians

These methods are search users in the system

```
QVector<PatronUser> search_patrons(int user_id, QString name, QString address, QString phone, bool faculty, bool or_and){
    QSqlQuery query;
    QString ins = or_and ? " AND " : " OR ";
    name = name.toLower();
    address = address.toLower();
    phone = phone.toLower();
    QString req = "SELECT * FROM patrons WHERE ";
    if (user_id != 0) req += "id = " + QString::number(user_id) + ins;
    if (name != "") req += "instr(lower(name), '"+name+"') > 0" + ins;
    if (address != "") req += "instr(lower(address), '"+address+"') > 0" + ins;
    if (phone != "") req += "instr(lower(phone), '"+phone+"') > 0" + ins;
    if (faculty) req += "faculty = 1" + ins;
    req += "1 = " + QString(or_and ? "1" : "0");//nice hack to finish statement correctly
    if (req.length() == 33)//no parameters given
        req = "SELECT * FROM patrons";
    query.exec(req);
    QVector<PatronUser> ans;
    while (query.next()) {
        int user_id = query.value(0).toInt();
        QString name = query.value(1).toString();
        QString address = query.value(2).toString();
        QString phone = query.value(3).toString();
        bool faculty = query.value(4).toInt();
        QString check_outs = query.value(5).toString();
        QString login = query.value(6).toString();
        QString password = query.value(7).toString();
        ans.push_back(PatronUser(user_id, name, address, phone, faculty, login, password, check_outs));
    }
    return ans;
}
```

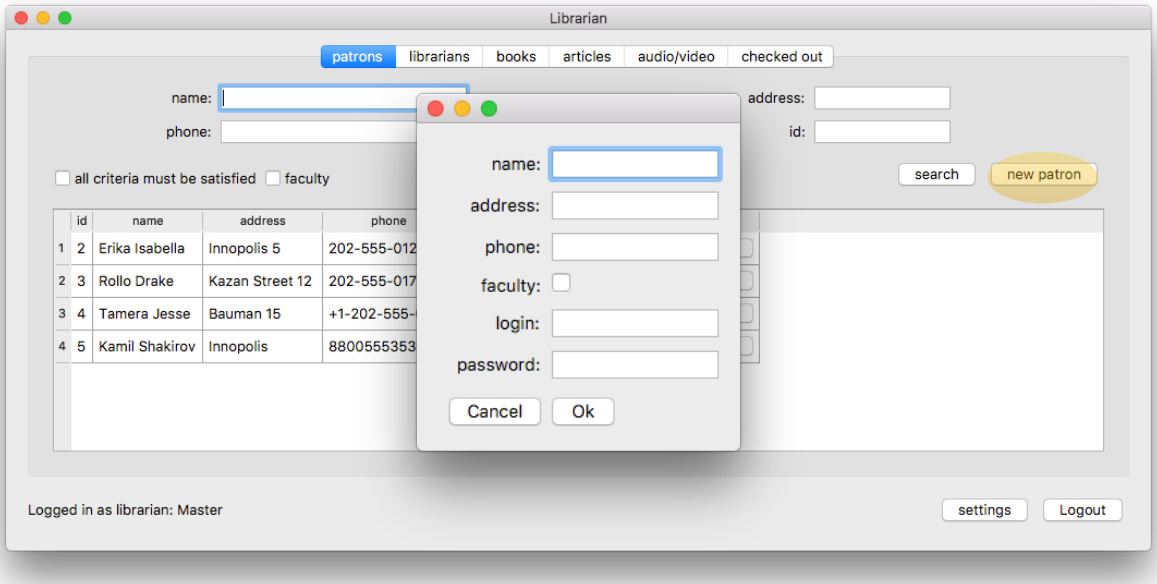


Class LibrarianUser:

- add_patron, add_librarian

With these methods, librarian can add new users

```
void add_patron(QString name, QString address, QString phone, bool faculty, QString login, QString password){
    password = Hasher::hash_password(login,password);
    QSqlQuery query;
    query.prepare("INSERT INTO patrons (name, address, phone, faculty, login, password) VALUES(:name, :address, :phone, :faculty, :login, :password)");
    query.bindValue(":name", name);
    query.bindValue(":address", address);
    query.bindValue(":phone", phone);
    query.bindValue(":faculty", (faculty ? 1 : 0));
    query.bindValue(":login", login);
    query.bindValue(":password", password);
    query.exec();
}
```

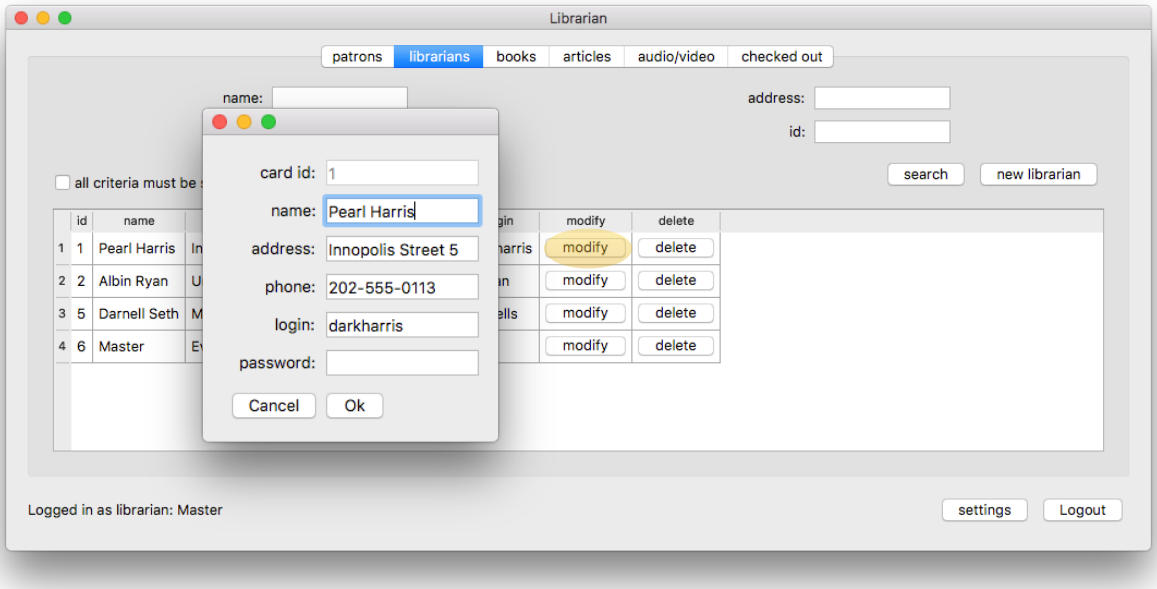


Class LibrarianUser:

- modify_patron, modify_librarian

With these methods, librarian can modify users

```
bool modify_librarian(int user_id, QString name, QString address, QString phone, QString login, QString password){
    password = Hasher::hash_password(login,password);
    QSqlQuery query;
    query.prepare("UPDATE librarians SET name = :name, address = :address, phone = :phone, login = :login, password = :password WHERE id = :user_id");
    query.bindValue(":name", name);
    query.bindValue(":address", address);
    query.bindValue(":phone", phone);
    query.bindValue(":login", login);
    query.bindValue(":password", password);
    query.bindValue(":user_id", user_id);
    query.exec();
    return 1;
}
```

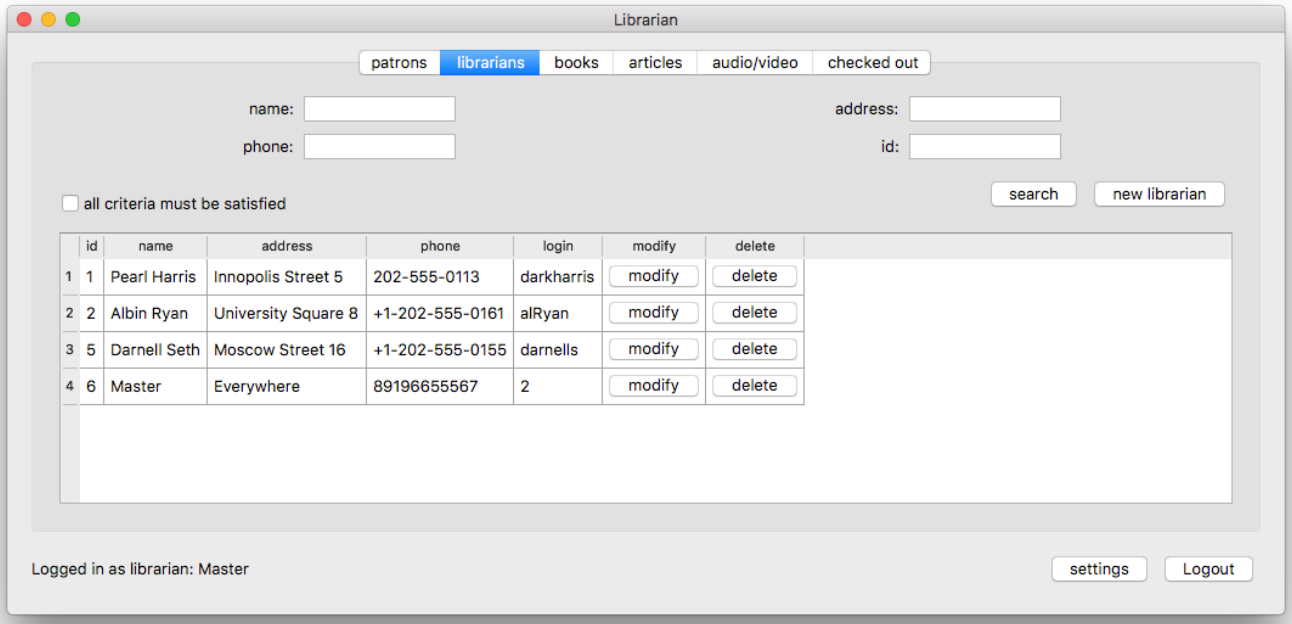


Class LibrarianUser:

- delete_patron, delete_librarian

Deleting users

```
bool delete_patron(int user_id){
    QSqlQuery query;
    if (get_patron(user_id).check_outs.size() > 0)
        return 0; //cant delete if has check outs
    query.exec("DELETE FROM patrons WHERE id = " + QString::number(user_id));
    return 1;
}
bool delete_librarian(int user_id){
    QSqlQuery query;
    query.exec("DELETE FROM librarians WHERE id = " + QString::number(user_id));
    return 1;
}
```

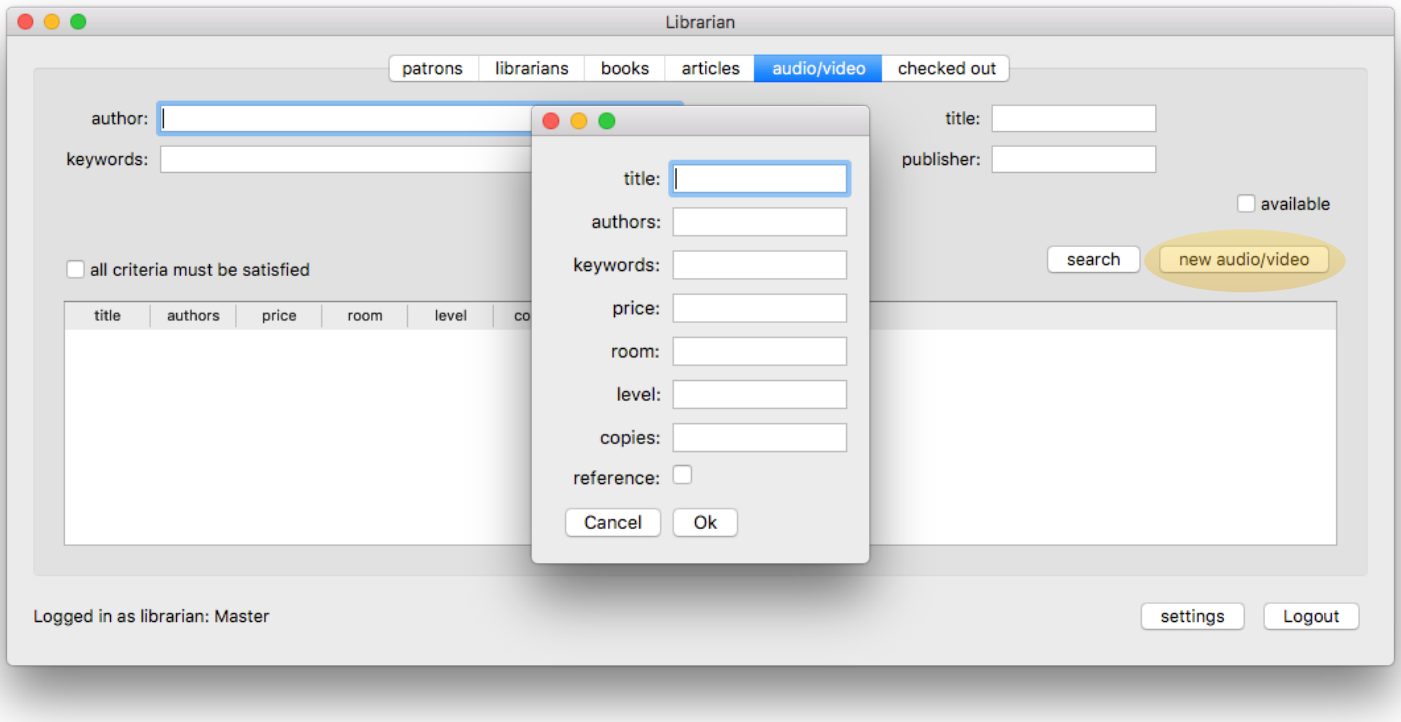


Class LibrarianUser:

- add_book, add_article, add_va

Methods for adding new document/va to the system

```
bool add_va(QString title, QString authors, QString keywords, int price, int room, int level, int copies, bool reference){
    QSqlQuery query;
    query.prepare("INSERT INTO vas (title, authors, keywords, price, room, level, copies, reference) VALUES(:title, :authors, :keywords, :price, :room, :level, :copies, :reference)");
    query.bindValue(":title", title);
    query.bindValue(":authors", authors);
    query.bindValue(":keywords", keywords);
    query.bindValue(":price", price);
    query.bindValue(":room", room);
    query.bindValue(":level", level);
    query.bindValue(":copies", copies);
    query.bindValue(":reference", reference);
    query.exec();
    return 1;
}
```

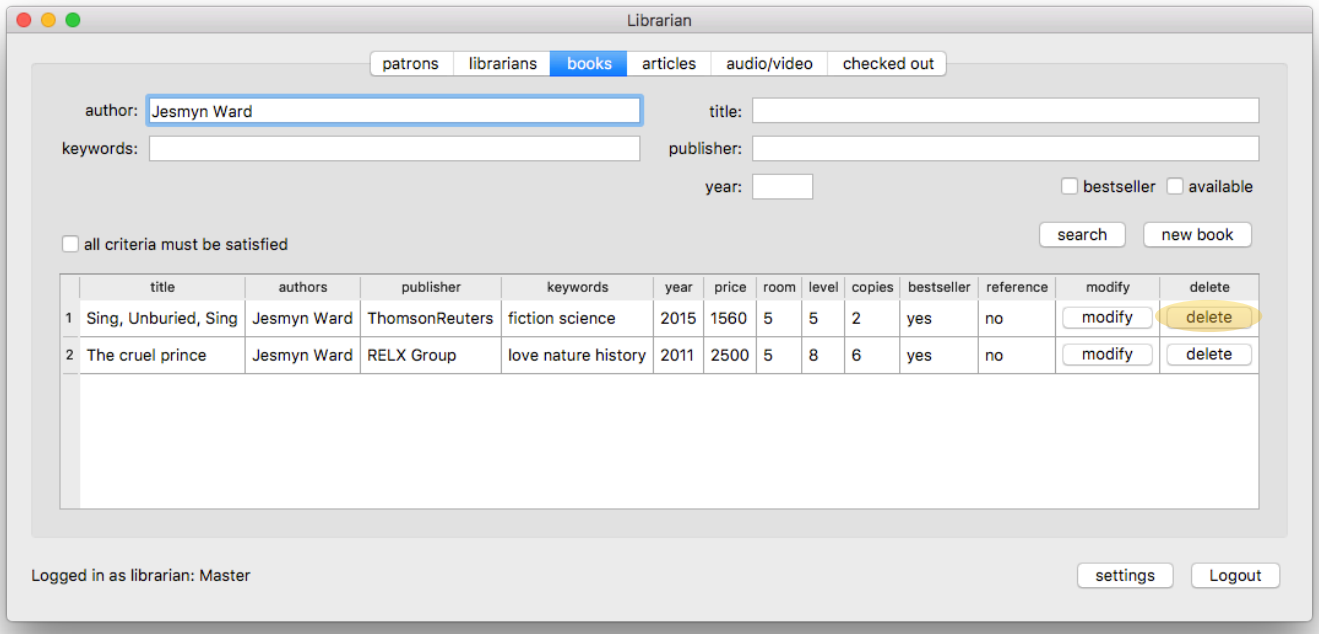


Class LibrarianUser:

- delete_book, delete_article, delete_va

Methods for deleting documents/vas from the system

```
bool delete_book(int document_id){
    QSqlQuery query;
    query.exec("DELETE FROM books WHERE id = " + QString::number(document_id));
    return 1;
}
bool delete_article(int document_id){
    QSqlQuery query;
    query.exec("DELETE FROM articles WHERE id = " + QString::number(document_id));
    return 1;
}
bool delete_va(int document_id){
    QSqlQuery query;
    query.exec("DELETE FROM vas WHERE id = " + QString::number(document_id));
    return 1;
}
```



Class LibrarianUser:

- modify_book, modify_article, modify_va

Methods for documents/vas modification

```
bool modify_va(int document_id, QString title, QString authors, QString keywords, int price, int room, int level, int copies, bool reference){
    QSqlQuery query;
    query.prepare("UPDATE vas SET title = :title, authors = :authors, keywords = :keywords, price = :price, room = :room, level = :level, copies = :copies, reference = :reference WHERE id = :document_id");
    query.bindValue(":title", title);
    query.bindValue(":authors", authors);
    query.bindValue(":keywords", keywords);
    query.bindValue(":price", price);
    query.bindValue(":room", room);
    query.bindValue(":level", level);
    query.bindValue(":copies", copies);
    query.bindValue(":document_id", document_id);
    query.bindValue(":reference", reference);
    query.exec();
    return 1;
}
```