

Drop booX

Description

Our team

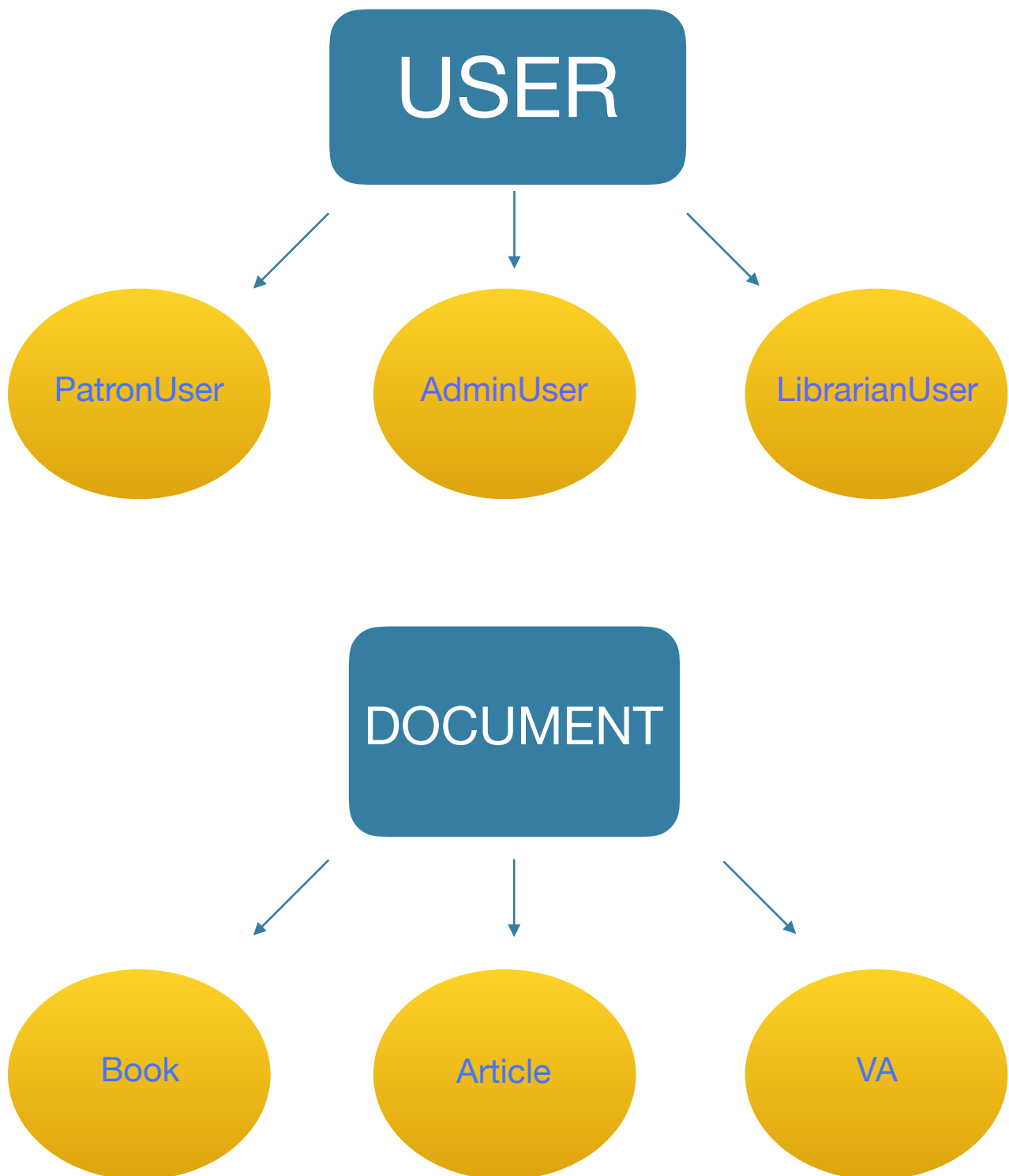
Nikolai Mikriukov

Nikola Novarlic

Ilnur Garipov

Kamil Shakirov





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`

Librarian

patrons librarians **books** articles audio/video checked out

author: title:

keywords: publisher:

year: ☐ bestseller ☐ available

☐ all criteria must be satisfied

	title	authors	publisher	keywords	year	price	room	level	copies	bestseller	reference	modify	delete
1	Turtles, all the way down	John Green	RELX Group	animal	2007	1400	5	15	12	no	no	<input type="button" value="modify"/>	<input type="button" value="delete"/>
2	The Hazel Wood	John Green	ThomsonReuters	history	1984	3000	5	21	2	no	no	<input type="button" value="modify"/>	<input type="button" value="delete"/>

Logged in as librarian: Master

Librarian

patrons librarians books articles **audio/video** checked out

author: title:

keywords: publisher:

☐ available

☐ all criteria must be satisfied

	title	authors	price	room	level	copies	reference	modify	delete
1	Revival	Eminem	3000	14	12	11	yes	<input type="button" value="modify"/>	<input type="button" value="delete"/>

Logged in as librarian: Master

Librarian

patrons librarians books **articles** audio/video checked out

author: title:

keywords: publisher:

editors: year: month: ☐ available

journal: ☐ all criteria must be satisfied

	title	authors	publisher	journal	editors	year
1	Work organization an...	Katia Levecqu...	Ghent Univ...	Research Poli...	Alain De Beuckelaer	2010

Logged in as librarian: Master

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»

Patron

books articles audio/video checked out

author: Jesmyn Ward title:

keyword: publisher:

year: ☐ bestseller ☐ available ☐ all criteria must be satisfied

	title	authors	publisher	year	price	room	level	copies	bestseller	check out
1	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	3	yes	<input type="button" value="check out"/>
2	The cruel prince	Jesmyn Ward	RELX Group	2011	2500	5	8	6	yes	<input type="button" value="check out"/>

Logged in as patron: Kamil Shakirov

Patron

books articles audio/video checked out

books

	title	authors	publisher	year	price	room	level	bestseller	date checked out	date to return	renew
1	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	yes	31.3.2018	14.4.2018	<input type="button" value="renew"/>

articles

audio/video

Logged in as patron: Kamil Shakirov

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();
    }
}
```

The screenshot shows a web application window titled "Patron". It has a navigation bar with tabs: "books" (selected), "articles", "audio/video", and "checked out". Below the tabs are search fields for "author:" (containing "Finn, A. J."), "keyword:", "year:", "title:", and "publisher:". There are also checkboxes for "bestseller", "available", and "all criteria must be satisfied", along with a "search" button. Below the search fields is a table with the following data:

	title	authors	publisher	year	price	room	level	copies	bestseller	check out
1	The woman in the window	Finn, A. J.	Pearson	2018	1200	5	2	0	yes	<input type="button" value="want it"/>

At the bottom of the window, it says "Logged in as patron: Kamil Shakirov" and has two buttons: "delete my account" and "Logout".

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

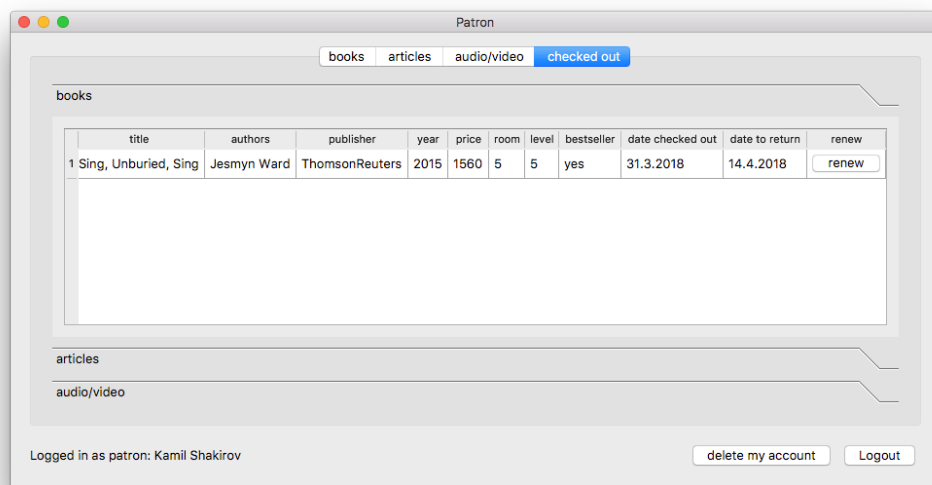
    int book_id = query.value(3).toInt();
    int year_start = query.value(4).toInt();
    int month_start = query.value(5).toInt();
    int day_start = query.value(6).toInt();
    int renew_state = query.value(10).toInt();

    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;
}
```

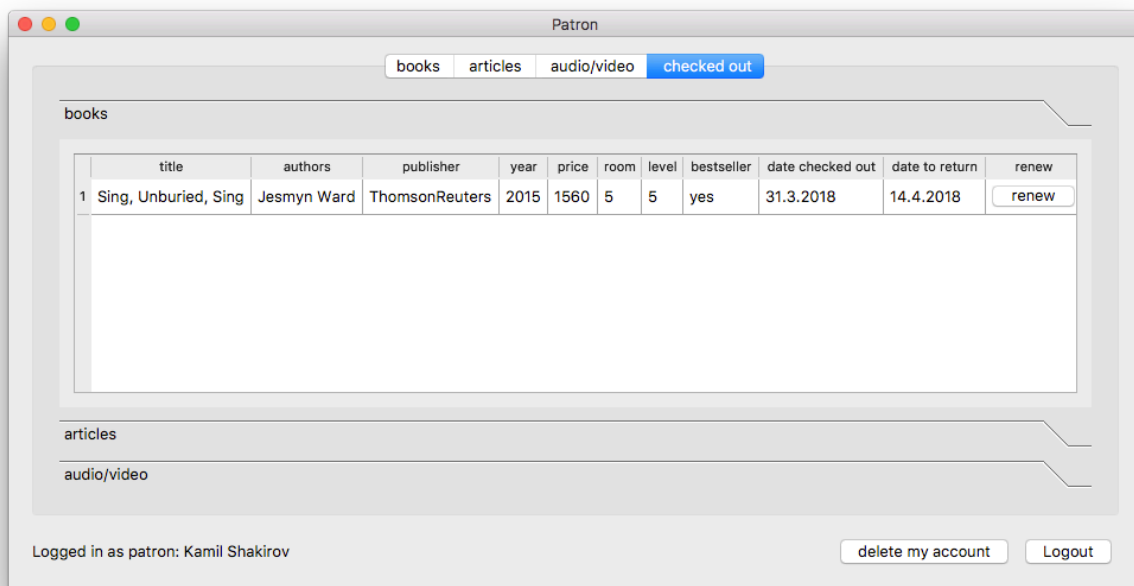


Class PatronUser:

- `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;
}
```

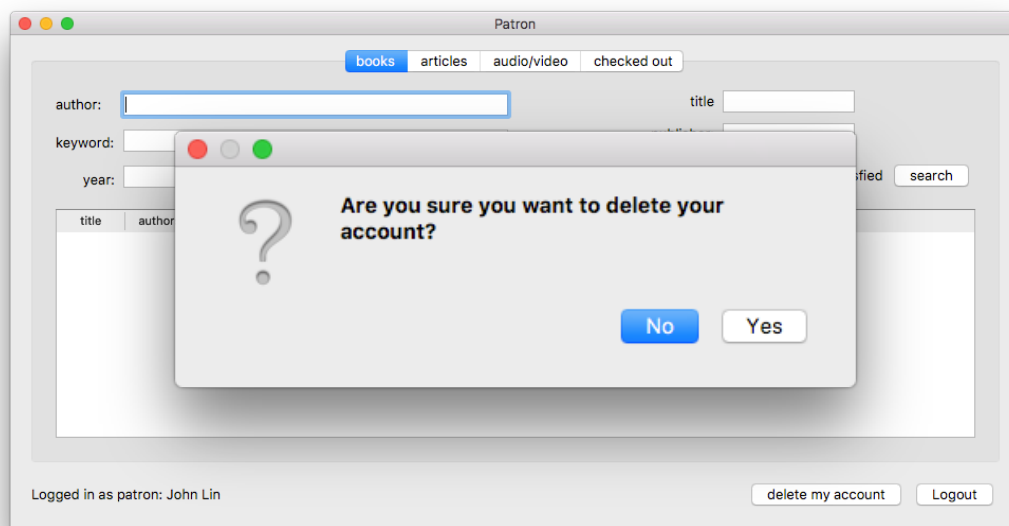


Class PatronUser:

- 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;
}
```

The screenshot shows the 'Librarian' application window with the 'checked out' tab selected. The interface includes search filters for books, a table of checked-out items, and navigation buttons for other sections like patrons, librarians, books, articles, and audio/video. The bottom status bar indicates the user is logged in as 'Master'.

Search Filters:

- author: Jesmyn Ward
- title:
- keywords:
- publisher:
- user id:
- year:
- ☐ bestseller ☐ overdue
- ☐ all criteria must be satisfied
-

Table of Checked-Out Books:

	user id	title	authors	publisher	year	price	room	level	bestseller	date checked out	date to return	fine	return
1	4	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	yes	22.2.2018	22.3.2018	1000	<input type="button" value="return"/>
2	5	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	yes	31.3.2018	14.4.2018	0	<input type="button" value="return"/>

Navigation and Status:

- Logged in as librarian: Master
-

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);
}
```

Librarian

patrons librarians books articles audio/video **checked out**

books

author: Jesmyn Ward title: _____

keywords: _____ publisher: _____

user id: _____ year: _____ ☐ bestseller ☐ overdue

☐ all criteria must be satisfied

	user id	title	authors	publisher	year	price	room	level	bestseller	date checked out	date to return	fine	return
1	4	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	yes	22.2.2018	22.3.2018	1000	<input type="button" value="return"/>
2	5	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	2015	1560	5	5	yes	31.3.2018	14.4.2018	0	<input type="button" value="return"/>

articles

audio/video

Logged in as librarian: Master

Class LibrarianUser:

- search_patrons

This method is search patrons 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;
}
```

The screenshot shows a window titled "Librarian" with several tabs: "patrons" (selected), "librarians", "books", "articles", "audio/video", and "checked out". Below the tabs are input fields for "name" (containing "Rollo"), "address", "phone", and "id". There are checkboxes for "all criteria must be satisfied" and "faculty". "search" and "new patron" buttons are present. Below these is a table with columns: id, name, address, phone, faculty, login, modify, and delete. The table contains one row for "Rollo Drake" with address "Kazan Street 12", phone "202-555-0173", and login "drake66". At the bottom, it says "Logged in as librarian: Master" and has "settings" and "Logout" buttons.

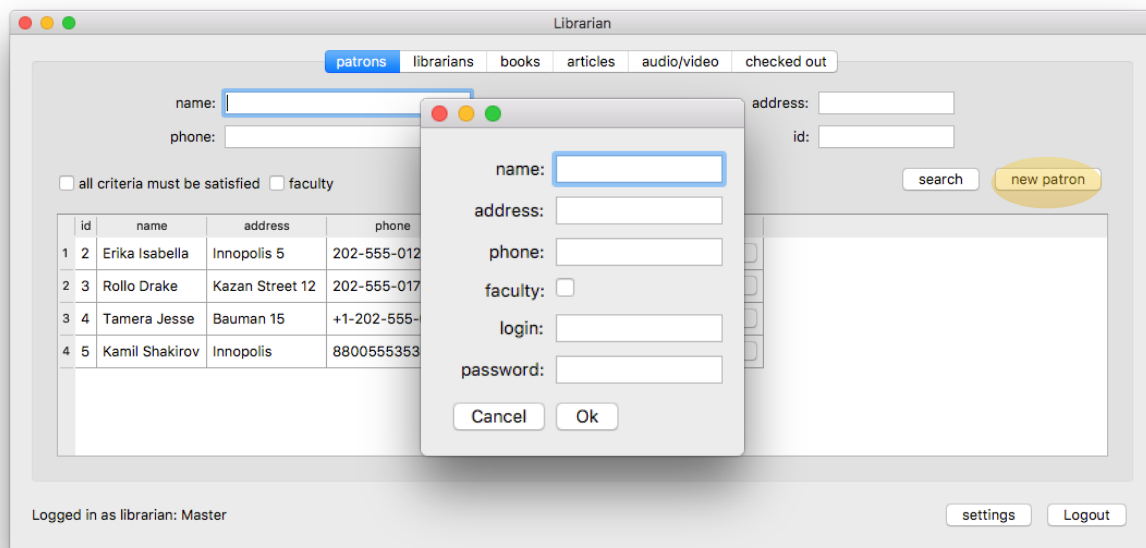
id	name	address	phone	faculty	login	modify	delete
1 3	Rollo Drake	Kazan Street 12	202-555-0173	no	drake66	modify	delete

Class LibrarianUser:

- add_patron

With this method, librarian can add new patrons

```
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();
}
```

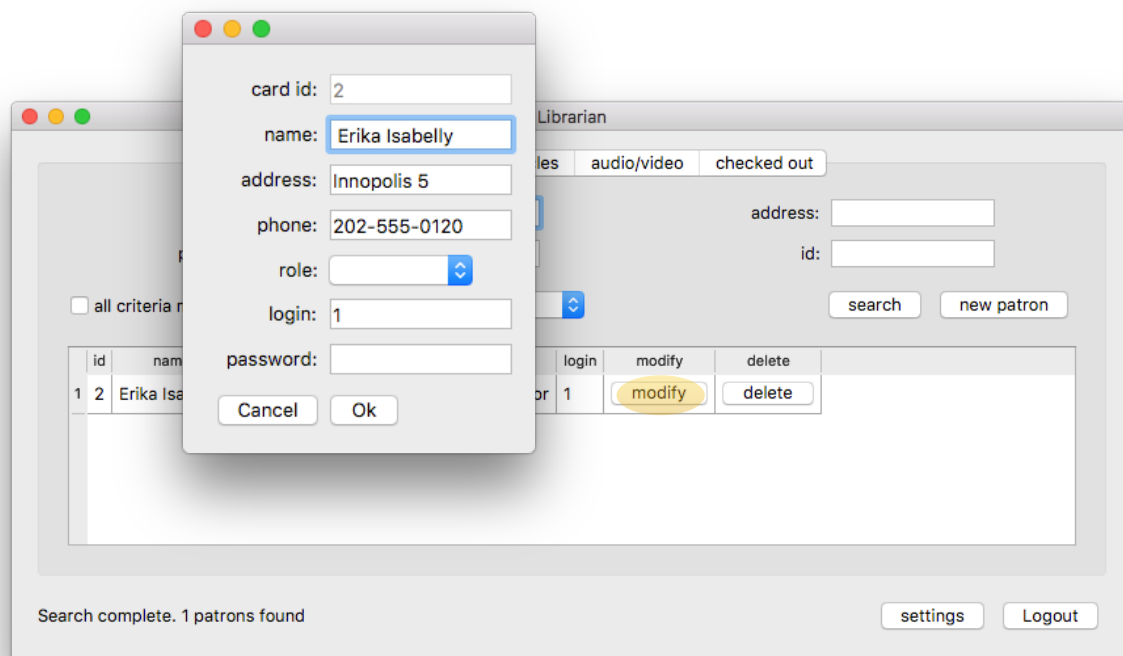


Class LibrarianUser:

- `modify_patron`

With this method, librarian can modify patrons

```
bool modify_patron(int user_id, QString name, QString address, QString phone, int role, QString login, QString password){  
    password = Hasher::hash_password(login,password);  
    QSqlQuery query;  
    query.prepare("UPDATE patrons SET name = :name, address = :address, phone = :phone, role = :role, login = :login, password = :password WHERE id = :user_id");  
    query.bindValue(":name", name);  
    query.bindValue(":address", address);  
    query.bindValue(":phone", phone);  
    query.bindValue(":role", role);  
    query.bindValue(":login", login);  
    query.bindValue(":password", password);  
    query.bindValue(":user_id", user_id);  
    query.exec();  
    return 1; //mb change to void  
}
```



Class LibrarianUser:

- delete_patron

Deleting patrons from the system

```
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;
}
```

Librarian

patrons books articles audio/video checked out

name: address:

phone: id:

☐ all criteria must be satisfied role: search new patron

	id	name	address	phone	role	login	modify	delete
1	2	Erika Isabelly	Innapolis 5	202-555-0120	Instructor	1	<input type="button" value="modify"/>	<input type="button" value="delete"/>

Search complete. 1 patrons found settings Logout

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;
}
```

The screenshot shows the 'Librarian' application window with a tabbed interface. The 'audio/video' tab is selected. The main window contains input fields for 'author:', 'keywords:', 'title:', and 'publisher:'. There is a checkbox for 'all criteria must be satisfied' and a 'search' button. A modal dialog is open in the center, titled 'Librarian', with input fields for 'title:', 'authors:', 'keywords:', 'price:', 'room:', 'level:', 'copies:', and a 'reference:' checkbox. The dialog has 'Cancel' and 'Ok' buttons. The 'new audio/video' button in the main window is highlighted with a yellow oval. The status bar at the bottom indicates 'Logged in as librarian: Master' and has 'settings' and 'Logout' buttons.

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;
}
```

The screenshot shows a window titled "Librarian" with several tabs: patrons, librarians, books (selected), articles, audio/video, and checked out. The "books" tab contains search fields for author (Jesmyn Ward), keywords, title, publisher, and year. There are checkboxes for "bestseller" and "available", and a checkbox for "all criteria must be satisfied". Search and new book buttons are present. Below the search fields is a table with columns: title, authors, publisher, keywords, year, price, room, level, copies, bestseller, reference, modify, and delete. The table contains two rows of book data. The first row has title "Sing, Unburied, Sing", author "Jesmyn Ward", publisher "ThomsonReuters", keywords "fiction science", year "2015", price "1560", room "5", level "5", copies "2", bestseller "yes", reference "no", and delete button highlighted. The second row has title "The cruel prince", author "Jesmyn Ward", publisher "RELX Group", keywords "love nature history", year "2011", price "2500", room "5", level "8", copies "6", bestseller "yes", reference "no", and delete button. At the bottom, it says "Logged in as librarian: Master" and has settings and Logout buttons.

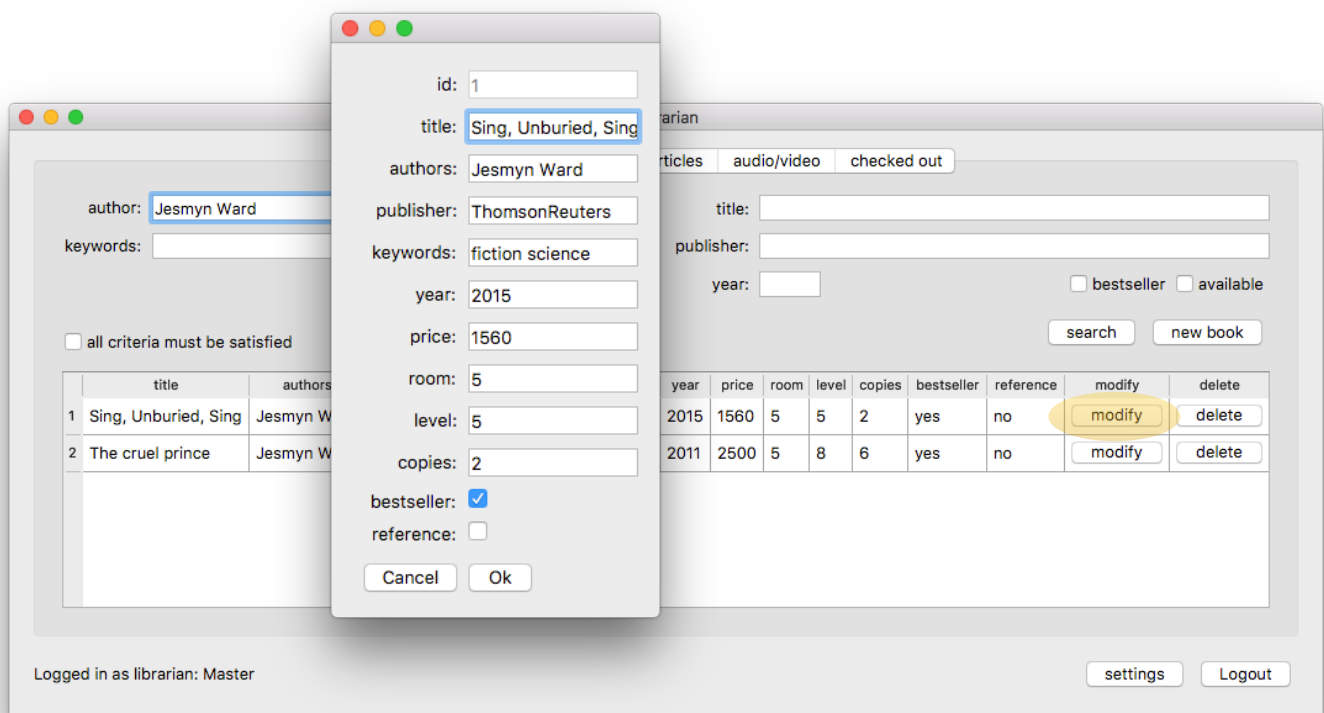
	title	authors	publisher	keywords	year	price	room	level	copies	bestseller	reference	modify	delete
1	Sing, Unburied, Sing	Jesmyn Ward	ThomsonReuters	fiction science	2015	1560	5	5	2	yes	no	modify	delete
2	The cruel prince	Jesmyn Ward	RELX Group	love nature history	2011	2500	5	8	6	yes	no	modify	delete

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;
}
```



Class LibrarianUser:

- outstanding_book, outstanding_article, outstanding_va

Thanks to these methods, librarian can do outstanding request

```
void outstanding_book(int document_id){
    QSqlQuery query;
    query.exec("SELECT wants FROM books WHERE id = " + document_id);
    query.next();
    QString wants = query.value(0).toString();

    Queue queue;
    queue.from_db(wants);

    while(queue.next_user() != -1){
        int next_id = queue.next_user();
        queue.delete_next();
        notify_patron(next_id, "You was removed from waiting list for a book due to outstanding request");
    }
    query.exec("UPDATE books SET wants = ';' WHERE id = " + QString::number(document_id));
}
```

The screenshot shows a window titled "Librarian" with tabs for "patrons", "books", "articles", "audio/video", and "checked out". The "books" tab is active. Below the tabs are input fields for "author:" (Jesmyn Ward), "keywords:", "title:", "publisher:", and "year:". There are checkboxes for "bestseller" and "available", and a checkbox for "all criteria must be satisfied". "search" and "new book" buttons are present. Below these is a table with 12 columns: "id", "keywords", "year", "price", "room", "level", "copies", "bestseller", "reference", "modify", "delete", and "outstanding". Two rows of search results are shown. The "request" button in the "delete" column of the second row is highlighted with a yellow circle. At the bottom, it says "Search complete. 2 books found" and has "settings" and "Logout" buttons.

id	keywords	year	price	room	level	copies	bestseller	reference	modify	delete	outstanding
1	euters fiction science	2015	1560	5	5	1	yes	no	modify	request	delete
2	p love nature history	2011	2500	5	8	6	yes	no	modify	request	delete

AdminUser:

- search_librarians

This method is search librarians in the system

```
QVector<LibrarianUser> search_librarians(int user_id, QString name, QString address, QString phone, int privileges, bool or_and){
    QSqlQuery query;
    QString ins = or_and ? " AND " : " OR ";
    name = name.toLower();
    address = address.toLower();
    phone = phone.toLower();
    QString req = "SELECT * FROM librarians 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 (privileges != 0) req += "privileges = " + QString::number(privileges) + ins;
    req += "1 = " + QString(or_and ? "1" : "0");//nice hack to finish statement correctly
    if (req.length() == 36)//no parameters given
        req = "SELECT * FROM librarians";
    query.exec(req);
    QVector<LibrarianUser> 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();
        QString login = query.value(4).toString();
        QString password = query.value(5).toString();
        int privileges = query.value(6).toInt();
        ans.push_back(LibrarianUser(user_id, name, address, phone, login, password, privileges));
    }
    return ans;
}
```

Form

name: address:

phone: id:

☒ all criteria must be satisfied Privileges

	id	name	address	phone	login	privileges	modify	delete
1	1	Pearl Harris	Innopolis Street 5	202-555-0113	darkharris	Priv1	<input type="button" value="modify"/>	<input type="button" value="delete"/>

Search complete. 1 librarians found

Class AdminUser:

- `add_librarian`

With this method, admin can add new librarians

```
void add_librarian(QString name, QString address, QString phone, QString login, QString password, int privileges){
    password = Hasher::hash_password(login,password);
    QSqlQuery query;
    query.prepare("INSERT INTO librarians (name, address, phone, login, password, privileges) VALUES(:name, :address, :phone, :login, :password, :privileges)");
    query.bindValue(":name", name);
    query.bindValue(":address", address);
    query.bindValue(":phone", phone);
    query.bindValue(":login", login);
    query.bindValue(":password", password);
    query.bindValue(":privileges", privileges);
    query.exec();
}
```

The screenshot displays a web application interface with a modal dialog box open for adding a new librarian. The modal dialog has the following fields: name (text input), address (text input), phone (text input), login (text input), password (text input), and privileges (dropdown menu with 'Priv1' selected). It includes 'Cancel' and 'Ok' buttons at the bottom. In the background, the main application window is visible, showing a 'new librarian' button highlighted in yellow. The main window also contains a search bar, a table with columns 'privileges', 'modify', and 'delete', and a status bar at the bottom indicating 'Logged in as admin' with 'My profile' and 'Log out' buttons.

Class AdminUser:

- `modify_librarian`

With this method, admin can modify librarians

```
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;
}
```

The screenshot displays a web application interface. In the foreground, a modal dialog box is open for modifying a librarian. The dialog contains the following fields and controls:

- card id: 1
- name: Pearl Harris
- address: Innopolis Street 5
- phone: 202-555-0113
- login: darkharris
- password: (empty field)
- privileges: Priv1 (dropdown menu)
- Buttons: Cancel, Ok

In the background, a search results table is visible with the following data:

id	name
1	Pearl Harris

Below the table, a status message reads: "Search complete. 1 librarians found". At the bottom right, there are buttons for "My profile" and "Log out".

Class AdminUser:

- `delete_librarian`

Deleting librarians from the system

```
bool delete_librarian(int user_id){  
    QSqlQuery query;  
    query.exec("DELETE FROM librarians WHERE id = " + QString::number(user_id));  
    return 1;  
}
```

Form

name: address:

phone: id:

☐ all criteria must be satisfied Privileges

	id	name	address	phone	login	privileges	modify	delete
1	1	Pearl Harris	Innopolis Street 5	202-555-0113	darkharris	Priv1	<input type="button" value="modify"/>	<input type="button" value="delete"/>

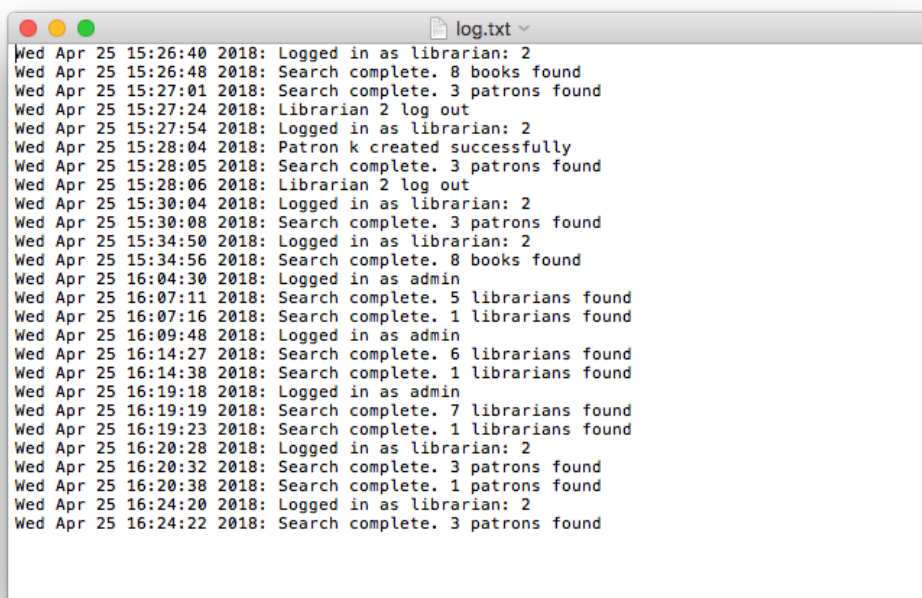
Search complete. 1 librarians found

"log.txt" file

We have added to our system log.txt file, which contains all actions

append_log method added last action to the end of log.txt file and it calls from almost all functions from LibrarianUser, PatronUser and AdminUser

```
static void append_log(QString data){
    QDateTime cur = QDateTime::currentDateTime();
    QFile writeFile("log.txt");
    if(!writeFile.open(QFile::Append | QFile::Text)) return;
    QTextStream in(&writeFile);
    in << cur.toString() << ": " << data << "\n";
    writeFile.close();
}
```



Conclusion

Thanks for reading this documentation. We hope that this information was useful to you.

If you have any questions, please contact us

- @Nmikriukov
- @Akkyma
- @greendate
- @kamsh

