

Programowanie obiektowe
INEW0003P
Projekt

Wydział elektroniki	Kierunek: informatyka
Grupa zajęciowa: Cz 18:55	Semestr: 2017/2018 Lato
Nazwisko i Imię: Poręba Dominika	Nr indeksu: 241126
Nr grupy projektowej:	60
Prowadzący:	mgr inż. Piotr Lechowicz

TEMAT:
Wypożyczalnia książek

Ocena:
Punkty:
Data:

1. Założenia i opis funkcjonalny programu

Poruszany problem:

Program będzie obsługiwał system zarządzania biblioteką.

Użytkownik docelowy:

Program dedykowany jest dla klientów biblioteki oraz jej pracowników.

Założenia:

Klientom zostanie umożliwione m. in.: rejestracja, logowanie, przeglądanie zasobów biblioteki, rezerwacja pozycji oraz prolongowanie wypożyczonych już dzieł.

Pracownicy będą mogli wypożyczać zasoby klientom, dodawać nowe pozycje oraz usuwać istniejące. Tak jak i klienci będą mieli dostęp do rejestracji, logowania oraz katalogu bibliotek.

Dodatkowe założenia:

Klient będzie mógł wyświetlić swój profil, a w nim sprawdzić historię wypożyczeń, listę swoich rezerwacji, usunąć swoje konto oraz wylogować się.

Użyte języki, środowiska i frameworki:

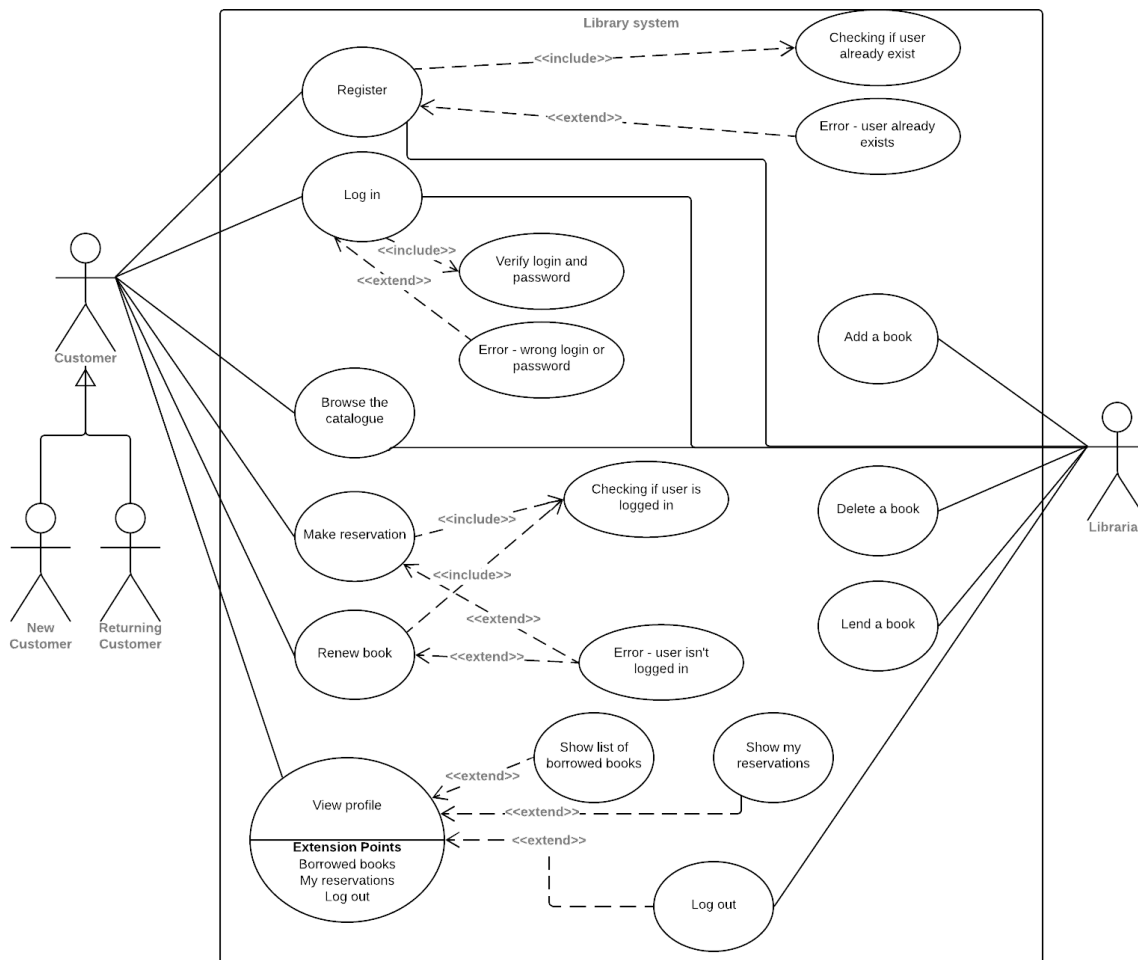
- C++, Qt Creator
- Python, CSS, HTML, Jinja2, FireBase, Flask

2. Diagramy UML

a) diagram przypadków użycia

USE CASE DIAGRAM

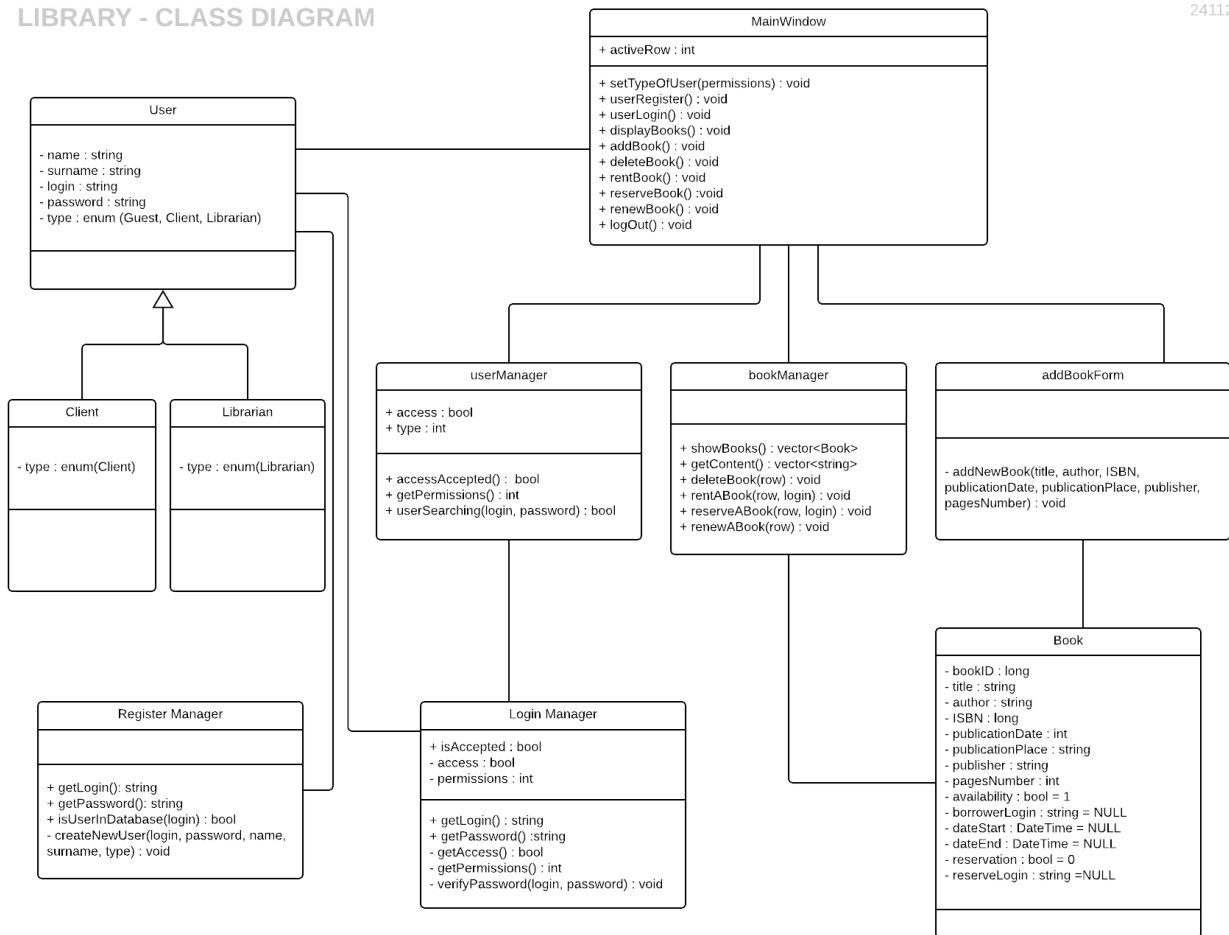
241126 |



b) diagram klas

LIBRARY - CLASS DIAGRAM

241126 |



3. Kod klas C++

Z kodu klas zostały usunięte fragmenty dotyczące GUI.

AddBookForm class

```
class addbookform : public QDialog
{
public:
    explicit addbookform(QWidget *parent = 0);
    ~addbookform();
private:
    /**
Adds a new book. It takes: title, author, ISBN number, publication date,
publication place, publisher and number of pages as arguments.
    */
    void addNewBook(QString title, QString author, QString ISBN, QString
        publicationDate, QString publicationPlace, QString publisher, QString
        pageNumber);
};
```

Book class

```
class Book
{
public:
    long m_bookID;
    std::string m_title;
    std::string m_author;
    long m_ISBN;
    int m_publicationDate;
    std::string m_publicationPlace;
    std::string m_publisher;
    int m_pagesNumber;
    bool m_availability;
    std::string m_borrowerlogin;
    QDateTime m_startdate;
    QDateTime m_enddate;
    bool m_reservation;
    std::string m_reserveLogin;

    /**
```

Getters

```
*/  
  
    long getBookID() const { return m_bookID; }  
    std::string getTitle() const {return m_title;}  
    std::string getAuthor() const {return m_author;}  
    long getISBN() const {return m_ISBN;}  
    int getPublicationDate() const {return m_publicationDate;}  
    std::string getPublicationPlace() const {return m_publicationPlace;}  
    std::string getPublisher() const {return m_publisher;}  
    int getPagesnumber() const {return m_pagesNumber;}  
    bool getAvailability() const {return m_availability;}  
    std::string getBorrowerLogin() const {return m_borrowerlogin;}  
    QDateTime getStartDate() const {return m_startdate;}  
    QDateTime getEndDate() const {return m_enddate;}  
    bool getReservation() const {return m_reservation;}  
    std::string getReserveLogin() const{return m_reserveLogin;}  
    Book()=default;  
  
/**  
Creates a book given its data.  
*/  
  
    Book(long i_bookID,  
        std::string i_title,  
        std::string i_author,  
        long i_ISBN,  
        int i_publicationDate,  
        std::string i_publicationPlace,  
        std::string i_publisher,  
        int i_pagesNumber,  
        bool i_availability,  
        std::string i_borrowerlogin,  
        QDateTime i_startdate,  
        QDateTime i_enddate,  
        bool i_reservation,  
        std::string i_reserveLogin);  
  
};
```

BookManager class

```
class bookmanager  
{  
public:  
    /**  
    Converts the raw database text data into a vector of Books.  
    */
```

```

        std::vector<Book> showBooks();
/**
Loads the raw database text data.
*/
        std::vector<QString> getContent();
/**
Deletes a book given its row.
*/
        void deleteBook(int row);

/**
Rents a book given its row and reservers login.
*/
        void rentABook(int row, QString login);

/**
Reserves a book given its row and reservers login.
*/
        void reserveABook(int row, QString login);

/**
Renews a book given its row
*/
        void renewABook (int row);
};

```

LoginManager class

```

class LoginManager : public QDialog
{
public:
/**
Helper functions
*/
        QString getLogin();
        QString getPassword();
        bool getAccess() const {return access;}
        int getPermissions() const {return permissions;}

/**
Helper variable
*/
        bool isAccepted;

```

```
private:
/**
Helper variables
*/
    int permissions=0;
    bool access=false;

/**
Verifies if password and login match. Takes login and password as arguments.
*/
    void verifyPassword(QString user_login, QString user_password);
};
```

MainWindow class

```
class MainWindow : public QMainWindow
{
public:
    LoginManager *loginmanager;
    User::Type type;
    RegisterManager *registermanager;
    addbookform *addbookdialog;
    rentbookform *rentbookdialog;

/**
Changes the users permissions.
*/
    void setTypeOfUser(int permissions);

/**
Contains currently selected row.
*/
    int activeRow;

/**
The following functions have self-explanatory names

To clear the confusion: they use the GUI to do their job,
so they take no arguments.
*/
    void userRegister();
    void userLogin();
    void displayBooks();
    void addBook();
    void deleteBook();
    void rentBook();
```



```

    void reserveBook();
    void renewBook();
    void logOut();
};

```

RegisterManager class

```

class RegisterManager : public QDialog
{
    /**
    Getters
    */
    QString getLogin();
    QString getPassword();

    /**
    Safety check, used in operations on users.
    */
    bool isUserInDatabase(QString login);

private:
    /**
    Creates a new user. Takes login, password, name, surname and type
    as arguments.
    */
    void createNewUser(QString login, QString password, QString name,
        QString surname, QString type);
};

```

User class

```

class User
{
public:
    std::string name;
    std::string surname;
    std::string login;
    std::string password;
    enum class Type
    {
        Guest,
        Client,
        Librarian
    };
};

```

```
    Type type;
    User();
};
```

UserManager class

```
class userManager
{
public:
    /**
    Helper variables
    */
    bool access;
    int type;

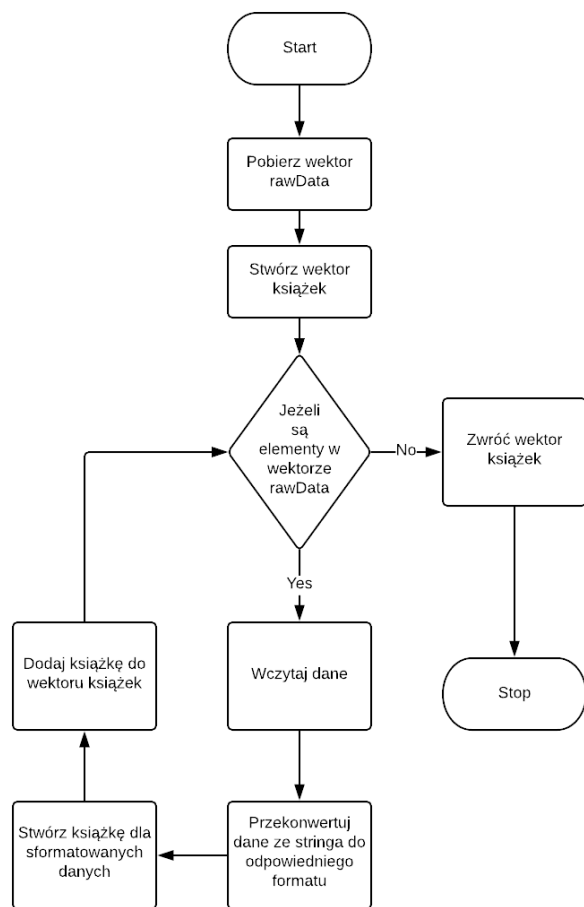
    /**
    Helper functions
    */
    bool accessAccepted() const {return access;}
    int getPermissions() const{return type;}

    /**
    Searches user in database.
    */
    bool userSearching(std::string login, std::string password);

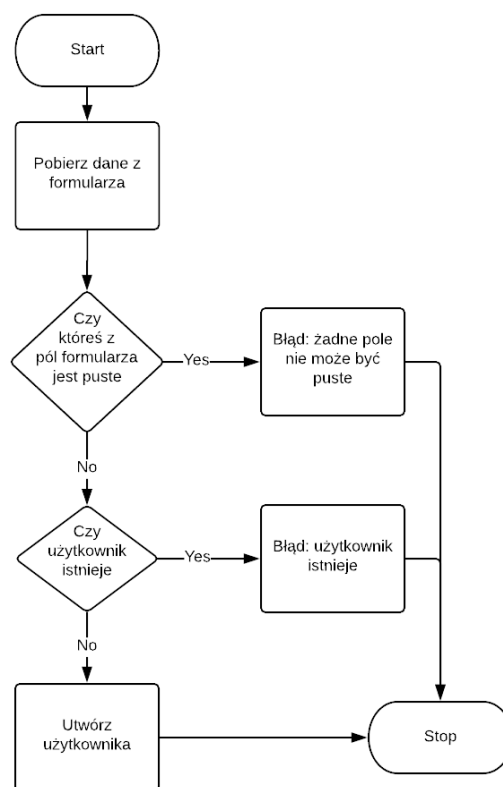
    userManager();
};
```

4. Schematy blokowe oraz kod własnych funkcji

Poniższe schematy blokowe ukazują działanie dwóch wybranych przeze mnie funkcji. Jedna z nich jest używana do wyświetlania wszystkich książek, natomiast druga służy do rejestrowania nowego użytkownika.



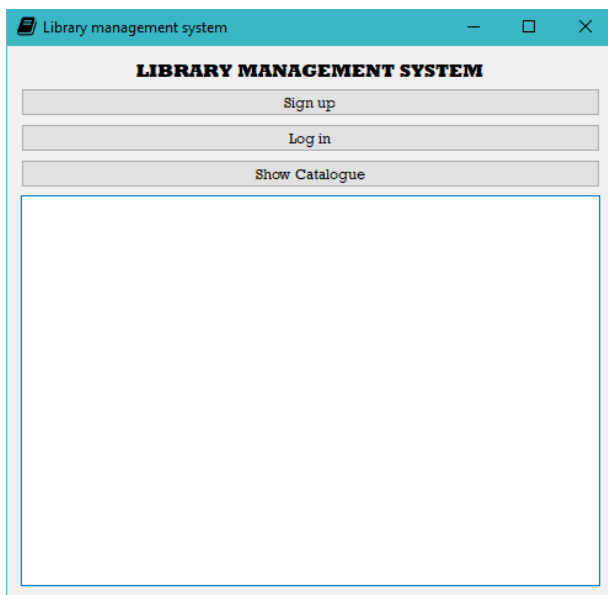
Schemat blokowy funkcji `showBooks`



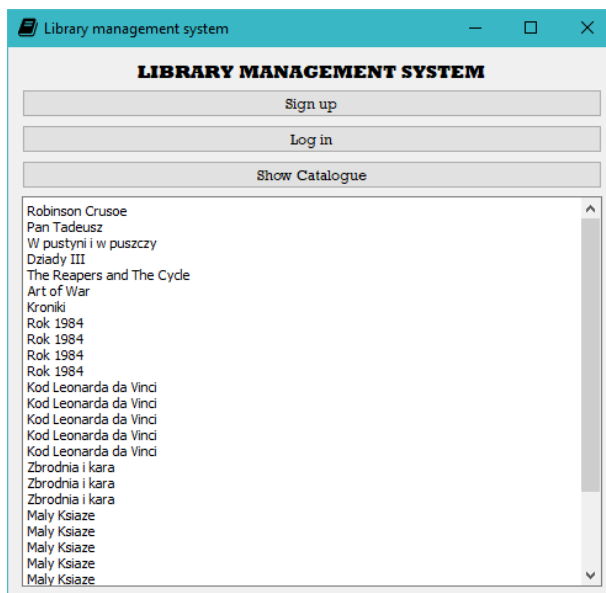
Schemat blokowy funkcji rejestrującej użytkownika

5. Opis użytkowy programu C++

Poruszanie się po programie jest bardzo intuicyjne. Po uruchomieniu aplikacji użytkownikowi ukazuje się menu biblioteki, w którym może się zarejestrować, zalogować lub wybrać opcję "Show Catalogue", która wyświetli tytuły wszystkich dostępnych książek.

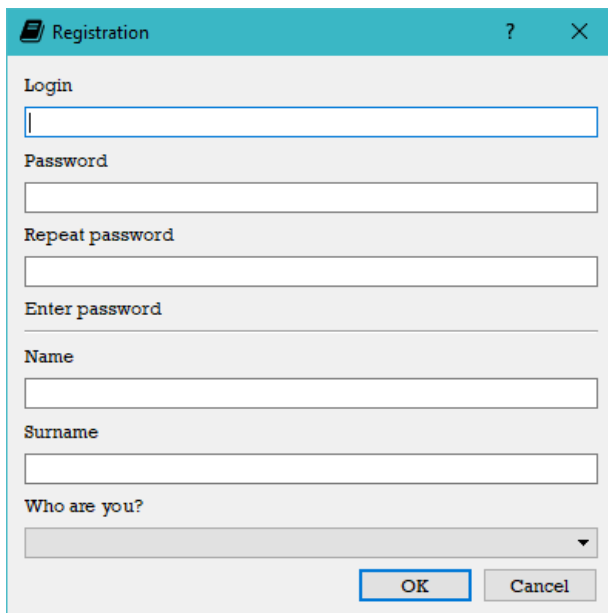


Menu główne



Menu po wybraniu opcji wyświetlenia katalogu

Po wybraniu opcji rejestracji wyświetla się formularz do wypełnienia. Jeżeli rejestracji próbuje dokonać użytkownik, który istnieje już w bazie, program informuje o tym. Pozostawienie któregoś z pól pustym również wyświetla ostrzeżenie. Dodatkowym udogodnieniem, które ułatwia rejestrację jest opcja, która sprawdza czy oba podane hasła są identyczne.



Registration

Login

Password

Repeat password

Enter password

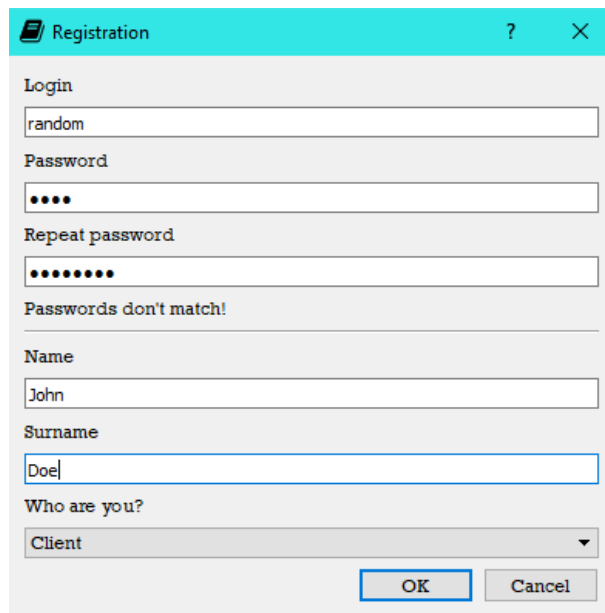
Name

Surname

Who are you?

OK Cancel

Menu rejestracji



Registration

Login

random

Password

.....

Repeat password

.....

Passwords don't match!

Name

John

Surname

Doel

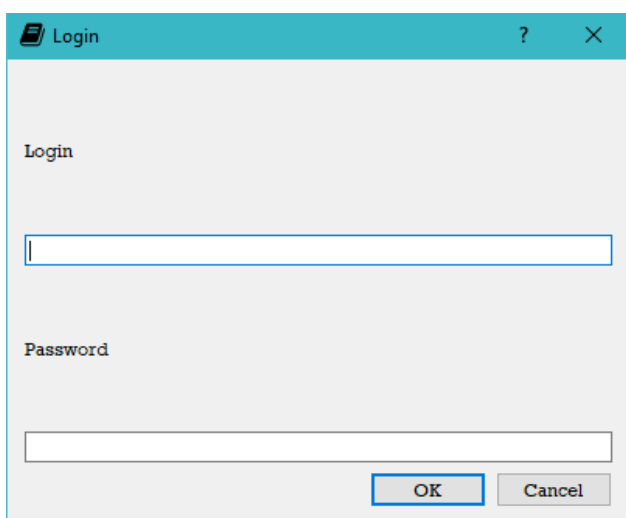
Who are you?

Client

OK Cancel

Panel rejestracyjny z przykładowymi danymi

Po wybraniu opcji logowania wyświetla się formularz do wpisania loginu i hasła. W przypadku wpisania złego hasła, otrzymujemy komunikat o tym informujący lub w przypadku, gdy program podejrzewa że podanego użytkownika nie ma w bazie, również otrzymujemy stosowny komunikat.



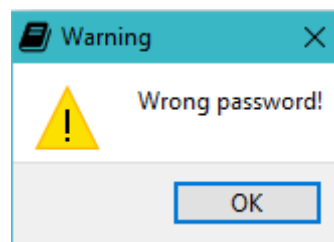
Login

Login

Password

OK Cancel

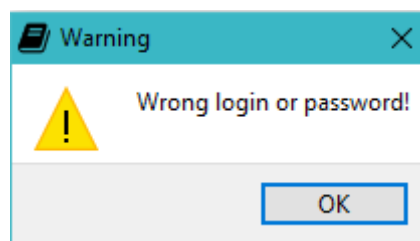
Menu logowania



Warning

! Wrong password!

OK



Warning

! Wrong login or password!

OK

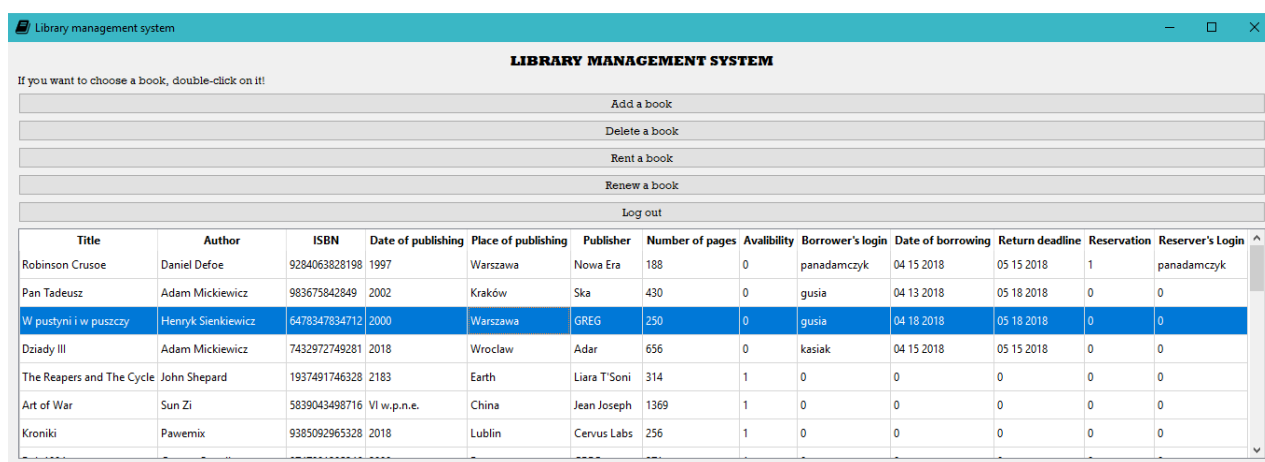
Przykładowe komunikaty

Po zalogowaniu się jako klient uzyskujemy dostęp do podstawowych funkcji programu. Otrzymujemy katalog książek wraz z większością ich danych: autorem, numerem ISBN, liczbą stron, informacje dotyczące publikacji (miejsce, czas i wydawnictwo) oraz dostępności książki (czy jest wypożyczona, czy jest zarezerwowana). Możemy zarezerwować książkę, którą chcemy wypożyczyć lub prolongować wypożyczoną już przez nas pozycję. Po wykonaniu czynności które nas interesują możemy się wylogować.



Wygląd programu dla klienta

Po zalogowaniu się jako bibliotekarz uzyskujemy dostęp do większości funkcji programu. Otrzymujemy katalog książek wraz z ich wszystkimi danymi: autorem, numerem ISBN, liczba stron, informacje dotyczące publikacji (miejsce, czas i wydawnictwo) oraz dostępności książki (czy jest wypożyczona, kto ja wypożyczył, od kiedy i do kiedy, czy jest zarezerwowana, jeśli tak to przez kogo). Mamy możliwość dodania nowej książki, usunięcia istniejącej lub wypożyczenia.



Wygląd programu dla bibliotekarza

Do uruchomienia programu w systemie Windows potrzebne są pliki konfiguracyjne *.dll, które znajdują się w folderze wraz z plikiem wykonywalnym.

6. Listing kodu C++ - wraz z komentarzami

W niektórych funkcjach nadal znajdują się elementy GUI, które nie dało się wyciąć ze względu na połączenie warstwy logicznej z warstwą graficzną programu.

AddBookForm class

```
/**
Collects data from GUI. Then uses function addNewBook.
*/
void addbookform::on_pushButton_clicked()
{
    QString book_title;
    QString book_author;
    QString book_ISBN;
    QString book_publicationDate;
    QString book_publicationPlace;
    QString book_publisher;
    QString book_pagesNumber;

    if (!ui->lineEdit->text().trimmed().isEmpty())
        book_title = ui->lineEdit->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Title of book cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_2->text().trimmed().isEmpty())
        book_author = ui->lineEdit_2->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Author cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_3->text().trimmed().isEmpty())
        book_ISBN = ui->lineEdit_3->text().trimmed();
    else
    {
        QMessageBox::warning(this,
```

```

        tr("Warning"), tr("ISBN cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_4->text().trimmed().isEmpty())
        book_publicationDate = ui->lineEdit_4->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Publication date cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_5->text().trimmed().isEmpty())
        book_publicationPlace = ui->lineEdit_5->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Publication place cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_6->text().trimmed().isEmpty())
        book_publisher = ui->lineEdit_6->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Publisher cannot be empty!"));
        return;
    }

    if (!ui->lineEdit_7->text().trimmed().isEmpty())
        book_pagesNumber = ui->lineEdit_7->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Number of pages cannot be empty!"));
        return;
    }

    addNewBook(book_title, book_author, book_ISBN, book_publicationDate,
        book_publicationPlace, book_publisher, book_pagesNumber);

    QMessageBox::information(this,
        tr("Information"), tr("Book added!"));
    return;
}

```



```

/**
Adds a new book. It takes: title, author, ISBN number, publication date,
publication place, publisher and number of pages as arguments.
*/
void addbookform::addNewBook(QString title, QString author, QString ISBN,
QString publicationDate, QString publicationPlace, QString publisher,
QString pageNumber)
{
    QString path = QApplication::applicationDirPath();
        path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly))
        QMessageBox::information(0, "Error", file.errorString());
    QTextStream in(&file);
    QString text;
    text = in.readAll();
    file.close();
    QRegExp rx("(\\;);");
    QStringList query = text.split(rx);

    query.pop_back();
    query.pop_back();

    int numberOfbooks = query.size()/14 + 1;
    QString lastID = query[(numberOfbooks-2)*14];
    int newID = lastID.toInt() + 2;

    QString bookID = QString::number(newID);
    QString availability = "1";
    QString borrowerID = "0";
    QString startDate = "0";
    QString endDate = "0";
    QString reservation = "0";
    QString reserveLogin = "0";

    QString insertQuery = text +
        bookID + ";" +
        title + ";" +
        author + ";" +
        ISBN + ";" +
        publicationDate + ";" +
        publicationPlace + ";" +
        publisher + ";" +
        pageNumber + ";" +
        availability + ";" +
        borrowerID + ";" +
        startDate + ";" +

```

```

        endDate + ";" +
        reservation + ";" +
        reserveLogin + ";";

QString path2 = QApplication::applicationDirPath();
    path2.append("\\bookdatabase.txt");
QFile file2(path);
if (!file2.open(QIODevice::WriteOnly | QIODevice::Text))
    return;

QTextStream out(&file2);
out << insertQuery;

}

```

Book class

```

/**
Constructor. Creates Book objects.
*/
Book::Book(long i_bookID,
            std::string i_title,
            std::string i_author,
            long i_ISBN,
            int i_publicationDate,
            std::string i_publicationPlace,
            std::string i_publisher,
            int i_pagesNumber,
            bool i_availability,
            std::string i_borrowerLogin,
            QDateTime i_startdate,
            QDateTime i_enddate,
            bool i_reservation,
            std::string i_reserveLogin)
: m_bookID(i_bookID),
  m_title (i_title),
  m_author (i_author),
  m_ISBN (i_ISBN),
  m_publicationDate (i_publicationDate),
  m_publicationPlace (i_publicationPlace),
  m_publisher (i_publisher),
  m_pagesNumber (i_pagesNumber),
  m_availability (i_availability),
  m_borrowerlogin (i_borrowerLogin),
  m_startdate (i_startdate),

```

```

        m_enddate (i_enddate),
        m_reservation (i_reservation),
        m_reserveLogin (i_reserveLogin)
    {}

```

BookManager class

```

/**
Converts the raw database text data into a vector of Books.
*/
std::vector<Book> bookmanager::showBooks()
{
    QList<QString> query =
    QList<QString>::fromVector(QVector<QString>::fromStdVector(getContent()));

    std::vector<Book> temp_books;

    for(int i=0;i<(query.size()-1);i++)
    {
        long i_bookID = query[0].toLong();
        std::string i_title = query[1].toStdString();
        std::string i_author = query[2].toStdString();
        long i_ISBN = query[3].toLong();
        int i_publicationDate = query[4].toInt();
        std::string i_publicationPlace = query[5].toStdString();
        std::string i_publisher = query[6].toStdString();
        int i_pagesNumber = query[7].toInt();
        bool i_availability = query[8].toInt();
        std::string i_borrowerLogin = query[9].toStdString();
        QDateTime i_startdate = QDateTime::fromString(query[10], "MM dd yyyy");
        QDateTime i_enddate = QDateTime::fromString(query[11], "MM dd yyyy");
        bool i_reservation = query[12].toInt();
        std::string i_reserveLogin = query[13].toStdString();

        Book temp_book(i_bookID,
                       i_title,
                       i_author,
                       i_ISBN,
                       i_publicationDate,
                       i_publicationPlace,
                       i_publisher,
                       i_pagesNumber,
                       i_availability,
                       i_borrowerLogin,
                       i_startdate,

```

```

        i_enddate,
        i_reservation,
        i_reserveLogin);

    temp_books.push_back(temp_book);

    for(int i=0;i<14;i++)
        query.pop_front();
}
return temp_books;
}

/**
Loads the raw database text data from *.txt file into a vector.
*/
std::vector<QString> bookmanager::getContent()
{
    QString path = QApplication::applicationDirPath();
    path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly))
        QMessageBox::information(0, "Error", file.errorString());
    QTextStream in(&file);
    QString text;
    text = in.readAll();
    file.close();
    QRegExp rx("(\\;);"); //RegEx for ';'
    QStringList query = text.split(rx);

    std::vector<QString> vec = query.toVector().toStdVector();
    return vec;
}

/**
Deletes a book given its row. It updates the database (*.txt file).
*/
void bookmanager::deleteBook(int row)
{
    std::vector<QString> content = getContent();
    int start = 14*row;
    int end= start+14;
    content.erase(content.begin()+start, content.begin()+end);

    QString query;
    for (auto x : content)
    {
        query+=x+";";
    }
}

```

```

    }
    query.truncate(query.size()-1);

    QString path = QApplication::applicationDirPath();
        path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);
    out << query;
}

/**
Rents a book given its row and reserves login. Adds current date as
a beginnig of duration of renting book process. Adds deadline for book
returning by adding 30 days to current date. Updates database.
*/
void bookmanager::rentABook(int row, QString login)
{
    std::vector<QString> content = getContent();

    int start = 14*row;
    content[start+8]="0";
    content[start+9]=login;

    QDateTime today = QDateTime::currentDateTime();
    QDateTime later = today.addDays(30);

    content[start+10]= today.toString("MM dd yyyy");
    content[start+11]=later.toString("MM dd yyyy");

    QString query;
    for (auto x : content)
    {
        query += x + ";";
    }

    QString path = QApplication::applicationDirPath();
        path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);

```

```

        out << query;
    }

/**
Reserves a book given its row and reservers login. Updates database.
*/
void bookmanager::reserveABook(int row, QString login)
{
    std::vector<QString> content = getContent();
    int start = 14*row;
    content[start+12]="1";
    content[start+13]=login;

    QString query;
    for (auto x : content)
    {
        query+=x+" ";
    }

    QString path = QApplication::applicationDirPath();
    path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);
    out << query;
}

/**
Renews a book given its row. Postpones the deadline of book returning by
adding 30 daysto current date and setting is as a new dealine.
Updates database.
*/
void bookmanager::renewABook(int row)
{
    std::vector<QString> content = getContent();
    int start = 14*row;

    QDateTime current = QDateTime::currentDateTime();
    QDateTime renew = current.addDays(30);

    content[start+11]=renew.toString("MM dd yyyy");

    QString query;
    for (auto x : content)

```

```

    {
        query+=x+";";
    }

    QString path = QCoreApplication::applicationDirPath();
    path.append("\\bookdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);
    out << query;
}

```

LoginManager class

```

/**
Helper functions. Gets login and password from GUI.
*/
QString LoginManager::getLogin()
{
    return ui->lineEdit_2->text();
}

QString LoginManager::getPassword()
{
    return ui->lineEdit->text();
}

/**
Logging function. Collects data (login and password) from GUI. Then uses
function verifyPassword.
*/
void LoginManager::on_buttonBox_accepted()
{
    QString user_login;
    QString user_password;

    if (!ui->lineEdit_2->text().trimmed().isEmpty())
        user_login = ui->lineEdit_2->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Login cannot be empty!"));
        return ;
    }
}

```

```

    }

    if (!ui->lineEdit->text().trimmed().isEmpty())
        user_password = ui->lineEdit->text().trimmed();
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Password cannot be empty!"));
        return ;
    }
    verifyPassword(user_login, user_password);
}

/**
Verifies if password and login match. Takes login and password as arguments.
*/
void LoginManager::verifyPassword(QString user_login, QString user_password)
{
    userManager manager;
    if (manager.userSearching(user_login.toStdString() ,
        user_password.toStdString() ))
    {
        if (manager.access==true)
        {
            this->access= true;
            this->permissions=manager.getPermissions();
        }
        else
        {
            QMessageBox::warning(this,
                tr("Warning"), tr("Wrong password!"));
            return ;
        }
    }
    else
    {
        QMessageBox::warning(this,
            tr("Warning"), tr("Wrong login or password!"));
        return ;
    }
}
}

```


MainWindow class

```
/**
Changes the users permissions.
*/
void MainWindow::setTypeOfUser(int permissions)
{
    if (permissions==0)
    {
        this->type= User::Type::Guest;
    }
    if (permissions==1)
    {
        this->type= User::Type::Client;
    }
    if (permissions==2)
    {
        this->type= User::Type::Librarian;
    }
}

/**
The following functions have self-explanatory names

To clear the confusion: they use the GUI to do their job,
so they take no arguments.
*/
void MainWindow::userRegister()
{
    registermanager = new RegisterManager(this);
    if(registermanager->exec())
    {}
}

void MainWindow::userLogin() {
    loginmanager = new LoginManager(this);
    if(loginmanager->exec())
    {
        if(loginmanager->getAccess())
            setTypeOfUser(loginmanager->getPermissions());
    }
}

void MainWindow::displayBooks()
{

```

```

bookmanager manager;
std::string stext;
for(auto book : manager.showBooks())
    stext+=book.getTitle()+'\n';
QString text = QString::fromStdString(stext);
ui->textBrowser->setText(text);
}

void MainWindow::addBook()
{
    addbookdialog= new addbookform(this);
    if(addbookdialog->exec())
    {}
    setTypeOfUser(loginmanager->getPermissions());
}

void MainWindow::deleteBook()
{
    bookmanager manager;
    manager.deleteBook(activeRow);
    setTypeOfUser(loginmanager->getPermissions());
}

void MainWindow::rentBook()
{
    rentbookdialog = new rentbookform(this);
    if(rentbookdialog->exec())
    {
        QString login = rentbookdialog->getLogin();
        if (!login.isEmpty())
        {
            bookmanager manager;
            manager.rentABook(activeRow, login);
        }
    }
    setTypeOfUser(loginmanager->getPermissions());
}

void MainWindow::reserveBook()
{
    bookmanager manager;
    QString login = loginmanager->getLogin();
    manager.reserveABook(activeRow, login);
    setTypeOfUser(loginmanager->getPermissions());
}

```

```

void MainWindow::renewBook()
{
    bookmanager manager;
    manager.renewABook(activeRow);
    setTypeOfUser(loginmanager->getPermissions());
}

void MainWindow::logout()
{
    setTypeOfUser(0);
}

```

RegisterManager class

```

/**
Helper functions. Gets login and password from GUI.
*/
QString RegisterManager::getLogin()
{
    return ui->lineEdit->text();
}

QString RegisterManager::getPassword()
{
    return ui->lineEdit_2->text();
}

/**
Safety check, used in operations on users. Checks if user exists in database.
*/
bool RegisterManager::isUserInDatabase(QString login)
{
    QString path = QApplication::applicationDirPath();
    path.append("\\userdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly))
        QMessageBox::information(0, "Error", file.errorString());
    QTextStream in(&file);
    QString text;
    text = in.readAll();
    file.close();
    QRegExp rx("(\\;);");
    QStringList query = text.split(rx);
    QStringList logins;
    query.pop_back();

```

```

while (!query.isEmpty())
{
    logins.push_back(query[2]);
    for(int i=0;i < 5;i++)
        query.pop_front();
}
int index = logins.indexOf(login);
if(index!=-1)
{
    return true;
}
else return false;
}

/**
Creates a new user. Takes login, password, name, surname and type (taken from
GUI) as arguments. Updates users database.
*/
void RegisterManager::createNewUser(QString login, QString password, QString
name, QString surname, QString type)
{
    QString path2 = QApplication::applicationDirPath();
        path2.append("\\userdatabase.txt");
    QFile file2(path2);
    if (!file2.open(QIODevice::ReadOnly))
        QMessageBox::information(0,"Error", file2.errorString());
    QTextStream in(&file2);
    QString text;
    text = in.readAll();
    file2.close();

    QString newQuery = text + name + ";" + surname + ";" + login +
        ";" + password + ";" + type + ";";

    QString path = QApplication::applicationDirPath();
        path.append("\\userdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);
    out << newQuery;
}

```

UserManager class

```
/**
Searches user in database. Chcecks correctness of given password
and returns access.
*/
bool userManager::userSearching(std::string login, std::string password)
{
    QString path = QApplication::applicationDirPath();
    path.append("\\userdatabase.txt");
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly))
        QMessageBox::information(0, "Error", file.errorString());
    QTextStream in(&file);
    QString text;
    text = in.readAll();
    file.close();
    QRegExp rx("(\\;);");
    QStringList query = text.split(rx);

    int index = query.indexOf(QString::fromStdString(login));
    if(index!=-1)
    {
        QString realPassword = query[index+1];
        QString md5password = QString(
            QCryptographicHash::hash(
                QString::fromStdString(password).toUtf8(),
                QCryptographicHash::Md5
            ).toHex());

        if (realPassword==md5password)
        {
            access=true;
            type = query[index+2].toInt();
        }
        else access = false;
        return true;
    }
    else return false;
}
```

7. Wnioski

- W trakcie realizacji projektu w C++ udało mi się zrealizować wszystkie jego początkowe założenia. Ponadto w Pythonie zrealizowałam 2 dodatkowe założenia: klienci mogą sprawdzić listę swoich rezerwacji oraz historię wypożyczeń.
- Realizacja projektu uświadomiła mi, jak ważne jest oddzielenie warstwy logicznej programu od warstwy graficznej.
- W przypadku Pythona oddzielenie warstwy logicznej od graficznej jest wymuszone, gdzie w C++ nie (da się bez tego obejść, chociaż odbywa się to kosztem przejrzystości kodu).
- W Pythonie o wiele prostsze jest instalowanie zewnętrznych bibliotek.
- Wirtualne środowisko pythona (virtualenv) jest zupełnie inne niż C++'owskie dynamiczne/statyczne linkowanie bibliotek.