

*03/29/2019 22:40:27*

*8*

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №9**

з дисципліни “Основи програмування 2”

тема “**Реляційна модель даних**”

|  |  |  |
| --- | --- | --- |
| Виконав  Студент I курсу  групи КП-83  Мортіков Владислав Євгенович  (*прізвище, ім’я, по батькові*)  варіант № |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2019

**Мета роботи**

Навчитись створювати мультивіконні графічні додатки.  
Реалізувати ідентифікацію, аутентифікацію та авторизацію у програмі.  
Опанувати різні види зв'язків між таблицями реляційної бази даних.

### **Загальні вимоги до завдання**

Кроки виконання:

1. Скопіювати у проект даного завдання ваше GUI рішення [лабораторної роботи №7](https://progbase.herokuapp.com/modules/progbase2/tasks/lab7).
2. Скопіювати у проект даного завдання модуль sqlite\_storage та файл бази даних data/sqlite/data.sqlite з [лабораторної роботи №8](https://progbase.herokuapp.com/modules/progbase2/tasks/lab8).
3. Модель даних. Створити базу даних за схемою з ERD діаграми **додатку А**:
   1. Створити у базі даних нову таблицю users з стовпцями:
      1. id - цілочисельний автоінкрементний ключ
      2. username - рядок, логін користувача
      3. password\_hash - рядок, хеш пароля користувача
   2. Обрати алгоритм хешування паролів (наприклад, [MD5](https://www.md5hashgenerator.com/)).
   3. Додати через SQLite клієнт 2-3 записи у таблицю користувачів, password\_hash згенерувати з паролю за допомогою обраного алгоритму хешування.
   4. Сформувати між таблицею users та таблицею основних сутностей відношення One-To-Many.  
      Для цього додати у таблицю основних сутностей foreign key поле user\_id.
   5. Сформувати між таблицею основних сутностей та таблицею додаткових сутностей відношення Many-To-Many.  
      Для цього створити таблицю links з трьома полями:
      1. PK Ідентифікатором зв'язку
      2. FK Ідентифікатором запису з таблиці основних сутностей
      3. FK Ідентифікатором запису з таблиці додаткових сутностей
4. Розширення функціональності сховища. Додати [нові чисті віртуальні функції](https://docs.google.com/document/d/1XI_8fQ7NscgXDIbTqmLGv-mxJRkc06Aznbte1L-IqkE/edit" \l "heading=h.ck28icczjplq) у абстрактний клас Storageі реалізувати їх у всіх класах-наслідниках (у CsvStorage та XmlStorage можна поставити методи-заглушки, які нічого не роблять):
   1. Отримання опціонала користувача за його логіном та паролем
   2. Отримання вектора тих основних сутностей, що належать залогіненому користувачу
   3. Отримання вектора всіх додаткових сутностей, що належать даній основній сутності
   4. Додавання вказаної додаткової сутності до вказаної основної сутності
   5. Видалення вказаної додаткової сутності із вказаної основної сутності
5. Розширення графічного інтерфейсу. Вікна:
   1. Реалізувати вікно аутентифікації:
      1. Розмістити форму вводу логіна і пароля та кнопку підтвердження
      2. Після вводу даних вікно перевіряє чи існує в базі даних користувач з таким логіном та хешем паролю і відкриває головне вікно програми або показує стандартне діалогове вікно з помилкою аутентифікації і пропонує ввести дані ще раз.
   2. На головному вікні відображати тільки ті основні сутності, що належать поточному користувачу, що увійшов у систему.  
      Всі новостворені сутності будуть належати лише цьому користувачу.
   3. На вікні редагування основної сутності додати графічний список додаткових сутностей, що зв'язані зв'язками з обраною основною сутністю:
      1. Можна додати у цей список сутність із існуючих додаткових сутностей (створити новий зв'язок, а не додаткову сутність)
      2. Можна видалити з цього списку обрану додаткову сутність (видаляється лише зв'язок, а не сама додаткова сутність)

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| ##include "mainwindow.h"  #include <QApplication>  int **main**(int argc, char \*argv[])  {  QApplication a(argc, argv);  MainWindow w;  w.show();  return a.exec();  } |

|  |
| --- |
| **mainwindow.h** |
| #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include "storage.h"  #include <QListWidget>  namespace **Ui** {  class **MainWindow**;  }  class **MainWindow** : public QMainWindow  {  Q\_OBJECT  public:  explicit **MainWindow**(QWidget \*parent = nullptr);  ~***MainWindow***();  private slots:  void **on\_pushButton\_clicked**();  void **on\_listWidget\_itemClicked**(QListWidgetItem \*item);  void **on\_pushButton\_2\_clicked**();  void **on\_pushButton\_3\_clicked**();  private:  Ui::MainWindow \*ui;  Storage \* storage\_=nullptr;  void **enableEditing**();  void **disableEditing**();  void **setDefault**();  void **onNew**();  void **onOpen**();  void **fillListWithUserId**(int user\_id);  void **authfunc**();  };  #endif // MAINWINDOW\_H |

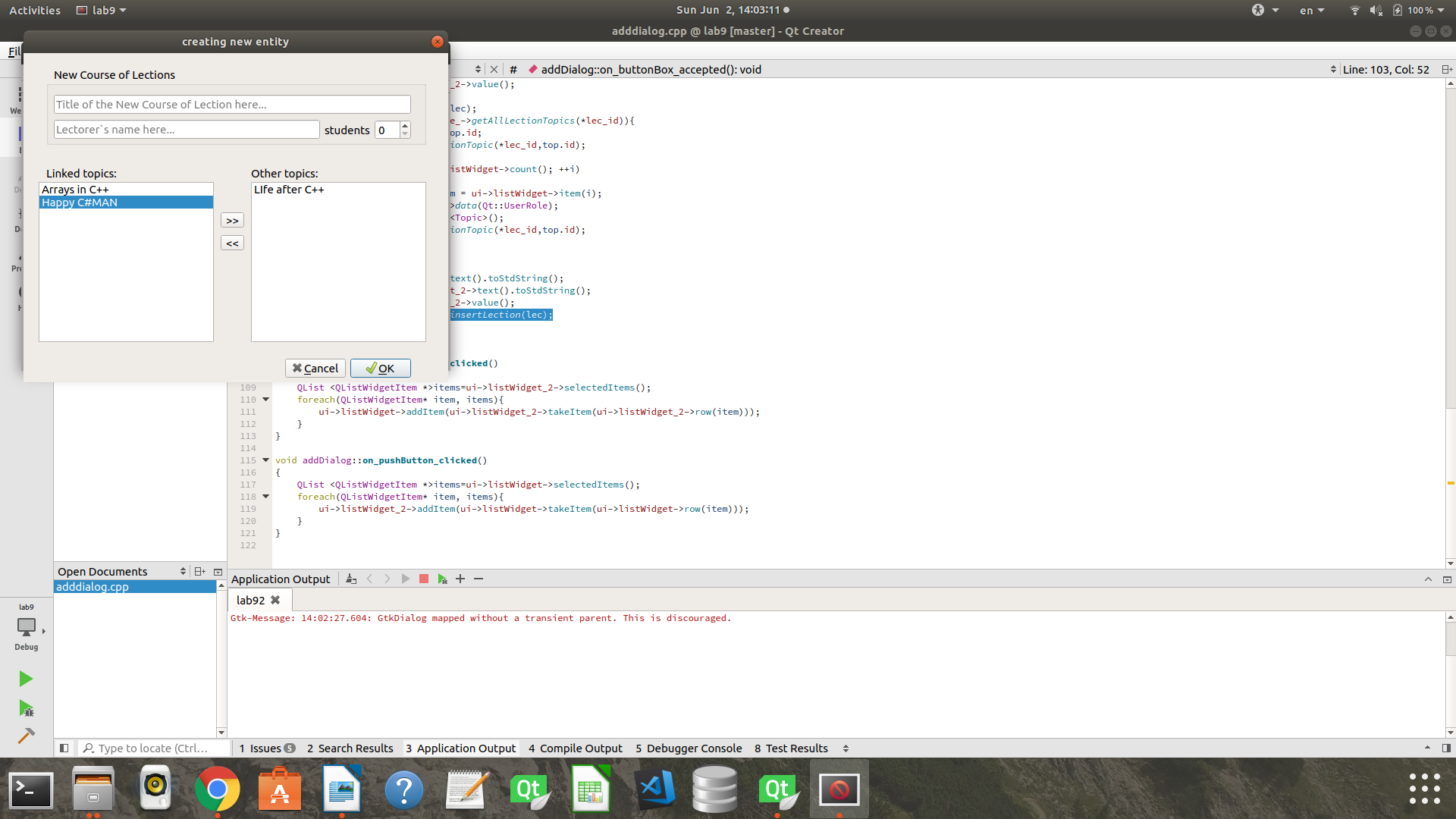
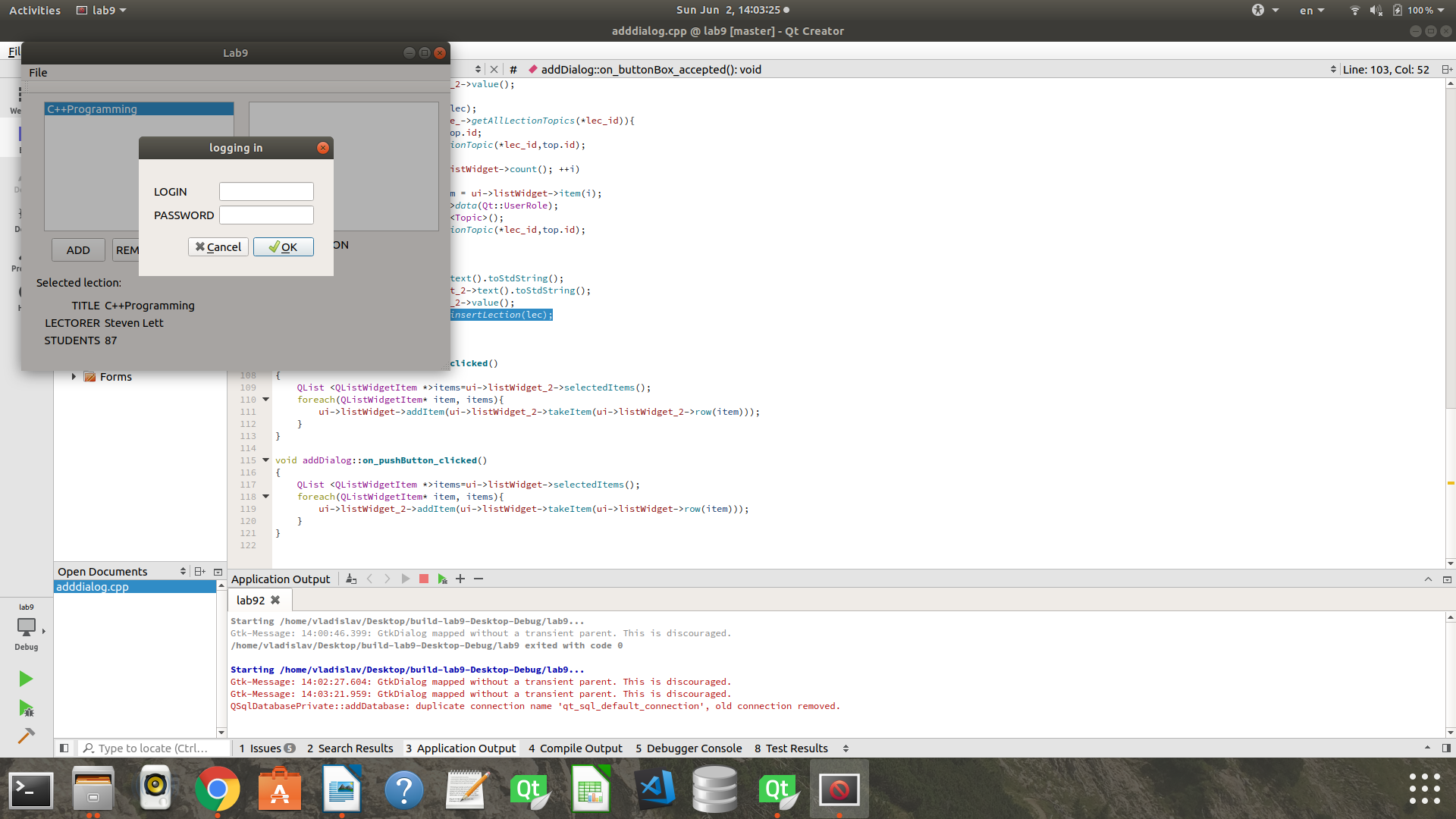
|  |
| --- |
| **Mainwindow.cpp** |
| #include "ui\_mainwindow.h"  #include "QDebug"  #include <QFileDialog>  #include "sqlite\_storage.h"  #include <QMessageBox>  #include "adddialog.h"  #include "auth.h"  MainWindow::**MainWindow**(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  connect(ui->actionOpen\_storage,&QAction::triggered,this,&MainWindow::onOpen);  setDefault();  }  MainWindow::~***MainWindow***()  {  delete ui;  }  void MainWindow::**onOpen**(){  if(storage\_!=nullptr)  delete(SQLiteStorage\*)storage\_;  storage\_=nullptr;  QFileDialog dialog(this);  dialog.setFileMode(QFileDialog::Directory);  QString current\_dir = QDir::currentPath();  QString folder\_path = dialog.getExistingDirectory(  this,  "Select Storage Folder",  current\_dir + "/../../../../lab9/data/sql",  QFileDialog::ShowDirsOnly);  if (folder\_path.isNull())  return;  ifstream file;  file.open(folder\_path.toStdString()+"/data.sqlite");  if(!file)  return ;  storage\_ = new SQLiteStorage(folder\_path.toStdString());  if(storage\_ -> *open*()){  authfunc();  ui->pushButton->setEnabled(true);  }  }  void MainWindow::**on\_pushButton\_clicked**()  {  addDialog addialog\_(storage\_,this);  addialog\_.setWindowTitle("creating new entity");  int status=addialog\_.*exec*();  fillListWithUserId(storage\_->*getUser*().user\_id);  }  void MainWindow::**fillListWithUserId**(int user\_id){  ui->listWidget->clear();  for (Lection lec : storage\_->*getUserLections*(user\_id)){  QVariant qVariant;  qVariant.setValue(lec);  QListWidgetItem \*qLectionListItem = new QListWidgetItem();  qLectionListItem->setText(QString::fromStdString(lec.title));  qLectionListItem->*setData*(Qt::UserRole, qVariant);  ui->listWidget->addItem(qLectionListItem);  }  }  void MainWindow::**on\_listWidget\_itemClicked**(QListWidgetItem \*item)  {  QVariant var = item->*data*(Qt::UserRole);  Lection lec =var.value<Lection>();  ui->label\_2->setText(QString::fromStdString(lec.title));  ui->label\_4->setText(QString::fromStdString(lec.lectorer));  ui->label\_6->setText(QString::number(lec.students));  enableEditing();  }  void MainWindow::**on\_pushButton\_2\_clicked**()  {  QMessageBox::StandardButton reply;  reply = QMessageBox::question(  this,  "On delete",  "Are you sure?",  QMessageBox::Yes|QMessageBox::No);  if (reply == QMessageBox::Yes) {  QList <QListWidgetItem \*>items=ui->listWidget->selectedItems();  QListWidgetItem \* selectedItem=items.at(0);  QVariant var = selectedItem->*data*(Qt::UserRole);  Lection lec =var.value<Lection>();  storage\_->*removeLection*(lec.id);  fillListWithUserId(storage\_->*getUser*().user\_id);  setDefault();  }  }  void MainWindow::**on\_pushButton\_3\_clicked**()  {  QList <QListWidgetItem \*>items=ui->listWidget->selectedItems();  QListWidgetItem \* selectedItem=items.at(0);  QVariant var = selectedItem->*data*(Qt::UserRole);  Lection lec =var.value<Lection>();  addDialog addialog\_(storage\_,lec.id,this);  addialog\_.setWindowTitle("editing entity");  addialog\_.*exec*();  disableEditing();  fillListWithUserId(storage\_->*getUser*().user\_id);  }  void MainWindow::**setDefault**(){  if (storage\_==nullptr){  disableEditing();  }  else{  ui->pushButton->setEnabled(true );  QList <QListWidgetItem \*>items=ui->listWidget->selectedItems();  if (items.size()<1){  disableEditing();  }  }  }  void MainWindow::**authfunc**(){  auth authdialog(this);  User user;  authdialog.setWindowTitle("logging in");  while(true){  if(authdialog.*exec*()){  string login = authdialog.getData().at(0);  string password = authdialog.getData().at(1);  if (storage\_->*getUserAuth*(login,password)) {  storage\_->*setUser*(\*(storage\_->*getUserAuth*(login,password)));  break;  } else  authdialog.setWindowTitle("try one more time");  }else {  exit(0);  }  }  fillListWithUserId(storage\_->*getUser*().user\_id);  }  void MainWindow::**disableEditing**(){  ui->pushButton\_2->setEnabled(false);  ui->pushButton\_3->setEnabled(false);  ui->label->*setVisible*(false);  ui->label\_2->*setVisible*(false);  ui->label\_3->*setVisible*(false);  ui->label\_4->*setVisible*(false);  ui->label\_5->*setVisible*(false);  ui->label\_6->*setVisible*(false);  ui->label\_7->*setVisible*(false);  }  void MainWindow::**enableEditing**(){  ui->pushButton\_2->setEnabled(true);  ui->pushButton\_3->setEnabled(true);  ui->label->*setVisible*(true);  ui->label\_2->*setVisible*(true);  ui->label\_3->*setVisible*(true);  ui->label\_4->*setVisible*(true);  ui->label\_5->*setVisible*(true);  ui->label\_6->*setVisible*(true);  ui->label\_7->*setVisible*(true);  }  ui->label\_7->setVisible(true);  } |

|  |
| --- |
| **storage.h** |
| #pragma once  #include <iostream>  #include <fstream>  #include <vector>  #include <sstream>  #include "optional.h"  #include "lection.h"  #include "topic.h"  #include "user.h"  using namespace std;  class Storage  {  User user;  public:  virtual ~***Storage***() {} // UPD (8.5.2019): added  virtual bool ***open***() = 0;  virtual bool ***close***() = 0;  // lections  virtual vector<Lection> ***getAllLections***(void) = 0;  virtual vector<Lection> ***getUserLections***(int user\_id) = 0;  virtual optional<Lection> ***getLectionById***(int lection\_id) = 0;  virtual bool ***updateLection***(const Lection &lection) = 0;  virtual bool ***removeLection***(int lection\_id) = 0;  virtual int ***insertLection***(const Lection &lection) = 0;  // topics  virtual vector<Topic> ***getAllTopics***(void) = 0;  virtual optional<Topic> ***getTopicById***(int topic\_id) = 0;  virtual bool ***updateTopic***(const Topic &topic) = 0;  virtual bool ***removeTopic***(int topic\_id) = 0;  virtual int ***insertTopic***(const Topic &topic) = 0;  // users  virtual optional<User> ***getUserAuth***(string & username, string & password) = 0;  // links  virtual vector<Topic> ***getAllLectionTopics***(int lection\_id) = 0;  virtual bool ***insertLectionTopic***(int lection\_id, int topic\_id) = 0;  virtual bool ***removeLectionTopic***(int lection\_id, int topic\_id) = 0;  virtual void ***setUser***(User user) = 0;  virtual User ***getUser***() = 0;  } |

|  |
| --- |
| **adddialog.cpp** |
| #include "adddialog.h"  #include "ui\_adddialog.h"  addDialog::**addDialog**(Storage\* storage\_,int lec\_id,QWidget \*parent):  QDialog(parent),  ui(new Ui::addDialog)  {  ui->setupUi(this);  this->storage\_=storage\_;  this->lection\_id=lec\_id;  this->lec\_id=&lection\_id;  Lection lec=storage\_->*getLectionById*(lec\_id).value();  ui->label->setText("Editing...");  ui->lineEdit->setText(QString::fromStdString(lec.title));  ui->lineEdit\_2->setText(QString::fromStdString(lec.lectorer));  ui->spinBox\_2->setValue(lec.students);  fillListsForEditedMainEntity(lec\_id);  }  addDialog::**addDialog**(Storage\* storage\_,QWidget \*parent ):  QDialog(parent),  ui(new Ui::addDialog)  {  ui->setupUi(this);  this->storage\_=storage\_;  fillListsForNewMainEntity();  ////  }  addDialog::~***addDialog***()  {  delete ui;  }  Lection addDialog::**data**(){  Lection lec;  lec.title=ui->lineEdit->text().toStdString();  lec.lectorer=ui->lineEdit\_2->text().toStdString();  lec.students=ui->spinBox\_2->value();  return lec;  }  void addDialog::**fillListsForNewMainEntity**(){  ui->listWidget->clear();  ui->listWidget\_2->clear();  for (Topic top : storage\_->*getAllTopics*()){  QVariant qVariant;  qVariant.setValue(top);  QListWidgetItem \*qLectionListItem = new QListWidgetItem();  qLectionListItem->setText(QString::fromStdString(top.topic));  qLectionListItem->*setData*(Qt::UserRole, qVariant);  ui->listWidget\_2->addItem(qLectionListItem);  }  }  void addDialog::**fillListsForEditedMainEntity**(int lec\_id){  ui->listWidget->clear();  ui->listWidget\_2->clear();  for (Topic top : storage\_->*getAllTopics*()){  QVariant qVariant;  qVariant.setValue(top);  QListWidgetItem \*qLectionListItem = new QListWidgetItem();  qLectionListItem->setText(QString::fromStdString(top.topic));  qLectionListItem->*setData*(Qt::UserRole, qVariant);  for (Topic top1 : storage\_->*getAllLectionTopics*(lec\_id)){  if (top.id==top1.id){  ui->listWidget->addItem(qLectionListItem);  continue;  }  }  ui->listWidget\_2->addItem(qLectionListItem);  }  }  void addDialog::**on\_buttonBox\_accepted**()  {  qDebug()<<"we are here";  if(lec\_id != nullptr){  Lection lec;  lec.id=\*lec\_id;  lec.title=ui->lineEdit->text().toStdString();  lec.lectorer=ui->lineEdit\_2->text().toStdString();  lec.students=ui->spinBox\_2->value();  qDebug()<<\*(lec\_id);  storage\_->*updateLection*(lec);  foreach(Topic top,storage\_->*getAllLectionTopics*(\*lec\_id)){  qDebug()<<\*lec\_id<<top.id;  storage\_->*removeLectionTopic*(\*lec\_id,top.id);  }  for(int i = 0; i < ui->listWidget->count(); ++i)  {  QListWidgetItem\* item = ui->listWidget->item(i);  QVariant var = item->*data*(Qt::UserRole);  Topic top =var.value<Topic>();  storage\_->*insertLectionTopic*(\*lec\_id,top.id);  }  }else {  Lection lec;  lec.title=ui->lineEdit->text().toStdString();  lec.lectorer=ui->lineEdit\_2->text().toStdString();  lec.students=ui->spinBox\_2->value();  int newlec\_id=storage\_->*insertLection*(lec);  }  }  void addDialog::**on\_pushButton\_2\_clicked**()  {  QList <QListWidgetItem \*>items=ui->listWidget\_2->selectedItems();  foreach(QListWidgetItem\* item, items){  ui->listWidget->addItem(ui->listWidget\_2->takeItem(ui->listWidget\_2->row(item)));  }  }  void addDialog::**on\_pushButton\_clicked**()  {  QList <QListWidgetItem \*>items=ui->listWidget->selectedItems();  foreach(QListWidgetItem\* item, items){  ui->listWidget\_2->addItem(ui->listWidget->takeItem(ui->listWidget->row(item)));  }  } |

|  |
| --- |
| **adddialog.h** |
| #ifndef ADDDIALOG\_H  #define ADDDIALOG\_H  #include<QDebug>  #include <QDialog>  #include "storage.h"  #include <QListWidget>  namespace Ui {  class addDialog;  }  class addDialog : public QDialog  {  Q\_OBJECT  Storage\* storage\_;  int lection\_id;  int\* lec\_id=nullptr;  public:  explicit addDialog(Storage\* storage\_,int cap\_id,QWidget \*parent = nullptr);  explicit addDialog(Storage\* storage\_,QWidget \*parent = nullptr);  ~***addDialog***();  void **fillListsForNewMainEntity**();  void **fillListsForEditedMainEntity**(int lec\_id);  Lection **data**();  private slots:  void **on\_buttonBox\_accepted**();  void **on\_pushButton\_2\_clicked**();  void **on\_pushButton\_clicked**();  private:  Ui::addDialog \*ui;  };  #endif // ADDDIALOG\_H |

**Приклади результатів**



**Висновки**

За час виконання лабораторної роботи навчився створювати мультивіконні графічні додатки. Реалізував ідентифікацію, аутентифікацію та авторизацію у програмі та опанував різні види зв'язків між таблицями реляційної бази даних.

**Iдентифікація** - процедура розпізнавання користувача у системі, за допомогою наперед визначеного унікального **імені (ідентифікатора)**. **Аутентифікація** - дія підтвердження належності користувачу наданої ним інформації про себе (ідентифікатора). **Авторизація** - це функція для визначення ***прав доступу*** користувача до ресурсів.

**Реєстрація користувача в системі** – процес надання ідентифікатора та аутентифікатора в систему.

**Моделі даних:** one-to-one, one-to-many, many-to-one, many-to-many.

**Хеш-функція** – функція-шифратор, що ускладнює доступ до аутентифікаторів та ідентифікаторів стороннім особам. **Хешувати паролі** потрібно для того, щоб забезпечити безпеку даних користувачів. **Коректність** введених **даних** перевіряється за допомогою хеш-функції, вшитої в програму.