Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(ФГБОУ ВО «СибГУТИ»)

*Кафедра прикладной математики и кибернетики*

Курсовая работа по дисциплине

«Визуальное программирование и человеко-машинное взаимодействие»

Выполнил: студент гр. ИП-513 Санин И. В.

Проверила: ст. пр. кафедры ПМиК

Лапова С. Г.

Новосибирск

2017

Оглавление

[1. Постановка задачи 3](#_Toc501744553)

[2. Результаты CWT-анализа 3](#_Toc501744554)

[3. Результаты GOMS-анализа 3](#_Toc501744555)

[4. Соответствие правилам Нильсена-Молиха 3](#_Toc501744556)

[5. Класс MainWindow 4](#_Toc501744557)

[6. Класс Dialog 4](#_Toc501744558)

[7. Класс userdata 4](#_Toc501744559)

[8. Класс DataBase 5](#_Toc501744560)

[9. Листинг 5](#_Toc501744561)

[10. Результаты работы программы 21](#_Toc501744562)

[11. Заключение 26](#_Toc501744563)

# Постановка задачи

Разработать программу для учета посетителей бассейна. Реализовать базу данных посетителей с возможностью поиска клиентов, вывода индивидуальной информации, информации об абонементах, редактировании информации.

# Результаты CWT-анализа

В результате проведения CWT-анализа было выявлено, что при размещении функционала на панель инструментов пользователь мог не обнаружить необходимый элемент управления для осуществления необходимого действия.

# Результаты GOMS-анализа

Рассматривалось два варианта реализации открытия личной карточки посетителя. Первым вариантом была кнопка, активирующаяся при выделении записи кликом мыши по ней, вторым – открытие двойным кликом мыши по записи. В первом случае для того, чтобы пользователь переключался между функциональными блоками, ему понадобится следующая последовательность операций - MPBPB, где *B* – клик кнопкой мыши, *P* – наведение указателя мыши, *M* – мыслительная подготовка. В итоге время, которое пользователь в среднем будет затрачивать на достижение данной цели, будет равно 1,35 + 2 \* (1,1 + 0,2) = 3,95 секундам. Во втором случае будем иметь такую последовательность – MPBB, что равно 1,35 + 1,1 + 0,2 + 0.2 = 2,85 секундам, что значительно меньше, чем в первом случае. Поэтому для реализации был выбран второй, более быстрый вариант.

# Соответствие правилам Нильсена-Молиха

Вся важная информация и весь основной функционал размещены так, чтобы добиться максимального следования задачам пользователя. При работе пользователя с программой представляется только необходимая пользователю информация. Все не нужные в выполняемой задаче детали интерфейса не будут отображаться в приложении из-за разделения приложения на функциональные блоки. В приложении нет слов и понятий, которые бы были не понятны предполагаемому пользователю данной программы. При перемещениях по разным частям приложения, вся информация и наработки останутся не тронутыми и готовыми к редактированию. Если пользователь попадёт в часть системы, которая его не интересует, он с лёгкостью сможет покинуть её, просто перейдя в нужную ему вкладку. В программе предусмотрены сообщения об ошибках, в которых содержится информация о допущенной ошибке.

# Класс MainWindow

Этот класс реализует главное окно программы. В данном классе реализован интерфейс отображения базы данных пользователей и абонементов при помощи виджетов QTableView (dbView и dbView\_2) и отображении в них моделей базы данных при помощи класса QSqlTableModel. При помощи обработки сигналов doubleClicked(QModelIndex &) происходит вывод информации о записи в таблице (пользователе или абонементе) в отдельном окне. Также в данном классе реализован поиск клиентов по ФИО и дате посещения при помощи виджетов QLineEdit и связанных с ними сигналов textChanged(). При изменении текста в виджете, связанном с ФИО (FNsearch), происходит поиск всех ФИО в базе данных, содержащих сочетания символов, которые ввел пользователь, модель отображения обновляется и перерисовывается таблица. При изменении текста в виджете, связанном с датой (minData и maxData), осуществляется поиск такими же методами.

# Класс Dialog

Этот класс реализует диалоговое окно для редактирования данных. Параметры конструктора данного класса определяют, какие элементы для редактирования информации необходимо вывести. Класс обладает методами получения изначальной информации и названий типов данных информации (для отображения пользователю), которую необходимо редактировать и передачи отредактированной информации.

# Класс userdata

Это класс окна – карточки о клиенте. В конструкторе происходит подключение к базе данных. По умолчанию данное окно является скрытым. При попытке пользователя отобразить индивидуальную информацию о клиенте из главного окна запускается метод setData(…), передающий уникальный идентификатор клиента, благодаря которому данное окно получает всю информацию о нем из базы данных и производит отображение информации с помощью элементов интерфейса. Эти же элементы позволяют при необходимости редактировать информацию.

# Класс DataBase

Данный класс является интерфейсом подключения базы данных, используя класс QSqlDatabase, и выведения возможных ошибок. Визуальной части не имеет.

# Листинг

*main.cpp:*

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QFile styleF;

styleF.setFileName(":/qss/styles.css");

styleF.*open*(QFile::ReadOnly);

QString qssStr = styleF.readAll();

a.setStyleSheet(qssStr);

MainWindow w;

w.show();

return a.exec();

}

*database.h:*

#ifndef DATABASE\_H

#define DATABASE\_H

#include <QObject>

#include <QtSql>

#include <QSqlQuery>

#include <QSqlError>

#include <QSqlDatabase>

#include <QFile>

#include <QDate>

#include <QDebug>

/\* Директивы имен таблицы, полей таблицы и базы данных \*/

#define DATABASE\_HOSTNAME "PoolDataBase"

#define TABLE1 "PoolVisit"

#define TABLE1\_FN "Name"

#define TABLE1\_PNUM "Phone"

#define TABLE1\_GND "Gender"

#define TABLE1\_PHOTO "Doc\_note"

#define TABLE1\_NOTDR "Doc\_note\_dur"

// Первая колонка содержит Autoincrement ID

#define TABLE2 "Abonements"

#define TABLE2\_DOF "DateOfReg"

#define TABLE2\_SUBDR "SubDuration"

#define TABLE2\_COST "Cost"

#define TABLE2\_USRID "UserID"

#define TABLE2\_ISPAID "IsPaid"

#define TABLE3 "AbonType"

#define TABLE3\_NAME "Name"

#define TABLE3\_COST "Cost"

#define TABLE3\_DUR "Dur"

class DataBase : public QObject

{

Q\_OBJECT

public:

explicit DataBase(QObject \*parent = 0);

~*DataBase*();

/\* Методы для непосредственной работы с классом

\* Подключение к базе данных и вставка записей в таблицу

\*/

void connectToDataBase(const QString &name);

QSqlDatabase getDB();

private:

// Сам объект базы данных, с которым будет производиться работа

QSqlDatabase db;

private:

// Внутренние методы для работы с базой данных

bool openDataBase(const QString &name); // Открытие базы данных

bool restoreDataBase(const QString &name); // Восстановление базы данных

void closeDataBase(); // Закрытие базы данных

};

#endif // DATABASE\_H

*database.cpp:*

#include "database.h"

DataBase::DataBase(QObject \*parent) : QObject(parent)

{

}

DataBase::~*DataBase*()

{

this->closeDataBase();

}

// Метод для подключения к базе данных

void DataBase::connectToDataBase(const QString &name)

{

/\* Перед подключением к базе данных производим проверку на её существование.

\* В зависимости от результата производим открытие базы данных или её восстановление

\*/

if(!QFile(name).exists()){

this->restoreDataBase(name);

} else {

this->openDataBase(name);

}

}

QSqlDatabase DataBase::getDB()

{

return db;

}

// Метод для восстановления базы данных

bool DataBase::restoreDataBase(const QString &name)

{

// Если база данных открылась ...

if(this->openDataBase(name)){

qDebug() << "Can open DB";

return true;

} else {

qDebug() << "Can't open DB";

qDebug() << db.lastError().text();

return false;

}

return false;

}

// Метод для открытия базы данных

bool DataBase::openDataBase(const QString &name)

{

/\* База данных открывается по заданному пути

\* и имени базы данных, если она существует

\*/

db = QSqlDatabase::addDatabase("QSQLITE");

db.setHostName(DATABASE\_HOSTNAME);

db.setDatabaseName(name);

if(db.open()){

return true;

} else {

return false;

}

}

// Метод для закрытия базы данных

void DataBase::closeDataBase()

{

db.close();

}

*dialog.h:*

#ifndef DIALOG\_H

#define DIALOG\_H

#include <QDialog>

#include <QDebug>

#include <QSqlTableModel>

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

Q\_OBJECT

public:

Dialog(QWidget \*parent = 0,

bool firstLine = false,

bool secLine = false,

bool thirdLine = false,

bool radios = false,

bool checkbox = false,

bool combobox = false);

~*Dialog*();

void getData(QString &first, QString &second, QString &third,

bool &radio, bool &checkbox, QString &combobox);

void setFirstLine(QString label, QString line);

void setSecLine(QString label, QString line);

void setThirdLine(QString label, QString line);

void setRadio(QString first, QString second);

void setCheckBox(QString check, bool state);

void setComboBox(QSqlTableModel \*model, int column);

void setHeader(QString header);

private:

Ui::Dialog \*ui;

};

#endif // DIALOG\_H

*dialog.cpp:*

#include "dialog.h"

#include "ui\_dialog.h"

Dialog::Dialog(QWidget \*parent,

bool firstLine,

bool secLine,

bool thirdLine,

bool radios,

bool checkbox,

bool combobox) :

QDialog(parent),

ui(new Ui::Dialog)

{

ui->setupUi(this);

ui->firstLabel->hide();

ui->firstLine->hide();

ui->secLabel->hide();

ui->secLine->hide();

ui->thirdLabel->hide();

ui->thirdLine->hide();

ui->radioGroup->hide();

ui->checkBox->hide();

ui->comboBox->hide();

ui->radio1->setChecked(true);

if(firstLine) {

ui->firstLabel->show();

ui->firstLine->show();

}

if(secLine) {

ui->secLabel->show();

ui->secLine->show();

}

if(thirdLine) {

ui->thirdLabel->show();

ui->thirdLine->show();

}

if(radios)

ui->radioGroup->show();

if(checkbox)

ui->checkBox->show();

if(combobox)

ui->comboBox->show();

}

Dialog::~*Dialog*()

{

delete ui;

}

void Dialog::getData(QString &first, QString &second, QString &third, bool &radio, bool &checkbox, QString &combobox)

{

first = ui->firstLine->text();

second = ui->secLine->text();

third = ui->thirdLine->text();

if(ui->radio1->isChecked())

radio = true;

else if(ui->radio2->isChecked())

radio = false;

else

qDebug() << "error";

if(ui->checkBox->isChecked())

checkbox = true;

else

checkbox = false;

combobox = ui->comboBox->currentText();

}

void Dialog::setFirstLine(QString label, QString line)

{

ui->firstLabel->setText(label);

ui->firstLine->setText(line);

}

void Dialog::setSecLine(QString label, QString line)

{

ui->secLabel->setText(label);

ui->secLine->setText(line);

}

void Dialog::setThirdLine(QString label, QString line)

{

ui->thirdLabel->setText(label);

ui->thirdLine->setText(line);

}

void Dialog::setRadio(QString first, QString second)

{

ui->radio1->setText(first);

ui->radio2->setText(second);

}

void Dialog::setCheckBox(QString check, bool state)

{

ui->checkBox->setText(check);

ui->checkBox->setChecked(state);

}

void Dialog::setComboBox(QSqlTableModel \*model, int column)

{

ui->comboBox->setModel(model);

ui->comboBox->setModelColumn(column);

}

void Dialog::setHeader(QString header)

{

this->setWindowTitle(header);

}

*mainwindow.h:*

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMessageBox>

#include "database.h"

#include "userdata.h"

#include "dialog.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_FNsearch\_textChanged(const QString &arg1);

void on\_minData\_textChanged(const QString &arg1);

void on\_maxData\_textChanged(const QString &arg1);

void on\_dbView\_doubleClicked(const QModelIndex &index);

void on\_addAbon\_clicked();

void on\_dbView\_2\_doubleClicked(const QModelIndex &index);

void on\_newUser\_clicked();

private:

Ui::MainWindow \*ui;

QSqlTableModel \*mod;

QSqlTableModel \*mod2;

QSqlRelationalTableModel \*mod3;

DataBase \*db;

userdata \*ud;

void date\_comp(const QString &arg1, const QString &arg2);

};

#endif // MAINWINDOW\_H

*mainwindow.cpp:*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

db = new DataBase(this);

db->connectToDataBase("/home/monstruos/GitHub/Course\_projects/3\_course/1\_part/VPandHCI/database/poolbase.db");

this->setWindowTitle("База данных");

ud = new userdata(db, this);

ud->hide();

this->setFocus();

ui->FN->setStyleSheet("color: #ffffff");

ui->From->setStyleSheet("color: #ffffff");

ui->To->setStyleSheet("color: #ffffff");

ui->dbView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->dbView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->dbView->setEditTriggers(QAbstractItemView::NoEditTriggers);

mod = new QSqlTableModel();

mod->*setTable*(TABLE1);

mod->*setHeaderData*(1, Qt::Horizontal, QObject::tr("ФИО"));

mod->*setHeaderData*(2, Qt::Horizontal, QObject::tr("Мобильный телефон"));

mod->*setHeaderData*(3, Qt::Horizontal, QObject::tr("Пол"));

mod->*select*();

ui->dbView->*setModel*(mod);

ui->dbView->setColumnHidden(0, true);

ui->dbView->setColumnHidden(4, true);

ui->dbView->setColumnHidden(5, true);

ui->dbView->setAlternatingRowColors(true);

ui->dbView->resizeColumnsToContents();

ui->dbView->resizeRowsToContents();

ui->dbView->horizontalHeader()->setStretchLastSection(true);

mod2 = new QSqlTableModel();

mod2->*setTable*(TABLE3);

mod2->*setHeaderData*(1, Qt::Horizontal, QObject::tr("Название"));

mod2->*setHeaderData*(2, Qt::Horizontal, QObject::tr("Стоимость"));

mod2->*setHeaderData*(3, Qt::Horizontal, QObject::tr("Длительность (дней)"));

mod2->*select*();

ui->dbView\_2->setSelectionMode(QAbstractItemView::SingleSelection);

ui->dbView\_2->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->dbView\_2->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->dbView\_2->*setModel*(mod2);

ui->dbView\_2->setColumnHidden(0, true);

ui->dbView\_2->setAlternatingRowColors(true);

ui->dbView\_2->resizeColumnsToContents();

ui->dbView\_2->resizeRowsToContents();

ui->dbView\_2->horizontalHeader()->setStretchLastSection(true);

mod3 = new QSqlRelationalTableModel();

mod3->*setTable*(TABLE2);

mod3->*setHeaderData*(1, Qt::Horizontal, QObject::tr("Дата регистрации абонемента"));

mod3->*setHeaderData*(2, Qt::Horizontal, QObject::tr("Длительность абонемента"));

mod3->*setHeaderData*(3, Qt::Horizontal, QObject::tr("ФИО"));

mod3->*setHeaderData*(4, Qt::Horizontal, QObject::tr("Оплачено"));

mod3->*setHeaderData*(5, Qt::Horizontal, QObject::tr("Тип абонемента"));

mod3->*setRelation*(3, QSqlRelation("PoolVisit", "id", "Name"));

mod3->*setRelation*(5, QSqlRelation("AbonType", "AbonId", "Name"));

mod3->*select*();

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::date\_comp(const QString &arg1, const QString &arg2)

{

ui->dbView->setSelectionMode(QAbstractItemView::MultiSelection);

ui->dbView->clearSelection();

QDate pointFrom = QDate::fromString(arg1, "yyyy-MM-dd");

QDate pointTo = QDate::fromString(arg2, "yyyy-MM-dd");

if (!pointFrom.isValid() && !pointTo.isValid()) {

if(ui->dbView->model() != mod) {

ui->dbView->*setModel*(mod);

ui->dbView->horizontalHeader()->swapSections(2, 3);

ui->dbView->horizontalHeader()->swapSections(1, 3);

}

ui->dbView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->dbView->setColumnHidden(4, true);

ui->dbView->setColumnHidden(5, true);

return;

}

if(pointTo.isValid() && pointFrom.isValid()) {

mod3->*setFilter*(QString("DateOfReg <= '" + pointTo.toString("yyyy-MM-dd") + "' AND SubDuration >= '" + pointFrom.toString("yyyy-MM-dd") + "'"));

} else {

if(pointTo.isValid())

mod3->*setFilter*(QString("DateOfReg <= " + pointTo.toString("yyyy-MM-dd")));

if(pointFrom.isValid())

mod3->*setFilter*(QString("SubDuration >= " + pointFrom.toString("yyyy-MM-dd")));

}

if(ui->dbView->model() != mod3) {

ui->dbView->*setModel*(mod3);

ui->dbView->horizontalHeader()->swapSections(1, 3);

ui->dbView->horizontalHeader()->swapSections(2, 3);

}

ui->dbView->setColumnHidden(4, false);

ui->dbView->setColumnHidden(5, false);

ui->dbView->setAlternatingRowColors(true);

ui->dbView->resizeColumnsToContents();

ui->dbView->resizeRowsToContents();

}

void MainWindow::on\_FNsearch\_textChanged(const QString &arg1)

{

ui->dbView->setSelectionMode(QAbstractItemView::MultiSelection);

ui->dbView->clearSelection();

QSqlTableModel \*m = (QSqlTableModel \*)ui->dbView->model();

m->*setFilter*(QString("Name LIKE '%" + arg1 + "%'"));

m->*select*();

if(arg1 == "") {

ui->dbView->setSelectionMode(QAbstractItemView::SingleSelection);

}

}

void MainWindow::on\_minData\_textChanged(const QString &arg1)

{

date\_comp(arg1, ui->maxData->text());

}

void MainWindow::on\_maxData\_textChanged(const QString &arg1)

{

date\_comp(ui->minData->text(), arg1);

}

void MainWindow::on\_dbView\_doubleClicked(const QModelIndex &index)

{

QString iden = mod->*data*(mod->*index*(index.row(), 1)).toString();

mod->*setFilter*(QString("Name = '" + iden + "'"));

mod->*select*();

ud->setData(mod->*data*(mod->*index*(0, 0)).toInt());

ud->*exec*();

mod->*setFilter*("");

mod->*select*();

ui->dbView->repaint();

}

void MainWindow::on\_addAbon\_clicked()

{

Dialog a(this, true, true, true, false, false);

a.setHeader("Новый абонемент");

a.setFirstLine("Название:", "");

a.setSecLine("Стоимость:", "");

a.setThirdLine("Длительность:", "");

int res = a.*exec*();

if(res == Dialog::Accepted) {

QSqlQuery query;

QString name, cost, dur, buf3;

bool buf1, buf2;

a.getData(name, cost, dur, buf1, buf2, buf3);

if(name.isEmpty() || cost.isEmpty() || dur.isEmpty()) {

QMessageBox::warning(this,

"Ошибка",

"Не удалось создать абонемент: некоторые поля не были заполнены");

return;

}

query.prepare("INSERT INTO AbonType (Name, Cost, Dur)"

"VALUES(:name, :cost, :dur);");

query.bindValue(":name", name);

query.bindValue(":cost", cost.toInt());

query.bindValue(":dur", dur.toInt());

query.exec();

mod2->*select*();

ui->dbView\_2->repaint();

ui->dbView\_2->resizeColumnsToContents();

ui->dbView\_2->resizeRowsToContents();

}

}

void MainWindow::on\_dbView\_2\_doubleClicked(const QModelIndex &index)

{

int row = index.row();

QString name = mod2->*data*(mod2->*index*(row, 1)).toString();

QString cost = mod2->*data*(mod2->*index*(row, 2)).toString();

QString dur = mod2->*data*(mod2->*index*(row, 3)).toString();

Dialog a(this, true, true, true, false, false);

a.setHeader("Изменить абонемент");

a.setFirstLine("Название:", name);

a.setSecLine("Стоимость:", cost);

a.setThirdLine("Длительность:", dur);

int res = a.*exec*();

if(res == Dialog::Accepted) {

QSqlQuery query;

bool buf1, buf2;

QString buf3;

a.getData(name, cost, dur, buf1, buf2, buf3);

query.prepare("UPDATE AbonType SET "

"Name = :name, Cost = :cost, Dur = :dur "

"WHERE AbonId = :id");

query.bindValue(":id", row + 1);

query.bindValue(":name", name);

query.bindValue(":cost", cost.toInt());

query.bindValue(":dur", dur.toInt());

query.exec();

mod2->*select*();

ui->dbView\_2->repaint();

}

}

void MainWindow::on\_newUser\_clicked()

{

Dialog a(this, true, true, false, true, false);

a.setHeader("Новый пользователь");

a.setFirstLine("ФИО:","");

a.setSecLine("Телефон:", "");

a.setRadio("Мужчина", "Женщина");

int res = a.*exec*();

if(res == Dialog::Accepted) {

QSqlQuery query;

QString name, phone, buf1, buf3; // variables for getData()

bool gender, buf2;

a.getData(name, phone, buf1, gender, buf2, buf3);

if(name.isEmpty() || phone.isEmpty())

{

QMessageBox::warning(this,

"Ошибка",

"Не удалось создать посетителя: некоторые поля не были заполнены");

return;

}

QString g;

if(gender == true) {

g = "Мужской";

} else {

g = "Женский";

}

query.prepare("INSERT INTO PoolVisit (Name, Phone, Gender)"

"VALUES(:name, :phone, :gend);");

query.bindValue(":name", name);

query.bindValue(":phone", phone.toInt());

query.bindValue(":gend", g);

query.exec();

mod->*setFilter*("");

mod->*select*();

mod->*setFilter*(QString("Name='" + name + "' AND Phone=" + phone + " AND Gender='" + g + "'"));

mod->*select*();

int rec = mod->record(0).field(0).value().toInt();

ud->setData(rec);

ud->*exec*();

mod->*setFilter*("");

mod->*select*();

ui->dbView->repaint();

}

}

*userdata.h:*

#ifndef USERDATA\_H

#define USERDATA\_H

#include <QDialog>

#include "database.h"

#include "dialog.h"

#include <QMessageBox>

#include <QFileDialog>

namespace Ui {

class userdata;

}

class userdata : public QDialog

{

Q\_OBJECT

public:

explicit userdata(DataBase \*db, QWidget \*parent = 0);

~*userdata*();

void setData(int userID);

private slots:

void on\_addDocNote\_clicked();

void on\_addAbon\_clicked();

void on\_includeAbon\_clicked();

void on\_cancelAbon\_clicked();

void on\_prevAbon\_doubleClicked(const QModelIndex &index);

void on\_pushButton\_clicked();

private:

Ui::userdata \*ui;

DataBase \*b;

QSqlRelationalTableModel \*model;

QSqlTableModel \*abonModel;

int id;

QString name;

QString phone;

QString gender;

QDate DateOfReg;

QDate SubDuration;

QPixmap DocNote;

QDate DocNoteDur;

};

#endif // USERDATA\_H

*userdata.cpp:*

#include "userdata.h"

#include "ui\_userdata.h"

userdata::userdata(DataBase \*db, QWidget \*parent) :

QDialog(parent),

ui(new Ui::userdata)

{

b = db;

ui->setupUi(this);

ui->username->setStyleSheet("color:#ffffff; font-size: 16px");

ui->gender->setStyleSheet("color:#ffffff; font-size: 14px");

ui->phone->setStyleSheet("color:#ffffff; font-size: 14px");

ui->abonDurFromLabel->setStyleSheet("color:#ffffff");

ui->abonDurToLabel->setStyleSheet("color:#ffffff");

ui->untilLabel->setStyleSheet("color:#ffffff");

ui->abonDurToLine->setText("");

ui->abonDurToLine->setText("");

ui->isCost->setChecked(false);

ui->prevAbon->setSelectionMode(QAbstractItemView::SingleSelection);

ui->prevAbon->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->prevAbon->setEditTriggers(QAbstractItemView::NoEditTriggers);

this->setWindowTitle("Информация о пользователе/Редактирование");

model = new QSqlRelationalTableModel();

model->*setTable*(TABLE2);

model->*setHeaderData*(1, Qt::Horizontal, QObject::tr("Дата регистрации"));

model->*setHeaderData*(2, Qt::Horizontal, QObject::tr("Длительность абонемента"));

model->*setHeaderData*(4, Qt::Horizontal, QObject::tr("Уплачено:"));

model->*setHeaderData*(5, Qt::Horizontal, QObject::tr("Тип абонемента:"));

model->*setRelation*(4, QSqlRelation("YN", "id", "Name"));

model->*setRelation*(5, QSqlRelation("AbonType", "AbonId", "Name"));

abonModel = new QSqlTableModel();

abonModel->*setTable*(TABLE3);

}

userdata::~*userdata*()

{

delete ui;

}

void userdata::setData(int userID)

{

model->*setFilter*("");

model->*select*();

abonModel->*setFilter*("");

abonModel->*select*();

ui->abonType->setModel(abonModel);

ui->abonType->setModelColumn(1);

QSqlQuery \*s = new QSqlQuery();

s->exec("SELECT \* FROM PoolVisit WHERE id = " + QString::number(userID));

s->next();

id = s->value(0).toInt();

name = s->value(1).toString();

phone = s->value(2).toString();

gender = s->value(3).toString();

QByteArray BDN = s->value(4).toByteArray();

DocNote.loadFromData(BDN);

if(!DocNote.isNull())

DocNote = DocNote.scaledToWidth(471);

DocNoteDur = QDate::fromString(s->value(5).toString(), "yyyy-MM-dd");

ui->isCost->setChecked(s->value(6).toInt());

ui->username->setText(name);

ui->phone->setText("Телефон: " + phone);

ui->gender->setText("Пол: " + gender);

ui->docphoto->setPixmap(DocNote);

ui->untilLine->setText(DocNoteDur.toString("yyyy-MM-dd"));

int status = DocNoteDur.daysTo(QDate::currentDate());

QPalette pal = ui->untilLine->palette();

if(status >= 0)

pal.setColor(ui->untilLine->foregroundRole(), Qt::red);

else if(status > -7)

pal.setColor(ui->untilLine->foregroundRole(), Qt::darkYellow);

else

pal.setColor(ui->untilLine->foregroundRole(), Qt::green);

ui->untilLine->setPalette(pal);

QString cond = "SELECT DateOfReg, SubDuration, AbonId FROM Abonements WHERE UserID = " + QString::number(id) +

" AND DateOfReg <= '" + QDate::currentDate().toString("yyyy-MM-dd") +

"' AND SubDuration >= '" + QDate::currentDate().toString("yyyy-MM-dd") + "'";

s->exec(cond);

ui->includeAbon->hide();

ui->cancelAbon->hide();

if(s->next()) {

ui->addAbon->hide();

ui->groupBox\_2->show();

ui->abonDurFromLine->setText(s->value(0).toString());

ui->abonDurToLine->setText(s->value(1).toString());

ui->abonType->setCurrentIndex(s->value(2).toInt() - 1);

} else {

ui->addAbon->show();

ui->groupBox\_2->hide();

}

model->*setFilter*("UserID = " + QString::number(id) +

" AND NOT (DateOfReg <= '" + QDate::currentDate().toString("yyyy-MM-dd") +

"' AND SubDuration >= '" + QDate::currentDate().toString("yyyy-MM-dd") + "')");

model->*select*();

ui->prevAbon->*setModel*(model);

ui->prevAbon->setColumnHidden(0, true);

ui->prevAbon->setColumnHidden(3, true);

ui->prevAbon->setAlternatingRowColors(true);

ui->prevAbon->resizeColumnsToContents();

ui->prevAbon->resizeRowsToContents();

}

void userdata::on\_addDocNote\_clicked()

{

QString picPath = QFileDialog::getOpenFileName(this,

tr("Открыть изображение"),

"",

tr("Images (\*.png \*.jpg \*.jpeg)"));

if(!picPath.isEmpty()) {

QFile file(picPath);

file.*open*(QFile::ReadWrite);

QByteArray bImage = file.readAll();

DocNote.loadFromData(bImage);

DocNote = DocNote.scaledToWidth(471);

ui->docphoto->setPixmap(DocNote);

QSqlQuery query;

query.prepare("UPDATE PoolVisit SET Doc\_Note = :Note WHERE id = :id");

query.bindValue(":Note", bImage);

query.bindValue(":id", id);

if(!query.exec()) {

qDebug() << "Failed to load data in database";

}

}

}

void userdata::on\_addAbon\_clicked()

{

ui->abonDurFromLine->setText("");

ui->abonDurToLine->setText("");

ui->groupBox\_2->setTitle("Новый абонемент");

ui->addAbon->hide();

ui->includeAbon->show();

ui->cancelAbon->show();

ui->groupBox\_2->show();

ui->abonDurFromLine->setText("");

ui->abonDurToLine->setText("");

ui->isCost->setChecked(false);

}

void userdata::on\_includeAbon\_clicked()

{

QDate DateOfReg;

DateOfReg = QDate::fromString(ui->abonDurFromLine->text(), "yyyy-MM-dd");

QDate SubDuration;

SubDuration = QDate::fromString(ui->abonDurToLine->text(), "yyyy-MM-dd");

int isPaid = ui->isCost->isChecked();

int AbonID = ui->abonType->currentIndex() + 1;

if(!DateOfReg.isValid() || !SubDuration.isValid()) {

ui->groupBox\_2->setTitle("Неправильно введена дата");

return;

}

QSqlQuery query;

query.prepare("INSERT INTO Abonements (DateOfReg, SubDuration, UserID, isPaid, AbonId) VALUES (:DoR, :SDur, :UID, :isP, :AID)");

query.bindValue(":DoR", DateOfReg.toString("yyyy-MM-dd"));

query.bindValue(":SDur", SubDuration.toString("yyyy-MM-dd"));

query.bindValue(":UID", id);

query.bindValue(":isP", isPaid);

query.bindValue(":AID", AbonID);

if(!query.exec()) {

qDebug() << "Failed to load data in database";

}

ui->groupBox\_2->setTitle("Текущий абонемент");

ui->includeAbon->hide();

ui->cancelAbon->hide();

}

void userdata::on\_cancelAbon\_clicked()

{

ui->addAbon->show();

ui->includeAbon->hide();

ui->cancelAbon->hide();

ui->groupBox\_2->hide();

}

void userdata::on\_prevAbon\_doubleClicked(const QModelIndex &index)

{

int iden = model->*data*(model->*index*(index.row(), 0)).toInt();

Dialog a(this, true, true, false, false, true, true);

model->*setFilter*(QString("idAbonements=" + QString::number(iden)));

model->*select*();

QSqlRecord rec = model->record(0);

QString from = rec.field(1).value().toString();

QString to = rec.field(2).value().toString();

QString isCost = rec.field(4).value().toString();

QString AbonId = rec.field(5).value().toString();

int intCost, idAbonement;

QSqlQuery \*s = new QSqlQuery();

s->exec(QString("SELECT id FROM YN WHERE Name = '" + isCost + "'"));

s->next();

intCost = s->value(0).toInt();

s->exec(QString("SELECT AbonId FROM AbonType WHERE Name = '" + AbonId + "'"));

s->next();

idAbonement = s->value(0).toInt();

a.setHeader("Просмотр и редактирование абонемента");

a.setFirstLine("Действителен с:", from);

a.setSecLine("Действителен до:", to);

a.setCheckBox("Оплачено:", (bool)intCost);

a.setComboBox(abonModel, 1, idAbonement);

int res = a.*exec*();

if(res == Dialog::Accepted) {

QSqlQuery query;

QString RegFrom, RegTo, buf1, TypeAbon;

bool buf2, cost;

a.getData(RegFrom, RegTo, buf1, buf2, cost, TypeAbon);

int c = cost;

//parsing to Date

QDate dateFrom = QDate::fromString(RegFrom, "yyyy-MM-dd");

QDate dateTo = QDate::fromString(RegTo, "yyyy-MM-dd");

if(!dateFrom.isValid() || !dateTo.isValid()) {

QMessageBox::warning(this,

"Ошибка",

"Не удалось изменить данные. Возможно, даты не соответствовали формату (YYYY-MM-DD).");

return;

}

//finding Abonement type

abonModel->*setFilter*(QString("Name = " + TypeAbon));

abonModel->*select*();

query.prepare("UPDATE Abonements SET "

"DateOfReg = :dof, "

"SubDuration = :sd, "

"isPaid = :pd, "

"AbonId = :aid "

"WHERE idAbonements = :id");

query.bindValue(":id", iden);

query.bindValue(":dof", dateFrom.toString("yyyy-MM-dd"));

query.bindValue(":sd", dateTo.toString("yyyy-MM-dd"));

query.bindValue(":pd", c);

query.bindValue(":aid", abonModel->record(0).field(0).value().toInt() + 1);

query.exec();

abonModel->*setFilter*("");

abonModel->*select*();

model->*setFilter*("UserID = " + QString::number(id) +

" AND NOT (DateOfReg <= '" + QDate::currentDate().toString("yyyy-MM-dd") +

"' AND SubDuration >= '" + QDate::currentDate().toString("yyyy-MM-dd") + "')");

model->*select*();

ui->prevAbon->repaint();

}

}

void userdata::on\_pushButton\_clicked()

{

Dialog a(this, true, true, false, true);

a.setHeader("Редактирование контактных данных");

a.setFirstLine("ФИО:", name);

a.setSecLine("Телефон:", phone);

a.setRadio("Мужчина", "Женщина");

int res = a.*exec*();

if(res == Dialog::Accepted) {

QSqlQuery query;

QString n, p, buf1, buf3;

bool radio, buf2;

a.getData(n, p, buf1, radio, buf2, buf3);

//parsing to phone to int

bool ok = true;

p.toLongLong(&ok);

if(!ok) {

QMessageBox::warning(this,

"Ошибка",

"Не удалось изменить данные. Несоответствие данных формату");

return;

}

name = n;

phone = p;

radio ? gender = "Мужской" : gender = "Женский";

query.prepare("UPDATE PoolVisit SET "

"Name = :n, "

"Phone = :p, "

"Gender = :g "

"WHERE id = :id");

query.bindValue(":id", id);

query.bindValue(":n", name);

query.bindValue(":p", phone);

query.bindValue(":g", gender);

query.exec();

ui->phone->setText("Телефон: " + phone);

ui->gender->setText("Пол: " + gender);

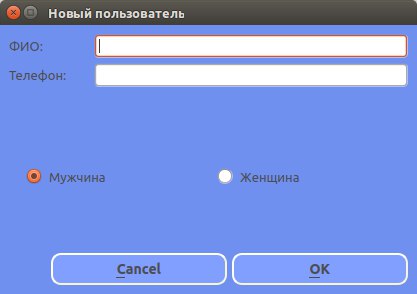
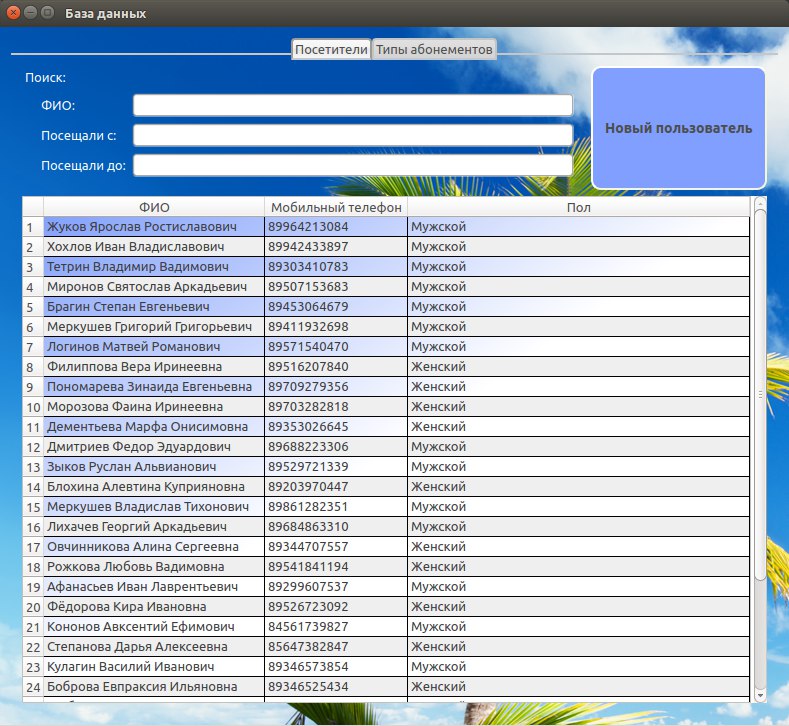
ui->username->setText(name);

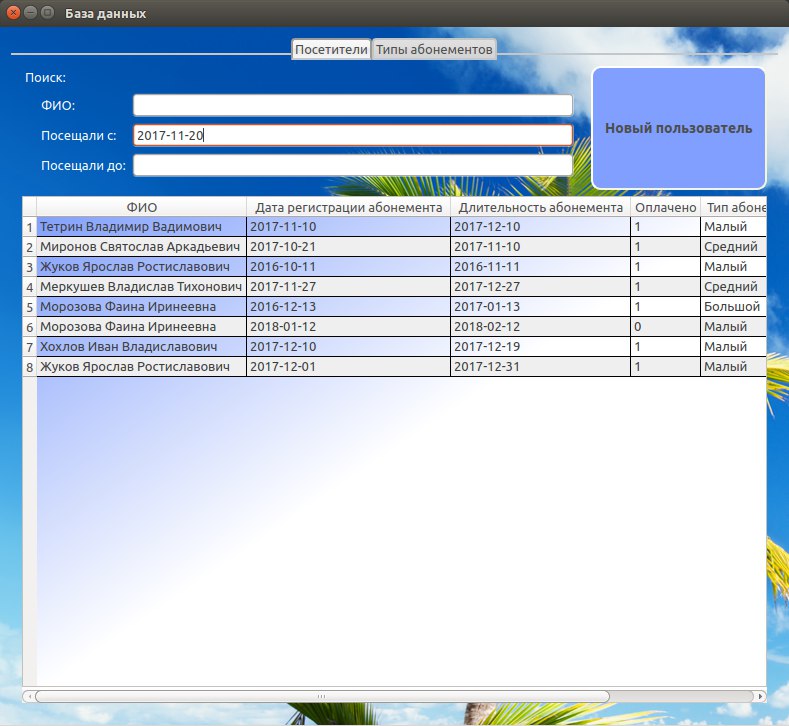
ui->prevAbon->repaint();

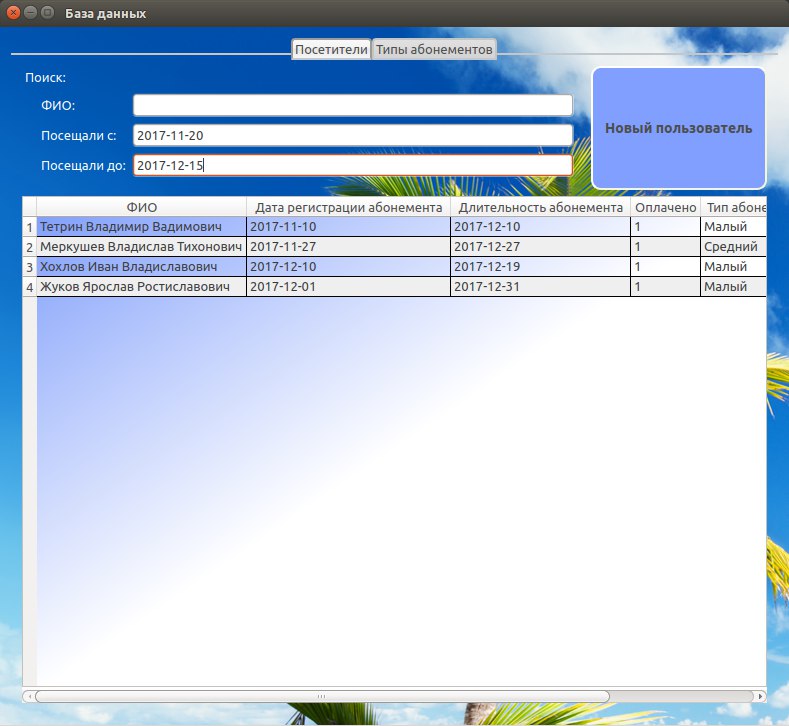
}

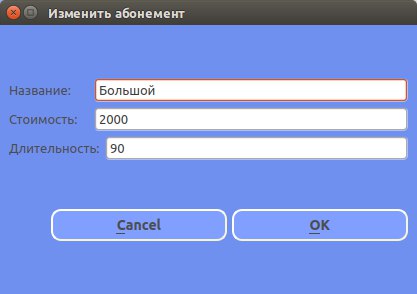
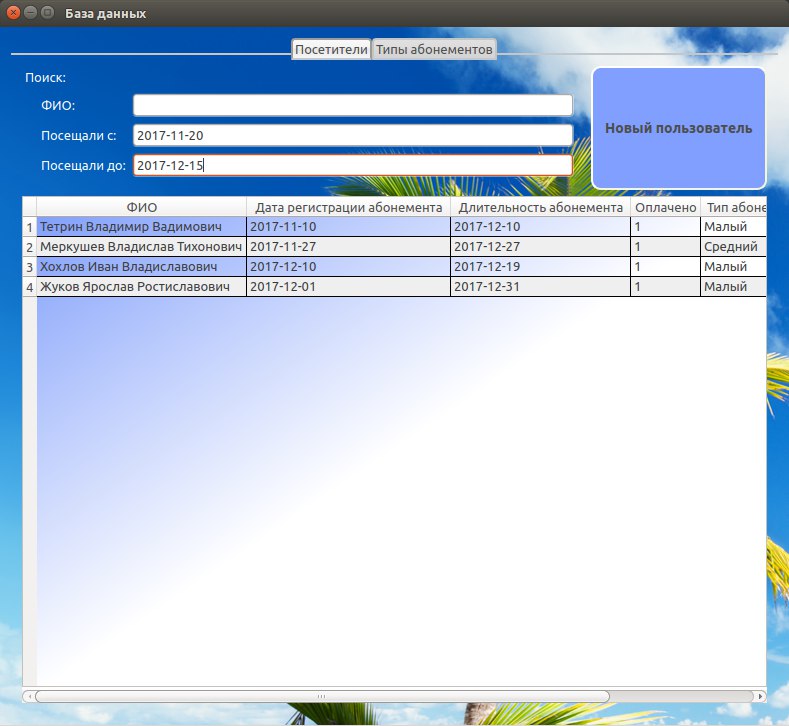
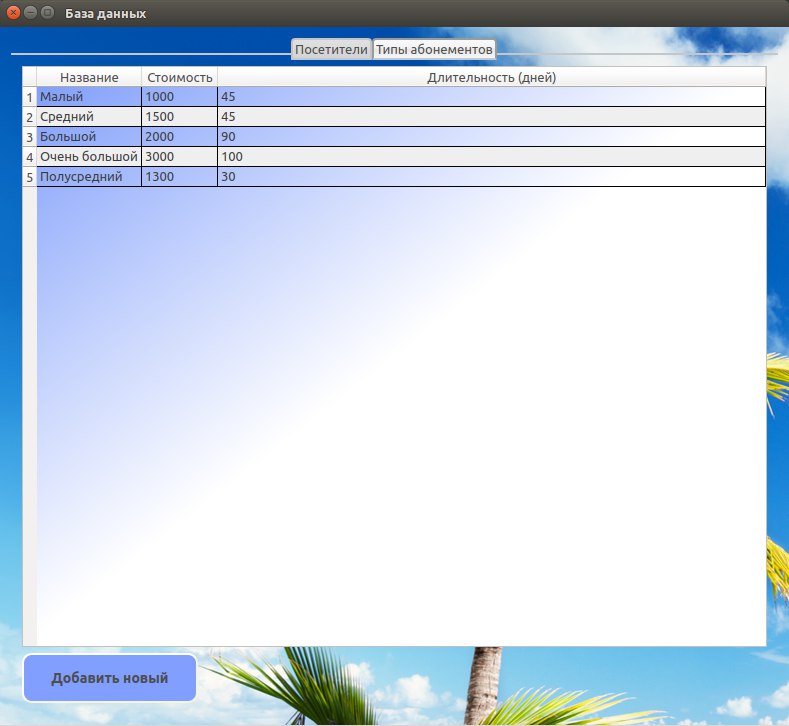
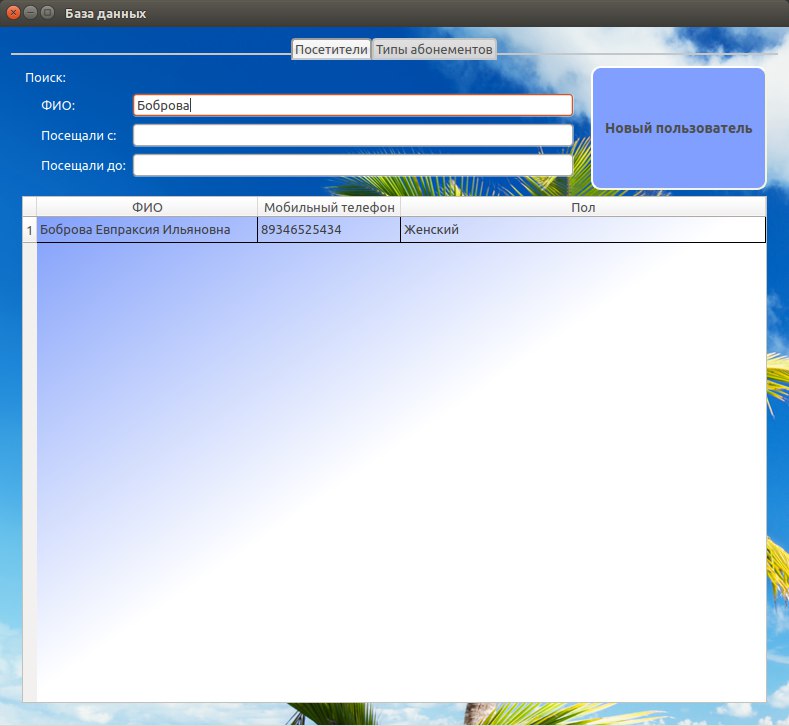
}

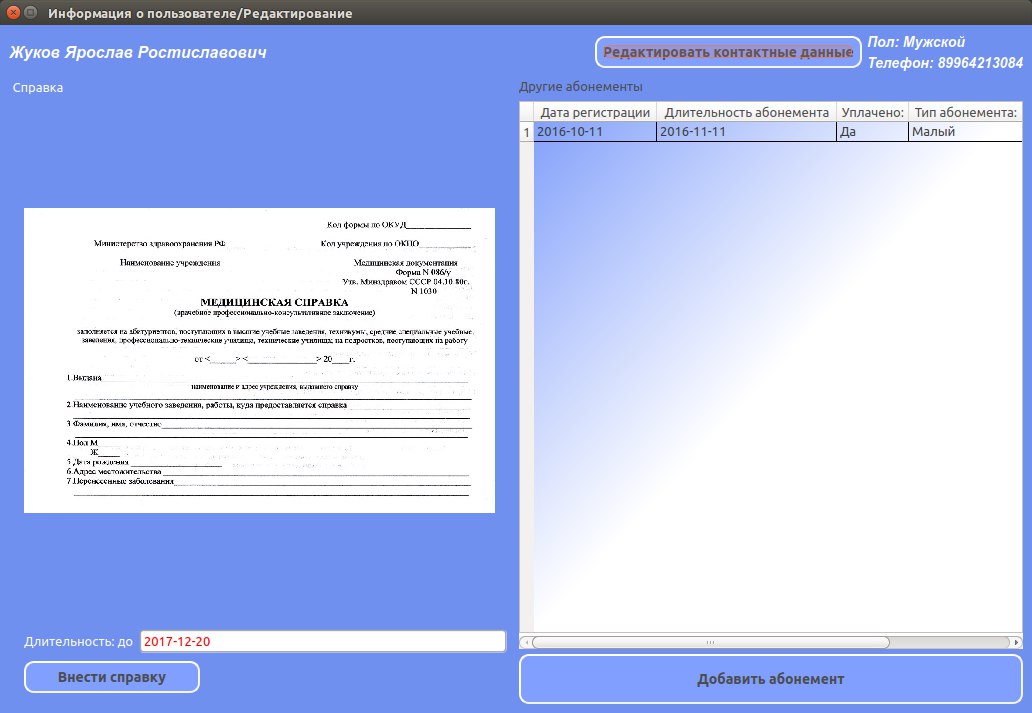
# Результаты работы программы

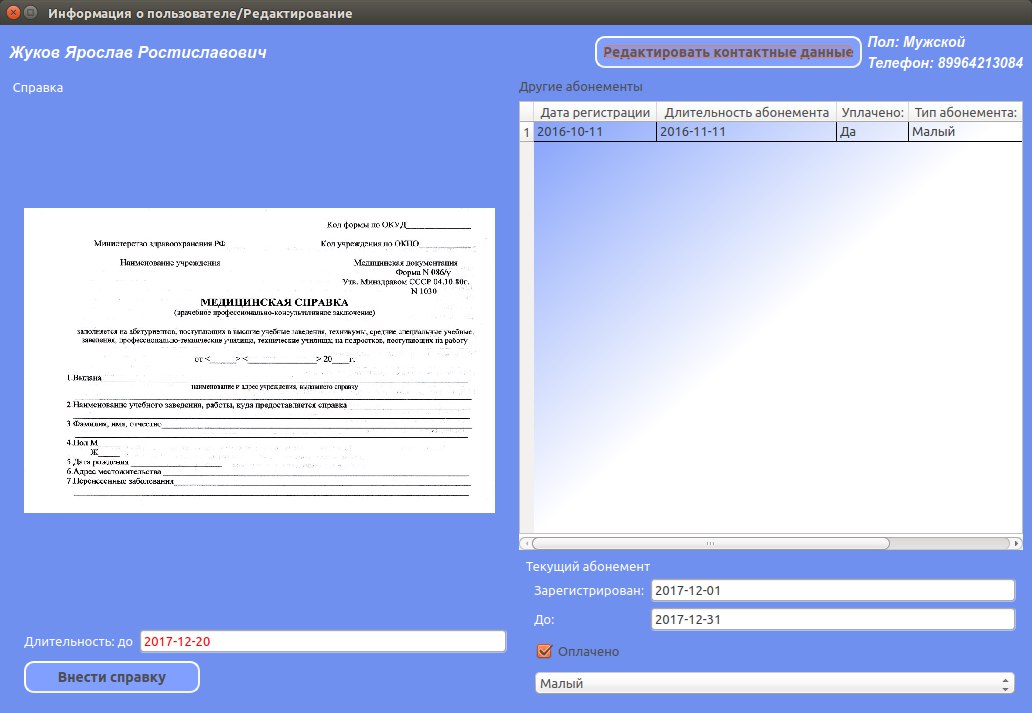












# Заключение

Успешно удалось реализовать программу для учета посетителей бассейна. Были реализованы:

* база данных посетителей и абонементов, где указана информация о них;
* функционал работы с базой данных и её изменение;
* раздел просмотра списка пользователей;
* раздел просмотра списка абонементов;
* раздел просмотра индивидуальной информации о посетителе;
* разделы редактирования информации в базе данных.

Так же было проведено изменение стандартного стиля виджетов с использованием каскадных таблиц стилей CSS.