Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

**ПО ТВОРЧЕСКОЙ РАБОТЕ №1**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Автоматизация рабочего места врача-сомнолога»

Выполнила работу

Студентка группы РИС-23-3Б

Шуракова А.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2024

**Введение**

Приложение разработано с использованием фреймворка Qt, который предоставляет широкий набор инструментов для создания кросс-платформенных приложений на C++. В основе приложения лежит объектно-ориентированный подход, использующий классы и сигналы/слоты для обработки событий и связи между компонентами.

**Постановка задачи**

Реализовать автоматизированное рабочее место специалиста (АРМ) – врача-сомнолога. Программа должна выполнять следующие функции:

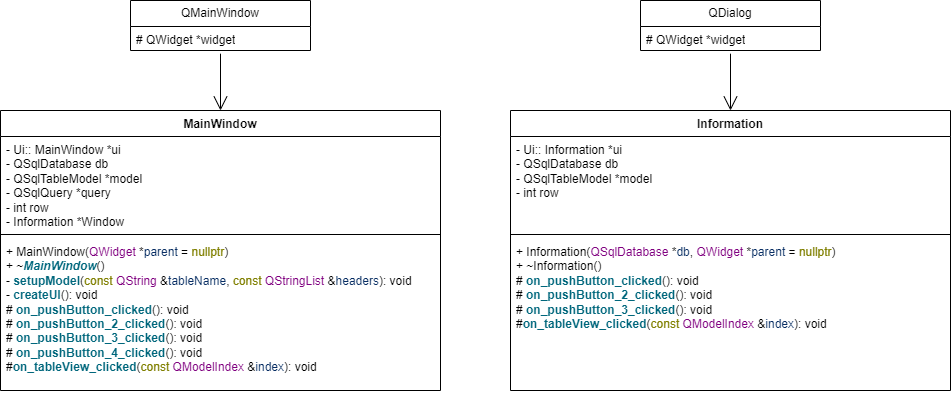
1. Запись пациентов на прием

* Добавление записи
* Удаление записи
* Обновление данных

1. Информация о пациентах

* Добавление информации
* Удаление информации
* Обновление данных

**UML-диаграмма**



**Код программы**

**arm.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtSql/QSqlDatabase> // для работы с базами данных

#include <QSqlQuery> // для выполнения SQL-запросов

#include <QSqlError> // для обработки ошибок SQL

#include <QDebug> // для вывода отладочной информации

#include <QSqlTableModel> // для представления таблицы базы данных

#include <QSqlQueryModel> // для представления результатов SQL-запроса

#include <QSortFilterProxyModel> // для сортировки и фильтрации данных

#include <QDate> //для работы с датами

#include "information.h"

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_pushButton\_2\_clicked**(); // Слот для обработки нажатия кнопки добавить запись

void **on\_pushButton\_3\_clicked**(); // Слот для обработки нажатия кнопки удалить запись

void **on\_pushButton\_4\_clicked**(); // Слот для обработки нажатия кнопки обновить

void **on\_tableView\_clicked**(const QModelIndex &index);// Слот для обработки нажатия на ячейку таблицы

void **on\_pushButton\_clicked**(); // Слот для обработки нажатия кнопки информация о пациентах

private:

Ui::MainWindow \*ui;

QSqlDatabase db; // Объект базы данных

QSqlTableModel \*model; // Указатель на модель таблицы базы данных

QSqlQuery \*query; // Указатель на объект SQL-запроса

int row; // номер активной строки

void **setupModel**(const QString &tableName, const QStringList &headers); // Функция для настройки модели

void **createUI**(); // Функция для создания пользовательского интерфейса

Information \*Window;

};

#endif // MAINWINDOW\_H

**arm.cpp**

#include "arm.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this); // Настройка пользовательского интерфейса MainWindow

db = QSqlDatabase::addDatabase("QSQLITE"); // Добавление драйвера базы данных SQLite

db.setDatabaseName("C:/Users/Nastya/Desktop/ARM/ARM.db"); // Установка имени файла базы данных

// Если не удалось открыть базу данных

if (!db.open())

{

// Показать сообщение об ошибке

ui->statusbar->showMessage("error: " + db.lastError().databaseText());

}

else

{

// Показать сообщение об успешном открытии базы данных

ui->statusbar->showMessage("ok");

}

model = new QSqlTableModel(this, db); // Создание нового объекта модели таблицы базы данных

model->*setTable*("patients"); //выбор таблицы из базы данных

model->lastError(); // Получение последней ошибки модели (если есть)

this->createUI(); // Вызов функции createUI для создания пользовательского интерфейса

ui->tableView->*setModel*(*model*); //добавляем информацию в таблицу

ui->tableView->setColumnHidden(0, true); // Скрытие первой колонки (ID)

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

model->insertRow(model->*rowCount*()); // Вставка новой строки в конец модели

}

void MainWindow::**on\_pushButton\_3\_clicked**()

{

model->removeRow(row); // Удаление строки с индексом row из модели

}

void MainWindow::**on\_pushButton\_4\_clicked**()

{

model->*sort*(2,Qt::AscendingOrder); // Сортировка модели по третьему столбцу (индекс 2) в возрастающем порядке

}

void MainWindow::**setupModel**(const QString &tableName, const QStringList &headers)

{

/\* Производим инициализацию модели представления данных

\* с установкой имени таблицы в базе данных, по которому

\* будет производится обращение в таблице \*/

model = new QSqlTableModel(this); // Создание нового объекта модели таблицы базы данных

model->*setTable*(tableName); // Установка имени таблицы для модели

// Устанавливаем названия колонок в таблице с сортировкой данных

// Устанавливаем названия колонок в таблице с сортировкой данных

for(int i = 0, j = 0; i < model->*columnCount*(); i++, j++){

// Установка заголовка для колонки i

model->*setHeaderData*(i,Qt::Horizontal,headers[j]);

}

// Устанавливаем сортировку по возрастанию данных по нулевой колонке

model->*setSort*(1,Qt::AscendingOrder);

}

void MainWindow::**createUI**()

{

ui->tableView->*setModel*(*model*); // Устанавливаем модель на созданную таблицу

ui->tableView->setColumnHidden(0, true); // Скрываем колонку с id записей

// Разрешаем выделение строк

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

// Устанавливаем размер колонок по содержимому

ui->tableView->resizeColumnsToContents();

// Растяжение последней колонки на всю оставшуюся ширину

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

model->*select*(); // Делаем выборку данных из таблицы

}

void MainWindow::**on\_tableView\_clicked**(const QModelIndex &index)

{

row = index.row(); // Получение индекса строки, на которую было нажатие

}

void MainWindow::**on\_pushButton\_clicked**()

{

// Создание нового окна Information с передачей указателя на базу данных

Window = new Information(*&db*);

// Показать окно Information

Window -> show();

}

**information.h**

#ifndef INFORMATION\_H

#define INFORMATION\_H

#include <QDialog> //для создания диалоговых окон

#include <QSqlDatabase> //для работы с базами данных

#include <QSqlQuery> //для выполнения SQL-запросов

#include <QSqlQueryModel> //для представления результатов SQL-запроса

#include <QSqlTableModel> //для представления таблицы базы данных

namespace **Ui** {

class **Information**;

}

class **Information** : public QDialog

{

Q\_OBJECT

public:

explicit **Information**(QSqlDatabase \*db, QWidget \*parent = nullptr);

~***Information***();

private slots:

void **on\_pushButton\_clicked**(); // Слот для обработки нажатия кнопки добавить пациента

void **on\_pushButton\_2\_clicked**(); // Слот для обработки нажатия кнопки удалить пациента

void **on\_tableView\_clicked**(const QModelIndex &index);// Слот для обработки нажатия на ячейку таблицы

void **on\_pushButton\_3\_clicked**(); // Слот для обработки нажатия кнопки обновить

private:

Ui::Information \*ui;

QSqlDatabase \*db; // Указатель на объект базы данных

QSqlTableModel \*model; // Указатель на модель таблицы базы данных

int row; // Номер активной строки

};

#endif // INFORMATION\_H

**information.cpp**

#include "information.h"

#include "ui\_information.h"

Information::**Information**(QSqlDatabase \*db, QWidget \*parent) :

QDialog(*parent*),

ui(new Ui::Information)

{

ui->setupUi(this); // Настройка пользовательского интерфейса Information

this ->db = db; // Присваивание указателя на базу данных

model = new QSqlTableModel(this, \*db); // Создание нового объекта модели таблицы базы данных

model->*setTable*("information"); // Установка таблицы "information" для модели

model->*select*(); // Выборка данных из таблицы

// установка "шаблона" в виджет

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

ui->tableView->resizeColumnsToContents(); // Автоматическая подгонка размера колонок

ui->tableView->horizontalHeader()->setStretchLastSection(true); // Растяжение последней колонки на всю оставшуюся ширину

ui->tableView->setColumnHidden(0, true); // Скрытие первой колонки (ID)

}

Information::~***Information***()

{

delete ui;

}

void Information::**on\_pushButton\_clicked**()

{

model->insertRow(model->*rowCount*()); // Вставка новой строки в конец модели

}

void Information::**on\_pushButton\_2\_clicked**()

{

model->removeRow(row); // Удаление строки с индексом row из модели

}

void Information::**on\_tableView\_clicked**(const QModelIndex &index)

{

row = index.row(); // Получение индекса строки, на которую было нажатие

}

void Information::**on\_pushButton\_3\_clicked**()

{

model->*sort*(2,Qt::AscendingOrder); // Сортировка модели по третьему столбцу (индекс 2) в возрастающем порядке

}

**main.cpp**

#include "arm.h"

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

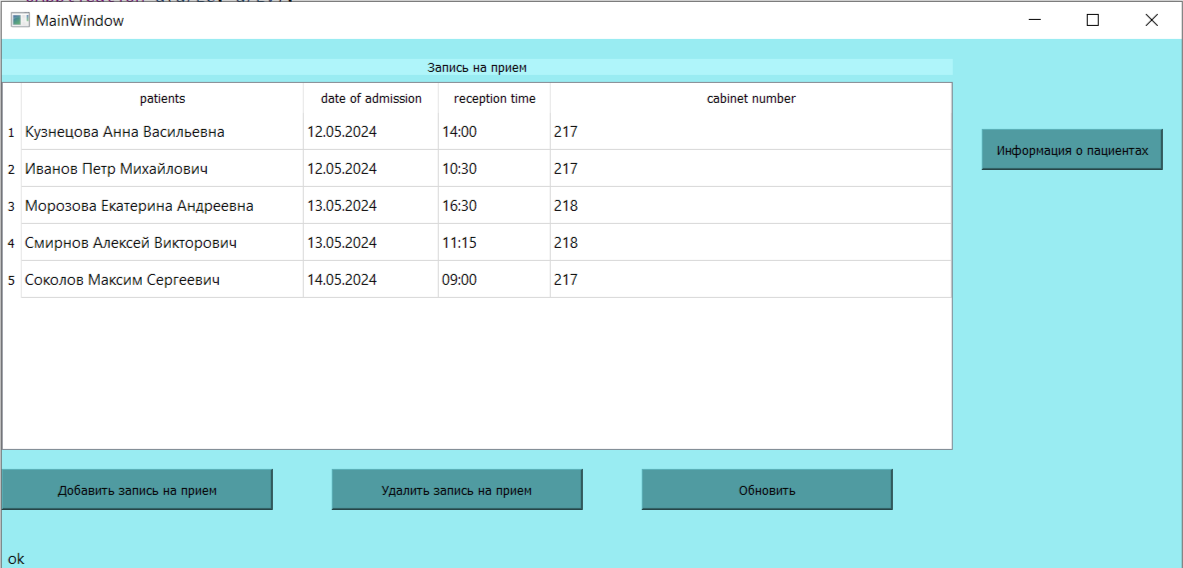
MainWindow w;

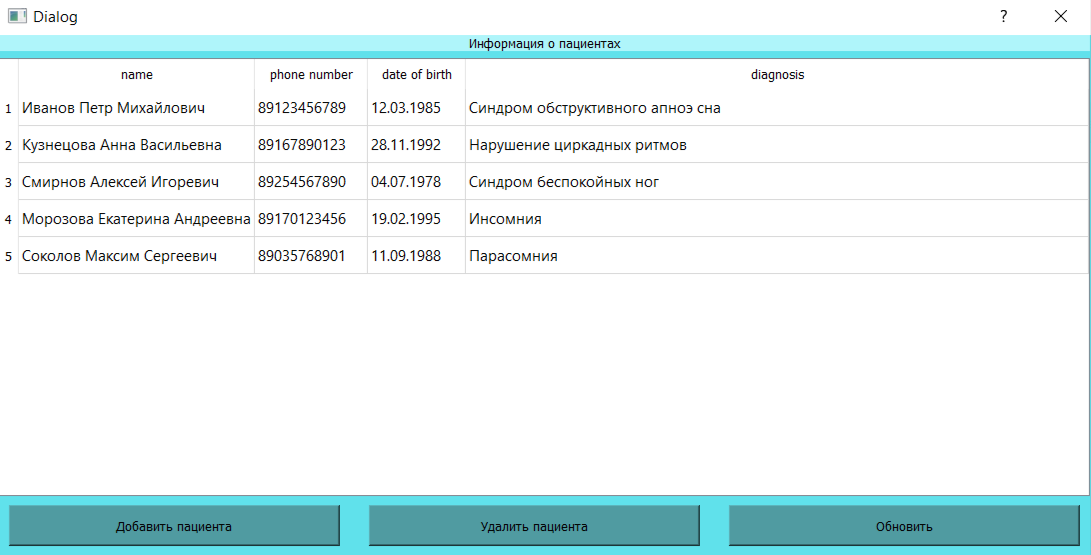
w.show();

return a.exec();

}

**Результат работы программы**

****

****

**Ссылка на Git**

https://github.com/Ananasic07/ARM-TSP/tree/main/ARM

**Ссылка на YouTybe**

https://youtu.be/v\_AsSs31x5M?si=d7vvq1emzgaoPZk2