Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное средство «Поликлиника. Учёт пациентов с помощью базы данных»**

|  |  |  |
| --- | --- | --- |
| Студент |  | В.В Кашевский |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc28015794)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc28015795)

[1.1 Что такое база данных 8](#_Toc28015796)

[1.2 Классификация баз данных по модели данных 9](#_Toc28015797)

[1.3 Нереляционные модели баз данных 9](#_Toc28015798)

[1.3.1 Иерархическая модель 9](#_Toc28015799)

[1.3.2 Сетевая модель 11](#_Toc28015800)

[1.4 Реляционная модель баз данных 13](#_Toc28015801)

[1.4.1 Понятие и история реляционных БД 13](#_Toc28015802)

[1.4.2 Структура реляционных БД 14](#_Toc28015803)

[1.4.3 Пример реляционной системы баз данных 16](#_Toc28015804)

[1.4.4 Реляционные СУБД и язык SQL 16](#_Toc28015805)

[1.4.5 Понятие сущности, виды связей между сущностями 17](#_Toc28015806)

[1.5 Постановка задачи 18](#_Toc28015807)

[2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 21](#_Toc28015808)

[2.1 Структура программы 21](#_Toc28015809)

[2.2 Проектирование таблиц в базе данных 23](#_Toc28015810)

[2.2.1 Таблица пользователей(users) 23](#_Toc28015811)

[2.2.2 Таблица пациентов(patients) 23](#_Toc28015812)

[2.2.3 Таблица докторов(doctors) 23](#_Toc28015813)

[2.2.4 Таблица отделений(departments) 24](#_Toc28015814)

[2.2.5 Таблица услуг(services) 24](#_Toc28015815)

[2.2.6 Таблица талонов(talons) 24](#_Toc28015816)

[3. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 26](#_Toc28015817)

[3.1 Тест 1 26](#_Toc28015818)

[3.2 Тест 2 26](#_Toc28015819)

[3.3 Тест 3 27](#_Toc28015820)

[3.4 Тест 4 27](#_Toc28015821)

[3.5 Тест 5 28](#_Toc28015822)

[3.6 Тест 6 29](#_Toc28015823)

[3.7 Тест 7 30](#_Toc28015824)

[4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 31](#_Toc28015825)

[4.1 Интерфейс программы и порядок работы 31](#_Toc28015826)

[ЗАКЛЮЧЕНИЕ 36](#_Toc28015827)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc28015828)

[ПРИЛОЖЕНИЕ A 38](#_Toc28015829)

[Исходный код программы 38](#_Toc28015830)

# **ВВЕДЕНИЕ**

Человек в своей жизни использует большие объемы информации. Множество фактов, необходимых человеку достаточно велико. Но человек использует не весь объём своей памяти, поэтому часто нужная информация может забываться или теряться. Чтобы избежать этого, люди прибегают к записям. Однако недостаточно только лишь записать информацию – необходимо уметь быстро ее найти в нужное время, т.е. тогда, когда это необходимо. Для хранения и облегчения поиска нужной информации человеком придуманы специальные устройства - базы данных.

В широком смысле понятие истории баз данных сводится к истории любых средств, с помощью которых человечество хранило и обрабатывало данные. В таком контексте упоминаются, например, средства учёта царской казны и налогов в древнем Шумере, узелковая письменность инков —кипу, клинописи, содержащие документы Ассирийского царства и т. п. Следует помнить, что недостатком этого подхода является размывание понятия «база данных» и фактическое его слияние с понятиями «архив» и даже «письменность».

История понятия базы данных в современном понимании начинается с 1955 года, когда появилось программируемое оборудование обработки записей. Оперативные сетевые базы данных появились в середине 1960-х. Операции над оперативными базами данных обрабатывались в интерактивном режиме с помощью терминалов. Простые индексно-последовательные организации записей быстро развились к более мощной модели записей, ориентированной на наборы.

В начале 1970-х, благодаря работам  Эдгара Кодда, появляется реляционная модель данных. Работы Кодда открыли путь к тесной связи прикладной технологии баз данных с математикой и логикой.

Сам термин «база данных» появился в начале 1960-х годов, и был введён в употребление на симпозиумах, организованных компанией SDC в 1964 и 1965 годах, хотя понимался сначала в довольно узком смысле, в контексте систем искусственного интеллекта. В широкое употребление в современном понимании термин вошёл лишь в 1970-е годы.

В современном понимании, база данных – это один или несколько файлов данных, предназначенных для хранения, изменения и обработки больших объемов взаимосвязанной информации.

Современный мир информационных технологий трудно представить себе без использования баз данных. Практически все системы в той или иной степени связаны с функциями долговременного хранения и обработки информации. Фактически информация становится фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. Любые административные решения требуют четкой и точной оценки текущей ситуации и возможных перспектив ее изменения. И если раньше в оценке ситуации участвовало несколько десятков факторов, которые могли быть вычислены вручную, то теперь таких факторов сотни и сотни тысяч, и ситуация меняется не в течение года, а через несколько минут, а обоснованность принимаемых решений требуется большая, потому что и реакция на неправильные решения более серьезная, более быстрая и более мощная, чем раньше. И, конечно, обойтись без информационной модели производства, хранимой в базе данных, в этом случае невозможно.

# **1.** **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1 Что такое база данных**

Развитие вычислительной техники и появление емких внешних запоминающих устройств прямого доступа предопределило интенсивное развитие автоматических и автоматизированных систем разного назначения и масштаба, в первую очередь заметное в области бизнес-приложений. Такие системы работают с большими объемами информации, которая обычно имеет достаточно сложную структуру, требует оперативности в обработке, часто обновляется и в то же время требует длительного хранения. Примерами таких систем являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи билетов и т.д.

Другими направлениями, стимулировавшими развитие, стали, с одной стороны, системы управления физическими экспериментами, обеспечивающими сверхоперативную обработку в реальном масштабе времени огромных потоков данных от датчиков, а с другой – автоматизированные библиотечные информационно-поисковые системы.

Это привело к появлению новой информационно технологии интегрированного хранения и обработки данных – концепции баз данных, в основе которой лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе.

**Банк данных** (БнД) — это система специальным образом организованных данных - баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

**База данных**(БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Базы данных могут быть централизованными(БД хранятся на одном компьютере) или же распределенными(БД хранятся на нескольких компьютерах).

**Система управления базами данных**(СУБД) — это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.

Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Особенностью БД является особый режим использования данных, когда только часть данных находится под оперативным управлением СУБД. Все остальные данные обычно располагаются на носителях, оперативно не управляемых СУБД.

Эффективное управление внешней памятью является основной функцией СУБД. Эти обычно специализированные средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи.

Необходимые условия для наиболее эффективной работы банков данных:

* Многократное использование данных
* Простота
* Легкость и гибкость использования
* Адаптивность и расширяемость
* Контроль за целостностью данных
* Восстановление данных после сбоев[1].

## **1.2 Классификация баз данных по модели данных**

Хранимые в базе данные имеют определенную логическую структуру — иными словами, описываются некоторой **моделью представленияданных**(моделью данных), поддерживаемой СУБД. К числу классических относятся следующие модели данных:

* Иерархическая
* Сетевая
* Реляционная

Кроме того, в последние годы активно внедряются на практике объектно-ориентрованная, постреляционная и многомерная модели. Разрабатываются также всевозможные системы, основанные на других моделях данных, расширяющих известные модели. В их числе можно назвать объектно-реляционные, дедуктивно-объектно-ориентированные, семантические, концептуальные и ориентированные модели. Некоторые из этих моделей служат для интеграции баз данных, баз знаний и языков программирования[3].

## **1.3 Нереляционные модели баз данных**

### **1.3.1 Иерархическая модель**

Иерархическая модель данных является наиболее простой среди всех даталогических моделей. Исторически она появилась первой среди всех диалогических моделей: именно эту модель поддерживает первая из зарегистрированных промышленных СУБД IMS фирмы IBM.

Появление иерархической модели связано с тем, что в реальном мире очень многие связи соответствуют иерархии, когда один объект выступает как родительский, а с ним может быть связано множество подчиненных объектов.

Это и является особенностью иерархических систем баз данных - данные в ней представлены в виде одной или нескольких древовидных структур.

Для описания структуры (схемы) иерархической БД на некотором языке программирования используется тип данных «дерево».

Тип «дерево» является составным. Он включает в себя подтипы («поддеревья»), каждый из которых, в свою очередь, является типом «дерево». Каждый из типов «дерево» состоит из одного «корневого» типа и упорядоченного набора (возможно, пустого) подчиненных типов. Каждый из элементарных типов, включенных в тип «дерево», является простым или составным типом «запись». Простая «запись» состоит из одного типа, например числового, а составная «запись» объединяет некоторую совокупность типов, например, целое, строку символов и указатель (ссылку).

На рис.1.1 представлена схема иерархической системы баз данных. Она предоставляет средства для нахождения в дереве конкретного элемента и последующего обхода этого дерева в поисках нужных авторов, статей и сборников. У каждого узла дерева может быть ни одного или один родитель и ни одного, один или много дочерних узлов. Такую конфигурацию называют иерархией с одним родителем.

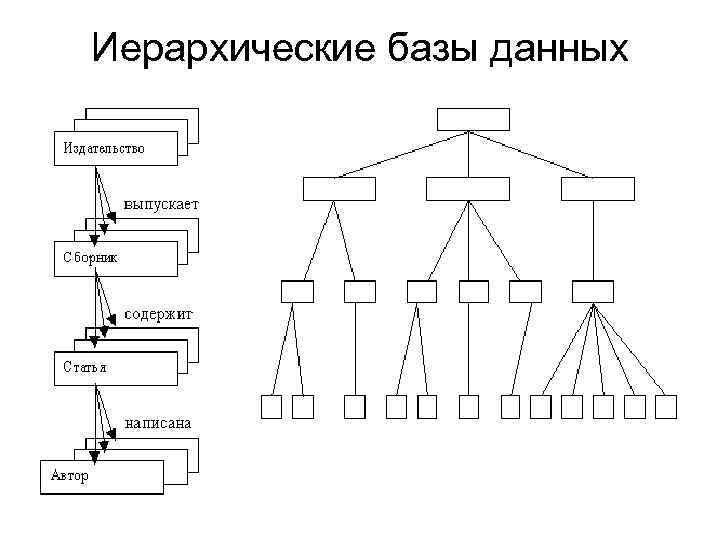
****

Рис. 1.1

К *достоинствам* иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения основных операций над данными. Иерархическая модель данных удобна для работы с иерархически упорядоченной информацией.

*Недостатком*иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя.

На иерархической модели данных основано сравнительно ограниченное количество СУБД, в числе которых можно назвать зарубежные системы IMS, PC/Focus, Team-Upи Data Edge, а также отечественные системы Ока, ИНЭС и МИРИС[3].

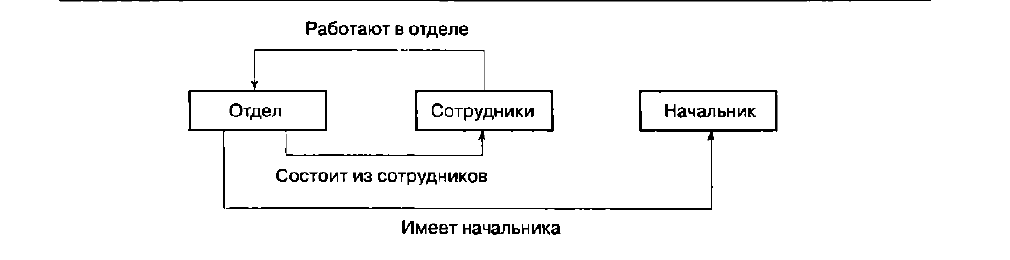
### **1.3.2 Сетевая модель**

Сетевая модель данных позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных. Наиболее полно концепция сетевых БД впервые была изложена в Предложениях группы КОДАСИЛ (KODASYL).

Для описания схемы сетевой БД используется две группы типов: «запись» и «связь». Тип «связь» определяется для двух типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами связей.

Сетевая БД состоит из набора записей и набора соответствующих связей. На формирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла иметь только одну запись-предка, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков(сводных родителей).

На рис.1.2 представлена схема сетевой системы баз данных.

Рис 1.2

К числу важнейших операций манипулирования данными баз сетевого типа можно отнести следующие:

1. Поиск записи в БД;
2. Переход от предка к первому потомку;
3. Поиск от потомка к предку;
4. Создание новой записи;
5. Удаление текущей записи;
6. Обновление текущей записи;
7. Включение записи в связь;
8. Исключение записи из связи;
9. Изменение связей и т.д.

У сетевой модели баз данных также существуют свои достоинства и недостатки.

*Достоинством* сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей.

*Недостатком*сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения обработки информации в БД обычным пользователем. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

## **1.4 Реляционная модель баз данных**

### **1.4.1 Понятие и история реляционных БД**

В настоящее время иерархические и сетевые базы данных довольно часто используются в разработке. Кроме того, иерархические системы БД возродились в службах каталогов, таких как Active Directory компании Microsoft и Directory Server компании Netscape, а также с появлением XML (Extensible Markup Language, расширяемый язык разметки). Однако, начиная с 1970-х годов все большую популярность приобретает новый способ представления данных, более строгий, но при этом более понятный и удобный.

Появление теоретико-множественных моделей в системах баз данных было предопределено настоятельной потребностью пользователей в переходе от работы с элементами данных, как это делается в графовых моделях, к работе с некоторыми макрообъектами. Основной моделью в этом классе является реляционная модель данных. Простота и наглядность модели для пользователей-непрограммистов, с одной стороны, и серьезное теоретическое обоснование, с другой стороны, определили большую популярность этой модели. Кроме того, развитие формального аппарата представления и манипулирования данными в рамках реляционной модели сделали се наиболее перспективной для использования в системах представления знаний, что обеспечивает качественно иной подход к обработке данных в больших информационных системах.

Теоретической основой реляционной модели стала теория отношений, основу которой заложили два логика — американец Чарльз Содерс Пире и немец Эрнст Шредер. В руководствах по теории отношении было показано, что множество отношений замкнуто относительно некоторых специальных операций, то есть образует вместе с этими операциями абстрактную алгебру. Это важнейшее свойство отношений было использовано в реляционной модели для разработки языка манипулирования данными, связанного с исходной алгеброй.

В 1970 году сотрудник исследовательской лаборатории IBM доктор Е. Ф.Кодд опубликовал статью под названием «A Relational Model of Data for Large Shared Data Banks» (Реляционная модель данных для больших банков данных коллективного пользования), в которой предложил представлять данные как наборы таблиц. Вместо указателей для навигации по взаимосвязанным сущностям используются избыточные данные, связывающие записи разных таблиц.

В своей статье Е.Ф.Кодд ограничил набор операций, оставив лишь семь основных и одну дополнительную. Предложения Кодда были настолько эффективны для систем баз данных, что за эту модель он получил премию Тьюринга в области теоретических основ вычислительной техники[1].

### **1.4.2 Структура реляционных БД**

Одной из главных характеристик реляционной БД является её однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат. Каждая строка представляет некоторый объект реального мира или отношение между объектами.

Основными понятиями, с помощью которых определяется реляционная модель, являются домен, отношение, кортеж, запись, кардинальность, атрибуты, степень, первичный ключ.

*Атрибуты* – это непосредственно столбцы таблицы, которые содержат различные характеристики объектов.

*Домен –* это совокупность значений, из которых берутся значения соответствующих атрибутов определенного отношения. С точки зрения программирования, домен – это тип данных, определяемый системой или пользователем.

*Кардинальность* – это количество строк в таблице.

*Запись* – это совокупность различной информации об одном конкретном объекте.

*Степень отношения* – это количество полей таблицы.

*Кортеж* – это непосредственно вся таблица.

Важным понятием в реляционной модели баз данных является *первичный ключ*. Первичный ключ – это столбец или некоторое подмножество столбцов, которое уникально, т.е. единственным образом определяют строки. Первичный ключ, который включает в себя более одного столбца, называется множественным, или комбинированным, или составным. Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т.е. иметь значение NULL.

Остальные ключи, которые можно также использовать в качестве первичных, называются потенциальными или альтернативными ключами.

Вместе с понятием *первичного ключа* в структуре реляционных БД неразрывно идет понятие *внешнего ключа*.

*Внешний ключ* – это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. Внешний ключ таблицы является ссылкой на первичный ключ другой таблицы. Внешний ключ может быть либо пустым, либо соответствовать значению первичного ключа, на который он ссылается. Таким образом, внешние ключи реализуют связи между таблицами базы данных[2].

Создатель теории реляционных БД Е.Ф. Кодд сформулировал 12 правил, которые определяют концепцию реляционной модели:

1. Вся информация в базе данных должна быть представлена исключительно на логическом уровне и только в виде значений, содержащихся в таблицах.
2. Доступ ко всем элементам данных должен обеспечиваться путем использования комбинации имени таблицы, первичного ключа и имени столбца.
3. В реляционной БД должна быть реализована поддержка недействительных значений.
4. Описание БД на логическом уровне должно быть представлено в том же виде, что и основные данные.
5. Должен существовать по крайней мере один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает определение данных, обработку данных, условия целостности и др.
6. Все представления, которые теоретически можно обновить, должны быть доступны для обновления.
7. Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.
8. Прикладные программы и утилиты для работы с данными должны оставаться нетронутыми при любых изменениях способов хранения данных.
9. Логические данные должны быть независимы.
10. Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге.
11. Реляционная СУБД не должна зависеть от потребностей конкретного клиента.
12. Если в реляционной системе есть низкоуровневый язык, то должна существовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня.

Как и у любых других моделей представления данных, у реляционной модели есть свои достоинства и недостатки.

*Достоинство*реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной их широкого использования. Проблемы же эффективности обработки данных этого типа оказались технически вполне разрешимыми.

Основными *недостатками*реляционной модели являются следующие: отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

### **1.4.3 Пример реляционной системы баз данных**

База данных будет выглядеть следующим образом(рис. 1.3).



Рис.1.3

Некоторые из таблиц содержат информацию, используемую для навигации к другой таблице. Например, в таблице Клиенты есть столбец ID, содержащий уникальный идентификатор заказа в таблице Заказы. Этот столбец и является примером *внешнего ключа*. В отличие от жесткой структуры иерархической/сетевой моделей, реляционные таблицы можно использовать по-разному (даже так, как разработчики этой базы данных и не представляли себе).

### **1.4.4 Реляционные СУБД и язык SQL**

Для работы с данными, находящимися в реляционной базе данных, используются реляционные *системы управления базами данных*(СУБД).

Реляционные СУБД используют структуры для хранения и работы с данными. Главными функциями реляционных СУБД являются определение данных, обработка данных, управление данными.

Реляционные СУБД позволяют добавлять новые записи в таблицы, удалять записи из таблицы, обновлять данные, которые располагаются в таблицах БД, выполнять поиск одной или нескольких записей с возможностью добавления определенных условий.

Существует множество РСУБД и у каждой есть свои преимущества и свои недостатки. Наиболее популярные РСУБД:

* SQLite
* MySQL
* PostgreSQL

Для создания, управления и модификации данных в реляционной БД, управляемой соответствующей СУБД используется специальный декларированный язык программирования – SQL.

SQL (Structured Query Language) — Структурированный Язык Запросов - стандартный язык запросов по работе с реляционными БД.

Язык SQL появился после реляционной алгебры, и его прототип был разработан в конце 70-х годов в компании IBM Research. Он был реализован в первом прототипе реляционной СУБД фирмы IBM System R. В дальнейшем этот язык применялся во многих коммерческих СУБД и в силу своего широкого распространения постепенно стал стандартом «де-факто» для языков манипулирования данными в реляционных СУБД.

SQL идет рука об руку с реляционной моделью, потому что результатом SQL запроса является таблица (в данном контексте также называемая *результирующим набором*). Таким образом, в реляционной базе данных можно создать новую постоянную таблицу, просто сохранив результирующий набор запроса. Аналогично в качестве входных данных запрос может использовать как постоянные таблицы, так и результирующие наборы других запросов.

### **1.4.5 Понятие сущности, виды связей между сущностями**

Сущность, в общем случае – это то, что представляет интерес для пользователей базы данных, например клиенты, запчасти, географическое положение и др. Каждая сущность имеет имя, уникальное в пределах моделируемой системы. Так как сущность соответствует некоторому классу однотипных объектов, то предполагается, что в системе существует множество экземпляров данной сущности. Набор атрибутов, однозначно идентифицирующий конкретный экземпляр сущности, называют ключевым.

Между сущностями могут быть установлены связи - бинарные ассоциации, показывающие, каким образом сущности соотносятся или взаимодействуют между собой. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Если связь устанавливается между двумя сущностями, то она определяет взаимосвязь между экземплярами одной и другой сущности.

Связи между сущностями делятся на три типа:

* Один-к-одному(1:1)
* Один-ко-многим(1:М)
* Многие-ко-многим(M:M)

Связь вида 1:1 образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Поскольку значения в ключевых полях обеих таблиц не повторяются, обеспечивается взаимно-однозначное соответствие записей из этих таблиц. Сами таблицы, по сути, здесь становятся равноправными.

На практике связи вида 1:1 используются сравнительно редко, так как хранимую в двух таблицах информацию легко объединить в одну таблицу, которая занимает гораздо меньше места в памяти ЭВМ. Возможны случаи, когда удобнее иметь не одну, а две и более таблицы. Причинами этого может быть необходимость ускорить обработку, повысить удобство работы нескольких пользователей с общей информацией, обеспечить более высокую степень защиты информации и т. д.

Связь 1:М имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы. Это наиболее используемый тип связи между сущностями. Подобная связь создается в том случае, когда только на один из связываемых столбцов наложено ограничение уникальности или он является первичным ключом.

Самый общий вид связи М:М возникает в случаях, когда нескольким записям основной таблицы соответствует несколько записей дополнительной таблицы.

Необходимо также учитывать и то, что на практике в связь обычно вовлекается сразу несколько таблиц. При этом одна из таблиц может иметь различного рода связи с несколькими таблицами. В случаях, когда связанные таблицы, в свою очередь, имеют связи с другими таблицами, образуется иерархия или дерево связей.

## **1.5 Постановка задачи**

Цель: применение теоретических знаний и умений для разработки desktop-приложения, имитирующего регистратуру поликлиники, с использованием баз данных.

Примером послужил известный белорусский сайт – интернет регистратура – Талон.бай. Он предоставляет возможности учёта пациентов в базе, оформления талона в любое из 187 включённых в базу учреждение здравоохранения, отображения выбранных услуг в личном кабинете и т. д.

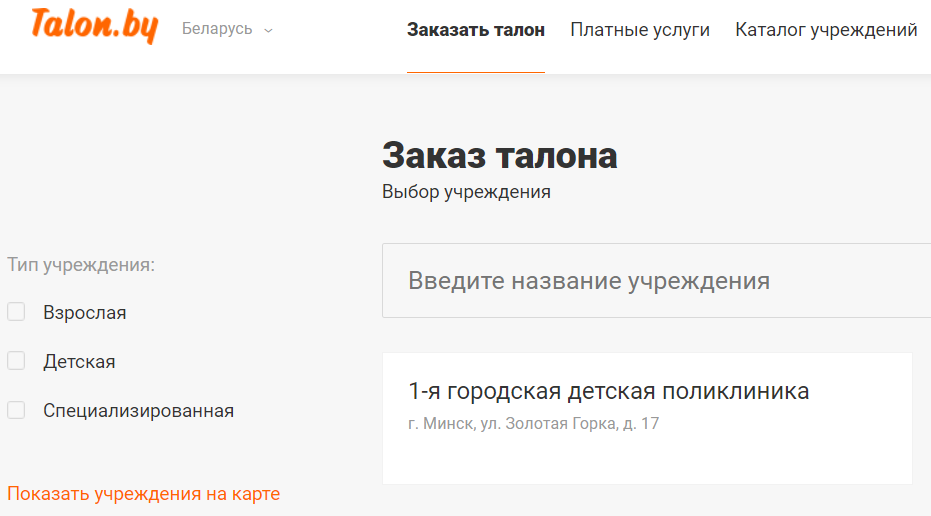


Рис 1.4

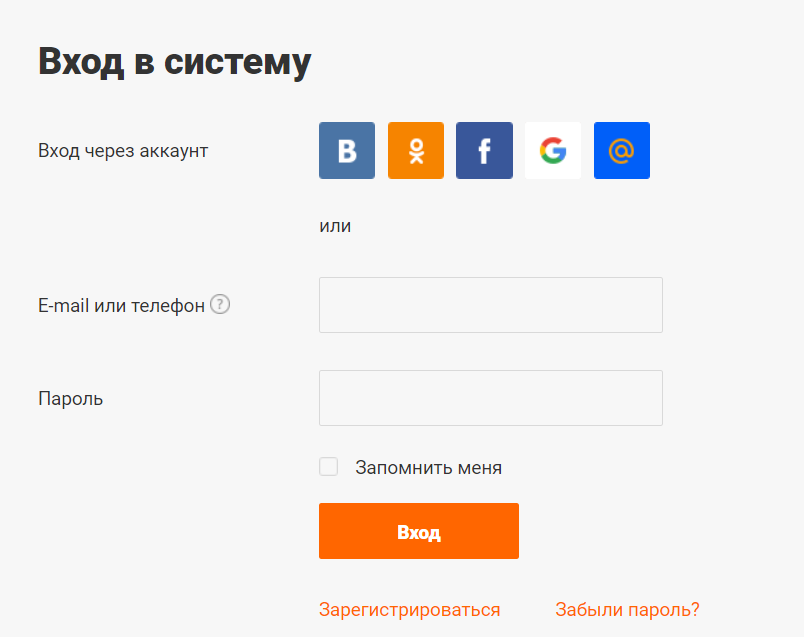


Рис. 1.5

Проект реализует следующие цели:

* Создание удобного, интуитивно понятного рабочего интерфейса приложения.
* Реализация возможности регистрации личного кабинета в регистратуре и последующего входа в свой личный кабинет.
* Присутствие возможности записи на желаемую услугу к определенному специалисту на удобное для пациента время.
* Отображение существующих записей на услуги и управление ими.
* Администрирование всех записей пациентов со стороны регистратуры поликлиники.
* Отображение полной информации о поликлинике и о работающих в ней специалистах.
* Внесенная контактная информация должна храниться в базе данных.

# **2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **2.1 Структура программы**

Данное приложение состоит из девяти заголовочных файлов с исходниками(соответственно есть 9 классов) и восьми оконных форм:

– **mainwindow** – главное окно, в котором пользователь может

производить разные манипуляции такие, как оформление талона, просмотр личных записей, удаление записи, просмотр информации о поликлинике, редактирование личной информации, изменение пароля и фото;

– **clinicinfodiaolog** – окно, в котором пользователю отображается информация о поликлинике и её специалистах;

– **datedialog** – окно, в котором пользователь выбирает дату талона;

– **editdialog** окно, в котором пользователь может изменить личную информацию;

– **removedialog** – окно, в котором пользователь может удалить талон;

– **logindialog** – окно входа, где пользователь вводит необходимые данные;

– **registrationdialog** – окно регистрации, в котором пользователь может встать на учёт;

– **passwordeditdialog** - окно, в котором пользователь может изменить свой пароль;

В приложении использовалась архитектура «клиент-сервер».

**Архитектура «клиент-сервер»** определяет общие принципы организации взаимодействия в сети, где имеются *серверы*, узлы-поставщики некоторых специфичных функций (сервисов) и *клиенты*, потребители этих функций.

Таким образом, **сервер**— это программа, представляющая какие-либо услуги другим программам и обслуживающая запросы клиентов на получение ресурсов определенного вида, а **клиент** – это программа, использующая услугу, предоставляемую программой сервера.

В случае созданного приложения, в качестве сервера выступала созданная локальная сеть MySQL Server, на которой располагалась база данных приложения.

В качестве же клиента выступало непосредственно разработанное приложение «Поликлиника». Клиент отвечает не только за формирование соединения с сервером, но и за формирование пользовательского интерфейса, логическую обработку данных и за манипулирование исходными данными.

При разработке архитектуры приложения учитывались основные правила и концепции архитектуры «клиент-сервер».

Так, например, взаимодействие между клиентом и сервером всегда начинается со стороны клиента. Клиент формирует и посылает запрос на сервер. Сервер, в свою очередь, производит необходимые манипуляции с данными, формирует результат и передает его клиенту. Клиент получает результат, отображает его на устройстве вывода и ждет дальнейших действий пользователя. Цикл повторяется, пока пользователь не закончит работу с сервером.

В ходе разработки программного продукта были использована среда Qt **–** кросплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Документация по использованию среды описана в [4, 6, 7].

Главным преимуществом фреймворка Qt является его кросс-платформенность. Программа, написанная на операционной системе Linux, может также работать и на операционной системе Windows, и на операционной системе MacOS. Приятным бонусом является возможность использования программ, написанных на Qt, на мобильных устройствах(Android, IOS и др.).

Qt предоставляет программисту не только удобный набор библиотек классов, но и определённую модель разработки приложений, определённый каркас их структуры.

Важным преимуществом Qt является хорошо продуманный, логичный и стройный набор классов, предоставляющий программисту очень высокий уровень абстракции.

Достоинства Qt Creator:

* Наличие сигнально-слотовых соединений
* Наличие удобного дизайнера для создания форм окон
* Удобный редактор текста

Несомненно, у Qt много и недостатков. Один из них, например, - большой объем приложения.

В настоящее время существует множество реляционных СУБД. Самыми популярными являются SQLLite, MySQL, PostgerSQL.

Я использовал **MySQL, так как** это самая распространенная полноценная серверная СУБД. Успешно работает с различными сайтами и веб приложениями.

Достоинствами MySQL являются:

* Простота в работе;
* Богатый функционал, даже несмотря на то, что некоторый функционал SQL здесь не был реализован;
* Безопасность;
* Масштабируемость;
* Скорость;

Также при разработке и администрировании баз данных, я использовал программу MySQL Workbench. MySQL Workbench – это инструмент для визуального проектирования баз данных MySQL. Благодаря MySQL Workbench упрощается процесс создания и редактирования баз данных. Документация по использованию описана в [5].

## **2.2 Проектирование таблиц в базе данных**

### **2.2.1 Таблица пользователей(users)**

Для физического хранения таблицы пользователей создадим модель users. Предусмотрим следующие поля:

* id\_users – INT(11) – уникальный идентификатор;
* password – VARCHAR(45) – пароль пользователя;
* email – VARCHAR(45) – адрес электронной почты пользователя;

С помощью данной модели происходит авторизация пользователей.

### **2.2.2 Таблица пациентов(patients)**

Пациент – это, в рамках программы, авторизированный пользователь, у которого есть некоторая контактная информация, определенное количество заказанных талонов на услуги, а также персональное фото. Для хранения более подробной информации о каждом пациенте создадим модель patients, в которой будут определены следующие поля:

* id\_number – INT(11) – уникальный идентификатор пациента;
* name – VARCHAR(45) – имя пациента;
* surname – VARCHAR(45) – фамилия пациента;
* phonenumber – VARCHAR(13) – мобильный телефон пользователя;
* email – VARCHAR(45) – адрес электронной почты пациента;
* patientsphoto – TEXT – путь к фотографии пациента.

### **2.2.3 Таблица докторов(doctors)**

Для хранения информации о специалистах, трудящихся в поликлинике, создадим таблицу, которая будет отображать непосредственно информацию о докторах. Предусмотрим следующие поля таблицы, представляющей данной таблицы:

* license\_number – VARCHAR(20) – номер лицензии врача;
* photo – TEXT – путь к фотографии специалиста;
* info – TEXT – краткая биография врача;
* name– VARCHAR(45) – имя и фамилия врача;
* qualification – VARCHAR(45) – специальность врача;
* department – VARCHAR(2) – отделение, в котором работает врач;
* service – TEXT – услуги, которые оказывает данный специалист;
* cabinet – VARCHAR(45) – номер кабинета;

### **2.2.4 Таблица отделений(departments)**

Создадим таблицу для отделений поликлиники. Разработаем следующие поля:

* id – VARCHAR(2) – идентификационное имя отделения;
* name – VARCHAR(45) – название отделения;

### **2.2.5 Таблица услуг(services)**

Таблица услуг будет отображать всевозможные предоставляемые поликлиникой услуги, а также название отделения, где эти услуги оказываются. Создадим поля:

* id\_number – VARCHAR(40) – идентификационный номер услуги;
* name – VARCHAR(45) – наименование услуги;
* department – VARCHAR(45) – название отделения, где оказывается данная услуга;

### **2.2.6 Таблица талонов(talons)**

Каждый пациент может записаться на желаемую услугу. При этом формируется талон – объект, на котором зафиксирован определенный номер, информация об услуге и о враче.

Модель talons содержит следующие поля:

* patient – INT(11) – идентификационный номер пациента;
* doctor – VARCHAR(45) – номер лицензии специалиста, который оказывает услугу;
* service – VARCHAR(45) – название услуги;
* date – VARCHAR(45) – дата приема;
* time – VARCHAR(45) – время приема;
* talon\_number – VARCHAR(45) – номер талона;

# **3. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Проведем тестирование всех функциональных возможностей программы.

## **3.1 Тест 1**

Тестовая ситуация: авторизация существующего пользователя.

Ожидаемый результат: успешная авторизация, загрузка личного кабинета.

Результат:

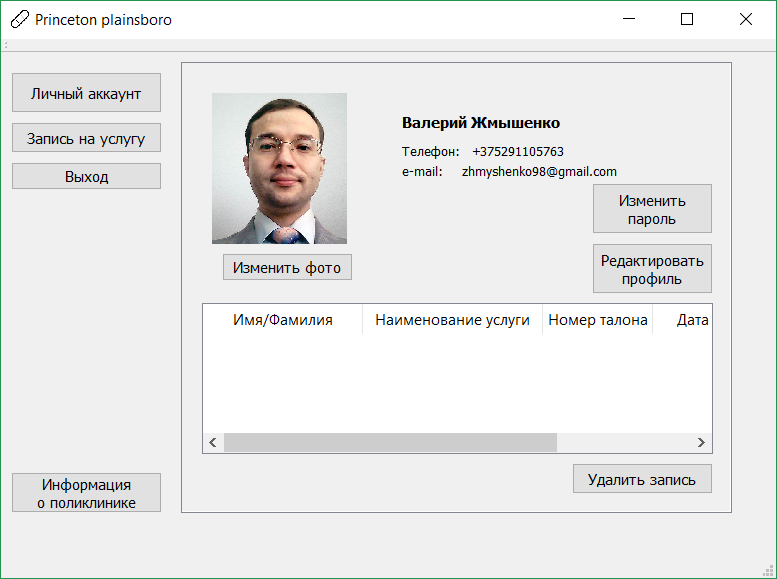
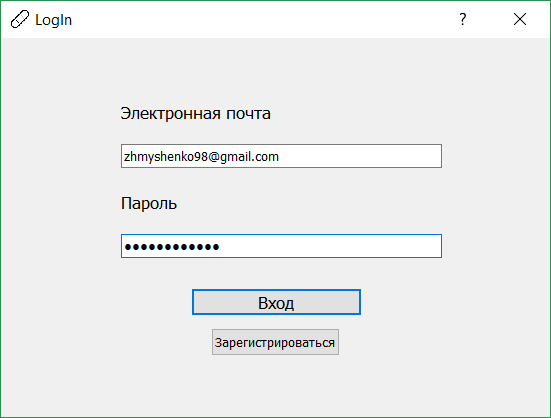


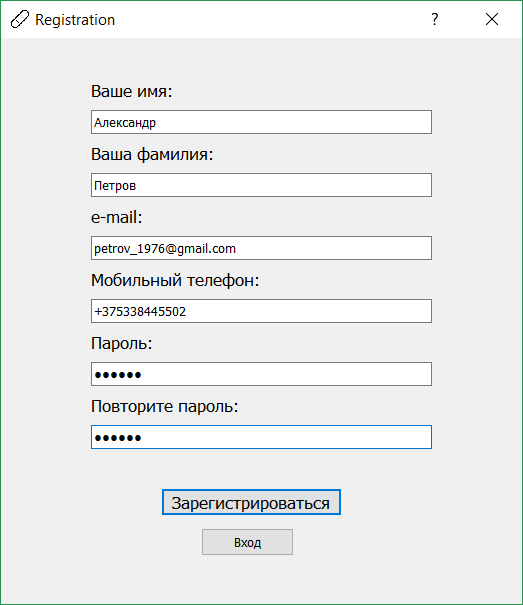
Рис. 3.1

Примечание: в случае некорректного ввода электронной почты или пароля будет показана ошибка.

## **3.2 Тест 2**

Тестовая ситуация: регистрация нового аккаунта.

Ожидаемый результат: успешная регистрация, создание личного кабинета.

Результат:

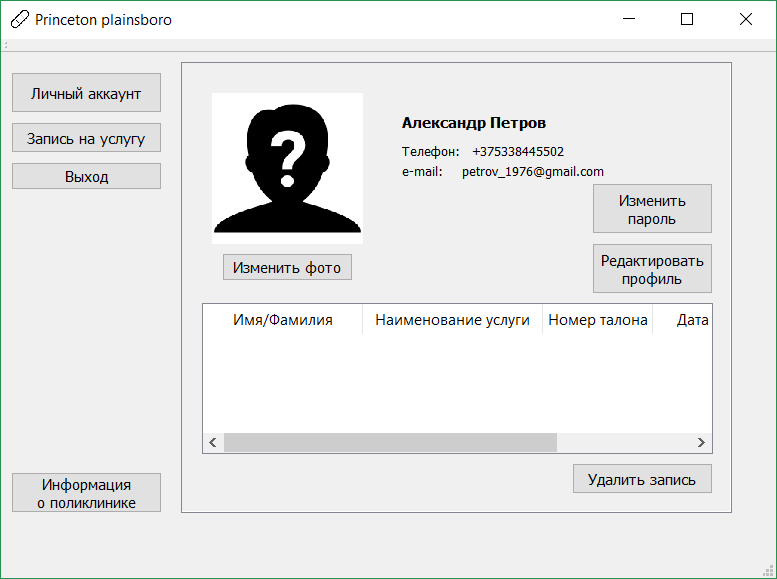


Рис 3.2

Примечание: в случае пропуска ввода одного из полей или при попытке создания аккаунта по уже существующей почте будет выведена ошибка.

## **3.3 Тест 3**

Тестовая ситуация: просмотр информации о врачах поликлиники

Ожидаемый результат: отображение информации о врачах поликлиники

Результат:

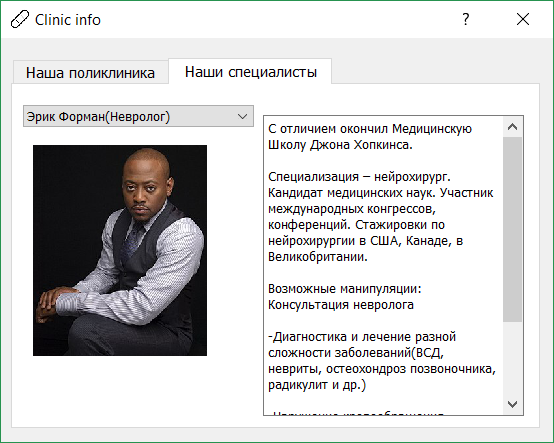


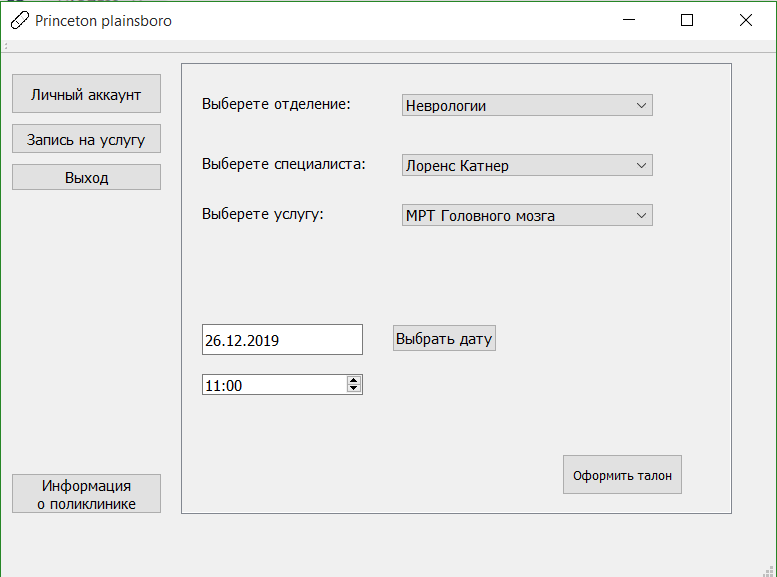
Рис 3.3

## **3.4 Тест 4**

Тестовая ситуация: запись на МРТ головного мозга.

Ожидаемый результат: успешная запись и создание талона.

Результат:



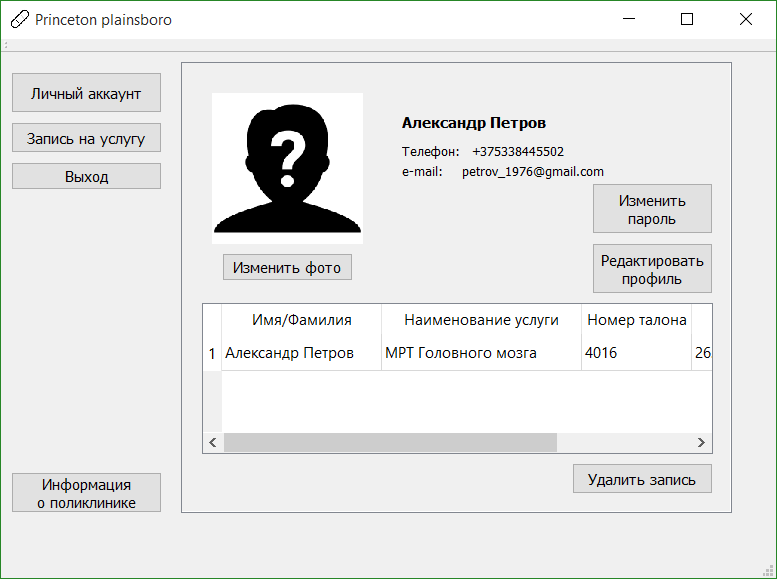


Рис. 3.4

## **3.5 Тест 5**

Тестовая ситуация: изменение имени.

Ожидаемый результат: успешное изменение имени пользователя в личном аккаунте.

Результат:

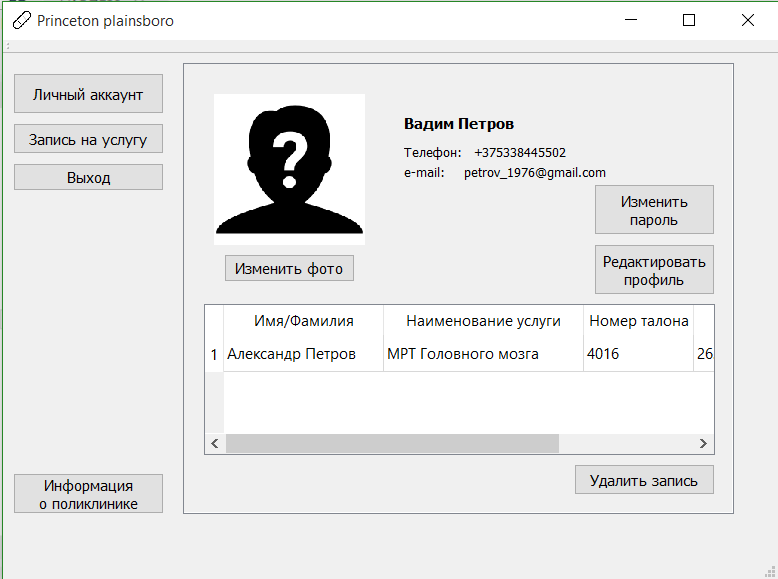
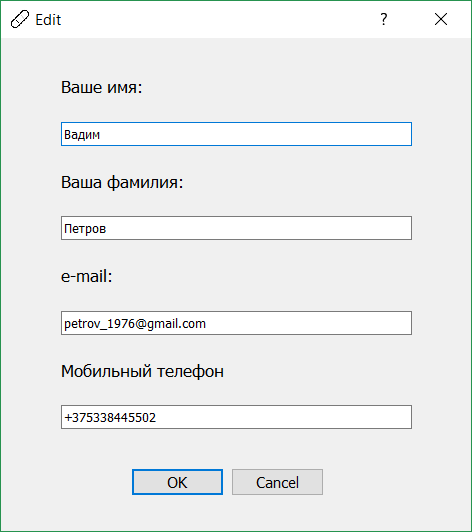


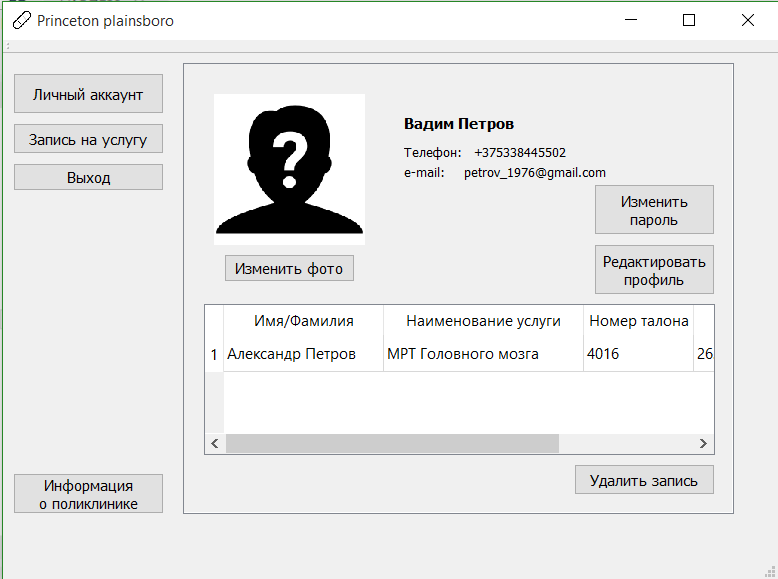
Рис. 3.5

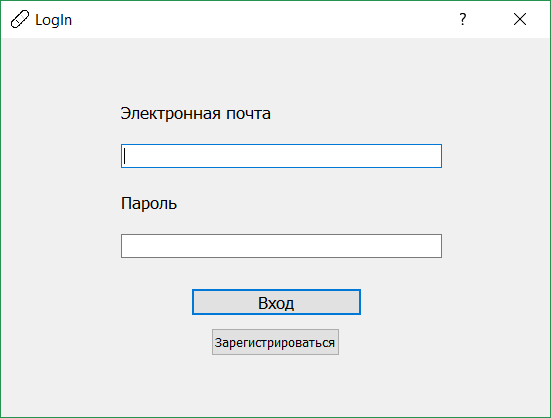
## **3.6 Тест 6**

Тестовая ситуация: выход из аккаунта.

Ожидаемый результат: успешный выход из аккаунта и возврат к окну входа.

Результат:

 Рис. 3.6

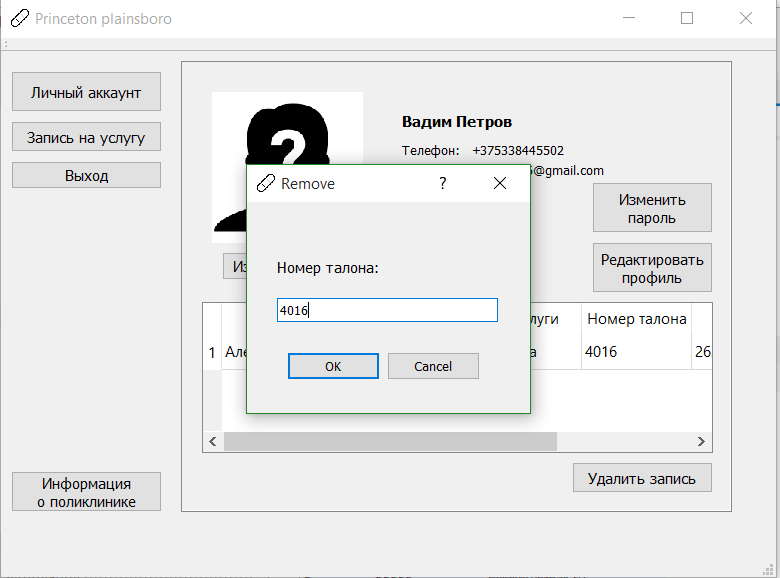


## **3.7 Тест 7**

Тестовая ситуация: удаление записи.

Ожидаемый результат: успешный удаление записи и отображение на окне

Результат:



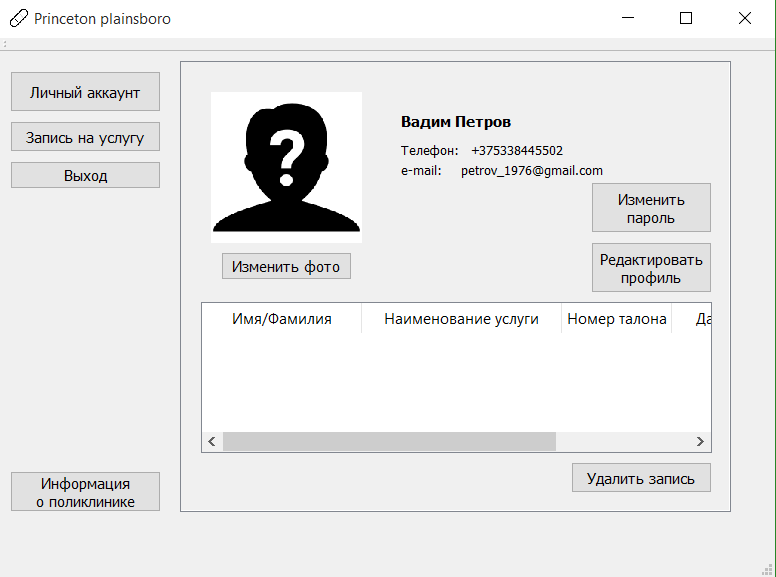


Рис. 3.7

# **4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## **4.1 Интерфейс программы и порядок работы**

Для приложения «Поликлиника» был разработан интуитивно понятный интерфейс.

При открытии приложения нас встречает окно входа. С помощью окна входа пользователь может авторизоваться для того, чтобы войти в свой личный кабинет пациента или перейти к окну регистрации, для создания личного кабинета пациента.(рис. 4.1)

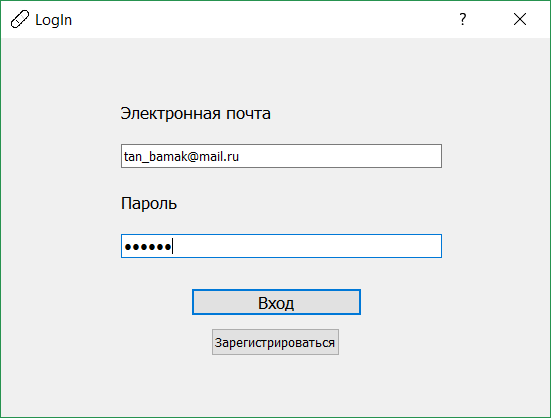


Рис. 4.1

При условии успешной авторизации, нас встречает окно личного кабинета пациента (Рис. 4.2). На нем представлена контактная информация пользователя, а также показаны активные заказы на услуги специалистов поликлинике. Правее центра располагаются кнопки управления аккаунтом, благодаря которым можно обновить информацию о пациенте, изменить пароль или фотографию аккаунта. В верхнем левом углу располагаются кнопки заказа услуг и выхода. При нажатии кнопки заказ услуг, откроется меню выбора услуги и даты проведения данной услуги (Рис. 4.3). В левом нижнем углу располагается кнопка вызова диалогового окна, отображающего информацию о специалистах, трудящихся в поликлинике.

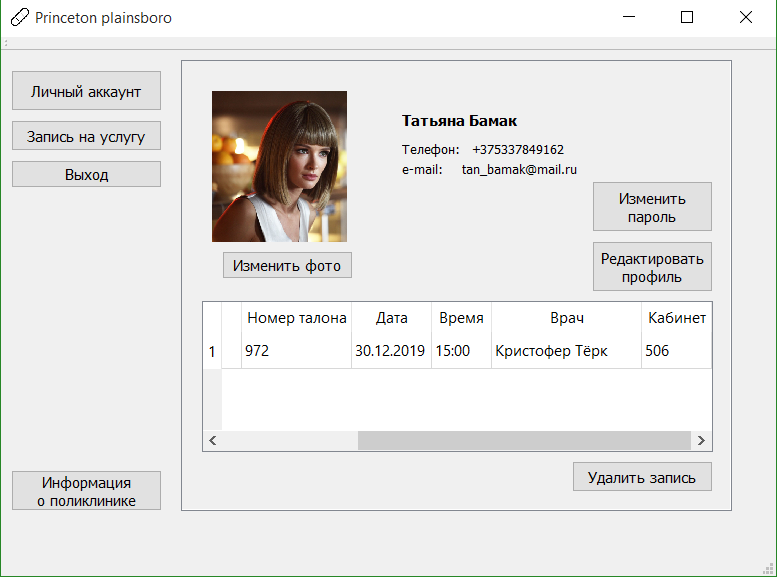
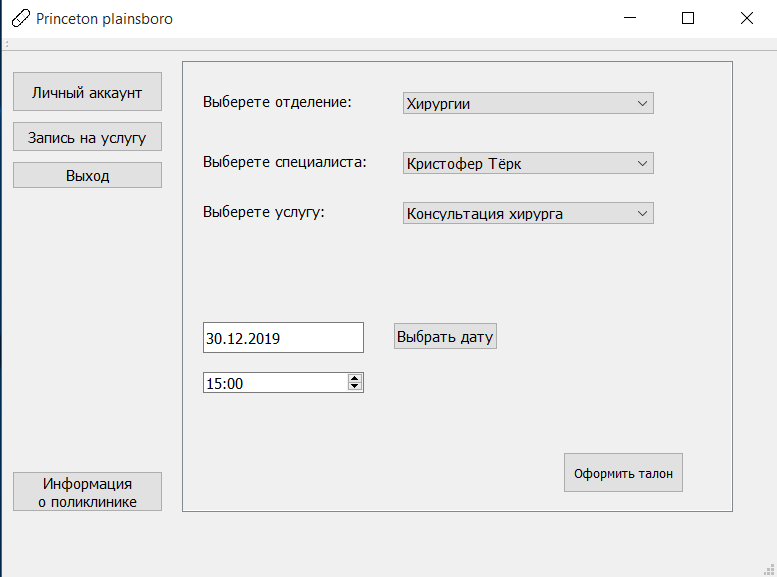


Рис. 4.2



По нажатии на выпадающие списки пользователь может выбрать соответствующее отделение, специалиста и услугу. Нажав кнопку “выбрать дату” появится окно с календарём (Рис. 4.4). Достаточно кликнуть на подходящий день, чтобы это отобразилось на окне оформления услуги.

Если все выпадающие списки заполнены и выбрана правильная дата(в выходной день поликлиника не работает), то при нажатии на кнопку “Оформить талон” будет отображено сообщение, что запись принята (Рис. 4.5). В том случае, когда идентичная запись уже существует, приложение проинформирует вас об этом (Рис. 4.6).

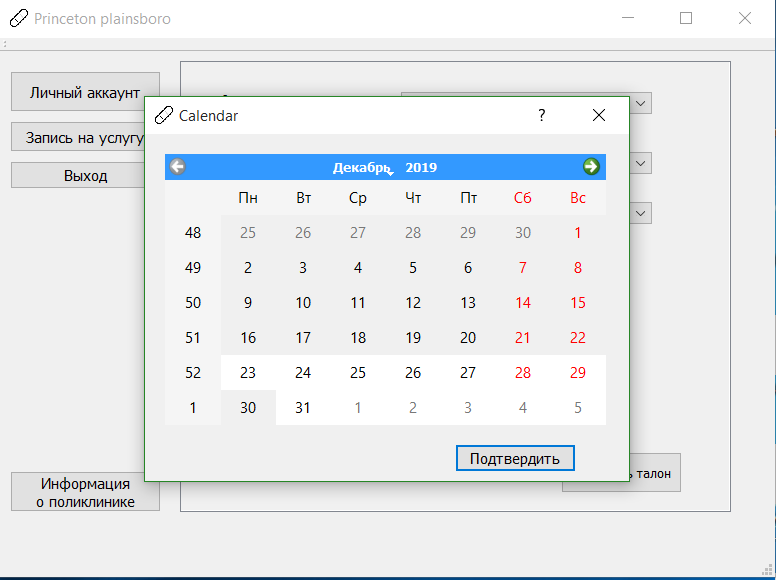


Рис.4.4

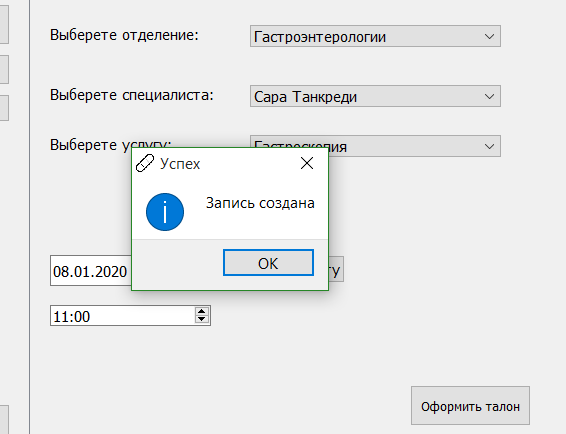


Рис.4.5

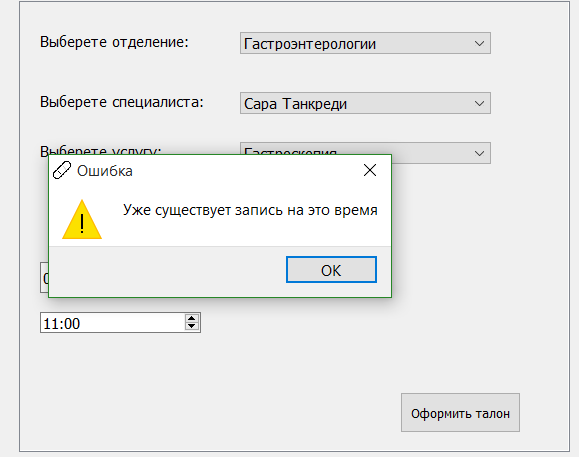


Рис.4.6

При нажатии в окне личного кабинета пациента (Рис. 4.2) кнопки “Изменить фото” пользователь может загрузить фото со своего носителя информации. Кнопки “Изменить пароль” и “Редактировать профиль” позволяют исправить какую-либо личную информацию. По их нажатии показываются окна (Рис. 4.7) и (Рис. 4.8) соответственно. Чтобы изменить пароль нужна правильно ввести старый, далее ввести новый и нажать “Ок”. При редактировании информации о себе просто измените её в нужных полях. О Вас позаботится приложение, выдав сообщение, если некоторые поля не были заполнены.

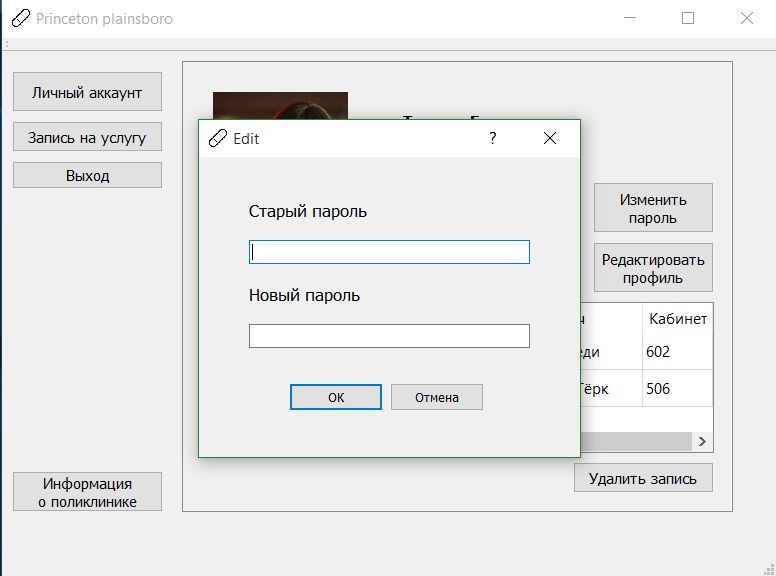


Рис.4.7

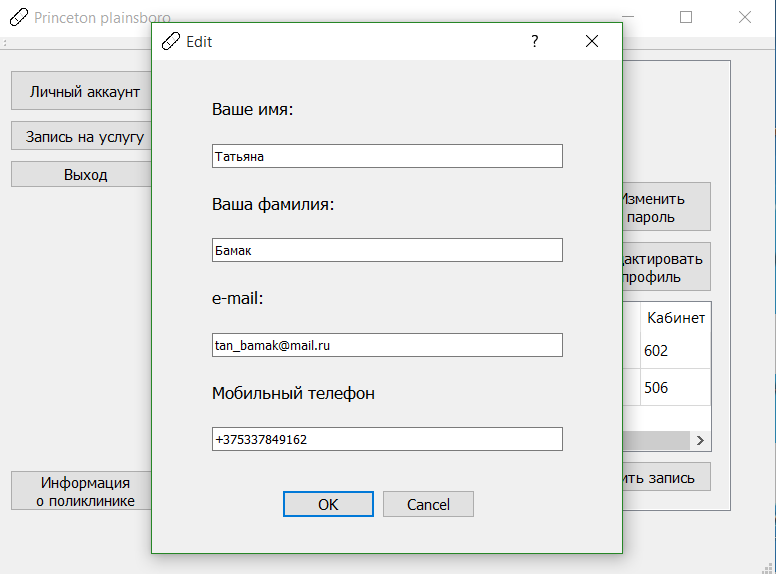


Рис.4.8

Если нажать на кнопку “Информация о поликлинике”, которая находится в левом нижнем углу окна (Рис. 4.2), то можно познакомиться с небольшой биографией каждого из работников поликлиники и с описании поликлиники в общем (Рис. 4.9). Для этого последовательно переключайте вкладки “Наша поликлиника” и “Наши специалисты”, при этом в последней, выбирая из ниспадающего списка работников, можно познакомиться с каждым из них! В целом, главная задача разработки пользовательского интерфейса – это стремление создать такой интерфейс приложения, который будет понятен абсолютно любому человеку без каких-либо объяснений.

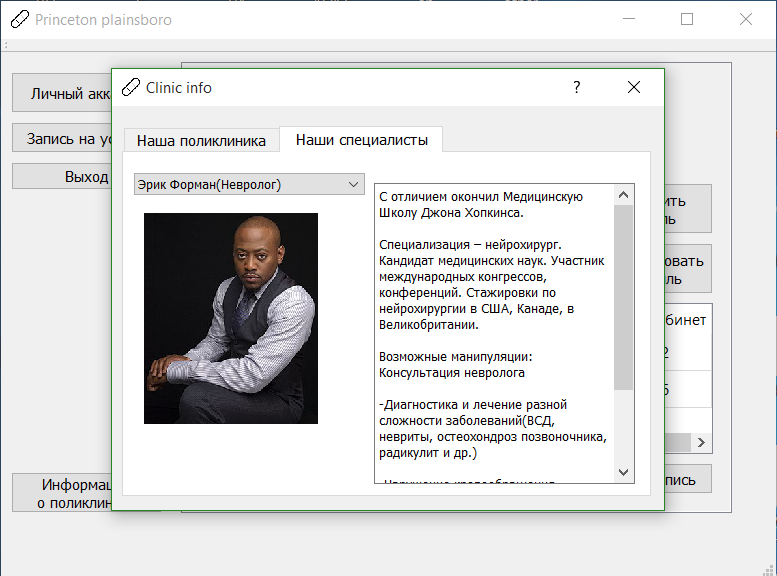


Рис.4.9

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта было разработано приложение «Поликлиника» с использованием базы данных. В ходе его разработки были получены теоретические знания по основам теории баз данных, на практике были изучены способы взаимодействия с ними. Были изучены базовые запросы языка SQL, а также способы работы с MySQL Server.

В ходе работы на практике появились навыки разработки приложений с использованием фреймворка Qt и языка программирования C++, навыки разработки графических интерфейсов с помощью дизайнера программы Qt Creator. Считаю, что в дальнейшем следует углубленно изучать подобную среду.

Разработанное приложение выполняет все основные функции, оглашенные в задании к курсовой работе. Приложение представляет собой удобную систему для заказа талонов у врачей поликлиники и последующего их администрирования. Также, в ходе работы был разработан интуитивно понятный пользовательский интерфейс.

# 

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Голицына О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб.пособие. – 2-е изд., испр. и доп. – М.: ФОРУМ: ИНФРА-М, 2009. – 400с.: ил. – (Профессиональное образование). ISBN 978-5-91134-098 – 8 .

[2] Карпова Т.С Базы данных: модели, разработка, реализация / Т, С. Карпова. — СПб.: Питер, 2001. —304 с.; ил. ISBN 5-272-00278-4 .

[3] Хомопепко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных: Учебник для высших учебных заведений / Под ред. np(xj). А. Д. Хомопепко. — 6-еизд., доп. - СПб.:КОРОНА-Век,2009. - 736 с. ISBN978-5-7931-0527-9 .

[4] Документация по QT [Электронный ресурс] – Электроннные данные. – Режим доступа: <https://doc.qt.io> .

[5] Бьюли А. Изучаем SQL. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 312 с., ил.ISBN-13 : 978-5-93286-051-9.

[6] Программирование на языке С++ Герберт Шилдт, Базовый курс – 3-е изд. – СПб. – 845с.

[7] Программирование на языке С++ в среде Qt Creator: Е.Р. Алексеев, Г. Г. Злобин, Д. А.Костюк,О.В.Чеснокова, А.С.Чмыхало \_ М. : ALT Linux, 2015. \_ 448 с. : ил. \_ (Библиотека ALT Linux).

# **ПРИЛОЖЕНИЕ A**

# **Исходный код программы**

Основные классы:

Класс MainWindow

Файл заголовка(.h)

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtSql>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QMessageBox>

#include "logindialog.h"

#include "registrationdialog.h"

#include "useraccount.h"

#include "clinicinfodialog.h"

#include "editdialog.h"

#include "passwordeditdialog.h"

#include "datedialog.h"

#include "removedialog.h"

#include <QFileDialog>

#include <QStandardItem>

#define DriverName "QMYSQL"

#define DatabaseConnectionName "MySQL80"

#define DatabaseName "clinic"

#define DatabasePassword "2438041"

#define DatabaseUseName "root"

#define DatabaseHostName "localhost"

#define Admin\_ID 2

class **LogInDialog**;

class **RegistrationDialog**;

class **ClinicInfoDialog**;

class **EditDialog**;

class **PasswordEditDialog**;

class **DateDialog**;

class **RemoveDialog**;

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

static int const EXIT\_CODE\_REBOOT;

signals:

void **registration**();

void **authorization**();

void **registrated**();

void **setPassword**();

void **setServices**();

void **selected**();

void **getted**();

void **accountGetted**();

private slots:

void **getDate**();

void **openAuthorization**();

void **openRegistration**();

void **setAuthorization**();

void **setNewAccount**();

void **setNewPassword**();

void **slot\_setServices**();

void **newView**();

void **on\_buttonClinicInfo\_clicked**();

void **on\_buttonAccount\_clicked**();

void **on\_buttonEdit\_clicked**();

void **on\_buttonPasswordEdit\_clicked**();

void **on\_buttonService\_clicked**();

void **on\_buttonDataSelect\_clicked**();

void **on\_buttonSetTalon\_clicked**();

void **on\_buttonRemoveTalon\_clicked**();

void **on\_buttonExit\_clicked**();

void **on\_buttomChangePhoto\_clicked**();

void **on\_comboBoxSpecialist\_currentIndexChanged**();

void **on\_comboBoxDepartment\_currentIndexChanged**();

private:

QString currentPhoto;

void **tableView**();

QStandardItemModel\* **createmodel**(QObject \*parent);

UserAccount currentAccount;

void **createAccount**();

void **LoadDataBase**();

QSqlDatabase clinic;

Ui::MainWindow \*ui;

LogInDialog \*mLogInDialog;

RegistrationDialog \*mRegistrationDialog;

ClinicInfoDialog \*mClinicInfoDialog;

EditDialog \*mEditDialog;

PasswordEditDialog \*mPasswordEditDialog;

DateDialog \*mDateDialog;

RemoveDialog \*mRemoveDialog;

};

#endif // MAINWINDOW\_H

Файл исходник(.cpp):

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(parent),

ui(new Ui::MainWindow),

mLogInDialog(new LogInDialog(this)),

mRegistrationDialog(new RegistrationDialog(this)),

mClinicInfoDialog(new ClinicInfoDialog(this)),

mEditDialog(new EditDialog(this)),

mPasswordEditDialog(new PasswordEditDialog(this)),

mDateDialog(new DateDialog(this)),

mRemoveDialog(new RemoveDialog(this))

{

ui->setupUi(this);

QTime time = QTime::currentTime();

qsrand((uint)time.msec());

connect(mLogInDialog,SIGNAL(registration()),this,SLOT(openRegistration()),Qt::UniqueConnection);

connect(mRegistrationDialog,SIGNAL(authorization()),this,SLOT(openAuthorization()),Qt::UniqueConnection);

connect(mRegistrationDialog,SIGNAL(registrated()),this,SLOT(setAuthorization()),Qt::UniqueConnection);

connect(mEditDialog,SIGNAL(accountGetted()),this,SLOT(setNewAccount()),Qt::UniqueConnection);

connect(mPasswordEditDialog,SIGNAL(setPassword()),this,SLOT(setNewPassword()),Qt::UniqueConnection);

connect(this,SIGNAL(setServices()),this,SLOT(slot\_setServices()),Qt::UniqueConnection);

connect(mDateDialog,SIGNAL(selected()),this,SLOT(getDate()),Qt::UniqueConnection);

connect(mRemoveDialog,SIGNAL(getted()),this,SLOT(newView()),Qt::UniqueConnection);

LoadDataBase();

mLogInDialog->getBase(clinic);

mRegistrationDialog->getBase(clinic);

this->hide();

mLogInDialog->*exec*();

this->show();

createAccount();

if(currentAccount.GetIdNumber() == Admin\_ID)

{

ui->buttonEdit->*setVisible*(false);

ui->buttonEdit->setEnabled(false);

ui->buttonAccount->*setVisible*(false);

ui->buttonAccount->setEnabled(false);

ui->buttonService->*setVisible*(false);

ui->buttonService->setEnabled(false);

ui->buttonPasswordEdit->*setVisible*(false);

ui->buttonPasswordEdit->setEnabled(false);

ui->buttomChangePhoto->*setVisible*(false);

ui->buttomChangePhoto->setEnabled(false);

}

tableView();

}

int const MainWindow::EXIT\_CODE\_REBOOT = -123456789;

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

{

delete ui;

}

void MainWindow::**setNewAccount**()

{

UserAccount temp = mEditDialog->setAccount();

currentAccount = temp;

ui->labelName->setText(currentAccount.GetName() + " " + currentAccount.GetSurname());

ui->labelEmail->setText(currentAccount.GetEmail());

ui->labelPhone->setText(currentAccount.GetPhoneNumber());

currentPhoto = currentAccount.GetPhotoPath();

QPixmap pixMap(currentPhoto);

int h = ui->personalPhoto->height();

int w = ui->personalPhoto->width();

ui->personalPhoto->setPixmap(pixMap.scaled(w,h,Qt::KeepAspectRatio));

on\_buttonAccount\_clicked();

QSqlQuery query(clinic);

query.exec("UPDATE patients SET patientsphoto = '" + currentAccount.GetPhotoPath() +

"',name = '" + currentAccount.GetName() +

"',surname = '" + currentAccount.GetSurname() +

"',email = '" + currentAccount.GetEmail() +

"',phonenumber = '" + currentAccount.GetPhoneNumber() +

"' WHERE (id\_number = '" + QString::number(currentAccount.GetIdNumber()) + "')");

query.exec("UPDATE users SET email = '" + currentAccount.GetEmail() +

"' WHERE (id\_users = '" + QString::number(currentAccount.GetIdNumber()) + "')");

}

void MainWindow::**setAuthorization**()

{

mLogInDialog->setRegAccount(mRegistrationDialog->getCurrentNumber());

}

void MainWindow::**openRegistration**()

{

mRegistrationDialog->*exec*();

}

void MainWindow::**openAuthorization**()

{

mRegistrationDialog->close();

}

void MainWindow::**createAccount**()

{

UserAccount temp = mLogInDialog->getAccount();

currentAccount = temp;

ui->labelName->setText(currentAccount.GetName() + " " + currentAccount.GetSurname());

ui->labelEmail->setText(currentAccount.GetEmail());

ui->labelPhone->setText(currentAccount.GetPhoneNumber());

currentPhoto = currentAccount.GetPhotoPath();

QPixmap pixMap(currentPhoto);

int h = ui->personalPhoto->height();

int w = ui->personalPhoto->width();

ui->personalPhoto->setPixmap(pixMap.scaled(w,h,Qt::KeepAspectRatio));

on\_buttonAccount\_clicked();

}

void MainWindow::**LoadDataBase**()

{

clinic = QSqlDatabase::addDatabase(DriverName, DatabaseConnectionName);

clinic.setDatabaseName(DatabaseName);

clinic.setPassword(DatabasePassword);

clinic.setUserName(DatabaseUseName);

clinic.setHostName(DatabaseHostName);

if(!clinic.open())

QMessageBox::warning(this,"Error","Произошла ошибка подключения базы данных!",QMessageBox::Ok);

}

void MainWindow::**setNewPassword**()

{

QString currentPassword = mPasswordEditDialog->getNewPassword();

QSqlQuery query = QSqlQuery(clinic);

query.exec("UPDATE users SET password = '" + currentPassword + "' WHERE (id\_users = '" +

QString::number(currentAccount.GetIdNumber()) + "')");

}

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

{

mClinicInfoDialog->*open*();

mClinicInfoDialog->getDataBase(clinic);

}

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

{

ui->accountPages->setCurrentIndex(0);

}

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

{

mEditDialog->show();

mEditDialog->getAccount(currentAccount);

}

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

{

mPasswordEditDialog->show();

QSqlQuery query = QSqlQuery(clinic);

query.exec("SELECT \* FROM users WHERE id\_users = '" + QString::number(currentAccount.GetIdNumber()) + "'");

query.next();

mPasswordEditDialog->getOldPassword(query.value(1).toString());

}

void MainWindow::**slot\_setServices**()

{

QSqlQuery query = QSqlQuery(clinic);

query.exec("SELECT \* FROM departments");

while(query.next())

{

ui->comboBoxDepartment->addItem(query.value(1).toString());

}

}

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

{

ui->accountPages->setCurrentIndex(1);

emit setServices();

}

void MainWindow::**on\_comboBoxDepartment\_currentIndexChanged**()

{

QString text = ui->comboBoxDepartment->currentText();

QSqlQuery query = QSqlQuery(clinic);

query.exec("SELECT \* FROM departments WHERE name = '" + text + "'");

query.next();

text = query.value(0).toString();

query.exec("SELECT \* FROM doctors WHERE department = '" + text + "'");

ui->comboBoxSpecialist->clear();

while(query.next())

{

ui->comboBoxSpecialist->addItem(query.value(3).toString());

}

}

void MainWindow::**getDate**()

{

ui->editDateTime->setText(QDate(mDateDialog->getDate()).toString("dd.MM.yyyy"));

}

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

{

mDateDialog->show();

}

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

{

QSqlQuery query = QSqlQuery(clinic);

query.exec("SELECT \* FROM services WHERE name = '" + ui->comboBoxService->currentText() + "'");

query.next();

QString service = query.value(0).toString();

query.exec("SELECT \* FROM doctors WHERE name = '" + ui->comboBoxSpecialist->currentText() + "'");

query.next();

QString specialist = query.value(0).toString();

if(QString(ui->editDateTime->text()).isEmpty())

QMessageBox::warning(this,"Ошибка","Вы не выбрали дату приема!",QMessageBox::Ok);

else

{

query.exec("SELECT \* FROM talons WHERE date = '" + ui->editDateTime->text() + "' AND time = '" +

QTime(ui->editTime->time()).toString("hh:mm") + "' AND doctor = '" + specialist + "'");

if(query.next())

{

QMessageBox::warning(this,"Ошибка","Уже существует запись на это время",QMessageBox::Ok);

while(query.next());

}

else

{

query.prepare("INSERT INTO talons (patient,doctor,service,date,time,talon\_number)"

" VALUES (?,?,?,?,?,?)");

query.addBindValue(currentAccount.GetIdNumber());

query.addBindValue(specialist);

query.addBindValue(service);

query.addBindValue(ui->editDateTime->text());

query.addBindValue(QTime(ui->editTime->time()).toString("hh:mm"));

query.addBindValue(QString::number(qrand()%5000));

query.exec();

QMessageBox::information(this,"Успех","Запись создана",QMessageBox::Ok);

}

tableView();

}

}

QStandardItemModel\* MainWindow::**createmodel**(QObject \*parent)

{

QSqlQuery query = QSqlQuery(clinic);

if(currentAccount.GetIdNumber() != Admin\_ID)

query.exec("SELECT \* FROM talons WHERE patient = " + QString::number(currentAccount.GetIdNumber()));

else

query.exec("SELECT \* FROM talons");

const int numRows = query.size();

const int numColumns = 7;

QStringList Headers = {"Имя/Фамилия","Наименование услуги","Номер талона","Дата","Время","Врач","Кабинет"};

QStandardItemModel\* model = new QStandardItemModel(numRows, numColumns);

model->setHorizontalHeaderLabels(Headers);

query.next();

for (int row = 0; row < numRows; ++row)

{

int column = 0;

QSqlQuery nameQuery = QSqlQuery(clinic);

nameQuery.exec("SELECT \* FROM patients WHERE id\_number = " + QString::number(query.value(0).toUInt()));

nameQuery.next();

QString text = nameQuery.value(1).toString() + " " + nameQuery.value(2).toString();

QStandardItem\* itemName = new QStandardItem(text);

model->setItem(row, column, itemName);

column++;

QSqlQuery doctorQuery = QSqlQuery(clinic);

doctorQuery.exec("SELECT \* FROM doctors WHERE license\_number = '" + query.value(1).toString() + "'");

doctorQuery.next();

QSqlQuery serviceQuery = QSqlQuery(clinic);

serviceQuery.exec("SELECT \* FROM services WHERE id\_number = '" + query.value(2).toString() + "'");

serviceQuery.next();

text = serviceQuery.value(1).toString();

QStandardItem\* itemService = new QStandardItem(text);

model->setItem(row, column, itemService);

text = query.value(5).toString();

column++;

QStandardItem\* itemTalon = new QStandardItem(text);

model->setItem(row, column, itemTalon);

column++;

text = query.value(3).toString();

QStandardItem\* itemDate = new QStandardItem(text);

model->setItem(row, column, itemDate);

column++;

text = query.value(4).toString();

QStandardItem\* itemTime = new QStandardItem(text);

model->setItem(row, column, itemTime);

column++;

text = doctorQuery.value(3).toString();

QStandardItem\* itemDoctor = new QStandardItem(text);

model->setItem(row, column, itemDoctor);

column++;

text = doctorQuery.value(7).toString();

QStandardItem\* itemCabinet = new QStandardItem(text);

model->setItem(row, column, itemCabinet);

column++;

query.next();

}

return model;

}

void MainWindow::**tableView**()

{

ui->viewTalon->*setModel*(createmodel(ui->viewTalon));

ui->viewTalon->setColumnWidth(0,160);

ui->viewTalon->setColumnWidth(1,200);

ui->viewTalon->setColumnWidth(2,110);

ui->viewTalon->setColumnWidth(3,80);

ui->viewTalon->setColumnWidth(4,60);

ui->viewTalon->setColumnWidth(5,150);

ui->viewTalon->setColumnWidth(6,70);

}

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

{

mRemoveDialog->show();

mRemoveDialog->getInfo(clinic,currentAccount.GetIdNumber());

}

void MainWindow::**newView**()

{

tableView();

}

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

{

qApp->exit( MainWindow::EXIT\_CODE\_REBOOT );

}

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

{

QString FileName = QFileDialog::getOpenFileName(this,"Open file",QDir::homePath(),"PNG files (\*.png);;All files (\*.\*)");

if(FileName != "")

currentPhoto = FileName;

QPixmap pixMap(currentPhoto);

int h = ui->personalPhoto->height();

int w = ui->personalPhoto->width();

ui->personalPhoto->setPixmap(pixMap.scaled(w,h,Qt::KeepAspectRatio));

QSqlQuery query(clinic);

query.exec("UPDATE patients SET patientsphoto = '" + currentPhoto +

"' WHERE (id\_number = '" + QString::number(currentAccount.GetIdNumber()) + "')");

}

void MainWindow::**on\_comboBoxSpecialist\_currentIndexChanged**()

{

QString text = ui->comboBoxSpecialist->currentText();

QSqlQuery query = QSqlQuery(clinic);

query.exec("SELECT \* FROM doctors WHERE name = '" + text + "'");

query.next();

text = query.value(6).toString();

QStringList list = text.split(",");

query.exec("SELECT \* FROM services");

ui->comboBoxService->clear();

while(query.next())

{

for(int i = 0; i < list.size();i++)

{

if(query.value(0).toString() == list.at(i))

ui->comboBoxService->addItem(query.value(1).toString());

}

}

}

Класс LogInDialog

Файл заголовка(.h)

#ifndef LOGINDIALOG\_H

#define LOGINDIALOG\_H

#include <QDialog>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QMessageBox>

#include "useraccount.h"

namespace **Ui** {

class **LogInDialog**;

}

class **LogInDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **LogInDialog**(QWidget \*parent = nullptr);

~***LogInDialog***();

UserAccount **getAccount**();

void **setRegAccount**(uint number);

private slots:

void **on\_buttonEnter\_clicked**();

void **on\_buttonRegistration\_clicked**();

signals:

void **registration**();

void **registrated**();

void **enter**();

public slots:

void **getBase**(QSqlDatabase base);

private:

QSqlQuery mQuery;

QSqlDatabase base;

Ui::LogInDialog \*ui;

};

#endif // LOGINDIALOG\_H

Файл реализации(.cpp)

#include "logindialog.h"

#include "ui\_logindialog.h"

LogInDialog::**LogInDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::LogInDialog)

{

ui->setupUi(this);

}

LogInDialog::~***LogInDialog***()

{

delete ui;

}

void LogInDialog::**getBase**(QSqlDatabase base)

{

this->base = base;

}

UserAccount LogInDialog::**getAccount**()

{

uint idNumber = mQuery.value(0).toUInt();

mQuery.exec("SELECT \* FROM patients WHERE id\_number = '" + QString::number(idNumber) + "'");

mQuery.next();

return UserAccount(mQuery.value(1).toString(),mQuery.value(2).toString(),

mQuery.value(4).toString(),mQuery.value(3).toString(),

mQuery.value(0).toUInt(),mQuery.value(5).toString());

}

void LogInDialog::**setRegAccount**(uint number)

{

mQuery = QSqlQuery(base);

mQuery.exec("SELECT \* FROM users WHERE id\_users = " + QString::number(number));

mQuery.next();

ui->editNickname->setText(mQuery.value(2).toString());

ui->editPassword->setText(mQuery.value(1).toString());

}

void LogInDialog::**on\_buttonEnter\_clicked**()

{

mQuery = QSqlQuery(base);

mQuery.exec("SELECT \* FROM users WHERE email = '" + ui->editNickname->text() +

"' AND password = '" + ui->editPassword->text() +"'");

if(!mQuery.next())

{

QMessageBox::warning(this,"Ошибка","Неверно введен логин/пароль!",QMessageBox::Ok);

}

else

{

ui->editNickname->clear();

ui->editPassword->clear();

this->close();

}

}

void LogInDialog::**on\_buttonRegistration\_clicked**()

{

emit registration();

}

Класс RegistrationDialog

Файл заголовка(.h):

#ifndef REGISTRATIONDIALOG\_H

#define REGISTRATIONDIALOG\_H

#include <QDialog>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QMessageBox>

namespace **Ui** {

class **RegistrationDialog**;

}

class **RegistrationDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **RegistrationDialog**(QWidget \*parent = nullptr);

~***RegistrationDialog***();

void **getBase**(QSqlDatabase base);

uint **getCurrentNumber**();

signals:

void **authorization**();

void **registrated**();

private slots:

void **on\_buttonEnter\_clicked**();

void **on\_buttonRegistration\_clicked**();

private:

uint currentNumber;

QSqlDatabase base;

Ui::RegistrationDialog \*ui;

};

#endif // REGISTRATIONDIALOG\_H

Файл реализации(.cpp)

#include "registrationdialog.h"

#include "ui\_registrationdialog.h"

RegistrationDialog::**RegistrationDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::RegistrationDialog)

{

ui->setupUi(this);

}

RegistrationDialog::~***RegistrationDialog***()

{

delete ui;

}

void RegistrationDialog::**getBase**(QSqlDatabase base)

{

this->base = base;

}

void RegistrationDialog::**on\_buttonEnter\_clicked**()

{

emit authorization();

}

uint RegistrationDialog::**getCurrentNumber**()

{

return currentNumber;

}

void RegistrationDialog::**on\_buttonRegistration\_clicked**()

{

if(QString(ui->editName->text()).isEmpty()||QString(ui->editSurname->text()).isEmpty()

||QString(ui->editPhoneNumber->text()).isEmpty()

||QString(ui->editEmail->text()).isEmpty()||QString(ui->editPassword->text()).isEmpty()

||QString(ui->editCheckPassword->text()).isEmpty())

{

QMessageBox::warning(this,"Ошибка!","Вы не заполнили некоторые поля!",QMessageBox::Ok);

}

else

if(ui->editPassword->text() != ui->editCheckPassword->text())

{

QMessageBox::warning(this,"Ошибка!","Пароли не совпадают",QMessageBox::Ok);

}

else

{

bool indicator = false;

QSqlQuery mQuery = QSqlQuery(base);

currentNumber = 1;

mQuery.exec("SELECT \* FROM users");

while(mQuery.next())

{

currentNumber++;

if(mQuery.value(2).toString() == ui->editEmail->text())

indicator = true;

}

if(indicator)

QMessageBox::warning(this,"Ошибка!","Пользователь с таким email уже существует!",QMessageBox::Ok);

else

{

mQuery.clear();

mQuery.prepare("INSERT INTO patients (id\_number,name,surname,phonenumber,email)"

" VALUES (?,?,?,?,?)");

mQuery.addBindValue(currentNumber);

mQuery.addBindValue(ui->editName->text());

mQuery.addBindValue(ui->editSurname->text());

mQuery.addBindValue(ui->editPhoneNumber->text());

mQuery.addBindValue(ui->editEmail->text());

mQuery.exec();

mQuery.prepare("INSERT INTO users (id\_users, password, email) VALUES (?,?,?)");

mQuery.addBindValue(currentNumber);

mQuery.addBindValue(ui->editPassword->text());

mQuery.addBindValue(ui->editEmail->text());

mQuery.exec();

mQuery.clear();

ui->editName->clear();

ui->editEmail->clear();

ui->editSurname->clear();

ui->editPassword->clear();

ui->editPhoneNumber->clear();

ui->editCheckPassword->clear();

emit registrated();

this->close();

}

}

}

Файл main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

int currentExitCode = 0;

do

{

QApplication a(*argc*, argv);

QCoreApplication::setApplicationName("Princeton plainsboro");

MainWindow w;

w.show();

currentExitCode = a.exec();

}

while(currentExitCode == MainWindow::EXIT\_CODE\_REBOOT);

return currentExitCode;

}

Класс EditDialog

Файл заголовка(.h)

#ifndef EDITDIALOG\_H

#define EDITDIALOG\_H

#include <QDialog>

#include <useraccount.h>

#include <QMessageBox>

namespace **Ui** {

class **EditDialog**;

}

class **EditDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **EditDialog**(QWidget \*parent = nullptr);

~***EditDialog***();

UserAccount **setAccount**();

public slots:

void **getAccount**(UserAccount current);

void **clear**();

void **setNewAccount**();

signals:

void **accountGetted**();

private:

UserAccount newUser;

uint number;

QString photo;

Ui::EditDialog \*ui;

};

#endif // EDITDIALOG\_H

Файл реализации(.cpp)

#include "editdialog.h"

#include "ui\_editdialog.h"

EditDialog::**EditDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::EditDialog)

{

ui->setupUi(this);

connect(ui->buttonBox,SIGNAL(rejected()),this,SLOT(clear()),Qt::UniqueConnection);

connect(ui->buttonBox,SIGNAL(accepted()),this,SLOT(setNewAccount()),Qt::UniqueConnection);

}

void EditDialog::**getAccount**(UserAccount current)

{

number = current.GetIdNumber();

ui->editName->setText(current.GetName());

ui->editEmail->setText(current.GetEmail());

ui->editSurname->setText(current.GetSurname());

ui->editPhoneNumber->setText(current.GetPhoneNumber());

photo = current.GetPhotoPath();

}

void EditDialog::**clear**()

{

ui->editName->clear();

ui->editEmail->clear();

ui->editSurname->clear();

ui->editPhoneNumber->clear();

}

void EditDialog::**setNewAccount**()

{

if(QString(ui->editName->text()).isEmpty()||QString(ui->editSurname->text()).isEmpty()

||QString(ui->editEmail->text()).isEmpty()

||QString(ui->editPhoneNumber->text()).isEmpty())

QMessageBox::warning(this,"Ошибка!","Некоторые поля не были заполнены!",QMessageBox::Ok);

else

{

UserAccount temp = UserAccount(ui->editName->text(),ui->editSurname->text(),

ui->editEmail->text(),ui->editPhoneNumber->text(),number,photo);

newUser = temp;

emit accountGetted();

}

}

UserAccount EditDialog::**setAccount**()

{

return newUser;

}

EditDialog::~***EditDialog***()

{

delete ui;

}

Класс RemoveDialog

Файл заголовка(.h)

#ifndef REMOVEDIALOG\_H

#define REMOVEDIALOG\_H

#include <QDialog>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QMessageBox>

#define admin\_id 2

namespace **Ui** {

class **RemoveDialog**;

}

class **RemoveDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **RemoveDialog**(QWidget \*parent = nullptr);

~***RemoveDialog***();

void **getInfo**(QSqlDatabase base,uint id);

signals:

void **getted**();

public slots:

void **deleteRecord**();

void **clear**();

private:

uint id;

QSqlDatabase base;

Ui::RemoveDialog \*ui;

};

#endif // REMOVEDIALOG\_H

Файл реализации(.cpp)

#include "removedialog.h"

#include "ui\_removedialog.h"

RemoveDialog::**RemoveDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::RemoveDialog)

{

ui->setupUi(this);

connect(ui->buttonBox,SIGNAL(accepted()),this,SLOT(deleteRecord()),Qt::UniqueConnection);

connect(ui->buttonBox,SIGNAL(rejected()),this,SLOT(clear()),Qt::UniqueConnection);

}

void RemoveDialog::**getInfo**(QSqlDatabase base,uint id)

{

this->base = base;

this->id = id;

}

void RemoveDialog::**clear**()

{

ui->editRemove->clear();

}

void RemoveDialog::**deleteRecord**()

{

QString text = ui->editRemove->text();

if(text.isEmpty())

QMessageBox::warning(this,"Ошибка","Поле не было заполнено!",QMessageBox::Ok);

else

{

QSqlQuery query = QSqlQuery(base);

if(id != admin\_id)

query.exec("DELETE FROM talons WHERE talon\_number = '" + text + "' AND patient = " + QString::number(id));

else

query.exec("DELETE FROM talons WHERE talon\_number = '" + text + "'");

ui->editRemove->clear();

emit getted();

}

}

RemoveDialog::~***RemoveDialog***()

{

delete ui;

}

Класс UserAccount

Файл заголовка(.h)

#ifndef USERACCOUNT\_H

#define USERACCOUNT\_H

#include <QString>

class **UserAccount**

{

QString name;

QString surname;

QString email;

QString phoneNumber;

uint idNumber;

QString photo;

public:

**UserAccount**(QString name, QString surname, QString email, QString phoneNumber, uint idNumber,QString photo)

{

this->name = name;

this->surname = surname;

this->email = email;

this->phoneNumber = phoneNumber;

this->idNumber = idNumber;

if(photo.isEmpty())

this->photo = "G:/3 term/C programs/Coursework/CoursWork/БД/Unknown.png";

else

this->photo = photo;

}

**UserAccount**()

{

name = "Неизвестно";

surname = "Неизвестно";

email = "-";

phoneNumber = "-";

idNumber = 0;

photo = "G:/3 term/C programs/Coursework/CoursWork/БД/Unknown.png";

}

void operator = (UserAccount& account2);

QString **GetName**();

QString **GetSurname**();

QString **GetEmail**();

QString **GetPhoneNumber**();

uint **GetIdNumber**();

QString **GetPhotoPath**();

void **SetName**(QString name);

void **SetSurname**(QString surname);

void **SetEmail**(QString email);

void **SetPhoneNumber**(QString phoneNumber);

void **SetIdNumber**(uint idNumber);

void **SetPhotoPath**(QString photoPath);

};

#endif // USERACCOUNT\_H

Файл реализации(.cpp)

#include "useraccount.h"

QString UserAccount::**GetName**()

{

return name;

}

QString UserAccount::**GetSurname**()

{

return surname;

}

QString UserAccount::**GetEmail**()

{

return email;

}

QString UserAccount::**GetPhoneNumber**()

{

return phoneNumber;

}

QString UserAccount::**GetPhotoPath**()

{

return photo;

}

uint UserAccount::**GetIdNumber**()

{

return idNumber;

}

void UserAccount::**SetName**(QString name)

{

this->name = name;

}

void UserAccount::**SetSurname**(QString surname)

{

this->surname = surname;

}

void UserAccount::**SetEmail**(QString email)

{

this->email = email;

}

void UserAccount::**SetPhoneNumber**(QString phoneNumber)

{

this->phoneNumber = phoneNumber;

}

void UserAccount::**SetIdNumber**(uint idNumber)

{

this->idNumber = idNumber;

}

void UserAccount::**SetPhotoPath**(QString photoPath)

{

this->photo = photoPath;

}

void UserAccount::operator = (UserAccount &account2)

{

this->SetName(account2.GetName());

this->SetEmail(account2.GetEmail());

this->SetSurname(account2.GetSurname());

this->SetIdNumber(account2.GetIdNumber());

this->SetPhotoPath(account2.GetPhotoPath());

this->SetPhoneNumber(account2.GetPhoneNumber());

}

Класс ClinicInfoDialog

Файл заголовка(.h)

#ifndef CLINICINFODIALOG\_H

#define CLINICINFODIALOG\_H

#include <QDialog>

#include <QPixmap>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QMessageBox>

namespace **Ui** {

class **ClinicInfoDialog**;

}

class **ClinicInfoDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **ClinicInfoDialog**(QWidget \*parent = nullptr);

~***ClinicInfoDialog***();

void **getDataBase**(QSqlDatabase base);

private slots:

void **on\_doctorSelect\_currentIndexChanged**();

private:

QSqlDatabase base;

Ui::ClinicInfoDialog \*ui;

};

#endif // CLINICINFODIALOG\_H

Файл реализации(.cpp)

#include "clinicinfodialog.h"

#include "ui\_clinicinfodialog.h"

ClinicInfoDialog::**ClinicInfoDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::ClinicInfoDialog)

{

ui->setupUi(this);

QPixmap pixMap = QPixmap("G:/3 term/C programs/Coursework/CoursWork/БД/Clinic.png");

int width = ui->clinicPic->width();

int height = ui->clinicPic->height();

ui->clinicPic->setPixmap(pixMap.scaled(width,height,Qt::KeepAspectRatio));

ui->plainEditInfo->setPlainText("Больница Принстон-Плейнсборо – это некоммерческая организация,""работающая на благотворительные средства,"

"которая является филиалом медицинского университета,"

"отвечающая за практику студентов-медиков, интернов. \n\n"

"Больница включает в себя несколько отделений, лекционный зал,"

"лаборатории, в которых можно сделать любые анализы."

"А также бесплатную поликлинику, где каждый житель штата Нью-Джерси"

"может получить бесплатное лечение при наличии" "страховки.");

}

void ClinicInfoDialog::**getDataBase**(QSqlDatabase base)

{

this->base = base;

QSqlQuery query = QSqlQuery(base);

query.exec("SELECT \* FROM doctors");

while(query.next())

{

ui->doctorSelect->addItem(query.value(3).toString() + "(" + query.value(4).toString() + ")");

}

}

ClinicInfoDialog::~***ClinicInfoDialog***()

{

delete ui;

}

void ClinicInfoDialog::**on\_doctorSelect\_currentIndexChanged**()

{

QSqlQuery query = QSqlQuery(base);

QString text = ui->doctorSelect->currentText();

text = text.left(text.indexOf("("));

query.exec("SELECT \* FROM doctors WHERE name = '" + text + "'");

if(query.next())

{

QPixmap pixMap(query.value(1).toString());

int width = ui->doctorPhoto->width();

int height = ui->doctorPhoto->height();

ui->doctorPhoto->setPixmap(pixMap.scaled(width,height,Qt::KeepAspectRatio));

QString text = query.value(2).toString();

text.replace("|","\n");

ui->editDoctorDescription->setPlainText(text);

}

}

Класс DateDialog

Файл заголовка(.h)

#ifndef DATEDIALOG\_H

#define DATEDIALOG\_H

#include <QDialog>

#include <QDateTime>

#include <QMessageBox>

namespace **Ui** {

class **DateDialog**;

}

class **DateDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **DateDialog**(QWidget \*parent = nullptr);

~***DateDialog***();

public slots:

QDate **getDate**();

private slots:

void **on\_buttonEnter\_clicked**();

signals:

void **selected**();

private:

Ui::DateDialog \*ui;

QDate currentDate;

};

#endif // DATEDIALOG\_H

Файл реализации(.cpp)

#include "datedialog.h"

#include "ui\_datedialog.h"

DateDialog::**DateDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DateDialog)

{

ui->setupUi(this);

ui->calendarWidget->setMinimumDate(QDate::currentDate());

}

QDate DateDialog::**getDate**()

{

return currentDate;

}

DateDialog::~***DateDialog***()

{

delete ui;

}

void DateDialog::**on\_buttonEnter\_clicked**()

{

currentDate = ui->calendarWidget->selectedDate();

if((currentDate.toString("ddd") == "Сб")||(currentDate.toString("ddd") == "Вс"))

{

QMessageBox::warning(this,"Ошибка","Вы выбрали выходной! Выберите, пожалуйста, другой день недели.",QMessageBox::Ok);

}

else

{

emit selected();

this->close();

}

}

Класс PasswordEditDialog

Файл заголовка(.h)

#ifndef PASSWORDEDITDIALOG\_H

#define PASSWORDEDITDIALOG\_H

#include <QDialog>

#include <QMessageBox>

namespace **Ui** {

class **PasswordEditDialog**;

}

class **PasswordEditDialog** : public QDialog

{

Q\_OBJECT

public:

explicit **PasswordEditDialog**(QWidget \*parent = nullptr);

~***PasswordEditDialog***();

void **getOldPassword**(QString password);

public slots:

void **clear**();

QString **getNewPassword**();

private slots:

void **checkPassword**();

signals:

void **setPassword**();

private:

QString oldPassword;

Ui::PasswordEditDialog \*ui;

};

#endif // PASSWORDEDITDIALOG\_H

Файл реализации(.cpp)

#include "passwordeditdialog.h"

#include "ui\_passwordeditdialog.h"

PasswordEditDialog::**PasswordEditDialog**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::PasswordEditDialog)

{

ui->setupUi(this);

connect(ui->buttonReject,SIGNAL(clicked()),this,SLOT(clear()),Qt::UniqueConnection);

connect(ui->buttonOk,SIGNAL(clicked()),this,SLOT(checkPassword()),Qt::UniqueConnection);

}

void PasswordEditDialog::**getOldPassword**(QString password)

{

oldPassword = password;

}

QString PasswordEditDialog::**getNewPassword**()

{

QString newPas = ui->editNewPassword->text();

ui->editNewPassword->clear();

ui->editOldPassword->clear();

return newPas;

}

void PasswordEditDialog::**clear**()

{

ui->editNewPassword->clear();

ui->editOldPassword->clear();

this->close();

}

void PasswordEditDialog::**checkPassword**()

{

if((QString(ui->editNewPassword->text()).isEmpty())||(QString(ui->editOldPassword->text()).isEmpty()))

QMessageBox::warning(this,"Ошибка","Некоторые поля не были заполнены!",QMessageBox::Ok);

else

{

if(oldPassword != ui->editOldPassword->text())

QMessageBox::warning(this,"Ошибка","Неверно введен старый пароль!",QMessageBox::Ok);

else

{

emit setPassword();

this->close();

}

}

}

PasswordEditDialog::~***PasswordEditDialog***()

{

delete ui;

}