Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
 высшего образования Московский авиационный институт

(национальный исследовательский университет)

Филиал «РКТ» МАИ в г. Химки Московской области

**ОТЧЁТ ПО УЧЕБНОЙ ПРАКТИКЕ**

Специальность 09.02.07 — Информационные системы и программирование

**Выполнил:**

Студент группы ИСП-41-19

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Серегин С.С./

Проверил**:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Гуров В.В./

Химки, 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc127793485)

[1. Проектирование Базы Данных и создание UML-диаграммы 4](#_Toc127793486)

[1.1. Концептуальная модель 4](#_Toc127793487)

[1.2. Логическая модель 7](#_Toc127793488)

[1.3. Диаграмма вариантов использования 8](#_Toc127793489)

[2. Разработка и настройка Базы Данных. 11](#_Toc127793490)

[3. Разработка форм для приложения 13](#_Toc127793491)

[4. Разработка приложения 15](#_Toc127793492)

[4 Создание исполняемого файла .exe и инсталлятора 18](#_Toc127793493)

[ЗАКЛЮЧЕНИЕ 22](#_Toc127793494)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc127793495)

[ПРИЛОЖЕННИЕ A. Запрос на создание БД с таблицами 24](#_Toc127793496)

[ПРИЛОЖЕННИЕ Б. Исходный код приложения 25](#_Toc127793497)

# **ВВЕДЕНИЕ**

Учебная практика является важной частью учебного процесса подготовки высококвалифицированных специалистов. В период прохождения практики студенты конкретизируют и закрепляют теоретические знания, приобретённые ими в процессе обучения, овладевают навыками практической работы, применяют на практике полученные знания. Практика является завершающим этапом в процессе подготовки специалиста к самостоятельной производственной деятельности.

Целью практики является закрепление и конкретизация теоретических знаний, практическая реализация desktop-приложения для работы с объектами базы данных.

Основными задачами производственной практикой является подготовка к разработке программного продукта, представляющего собой desktop-приложения для информационной системы горнолыжного курорта «Игора».

# **Анализ предметной области и построение диаграмм**

* 1. **Концептуальная модель**

На первом этапе проектирования базы данных необходимо изучить предметную область (что будет отражена в базе данных), а именно определить количество сущностей (таблиц), их атрибуты и связи между ними.  Предметной областью, рассматриваемой на текущей практике, является курортная точка проката горнолыжного курорта «Игора».. Путём изучения предметной области, было определено, какие таблицы (сущности) будут составлять будущую базу данных. Она будет состоять из пяти таблиц: Service, Order, Worker, Client, History.

Таблица Service (Услуги):

1. Код услуги
2. Наименование услуги
3. Стоимость, руб/час

Таблица Order (Заказы):

1. Код заказа
2. Дата создания
3. Время заказа
4. Услуги
5. Статус
6. Дата закрытия
7. Время проката
8. Код услуги
9. Код клиента
10. Код сотрудника

Таблица Worker (Работники «Игоры»):

1. Код сотрудника
2. ФИО
3. Должность
4. Логин
5. Пароль
6. Последний вход
7. Тип входа

Таблица Client (Клиенты):

1. Код клиента
2. ФИО
3. Паспортные данные
4. Дата рождения
5. Адрес
6. Электронная почта
7. Пароль

Таблица History (История входа):

1. Время
2. Логин
3. Количество входов
4. Код сотрудника

Концептуальная модель хранилища данных представляет собой описание главных (основных) сущностей и отношений между ними. Концептуальная модель является отражением предметной области, в рамках которой планируется построение хранилища данных.

Цель этапа концептуального проектирования — создание концептуальной модели данных исходя из представлений пользователей о предметной области. Для ее достижения

выполняется ряд последовательных процедур:

1. Определение сущностей и их документирование. Каждой сущности присваивается осмысленное имя, понятное пользователям. Имена и описания сущностей заносятся в словарь данных. Если возможно, то устанавливается ожидаемое количество экземпляров каждой сущности;
2. Определение связей между сущностями и их документирование. Определяются только те связи между сущностями, которые необходимы для удовлетворения требований к проекту базы данных. Устанавливается тип каждой из них. Выявляется класс принадлежности сущностей. Связям присваиваются осмысленные имена, выраженные глаголами;
3. Развернутое описание каждой связи с указанием ее типа и класса принадлежности сущностей, участвующих в связи, заносится в словарь данных;
4. Создание ER-модели предметной области. Для представления сущностей и связей между ними используются ЕR-диаграммы. На их основе создается единый наглядный образ моделируемой предметной области –  ЕR-модель предметной области;
5. Определение атрибутов и их документирование. Выявляются все атрибуты, описывающие сущности созданной ЕR-модели. Каждому атрибуту присваивается осмысленное имя, понятное пользователям;
6. Определение значений атрибутов и их документирование. Для каждого атрибута сущности, участвующей в ЕR-модели, определяется набор допустимых значений и ему;
7. Определение первичных ключей для сущностей и их документирование. На этом шаге руководствуются определением первичного ключа — как атрибута или набора атрибутов сущности, позволяющего уникальным образом идентифицировать ее экземпляры. Сведения о первичных ключах помещаются в словарь данных;
8. Обсуждение концептуальной модели данных с конечными пользователями. Если будут обнаружены несоответствия предметной области, то в модель вносятся изменения до тех пор, пока пользователи не подтвердят, что предложенная им модель адекватно отображает их личные представления[2];

В текущей практической работе будет реализована концептуальная модель нотации Чена. Построить её можно с помощью веб-ресурса ERDPlus[4]. Сайт позволяет с помощью удобного и простого интерфейса создать ER-диаграмму для нашей базы данных. После авторизации, в раздел “DOCUMENTS” по нажатию на кнопку “CREATE DIAGRAM…” создаётся новая диаграмма. С помощью интерфейса редактора (приложение А), на виртуальный холст добавляются новые сущности и их атрибуты. С помощью кнопки “CONNECT” между сущностями строится связь. Результат проделанной работы представлен на рисунке 1.

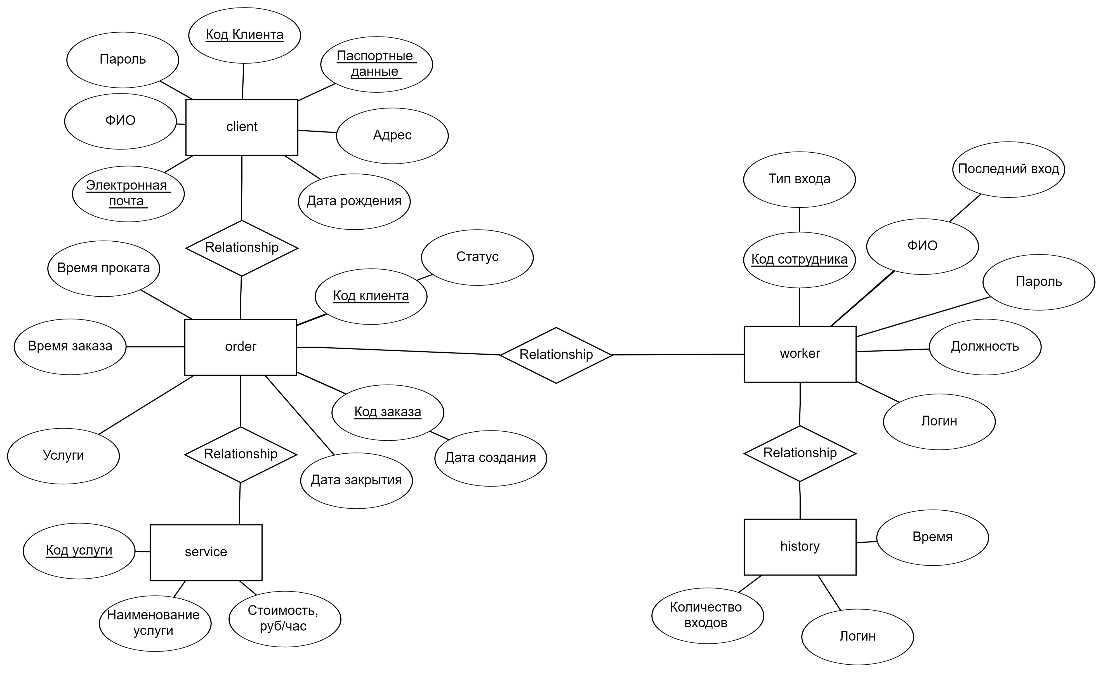


Рисунок 1 — ER-диаграмма.

* 1. **Логическая модель**

Логическая модель расширяет концептуальную, путём определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.

Цель этапа логического проектирования — преобразование концептуальной модели на основе выбранной модели данных в логическую модель, не зависимую от особенностей используемой в дальнейшем СУБД для физической реализации базы данных. Для ее достижения выполняются следующие процедуры:

1) Выбор модели данных. Чаще всего выбирается реляционная модель данных в связи с наглядностью табличного представления данных и удобства работы с ними;

2) Определение набора таблиц исходя из ЕR-модели и их документирование. Для каждой сущности ЕR-модели создается таблица. Имя сущности – имя таблицы;

3) Осуществляется формирование структуры. Устанавливаются связи между таблицами посредством механизма первичных и внешних ключей. Структуры таблиц и установленные связи между ними документируются;

4) Нормализация таблиц. На этом шаге он проверяет корректность структуры таблиц, созданных на предыдущем шаге, посредством применения к ним процедуры нормализации. В результате нормализации получается очень гибкий проект базы данных, позволяющий легко вносить в нее нужные расширения;

5) Проверка логической модели данных на предмет возможности выполнения всех транзакций, предусмотренных пользователями. Транзакция — это набор действий, выполняемых отдельным пользователем или прикладной программой с целью изменения содержимого базы данных;

6) Определение требований поддержки целостности данных и их документирование. На этом шаге вопросы целостности данных освещаются безотносительно к конкретным аспектам ее реализации;

7) Создание окончательного варианта логической модели данных и обсуждение его с пользователями[2];

Средствами всё того же ERDPlus можно создать не только ER-диаграмму, но ещё и реляционную схему. Построить схему можно как с нуля, так и на основе уже существующей ER-диаграммы. Во вкладке “DOCUMENTS” по нажатию на троеточие (справа от имени ER-диаграммы) станет доступен пункт “Convert to Relational Scheme”. По нажатию на этот пункт, создастся реляционная схема, представленная на рисунке 2.

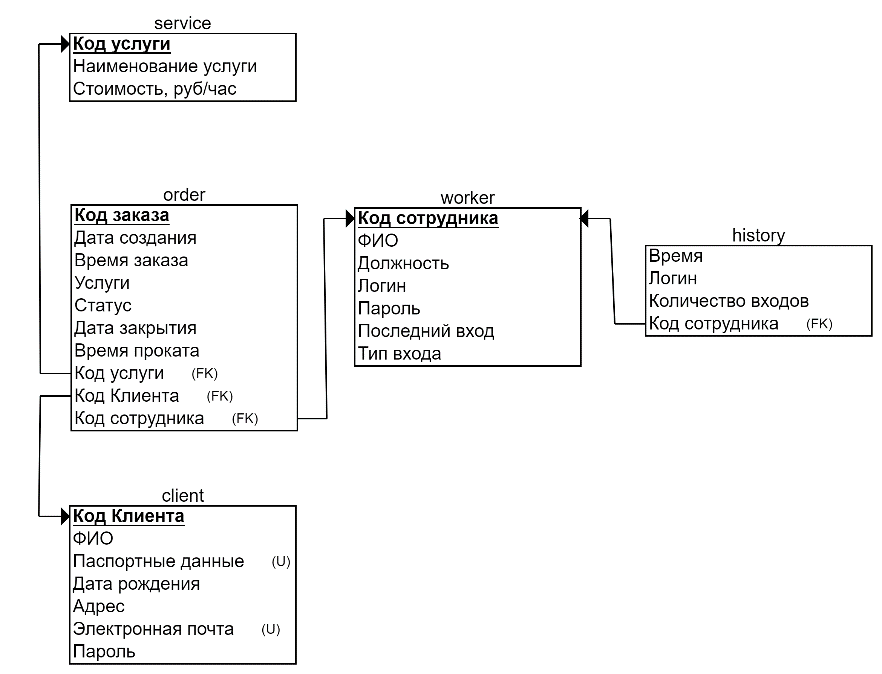


Рисунок 2 — Логическая модель.

* 1. **Диаграмма вариантов использования**

Unified Modeling Language (UML) — унифицированный язык моделирования. UML представляет собой набор лучших инженерных практик, которые доказали свою эффективность в моделировании больших и сложных систем и является очень важной частью разработки объектно-ориентированного программного обеспечения.

UML использует в основном графические обозначения, чтобы выразить дизайн программных проектов. Использование UML помогает проектным группам общаться, изучать потенциальные проекты и проверять архитектурный дизайн программного обеспечения. Цель UML — предоставить стандартную нотацию, которая может использоваться всеми объектно-ориентированными методами, а также выбрать и интегрировать лучшие элементы нотаций-предшественников. UML был разработан для широкого спектра приложений. Диаграмма вариантов использования (англ. Use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Для создания диаграммы следует воспользоваться инструментами веб-сервиса Lucidchart, представленного на рисунке 3. Ключевой особенность данного ресурса является возможность строить любые необходимые диаграммы и схемы без оформления платных подписок. Интерфейс сайта прост и понятен. Слева находится панель с объектами, а справа поле для рисования диаграм.

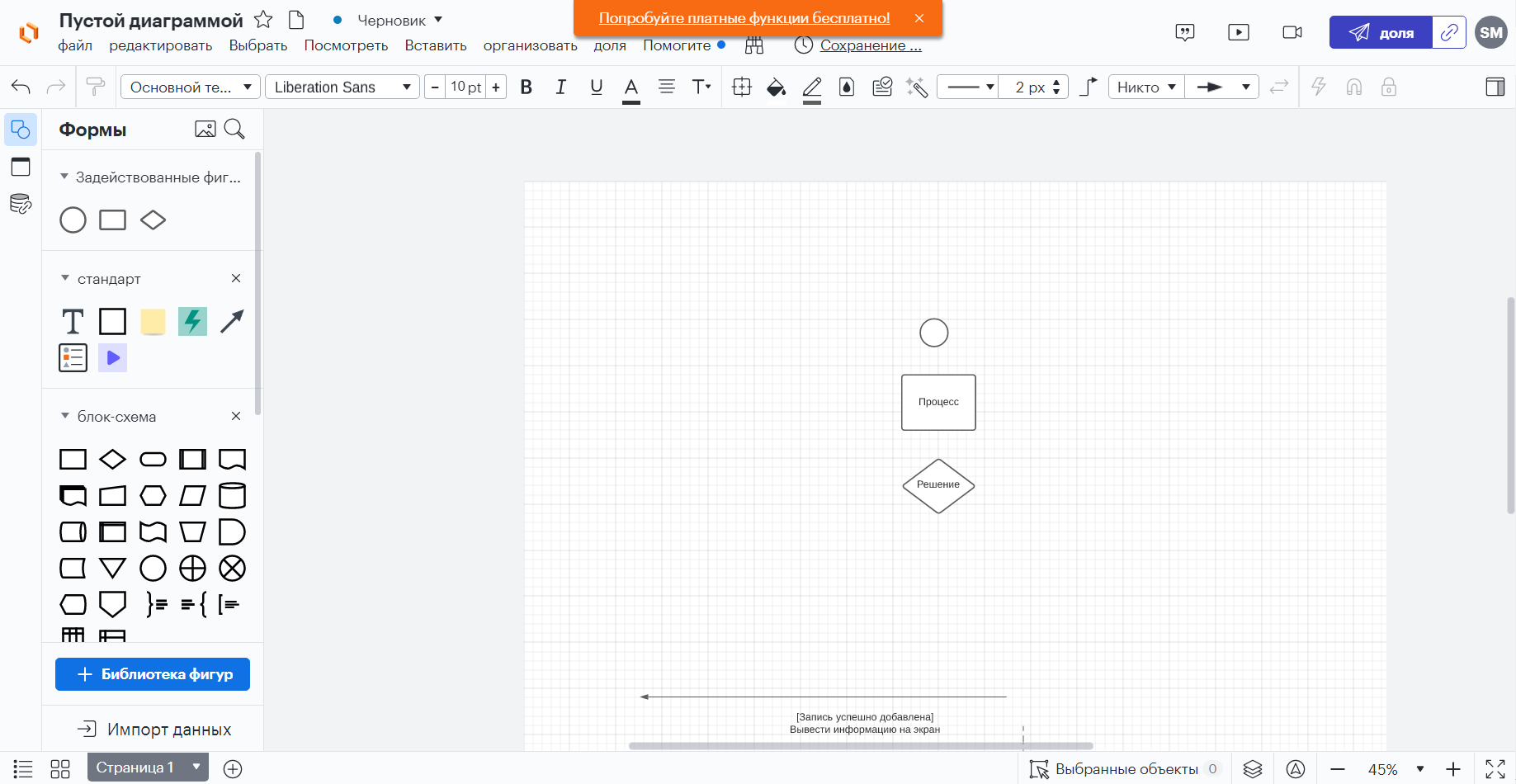


Рисунок 3 – Интерфейс Lucidchart.

Путём перетаскивания на поле редактора можно отразить все ключевые схемы и диаграммы. Помимо этого есть возможность соединить объекты между собой перетаскивая стрелок на промежуток между двумя объектами, либо посредством последовательного нажатия левого клика с одного объекта на другой.

В результате проделанных манипуляций была построена диаграмма вариантов использования, представленная на рисунке 4.

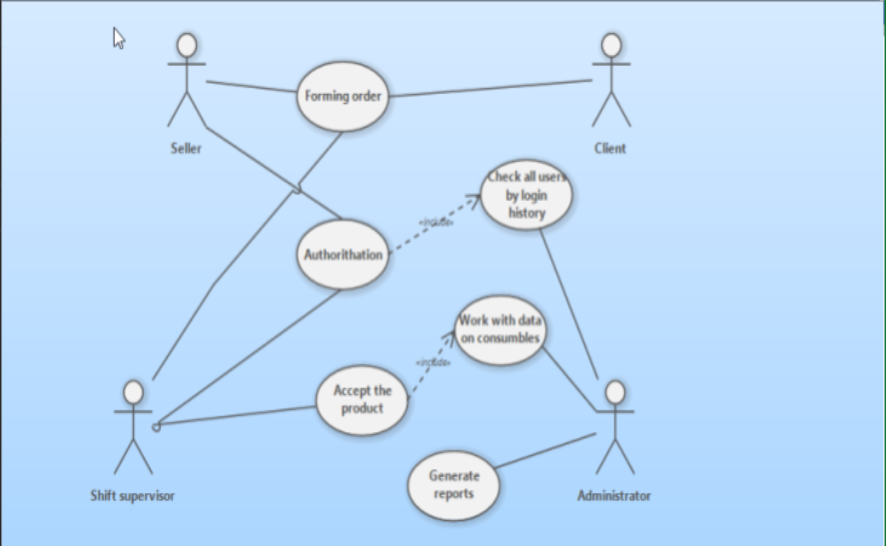


Рисунок 4 — Диаграмма вариантов использования.

1. **Разработка и настройка Базы Данных**.

В качестве СУБД выбор пал на SQLite Studio – инструмент для создания и работы с базами данными, написанными на диалекте sqlite языка SQL.

SQLite — это быстрая и легкая встраиваемая СУБД на языке C, которая не имеет сервера и позволяет хранить всю базу локально на одном устройстве. Для работы SQLite не нужны сторонние библиотеки или службы. Движок SQLite —не отдельно работающий процесс, с которым взаимодействует программа, а библиотека. Программа компонуется с ней, и движок служит составной частью программы. В качестве протокола обмена применяются вызовы функций (API) библиотеки SQLite. Для создания базы данных, используя СУБД SQLite Studio, необходимо провести следующие операции:

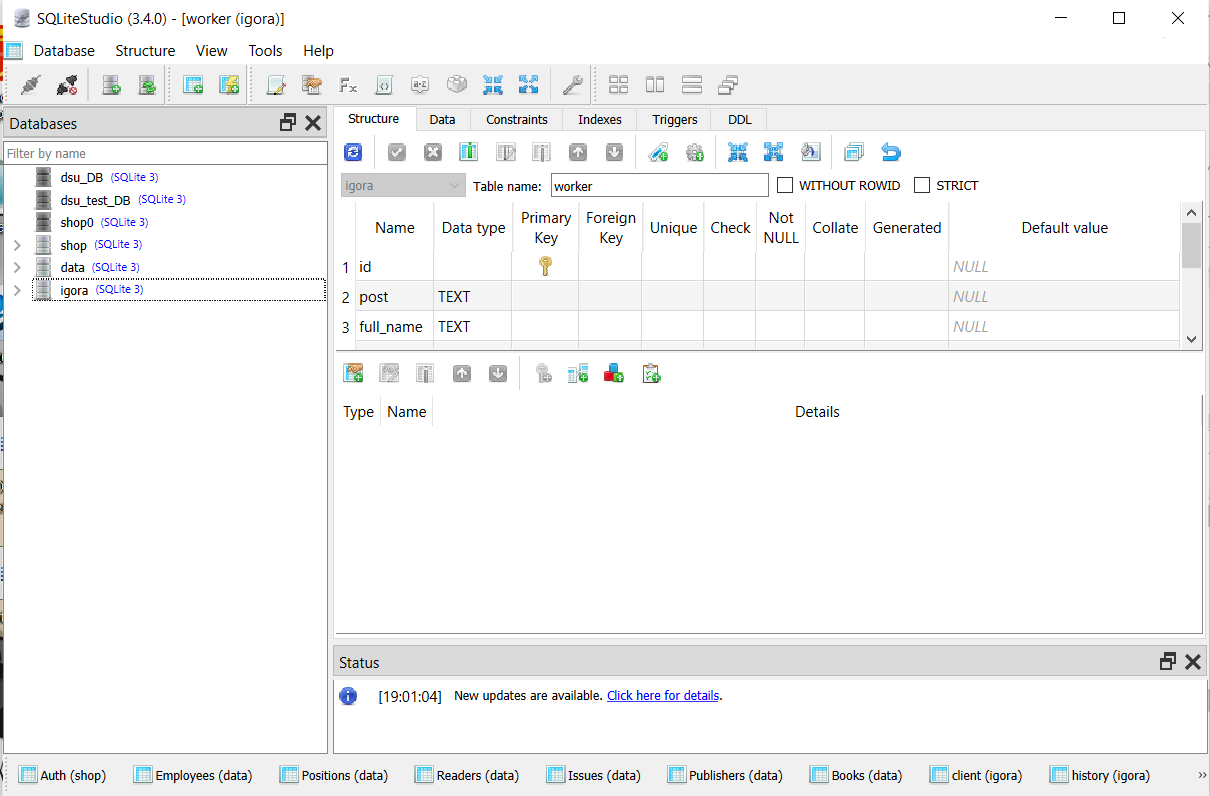
1. Скачать SQLite Studio с официального сайта: <https://sqlitestudio.pl/>
2. Дважды нажать ЛКМ по инсталлятору и произвести установку СУБД.
3. После установки необходимо отрыть приложение. Интерфейс СУБД представлен на рисунке 5.

Рисунок 5 – Интерфейс SQLite Studio.

1. После запуска СУБД необходимо нажать на значок «Create a database» на верхней панели
2. В окне, представленном на рисунке 6, указываем где будет храниться файл .db нашей будущей базы данных. После нужно нажать кнопку «ОК» и новая база данных появится в списке слева.
3. Следующим шагом следует нажатием ЛКП новосозданную БД раскрыть список содержимого. После нажать на пункт Tables ЛКМ и выбрать пункт Create table. А далее создать все нужные таблицы.

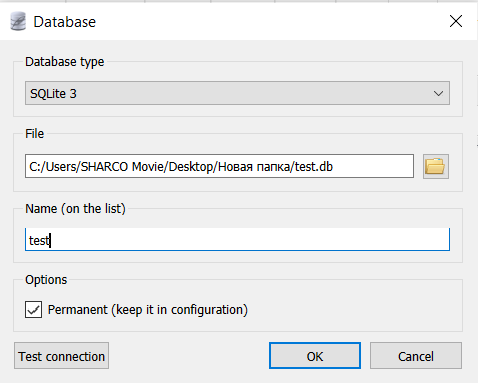


Рисунок 6 – Окно создания новой базы данных

1. По результатом проделанной работы получится изображённый на рисунке 7 набор таблиц необходимых для нашей программы.

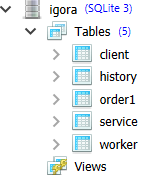


Рисунок 7 – Список таблиц базы данных.

# **Разработка форм для приложения**

Для построения пользовательского интерфейса приложения требуется создать некоторый набор связанных между собой форм. С помощью средств Qt Designer есть возможность разработать необходимую форму.

Qt Designer – инструмент Qt, предназначенный для проектирования и построения графических пользовательских интерфейсов (GUI) с виджетами Qt. Интерфейс приложения представлен на рисунке 8.

Инструкция при разработке форм в «Qt Designer»:

1) Для создания главного окна необходимо выбрать «Main Window», а для создания диалогового окна нужно выбрать «Dialog without Buttons»;

2) Для настройки отображения элементов UI и настройки элементов взаимодействия пользователя непосредственно с программой используются виджеты, представленные . Рассмотрим основные виджеты из списка:

* Vertical Layout – элемент, группирующий виджеты по вертикали;
* Horizontal Layout – элемент, группирующий виджеты по горизонтали;
* Grid Layout – элемент, группирующий виджеты по сетке;
* Form Layout – для элемент, группирующий виджеты;
* Horizontal Spacer – добавляет горизонтальное пространство между элементами формы;
* Vertical Spacer – необходим для создания вертикального отступа между элементами;
* Push Button представляет собой кнопка, с которой можно взаимодействовать. Есть возможность назначит на неё выполнение определённой команды;
* Table Widget – виджет, отображающий содержимое таблицы базы данных;
* StackedWidget предоставляет стек виджетов, предназначенный для размещения нескольких виджетов в пределе одного окна;
* LineEdit – это однострочный текстовый редактор, в который можно записать текст;
* Label – строка текста, выводимая на экран;
* GraphicsView предоставляет виджет для отображения содержимого QgraphicsScene;

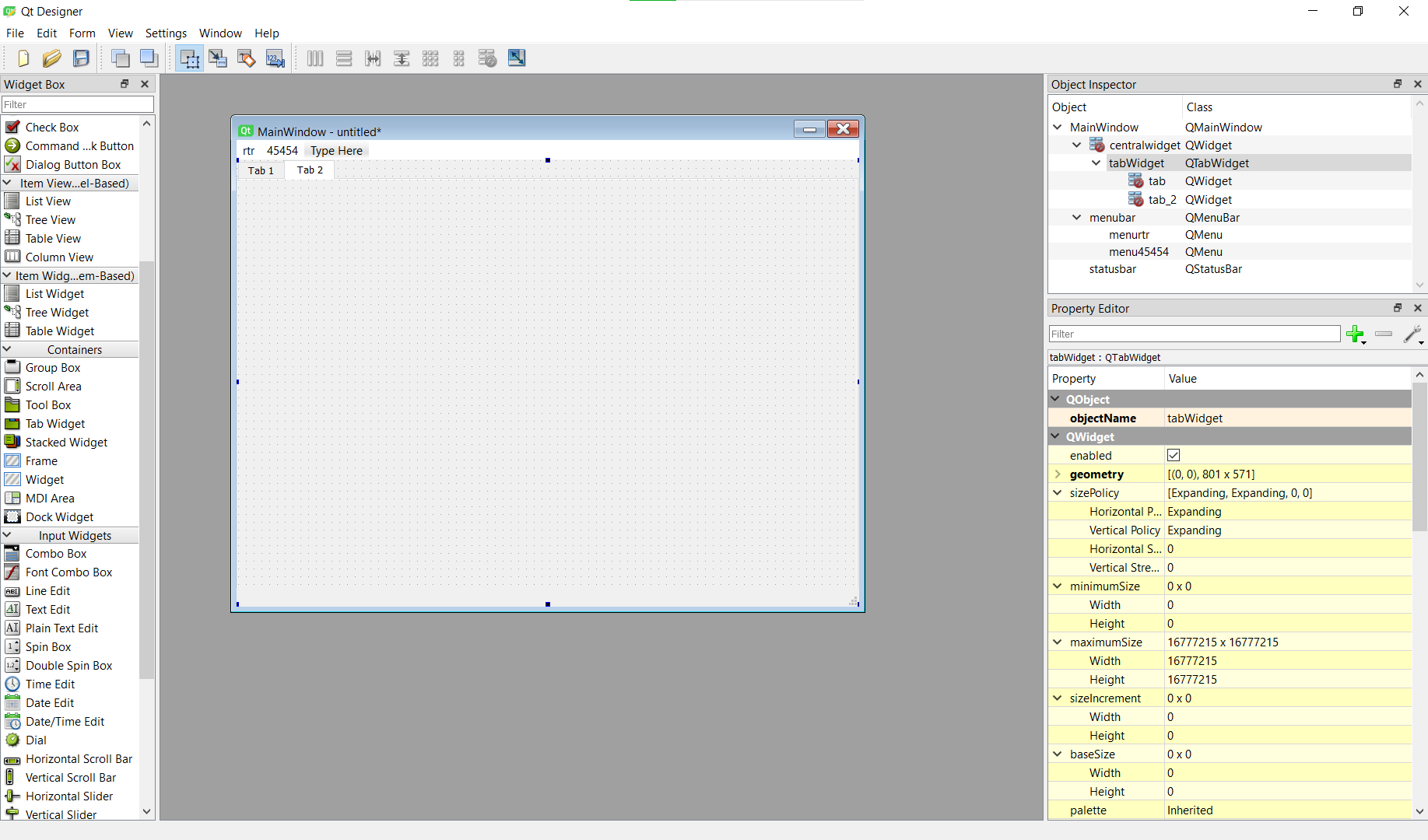


Рисунок 8 – Интерфейс Qt Designer.

3) Для группировки элементов необходимо разместить их в layout, а также указать layout для основного окна;

4) Для изменения названия виджета необходимо в правом меню дважды кликнуть на нужный, присвоив другое имя;

5) Для изменения цвета и шрифта виджета в том же меню справа выбрать пункт «Change Style Sheet…» и указать для выделенного объекта нужные параметры оформления в соответствии с требованиями в ТЗ:

* Использование шрифта Comic Sans MS;
* В качестве основного фона используется белый цвет, а в качестве дополнительного: RGB (118, 227, 131). Для акцентирования внимания пользователя на целевое действие интерфейса используйте цвет RGB (73, 140,81). Таблица цветов представлена на рисунке 9.

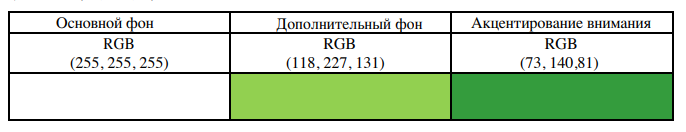


Рисунок 9 — Цветовая схема.

6) Был использован виджет «QtabWidget» и «Stacked Widget» для переключения между таблицами в пределах основного окна.

7) Для добавления изображения на форму необходимо добавить объект label и в пункте «pixmap» выбрать изображение;

# **Разработка приложения**

Для входа сотрудника в приложение ему необходимо войти с помощью окна авторизации в свою учётную запись, указав корректный логин и пароль. Для этого

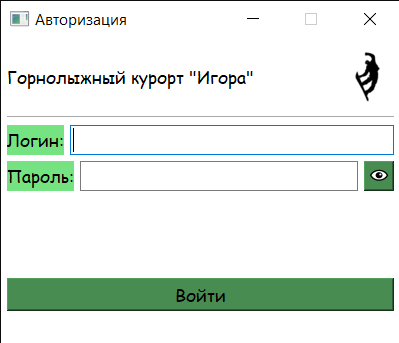


Рисунок 10 — Окно входа в приложении.

Класс «Auth» представляет собой форму авторизации и выполняет следующие функции:

1) отвечает за подключением кнопок формы авторизации, объявление переменных, создание сцены для «graphicsView».

2) проверяет есть ли указанный логин и пароль в базе данных или нет. В случае ввода неверных или несуществующих данных. Выводит на окно авторизации сообщение о некорректно введённых данных. При успешном входе присваивает виджету «Stacked Widget» нужные значения страниц. Этот метод позволяет разграничить доступ между пользователями к определённым таблицам в базе данных.

После входа пользователя в приложение, ему должна быть предоставлена возможность взаимодействовать с графическим интерфейсом. Для этого был реализован класс «MainWindow» и «DialogNewClient», с использованием библиотеки PyQt (Смотреть приложение Б).

Класс «MainWindow» является главным окном приложения, в нем осуществляется загрузка формы интерфейса и его основных элементов:

* окно для вывода информации на экран;
* вкладки для переключения между формами ввода данных;
* кнопку просмотра всех клиентов;
* кнопку для показа всех предоставляемых услуг;
* кнопка для показа списка заказов;
* кнопки добавления, сохранения заказа;
* кнопку добавления услуги к заказу.

Все данные загружаются в элемент «QtableWidget» и «QtableView»

Вкладки сверху полей ввода и таблиц осуществляют переход по страницам формы.

На странице, представленной на рисунке 11, отображается основную информации о сотруднике — его ФИО, должность и аватар профиля. Это страница доступна для всех пользователей информационной системы (ИС).

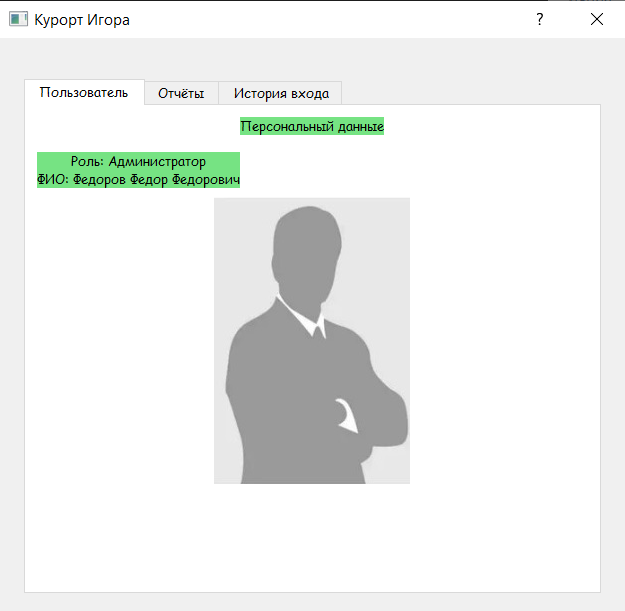


Рисунок 11 — Первая страница пользователя Администратор

Так как у нас есть три разных класса пользователей, а именно: продавец, старший смены и администратор ИС – для каждого доступны только специально отведённые страницы. Так, например, администратор помимо вкладки с его личной информацией, имеет возможность посмотреть отчёты о осуществляемых заказах и просмотреть историю входа каждого из пользователей системы.

Вкладки для продавца и старшего смены, представленные на рисунке 12, позволяют просмотреть личную информацию о себе, а во вкладке «Заказы» создать новый заказ, просмотреть список всех клиентов, просмотреть список всех оказываемых услуг, а также просмотреть список всех клиентов «Игоры».

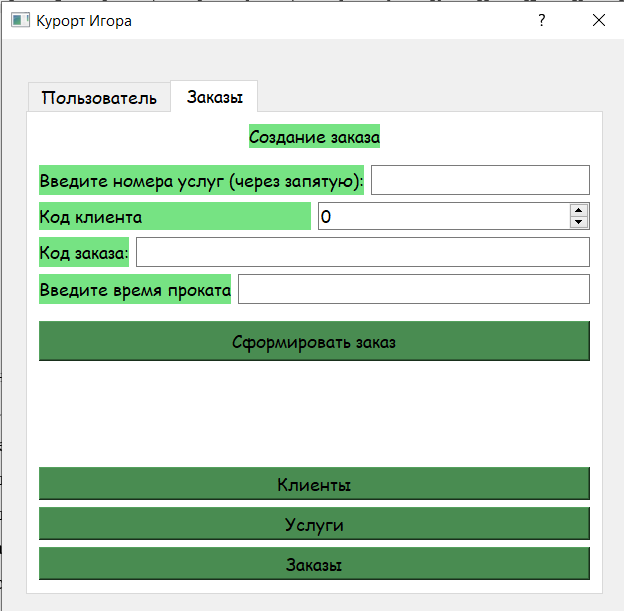


Рисунок 12 — Вкладка «Заказы» пользователя Продавец и Старший смены.

# **Создание исполняемого файла .exe и инсталлятора**

Язык Python не компилируемый язык, в нём не была предусмотрена возможность преобразования программ в исполняемый вид. Вместо этого, в языке используется интерпретатор, который переводит код языка в машинный и выполняет его на месте.

В теории, можно взять интерпретатор Python и объединить его с программой и её компонентами, получив портативную программу, которую можно запускать без установки языка. Скорее всего эта мысль и пришла в голову разработчикам пакета Pyinstaller, выполняющего все эти действия.

Одним из преимуществ этого пакета является то, что он собирает только необходимые зависимости, а не все подряд, поэтому приложение на выходе имеет меньший вес по сравнению с другими библиотеками.

PyInstaller собирает в один пакет Python-приложение (.exe) и все необходимые ему библиотеки следующим образом:

1. Считывает файл скрипта.
2. Анализирует код для выявления всех зависимостей, необходимых для работы.
3. Создает файл spec, который содержит название скрипта, библиотеки-зависимости, любые файлы, включая те параметры, которые были переданы в команду PyInstaller.
4. Собирает копии всех библиотек и файлов вместе с активным интерпретатором Python.
5. Создает папку BUILD в папке со скриптом и записывает логи вместе с рабочими файлами в BUILD.
6. Создает папку DIST в папке со скриптом, если она еще не существует.
7. Записывает все необходимые файлы вместе со скриптом или в одну папку, или в один исполняемый файл.

Pyinstaller поддерживает Python 3.3 и выше. Команда для установки pyinstaller через pip – «pip install pyinstaller».

Для создания исполняемого файла в командной строке надо ввести: pyinstaller –onefile название\_файла.py. Если же при открытии .exe файла не нужна консоль, выводящая дополнительную информацию о работе программы, тогда в команду надо добавить параметр -w: pyinstaller -w –onefile gui.py.

Для более комфортной работы, без необходимости прописывать команды в консоль, можно воспользоваться библиотекой «auto-exe-to-py». Для её установки нужно прописать в консоли команду «pip install auto-py-to-exe». После установки в консоль нужно написать команду «auto-py-to-exe», после чего появится окно, что представлено на рисунке 13, в котором нужно выбрать путь до файла .py и указать выходную директорию для выходного исполняемого файла .exe. После чего нажать кнопку «CONVERT .PY TO .EXE».

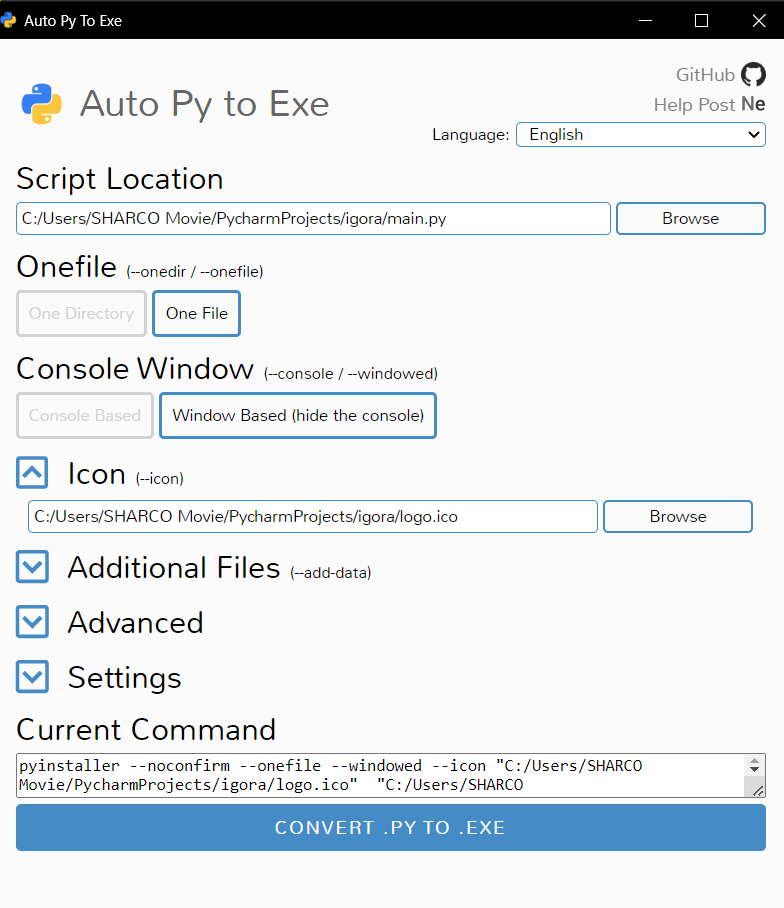


Рисунок 13 – Окно auto-py-to-exe.

В результате выполнения команды в папке, которая была указана для вывода будет находится исполняемый файл приложения.

После создания исполняемого файла .exe директория будет выглядеть как показано на рисунке 14. Далее нужно скопировать папки с формами и изображениями.

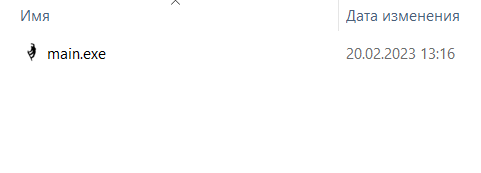


Рисунок 14 – Папка с исполняемым файлом

Следующим шагом необходимо создать инсталлятор нашего приложения. Для этого нужно воспользоваться программой Inno Setup Compiler. После запуска программы в появившимся окне, представленном на рисунке 15, выбираем пункт «Create a new script file using the Script Wizard». После чего нажать «OK» и «Next».

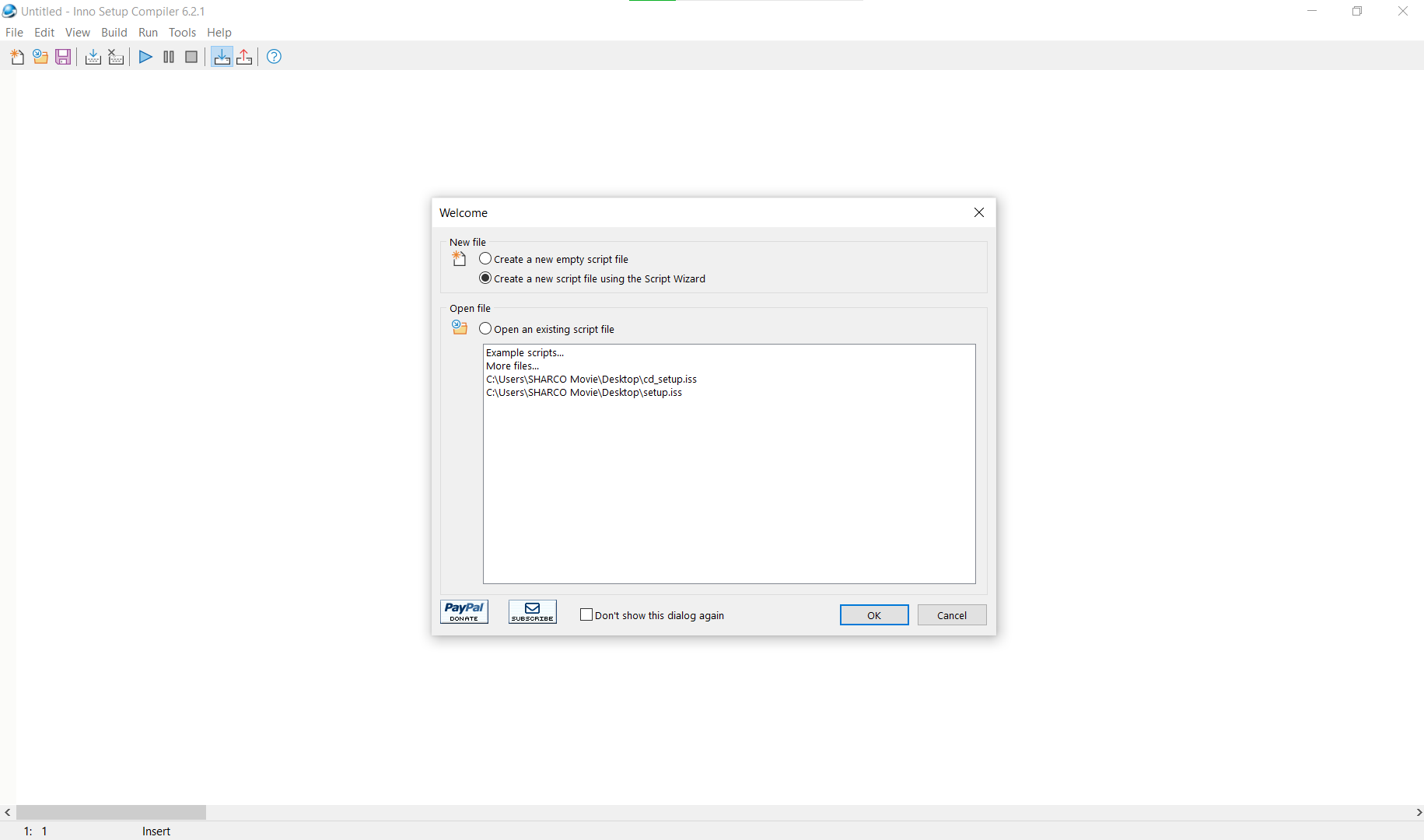


Рисунок 15 – Окно создания нового скрипта.

Следующим шагом необходимо заполнить информацию о нашей программе и нажать дважды «Next». После нужно указать путь до исполняемого файла и путь до папки с файлами программы, как показано на рисунке 16 и нажать «Next», убрать галочку сверху и нажать три раза «Next». После в открывшимся окне выбираем пункт «Non administrative mode» и нажимаем «Next». Далее следует выбрать язык инсталлятора, после чего нажать «Next».

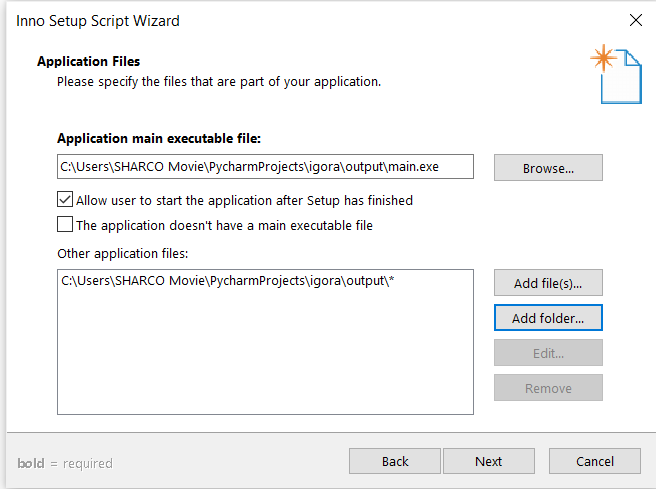


Рисунок 16 – Настройка инсталлятора.

В следующем окне указываем папку (в которой создаться итоговый файл инсталлятора) и иконку как показано на рисунке 17.

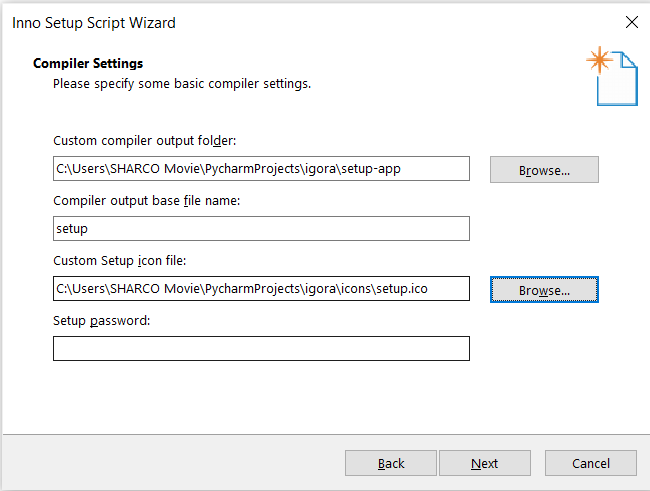


Рисунок 17 – Окно настройки

После чего необходимо нажать дважды кнопку «Next» и «Finish». В появившимся окне нужно подтвердить исполнение сгенерируемого кода и нажать «Да», и дождаться компиляции. В результате исполнения скрипта будет создан инсталлятор программы, показанный на рисунке 18.

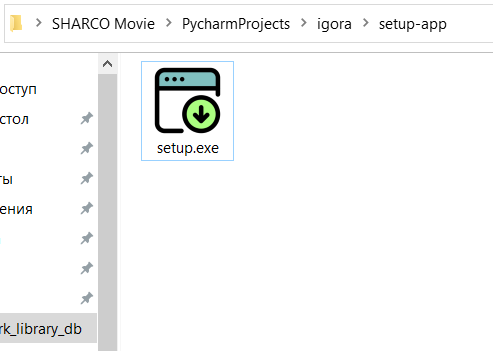


Рисунок 18 – Созданный инсталлятор приложения.

# **ЗАКЛЮЧЕНИЕ**

В ходе прохождения учебной практики в филиале «РКТ» МАИ была достигнута её цель и решены поставленные задачи:

1) Изучена литература, посвящённая теме разработки настольного приложения, а также проектированию, администрированию базы данных.

2) Спроектирована база данных в соответствии с рассматриваемой предметной областью.

3) Разработаны программные модули предназначенные для взаимодействия с данными:

* Модуль графический интерфейс, в соответствии с требованиями технического задания, предоставляющий возможность просмотра данных и возможность взаимодействия с ними.
* Модуль взаимодействие с базой данных посредством SQL-запросов.

4) Закреплены необходимые навыки работы в среде разработки PyCharm.

5) Были получены навыки работы с хранилищем репозиториев Gogs.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 7.32-2017 отчёт о научно-исследовательской работе. М., Стандартинформ, 2017.
2. Лекция на тему «Проектирование Базы Данных». – URL: http://bseu.by/it/tohod/lekcii4\_6.htm (дата обращения 12-02-2023)
3. Python: Базы данных. — URL: <https://www.cyberforum.ru/python-db/> (дата обращения 13-02-2023)
4. Ресурс ERDplus – URL: https://erdplus.com (дата обращения 12-02-2023)
5. Алгоритмы: построение и анализ, 3-е издание: пер. с англ.: учебное пособие / Кормен Томас Х., Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн; Москва, Санкт-Петербург, Киев: ООО “И. Д. Вильямс”, 2013. – 1238 с.
6. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений. — СПб.: БХВ-Петербург, 2012. — 704 с.: ил.

7. Изучаем программирование на Python / Пол Бэрри; [пер. с англ. М. А. Райтман]. – Москва: Издательство «Э», 2017. – 624 с.

# **ПРИЛОЖЕННИЕ A. Запрос на создание БД с таблицами**

PRAGMA foreign\_keys = off;

BEGIN TRANSACTION;

CREATE TABLE IF NOT EXISTS client (ФИО TEXT, "Код клиента" TEXT, "Паспортные данные" TEXT, "Дата рождения" TEXT, Адрес TEXT, "e-mail" TEXT, password TEXT);

CREATE TABLE IF NOT EXISTS history (time TEXT, login TEXT, try\_enter BOOLEAN);

CREATE TABLE IF NOT EXISTS order1 (ID INTEGER PRIMARY KEY AUTOINCREMENT, "Код заказа" TEXT, "Дата создания" TEXT, "Время заказа" TEXT, "Код клиента" INTEGER, Услуги TEXT, Статус TEXT, "Дата закрытия" TEXT, "Время проката" TEXT);

CREATE TABLE IF NOT EXISTS service (ID INTEGER, "Наименование услуги" TEXT, "Код услуги" TEXT, "Стоимость, руб. за час " INTEGER);

CREATE TABLE IF NOT EXISTS worker (id PRIMARY KEY, post TEXT, full\_name TEXT, login TEXT, password TEXT, last\_enter TEXT, type\_enter TEXT);

COMMIT TRANSACTION;

PRAGMA foreign\_keys = on;

**ПРИЛОЖЕННИЕ Б. Исходный код приложения**

import sqlite3

import sys

from datetime import datetime

from PyQt5 import uic

from PyQt5.QtSql import QSqlDatabase, QSqlQueryModel

from PyQt5.QtWidgets import \*

from PyQt5.QtWidgets import QLineEdit

class Auth(QMainWindow):

def \_\_init\_\_(self, parent = None):

super().\_\_init\_\_(parent)

self.DB = DataBase()

self.ui = uic.loadUi('forms/auth.ui', self)

self.ui.show()

self.btn\_enter.clicked.connect(self.auth)

self.btn\_pas.clicked.connect(self.hide\_pas)

self.hide\_password = True

def hide\_pas(self):

self.password = self.ui.edit\_password

if self.hide\_password:

self.password.setEchoMode(QLineEdit.Normal)

self.hide\_password = False

else:

self.password.setEchoMode(QLineEdit.Password)

self.hide\_password = True

def auth(self):

log = self.ui.edit\_login.text()

password = self.ui.edit\_password.text()

data = self.DB.get\_auth\_info(log, password) # если неправильные данные, то вернет False

if data:

self.DB.add\_entry(datetime.now().strftime('%d.%m.%y %H:%M'), log, True)

self.ui.hide()

post, full\_name = data[0]

main\_win = Window(post, full\_name)

main\_win.setWindowTitle('Курорт Игора')

main\_win.exec()

else:

self.error.setStyleSheet("color:red") # Изменение цвета шрифта на зелёный

self.error.setText('Ошибка входа')

self.DB.add\_entry(datetime.now().strftime('%d.%m.%y %H:%M'), log, False)

class Clients(QDialog):

def \_\_init\_\_(self, parent = None):

super().\_\_init\_\_(parent)

self.ui = uic.loadUi('forms/table\_client.ui', self)

con = QSqlDatabase.addDatabase("QSQLITE")

con.setDatabaseName("igora.db")

con.open()

self.model = QSqlQueryModel() # модель (таблица) без редактирования данных

self.model.setQuery("SELECT \* FROM client")

self.tableView.setModel(self.model)

self.tableView.horizontalHeader().setStretchLastSection(True)

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

class Services(QDialog):

def \_\_init\_\_(self, parent = None):

super().\_\_init\_\_(parent)

self.ui = uic.loadUi('forms/table\_serv.ui', self)

con = QSqlDatabase.addDatabase("QSQLITE")

con.setDatabaseName("igora.db")

con.open()

self.model = QSqlQueryModel() # модель (таблица) без редактирования данных

self.model.setQuery("SELECT \* FROM service")

self.tableView.setModel(self.model)

self.tableView.horizontalHeader().setStretchLastSection(True)

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

class Order(QDialog):

def \_\_init\_\_(self, parent = None):

super().\_\_init\_\_(parent)

self.ui = uic.loadUi('forms/table\_ord.ui', self)

con = QSqlDatabase.addDatabase("QSQLITE")

con.setDatabaseName("igora.db")

con.open()

self.model = QSqlQueryModel() # модель (таблица) без редактирования данных

self.model.setQuery("SELECT \* FROM order1")

self.tableView.setModel(self.model)

self.tableView.horizontalHeader().setStretchLastSection(True)

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

class Window(QDialog):

def \_\_init\_\_(self, post, fullName, parent = None):

super().\_\_init\_\_(parent)

self.DB = DataBase()

self.ui = uic.loadUi('forms/main.ui', self)

self.ui.lbl\_role.setText('Роль: ' + post + '\n' + 'ФИО: ' + fullName)

self.ui.btn\_clients.clicked.connect(self.clients)

self.ui.btn\_serv.clicked.connect(self.sevices)

self.ui.btn\_add\_order.clicked.connect(self.add\_order)

self.ui.btn\_ord.clicked.connect(self.orders)

self.ui.btn\_exit.clicked.connect(self.exit)

self.ui.btn\_service.clicked.connect(self.report\_service)

self.ui.btn\_order.clicked.connect(self.report\_order)

if post == 'Продавец' or post == 'Старший смены':

self.ui.lbl\_role.setText('Роль: ' + post + '\n' + 'ФИО: ' + fullName)

self.ui.stackedWidget.setCurrentIndex(0)

elif post == 'Администратор':

self.ui.lbl\_role2.setText('Роль: ' + post + '\n' + 'ФИО: ' + fullName)

self.ui.stackedWidget.setCurrentIndex(1)

self.update\_table\_history()

def exit(self):

self.close()

auth = Auth()

auth.exec()

def report\_service(self):

s\_date = self.ui.start\_date.text().split('.')[::-1]

e\_date = self.ui.end\_date.text().split('.')[::-1]

data = self.DB.get\_order()

new\_data = {}

for d in data:

if d[2].split('.')[-1::-1] <= e\_date and d[2].split('.')[-1::-1] >= s\_date:

try:

new\_data[d[2]] = str(int(new\_data[d[2]])+len(d[5].split(',')))

except Exception:

new\_data[d[2]] = str(len(d[5].split(',')))

self.table\_update(list(new\_data.items()), ['Дата', 'Кол-во заказов'])

def report\_order(self):

s\_date = self.ui.start\_date.text().split('.')[::-1]

e\_date = self.ui.end\_date.text().split('.')[::-1]

data = self.DB.get\_order()

new\_data = {}

for d in data:

if d[2].split('.')[-1::-1] <= e\_date and d[2].split('.')[-1::-1] >= s\_date:

try:

new\_data[d[2]] = str(int(new\_data[d[2]])+1)

except Exception:

new\_data[d[2]] = '1'

self.table\_update(list(new\_data.items()), ['Дата', 'Кол-во оказанных услуг'])

def table\_update(self, data, titels):

numrows = len(data)

numcols = len(titels)

self.ui.tableWidget.setColumnCount(numcols)

self.ui.tableWidget.setRowCount(numrows)

self.ui.tableWidget.setHorizontalHeaderLabels(titels)

for row in range(numrows):

for column in range(numcols):

self.ui.tableWidget.setItem(row, column, QTableWidgetItem((data[row][column])))

self.ui.tableWidget.horizontalHeader().setStretchLastSection(True)

self.ui.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

def update\_table\_history(self):

con = QSqlDatabase.addDatabase('QSQLITE')

con.setDatabaseName('igora.db')

con.open()

model = QSqlQueryModel() # без редактирования

model.setQuery('SELECT \* FROM history')

self.ui.tableView.setModel(model)

self.ui.tableView.horizontalHeader().setStretchLastSection(True)

self.ui.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

def add\_order(self):

service = self.ui.edit\_service.text()

client = self.ui.edit\_client.value()

order = self.ui.edit\_order.text()

time = self.ui.edit\_time.text()

self.DB.add\_order(service,client,order,time)

def clients(self):

clients = Clients()

clients.setWindowTitle('Клиенты')

clients.exec()

def sevices(self):

sevices = Services()

sevices.setWindowTitle('Услуги')

sevices.exec()

def orders(self):

orders = Order()

orders.setWindowTitle('Услуги')

orders.exec()

class DataBase():

def \_\_init\_\_(self):

self.con = sqlite3.connect('igora.db')

def get\_order(self):

cur = self.con.cursor()

cur.execute("SELECT \* FROM order1")

return cur.fetchall()

def add\_entry(self, time, log, try\_entry):

cur = self.con.cursor()

cur.execute("INSERT INTO history VALUES (?,?,?)", (time, log, try\_entry))

self.con.commit()

def get\_auth\_info(self, log, password):

cur = self.con.cursor()

cur.execute(f'SELECT post, full\_name FROM worker WHERE login="{log}" and password="{password}"')

data = cur.fetchall()

cur.close()

if data != []:

return data

else:

return False

def add\_order(self, service, id\_client, id\_order, time\_serv):

now = datetime.now()

times = now.strftime("%H:%M")

date = now.strftime("%d.%m.20%y")

id = 1

try:

cur = self.con.cursor()

cur.execute("""INSERT INTO order1 VALUES (NULL,?,?,?,?,?,?,?,?)""", (id\_order, date, times, id\_client, service, "Новая", '', time\_serv))

self.con.commit()

cur.close()

except sqlite3.Error as error:

print("Ошибка при работе с SQLite", error)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

window = Auth()

app.exec()