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

Специальность 09.02.07 Информационные системы и программирование

Группа ИСП41-19

Квалификация Программист

КУРСОВОЙ ПРОЕКТ (РАБОТА)

По МДК 02.01 Технология разработки программного обеспечения
(указывается код и название дисциплины, МДК, ПМ)

На тему: Разработка информационной системы станции техобслуживания компьютеров.

Автор курсового проекта (работы) Серегин Степан Сергеевич /
(Фамилия Имя Отчество) (подпись)

Руководитель Жилина Татьяна Анатольевна /
(Фамилия Имя Отчество) (подпись)

Химки
2023 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (МАИ)
филиал «РКТ» МАИ в г. Химки Московской области

Специальность 09.02.07 Информационные системы и программирование

Группа ИСП41-19

Квалификация Программист

З А Д А Н И Е

на курсовой проект (работу)

По МДК 02.01 Технология разработки программного обеспечения
(указывается код и название дисциплины, МДК, ПМ)

Студенту Серегину Степану Сергеевичу /
(Фамилия Имя Отчество) (подпись)

Руководитель Жилина Татьяна Анатольевна /
(Фамилия Имя Отчество) (подпись)

Преподаватель, филиал «РКТ» МАИ в г. Химки Московской области
(Ученая степень, ученое звание, должность и место работы)

1. Наименование темы: Разработка информационной системы станции техобслуживания компьютеров

2. Срок сдачи студентом законченной работы 22.02.23

3. Техническое задание и исходные данные к работе

Разработать информационную систему станции техобслуживания компьютеров, и реализовать её функционал: создание заказа, добавление комплектующих на склад, а также реализовать разграничение доступа.

4. Перечень подлежащих разработке разделов и этапы выполнения работы

№ п/п	Наименование раздела или этапа	Трудоёмкость в % от полной трудоёмкости курсовой проекта(работы)	Срок выполнения
1	Анализ инструментальных средств	5%	29.11.2022
2	Подготовка к разработке	10%	08.12.2022
3	Разработка приложения	50%	15.12.2022
4	Тестирование функционала программы	5%	13.01.2023
5	Создание исполняемого файла и инсталятора	5%	16.01.2023
6	Подготовка расчетно-пояснительной записки	20%	26.01.2023
7	Защита курсовой работы	5%	13.03.2023

5. Перечень иллюстративно-графических материалов:

№ п/п	**Наименование**	**Количество листов**
1	Иллюстрация к пояснительной записке	14
2	Презентация курсового проекта	15

6. Исходные материалы и пособия

1 Изучаем Python, 3 е издание / Лутц М; [пер. с англ. А. Киселева]. — СПб.: Символ Плюс, 2009. — 848 с., ил.

2 Технология разработки программных продуктов. Практикум: учеб. пособие для студ. учреждений сред. проф. образования / А. В.Рудаков, Г. Н. Федорова. — 4-е изд., стер. — М.: Издательский центр «Академия»; 2014. — 192 с. ISBN 978-5-4468-0465-8

3 Python 3 и PyQt 5. Разработка приложений. — 2-е изд., перераб. и доп. / Н. А.Прохоренок, В.А. Дронов. — СПб.: БХВ-Петербург, 2018. — 832 с.: ил. — (Профессиональное программирование)

7. Дата выдачи задания 10.11.22

Руководитель _____

(подпись)

Задание принял к исполнению _____

(подпись)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5
1 Анализ инструментальных средств 6
2 Подготовка к разработке 12
2.1 Диаграмма переходов состояний 12
2.2 Функциональная диаграмма 13
2.3 Диаграмма «Сущность-Связь» 15
2.4 Структурная схема 16
2.5 Функциональная схема 17
2.6 Диаграммы вариантов использования 19
3 Разработка приложения 21
3.1 Разработка базы данных 21
3.2 Разработка шаблонов и макетов 24
3.3 Разработка интерфейсов 26
3.4 Организация связей с БД 28
3.5 Функционал программы 29
4 Тестирование функционала программы 32
5 Создание исполняемого файла и инсталятора 34
ЗАКЛЮЧЕНИЕ 38
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39
ПРИЛОЖЕНИЕ А. Запросы на создание таблиц и добавление в них данных 40
ПРИЛОЖЕНИЕ Б. Иллюстрация форм 41
ПРИЛОЖЕНИЕ В. Исходный код программы 44
ПРИЛОЖЕНИЕ Г. Исходный код тестов 52

ВВЕДЕНИЕ

На данный момент в мире идёт полным ходом цифровизация и автоматизация процессов. Они позволяют получить быстрый доступ к необходимой информации, сократить время на выполнение задач, уменьшить затраты на производство, повысить эффективность работы организаций и улучшить качество жизни людей.

Во всех отраслях используются компьютеры, которые зачастую работают много времени без перерыва, и, соответственно, они подвержены повышенному износу. В связи с этим повышается спрос на станции технического обслуживания компьютеров, которые занимаются починкой, заменой и чисткой компьютерного оборудования.

Для более эффективного управления станцией техобслуживания компьютеров необходимо разработать информационную систему, в которой будут отображены все оказанные услуги, их стоимость, заказчик услуги и сотрудник, оказавший услугу. Также в информационной системе будут отображены новые комплектующие, расходные материалы и бывшие в употреблении комплектующие, находящиеся на складе.

Данная работа имеет практическую значимость, так как создание программного обеспечения для станции техобслуживания компьютеров является актуальной задачей в наше время. Разработанная информационная система может быть использована для автоматизации процессов, ведения учета и повышения качества обслуживания клиентов в фирмах, занимающихся обслуживанием рабочих компьютеров.

В данной курсовой работе будет рассмотрена программная реализация информационной системы, которая позволит быстрее и точнее отслеживать заказы и вести более точный складской учёт.

Одной из главных задач информационной системы станции технического обслуживания компьютеров является представление для сотрудников более наглядной информации об оказываемых услугах. Система должна представлять удобный и интуитивно понятный интерфейс для пользователей, позволяющий быстро и просто оформлять заказы, отслеживать наличие необходимых комплектующих на складе, чтобы дать возможность назвать заказчику более точные сроки выполнения работ.

В результате выполнения данной курсовой работы будет создано программное обеспечение, которое позволит удовлетворить потребности фирмы, занимающейся техобслуживанием компьютеров.

1 Анализ инструментальных средств

На сегодняшний день имеется масса различных инструментов для разработки приложений, для работы с базой данных. Чтобы определиться с конечным вариантом используемого инструментария, необходимо провести сравнительный анализ — отобрать 2-3 программы для каждой задачи, сравнить их функционал, выбрать наиболее подходящую, удобную для разработчика. На сегодняшний день есть много различных СУБД. Их выбор обширен.

Для сравнительного анализа были выбраны СУБД Oracle Database, MySQL, SQLite, так как они наиболее популярны и часто используемые в разработке и администрировании баз данных. Оценивались по следующим критериям: особенности, стоимость и максимальный размер базы данных. Результат сравнительного анализа средств СУБД представлен в таблице 1.

Таблица 1 — Сравнительный анализ средств СУБД

Критерии	Oracle Database	MySQL	SQLite
Особенности	Обрабатывает большие данные. Поддерживает SQL, к нему можно получить доступ из реляционных БД Oracle. Oracle NoSQL Database с Java/C API для чтения и записи данных.	Удобный доступ к базам данных. Корректное распределенное хранение данных на сервере. Быстрый поиск нужной информации в базе с помощью языка SQL. Создание, редактирование и удаление записей, которые есть в базе. Множественный доступ к базе с разных устройств —	Быстрая и легкая встраиваемая однофайловая 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.

Продолжение таблицы 1

Критерии	Oracle Database	MySQL	SQLite
		например, из браузеров нескольких пользователей.	
Стоимость	Стандартная версия $17500	Бесплатная	Бесплатная
Максимальный размер базы данных	11 ГБ	Неограниченно	140 ТБ

В результате проведенного анализа СУБД, исходя из особенностей каждого из представленных вариантов, для использования в данной работе будет использоваться СУБД SQLite, поскольку функционал данной СУБД обширен, интерфейс интуитивно понятен.

Следующий этап анализа инструментальных средства — выбор языка программирования и среды разработки самого приложения.

Для разработки приложения было выбрано три языка программирования — C#, Java и Python. Выбор конкретно данных языков обусловлен тем, что они наиболее востребованы на рынке профессий в данный момент, охватывают больше сфер разработки — от написания простых программ до создания сложных проектов с разработкой графического интерфейса и работы с базами данных. Для оценивания были выбраны следующие критерии: простота в изучении, процесс разработки программ, количество библиотек и производительность. Результат сравнительного анализа языков программирования представлен в таблице 2.

Таблица 1 — Анализ языков программирования

Критерии	C#	Java	Python
Простота в изучении	Как C-типичный язык, язык довольно сложен для изучения	C-типичные конструкции языка усложняют процесс осваивания языка	Упрощенные C-типичные конструкции и простой синтаксис обеспечивают более низкий порог вхождения в язык, способствуют более скорому изучению языка
Разработка программ	Написание и выполнение кода возможно только в специализированной среде разработки с использованием необходимого компилятора	Разработка ведется только в специализированной IDE, либо в IDE, где реализована поддержка всех языков программирования, включая Java	Написание кода возможно, как в IDE, так и в блокноте. При установке интерпретатора Python, выполнение программы возможно даже в командной строке
Библиотеки	Количество библиотек очень мало	Малое количество библиотек	Большое количество библиотек
Производительность	Высокая	Высокая	Низкая

По итогу анализа в качестве используемого языка программирования был выбран Python [1], поскольку он довольно прост в сравнении с другими языками программирования, а его возможности работы с различными сторонними библиотеками очень сильно упрощают и ускоряют процесс разработки и позволяют наполнить программу обширным функционалом.

Заключительный этап — выбор среды разработки.

Так как в качестве используемого языка программирования был выбран Python, необходимо выбирать и анализировать среды разработки, поддерживающие данный язык программирования, либо специализирующиеся только на данном языке программирования.

Интегрированная среда разработки (IDE) — комплекс программных средств, используемый программистами для разработки программного обеспечения.

Среда разработки включает в себя:

— Текстовый редактор;

— Транслятор (компилятор и/или интерпретатор);

— Средства автоматизации сборки;

— Отладчик.

На данный момент существует множество различных сред разработки. Выбор ограничивается лишь желанием разработчика.

Для сравнительного анализа были выбраны среды разработки Microsoft Visual Studio, Eclipse, PyCharm, так как являются наиболее популярными и часто используемыми. Проанализировав данные среды разработки, были определены преимущества и недостатки каждой IDE. Результат сравнительного анализа сред разработки представлен в таблице 3.

Таблица 2 — Сравнительный анализ сред разработки

Критерии	Microsoft Visual Studio	Eclipse	PyCharm
Описание	Премиум IDE, стоимость которой зависит от редакции и типа подписки. Она позволяет создавать самые разные проекты, начиная с мобильных и веб-приложений и заканчивая видеоиграми.	Бесплатная среда разработки, которая хорошо подойдет как новичкам, так и опытным разработчикам. Помимо инструментов отладки и поддержки Git/CVS, Eclipse поставляется с Java и	Кроссплатформенная среда разработки для языка программирования Python. Для разработки приложения используется язык программирования Python и среда разработки PyCharm. Данная IDE является

Продолжение таблицы 3

Критерии	Microsoft Visual Studio	Eclipse	PyCharm
	Microsoft Visual Studio включает в себя множество инструментов для тестирования совместимости.	инструментом для создания плагинов. Изначально Eclipse использовалась только для Java, но сейчас, благодаря плагинам и расширениям, ее функции значительно расширились. Именно из-за возможности расширить Eclipse своими модулями эта платформа и завоевала свою популярность среди разработчиков.	одной из самых удобных и широко используемых сред разработки.
Преимущества	Огромная коллекция всевозможных расширений, которая постоянно пополняется. Технология автодополнения IntelliSense. Возможность кастомизировать рабочую панель.	Возможность программировать на множестве языков. Значительная гибкость среды за счет модульности. Возможность интеграции JUnit. Удаленная отладка (при использовании JVM.	Специализированный под python инструмент, который умеет практически все - отладка, работа с бд, гит, автодополнения, плагины и так далее. Есть версии для всех основных ОС (Windows/Linux/Mac OS).

Продолжение таблицы 3

Критерии	Microsoft Visual Studio	Eclipse	PyCharm
	Поддержка разделенного экрана (split screen).		Есть полноценная бесплатная версия Широко используется в большинстве компаний по разработке программных продуктов.
Недостатки	Тяжеловесность IDE.	Новичкам может быть сложно разобраться в многообразии возможностей.	Потребление ресурсов (место на диске, оперативная память, ЦПУ). Система сложная и большая, на старом железе будет тормозить.

В результате выбор пал на среду разработки PyCharm, так как данная IDE специализируется только на языке Python, а значит, не имеет ничего лишнего, а также обладает всеми возможностями для работы — установка различных пакетов, в том числе и для работы с базами данных и разработки полноценного приложения Windows.

2 Подготовка к разработке

Для создания базы данных необходимо определить основные сущности, атрибуты сущностей, а также реализовать связи между сущностями.

Сущности определяются в процессе анализа предметной области. Вычленяются ключевые элементы, определяются их свойства, устанавливаются взаимосвязи между ними.

Предметная область — Станция техобслуживания компьютеров. Сущности и атрибуты:

— Сущность «Заказ». Атрибуты: ID, дата создания, время создания, клиент, услуга, стоимость услуги, комплектующая, стоимость комплектующей, сумма заказа, дополнительная информация, статус, работник;

— Сущность «Услуга». Атрибуты: ID, название, стоимость;

— Сущность «Клиент». Атрибуты: ID, имя, телефон;

— Сущность «Работник». Атрибуты: ID, ФИО, должность, логин, пароль;

— Сущность «Склад». Атрибуты: ID, тип комплектующей, наименование, количество, стоимость, работник.

2.1 Диаграмма переходов состояний

Диаграмма переходов состояний — это схема переходов и состояния, специальная техника для перехода ТЗ из одного статуса в другой. С ее помощью пользователь в наглядной форме может просматривать переход продукта из одной стадии в другую.

С помощью STD-диаграмм можно моделировать последующее функционирование системы на основе ее предыдущего и текущего функционирования. Моделируемая система в любой заданный момент времени находится точно в одном из конечного множества состояний. С течением времени она может изменить свое состояние, при этом переходы между состояниями должны быть точно определены [2].

Процесс реализации данной диаграммы состоит из нескольких этапов:

— Для начала необходимо определить основные состояния, взаимодействия (условия перехода), выполняемые действия, возможные переходы

— Начальным состоянием является поступление запроса на создание заказа. На схеме начальное состояние обозначается черным кружком. От начального состояния следует совершение действия на получение строки запроса. Схематично действие обозначается стрелочкой, ведущей к промежуточному состоянию;

— Промежуточным состоянием является проверка строки заказа. Исполнение процесса зависит от того, какое из трех заданных условий выполняется. Если проверены не все строки, то данный процесс повторяется до тех пор, пока все строки не будут проверены (на схеме действие обозначается круговой стрелкой исходящей и ведущей в одно место — промежуточное состояние);

— Если все строки проверены, и некоторые позиции отсутствуют на складе, от промежуточного состояния проверки строки действие переходит к состоянию ожидания. Если в процессе ожидания книга была получена, но некоторые позиции также отсутствуют на складе, данный процесс будет совершаться до тех пор, пока все позиции не будут доступны на складе. Когда все позиции доступны, программа переходит в промежуточное состояние формирования заказа книги. После совершения выдачи процесс окончательно завершается;

— В случае, если все строки проверены и все позиции доступны, процесс ожидания минуется и напрямую выполняется процесс формирования выдачи заказанной книги. После этого процесс также успешно завершается.

Результат реализации диаграммы переходов состояний представлен на рисунке 1.

Рисунок 1 — Диаграмма переходов состояний

2.2 Функциональная диаграмма

Функциональная диаграмма – диаграмма, отражающая взаимные связи функций, разрабатываемого программного обеспечения. Они создаются на ранних стадиях

проектирования, для того чтобы помочь проектировщику выявить основные функции и по возможности обнаружить и устранить существующие ошибки.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, представленные в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах

Процесс реализации данной диаграммы состоит из нескольких этапов:

— Определяются функции системы и управляемая информация;

— Функции представляются блоками. Обрабатываемая информацию поступает в блок, то есть добавляется стрелка в левую часть блока, в верхнюю и нижнюю;

— Результат работы блока, то есть функции, выдается с правой стороны блока, с исходящей стрелкой, идущей в другой блок с левой стороны;

— Новый блок является дочерним для предыдущего — родительского.

В него также может поступать информация с внешней стороны (сверху или снизу), если в качестве информации выступает результат работы родительского блока, то стрелка идет в левую часть дочернего блока.

Результат реализации функциональной диаграммы представлен на рисунке 2.

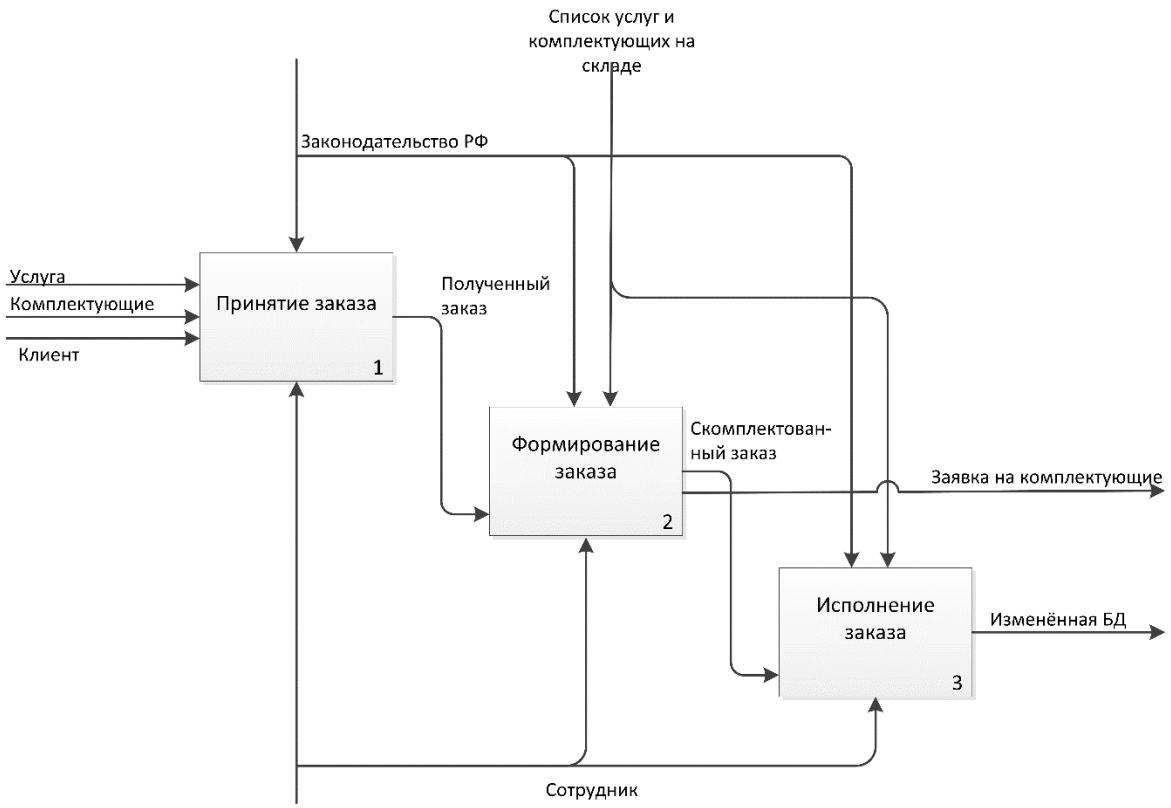


Рисунок 2 — Функциональная диаграмма

2.3 Диаграмма «Сущность-Связь»

Данная диаграмма — (ER-модель данных) обеспечивает стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области.

Диаграмма «Сущность-Связь» — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса.

Пример разработки диаграммы «Сущность-Связь» описан ниже:

— Выделяются основные сущности и связи;

— Перечисляются объекты, которые могут быть полезны при проектировании данных организации, которым соответствуют сущности;

— Между сущностями назначаются связи в зависимости от их назначения;

— В зависимости от сущностей указываются типы установленных связей;

Результат реализации диаграммы представлен на рисунке 3.

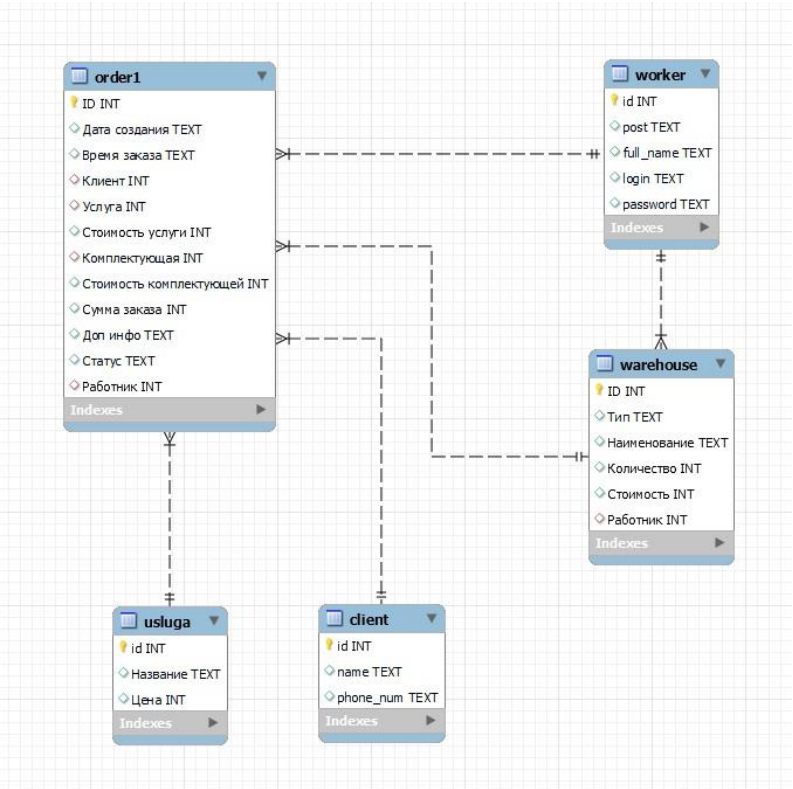


Рисунок 3 — Диаграмма «Сущность-Связь»

2.4 Структурная схема

Структурной называют схему, отражающую состав и взаимодействие по управлению частей разрабатываемого программного обеспечения. Структурная схема программной системы, как правило, показывает наличие подсистем или других структурных компонентов. В отличие от программного комплекса отдельные части (подсистемы) программной системы интенсивно обмениваются данными между собой и, возможно, с основной программой.

Структурные схемы пакетов программ не информативны, поскольку организация программ в пакеты не предусматривает передачи управления между ними. Поэтому структурные схемы разрабатывают для каждой программы пакета, а список программ пакета определяют, анализируя функции, указанные в техническом задании.

Самый простой вид программного обеспечения - программа, которая в качестве структурных компонентов может включать только подпрограммы и библиотеки ресурсов. Разработку структурной схемы программы обычно выполняют методом пошаговой детализации.

Структурными компонентами программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиотеки ресурсов и тому подобное.

Для АИС структурная схема состоит из следующих компонентов:

— Непосредственно, сама информационная система, подразделяющаяся на две подсистемы:

а) Подсистема обработки первичных документов. Данная подсистема отвечает за оформление заказов услуг;

б) Подсистема ведения справочников. Задачей данной подсистемы является ведение учёта комплектующих, сотрудников, клиентов и услуг.

Визуально структурная схема выглядит как граф, где главной вершиной является сама информационная система, а дочерними вершинами являются подсистема обработки первичных документов и подсистема ведения справочников, отвечающие, соответственно, за оформление заказов и за хранение сведений о комплектующих, сотрудниках, клиентах и услугах.

Результат реализации структурной схемы представлен на рисунке 4.

Рисунок 4 — Структурная схема

2.5 Функциональная схема

Функциональная схема — это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функциональных схем используют специальные обозначения, установленные стандартом.

Функциональные схемы, более информативны, чем структурные. Все компоненты структурных и функциональных схем должны быть описаны. При структурном подходе особенно тщательно необходимо прорабатывать спецификации межпрограммных интерфейсов, так как от качества их описания зависит количество самых дорогостоящих ошибок. К самым дорогим относятся ошибки, обнаруживаемые при комплексном тестировании, так как для их устранения могут потребоваться серьезные изменения уже отлаженных текстов.

Для АИС функциональная схема реализуется по следующему алгоритму:

— Начало работы, то есть инициализация всех процессов. Схематично обозначается прямоугольником с закругленными краями;

— Следующий этап — ввод запроса на создание заказа. Действие схематично изображается параллелограммом;

— В зависимости от выбранного варианта действия, программа выполняет следующие функции:

1) Происходит проверка на наличие комплектующих на складе. Схематично изображается блоком с ограниченными полями по краям. Проверка наличия комплектующих на складе (схематично изображается ромбом) может выдать два результата — «Да» или «Нет». В случае «Да» происходит проверка на наличие услуги. В случае «Нет» происходит отмена заказа и завершение рабочего процесса программы;

2) Происходит проверка на наличие услуги, которая может выдать два результата — «Да» или «Нет». В случае «Да» происходит оформление заказа на оказание услуги. В случае «Нет» происходит отмена заказа и завершение рабочего процесса программы;

3) Процесс оформления заказа на получение книги. Данные о заказе сохраняются в хранилище данных (схематично обозначается цилиндр). Следом за сохранением происходит вывод сведений о заказе. Параллельно с этим программа завершает данный рабочий цикл.

Функциональная схема представлена на рисунке 5.

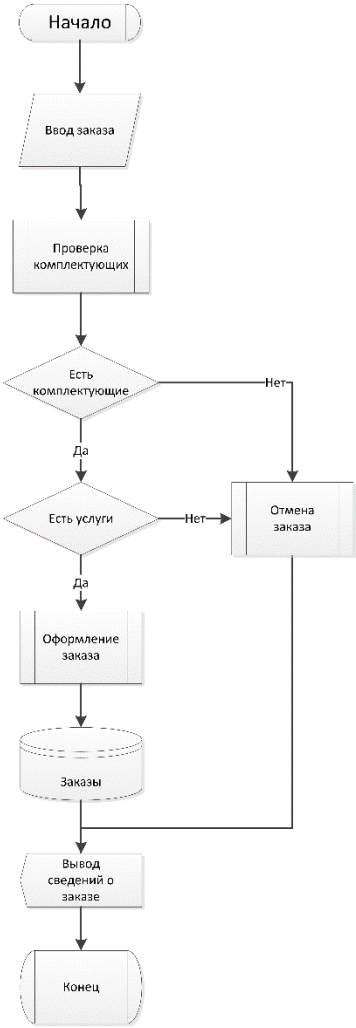


Рисунок 5 — Функциональная схема

2.6 Диаграммы вариантов использования

Диаграмма вариантов использования — это графическое изображение возможных взаимодействий пользователя с системой. Позволяет наглядно представить ожидаемое поведение программной системы.

Главное назначение диаграммы вариантов использования заключается в формализации функциональных требований к системе и возможности согласования полученной модели с заказчиком на ранней стадии проектирования.

Для разработки диаграммы вариантов использования соблюдать некоторую последовательность действий:

— Определить группы пользователей — «Клиент», «Кассир», «Кладовщик»;

— Определить отношения ассоциации между пользователями:

а) «Клиент» — «Оформление заказа»;

б) «Кассир» — «Оформление заказа», «Авторизация», «Просмотр таблицы Услуга», «Просмотр таблицы Склад» и «Просмотр таблицы Заказ»;

в) «Кладовщик» — «Добавление комплектующих на склад», «Авторизация», «Просмотр таблицы Склад» и «Просмотр таблицы Заказ».

— Пользователи схематично обозначаются «фигурой человека»;

— Варианты использования и отношения обозначаются овалом и линиями, указывающими взаимодействие пользователя с вариантом использования.

Реализация диаграммы вариантов использования представлена на рисунке 6.

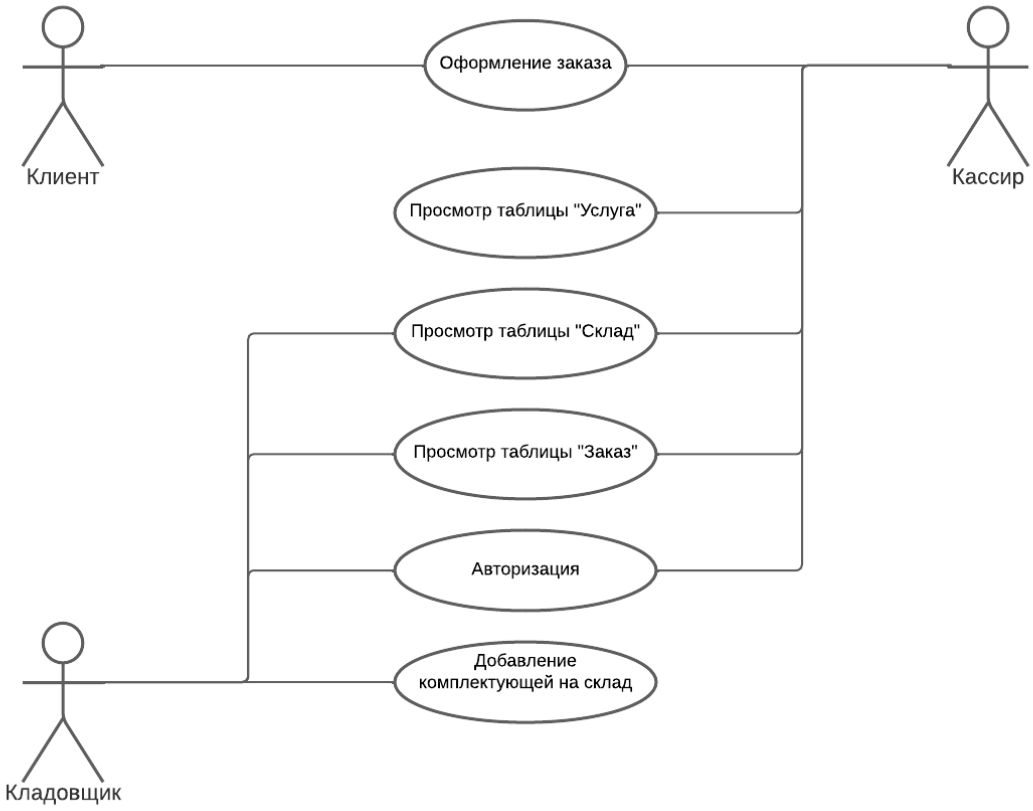


Рисунок 6 — Диаграмма вариантов использования

3 Разработка приложения

3.1 Разработка базы данных

Для разработки базы данных, используя SQLite необходимо провести следующие операции:

Настройка СУБД проводилась следующим образом:

1) Скачать СУБД SQLiteStudio по следующей ссылке: https://sqlitestudio.pl/

2) Откроется окно установщика. Необходимо нажать кнопку «Next», затем согласиться с лицензионным соглашением, выбрав пункт «I accept the agreement» и снова нажать кнопку «Next».

3) Выбрать пункт «Yes» в следующем окне, чтобы файлы SQLite ассоциировались с СУБД SQLiteStudio.

4) Теперь необходимо выбрать директорию, в которую установится SQLiteStudio и нажать кнопку «Next» дважды.

5) Дожидаемся окончания установки и нажимаем кнопку «Finish».

В открывшемся окне СУБД необходимо создать базу данных. Для этого надо нажать на кнопку «Добавит базу данных», после чего откроется окно выбора файла базы данных, в котором необходимо выбрать/создать файл базы данных через проводник. Данное окно представлено на рисунке 7.

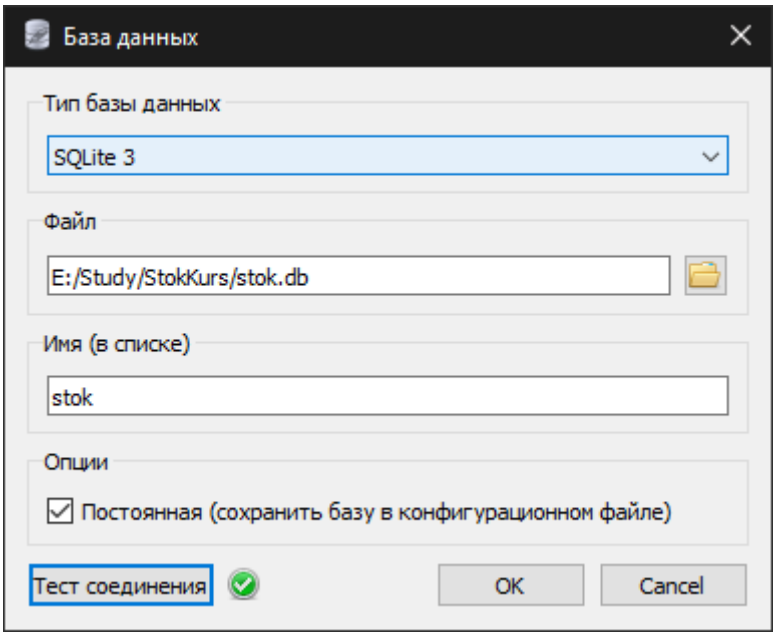


Рисунок 7 — Добавление базы данных

Теперь необходимо выбрать базу данных и нажать на кнопку «Подключится к базе данных», как показано на рисунке 8.

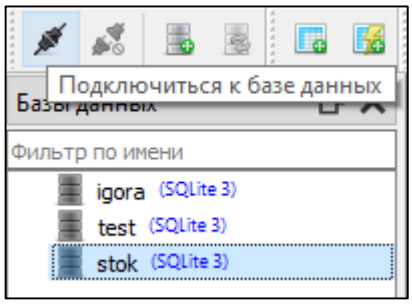


Рисунок 8 — Подключение к базе данных

Также создать базу данных можно, используя SQL-запрос:

“CREATE DATABASE IF NOT EXISTS stok”

Результатом будет появившаяся схема, или же база данных, представленная на рисунке 9.

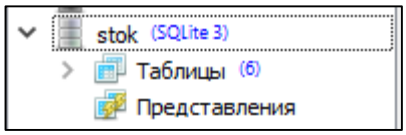


Рисунок 9 — Схема базы данных «stok»

Далее создаются таблицы [3]. Ниже представлены SQL-запросы на создание таблиц:

1) Создание таблицы «Услуга»:

```
CREATE TABLE usluga (
   id      INTEGER PRIMARY KEY,
   Название TEXT,
   Цена    INTEGER
);
```

2) Создание таблицы «Склад»:

```
CREATE TABLE warehouse (
   ID          INTEGER PRIMARY KEY,
   Тип         TEXT,
   Наименование TEXT,
   Количество   INTEGER,
   Стоимость    INTEGER,
```

Работник TEXT

CONSTRAINT `warehouse_FK` FOREIGN KEY (`Работник`) REFERENCES `worker` (`ID`),

);

Остальные запросы на создание таблиц по структуре идентичны представленным выше и приведены в приложении А. Результат выполнения запросов на создание таблиц представлен на рисунке 10.

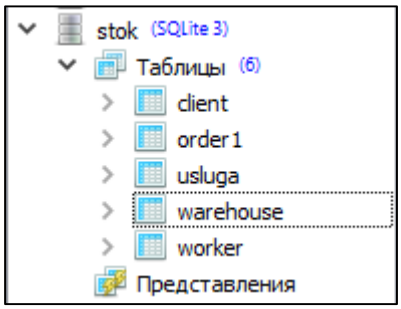


Рисунок 10 — Создана база данных и таблицы

Благодаря функционалу SQLiteStudio у разработчиков есть возможность добавлять данные несколькими способами:

— Написание SQL-запросов в поле ввода текста;

— Ручное добавление данных в ячейки таблиц базы данных;

В качестве добавления данных в БД был выбран способ ручного добавления данных в ячейки таблиц базы данных, поскольку этот способ прост и интуитивно понятен, также этот способ подходит в связи с тем, что объем данных для добавления небольшой. На рисунке 11 представлен результат добавления данных в таблицу «Услуга».

	id	Название	Цена
1	1	Диагностика	444
2	2	Замена процессора	500
3	3	Замена видеокарты	500
4	4	Замена материнской платы	500
5	5	Замена ОЗУ	500
6	6	Обновление ПО	750
7	7	Установка ОС	1000
8	8	Продувка от пыли	250

Рисунок 11 — Результат добавления данных

Важнейшей частью данной работы является разработка Windows-приложения для взаимодействия с базой данных.

Главная цель разработки приложения — обеспечить вывод всех требуемых данных из базы данных, реализовать возможность добавления новых данных: новых заказов, новых комплектующих. Реализовать форму авторизации как средство разделения уровня доступа к базе данных.

3.2 Разработка шаблонов и макетов

Для разработки приложения необходимо изначально спроектировать макеты форм интерфейса приложения:

— Макет формы главного окна приложения;

— Макет формы авторизации;

— Макет вывода данных.

На рисунке 12 представлен разработанный макет формы авторизации. В центральной части макета расположены поля для ввода логина и пароля, рядом с полем ввода пароля расположена кнопка «показать пароль». Внизу расположена кнопка входа в систему.

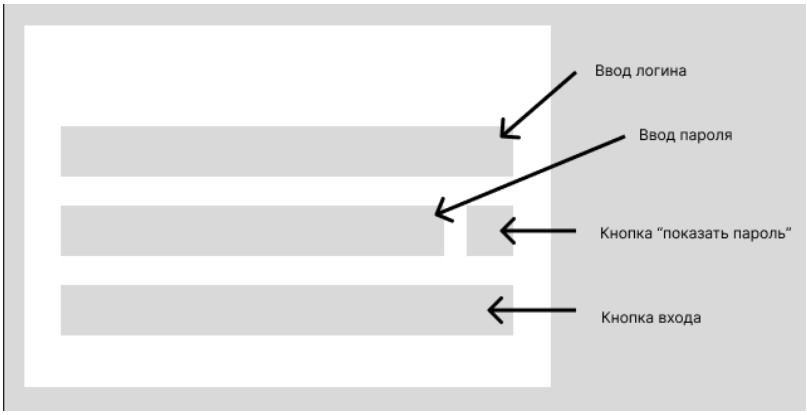


Рисунок 12 — Макет формы авторизации

Затем проектируется макет формы главного окна приложения. В верхней части макета расположены кнопки переключения между окнами, ниже расположено само окно. Внизу располагается кнопка выхода. Результат представлен на рисунке 13.

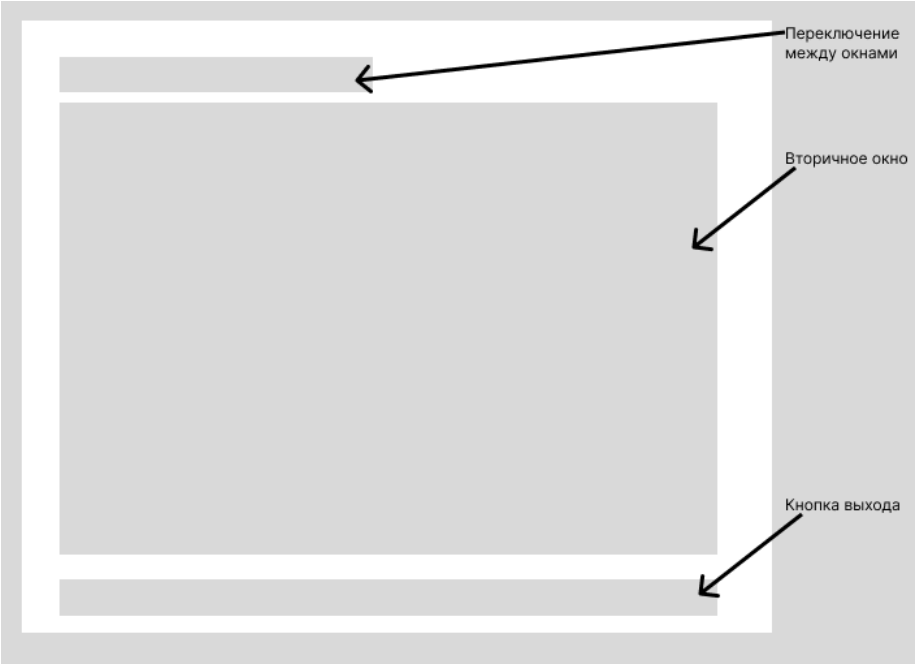


Рисунок 13 — Макет формы главного окна приложения

Следующий макет — макет формы окна вывода данных. В макете присутствует таблица для вывода необходимых данных. Результат представлен на рисунке 14.

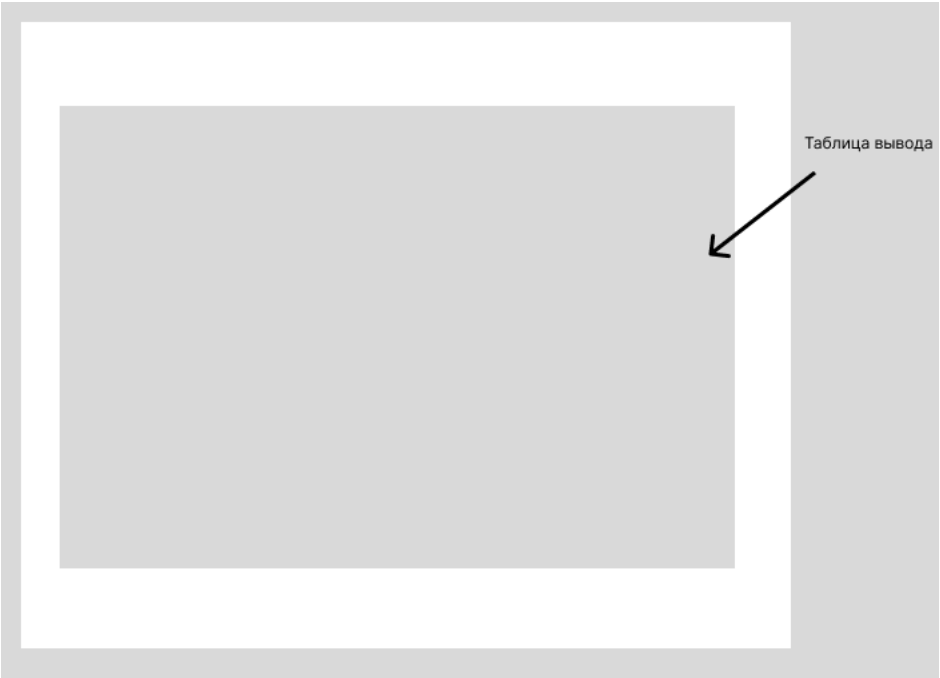


Рисунок 14 — Макет формы окна вывода данных

3.3 Разработка интерфейсов

Для создания форм интерфейса используется инструмент QtDesigner [4] из пакета pyqt5-tools. Данный инструмент необходим при разработке форм интерфейса Windows-приложения. Он позволяет создавать окна для вывода любой необходимой информации, создавать кнопки и назначать им определенные функции, обеспечивать возможность ввода данных с клавиатуры, представлять данные в табличном виде, а также работать с графическими элементами.

Для реализации всех форм используются следующие объекты QtDesigner:

— QPushButton — кнопка, совершающая назначенное действие;

— QMainWindow — класс главного окна;

— QWidget — класс виджета, всплывающего окна;

— QTableWidget — для вывода данных в табличном виде;

— QListWidget — для вывода данных в виде списка.

Алгоритм создания формы в QtDesigner на примере формы авторизации:

1) Выбираем «Main Window» при выборе типа формы.

2) Далее из списка элементов WidgetBox необходимо переместить все используемые в форме элементы:

— QLineEdit — для ввода данных с клавиатуры;

— QLabel — для установления текста в конкретной области формы;

— QPushButton — для кнопок, выполняющих определённые действия;

— QHVoxLayout — для группировки виджетов горизонтально;

— QVBoxLayout — для группировки виджетов вертикально;

— Line — для визуального отделения виджетов.

3) Чтобы сгруппировать все объекты используются элементы QHBoxLayout и QVBoxL.

4) Чтобы изменить название виджета, выбирается нужный виджет по нажатию левой кнопки мыши, в свойствах объекта вносятся изменения в поле «objectName».

5) Чтобы установить изображение в QLabel, необходимо изменить поле «pixmap», указав путь к файлу изображения

6) Чтобы изменить цвет фона элемента, необходимо изменить поле «styleSheet».

Форма авторизации представлена на рисунке 15.

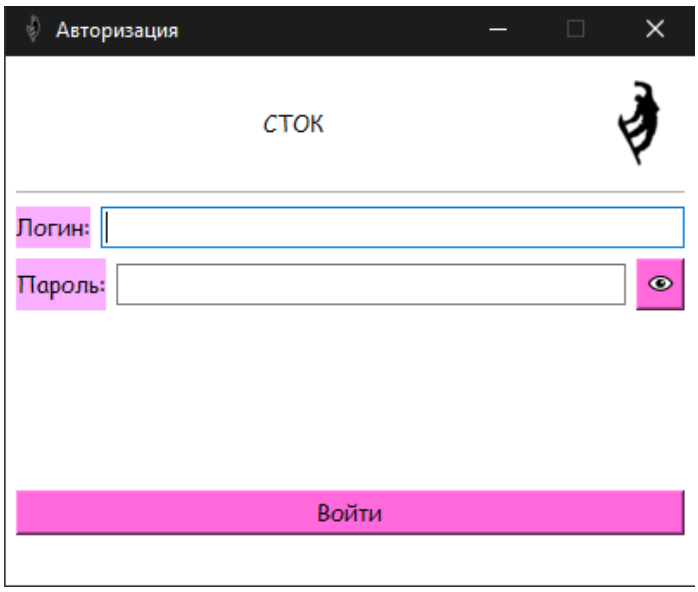


Рисунок 15 — Форма авторизации

В зависимости от должности сотрудника, в программе реаилизовано разделение уровня доступа к содержимому базы данных и функционала программы. На рисунке 16 показана форма окна информации о пользователе.

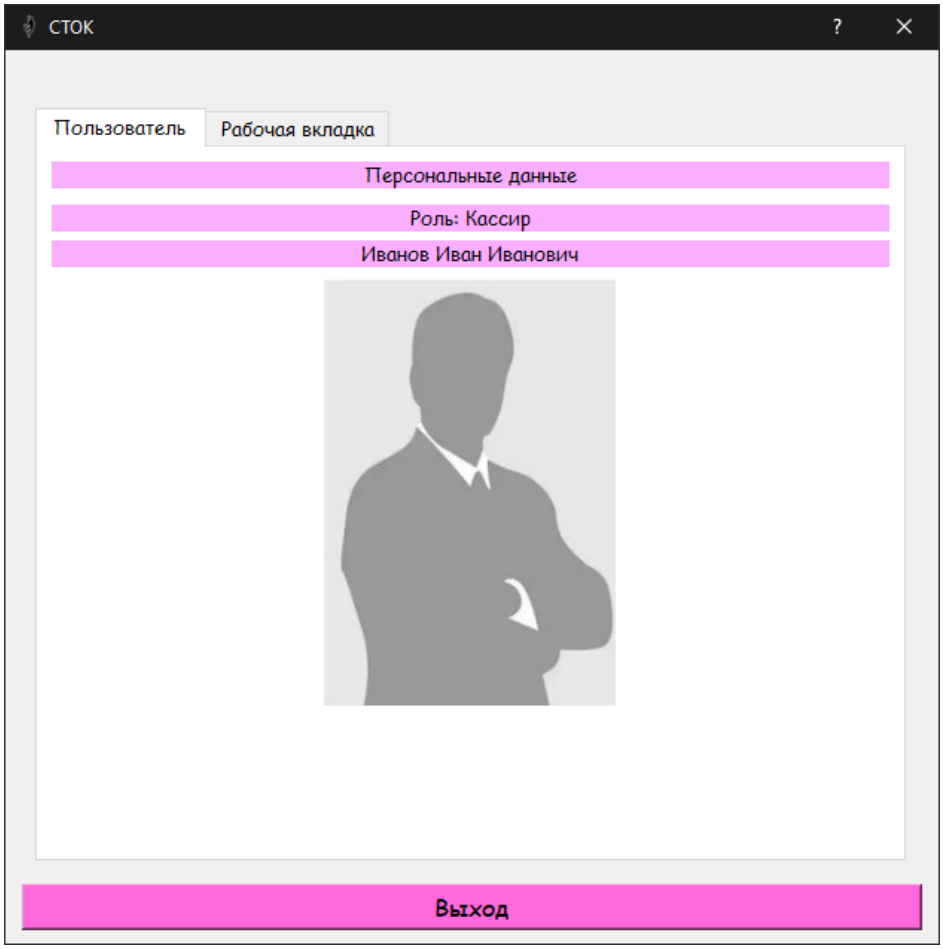


Рисунок 16 — Форма окна информации о пользователе

В зависимости от должности сотрудник видит только ту информацию, которая ему предоставлена. Роль кассира позволяет просматривать следующие формы:

— Создание заказа;

— Просмотр заказов;

— Просмотр склада;

— Просмотр услуг.

Для кладовщика реализованы следующие формы:

— Добавление товара на склад;

— Просмотр склада;

— Просмотр заказов.

Формы окон приложения представлены в приложении Б.

3.4 Организация связей с БД

Реализовать модуль для взаимодействия с базой данных необходимо для полноценной работы программы.

Сам модуль реализован на языке Python через класс DataBase с использованием библиотеки «sqlite3» для связи с созданной БД в СУБД SQLiteStudio. В модуле обеспечивается подключение к файлу базы данных, с помощью следующего кода:

“self.con = sqlite3.connect('stok.db')”

где ‘stok.db’ — название файла базы данных.

Взаимодействие модулей графического интерфейса и базы данных осуществлено при помощи строки кода в классе окна:

“self.DB = DataBase()”

При авторизации в системе выполняется запрос на получение данных о пользователе по введенному логину. Делается это для того, чтобы реализовать разделенный доступ к базе данных в целях безопасности.

В программе присутствует разделение уровня доступа к базе данных по должности кассира и кладовщика с различными полномочиями.

Для пользователя «Кладовщик» реализованы следующие запросы:

В окне личной информации реализованы запросы:

— Получение ФИО;

— Получение должности.

В рабочем окне реализованы запросы:

— Добавление записи на склад;

— Просмотр склада;

— Просмотр заказов.

Для пользователя «Кассир» реализованы следующие запросы:

В окне личной информации реализованы запросы:

— Получение ФИО;

— Получение должности.

В рабочем окне реализованы запросы:

— Добавление заказа;

— Просмотр заказов;

— Просмотр склада;

— Просмотр услуг.

Листинг класса базы данных представлен в приложении В.

3.5 Функционал программы

Для реализации данной системы был использован один из самых используемых подходов к программированию — объектно-ориентированное (ООП).

Суть данного подхода заключается в том, что каждый модуль программы является отдельным классом, функционал модуля — это метод класса, а свойства модуля — атрибуты данного класса.

При разработке программы было реализовано несколько модулей — классов.

Для модуля, обеспечивающего работу с базой данных, создан класс DataBase. Он используется библиотеку языка Python под названием «sqlite3» для обеспечения работы с СУБД SQLiteStudio и налаживания связи с самой базой данных. Ниже перечислены методы данного класса:

— __init__(self) — подключение к базе данных;

— get_order(self) — получение всех данных из таблицы «Заказ»;

— get_warehouse(self) — получение всех данных из таблицы «Склад»;

— get_auth_info(self, log, password) — получение данных для авторизации;

— add_order(self, service, client, serv_cost, kompl, kompl_cost, summ, info,worker) — добавление нового заказа;

— add_wh(self, type, name, count, cost, worker) — добавление новой записи на склад;

— get_client(self) — получение имени клиента;

— get_service(self) — получение названия услуги;

— get_serv_c(self, service) — получение стоимости услуги;

— get_kompl(self) — получение названия комплектующей;

— get_kompl_c(self, kompl) — получение стоимости комплектующей.

Следующий модуль — модуль графического интерфейса. С использованием библиотеки PyQt5 для языка Python было реализовано несколько классов данного модуля.

В процессе проектирования было установлено, что модуль графического интерфейса будет иметь несколько разделов для каждой функции. В связи с этим и проводилась разработка нескольких классов для каждого окна интерфейса

Для основного окна интерфейса программы разработан класс Window(QDialog). Ниже перечислены методы данного класса:

— __init__(self, post, fullName, parent = None) — отвечает за подключение к кнопкам, объявление переменных, заполнение таблиц, получение списка страниц StackedWidget;

— update_serv_cost(self) — обновляет отображаемую стоимость услуги;

— update_kompl_cost(self) — обновляет отображаемую стоимость комплектующей;

— update_sum(self) — обновляет отображаемую сумму услуги и комплектующей;

— exit(self) — кнопка выхода из окна и открытие окна авторизации;

— add_order(self) — добавление заказа в базу данных;

— add_wh(self) — добавление комплектующей в базу данных;

— orders(self) — открытие окна заказов;

— warehouse(self) — открытие окна склада;

— services(self) — открытие окна услуг;

— build_combobox_client(self) — построение QComboBox с именем клиента;

— build_combobox_service(self) — построение QComboBox с названием услуги;

— build_serv_cost(self) — построение стоимости услуги;

— build_combobox_kompl(self) — построение QComboBox с названием комплектующей;

— build_kompl_cost(self) — построение стоимости комлпектующей.

Для окна авторизации разработан класс Auth(QMainWindow). Ниже перечислены методы данного класса:

— __init__(self, parent = None) — отвечает за подключение к кнопкам, подключение к базе данных;

— hide_pas(self) — отвечает за кнопку «показать пароль»;

— auth(self) — функция авторизации.

Для окон просмотра заказов, склада и услуг были разработаны классы Order(QDialog), Warehouse(QDialog) и Srevice(QDialog) соответственно. Метод у этих классов один и имеет общую концепцию во всех трёх классах:

— __init__(self, parent = None) — отвечает за открытие окна и заполнение таблицы данными в зависимости от того, какой класс был вызван.

Листинг основного кода программы можно посмотреть в приложении В.

4 Тестирование функционала программы

Тестированием называется выполнение программы в целях обнаружения ошибок. Отладкой называется локализация и исправление ошибок. Тесты разрабатывают в определённом порядке. При этом по внешней спецификации различают тесты:

— для каждого класса входных данных;

— для граничных и особых значений входных данных.

Контролируется, все ли классы выходных данных при этом проверены, и добавляются при необходимости нужные тесты.

Для тестирования функционала информационной системы «Станция техобслуживания компьютеров», с использованием библиотеки «unittest», был реализован класс TestPush(TestCase) для тестирования функционала добавления данных.

Для класса TestPush() реализованы следующие модули — тесты конкретных функций:

— setUp() — инициализация класса, подключение ко всем модулям и классам для тестирования их функций;

— test_push_order() — тестирование функции добавления заказа. Имитируется нажатие кнопки «btn_add_order» — «Сформировать заказ» — с заданными значениями;

— test_push_wh() — тестирование функции добавления добавления записи на склад. Имитируется ввод данных о записи с тестовыми данными и нажатие кнопки «btn_add_wh» — «Добавить на склад».

Результаты проведения тестирования функционала представлены в таблице 4.

Таблица 4 — Результаты тестирования функционала

Название теста	Входные данные	Результат
test_push_order	В интерфейсе приложения выбраны данные о клиенте из QComboBox, услуга из QComboBox и комлпектующая из QComboBox. Левый клик мыши по кнопке btn_add_order — «Сформировать заказ».	Добавление записи о заказе в базу данных.

Продолжение таблицы 4

test_push_wh	В интерфейс приложения введен тип комплектующей (test), название комплектующей (test), количество (99) и стоимость (99). Левый клик мыши по кнопке btn_add_wh — «Добавить на склад».	Добавление записи о комплектующей в базу данных.

По результатам тестирования функционала программы ошибки обнаружены не были. Листинг кода тестов представлен в приложении Г.

5 Создание исполняемого файла и инсталятора

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.

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, 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». Окно представлено на рисунке 17.

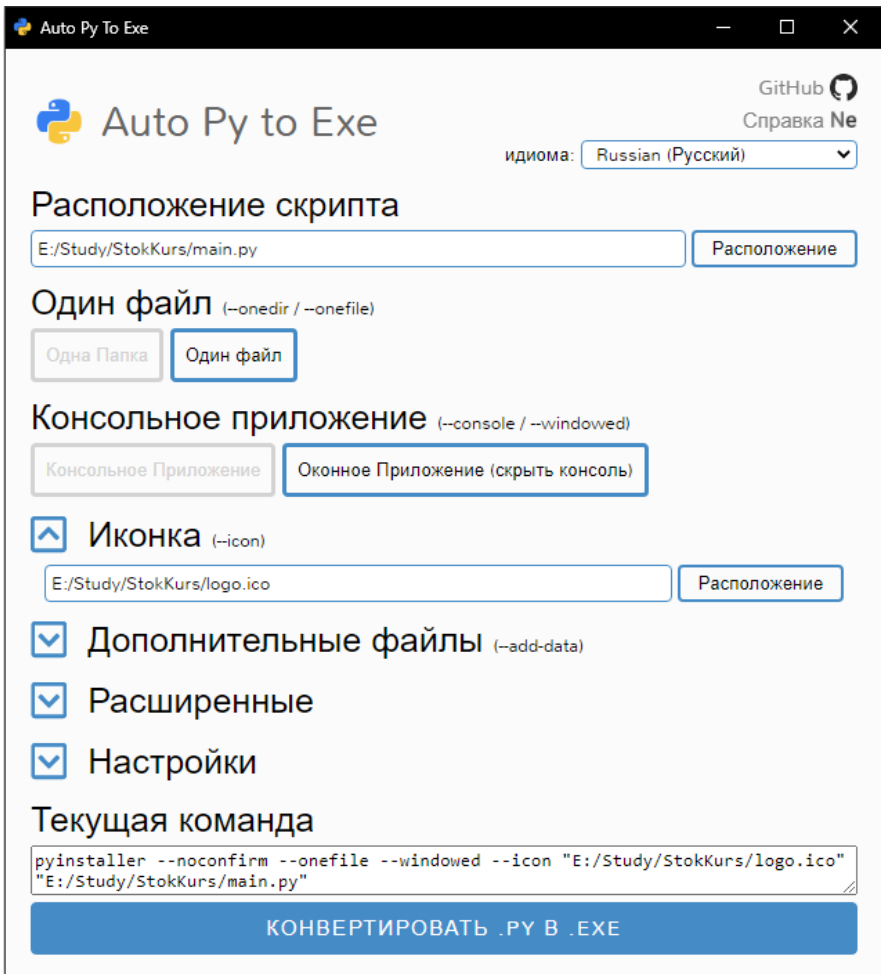


Рисунок 17 — Окно auto-py-to-exe

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

После создания исполняемого файла .exe появится директория output. В неё необходимо скопировать папки с формами и изображениями и файл базы данных. Результат представлен на рисунке 18.

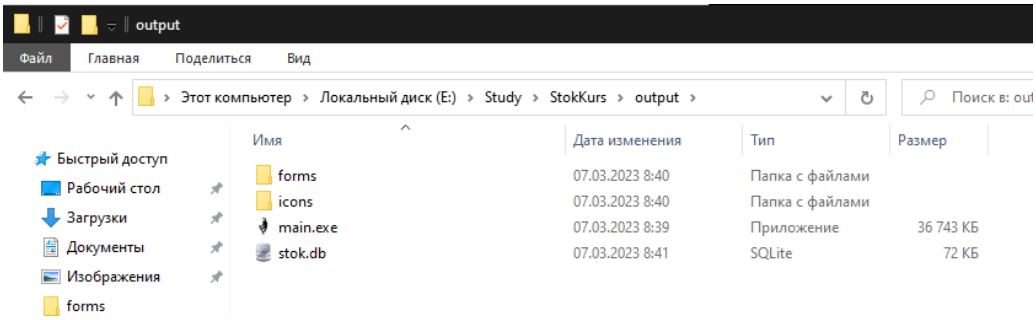


Рисунок 18 — Папка с исполянемым файлом

Следующим шагом необходимо создать инсталлятор приложения. Для этого нужно воспользоваться программой Inno Setup Compiler. После запуска программы в появившимся окне выбираем пункт «Create a new script file using the Script Wizard». После чего нажимаем «OK» и «Next». Окно представлено на рисунке 19.

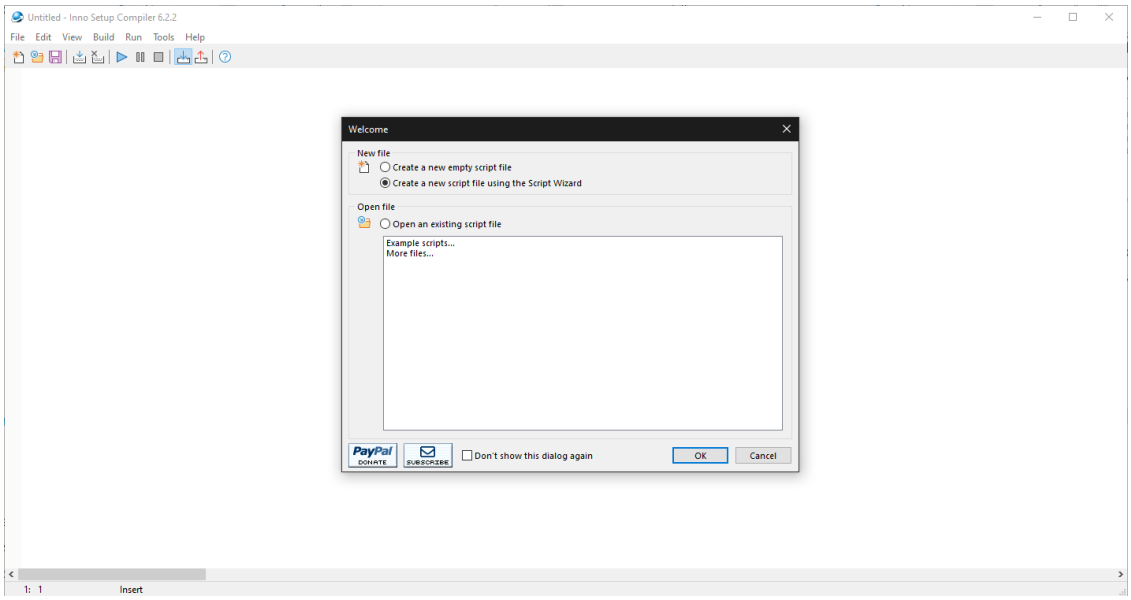


Рисунок 19 — Окно создания нового скрипта

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 и нажать «Next», убрать галочку сверху и нажать три раза «Next». Окно представлено на рисунке 20.

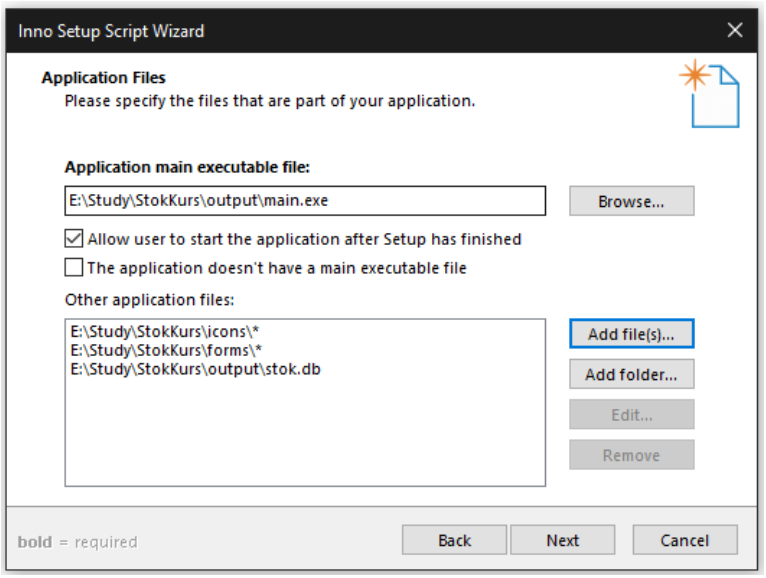


Рисунок 20 — Окно настройки инсталлятора

После в открывшимся окне выбираем пункт «Non administrative mode» и нажимаем «Next». Далее следует выбрать язык инсталлятора, после чего нажать «Next». В следующем окне указывается папка (в которой будет создан итоговый файл инсталлятора) и иконка.

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.

ЗАКЛЮЧЕНИЕ

Проектирование информационной системы является важным и сложным этапом в разработке любого программного продукта. В процессе выполнения данной курсовой работы были рассмотрены различные этапы проектирования, начиная анализом требований и заканчивая созданием макетов и шаблонов интерфейса.

В ходе выполнения работы были проанализированы требования к информационной системе станции техобслуживания компьютеров, определены её функциональные и нефункциональные требования. Была разработана модель данных с использованием UML- и ER-диаграмм. Также были разработаны макеты экранных форм с использованием PyQt5.

В результате выполнения курсовой работы были получены теоретические и практические знания в проектировании информационных систем, разработке ER-диаграммы и функциональной модели. Также был получен опыт работы с библиотекой PyQt5 для создания графического интерфейса программного обеспечения.

В результате проведённой работы была создана информационная система, которая может удовлетворить потребности станции технического обслуживания компьютеров в систематическом хранении и отслеживании данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Изучаем Python, 3-е издание / Лутц М; [пер. с англ. А. Киселева]. — СПб.: Символ-Плюс, 2009. — 848 с., ил.

2 Технология разработки программных продуктов. Практикум: учеб. пособие для студ. учреждений сред. проф. образования / А. В.Рудаков, Г. Н. Федорова. — 4-е изд., стер. — М. : Издательский центр «Академия»; 2014. — 192 с. ISBN 978-5-4468-0465-8

3 Основы проектирования баз данных / Г. Н. Федорова. - Москва : Академия, 2017. - 218, [1] с. : ил.

4 Python 3 и PyQt 5. Разработка приложений. — 2-е изд., перераб. и доп. / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2018. — 832 с.: ил. — (Профессиональное программирование)

ПРИЛОЖЕНИЕ А

Запросы на создание таблиц и добавление в них данных

Создание таблицы «Клиент»:

```
CREATE TABLE client (
   id        INTEGER PRIMARY KEY,
   name      TEXT,
   phone_num TEXT
);
```

Создание таблицы «Работник»:

```
CREATE TABLE worker (
   id            PRIMARY KEY,
   post      TEXT,
   full_name  TEXT,
   login      TEXT,
   password   TEXT,
);
```

Создание таблицы «Заказ»:

```
CREATE TABLE order1 (
   ID                    INTEGER PRIMARY KEY AUTOINCREMENT,
   [Дата создания]          TEXT,
   [Время заказа]           TEXT,
   Клиент                   TEXT,
   Услуга                   TEXT,
   [Стоимость услуги]       INTEGER,
   Комплектующая            TEXT,
   [Стоимость комплектующей] INTEGER,
   [Сумма заказа]           INTEGER,
   [Доп инфо]               TEXT,
   Статус                   TEXT,
   Работник                 TEXT,
);
```

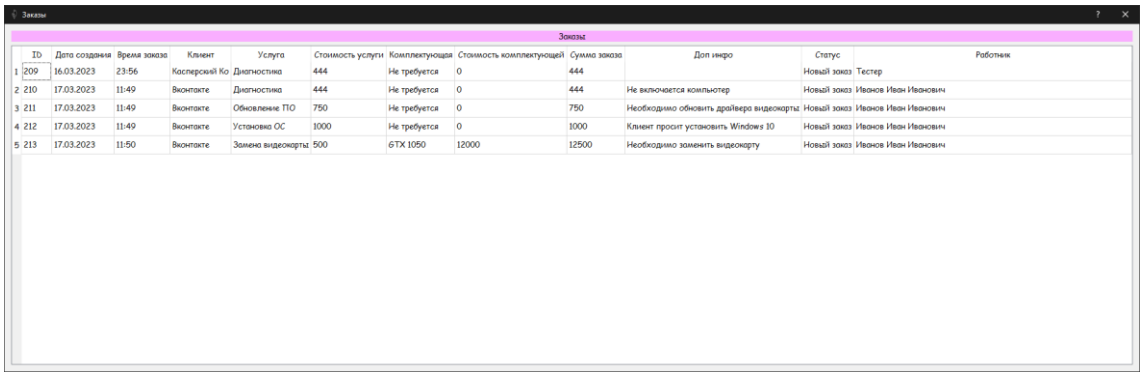
ПРИЛОЖЕНИЕ Б

Иллюстрация форм

Форма создания заказа представлена на рисунке Б.1.

Рисунок Б.1 — Форма создания заказа

Форма просмотра заказов представлена на рисунке Б.2.



Заказы

	ID	Дата создания	Время заказа	Клиент	Услуга	Стоимость услуги	Комплектующая	Стоимость комплектующей	Сумма заказа	Доп инфо	Статус	Работник
1	209	16.03.2023	23:56	Касперский Ко	Диагностика	444	Не требуется	0	444		Новый заказ	Тестер
2	210	17.03.2023	11:49	Вконтакте	Диагностика	444	Не требуется	0	444	Не включается компьютер	Новый заказ	Иванов Иван Иванович
3	211	17.03.2023	11:49	Вконтакте	Обновление ПО	750	Не требуется	0	750	Необходимо обновить драйвера видеокарты	Новый заказ	Иванов Иван Иванович
4	212	17.03.2023	11:49	Вконтакте	Установка ОС	1000	Не требуется	0	1000	Клиент просит установить Windows 10	Новый заказ	Иванов Иван Иванович
5	213	17.03.2023	11:50	Вконтакте	Замена видеокарты	500	GTX 1050	12000	12500	Необходимо заменить видеокарту	Новый заказ	Иванов Иван Иванович

Рисунок Б.2 — Форма просмотра заказов

Форма просмотра склада представлена на рисунке Б.3.

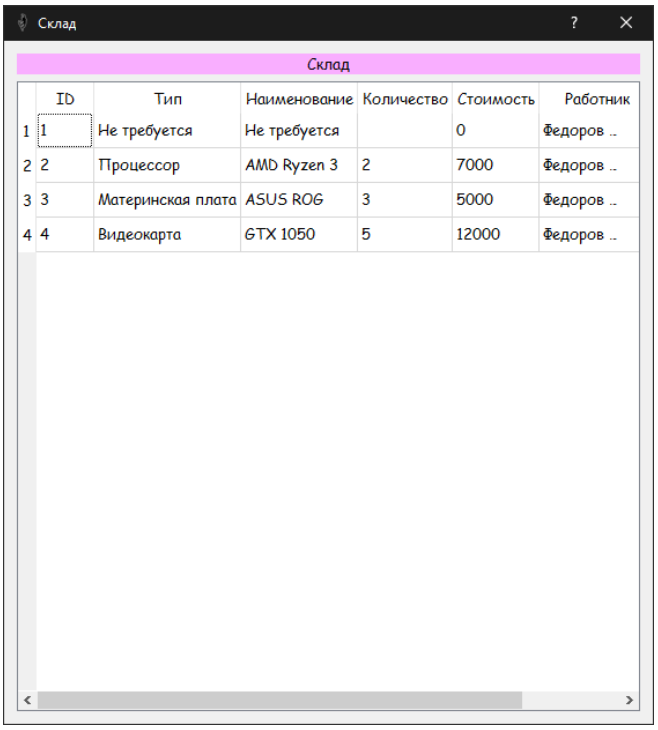


Рисунок Б.3 — Форма просмотра склада

Форма просмотра услуг представлена на рисунке Б.4.

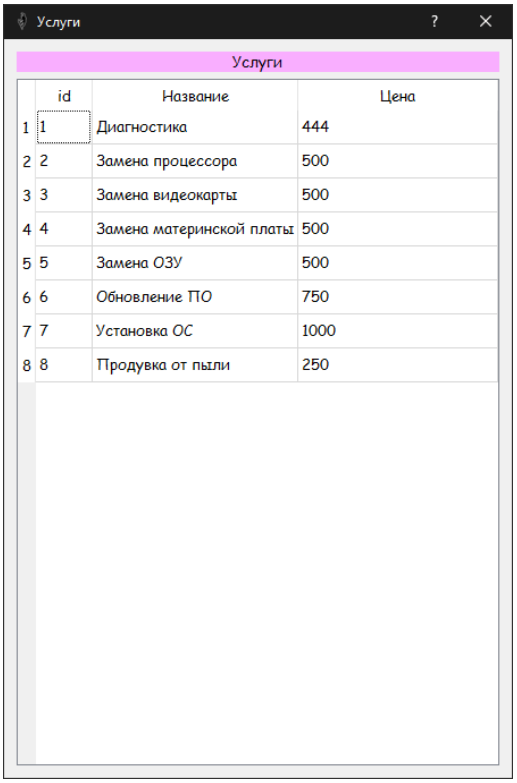


Рисунок Б.4 — Форма просмотра услуг

Форма добавления записи на склад представлена на рисунке Б.5.

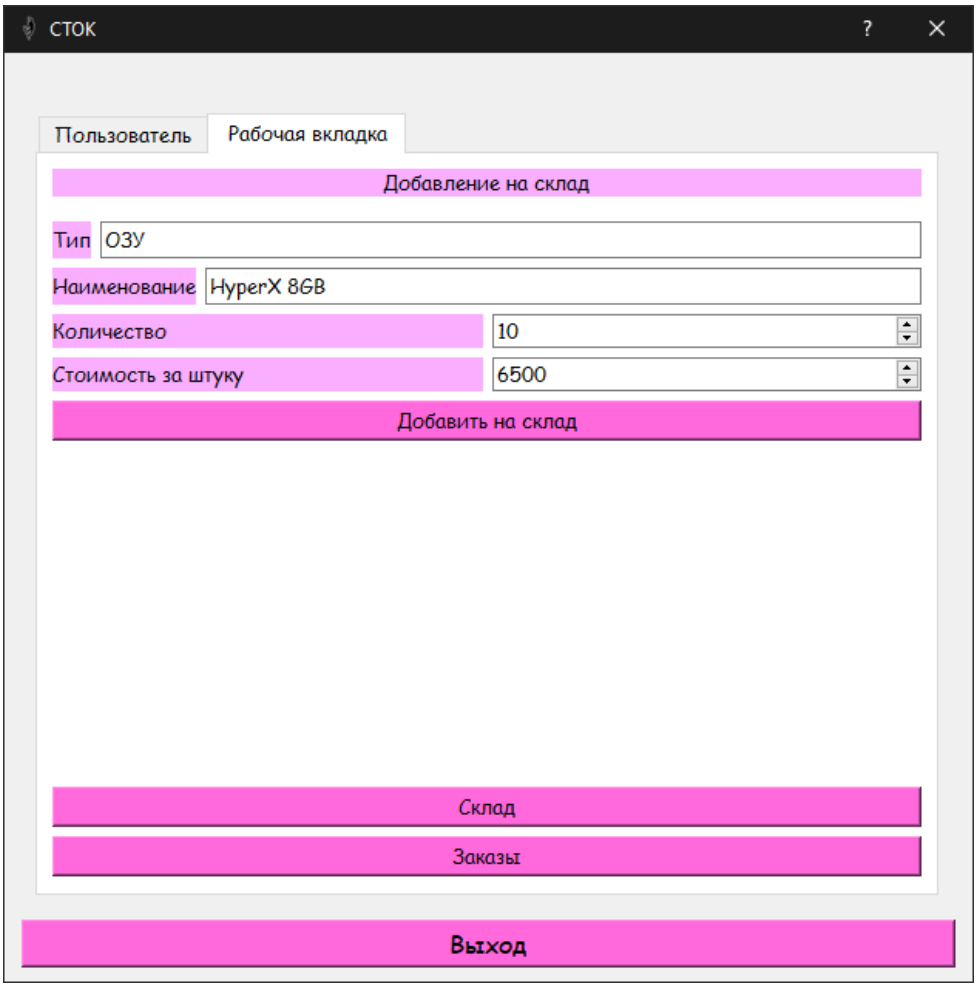


Рисунок Б.5 — Форма добавления записи на склад

ПРИЛОЖЕНИЕ В

Исходный код программы

main.py

```
import sqlite3
import sys
from datetime import datetime

from PyQt5 import uic
from PyQt5.QtGui import QIcon
from PyQt5.QtSql import QSqlDatabase, QSqlQueryModel
from PyQt5.QtWidgets import *
from PyQt5.QtWidgets import QLineEdit, QComboBox


class Auth(QMainWindow):
    def __init__(self, parent = None):
        super().__init__(parent)
        self.DB = DataBase()
        self.ui = uic.loadUi('forms/auth.ui', self)
        self.setWindowIcon(QIcon('logo.ico'))
        self.ui.show()
        self.btn_enter.clicked.connect(self.auth)
        self.btn_pas.clicked.connect(self.hide_pas)
        self.hide_password = True
        worker = self.auth

    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('СТОК')

            main_win.exec()
```

```
            return full_name
        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 Order(QDialog):
    def __init__(self, parent = None):
        super().__init__(parent)
        self.ui = uic.loadUi('forms/table_ord.ui', self)
        self.setWindowIcon(QIcon('logo.ico'))

        con = QSqlDatabase.addDatabase("QSQLITE")
        con.setDatabaseName("stok.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 Warehouse(QDialog):
    def __init__(self, parent = None):
        super().__init__(parent)
        self.ui = uic.loadUi('forms/table_wh.ui', self)
        self.setWindowIcon(QIcon('logo.ico'))

        con = QSqlDatabase.addDatabase("QSQLITE")
        con.setDatabaseName("stok.db")
        con.open()

        self.model = QSqlQueryModel()   # модель (таблица) без редактирования данных
        self.model.setQuery("SELECT * FROM warehouse")
        self.tableView.setModel(self.model)
        self.tableView.horizontalHeader().setStretchLastSection(True)

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


class Service(QDialog):
    def __init__(self, parent = None):
        super().__init__(parent)
        self.ui = uic.loadUi('forms/table_serv.ui', self)
        self.setWindowIcon(QIcon('logo.ico'))

        con = QSqlDatabase.addDatabase("QSQLITE")
        con.setDatabaseName("stok.db")
```

```
        con.open()

        self.model = QSqlQueryModel()   # модель (таблица) без редактирования данных
        self.model.setQuery("SELECT * FROM usluga")
        self.tableView.setModel(self.model)
        self.tableView.horizontalHeader().setStretchLastSection(True)

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


class Info(QDialog):
    def __init__(self, parent = None):
        super().__init__(parent)
        self.ui = uic.loadUi('forms/info.ui', self)
        self.setWindowIcon(QIcon('logo.ico'))

        self.ui.info_lbl.setText('Заказ оформлен')
        self.ui.btn_ok.clicked.connect(self.exit)

    def exit(self):
        self.close()


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.setWindowIcon(QIcon('logo.ico'))
        self.ui.lbl_role.setText('Роль: ' + post + '\n' + 'ФИО: ' + fullName)
        self.ui.btn_add_order.clicked.connect(self.add_order)
        self.ui.btn_add_wh.clicked.connect(self.add_wh)
        self.ui.btn_ord.clicked.connect(self.orders)
        self.ui.btn_ord_2.clicked.connect(self.orders)
        self.ui.btn_wh.clicked.connect(self.warehouse)
        self.ui.btn_wh_2.clicked.connect(self.warehouse)
        self.ui.btn_usl.clicked.connect(self.services)
        self.ui.btn_exit.clicked.connect(self.exit)

        self.build_combobox_client()
        self.build_combobox_service()
        self.build_serv_cost()
        self.build_combobox_kompl()
        self.build_kompl_cost()

        if post == 'Кассир':
            self.ui.lbl_role.setText('Роль: ' + post)
            self.ui.lbl_name.setText(fullName)
            self.ui.stackedWidget.setCurrentIndex(0)
        elif post == 'Кладовщик':
            self.ui.lbl_role2.setText('Роль: ' + post)
            self.ui.lbl_name_2.setText(fullName)
```

```
        self.ui.stackedWidget.setCurrentIndex(1)
        # self.update_table_history()

    self.ui.usluga_box.currentIndexChanged.connect(self.update_serv_cost)
    self.ui.usluga_box.currentIndexChanged.connect(self.update_sum)
    self.update_serv_cost()
    self.ui.kompl_box.currentIndexChanged.connect(self.update_kompl_cost)
    self.ui.kompl_box.currentIndexChanged.connect(self.update_sum)
    self.update_kompl_cost()
    self.update_sum()

def update_serv_cost(self):
    service = self.ui.usluga_box.currentText()
    self.ui.usl_cost.setText(str(self.DB.get_serv_c(service)[0][0]))

def update_kompl_cost(self):
    kompl = self.ui.kompl_box.currentText()
    self.ui.kompl_cost.setText(str(self.DB.get_kompl_c(kompl)[0][0]))

def update_sum(self):
    service_cost = self.ui.usl_cost.text()
    kompl_cost = self.ui.kompl_cost.text()
    summary = int(service_cost) + int(kompl_cost)
    self.ui.summary.setText(str(summary))

def exit(self):
    self.close()
    auth = Auth()

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 add_order(self):
            client = self.ui.client_box.currentText()
            service = self.ui.usluga_box.currentText()
            service_cost = self.ui.usl_cost.text()
            kompl = self.ui.kompl_box.currentText()
            kompl_cost = self.ui.kompl_cost.text()
            summary = int(service_cost) + int(kompl_cost)
            info = self.ui.info.text()
            worker = self.ui.lbl_name.text()

            self.DB.add_order(service, client, service_cost, kompl, kompl_cost, summary, info,
worker)
            # pine = 'order'
            self.info_window()
            self.ui.info.setText('')

        def add_wh(self):
            type = self.ui.edit_type.text()
            name = self.ui.edit_name.text()
            count = self.ui.edit_count.value()
            cost = self.ui.edit_cost.value()
            worker = self.ui.lbl_name_2.text()

            self.DB.add_wh(type, name, count, cost, worker)
            # pine = 'wh'
            self.info_window()

        def orders(self):
            orders = Order()
            orders.setWindowTitle('Заказы')

            orders.exec()

        def warehouse(self):
            warehouse = Warehouse()
            warehouse.setWindowTitle('Склад')

            warehouse.exec()

        def services(self):
            services = Service()
            services.setWindowTitle('Услуги')

            services.exec()

        def build_combobox_client(self):
```

```
        clients = self.DB.get_client()
        self.client_box.clear()
        if self.client_box is not None:
            self.client_box.addItems(clients)

    def build_combobox_service(self):
        services = self.DB.get_service()
        self.usluga_box.clear()
        if self.usluga_box is not None:
            self.usluga_box.addItems(services)

    def build_serv_cost(self):
        self.usl_cost.clear()
        self.ui.usl_cost.setText(str(self.DB.get_serv_c(self.ui.usluga_box.currentText())))
        self.usl_cost.update()
        print()

    def build_combobox_kompl(self):
        kompls = self.DB.get_kompl()
        self.kompl_box.clear()
        if self.kompl_box is not None:
            self.kompl_box.addItems(kompls)

    def build_kompl_cost(self):
        self.kompl_cost.clear()

self.ui.kompl_cost.setText(str(self.DB.get_kompl_c(self.ui.kompl_box.currentText())))
        self.kompl_cost.update()
        print()

    def info_window(self):
        info = Info()
        info.setWindowTitle('Информация')

        info.exec()


class DataBase():
    def __init__(self):
        self.con = sqlite3.connect('stok.db')

    def get_order(self):
        cur = self.con.cursor()
        cur.execute("SELECT * FROM order1")
        return cur.fetchall()

    def get_warehouse(self):
        cur = self.con.cursor()
        cur.execute("SELECT * FROM warehouse")
        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, client, serv_cost, kompl, kompl_cost, summ, info,worker):
        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,?,?,?,?,?,?,?,?,?,?,?)""",
(date, times, client, service,
                                                serv_cost, kompl, kompl_cost, summ,
                                                info, "Новый заказ", worker))
            self.con.commit()
            cur.close()
        except sqlite3.Error as error:
            print("Ошибка при работе с SQLite", error)

    def add_wh(self, type, name, count, cost, worker):
        now = datetime.now()
        id = 1
        try:
            cur = self.con.cursor()
            cur.execute("""INSERT INTO warehouse VALUES (NULL,?,?,?,?,?)""", (type,
name, count, cost, worker))
            self.con.commit()
            cur.close()
        except sqlite3.Error as error:
            print("Ошибка при работе с SQLite", error)

    def get_client(self):
        clients = []
        cursor = self.con.cursor()
        cursor.execute(f"SELECT `name` FROM client")
        rows = cursor.fetchall()

        for i in rows:
            clients.append(str(i)[2:-3])
        return clients
```

```
    def get_service(self):
        services = []
        cursor = self.con.cursor()
        cursor.execute(f"SELECT `Название` FROM usluga")
        rows = cursor.fetchall()

        for i in rows:
            services.append(str(i)[2:-3])
        return services

    def get_serv_c(self, service):
        cur = self.con.cursor()
        cur.execute(f'SELECT Цена FROM usluga WHERE Название="{service}"')
        serv_c = cur.fetchall()
        cur.close()
        return serv_c

    def get_kompl(self):
        kompls = []
        cursor = self.con.cursor()
        cursor.execute(f"SELECT `Наименование` FROM warehouse")
        rows = cursor.fetchall()

        for i in rows:
            kompls.append(str(i)[2:-3])
        return kompls

    def get_kompl_c(self, kompl):
        cur = self.con.cursor()
        cur.execute(f'SELECT Стоимость FROM warehouse WHERE
Наименование="{kompl}"')
        kompl_c = cur.fetchall()
        cur.close()
        return kompl_c


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = Auth()

    app.exec()
```

ПРИЛОЖЕНИЕ Г

Исходный код тестов

test_main.py

```
import sys
from unittest import TestCase

from PyQt5 import QtCore
from PyQt5.QtTest import QTest
from PyQt5.QtWidgets import QApplication

from main import Window


class TestPush(TestCase):
    def setUp(self):
        self.qapp = QApplication(sys.argv)
        self.window1 = Window('Тестер', '1')
        self.window2 = Window('Кладовщик', 'Федоров Федор Федорович')

    def test_push_order(self):
        btn_add = self.window1.ui.btn_add_order
        self.window1.ui.lbl_name.setText('Тестер')
        QTest.mouseClick(btn_add, QtCore.Qt.MouseButton.LeftButton)

    def test_push_wh(self):
        btn_add = self.window2.ui.btn_add_wh
        self.window2.ui.edit_type.setText('test')
        self.window2.ui.edit_name.setText('test')
        self.window2.ui.edit_count.setValue(99)
        self.window2.ui.edit_cost.setValue(99)
        self.window2.ui.lbl_name_2.setText('Федоров Федор Федорович')

        QTest.mouseClick(btn_add, QtCore.Qt.MouseButton.LeftButton)
```