|  |
| --- |
| **http://www.rags.ru/upload/iblock/41c/gerb-ranh_color.png**  Федеральное государственное бюджетное образовательное учреждение высшего образования  **«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**  КОЛЛЕДЖ МНОГОУРОВНЕВОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ |

**КУРСОВАЯ РАБОТА**

По дисциплине «МДК.01.01 Разработка программных модулей»

По теме «Разработка программного модуля системы 'Записная книжка'»

Выполнила:

студентка группы 34ИС-21

Абросимова Лада Сергеевна

Проверил:

преподаватель

Назаров Владимир Юрьевич

Москва, 2023

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc153296048)

[**1** **Анализ предметной области** 5](#_Toc153296049)

[**2** **ПРОЕКТИРОВАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ** 7](#_Toc153296050)

[2. 1 Разработка требований к проекту 7](#_Toc153296051)

[2.2 Построение диаграммы использования 8](#_Toc153296052)

[2.3 Разработка сценария проекта 12](#_Toc153296053)

[2.4 Построение диаграммы классов 13](#_Toc153296054)

[2.5 Проектирование базы данных 14](#_Toc153296055)

[**3** **РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ** 17](#_Toc153296056)

[3.1 Описание среды разработки 17](#_Toc153296057)

[3.2 Обоснование выбора инструментария по разработке 17](#_Toc153296058)

[3.3 Описание пользовательского интерфейса 18](#_Toc153296059)

[3.4 Методы класс ViewContactDialog 22](#_Toc153296060)

[3. 5 Тестирование 25](#_Toc153296061)

[**ЗАКЛЮЧЕНИЕ** 28](#_Toc153296062)

[**Список используемой литературы** 29](#_Toc153296063)

[**ПРИЛОЖЕНИЕ 1** 30](#_Toc153296064)

[**ПРИЛОЖЕНИЕ 2** 31](#_Toc153296065)

[**ПРИЛОЖЕНИЕ 3** 33](#_Toc153296066)

[**ПРИЛОЖЕНИЕ 4** 35](#_Toc153296067)

[**ПРИЛОЖЕНИЕ 5** 36](#_Toc153296068)

[**ПРИЛОЖЕНИЕ 6** 39](#_Toc153296069)

[**ПРИЛОЖЕНИЕ 7** 42](#_Toc153296070)

# 

# ВВЕДЕНИЕ

Сегодня, в эпоху цифровых технологий и высокой мобильности, эффективное управление контактами и заметками стало важной задачей для многих людей. Электронные записные книжки и приложения, используемые для записи контактов и важных заметок, очень популярны и предлагают легкий доступ к информации и удобное взаимодействие.

Целью моего проекта является создание приложения "Записная Книжка" на языке Python с использованием фреймворка PyQt. Это оконное приложение, которое будет предоставлять пользователям интуитивно понятный и многофункциональный инструмент для управления своими контактами и заметками. Суть этого проекта заключается в создании гибкой и легко настраиваемой системы, способной удовлетворить потребности широкого круга пользователей.

В рамках данной работы будет выполнена разработка программного модуля, обеспечивающего следующие ключевые функции:

1. Добавление и хранение контактов, включая информацию о имени, фамилии, номере телефона и дополнительных заметках(примечаниях).
2. Поиск контактов для быстрого доступа к нужной информации.
3. Создание и редактирование текстовых заметок, привязанных к определенным контактам.
4. Обеспечение безопасности данных и их резервного копирования.

Основной аудиторией проекта являются пользователи, которым необходимо эффективное управление своими контактами и заметками, начиная от частных лиц до предпринимателей.

Создание программ, применяемых в разных областях, от повседневных задач до промышленности, всё ещё остаётся актуальной задачей. Новые технологии программирования и среды разработки значительно упрощают этот процесс. Один из таких универсальных языков программирования - Python, широко применяемый для автоматизации, обучения, анализа данных и других областей. Его гибкость и мощные инструменты делают его одним из наиболее популярных языков в области программирования.

В мире образования Python становится первым языком программирования во многих учебных заведениях. Он успешно используется не только для школьных задач, но и для серьёзных проектов, включая обработку данных, машинное обучение, веб-разработку и автоматизацию процессов. Python обладает обширными методическими материалами, что делает его отличным выбором для обучения программированию.

В современном мире, где объемы информации постоянно растут, разработка удобного и надежного инструмента для организации контактов и заметок представляет значительную актуальность. Данный проект имеет потенциал упростить повседневную жизнь многих людей и улучшить управление информацией. Помимо этого, он предоставляет мне возможность овладеть навыками разработки программного обеспечения, что делает его не только актуальным, но и полезным для моего профессионального роста.

Этот проект позволит мне применить навыки программирования на Python и PyQt для создания полезного приложения, сочетая практическую реализацию и изучение основных технологий разработки программного обеспечения.

В рамках работы я намерена:

- Получить понимание основ Python и фреймворка PyQt для создания прикладных программ.

- Разработать механизм ввода и хранения информации о контактах и заметках.

- Реализовать приложение "Записная Книжка", провести тестирование и отладку функционала.

# Анализ предметной области

Концепции структурного и объектно-ориентированного программирования продолжают играть ключевую роль в современной разработке ПО. Парадигма объектно-ориентированного программирования особенно ценится за её способность упрощать представление сложных реальных процессов. При разработке программного обеспечения акцент на структурированность и поэтапное внедрение функциональности позволяет создавать код, который легко читается, отлаживается и содержит модули, пригодные для повторного использования, что сокращает расходы на разработку и сопровождение.

В моём проекте я стремлюсь совместить простоту с адаптивностью. Структурный подход ценится за его непосредственность и лёгкость внесения изменений, тогда как объектно-ориентированная методология предлагает модульность и интуитивное структурирование. Моя цель — разработать приложение с интуитивно понятным интерфейсом, основываясь на объектно-ориентированном дизайне, где данные и их обработка будут заключены внутри объектов для удобства использования.

Стоит подчеркнуть, что в крупных проектах, подобных моему, операция присваивания может уступать место вызовам методов и процедур. Отсюда следует важность точного определения и вызова соответствующих подпрограмм. Объектно-ориентированная методология позволяет чётко определять классы и их взаимосвязи, упрощая работу с обширными наборами данных и алгоритмов.

Основной характеристикой моего проекта является его модульное строение, где каждый модуль контролирует собственные данные и алгоритмы, предоставляя доступ к ним через объекты с конкретными именами. Это обеспечит необходимую гибкость и упростит процессы разработки и техподдержки.

Нельзя забывать о том, что как структурное, так и объектно-ориентированное программирование требуют глубокого понимания базовых концепций, включая циклы, условные операторы и переменные. Эти фундаментальные принципы будут применяться в моём проекте для создания приложения, которое будет одновременно интуитивно понятным и высокопроизводительным. Основные различия между ООП и структурным программирование представлены в таблице 1.

Таблица 1. Различия между ООП и структурным программированием

|  |  |  |
| --- | --- | --- |
| Характеристика | Объектно-ориентированное программирование | Структурное программирование |
| Основная концепция | Основано на объектах и их взаимодействии | Основано на последовательности операций |
| Подход к данным | Данные и методы их обработки объединены в объектах | Данные и операции с ними разделены |
| Модульность | Предоставляет модульность и инкапсуляцию | Модульность часто отсутствует, код часто линеен |
| Уровень абстракции | Позволяет создавать более высокий уровень абстракции и переиспользовать код | Обычно менее абстрактно и менее поддаётся переиспользованию |
| Пример языков | Python, Java, C++ | C, Pascal, Fortran |
| Примеры в приложениях | Веб-разработка, создание игр, приложений для мобильных устройств | Написание скриптов, разработка баз данных |

Исходя из целей данной курсовой работы, в качестве технологии для проектирования программного средства выбрана объектно-ориентированная парадигма, позволяющая хранить данные прикладной программы «Записная книжка» в виде объектов классов, представляющих записи записной книги.

# 2 ПРОЕКТИРОВАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

# 2. 1 Разработка требований к проекту

**Цель:**

Целью данной курсовой работы является разработка программного модуля системы "Записная книжка", который позволит пользователям вести электронную записную книжку для хранения контактов и важных заметок.

**Область применения:**

Программный модуль будет разработан для использования на персональных компьютерах под управлением операционных систем Windows, macOS или Linux.

**Требования к функциональности**

**Основные функции:**

1. **Добавление контактов**

Пользователь должен иметь возможность добавлять контакты в записную книжку, включая следующие поля:

- ФИО

- Номер телефона

- Дополнительные заметки(примечания)

1. **Просмотр и редактирование контактов**

Пользователь должен иметь возможность просматривать, редактировать и удалять существующие контакты.

1. **Поиск контактов**

Пользователь должен иметь возможность выполнять поиск контактов по имени или номеру телефона.

1. **Создание и редактирование заметок**

Пользователь должен иметь возможность создавать и редактировать текстовые примечания, привязанные к определенному контакту.

**Интерфейс:**

**Графический интерфейс**

Программный модуль должен иметь интуитивно понятный графический интерфейс, позволяющий пользователям легко выполнять все вышеперечисленные функции.

**Требования к надежности:**

**Резервное копирование данных**

Система должна автоматически выполнять резервное копирование данных, чтобы предотвратить потерю информации.

**Защита данных:**

Для обеспечения безопасности данных система должна предусматривать механизм аутентификации и авторизации пользователей.

**Требования к производительности:**

**Отклик интерфейса**

Интерфейс программного модуля должен быть отзывчивым, без заметных задержек при выполнении действий пользователя.

# 2.2 Построение диаграммы использования

***Деятели:***

- Пользователь

- Приложение

***Прецеденты:***

1. *Создание контакта*

* *ПОЛЬЗОВАТЕЛЬ в приложении добавляет новый контакт, вводя информацию: имя, номер телефона, заметку.*
* *ПРИЛОЖЕНИЕ сохраняет контакт в базе данных и отправляет уведомление ПОЛЬЗОВАТЕЛЮ об успешном добавлении.*

1. *Редактирование контакта*

* *ПОЛЬЗОВАТЕЛЬ вносит изменения в существующий контакт, занося новые данные в приложении.*
* *ПРИЛОЖЕНИЕ обновляет информацию о контакте в базе данных и уведомляет ПОЛЬЗОВАТЕЛЯ о успешном редактировании.*

1. *Удаление контакта*

* *ПОЛЬЗОВАТЕЛЬ отправляет запрос на удаление контакта, указывая его.*
* *ПРИЛОЖЕНИЕ удаляет контакт из базы данных и сообщает ПОЛЬЗОВАТЕЛЮ об успешном удалении.*

1. *Поиск контакта*

* *ПОЛЬЗОВАТЕЛЬ запрашивает в приложении контакт по имени.*
* *ПРИЛОЖЕНИЕ находит соответствующий контакт и отправляет его данные ПОЛЬЗОВАТЕЛЮ.*

*5. Подтверждение действия*

- ПРИЛОЖЕНИЕ отправляет ПОЛЬЗОВАТЕЛЮ сообщение с подтверждением действия.

- ПОЛЬЗОВАТЕЛЬ подтверждает или не подтверждает действие.

***Отношения между прецедентами и деятелями.***

- *Ассоциация* между деятелем "Пользователь" и прецедентами "Создание контакта", "Редактирование контакта", "Удаление контакта", "Поиск контакта". (Рисунок 1)

Изображение выглядит как текст, диаграмма, круг, линия

Автоматически созданное описание

Рисунок 1. Отношение ассоциации между деятелями и прецедентами

- *Включение* между прецедентами "Редактирование контакта", "Поиск контакта", "Удаление контакта" и "Создание контакта". (Рисунок 2)

Изображение выглядит как текст, круг, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 2. Отношение включения между прецедентами

- *Расширение* между прецедентами "Создание контакта", "Удаление контакта", "Редактирование контакта" и прецедентом "Подтверждение действия", который включает в себя отправку подтверждения пользователю. (Рисунок 3)

Изображение выглядит как текст, круг, диаграмма, снимок экрана

Автоматически созданное описание

Рисунок 3. Отношение расширения между прецедентами

- *Обобщение* между прецедентами "Создание контакта", "Поиск контакта", "Удаление контакта", "Редактирование контакта" и прецедентом "Управление контактами", который описывает общую функциональность приложения. (Рисунок 4)

Изображение выглядит как круг, зарисовка, диаграмма, линия

Автоматически созданное описание

Рисунок 4. Отношение обобщения между прецедентами

**Изображение выглядит как диаграмма, круг, линия, зарисовка

Автоматически созданное описание**

Рисунок 5. Основная диаграмма вариантов использования

**Изображение выглядит как текст, диаграмма, круг, линия

Автоматически созданное описание**

Рисунок 6. Диаграмма вариантов использования, поясняющая основной прецедент Управление контактами

# 2.3 Разработка сценария проекта

После открытия приложения "Записная книжка" перед пользователем появляется окно авторизации, где он вводит свои учетные данные: имя пользователя и пароль. После успешной авторизации пользователь попадаете в основное окно приложения.

На главном экране отображается список контактов. Пользователь может просматривать их, вносить изменения, добавлять новые или удалять ненужные контакты.

Если он решите добавить новый контакт, при нажатии кнопки "Добавить контакт", откроется окно, где предложено заполните данные о новом контакте: имя, номер телефона и краткие заметки. После сохранения контакт автоматически добавится в список.

Чтобы просмотреть детали конкретного контакта, достаточно выбрать его из списка. Появится окно с подробной информацией о выбранном контакте: его имя, номер телефона и заметки.

Пользователь также может редактировать контакты, изменяя или дополняя информацию. Если необходимо удалить контакт, можно сделать это, выбрав соответствующую опцию. Программа запросит подтверждение, чтобы избежать случайного удаления.

Помимо этого, у приложения есть функция поиска. В верхней части главного окна находится поле для ввода текста. Пользователь может ввести в него ключевые слова или часть имени контакта, и программа отобразит только те контакты, в названиях которых есть введенный текст.

Каждое изменение в записной книжке моментально отображается в приложении, позволяя пользователю легко и эффективно управлять списком его контактов.

На рисунке 7 приведен сценарий использования программы пользователем.

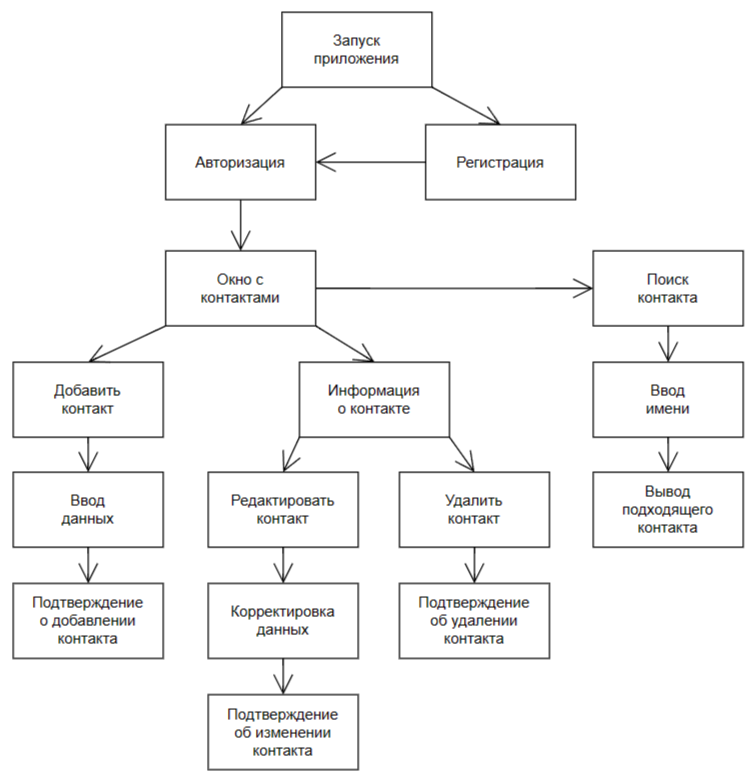
****

Рисунок 7. Сценарий использования программы пользователем

# 2.4 Построение диаграммы классов

**AddressBookApp** — это основное окно приложения, которое позволяет пользователям управлять своими контактами. Оно взаимодействует с несколькими окнами:

- LoginDialog используется для входа в систему. Если авторизация прошла успешно, пользователь получает доступ к своей записной книжке.

- AddEditContactDialog предоставляет возможность добавлять или редактировать контакты. Это окно встроено в приложение для комфортного управления контактами.

- ViewContactDialog — еще одно окно, которое позволяет просматривать информацию о конкретном контакте подробнее.

**AddEditContactDialog** является окном, где пользователь добавляет или изменяет информацию о контактах. Здесь происходит взаимодействие с объектами класса Contact, представляющими собой сами контакты.

**ViewContactDialog** используется для показа детальной информации о выбранном контакте. Он связан с классом Contact, чтобы получить и отобразить информацию о контакте.

**LoginDialog** — окно для авторизации пользователей. Оно работает с базой данных, проверяя имя пользователя и пароль для входа.

**Contact** отвечает за хранение информации о контактах и взаимодействие с базой данных для сохранения и получения информации о них. Таким образом, эти окна и классы сотрудничают между собой и с базой данных, обеспечивая пользователям удобное управление их контактами в приложении.

Диаграмма классов представлена на рисунке 8.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 8. Диаграмма классов

# 2.5 Проектирование базы данных

В процессе проектирования базы данных для приложения "Записная книжка" акцент делается на оптимальной структуре хранения информации о контактах пользователей.

**Таблица "Пользователи"** является основной, она содержит уникальные идентификаторы пользователей (*id*), их имена (*username*) и пароли (*password*). Структура данной сущности представлена на таблице 2.

Таблица 2 – Атрибуты сущности «Пользователи»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип, домен |
| id | Уникальный идентификатор | Целое число, Ключевой |
| username | Имя пользователя | varchar(50) |
| password | Пароль пользователя | varchar(50) |

**Таблица "Контакты"** связана с таблицей пользователей через *user\_id* для привязки контактов к определенным пользователям. Она содержит данные о контактах, такие как идентификатор контакта (*id*), имя контакта (*contact\_name*), номер телефона (*phone\_number*) и поле для заметок (*notes*). Структура сущности "Контакты" представлена на таблице 3.

Таблица 3 – Атрибуты сущности «Контакты»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип, домен |
| id | Уникальный идентификатор | Целое число, Ключевой |
| user\_id | Уникальный идентификатор пользователя | Целое число |
| name | Имя контакта | varchar(100) |
| phone\_number | Номер телефона контакта | varchar(20) |
| notes | Краткие заметки | Текст |

Структура базы данных приведена в виде диаграммы «Сущность-связь» в нотации Мартина на рисунке 9. Такая структура позволяет эффективно управлять контактами пользователей, обеспечивая безопасное хранение информации и облегчая поиск нужных данных. Отношения между таблицами создаются для установления связей между контактами и их владельцами, что обеспечивает целостность данных и удобство управления информацией.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 9. Диаграмма сущность-связь в нотации Чена

# 3 РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

# 3.1 Описание среды разработки

1. Среда разработки: PyCharm Community Edition 2023

Интегрированная среда разработки (IDE) для языка программирования Python. Она предоставляет обширный набор инструментов и функциональности, облегчающих процесс разработки на Python.

2. Язык программирования: Python

Высокоуровневый интерпретируемый язык программирования с чистым и понятным синтаксисом. Он известен своей простотой в изучении и широким применением в различных областях, от веб-разработки до научных вычислений.

3. Библиотека mysql.connector

Это библиотека для Python, предназначенная для взаимодействия с базами данных MySQL с использованием стандартного API Python Database API (PEP 249). Она обеспечивает простой и удобный способ подключения к MySQL и выполнения запросов к базе данных.

4. Другие инструменты и библиотеки:

PyQT — это набор инструментов для разработки графических интерфейсов пользователя на Python. Он базируется на библиотеке Qt и предоставляет обширные возможности для создания интерактивных и функциональных приложений.

# 3.2 Обоснование выбора инструментария по разработке

* PyCharm обеспечивает удобную навигацию по проекту и возможности рефакторинга кода, что упрощает его поддержку и развитие. Интеллектуальное автодополнение и анализ кода позволяют улучшить процесс написания кода, предлагая подсказки и исправления.
* Синтаксис Python легок для понимания и написания, что делает его привлекательным для начинающих и опытных программистов. Python имеет обширное сообщество и богатый экосистем, включая библиотеки для различных задач, таких как анализ данных, машинное обучение, веб-разработка и многое другое. Код, написанный на Python, может быть запущен на различных платформах без изменений (при условии, что используются не зависящие от платформы библиотеки).
* PyQT предоставляет множество инструментов для создания настраиваемых и стильных пользовательских интерфейсов. Библиотека содержит множество виджетов и элементов управления для создания разнообразных интерфейсов. PyQT использует событийную модель, что облегчает обработку действий пользователя и реакцию на события в приложении.
* mysql.connector специально разработана для взаимодействия с базой данных MySQL, обеспечивая полноценную поддержку всех основных функций и операций MySQL. Поддержка параметризованных запросов повышает безопасность взаимодействия с базой данных, предотвращая атаки SQL-инъекций. Библиотека предлагает оптимизированные методы работы с данными, что способствует более эффективной обработке запросов и данных. Поддержка различных типов данных MySQL и возможность управления транзакциями делают библиотеку гибкой и адаптивной к различным задачам и сценариям использования.

# 3.3 Описание пользовательского интерфейса

Как только пользователь открывает приложение перед ним открывается окно авторизации (Рисунок 10) с возможность регистрации (Рисунок11), реализованные в классах LoginDialog (Приложение 6) и RegistrationDialog (Приложение 4).

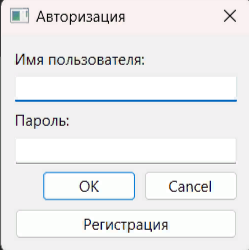
****

Рисунок 10. Окно авторизации

**Изображение выглядит как текст, Шрифт, число, линия

Автоматически созданное описание**

Рисунок 11. Окно регистрации

Код, который управляет регистрацией и аутентификацией пользователей, играет важную роль в обеспечении безопасности приложения для работы с контактами. Модуль регистрации позволяет пользователям создавать учетные записи, вводя уникальное имя пользователя и пароль.

Оба эти модуля осуществляют взаимодействие с базой данных MySQL с помощью библиотеки `mysql.connector`. Модуль регистрации вставляет новые записи (имя пользователя и пароль) в таблицу `users`, а модуль аутентификации осуществляет поиск существующих записей, соответствующих введенным учетным данным.

После успешной аутентификации приложение для работы с контактами (`AddressBookApp`) получает уникальный идентификатор пользователя из базы данных. Он используется для получения списка контактов, принадлежащих конкретному пользователю, и отображения их в приложении (Рисунок 12).

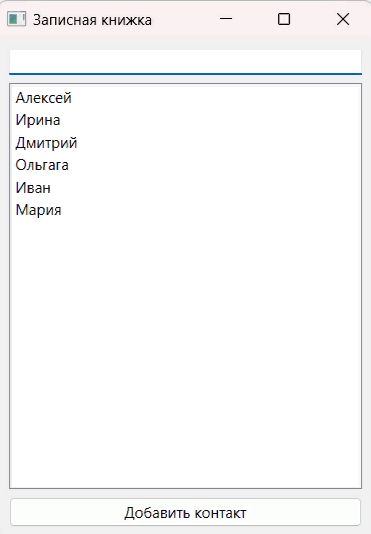
****

Рисунок 12. Главное окно приложения

Первоначальный аспект данного пользовательского интерфейса представлен полем поиска, обеспечивающим возможность поиска контактов по их именам в базе данных телефонной книги. Этот функционал представляет собой средство быстрого доступа к нужным записям, повышая эффективность пользовательского опыта. Поиск контактов осуществляется в классе AddressBookApp (Приложение 7). Этот класс содержит метод ‘search\_contacts’, который вызывается при изменении текста в поле поиска (Рисунок 13).



Рисунок 13. Поиск контакта

Центральным элементом интерфейса является список контактов, предоставляющий пользователю обзор доступных контактов по именам. Возможность выбора конкретного контакта из списка позволяет получить дополнительные сведения (Рисунок 14).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 14. Дополнительная информация о контакте

Интерфейс также предусматривает кнопку для добавления нового контакта в книгу, обеспечивая возможность моментального добавления информации о новых контактах (Рисунок 15).

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 15. Добавление контакта

Существует также диалоговое окно, предназначенное для ввода и редактирования информации о контакте. Здесь пользователь может ввести имя, номер телефона и примечания, а также сохранить или отменить операцию (Рисунок 16).

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 16. Редактирование контакта

Неотъемлемой частью пользовательского интерфейса является диалоговое окно подтверждения удаления контакта, где пользователь получает запрос на подтверждение удаления контакта с доступными кнопками для принятия решения (Рисунок 17).

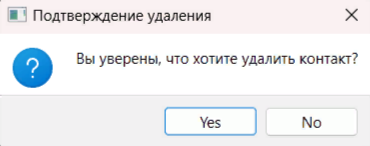
****

Рисунок 17. Подтверждение удаления контакта

# 3.4 Методы класс ViewContactDialog

Одним из основных классов в программе является класс ViewContactDialog, (Приложение 5) отвечающий за отображение информации о контакте и реализующий ряд методов для взаимодействия с этой информацией.

Метод **get\_contact\_id\_from\_db** выполняет запрос к базе данных для получения идентификатора контакта по его имени и номеру телефона. Этот метод используется для получения идентификатора контакта перед его редактированием. Листинг этого метода представлен ниже.

def get\_contact\_id\_from\_db(self, name, phone\_number):  
 contact\_id = None  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 select\_query = "SELECT id FROM contacts WHERE name = %s AND phone\_number = %s"  
 cursor.execute(select\_query, (name, phone\_number))  
 result = cursor.fetchone()  
 if result:  
 contact\_id = result[0]  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
 return contact\_id

Метод **edit\_contact** вызывается при нажатии кнопки "Редактировать контакт". Он создает диалоговое окно для редактирования контакта (AddEditContactDialog), обновляет отображаемую информацию о контакте и обновляет данные в базе данных. После успешного редактирования выводится сообщение об успешном изменении контакта.

def edit\_contact(self):  
 dialog = AddEditContactDialog(parent=self, contact=self.contact, contact\_id=self.contact\_id)  
 if dialog.exec() == QDialog.DialogCode.Accepted:  
 updated\_contact = dialog.get\_contact\_info()  
 self.name\_label.setText(f"Имя: {updated\_contact.name}")  
 self.phone\_label.setText(f"Телефон: {updated\_contact.phone\_number}")  
 self.notes\_label.setText(f"Примечания: {updated\_contact.notes}")  
 self.contact = updated\_contact  
 self.contact\_id = self.get\_contact\_id\_from\_db(updated\_contact.name, updated\_contact.phone\_number)  
  
 QMessageBox.information(  
 self, "Успех", "Контакт успешно изменен"  
 )  
 self.parent().update\_contacts\_list()   
  
 self.accept()

Метод **delete\_contact** вызывается при нажатии кнопки "Удалить контакт". Он запрашивает у пользователя подтверждение удаления контакта с помощью диалогового окна. При подтверждении контакт удаляется из базы данных, а затем из списка отображаемых контактов. В случае ошибки при удалении выводится сообщение об ошибке.

def delete\_contact(self):  
 confirmation = QMessageBox.question(  
 self, "Подтверждение удаления", "Вы уверены, что хотите удалить контакт?",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No  
 )  
  
 if confirmation == QMessageBox.StandardButton.Yes:  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 delete\_query = "DELETE FROM contacts WHERE name = %s AND phone\_number = %s AND notes = %s"  
 cursor.execute(delete\_query, (self.contact.name, self.contact.phone\_number, self.contact.notes))  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 QMessageBox.information(  
 self, "Успех", "Контакт успешно удален"  
 )  
 self.parent().remove\_contact\_from\_list(self.contact) # Remove from displayed list  
  
 self.accept()  
  
 except mysql.connector.Error as error:  
 QMessageBox.warning(  
 self, "Ошибка", f"Ошибка при удалении контакта: {error}"  
 )  
 else:  
 pass

# 3. 5 Тестирование

Итогом процесса тестирования приложения становится комплект тест-кейсов, на которых осуществляется верификация функционала разработанного программного продукта. Эти тест-кейсы включают в себя набор входных данных, которые подаются на вход программе, и ожидаемые результаты, которые программа должна выдать в ответ. Для наглядной демонстрации возможностей программы необходимо подготовить тестовые сценарии. Объем данных в этих сценариях должен быть достаточен, чтобы показать полный спектр функциональности программы, включая проверку корректности работы и обработку нестандартных ситуаций.

Для тестирования программы необходимо запустить программу. На экране появится окно авторизации. а после ввода необходимых данных пользователь попадает на главное окно приложения.

Если пользователь уже был зарегистрирован и производил различные манипуляции с контактами, все данные отобразятся в главном окне из базы данных (Рисунок 12).

Результат поиска по имени продемонстрирован на рисунке 13, результат добавления нескольких контактов приведен на рисунке 18 соответственно.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

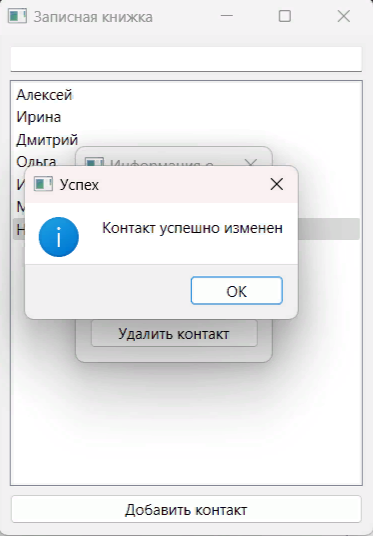
Рисунок 18. Добавление нескольких контактов в записную книжку

Результат удаления одного из контактов приведен на рисунке 19. Результат редактирования контакта заметен на рисунке 20.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 19. Результат удаления контакта

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 20. Результат редактирования контакта

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы было разработано приложение «Записная книжка», предназначенное для удобного хранения и управления контактной информацией; самостоятельно изучены функции и возможности интегрированной среды разработки PyCharm Community Edition 2023, а также закреплены навыки создания приложений в среде программирования PyCharm на языке Python и фреймворке PyQt. Разработка программного средства поспособствовала систематизации, закреплению и расширению знаний, полученных при изучении курса.

Закрепление теоретического материала происходило в ходе решения поставленной задачи – разработки, отладки и тестирование программного приложения. Итогом явилось написание курсовая работа, при выполнении которой также были закреплены основные знания по написанию отчетов в соответствии с государственными стандартами.

Разработанное приложение соответствует поставленным задачам. Оно обладает интуитивно понятным графическим интерфейсом, который предоставляет всю необходимую информацию и поддержку, делая его удобным для использования. Таким образом, основная цель курсовой работы была достигнута в полном объеме.

# Список используемой литературы

1. База данных MySQL: документация официального сайта.
2. Официальная документация по PyQt.
3. Официальный сайт Python - документация и руководства.
4. Дополнительные интернет-источники.

# ПРИЛОЖЕНИЕ 1

Листинг 1. Файл main.py

import sys  
from PyQt6.QtWidgets import QApplication, QDialog  
from login\_dialog import LoginDialog  
from address\_book\_app import AddressBookApp  
  
  
class Contact:  
 def \_\_init\_\_(self, name, phone\_number, notes):  
 self.name = name  
 self.phone\_number = phone\_number  
 self.notes = notes  
  
  
def main():  
 app = QApplication(sys.argv)  
  
 login\_dialog = LoginDialog()  
 if not login\_dialog.exec() == QDialog.DialogCode.Accepted:  
 sys.exit(-1)  
  
 user\_id = login\_dialog.user\_id  
  
 window = AddressBookApp(user\_id)  
 window.show()  
 sys.exit(app.exec())  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# ПРИЛОЖЕНИЕ 2

Листинг 2. Файл database.py

import mysql.connector  
  
  
def register\_user\_to\_db(username, password):  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 insert\_query = "INSERT INTO users (username, password) VALUES (%s, %s)"  
 user\_data = (username, password)  
 cursor.execute(insert\_query, user\_data)  
 conn.commit()  
 print("Пользователь зарегистрирован успешно!")  
 cursor.close()  
 conn.close()  
 else:  
 print("Не удалось подключиться к базе данных.")  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
  
def add\_contact\_to\_db(user\_id, name, phone\_number, notes):  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 insert\_query = "INSERT INTO contacts (user\_id, name, phone\_number, notes) VALUES (%s, %s, %s, %s)"  
 contact\_data = (user\_id, name, phone\_number, notes)  
 cursor.execute(insert\_query, contact\_data)  
 conn.commit()  
 print("Контакт добавлен успешно!")  
 cursor.close()  
 conn.close()  
 else:  
 print("Не удалось подключиться к базе данных.")

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 2**

except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
  
def update\_contact\_in\_db(contact\_id, name, phone\_number, notes):  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 update\_query = "UPDATE contacts SET name = %s, phone\_number = %s, notes = %s WHERE id = %s"  
 contact\_data = (name, phone\_number, notes, contact\_id)  
 cursor.execute(update\_query, contact\_data)  
 conn.commit()  
 print("Контакт обновлен успешно!")  
 cursor.close()  
 conn.close()  
 else:  
 print("Не удалось подключиться к базе данных.")  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)

# ПРИЛОЖЕНИЕ 3

Листинг 3. Файл add\_edit\_contact\_dialog.py

from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit, QTextEdit, QDialogButtonBox, QPushButton  
from database import update\_contact\_in\_db, add\_contact\_to\_db  
from main import Contact  
  
  
class AddEditContactDialog(QDialog):  
 def \_\_init\_\_(self, parent=None, contact=None, user\_id=None, contact\_id=None):  
 super().\_\_init\_\_(parent)  
 self.user\_id = user\_id  
 self.contact\_id = contact\_id  
 self.setWindowTitle("Добавить контакт" if not contact else "Редактировать контакт")  
  
 layout = QVBoxLayout()  
  
 self.name\_input = QLineEdit()  
 self.phone\_input = QLineEdit()  
 self.notes\_input = QTextEdit()  
  
 if contact:  
 self.name\_input.setText(contact.name)  
 self.phone\_input.setText(contact.phone\_number)  
 self.notes\_input.setPlainText(contact.notes)  
  
 layout.addWidget(QLabel("Имя:"))  
 layout.addWidget(self.name\_input)  
 layout.addWidget(QLabel("Телефон:"))  
 layout.addWidget(self.phone\_input)  
 layout.addWidget(QLabel("Примечания:"))  
 layout.addWidget(self.notes\_input)  
  
 buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)  
 buttons.accepted.connect(self.accept)  
 buttons.rejected.connect(self.reject)  
 layout.addWidget(buttons)  
  
 self.setLayout(layout)  
  
 def get\_contact\_info(self):  
 return Contact(self.name\_input.text(), self.phone\_input.text(), self.notes\_input.toPlainText())  
  
 def accept(self):  
 contact\_info = self.get\_contact\_info()

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 3**

if self.contact\_id: # Если contact\_id существует, это операция редактирования   
 update\_contact\_in\_db(self.contact\_id, contact\_info.name, contact\_info.phone\_number, contact\_info.notes)  
 else:  
 add\_contact\_to\_db(self.user\_id, contact\_info.name, contact\_info.phone\_number, contact\_info.notes)  
 super().accept()

# ПРИЛОЖЕНИЕ 4

Листинг 4. Файл registration\_dialog.py

from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit, QDialogButtonBox, QMessageBox  
from database import register\_user\_to\_db  
  
  
class RegistrationDialog(QDialog):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Регистрация")  
  
 layout = QVBoxLayout()  
  
 self.username\_input = QLineEdit()  
 self.password\_input = QLineEdit()  
 self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)  
  
 buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)  
 buttons.accepted.connect(self.register\_user)  
 buttons.rejected.connect(self.reject)  
  
 layout.addWidget(QLabel("Имя пользователя:"))  
 layout.addWidget(self.username\_input)  
 layout.addWidget(QLabel("Пароль:"))  
 layout.addWidget(self.password\_input)  
 layout.addWidget(buttons)  
  
 self.setLayout(layout)  
  
 def register\_user(self):  
 username = self.username\_input.text()  
 password = self.password\_input.text()  
  
 if username and password:  
 register\_user\_to\_db(username, password)  
 QMessageBox.information(self, "Успешная регистрация", "Пользователь зарегистрирован")  
 self.accept()  
 else:  
 QMessageBox.warning(self, "Ошибка", "Имя пользователя и пароль не могут быть пустыми")

# ПРИЛОЖЕНИЕ 5

Листинг 5. Файл view\_contact\_dialog.py

from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel, QPushButton, QMessageBox  
import mysql.connector  
from add\_edit\_contact\_dialog import AddEditContactDialog  
  
  
class ViewContactDialog(QDialog):  
 def \_\_init\_\_(self, parent=None, contact=None):  
 super().\_\_init\_\_(parent)  
 self.setWindowTitle("Информация о контакте")  
  
 layout = QVBoxLayout()  
  
 self.name\_label = QLabel()  
 self.phone\_label = QLabel()  
 self.notes\_label = QLabel()  
  
 layout.addWidget(self.name\_label)  
 layout.addWidget(self.phone\_label)  
 layout.addWidget(self.notes\_label)  
  
 edit\_button = QPushButton("Редактировать контакт")  
 edit\_button.clicked.connect(self.edit\_contact)  
 layout.addWidget(edit\_button)  
  
 delete\_button = QPushButton("Удалить контакт")  
 delete\_button.clicked.connect(self.delete\_contact)  
 layout.addWidget(delete\_button)  
  
 self.setLayout(layout)  
  
 self.contact = contact  
 if contact:  
 self.name\_label.setText(f"Имя: {contact.name}")  
 self.phone\_label.setText(f"Телефон: {contact.phone\_number}")  
 self.notes\_label.setText(f"Примечания: {contact.notes}")  
 self.contact\_id = self.get\_contact\_id\_from\_db(contact.name, contact.phone\_number)  
  
 def get\_contact\_id\_from\_db(self, name, phone\_number):  
 contact\_id = None  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 5**

database='phones'  
 )

if conn.is\_connected():  
 cursor = conn.cursor()  
 select\_query = "SELECT id FROM contacts WHERE name = %s AND phone\_number = %s"  
 cursor.execute(select\_query, (name, phone\_number))  
 result = cursor.fetchone()  
 if result:  
 contact\_id = result[0]  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
 return contact\_id  
  
 def edit\_contact(self):  
 dialog = AddEditContactDialog(parent=self, contact=self.contact, contact\_id=self.contact\_id)  
 if dialog.exec() == QDialog.DialogCode.Accepted:  
 updated\_contact = dialog.get\_contact\_info()  
 self.name\_label.setText(f"Имя: {updated\_contact.name}")  
 self.phone\_label.setText(f"Телефон: {updated\_contact.phone\_number}")  
 self.notes\_label.setText(f"Примечания: {updated\_contact.notes}")  
 self.contact = updated\_contact  
 self.contact\_id = self.get\_contact\_id\_from\_db(updated\_contact.name, updated\_contact.phone\_number)  
  
 QMessageBox.information(  
 self, "Успех", "Контакт успешно изменен"  
 )  
 self.parent().update\_contacts\_list() # Обновление списка контактов  
  
 self.accept()  
  
 def delete\_contact(self):  
 confirmation = QMessageBox.question(  
 self, "Подтверждение удаления", "Вы уверены, что хотите удалить контакт?",  
 QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 5**

)  
  
 if confirmation == QMessageBox.StandardButton.Yes:

try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 delete\_query = "DELETE FROM contacts WHERE name = %s AND phone\_number = %s AND notes = %s"  
 cursor.execute(delete\_query, (self.contact.name, self.contact.phone\_number, self.contact.notes))  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 QMessageBox.information(  
 self, "Успех", "Контакт успешно удален"  
 )  
 self.parent().remove\_contact\_from\_list(self.contact) # Remove from displayed list  
  
 self.accept()  
  
 except mysql.connector.Error as error:  
 QMessageBox.warning(  
 self, "Ошибка", f"Ошибка при удалении контакта: {error}"  
 )  
 else:  
 pass

# ПРИЛОЖЕНИЕ 6

Листинг 6. Файл login\_dialog.py

from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit, QDialogButtonBox, QPushButton, QMessageBox  
import mysql.connector  
from registration\_dialog import RegistrationDialog  
  
  
class LoginDialog(QDialog):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.user\_id = None  
 self.setWindowTitle("Авторизация")  
  
 layout = QVBoxLayout()  
  
 self.username\_input = QLineEdit()  
 self.password\_input = QLineEdit()  
 self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)  
  
 buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)  
 buttons.accepted.connect(self.authenticate)  
 buttons.rejected.connect(self.reject)  
  
 layout.addWidget(QLabel("Имя пользователя:"))  
 layout.addWidget(self.username\_input)  
 layout.addWidget(QLabel("Пароль:"))  
 layout.addWidget(self.password\_input)  
 layout.addWidget(buttons)  
  
 register\_button = QPushButton("Регистрация")  
 register\_button.clicked.connect(self.show\_registration\_dialog)  
 layout.addWidget(register\_button)  
  
 self.setLayout(layout)  
  
 def authenticate(self):  
 username = self.username\_input.text()  
 password = self.password\_input.text()  
  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 6**

if conn.is\_connected():

cursor = conn.cursor()  
 select\_query = "SELECT id FROM users WHERE username = %s AND password = %s"  
 cursor.execute(select\_query, (username, password))  
 user\_id = cursor.fetchone()  
  
 if user\_id:  
 self.user\_id = user\_id[0] # Получаем фактический user\_id из результата запроса  
 self.accept()  
 else:  
 QMessageBox.warning(self, "Ошибка", "Неверное имя пользователя или пароль")  
 self.username\_input.clear()  
 self.password\_input.clear()  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
 def get\_user\_id(self, username):  
 user\_id = None  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 select\_query = "SELECT id FROM users WHERE username = %s"  
 cursor.execute(select\_query, (username,))  
 user\_id = cursor.fetchone()  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
 return user\_id[0] if user\_id else None

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 6**

def show\_registration\_dialog(self):  
 registration\_dialog = RegistrationDialog()  
 registration\_dialog.exec()

# ПРИЛОЖЕНИЕ 7

Листинг 7. Файл address\_book\_app.py

from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QListWidget, QPushButton, QDialog, QLineEdit  
import mysql.connector  
from main import Contact  
from view\_contact\_dialog import ViewContactDialog  
from add\_edit\_contact\_dialog import AddEditContactDialog  
  
  
class AddressBookApp(QMainWindow):  
 def \_\_init\_\_(self, user\_id):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Записная книжка")  
 self.resize(300, 400)  
  
 self.user\_id = user\_id   
  
 self.contacts = self.get\_user\_contacts()   
  
 main\_layout = QVBoxLayout()  
  
 main\_layout = QVBoxLayout()   
  
 self.search\_input = QLineEdit()   
 self.search\_input.textChanged.connect(self.search\_contacts)   
  
 main\_layout.addWidget(self.search\_input)   
  
 self.contacts\_list = QListWidget()  
 self.contacts\_list.itemClicked.connect(self.view\_contact)  
 main\_layout.addWidget(self.contacts\_list)  
  
 add\_button = QPushButton("Добавить контакт")  
 add\_button.clicked.connect(self.add\_contact)  
 main\_layout.addWidget(add\_button)  
  
 central\_widget = QWidget()  
 central\_widget.setLayout(main\_layout)  
 self.setCentralWidget(central\_widget)  
  
 self.contacts = self.get\_user\_contacts()  
 self.update\_contacts\_list()  
  
 def search\_contacts(self):  
 search\_text = self.search\_input.text().strip().lower()

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 7**

self.contacts\_list.clear()

for contact in self.contacts:  
 if search\_text in contact.name.lower():  
 self.contacts\_list.addItem(contact.name)  
  
 def get\_user\_contacts(self):  
 contacts = []  
 try:  
 conn = mysql.connector.connect(  
 host='127.0.0.1',  
 user='root',  
 database='phones'  
 )  
  
 if conn.is\_connected():  
 cursor = conn.cursor()  
 select\_query = "SELECT name, phone\_number, notes FROM contacts WHERE user\_id = %s"  
 cursor.execute(select\_query, (self.user\_id,))  
 user\_contacts = cursor.fetchall()  
  
 for contact in user\_contacts:  
 contacts.append(Contact(contact[0], contact[1], contact[2]))  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as error:  
 print("Ошибка при работе с базой данных:", error)  
  
 return contacts  
  
 def add\_contact(self):  
 dialog = AddEditContactDialog(self, user\_id=self.user\_id)  
 if dialog.exec() == QDialog.DialogCode.Accepted:  
 self.contacts.append(dialog.get\_contact\_info())  
 self.update\_contacts\_list()  
  
 def view\_contact(self, item):  
 index = self.contacts\_list.row(item)  
 contact = self.contacts[index]  
 dialog = ViewContactDialog(self, contact)  
 if dialog.exec() == QDialog.DialogCode.Accepted:  
 self.update\_contacts\_list()

**ПРОДОЛЖЕНИЕ** **ПРИЛОЖЕНИЯ 7**

def update\_contacts\_list(self):  
 self.contacts\_list.clear()  
 self.contacts = self.get\_user\_contacts()  
 for contact in self.contacts:

self.contacts\_list.addItem(contact.name)  
  
 def remove\_contact\_from\_list(self, contact):  
 self.contacts.remove(contact)  
 self.update\_contacts\_list()