Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный автомобильно-дорожный университет (СибАДИ)»

Институт Информационные системы, экономика и управление

Кафедра Информационная безопасность

Специальность Информационная безопасность

Специализация Информационная безопасность

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине: «Технологии и методы программирования»

Тема проекта: Разработка информационной системы «Студенты ВУЗа»

Выполнил:

студент группы БИб-23Э1 Герасименко София Евгеньевна

Курсовой проект сдан на проверку:

«13» мая 2025 г.

Студент 

(подпись студента)

Курсовой проект допущен к защите:

«15» мая 2025 г.

Руководитель проекта

доцент кафедры, к.п.н\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Абрамченко)

Курсовой проект защищен с оценкой: отлично

«27»мая 2025 г.

Руководитель проекта

доцент кафедры, к.п.н\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Абрамченко

Омск 2025

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный автомобильно-дорожный университет (СибАДИ)»

Кафедра «Информационная безопасность»

УТВЕРЖДАЮ

И.О. Зав. кафедрой,

к.т.н., доцент кафедры

\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Толкачева

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ЗАДАНИЕ**

**к курсовому проекту**

студенту группы БИб-23Э1 Герасименко Софии Евгеньевне

1. Тема курсового проекта: Разработка и тестирование автоматизированной системы «Студенты ВУЗа»
2. Исходные данные к курсовому проекту:

* документация по языку программирования Python;
* требования к функционалу приложения:
  1. разработка базы данных с использованием Python и SQL;
  2. реализация графического интерфейса информационной системы на PyQt5;
  3. обеспечить ведение базы данных;
  4. организация различных уровней доступа;
  5. использование авторизации;
  6. модульное тестирование продукта;

1. Содержание пояснительной записки:

* титульный лист;
* задание к курсовому проекту;
* содержание;
* введение;
* обследование предметной области;
* формулирование требований и технического задания;
* проектирование информационной системы с помощью UML-диаграмм;
* реализация проекта информационной системы средствами языка Python и СУБД MySQL;
* обоснование выбора программных средств;
* разработка базы данных «Студенты ВУЗа»;
* описание состава (модулей) и логики работы программы;
* разработка модульных тестов и проведение тестирования приложения;
* заключение;
* список использованных источников;
* приложения;

1. Перечень демонстрационного материала для сопровождения доклада при защите курсового проекта:

* презентация Microsoft PowerPoint.

1. Срок сдачи проекта: « 20» мая 2025 г.

Задание выдано « 1» марта 2025 г.

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Абрамченко Н.В.

Задание к исполнению принял « 3» марта 2025 г.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

[**Введение** 5](#_Toc199032716)

[**1** **ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ ИНФОРМАЦИОННОЙ СИСТЕМЫ «СТУДЕНТЫ ВУЗА».** 6](#_Toc199032717)

[**Обследование предметной области** 6](#_Toc199032718)

[**Техническое задание и требования** 7](#_Toc199032719)

[**Проектирование информационной системы с помощью UML-диаграмм.** 13](#_Toc199032720)

[**2** **РЕАЛИЗАЦИЯ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ СРЕДСТВАМИ ЯЗЫКА PYTHON И СУБД MYSQL** 16](#_Toc199032721)

[**Обоснование выбора программных средств** 16](#_Toc199032722)

[**Разработка базы данных «Студенты вуза»** 17](#_Toc199032723)

[**Описание состава (модулей) и логики работы программы** 19](#_Toc199032724)

[**Разработка модульных тестов и проведение тестирования приложения;** 38](#_Toc199032725)

[**Заключение** 39](#_Toc199032726)

[**Список использованных источников** 40](#_Toc199032727)

[**Приложение А Исходный код приложения** 41](#_Toc199032728)

[**Приложение Б Руководство пользователя** 42](#_Toc199032729)

# **Введение**

Работа с базами данных является если не главным, то ключевым умением множества специалистов информационных технологий. В данной поставленной задаче база данных представляет собой модель списка личных данных всех лиц, обучающихся в некотором заведении, что напрямую перекликается с профилем обучения – информационной безопасностью.

Курсовой проект является совокупностью умений, таких как навык планирования и анализа разработки коммерческого или некоммерческого продукта, работы с данными в формате SQL-таблиц, автоматизации и оптимизации процессов, обеспечения целостности данных, путем внедрения уровней доступа, дополнительно ориентирование в объемном проекте и создание собственного приложения, с последующим тестированием работы – полезные навыки для работника IT-сферы.

**Цель проекта**: поэтапная реализация автоматизированного программного обеспечения работы с базой данных информационной системы «Студенты ВУЗа» посредством языка программирования Python.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. описать план разработки продукта;
2. провести анализ информационной системы и структуры кода;
3. реализовать интерфейс приложения;
4. реализовать функционал в соответствии с требованиями, изложенными в задании;
5. организовать тестирование продукта;

# **1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ ИНФОРМАЦИОННОЙ СИСТЕМЫ «СТУДЕНТЫ ВУЗА».**

В данном разделе будут рассмотрены теоретические аспекты разработки нашей информационной системы, для начала нам нужно определиться с предметной областью, проанализировать ее и описать. Требуется понять, кем она будет использоваться, какой функционал она должна нести и в общем – зачем она нужна.

# **Обследование предметной области**

Выбранная предметная область представляет собой высшее учебное заведение, заведомо заинтересованное в учете обучающихся в данных момент студентов. Потребности кроме учета могут разниться и на деле их огромное множество, далее мы опишем их подробно. Лица, которые пользуются и, соответственно, допущены к пользованию информационной системой, делятся на три категории: преподаватели, администраторы и сторонний персонал.

Администраторы, это те, кто обычно занимается учетом студентов, они следят за тем, чтобы отчисленных убирали из всех списков, а новых студентов добавляли, они также отвечают за корректность данных, указанных студентом. Эти работники особенно нуждаются в функциях редактирования, а, так же как и для остальных групп в легкой навигации. Преподавателям же может потребоваться добавить новоиспеченного студента в систему, если пока этого не случилось, например, после перевода из одного учебного заведения в другое. Да и использование информации для заполнения документов – тоже нужно и важно. Для прочих работников, таких как бухгалтерия, тоже важно использовать информацию об обучающихся.

Создание информационной системы выступает как автоматизация внутренних процессов учебного заведения. Цифровизация добавляет комфорт в работу сотрудников, увеличивает скорость их работы и экономит место, размещать всю информацию на бумаге – дорого и неудобно, еще и не безопасно.

Однако важно отметить, что наша предметная область очень обширна и многослойна, поэтому реализовать досконально точную информационную систему, отвечающую всем потребностям и, важно отметить, реальности – тяжело. Как и говоря о том, что такой цели поставлено не было. Поэтому формирование и обследование предметной области ограничено и упрощено. Для курсового проекта нам нужна лишь модель, пример того, для чего мы создаем информационную систему.

# **Техническое задание и требования**

1. Предметная область: ВУЗ

2. Этап разработки раздела «Общие сведения»:

Полное наименование ИС: Студенты ВУЗа

Предприятие-разработчик системы: Герасименко София Евгеньевна

Предприятие-заказчик системы: ВУЗ N/СИБАДИ

Система создается на основании технического задания (ТЗ). ТЗ на АС является основным документом, определяющим требования и порядок создания автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие. Кроме того, при создании системы используются ГОСТ 34.602-89 “Техническое задание на создание автоматизированной системы”.

Плановый срок начала работ: 3 мая 2025

Плановый срок окончания работ: 27 мая 2025

Автоматизируемая система создается на некоммерческой основе.

Порядок оформления и предъявления заказчику результатов работы по созданию системы определяется после получения начальной версии продукта, в которой должны быть реализованы все основные функции, определенные в ТЗ и утвержденные заказчиком.

3. Этап разработки раздела «Назначение и цели создания системы»:

Вид автоматизируемой деятельности: Ведение учета учащихся

Перечень автоматизируемых процессов: Учет, редактирование, доступ, навигация

Наименование и значение показателей, которые будут достигнуты в результате внедрения БД:

* увеличение скорости работы сотрудников
* сохранение целостности информации
* уменьшение физических носителей информации, конкретно – бумажных

4. Этап разработки раздела «Характеристики объекта автоматизации»

Стандартный региональный ВУЗ, состоящий из N-количества сотрудников и N-количества обучающихся (далее, от 20 человек).

Организационная структура изображена на рисунке 2.

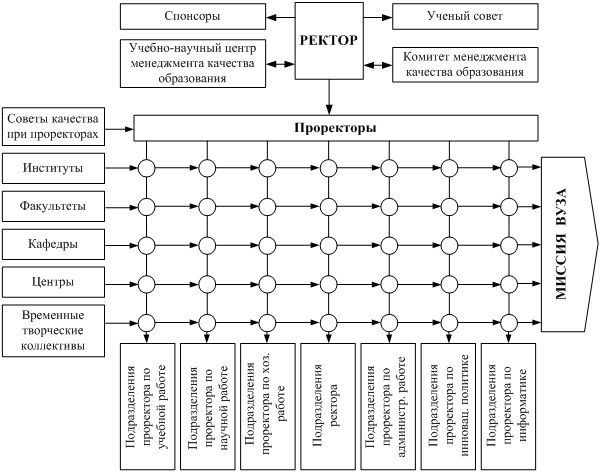


Рисунок 1– Организационная структура ВУЗа

Описание автоматизируемых процессов, информационные потоки автоматизируемых процессов: ведение списка обучающихся, навигация по спискам, редактирование данных, обеспечение уровней доступа или сохранение целостности данных.

5. Этап разработки раздела «Требования к ИС»

Информационная система должна состоять из базы данных на основе языка SQL, с интуитивно-понятным графическим интерфейсом. Должны быть осуществлены следующие требования:

1. Разграничение пользователей по группам/уровням доступа, только администраторы имеют право на удаление студента из базы данных, инструменты навигации доступны всем, элемент добавления студента доступен администраторам и преподавателям;
2. Наличие функций удаления, добавления студента в таблицу графического интерфейса и соответственно в базу данных;
3. Обеспечение навигации по таблице с помощью фильтров и поиска по совпадению;

ИС должна соответствовать требованиям технического задания на ее создание и развитие, а также требованиям нормативно-технических документов, действующих в ведомстве заказчика ИС.

Ввод в действие ИС должен приводить к полезным технико-экономическим, социальным результатам:

1. Повышение комфорта в рабочей среде для сотрудников
2. Экономия средств на физические носители информации

Технические средства ИС должны быть установлены так, чтобы обеспечивались их безопасная эксплуатация и техническое обслуживание. Требования безопасности устанавливаются в инструкциях по эксплуатации технических средств.

Требования к функциям, выполняемым системой:

1. Иметь высокую скорость выполнения
2. Исправно выполнять свою работу

Данная информационная система разрабатывается с расчетом на нескольких пользователей – администраторов, преподавателей и персонала. Таким образом, разрабатываемая система должна обеспечивать решение вышеперечисленных задач.

В готовом виде она должна быть максимально простой и удобной: все операции должны выполняться с помощью элементарных действий пользователя.

Требования к программному обеспечению ИС:

1. Поддержка Python 3.6 и выше
2. Требования к техническому обеспечению АС:

Минимальные требования:

|  |  |
| --- | --- |
| Компонент | Требования |
| Процессор | 1 ГГц (x86/x64) |
| ОЗУ | 512 МБ (1 ГБ для комфортной работы) |
| Место на диске | 50 МБ (для Python и зависимостей) |
| Экран | Разрешение 1024×768 |

Таблица 1 - Минимальные требования

Рекомендуемые характеристики:

|  |  |
| --- | --- |
| Компонент | Рекомендации |
| Процессор | 2+ ядерный, 2 ГГц |
| ОЗУ | 2 ГБ+ |
| ОС | Windows 10+, macOS 10.15+, Linux с современным DE |

Таблица 2- Рекомендуемые характеристики

6. Поддерживаемые операционные системы

Windows (7/10/11, x86/x64)

macOS (10.15 Catalina и новее)

Linux (Ubuntu 20.04+, Fedora 32+, Debian 11+ с графической средой)

7. Этап разработки раздела «Стадии и этапы разработки»

Разработка должна быть проведена в три стадии:

* разработка технического задания;
* рабочее проектирование;
* внедрение.

8. Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1. разработка модели автоматизируемых процессов и функциональной модели ИС;
2. разработки логической и физической моделей данных;
3. разработка программы;
4. разработка программной документации;
5. испытания программы.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах заказчика.

Приемо-сдаточные испытания должны проводиться на объекте заказчика в оговоренные сроки. Приемо-сдаточные испытания программы должны проводиться согласно разработанной исполнителем и согласованной заказчиком программы и методик испытаний. Ход проведения приемо-сдаточных испытаний заказчик и исполнитель документируют в протоколе проведения испытаний. На основании протокола проведения испытаний исполнитель совместно с заказчиком подписывает акт приемки-сдачи программы в эксплуатацию.

# **Проектирование информационной системы с помощью UML-диаграмм.**

Проектирование информационной системы нужно начать с составления сценария работы, а именно, нужно представить, как пользователи будут действовать в рамках системы, какие действия им нужно и придется выполнять. Поскольку у каждой категории есть определенные задачи и значимость, функции, которые они могут выполнять среди базы данных должны и будут ограничены. Систематизировав и обозначив эти зависимости в Use-Case диаграмме, мы можем наблюдать за взаимодействиями пользователей с этой информационной системой (Рис.2).

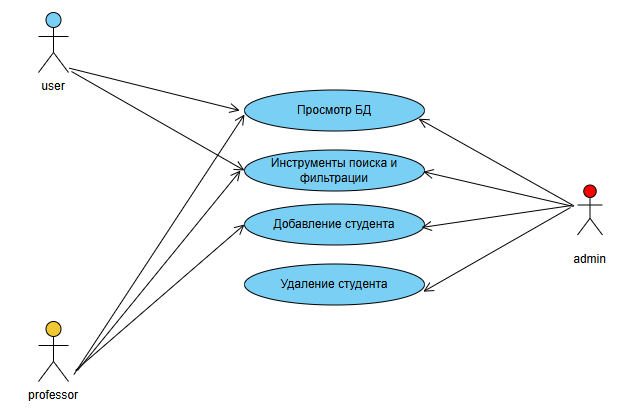


Рисунок 2 - Use-Case диаграмма информационной системы "Студенты ВУЗа"

На диаграмме обозначены «user», «professor» и «admin». Соответственно: пользователь (персонал), преподаватель и администратор.

Когда разработка приблизилась к созданию кода, нужно разобраться, какие взаимосвязи будут между элементами проекта и в какой последовательности будет действовать пользователь уже в отношении программного обеспечения. Для этого созданы UML-диаграммы последовательности и UML-диаграммы классов. В диаграмме последовательности мы проследим действия пользователя от входа в систему, до всех функций, которые можно использовать. С помощью этой диаграммы будет наглядно описаны зависимости между запросами пользователя и последовательностью работы кода. Непосредственно эту схему можно изучить на рисунке 3.

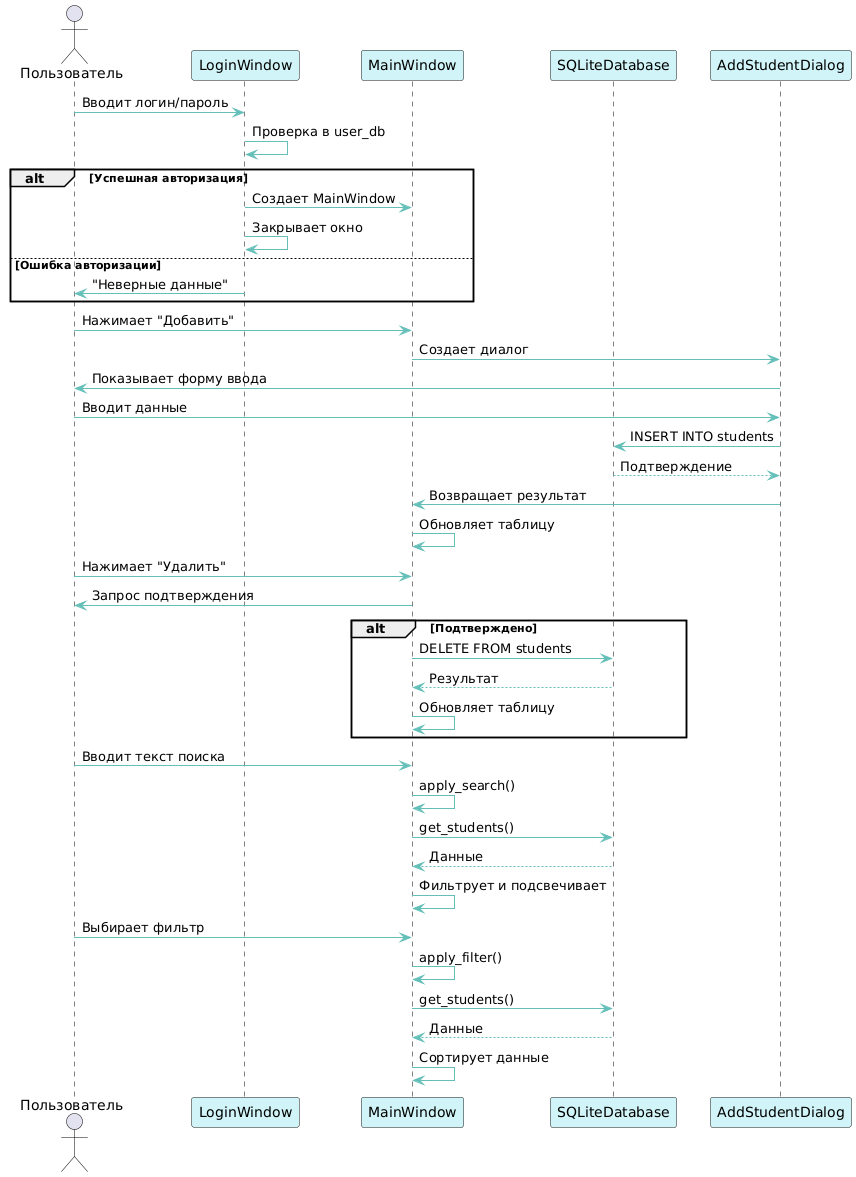


Рисунок 3 – Диаграмма последовательности

UML-диаграммы классов нужны для визуального представления структуры системы в объектно-ориентированном программировании, они помогают понять, из каких классов состоит система, как эти классы связаны между собой (ассоциации, наследование, агрегация и т. д.), а также какие атрибуты и методы есть у каждого класса.

Это упрощает проектирование, анализ и документирование кода, улучшает коммуникацию между разработчиками, позволяет выявить потенциальные проблемы на ранних этапах разработки и обеспечивает четкую архитектуру программы перед началом написания кода..разработки. (см рис. 4)

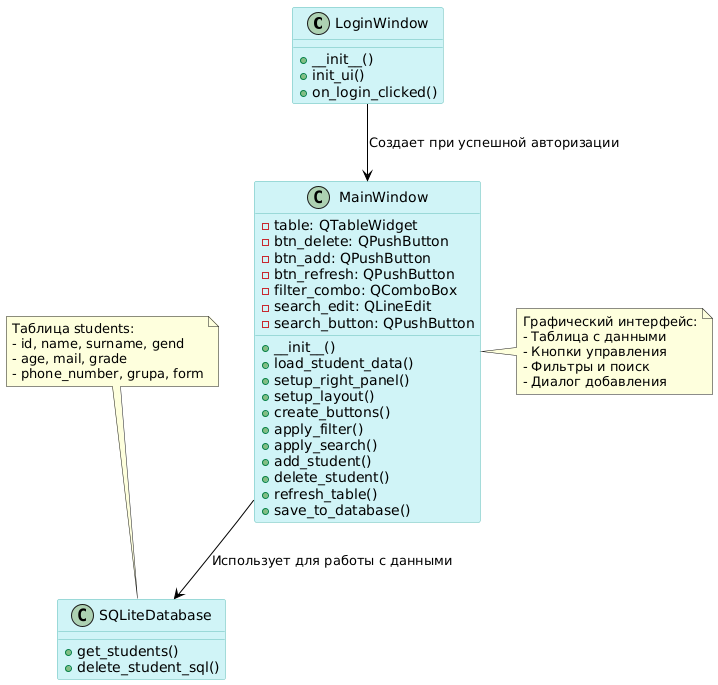


Рисунок 4 - Диаграмма классов

Таким образом, мы спланировали и спроектировали информационную систему, с помощью составления технического задания, анализа предметной области и UML- диаграмм.

# **2 РЕАЛИЗАЦИЯ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ СРЕДСТВАМИ ЯЗЫКА PYTHON И СУБД MYSQL**

Как только мы спланировали и спроектировали нашу информационную систему, пришло время ее создания как полноценного приложения. В данной главе будет описан выбор средств, процесс кодирования и общие принципы работы, также мы проведем тестирование системы.

# **Обоснование выбора программных средств**

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

Мы используем Python, так как, кроме прочих преимуществ, данный язык относится к группе языков, обеспечивающих быстрое освоение и свободу в действиях за счет собственной оптимизации, в сравнении с языком С ++, обучение которого было осуществлено на протяжении более одного учебного года, Python позволяет ускорить процесс разработки, прививает привычку к структурированию кода, параллельно открывая множество возможностей, таких как:

* Работа с xml/html файлами
* Работа с http запросами
* GUI (графический интерфейс)
* Создание веб-сценариев
* Работа с FTP.
* Работа с изображениями, аудио и видео файлами
* Программирование математических и научных вычислений

PyQt5 – объемная библиотека GUI для работ с интерфейсами и, соответственно, для их создания на языке Python, обеспечивающая достаточный объем функций и параметров для комфортного использования.

PyQt5 представляет собой мощный инструмент, позволяющий обширно и модульно произвести создание интерфейса программы, параллельно закрепляя знания объектно-ориентированного программирования, что является полезным дополнением в разработке, актуальным для выполнения курсового проекта.

SQLite3 является встроенной библиотекой языка Python для работы с одноименной реляционной системой управления базами данных. В отличие от клиент-серверных СУБД, SQLite реализует бессерверную архитектуру, что делает её идеальным решением для локального хранения структурированных данных в desktop-приложениях. Дополнительно, оказался наиболее удобным в работе.

# **Разработка базы данных «Студенты вуза»**

В отношении нашей информационной системы стоит начать c формирования источника информации - базы данных, содержащей требуемые категории данных, таких как:

* личные данные (ФИО, пол, возраст)
* контактные данные (электронная почта, номер мобильного телефона)
* специальные данные (курс обучения, название группы, форма обучения)

Реализовать данную базу будем с помощью SQL-таблиц, на основе СУБД – SQlite Studio и библиотеки sqlite3. Создаем файл в проекте с названием «sql.py» и импортируем библиотеку. Кратко изучив синтаксис SQL, сформируем базу данных с названием «students.db», для этого мы должны установить соединение с sqlite3 и записать нашу «шапку» таблицы – виды данных студентов, дополнительно уточняя тип данных и допустимое количество символов записи. Стоит уточнить, что мы прописываем создание пустой таблицы только в том случае, если она еще не существует, в коде – « CREATE TABLE IF NOT EXISTS». Полный код создания пустой таблицы находится на рисунке 5.

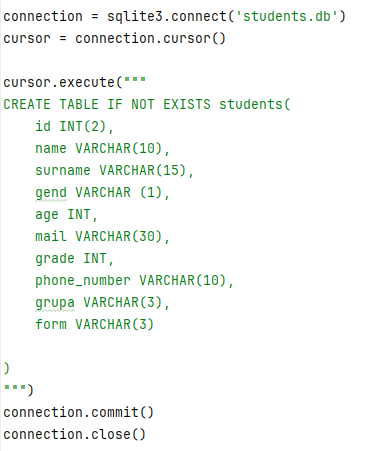


Рисунок 5 – Создание пустой базы данных

Далее, заполняем или дословно вставляем в таблицу записи о 20-ти тестовых студентах конструкцией «INSERT INTO». В коде это выглядит как монолитный поток текста, также стоит отметить, что после каждой манипуляции с базой, мы подтверждаем изменения конструкцией «.commit», отключаемся от базы и подключаемся заново. Такие действия можно охарактеризовать строго предосторожностью. На этом работа с созданием базы данных закончена и по значениям она примет следующий вид (см. рис. 6). Подтверждаем изменения и закрываем соединение.



Рисунок 6 – Заполнение данными

Таким образом, мы осуществили формирование базы данных и готовы приступать к формированию графического интерфейса.

# **Описание состава (модулей) и логики работы программы**

Создание графического интерфейса оказалось самым сложным и объемным процессом, по истечению которого, мы получили следующие результаты (Рис.7-8).

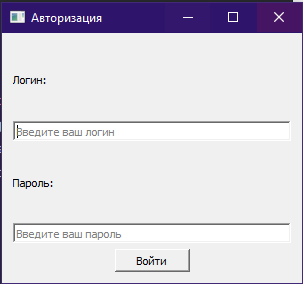


Рисунок 7 – Окно авторизации

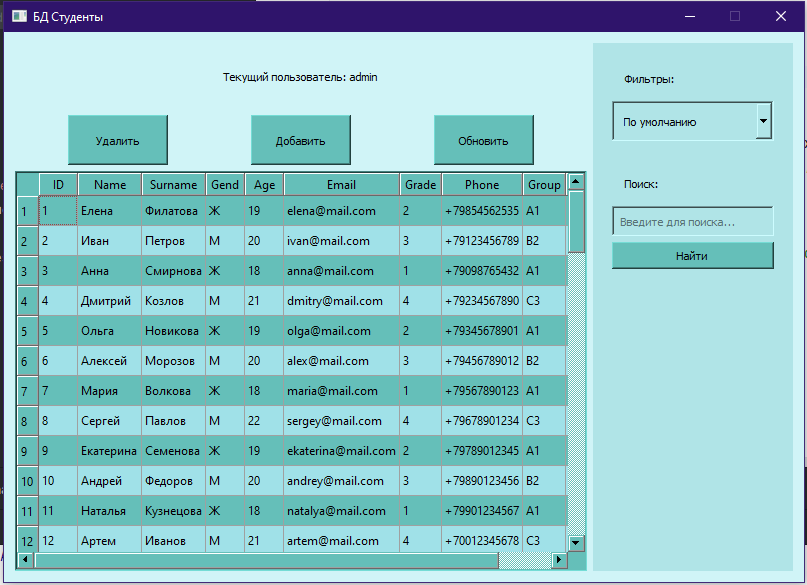


Рисунок 8 – Основное окно приложения

Для осуществления и запуска приложения в файле «sqlGUI.py» потребуются следующие библиотеки и функции:

1. import sys - используется для работы с системными функциями;
2. import re - стандартная библиотека для работы с регулярными выражениями;
3. from PyQt5 import Qt - основной модуль фреймворка pyqt5, содержит базовые классы для создания gui-приложений;
4. from PyQt5.QtCore import Qt as QtCoreQt, QSize  
   from PyQt5.QtGui import QFont, QIcon  
   from PyQt5.QtWidgets import (QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QMessageBox, QHBoxLayout, QFrame, QTableWidget, QTableWidgetItem, QInputDialog, QComboBox)- модули PyQt5 для графических элементов;
5. import sqlite3 – библиотека базы данных;
6. from sql import get\_students, delete\_student\_sql – функции;

**Пояснение кода окна авторизации**

Для создания окна авторизации, как и для последующих окон, мы создаем собственный класс. Мы выберем имя «LoginWindow», типа QWidget – виджет.

class LoginWindow(QWidget):  
 def \_\_init\_\_(self): - опишем метод инициализации класса  
 super().\_\_init\_\_() – позволим вызывать метод из родительских классов в определенном порядке  
 self.setWindowTitle('Авторизация')- укажем имя окна  
 self.setGeometry(200, 200, 300, 250)- установим размер окна  
 self.init\_ui()- метод создания остальных составляющих окна

В методе init\_ui()мы задействуем следующие модули:

1. QLabel – текстовая метка для обозначения окон ввода (см. рис. 9)
2. QLineEdit – окно ввода для логинов и паролей (см. рис. 10)
3. QPushButton – кнопка (см. рис. 11)
4. QVBoxLayout – модуль раскладки виджетов

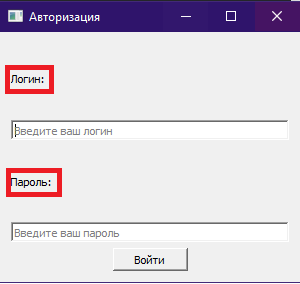


Рисунок 9 – QLabel

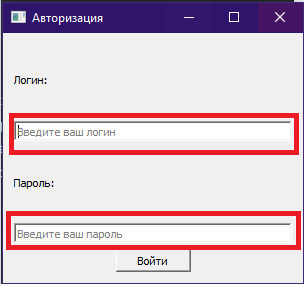


Рисунок 10 – QlineEdit

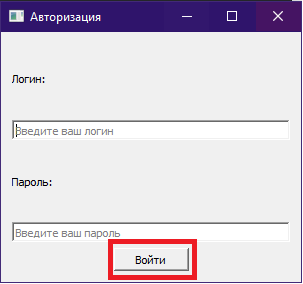


Рисунок 11- QPushButton

Полный код метода init\_ui (Рис.12):



Рисунок 12 - Полный код

Использованные функции:

1. setPlaceholderText – устанавливает текстовую подсказку в окне ввода
2. connect – соединяет интерактивный элемент с функцией
3. addWidget – добавляет виджет в раскладку
4. addStretch – растягивает элемент по раскладке
5. addLayout – добавляет раскладку в раскладку

В методе задействовано две раскладки, горизонтальная QHBoxLayout и вертикальная QVBoxLayout, посредством размещения горизонтальной раскладки в вертикальную мы получаем вертикальное расположение текстовых меток и полей ввода, а внизу одну кнопку посередине. По этому принципу работают все раскладки в коде.

Переходя к основной части – авторизации, мы подключаем функцию on\_login\_clicked к кнопке «Вход», полный код изображен ниже (рис.13).



Рисунок 13- Полный код функции on\_login\_clicked

Поясним принцип работы авторизации:

|  |
| --- |
| def on\_login\_clicked(self):  login = self.edit\_login.text()- собираем информацию из поля ввода для логина;  password = self.edit\_password.text() – для пароля;  auth = False  for user in user\_db:- ищем в массиве с логинами и паролями;  if user[0] == login and user[1] == password: - если логин соответствует правильному паролю ;  auth = True – авторизация истинна;  global current\_user  current\_user = login – нынешний пользователь это логин;  break   if auth:  self.main\_window = MainWindow()- если авторизация истина;  self.main\_window.current\_user\_label.setText(f"Текущий пользователь: {current\_user}") – установка текстовой метки нынешнего пользователя ;  self.main\_window.show()- открытие основного окна;  self.close()- окно авторизации закрывается;   else:  QMessageBox.warning(self, 'Ошибка', 'Неверные данные или доступ отклонен')- обработка неверного ввода; |

Таким образом, мы создали окно авторизации и теперь переходим к основному окну.

**Пояснение кода основного окна**

Аналогично окну авторизации создаем класс и прописываем элементы (рис.14):



Рисунок 14- Основное окно и его элементы

Использованные модули:

1. QLabel – текстовая метка для нынешнего пользователя (см. рис. 15)
2. QPushButton – кнопка (см. рис. 16)
3. QV/HBoxLayout – модули раскладки виджетов
4. QTableWidget- модуль таблицы (см. рис.17)

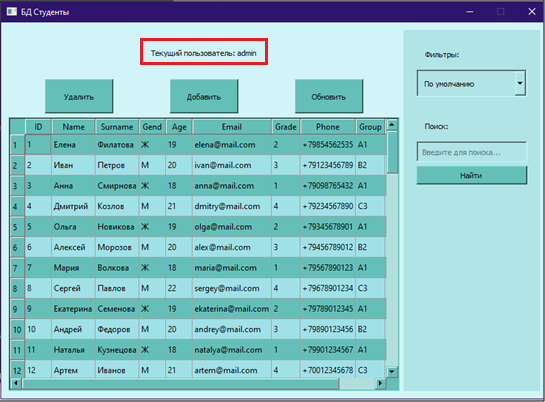


Рисунок 15 – QLabel

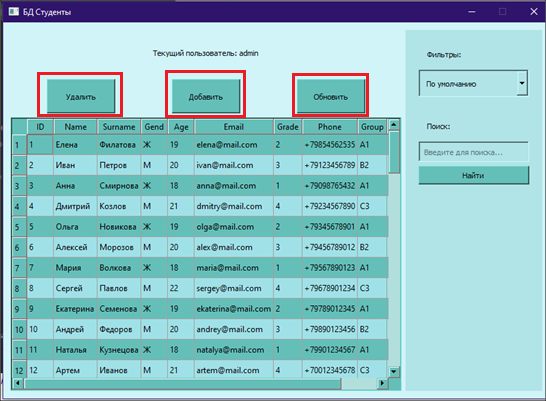


Рисунок 16 - QPushButton

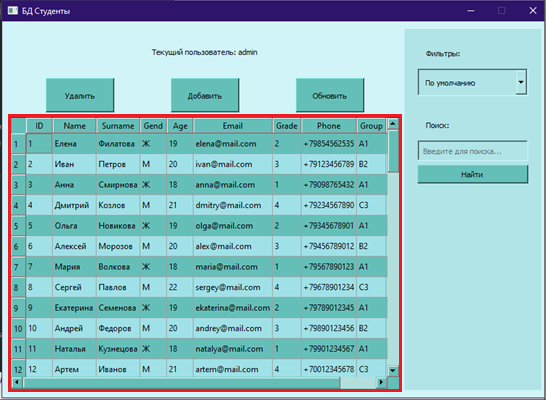


Рисунок 17- QTableWidget

Использованные функции:

1. setFixedSize – установить фиксированный размер;
2. setStyleSheet – установить стиль, синтаксис CSS;
3. setAlignment(QtCoreQt.AlignCenter)- установить расположение по раскладке, конкретно – по центру;
4. setFixedHeight – установить фиксированную высоту;
5. setEditTriggers(QTableWidget.NoEditTriggers) – отключение интерактивности таблицы;
6. load\_student\_data()  
   setup\_right\_panel()  
   create\_buttons()  
   setup\_layout() – функции/методы, далее подробнее;

Метод setup\_layout – основная раскладка окна, код можно наблюдать на рисунке 18.



Рисунок 18 - Метод setup\_layout

Список раскладок:

1. buttons\_layout – включает в себя кнопки;
2. user\_controls – включает одну метку нынешнего пользователя;
3. left\_layout - включает в себя раскладки выше;
4. right\_panel – элемент правой панели (рамка)далее подробнее;
5. main\_layout – включает в себя раскладки выше;

Использованные функции:

1. setSpacing – установка расстояния между элементами в раскладке;
2. setContentsMargins – установка полей раскладки;

Метод load\_student\_data – код нужен для выгрузки базы данных в таблицу, код ниже на рисунке 19.



Рисунок 19 - Метод load\_student\_data

Пояснение работы и функций:

|  |
| --- |
| def load\_student\_data(self):   students\_data = get\_students()- взять информацию студентов из функции get\_students – далее подробнее;  if students\_data and students\_data != "no data": - если данные поступили или нет;  if isinstance(students\_data, str): - если данные относятся к строкам;  try: - то;  students\_data = eval(students\_data)- выполняет строку, переданную ему в качестве аргумента;  except:- иначе;  students\_data = [] – данные пусты  if students\_data and isinstance(students\_data, (list, tuple)): - если данные принадлежат контейнеру лист;  self.table.setRowCount(len(students\_data))- длина строк равноценна длине количества студентов, setRowCount- установить счетчик строк;  self.table.setColumnCount(10)- установка количества колонок таблицы;  headers = ["ID", "Name", "Surname", "Gend", "Age", "Email", "Grade", "Phone", "Group", "Form"] – заголовки колонок;  self.table.setHorizontalHeaderLabels(headers) – уточнение,что заголовки горизонтальны;   for row\_idx, student in enumerate(students\_data): - для строки информации ;  for col\_idx, value in enumerate(student):- для столбца;  item = QTableWidgetItem(str(value))- создание ячейки информацией;  self.table.setItem(row\_idx, col\_idx, item) – наполнение ячейки;  self.table.resizeColumnsToContents()- установить размеры ячеек исходя из размера информации;  self.table.setAlternatingRowColors(True)- разрешить альтернативный цвет для строк;  self.table.setStyleSheet("alternate-background-color: -a0e1e8; background-color: #65bfb9;")- цвета таблицы; |

Подробнее о функции get\_students, находящейся в файле sql.py:

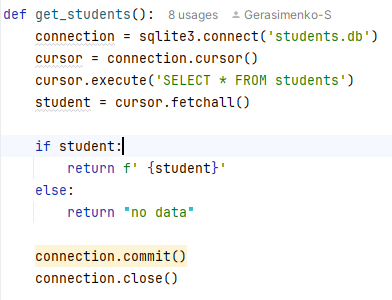
****

Рисунок 20- функция get\_students

|  |
| --- |
| def get\_students():  connection = sqlite3.connect('students.db')- подключение к БД;  cursor = connection.cursor() – использовать переменную как управление;  cursor.execute('SELECT \* FROM students') – выделить все из таблицы;  student = cursor.fetchall()- в буфере вся информация;   if student: - если не пусто;  return f' {student}'- возвращай информацию в переменной;  else:  return "no data"- иначе вывод сообщения;   connection.commit()- подтвердить;  connection.close()- закрыть соединение; |

Таким образом, мы описали принцип появления информации в таблице, перейдем к кнопкам.

Метод create\_buttons подключает кнопкам свои функции, таким образом, на рисунке 20:

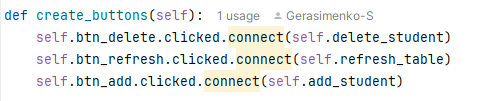


Рисунок 21- Метод create\_buttons

Соответственно опишем каждую функцию кнопок, первой опишем обновление таблиц - функцию refresh\_table.

|  |
| --- |
| def refresh\_table(self):  students\_data = get\_students()  self.table.setRowCount(0)   if students\_data and students\_data != "no data":  if isinstance(students\_data, str):  try:  students\_data = eval(students\_data)  except:  students\_data = []   self.table.setRowCount(len(students\_data))  for row\_idx, student in enumerate(students\_data):  for col\_idx, value in enumerate(student):  item = QTableWidgetItem(str(value))  self.table.setItem(row\_idx, col\_idx, item) |

Эта функция идентична функции load\_student\_data, работает одинаково, за исключением, что не создает таблицу заново, а лишь по базе данных перезаполняет содержимое ячеек.

Следующая функция – delete\_student, пояснение функций библиотек ведем параллельно:

|  |
| --- |
| def delete\_student(self):  selected = self.table.selectedItems()- выбранная ячейка;  if selected: - если ячейка чему-то равна;  row = selected[0].row() – то выбранный ряд - первый;  student\_id = self.table.item(row, 0).text()- ID студента равен номеру ряда ;   reply = QMessageBox.question( - вывести всплывающее окно с выбором ответа;  self, 'Подтверждение',  f"Удалить студента ID {student\_id}?",  QMessageBox.Yes | QMessageBox.No  ) – сообщение предупреждения (рисунок 21,1) , правда ли пользователь хочет удалить студента с этим ID;    Рисунок 21.1 – Обьект типа QMessageBox (всплывающее окно)  if reply == QMessageBox.Yes: - если ответ да;  self.table.removeRow(row)- убрать ряд из таблицы;  delete\_student\_sql(student\_id)- далее подробно;  Из файла «sql.py» мы используем функцию delete\_student\_sql, удаление из базы данных. Опишем ее тоже:  def delete\_student\_sql(student\_id):  connection = sqlite3.connect('students.db')  cursor = connection.cursor()   cursor.execute("DELETE FROM students WHERE id = ?", (student\_id,)) – удаляем по ID;   connection.commit()  return True |

Последняя кнопка – кнопка добавления, работает от двух функций, разберем их ниже:

|  |
| --- |
| def add\_student(self):  try:  dialog = QInputDialog(self)- создаем диалоговое окно с возможностью ввода информации (рис.22)    Рисунок 22 – Диалоговое окна типа QInputDialog  dialog.setWindowTitle("Добавить студента")  dialog.setLabelText(  "Введите данные через запятую:\nИмя,Фамилия,Пол(M/Ж),Возраст,Email,Курс,Телефон,Группа,Форма(Б/К/Ц/ВБ)") – правила ввода;   if dialog.exec\_():- если диалоговое окно продолжает существовать;  input\_data = dialog.textValue()- собираем данные из ввода;  data = [x.strip() for x in input\_data.split(',')] – делим данные по запятой;   if len(data) != 9:- если длина оказывается меньше 9;  raise ValueError("Нужно ввести ровно 9 параметров")- сообщаем об ошибке;    if data[2] not in ['М', 'Ж']:- если пол не соответствует правилам;   raise ValueError("Пол должен быть M или Ж")- сообщаем об этом;   if not data[3].isdigit() or not data[5].isdigit(): - если числовые данными такими не являются;  raise ValueError("Возраст и курс должны быть числами")- сообщаем;   if self.save\_to\_database(data):- сохраняем в базу;  self.refresh\_table()-обновляем таблицу;  QMessageBox.information(self, "Успех", "Студент добавлен")- информируем об успехе;   except ValueError as ve:  QMessageBox.warning(self, "Ошибка ввода", str(ve))- выводят сообщения на экран об ошибке ввода;  except Exception as e:  QMessageBox.critical(self, "Ошибка", f"Ошибка: {str(e)}")- о сторонней ошибке; |

Теперь нужно сохранить нового студента в базу данных, следующим образом:

|  |
| --- |
| def save\_to\_database(self, student\_data):  try:  connection = sqlite3.connect('students.db')  cursor = connection.cursor()   cursor.execute("""  INSERT INTO students (name, surname, gend, age, mail, grade, phone\_number, grupa, form)  VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)  """, student\_data)-вставляем студента в базу;   connection.commit()  connection.close()  return True  except Exception as e:  QMessageBox.critical(self, "Ошибка БД", f"Ошибка при сохранении: {str(e)}")-обработка ошибок;  return False |

Таким образом мы описали принцип работы всей основной части интерфейса, кроме определения уровней доступа по нынешнему пользователю, работает это так:

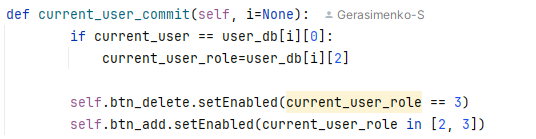


Рисунок 23 - Функция определения уровня доступа нынешнего пользователя

Смысл работы функции заключается в поиске уровня доступа из массива student\_db по отношению к действующему пользователю, ранее в коде массив уже появлялся и содержит в себе следующие записи - user\_db = [["admin", "admin", 3], ["prof", "123", 2], ["void", "100", 1]]. Соответственно – логин, пароль, уровень доступа.

Сначала мы по условию того, что логин нынешнего пользователя равен первой ячейке массива вне зависимости от строки, назначаем новой переменной current\_user\_role – уровень доступа – последнюю ячейку строки.

Далее при помощи функции – setEnabled, назначаем условие, при котором кнопки будут доступны. Для 3-го и 2-го доступны кнопки добавления, но удалить доступно только 3-му уровню. На этой функции описание основной части интерфейса закончено, переходим к правой панели.

Правая панель интерфейса, включает в себя использование фильтрации и возможности поиска по совпадению (см.рис.24).

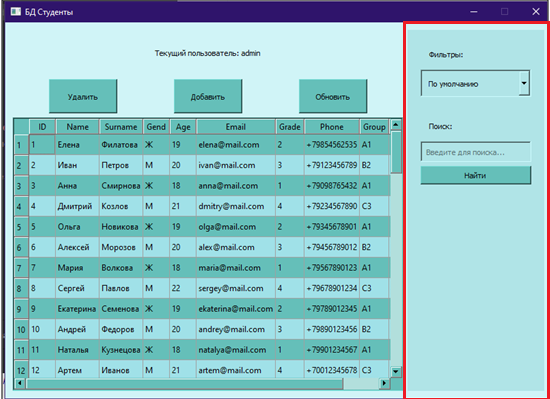


Рисунок 24 – Правая панель интерфейса

В этой части интерфейса использованы ранее нам неизвестные модули – Qframe и QComboBox.

QFrame – представляет собой рамку или заполненный цветом прямоугольник определенного размера. QComboBox- интерактивный элемент, с шторкой – выбором вариантов работы элемента (рисунки 25 и 26).

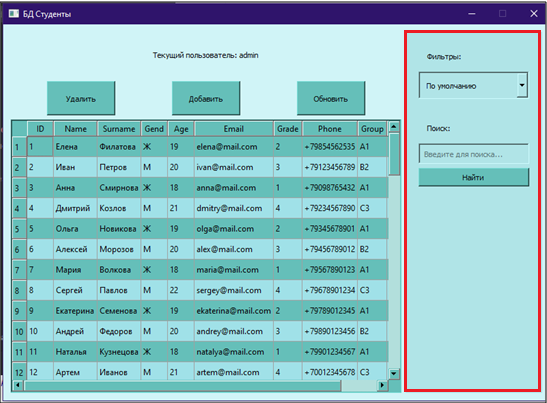


Рисунок 25 - Элемент QFrame

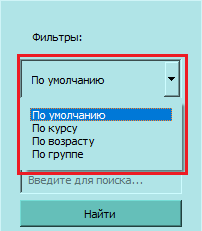


Рисунок 26 - Элемент QComboBox

Данные части интерфейса описываются в методе setup\_right\_panel.

Также в этом методе описывается раскладка правой панели, все текстовые метки и кнопка «Найти», отвечающая за начало поиска по соответствию, разберем этот метод последовательно.

Инициализируем рамку панели, устанавливаем размер и выбираем цвет, указываем расстояние между элементами. Фрагмент кода находится на рисунке 27.

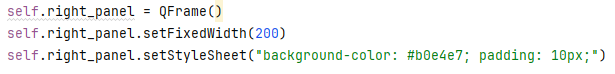


Рисунок 27 – Отрывок кода рамки

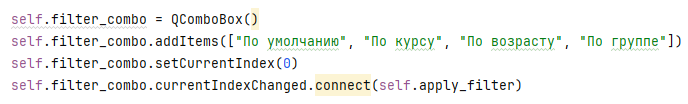
Теперь опишем наш ComboBox, укажем его вариации в шторке и начальную вариацию с индексом 0, дополнительно привяжем метод apply\_filter, далее разберем подробнее, часть кода наблюдаем на рисунке 28. 

Рисунок 28 – Отрывок кода ComboBox

На рисунках 29-3 указаны части кода отвечающие за элементы поиска по таблице, метод execute\_search и apply\_search подробно разберем позже.

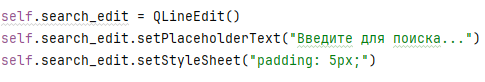


Рисунок 29 – Фрагмент кода ввода текста для поиска

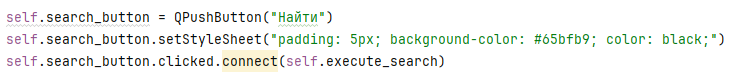


Рисунок 30- Фрагмент кода кнопки активирующей поиск

Расположение виджетов по правой панели выглядит следующим образом – все элементы располагаются вертикально, код на рисунке 31

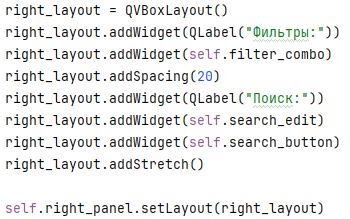


Рисунок 31 – Раскладка правой панели – фрагмент кода

Приступим к принципу работы функций фильтрации и поиска по соответствию. Фильтрация работает от метода apply\_filter, ниже поясним принцип его работы.

|  |
| --- |
| def apply\_filter(self, index):  students\_data = get\_students()- собираем данные студентов из БД;  if isinstance(students\_data, str):- если данные относятся к строкам  try:  students\_data = eval(students\_data)- возвращаем выполненную строку;  except:  students\_data = [] – иначе оставляем информацию пустой;   if index == 0:- если фильтруем по умолчанию;  sorted\_data = sorted(students\_data, key=lambda x: x[0])- выполняем встроенную сортировку по первому столбцу - ID  elif index == 1:- если фильтруем по курсу;  sorted\_data = sorted(students\_data, key=lambda x: x[6]) ])- выполняем встроенную сортировку по шестому столбцу;  elif index == 2: - если фильтруем по возрасту;  sorted\_data = sorted(students\_data, key=lambda x: x[4])- выполняем встроенную сортировку по 4 столбцу;   elif index == 3: - если фильтруем по группе;  sorted\_data = sorted(students\_data, key=lambda x: x[8])- выполняем встроенную сортировку по восьмому столбцу;   self.table.setRowCount(0) – устанавливаем счетчик строк на 0;  self.table.setRowCount(len(sorted\_data))- меняем счетчик на количество найденных строк;   for row\_idx, student in enumerate(sorted\_data):  for col\_idx, value in enumerate(student):  item = QTableWidgetItem(str(value))  self.table.setItem(row\_idx, col\_idx, item)- заполняем таблицу; |

Метод отвечающий за поиск называется – apply\_search, активируется нажатием кнопки черед метод execute\_search. Для начала рассмотрим метод apply\_search ниже:

|  |
| --- |
| def apply\_search(self, text):  try:  - приводим текст поиска к нижнему регистру и удаляем пробелы по краям  search\_text = text.lower().strip()  - если строка поиска пустая - обновляем таблицу (показываем все данные)  if not search\_text:  self.refresh\_table()  return  - получаем данные о студентах из базы данных  students\_data = get\_students()  -проверяем, есть ли данные для поиска  if not students\_data or students\_data == "no data":  QMessageBox.information(self, "Поиск", "Нет данных для поиска")  return  -если данные пришли в виде строки, преобразуем их в список  if isinstance(students\_data, str):  students\_data = eval(students\_data)  - создаем список для найденных совпадений  matched\_rows = []    - перебираем всех студентов  for student in students\_data:  -проверяем, содержится ли поисковый запрос в любом из полей студента  if any(search\_text in str(field).lower() for field in student):  matched\_rows.append(student) - добавляем студента в результаты  - очищаем таблицу перед выводом результатов  self.table.setRowCount(0)  - если найдены совпадения  if matched\_rows:  - устанавливаем количество строк в таблице по количеству найденных студентов  self.table.setRowCount(len(matched\_rows))    - заполняем таблицу данными  for row\_idx, student in enumerate(matched\_rows):  for col\_idx, value in enumerate(student):  -создаем ячейку таблицы  item = QTableWidgetItem(str(value))    - если поисковый запрос найден в этом поле - подсвечиваем желтым  if search\_text in str(value).lower():  item.setBackground(QtCore.Qt.yellow)    - добавляем ячейку в таблицу  self.table.setItem(row\_idx, col\_idx, item)  else:  - если совпадений нет - показываем сообщение  QMessageBox.information(self, "Поиск", "Совпадений не найдено")  - обработка возможных ошибок  except Exception as e:  QMessageBox.critical(self, "Ошибка поиска", f"Ошибка при поиске: {str(e)}") |

Теперь привяжем метод так, чтобы кнопка работала через execute\_search (см. рис. 32):

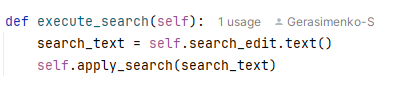


Рисунок 32 – Метод execute\_search

Таким образом, мы описали логику работы программы и готовы к приступать к тестированию продукта

# **Разработка модульных тестов и проведение тестирования приложения**

Данный фрагмент кода представляет собой набор автоматизированных тестов, разработанных для проверки функциональности приложения управления студентами, созданного с использованием Python и PyQt5. Тесты охватывают ключевые аспекты работы системы: авторизацию пользователей, работу с базой данных и обработку ввода. Все тесты используют pytest в сочетании с инструментами mock и monkeypatch для изоляции тестируемых компонентов. Для корректной работы тестов требуется использовать следующие библиотеки и зависимости (см. рис 33):

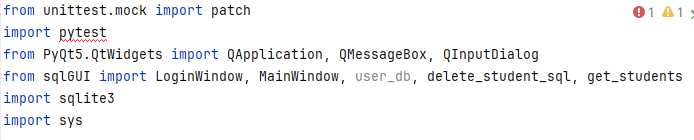


Рисунок 33 – Библиотеки для работы с тестами

Фикстура sqlapp обеспечивает правильную инициализацию QApplication, необходимого для работы Qt-компонентов:

|  |
| --- |
| - фикстура для создания qapplication (необходим для работы qt) @pytest.fixture(scope="session")  def sqlapp(): app = QApplication(sys.argv) - создаем qt-приложение  yield app - возвращаем приложение для тестов  app.quit() - завершаем приложение после всех тестов |

Тестирование авторизации:

В первых двух тестах проверяется система входа в приложение. Тест test\_login\_right проверяет успешную авторизацию при вводе правильных учетных данных (логин и пароль "admin"), ожидая возврата значения True. Тест test\_login\_wrong имитирует попытку входа с неверным паролем и проверяет, что система корректно отклоняет такую попытку, возвращая None. Эти тесты гарантируют, что система безопасности приложения работает должным образом. Отрывок кода и пояснение ниже:

|  |
| --- |
| def test\_login\_right(sqlapp):  login\_window = LoginWindow() - создаем окно входа  login\_window.edit\_login.setText("admin") - вводим логин  login\_window.edit\_password.setText("admin") - вводим пароль  assert login\_window.on\_login\_clicked() == True - проверяем успешный вход  - Тест неправильного входа (неверный пароль)  def test\_login\_wrong(sqlapp):  login\_window = LoginWindow()  login\_window.edit\_login.setText("admin")  login\_window.edit\_password.setText("wrong\_pass") - неверный пароль  assert login\_window.on\_login\_clicked() is None - проверяем, что вход не удался |

Тестирование интерфейса:

Тест test\_window\_open проверяет корректность перехода между окнами приложения. С помощью monkeypatch подменяется метод show() главного окна, что позволяет проверить факт его создания после успешной авторизации без реального отображения интерфейса. Этот подход демонстрирует принцип модульного тестирования, где проверяется изолированная функциональность. Отрывок кода и пояснение ниже:

|  |
| --- |
| def test\_window\_open(sqlapp, monkeypatch):  login\_window = LoginWindow()  login\_window.edit\_login.setText("admin")  login\_window.edit\_password.setText("admin")  mock\_mainwindow = None - переменная для хранения mock-окна  - функция-заглушка для метода show()  def mock\_show(self):  nonlocal mock\_mainwindow  mock\_mainwindow = self - сохраняем созданное окно  - подменяем метод show() на нашу заглушку  monkeypatch.setattr(MainWindow, 'show', mock\_show)  login\_window.on\_login\_clicked() - вызываем вход  assert isinstance(mock\_mainwindow, MainWindow) - проверяем, что окно создано |

Тестирование работы с базой данных:

Тест test\_delete\_student\_from\_db проверяет корректность удаления записей из базы данных. Он создает тестовую запись студента, проверяет ее наличие в БД, затем удаляет и подтверждает успешность этой операции. Этот тест важен для гарантии целостности данных в системе. Отрывок кода и пояснение ниже:

|  |
| --- |
| def test\_delete\_student\_from\_db():  - подключаемся к бд и добавляем тестового студента  connection = sqlite3.connect('students.db')  cursor = connection.cursor()  cursor.execute(  "INSERT INTO students (id, name, surname, gend, age, mail, grade, phone\_number)"  " VALUES (99, 'Test', 'Student', 'М', 20, 'test@mail.com', 1, '1234567890')")  connection.commit()  connection.close()  students\_before = get\_students() - получаем список студентов до удаления  assert "'test'" in students\_before - проверяем, что студент есть в бд  delete\_student\_sql(99) - удаляем студента  students\_after = get\_students() - получаем список после удаления  assert "'Test'" not in students\_after - проверяем, что студента больше нет |

Тестирование валидации ввода:

Последний тест test\_add\_student\_invalid\_input проверяет обработку некорректных данных при добавлении нового студента. С помощью mock-объектов имитируется ввод неверных данных (неправильный формат пола и возраста), и проверяется, что система выдает соответствующие предупреждения. Это обеспечивает надежную защиту от некорректного ввода.

Такой подход к тестированию позволяет гарантировать стабильность работы приложения при внесении изменений и соответствует современным стандартам разработки программного обеспечения. Отрывок кода и пояснение ниже:

|  |
| --- |
| def test\_add\_student\_invalid\_input():  app = MainWindow() - создаем главное окно  - подменяем диалоговые окна:  - exec\_() возвращает true (пользователь нажал ok)  - textvalue() возвращает неверные данные  - -отслеживаем вызов qmessagebox.warning()  with patch.object(QInputDialog, 'exec\_', return\_value=True), \  patch.object(QInputDialog, 'textValue',  return\_value="Иван,Иванов,X,двадцать,ivan@mail.com,2,+79123456789,A101,Б"), \  patch.object(QMessageBox, 'warning') as mock\_warning:  app.add\_student() - пытаемся добавить студента  - проверяем, что было показано предупреждение  mock\_warning.assert\_called\_once()  - и что текст предупреждения содержит информацию об ошибке  assert "Пол должен быть M или Ж" in mock\_warning.call\_args[0][2] or \  "Возраст и курс должны быть числами" in mock\_warning.call\_args[0][2] |

Таким образом, мы провели модульные тесты проекта, результат которых стал следующим (рис 34):

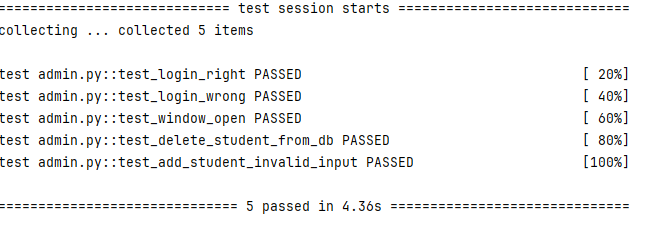


Рисунок 34 - Результаты тестирования

Все тесты вышли положительными, а значит проект готов к эксплуатации.

# **Заключение**

В ходе выполнения курсовой работы была разработана автоматизированная система «Студенты ВУЗа» с использованием языка программирования Python, фреймворка PyQt для создания пользовательского интерфейса, а также СУБД MySQL (или SQLite) для хранения и управления данными. На этапе проектирования применялись UML-диаграммы, что позволило наглядно представить структуру системы, взаимосвязи между классами и логику работы приложения.

В первой главе проведено обследование предметной области, сформулированы требования к системе и составлено техническое задание. Использование UML-диаграмм (таких как диаграммы классов, прецедентов и последовательности) помогло четко спроектировать архитектуру приложения до начала непосредственной разработки.

Во второй главе обоснован выбор программных средств, разработана база данных, описаны модули системы и их взаимодействие. Особое внимание уделено модульному тестированию, что позволило проверить корректность работы каждого компонента и выявить возможные ошибки на ранних этапах.

В результате создано работоспособное приложение, отвечающее поставленным требованиям. Разработка системы подтвердила важность этапов проектирования и тестирования, так как они значительно сокращают время на исправление ошибок и улучшают качество конечного продукта.

# **Список использованных источников**

1. Лучшие IDE и редакторы кода для Python [Электронный ресурс] // Tproger. - URL: <https://tproger.ru/translations/python-ide/> (дата обращения: 12.03.2025).
2. Основы PyCharm [Электронный ресурс] // Habr. - URL: <https://habr.com/ru/articles/720480/> (дата обращения: 15.04.2025).
3. PlantText - онлайн-редактор UML-диаграмм [Электронный ресурс]. - URL: <https://www.planttext.com/> (дата обращения: 20.04.2025).
4. Официальная документация Qt for Python (PyQt5/PySide6) [Электронный ресурс]. - URL: <https://doc.qt.io/qtforpython-5/index.html> (дата обращения: 25.04.2025).
5. SkillFactory. Глоссарий Python [Электронный ресурс]. - URL: <https://blog.skillfactory.ru/glossary/python/> (дата обращения: 01.05.2025).
6. Visual Paradigm - бесплатный инструмент для создания Use Case-диаграмм [Электронный ресурс]. - URL: <https://online.visual-paradigm.com/ru/diagrams/solutions/free-use-case-diagram-tool/> (дата обращения: 05.05.2025).
7. W3Schools. Python String lower() Method [Электронный ресурс]. - URL: <https://www.w3schools.com/python/ref_string_lower.asp> (дата обращения: 10.05.2025).

# **Приложение А Исходный код приложения**

«sqlGUI.py»

import sys  
import re  
from PyQt5 import Qt  
from PyQt5.QtCore import Qt as QtCoreQt, QSize  
from PyQt5.QtGui import QFont, QIcon  
from PyQt5.QtWidgets import (QApplication, QWidget, QLabel, QLineEdit, QPushButton,  
 QVBoxLayout, QMessageBox, QHBoxLayout, QFrame, QTableWidget,  
 QTableWidgetItem, QInputDialog, QComboBox)  
  
import sqlite3  
from sql import get\_students, delete\_student\_sql  
user\_db = [["admin", "admin", 3], ["prof", "123", 2], ["void", "100", 1]]  
  
  
  
class MainWindow(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.btn\_delete = QPushButton("Удалить")  
 self.btn\_add = QPushButton("Добавить")  
 self.btn\_refresh = QPushButton("Обновить")  
 self.setWindowTitle('БД Студенты')  
 self.setGeometry(600, 450, 800, 600)  
 self.setFixedSize(800, 550)  
 self.setStyleSheet("background-color: #d0f4f7;")  
  
 self.current\_user\_label = QLabel()  
 self.current\_user\_label.setAlignment(QtCoreQt.AlignCenter)  
  
 self.table = QTableWidget(self)  
 self.table.setFixedHeight(400)  
 self.table.setEditTriggers(QTableWidget.NoEditTriggers)  
 self.load\_student\_data()  
  
 self.setup\_right\_panel()  
  
 self.create\_buttons()  
 self.setup\_layout()  
  
 def load\_student\_data(self):  
 students\_data = get\_students()  
 if students\_data and students\_data != "no data":  
 if isinstance(students\_data, str):  
 try:  
 students\_data = eval(students\_data)  
 except:  
 students\_data = []  
 if students\_data and isinstance(students\_data, (list, tuple)):  
 self.table.setRowCount(len(students\_data))  
 self.table.setColumnCount(10)  
 headers = ["ID", "Name", "Surname", "Gend", "Age", "Email", "Grade", "Phone", "Group", "Form"]  
 self.table.setHorizontalHeaderLabels(headers)  
  
 for row\_idx, student in enumerate(students\_data):  
 for col\_idx, value in enumerate(student):  
 item = QTableWidgetItem(str(value))  
 self.table.setItem(row\_idx, col\_idx, item)  
 self.table.resizeColumnsToContents()  
 self.table.setAlternatingRowColors(True)  
 self.table.setStyleSheet("alternate-background-color: #a0e1e8; background-color: #65bfb9;")  
  
 def setup\_right\_panel(self):  
 self.right\_panel = QFrame()  
 self.right\_panel.setFixedWidth(200)  
 self.right\_panel.setStyleSheet("background-color: #b0e4e7; padding: 10px;")  
  
 self.filter\_combo = QComboBox()  
 self.filter\_combo.addItems(["По умолчанию", "По курсу", "По возрасту", "По группе"])  
 self.filter\_combo.setCurrentIndex(0)  
 self.filter\_combo.currentIndexChanged.connect(self.apply\_filter)  
  
 self.search\_edit = QLineEdit()  
 self.search\_edit.setPlaceholderText("Введите для поиска...")  
 self.search\_edit.setStyleSheet("padding: 5px;")  
  
 self.search\_button = QPushButton("Найти")  
 self.search\_button.setStyleSheet("padding: 5px; background-color: #65bfb9; color: black;")  
 self.search\_button.clicked.connect(self.execute\_search)  
  
 *#self.search\_edit.returnPressed.connect(self.execute\_search)* right\_layout = QVBoxLayout()  
 right\_layout.addWidget(QLabel("Фильтры:"))  
 right\_layout.addWidget(self.filter\_combo)  
 right\_layout.addSpacing(20)  
 right\_layout.addWidget(QLabel("Поиск:"))  
 right\_layout.addWidget(self.search\_edit)  
 right\_layout.addWidget(self.search\_button)  
 right\_layout.addStretch()  
  
 self.right\_panel.setLayout(right\_layout)  
 def setup\_layout(self):  
 main\_layout = QHBoxLayout(self)  
  
 left\_layout = QVBoxLayout()  
 user\_controls = QVBoxLayout()  
 user\_controls.addWidget(self.current\_user\_label)  
 buttons\_layout = QHBoxLayout()  
 buttons\_layout.addWidget(self.btn\_delete)  
 buttons\_layout.addWidget(self.btn\_add)  
 buttons\_layout.addWidget(self.btn\_refresh)  
 buttons\_layout.setSpacing(30)  
 buttons\_layout.setContentsMargins(0, 0, 0, 0)  
  
 self.btn\_delete.setFixedSize(100,50)  
 self.btn\_add.setFixedSize(100,50)  
 self.btn\_refresh.setFixedSize(100,50)  
 self.btn\_delete.setStyleSheet("background-color: #65bfb9; color: black;")  
 self.btn\_add.setStyleSheet("background-color: #65bfb9; color: black;")  
 self.btn\_refresh.setStyleSheet("background-color: #65bfb9; color: black;")  
 user\_controls.addLayout(buttons\_layout)  
 left\_layout.addLayout(user\_controls)  
 left\_layout.addWidget(self.table)  
  
 main\_layout.addLayout(left\_layout)  
 main\_layout.addWidget(self.right\_panel)  
  
 def create\_buttons(self):  
 self.btn\_delete.clicked.connect(self.delete\_student)  
 self.btn\_refresh.clicked.connect(self.refresh\_table)  
 self.btn\_add.clicked.connect(self.add\_student)  
  
 def current\_user\_commit(self, i=None):  
 if current\_user == user\_db[i][0]:  
 current\_user\_role=user\_db[i][2]  
  
 self.btn\_delete.setEnabled(current\_user\_role == 3)  
 self.btn\_add.setEnabled(current\_user\_role in [2, 3])  
  
 def apply\_filter(self, index):  
 students\_data = get\_students()  
 if isinstance(students\_data, str):  
 try:  
 students\_data = eval(students\_data)  
 except:  
 students\_data = []  
  
 if index == 0:  
 sorted\_data = sorted(students\_data, key=lambda x: x[0])  
 elif index == 1:  
 sorted\_data = sorted(students\_data, key=lambda x: x[6])  
 elif index == 2:  
 sorted\_data = sorted(students\_data, key=lambda x: x[4])  
 elif index == 3:  
 sorted\_data = sorted(students\_data, key=lambda x: x[8])  
  
  
 self.table.setRowCount(0)  
 self.table.setRowCount(len(sorted\_data))  
  
 for row\_idx, student in enumerate(sorted\_data):  
 for col\_idx, value in enumerate(student):  
 item = QTableWidgetItem(str(value))  
 self.table.setItem(row\_idx, col\_idx, item)  
  
  
  
 def execute\_search(self):  
 search\_text = self.search\_edit.text()  
 self.apply\_search(search\_text)  
  
 def apply\_search(self, text):  
  
 try:  
 search\_text = text.lower().strip()  
  
 if not search\_text:  
 self.refresh\_table()  
 return  
  
 students\_data = get\_students()  
  
 if not students\_data or students\_data == "no data":  
 QMessageBox.information(self, "Поиск", "Нет данных для поиска")  
 return  
  
 if isinstance(students\_data, str):  
 students\_data = eval(students\_data)  
  
 matched\_rows = []  
 for student in students\_data:  
 if any(search\_text in str(field).lower() for field in student):  
 matched\_rows.append(student)  
  
 self.table.setRowCount(0)  
  
 if matched\_rows:  
 self.table.setRowCount(len(matched\_rows))  
 for row\_idx, student in enumerate(matched\_rows):  
 for col\_idx, value in enumerate(student):  
 item = QTableWidgetItem(str(value))  
 if search\_text in str(value).lower():  
 item.setBackground(QtCoreQt.yellow)  
 self.table.setItem(row\_idx, col\_idx, item)  
 else:  
 QMessageBox.information(self, "Поиск", "Совпадений не найдено")  
  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка поиска", f"Ошибка при поиске: {str(e)}")  
  
 def delete\_student(self):  
 selected = self.table.selectedItems()  
 if selected:  
 row = selected[0].row()  
 student\_id = self.table.item(row, 0).text()  
  
 reply = QMessageBox.question(  
 self, 'Подтверждение',  
 f"Удалить студента ID {student\_id}?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
  
 if reply == QMessageBox.Yes:  
 self.table.removeRow(row)  
 delete\_student\_sql(student\_id)  
  
 def add\_student(self):  
 try:  
 dialog = QInputDialog(self)  
 dialog.setWindowTitle("Добавить студента")  
 dialog.setLabelText(  
 "Введите данные через запятую:\nИмя,Фамилия,Пол(M/Ж),Возраст,Email,Курс,Телефон,Группа,Форма(Б/К/Ц/ВБ)")  
  
 if dialog.exec\_():  
 input\_data = dialog.textValue()  
 data = [x.strip() for x in input\_data.split(',')]  
  
 if len(data) != 9:  
 raise ValueError("Нужно ввести ровно 9 параметров")  
  
  
 if data[2] not in ['М', 'Ж']:  
 raise ValueError("Пол должен быть M или Ж")  
  
 if not data[3].isdigit() or not data[5].isdigit():  
 raise ValueError("Возраст и курс должны быть числами")  
  
 if self.save\_to\_database(data):  
 self.refresh\_table()  
 QMessageBox.information(self, "Успех", "Студент добавлен")  
  
 except ValueError as ve:  
 QMessageBox.warning(self, "Ошибка ввода", str(ve))  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Ошибка: {str(e)}")  
 def save\_to\_database(self, student\_data):  
 try:  
 connection = sqlite3.connect('students.db')  
 cursor = connection.cursor()  
  
 *# SQL-запрос для вставки данных* cursor.execute("""  
 INSERT INTO students (name, surname, gend, age, mail, grade, phone\_number, grupa, form)  
 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)  
 """, student\_data)  
  
 connection.commit()  
 connection.close()  
 return True  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка БД", f"Ошибка при сохранении: {str(e)}")  
 return False  
 def refresh\_table(self):  
 students\_data = get\_students()  
 self.table.setRowCount(0)  
  
 if students\_data and students\_data != "no data":  
 if isinstance(students\_data, str):  
 try:  
 students\_data = eval(students\_data)  
 except:  
 students\_data = []  
  
 self.table.setRowCount(len(students\_data))  
 for row\_idx, student in enumerate(students\_data):  
 for col\_idx, value in enumerate(student):  
 item = QTableWidgetItem(str(value))  
 self.table.setItem(row\_idx, col\_idx, item)  
  
  
class LoginWindow(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle('Авторизация')  
 self.setGeometry(200, 200, 300, 250)  
 self.init\_ui()  
  
 def init\_ui(self):  
 self.label\_login = QLabel('Логин:')  
 self.edit\_login = QLineEdit()  
 self.edit\_login.setPlaceholderText('Введите ваш логин')  
  
 self.label\_password = QLabel('Пароль:')  
 self.edit\_password = QLineEdit()  
 self.edit\_password.setPlaceholderText('Введите ваш пароль')  
  
 self.button\_login = QPushButton('Войти')  
 self.button\_login.clicked.connect(self.on\_login\_clicked)  
  
 layout = QVBoxLayout()  
 layout.addWidget(self.label\_login)  
 layout.addWidget(self.edit\_login)  
 layout.addWidget(self.label\_password)  
 layout.addWidget(self.edit\_password)  
  
 h\_layout = QHBoxLayout()  
 h\_layout.addStretch()  
 h\_layout.addWidget(self.button\_login)  
 h\_layout.addStretch()  
  
 layout.addLayout(h\_layout)  
 self.setLayout(layout)  
  
  
 def on\_login\_clicked(self):  
 login = self.edit\_login.text()  
 password = self.edit\_password.text()  
 auth = False  
 for user in user\_db:  
 if user[0] == login and user[1] == password:  
 auth = True  
 global current\_user  
 current\_user = login  
 break  
  
 if auth:  
 self.main\_window = MainWindow()  
 *# Устанавливаем текст лейбла* self.main\_window.current\_user\_label.setText(f"Текущий пользователь: {current\_user}")  
 self.main\_window.show()  
 self.close()  
 return True  
 else:  
 QMessageBox.warning(self, 'Ошибка', 'Неверные данные или доступ отклонен')  
 return None  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = LoginWindow()  
 window.show()  
 sys.exit(app.exec\_())

# **Приложение Б Руководство пользователя**

«sql.py»

import sqlite3  
  
connection = sqlite3.connect('students.db')  
cursor = connection.cursor()  
  
cursor.execute("""  
CREATE TABLE IF NOT EXISTS students(  
 id INT(2),  
 name VARCHAR(10),  
 surname VARCHAR(15),  
 gend VARCHAR (1),  
 age INT,  
 mail VARCHAR(30),  
 grade INT,  
 phone\_number VARCHAR(10),  
 grupa VARCHAR(3),  
 form VARCHAR(3)  
  
)  
""")  
connection.commit()  
connection.close()  
  
connection = sqlite3.connect('students.db')  
cursor = connection.cursor()  
  
cursor.execute("""INSERT INTO students (id, name, surname, gend, age, mail, grade, phone\_number, grupa, form)  
VALUES  
 (1, "Елена", "Филатова", "Ж", 19, "elena@mail.com", 2, "+79854562535", "A1", "Б"),  
 (2, "Иван", "Петров", "М", 20, "ivan@mail.com", 3, "+79123456789", "B2", "ВБ"),  
 (3, "Анна", "Смирнова", "Ж", 18, "anna@mail.com", 1, "+79098765432", "A1", "ВБ"),  
 (4, "Дмитрий", "Козлов", "М", 21, "dmitry@mail.com", 4, "+79234567890", "C3", "Ц"),  
 (5, "Ольга", "Новикова", "Ж", 19, "olga@mail.com", 2, "+79345678901", "A1", "Б"),  
 (6, "Алексей", "Морозов", "М", 20, "alex@mail.com", 3, "+79456789012", "B2", "Ц"),  
 (7, "Мария", "Волкова", "Ж", 18, "maria@mail.com", 1, "+79567890123", "A1", "Б"),  
 (8, "Сергей", "Павлов", "М", 22, "sergey@mail.com", 4, "+79678901234", "C3", "ВБ"),  
 (9, "Екатерина", "Семенова", "Ж", 19, "ekaterina@mail.com", 2, "+79789012345", "A1", "Б"),  
 (10, "Андрей", "Федоров", "М", 20, "andrey@mail.com", 3, "+79890123456", "B2", "Ц"),  
 (11, "Наталья", "Кузнецова", "Ж", 18, "natalya@mail.com", 1, "+79901234567", "A1", "ВБ"),  
 (12, "Артем", "Иванов", "М", 21, "artem@mail.com", 4, "+70012345678", "C3", "Б"),  
 (13, "Юлия", "Лебедева", "Ж", 19, "yulia@mail.com", 2, "+70123456789", "A1", "Б"),  
 (14, "Максим", "Соколов", "М", 20, "maxim@mail.com", 3, "+70234567890", "B2", "Ц"),  
 (15, "Анастасия", "Козлова", "Ж", 18, "nastya@mail.com", 1, "+70345678901", "A1", "ВБ"),  
 (16, "Павел", "Егоров", "М", 22, "pavel@mail.com", 4, "+70456789012", "C3", "ВБ"),  
 (17, "Виктория", "Орлова", "Ж", 19, "vika@mail.com", 2, "+70567890123", "A1", "Б"),  
 (18, "Кирилл", "Андреев", "М", 20, "kirill@mail.com", 3, "+70678901234", "B2", "Ц"),  
 (19, "Алина", "Макарова", "Ж", 18, "alina@mail.com", 1, "+70789012345", "A1", "Б"),  
 (20, "Денис", "Захаров", "М", 21, "denis@mail.com", 4, "+70890123456", "C3", "Б")  
  
""")  
student = cursor.fetchall()  
connection.commit()  
connection.close()  
  
  
def get\_students():  
 connection = sqlite3.connect('students.db')  
 cursor = connection.cursor()  
 cursor.execute('SELECT \* FROM students')  
 student = cursor.fetchall()  
  
 if student:  
 return f' {student}'  
 else:  
 return "no data"  
  
 connection.commit()  
 connection.close()  
  
  
def delete\_student\_sql(student\_id):  
 connection = sqlite3.connect('students.db')  
 cursor = connection.cursor()  
  
 cursor.execute("DELETE FROM students WHERE id = ?", (student\_id,))  
 connection.commit()  
 return True

«admin test.py»

from unittest.mock import patch  
import pytest  
from PyQt5.QtWidgets import QApplication, QMessageBox, QInputDialog  
from sqlGUI import LoginWindow, MainWindow, user\_db, delete\_student\_sql, get\_students  
import sqlite3  
import sys  
  
@pytest.fixture(scope="session")  
def sqlapp():  
 app = QApplication(sys.argv)  
 yield app  
 app.quit()  
  
  
def test\_login\_right(sqlapp):  
 login\_window = LoginWindow()  
 login\_window.edit\_login.setText("admin")  
 login\_window.edit\_password.setText("admin")  
 assert login\_window.on\_login\_clicked() == True  
  
def test\_login\_wrong(sqlapp):  
 login\_window = LoginWindow()  
 login\_window.edit\_login.setText("admin")  
 login\_window.edit\_password.setText("admin1")  
 assert login\_window.on\_login\_clicked() is None  
  
def test\_window\_open(sqlapp, monkeypatch):  
 login\_window = LoginWindow()  
 login\_window.edit\_login.setText("admin")  
 login\_window.edit\_password.setText("admin")  
  
 mock\_mainwindow = None  
  
 def mock\_show(self):  
 nonlocal mock\_mainwindow  
 mock\_mainwindow = self  
  
 monkeypatch.setattr(MainWindow, 'show', mock\_show)  
  
 login\_window.on\_login\_clicked()  
 assert isinstance(mock\_mainwindow, MainWindow)  
  
def test\_delete\_student\_from\_db():  
  
 connection = sqlite3.connect('students.db')  
 cursor = connection.cursor()  
 cursor.execute(  
 "INSERT INTO students (id, name, surname, gend, age, mail, grade, phone\_number)"  
 " VALUES (99, 'Test', 'Student', 'М', 20, 'test@mail.com', 1, '1234567890')")  
 connection.commit()  
 connection.close()  
  
 students\_before = get\_students()  
 assert "'Test'" in students\_before  
  
 delete\_student\_sql(99)  
  
 students\_after = get\_students()  
 assert "'Test'" not in students\_after  
  
  
def test\_add\_student\_invalid\_input():  
  
 app = MainWindow()  
  
  
 with patch.object(QInputDialog, 'exec\_', return\_value=True), \  
 patch.object(QInputDialog, 'textValue',  
 return\_value="Иван,Иванов,X,двадцать,ivan@mail.com,2,+79123456789,A101,Б"), \  
 patch.object(QMessageBox, 'warning') as mock\_warning:  
 app.add\_student()  
  
  
 mock\_warning.assert\_called\_once()  
 assert "Пол должен быть M или Ж" in mock\_warning.call\_args[0][2] or \  
 "Возраст и курс должны быть числами" in mock\_warning.call\_args[0][2]