Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения» (ГУАП)

ОТЧЕТ ЗАЩИЩЕН

доцент Щёкин С.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Должность Ф.И.О дата, подпись

**Отчет**

**о лабораторной работе №7**

**«Создание двумерного пользовательского интерфейса в среде Qt Creator»**

по дисциплине «Проектирование человеко-машинного интерфейса»

ОТЧЕТ ВЫПОЛНИЛ:

Студент Самарин Д. В 4134К \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. группа дата, подпись

Санкт-Петербург

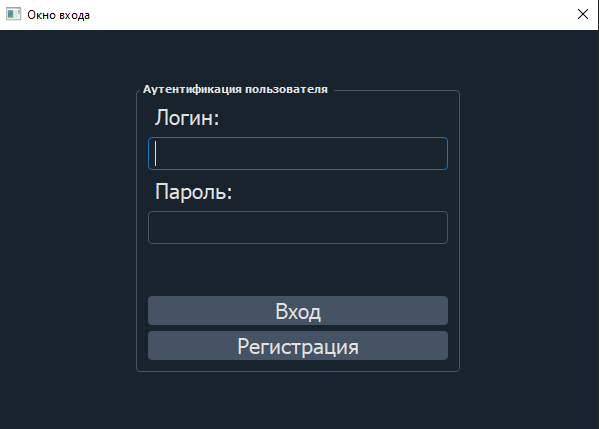
2024

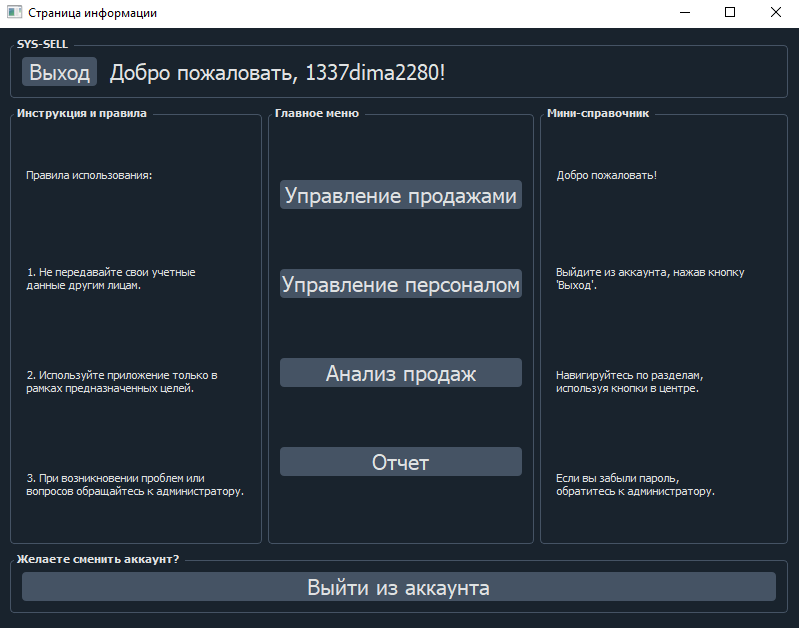
**Цель работы:** освоение кроссплатформенной интегрированной среды разработки, изучение базовых возможностей разработки и отладки приложения с двумерным пользовательским интерфейсом с использованием библиотеки Qt и компилятора MinGW

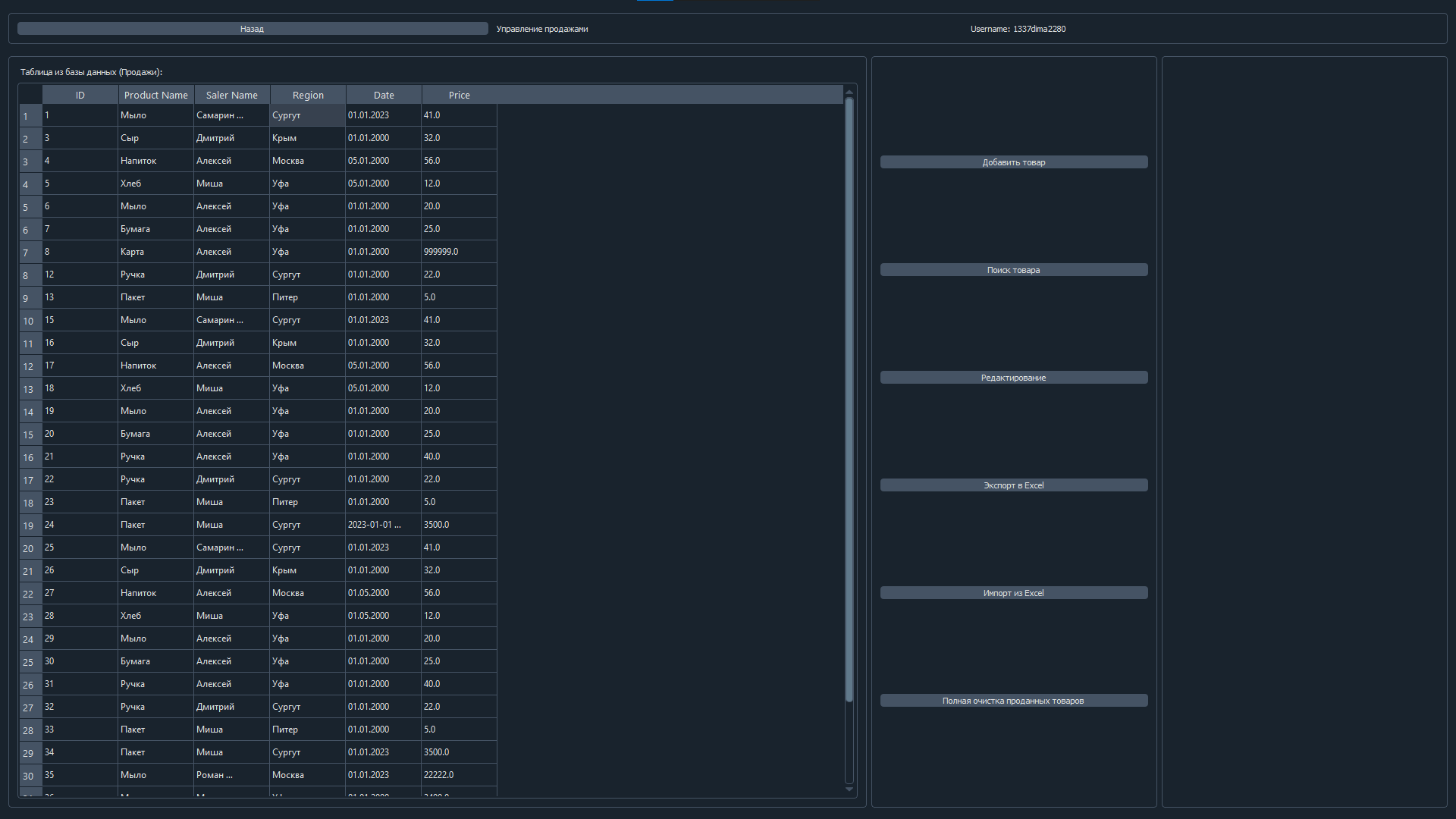
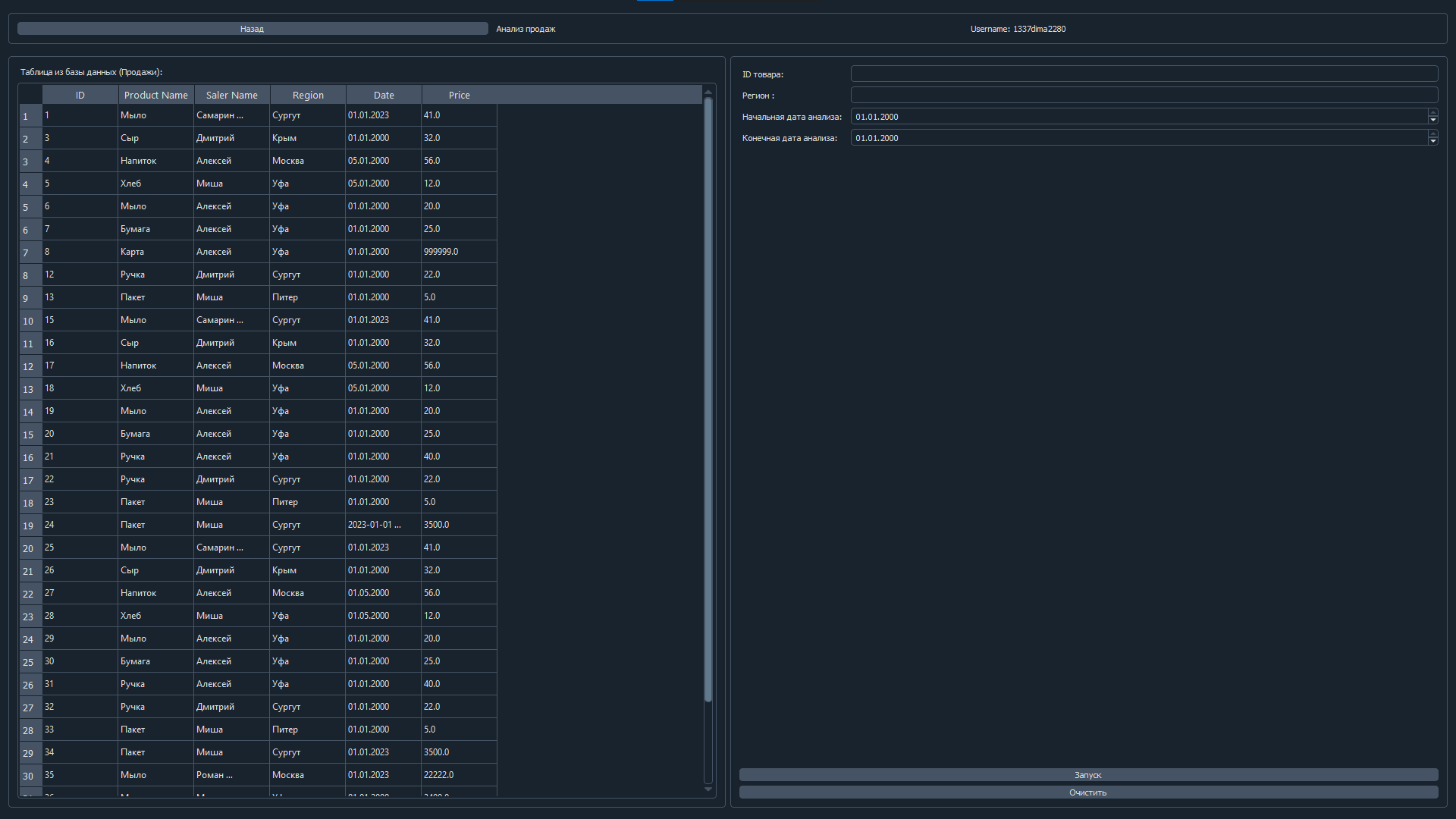
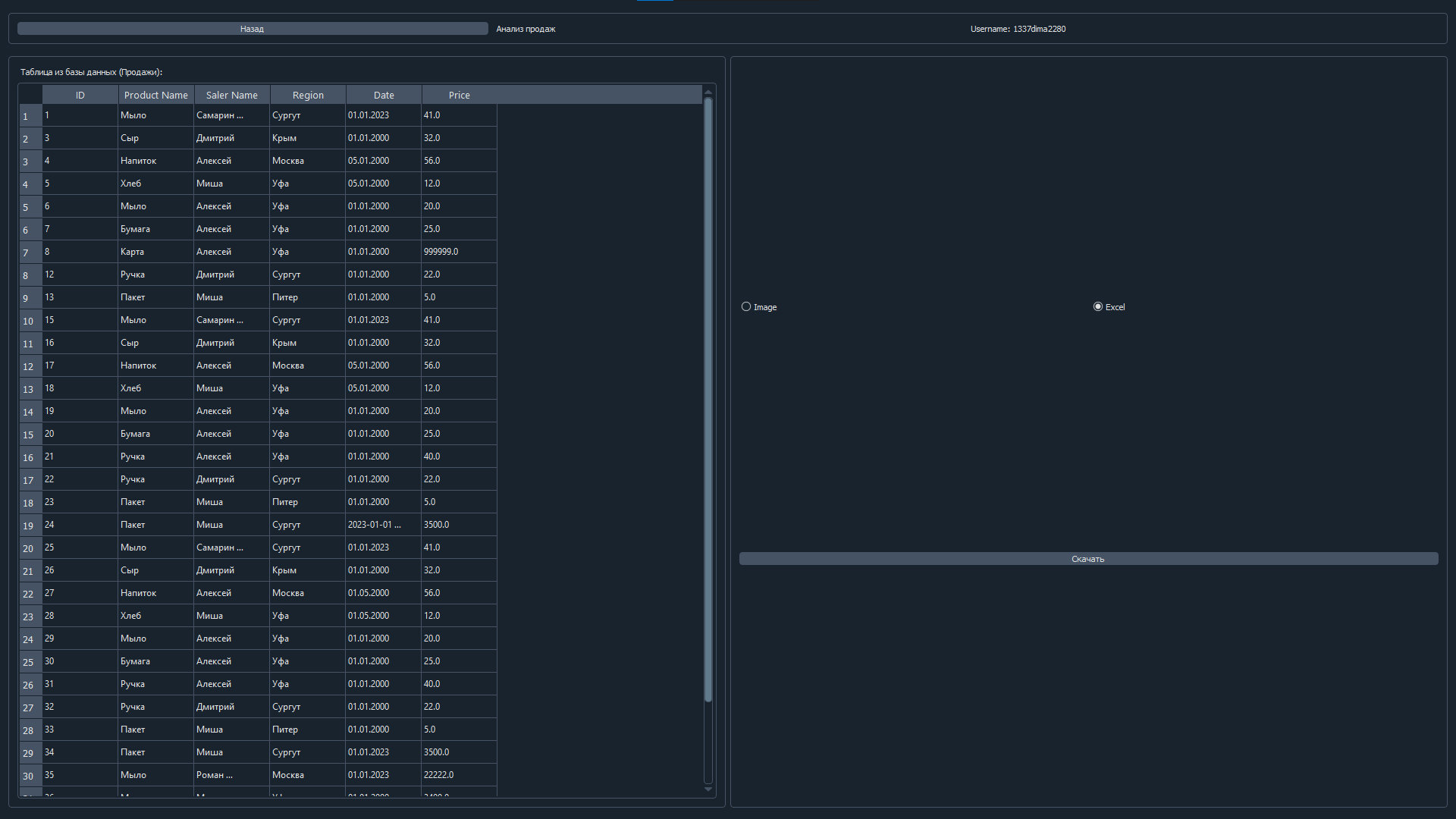
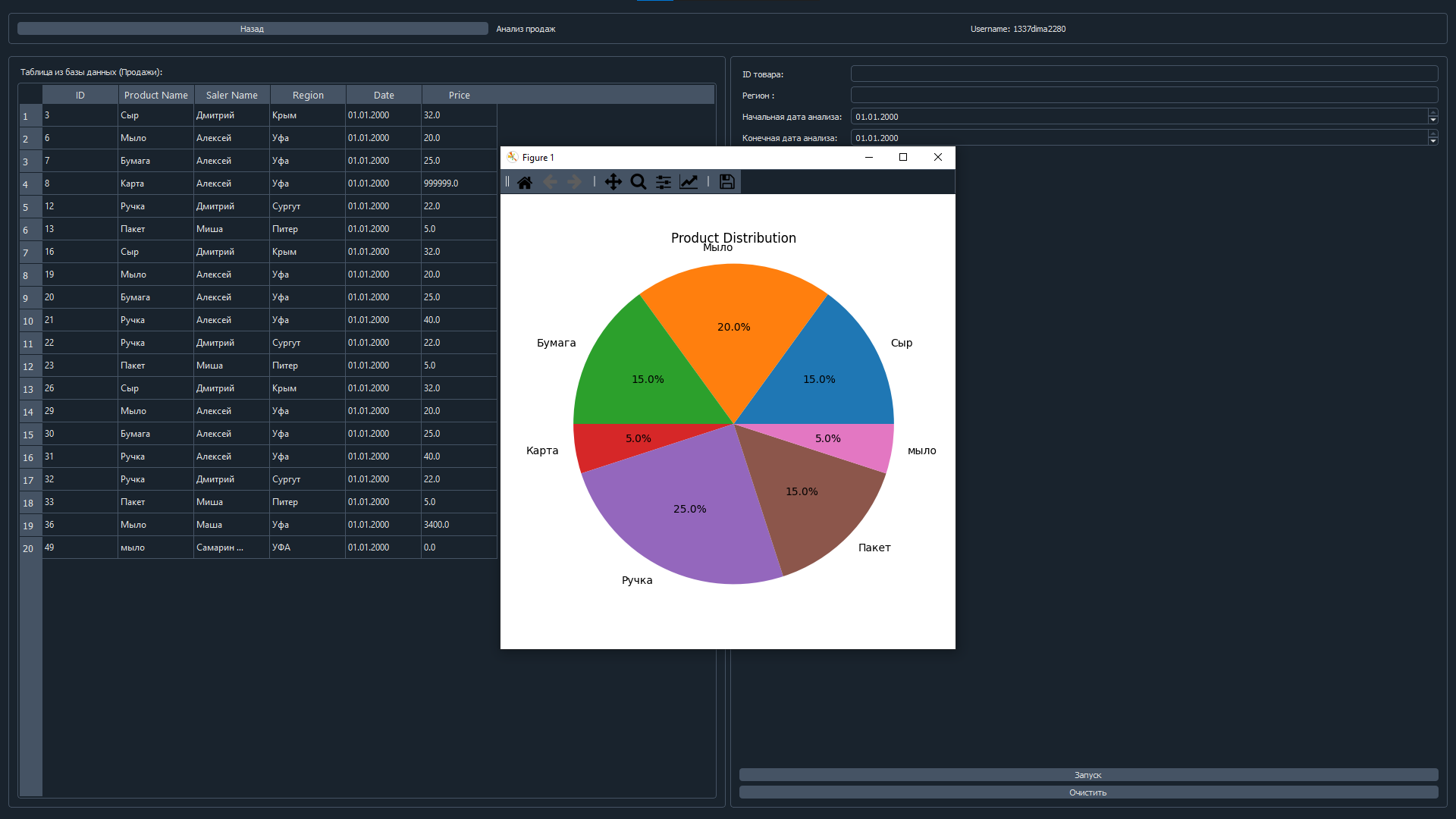
**Задание:**

Разработать и отладить приложение на языке С++ с графическим пользовательским интерфейсом, использующим основные виджеты Qt.

**Результаты работы:**

****

****

**** ****  

Из приведённых скриншоты мы видим, что был реализован полноценный интерфейс с интегрированной базой данных, позволяющий работе с проданными товарами. Программ включает в себя все основные виджеты.

Что реализовано:

* Логирование, включая регистрацию пользователей;
* Работа с базой данных пользователей/товаров;
* Анализ товаров;
* Отчётность.

Для каждого пользователя прописаны права 3-х уровней.

**Листинг:**

|  |
| --- |
| **app.py**  import sys  import sqlite3  from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QGroupBox, QHBoxLayout, QFormLayout, QDialog, QMessageBox, QDesktopWidget  from PyQt5.QtCore import Qt  from PyQt5.QtGui import QFont  import qdarkstyle  import re  from users\_edit\_window import UserManagementWindow  from sales\_manager\_window import SalesManagementWindow  from sales\_analysis import SalesAnalysisWindow  from report\_window import ReportAnalysisWindow  class LoginWindow(QWidget):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.setWindowTitle("Окно входа")          self.setGeometry(0, 0, 600, 400)  # Increase initial size          self.setWindowFlags(Qt.Window | Qt.CustomizeWindowHint | Qt.WindowTitleHint | Qt.WindowCloseButtonHint)          # Initialize the database connection          self.init\_db()          self.init\_ui()          self.register\_window = RegisterWindow()          # Center the window after all widgets are added          self.center\_on\_screen()      def show\_register\_window(self):          self.hide()  # Hide the login window          self.register\_window.show()      def init\_db(self):          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS users (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  full\_name TEXT NOT NULL,                  username TEXT NOT NULL,                  password TEXT NOT NULL,                  permissions TEXT NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          self.label\_username = QLabel("Логин:")          self.label\_password = QLabel("Пароль:")          self.input\_username = QLineEdit()          self.input\_password = QLineEdit()          self.warning\_label = QLabel()          self.login\_button = QPushButton("Вход")          self.register\_button = QPushButton("Регистрация")            # Increase font size for labels and buttons          font = QFont()          font.setPointSize(16)  # Set font size to 16          self.label\_username.setFont(font)          self.label\_password.setFont(font)          self.login\_button.setFont(font)          self.register\_button.setFont(font)          self.input\_username.setFont(font)          self.input\_password.setFont(font)          # Increase size of input fields          self.input\_username.setMinimumWidth(300)          self.input\_password.setMinimumWidth(300)          # Increase size of the warning label          self.warning\_label.setFixedHeight(40)          self.input\_password.setEchoMode(QLineEdit.Password)          group\_box = QGroupBox("Аутентификация пользователя")          group\_layout = QVBoxLayout()          group\_layout.addWidget(self.label\_username)          group\_layout.addWidget(self.input\_username)          group\_layout.addWidget(self.label\_password)          group\_layout.addWidget(self.input\_password)          group\_layout.addWidget(self.warning\_label)          group\_layout.addWidget(self.login\_button)          group\_layout.addWidget(self.register\_button)          group\_box.setLayout(group\_layout)          main\_layout = QVBoxLayout()          main\_layout.addStretch(1)          main\_layout.addWidget(group\_box, 2, alignment=Qt.AlignCenter)          main\_layout.addStretch(1)          self.setLayout(main\_layout)          self.login\_button.clicked.connect(self.authenticate)          self.register\_button.clicked.connect(self.show\_register\_window)        def authenticate(self):          username = self.input\_username.text()          password = self.input\_password.text()          query = "SELECT \* FROM users WHERE username=? AND password=?"          self.cursor.execute(query, (username, password))          user = self.cursor.fetchone()          if user:              self.warning\_label.clear()              self.info\_page = InfoPage(self, username)              self.info\_page.show()              self.hide()          else:              self.warning\_label.setText("Неправильный логин пользователя или пароль.")              self.warning\_label.setStyleSheet("color: red;")      def center\_on\_screen(self):          screen\_geometry = QDesktopWidget().screenGeometry()          center\_point = screen\_geometry.center()          new\_x = center\_point.x() - self.width() // 2          new\_y = center\_point.y() - self.height() // 2          self.move(new\_x, new\_y)  class RegisterWindow(QDialog):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.setWindowTitle("Окно регистрации")          self.setGeometry(0, 0, 600, 400)  # Set initial size, will be adjusted later          self.setWindowFlags(Qt.Window | Qt.CustomizeWindowHint | Qt.WindowTitleHint | Qt.WindowCloseButtonHint)          # Create input widgets          self.username\_input = QLineEdit(self)          self.full\_name = QLineEdit(self)          self.password\_input = QLineEdit(self)          self.pin\_input = QLineEdit(self)          # Set up the layout          self.init\_ui()          self.init\_db()          # Center the window after all widgets are added          self.center\_on\_screen()      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store user credentials if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS users (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  full\_name TEXT NOT NULL,                  username TEXT NOT NULL,                  password TEXT NOT NULL,                  permissions TEXT NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Set up the layout for the register window          form\_layout = QFormLayout()          form\_layout.addRow("Полное имя:", self.full\_name)          form\_layout.addRow("Логин:", self.username\_input)          form\_layout.addRow("Пароль:", self.password\_input)          form\_layout.addRow("ПИН-код:", self.pin\_input)          # Create register button          register\_button = QPushButton("Регистрация", self)          register\_button.clicked.connect(self.register\_user)          # Create back button          back\_button = QPushButton("Назад", self)          back\_button.clicked.connect(self.back\_to\_login)          # Apply same font style          font = QFont()          font.setPointSize(16)  # Set font size to 16          register\_button.setFont(font)          back\_button.setFont(font)          self.full\_name.setFont(font)          self.username\_input.setFont(font)          self.password\_input.setFont(font)          self.pin\_input.setFont(font)          # Add widgets to the layout          main\_layout = QVBoxLayout(self)          main\_layout.addLayout(form\_layout)          main\_layout.addWidget(register\_button)          main\_layout.addWidget(back\_button)          # Add QLabel for warning message          self.warning\_label = QLabel()          self.warning\_label.setStyleSheet("color: red;")          main\_layout.addWidget(self.warning\_label)          # Create a group box and add the main layout to it          group\_box = QGroupBox("Регистрация нового пользователя")          group\_box.setLayout(main\_layout)          # Create a main layout for the dialog and add the group box to it          dialog\_layout = QVBoxLayout()          dialog\_layout.addWidget(group\_box)          # Set the layout for the dialog          self.setLayout(dialog\_layout)      def check\_password\_strength(self, password):          # Check if password contains at least one uppercase letter and one digit          return len(password) >= 8 and re.search(r'[A-Z]', password) and re.search(r'\d', password)      def register\_user(self):          full\_name = self.full\_name.text()          username = self.username\_input.text()          password = self.password\_input.text()          pin = self.pin\_input.text()          # Проверяем, существует ли уже такой логин          query = "SELECT \* FROM users WHERE username = ?"          self.cursor.execute(query, (username,))          existing\_user = self.cursor.fetchone()            if pin != '1234':              QMessageBox.warning(self, "Неверный ПИН-код", "Пожалуйста, введите правильный ПИН-код.", QMessageBox.Ok)              return            if existing\_user:              self.warning\_label.setText("Такой логин уже существует. Пожалуйста, выберите другой.")              return            # Check if full name length is between 1 and 70 characters          if not (1 <= len(full\_name) <= 70):              self.warning\_label.setText("Ошибка, проверьть ФИО пользователя")              return            # Проверяем, содержит ли логин только английские буквы и спецсимволы          if not re.match(r'^[a-zA-Z0-9\_]+$', username):              self.warning\_label.setText("Логин может содержать только английские буквы и спецсимволы.")              return          # Check if password meets strength requirements          if not self.check\_password\_strength(password):              self.warning\_label.setText("Слишком слабый пароль (минимум 8 символов и 1 заглаваная буква, язык - анг-й).")              return          permissions = 'Average'          # Insert new user data into the database          query = "INSERT INTO users (full\_name, username, password, permissions) VALUES (?, ?, ?, ?)"          self.cursor.execute(query, (full\_name, username, password, permissions))          self.conn.commit()          # Show success message          QMessageBox.information(self, "Успешная регистрация", "Вы успешно зарегистрированы.", QMessageBox.Ok)          # Hide the dialog after registration          self.hide()          # Show the login window          self.login\_window = LoginWindow()          self.login\_window.show()      def back\_to\_login(self):          self.hide()          self.login\_window = LoginWindow()          self.login\_window.show()      def center\_on\_screen(self):          screen\_geometry = QDesktopWidget().screenGeometry()          center\_point = screen\_geometry.center()          new\_x = center\_point.x() - self.width() // 2          new\_y = center\_point.y() - self.height() // 2          self.move(new\_x, new\_y)  class InfoPage(QWidget):      def \_\_init\_\_(self, login\_window, username):          super().\_\_init\_\_()          self.setWindowTitle("Страница информации")          self.setGeometry(0, 0, 800, 600)  # Set initial size, will be adjusted later          self.username = username          self.login\_window = login\_window          self.init\_ui()          self.init\_db()          # Center the window after all widgets are added          self.center\_on\_screen()      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store user credentials if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS users (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  full\_name TEXT NOT NULL,                  username TEXT NOT NULL,                  password TEXT NOT NULL,                  permissions TEXT NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Widgets          welcome\_label = QLabel(f"Добро пожаловать, {self.username}!")          exit\_button = QPushButton("Выход")          main\_menu\_button = QPushButton("Управление продажами")          manager\_button = QPushButton("Управление персоналом")          analysis\_button = QPushButton("Анализ продаж")          report\_button = QPushButton("Отчет")          patterns\_button = QPushButton("Паттерн")          logout\_button = QPushButton("Выйти из аккаунта")          # Increase font size for labels and buttons          font = QFont()          font.setPointSize(16)  # Set font size to 16          welcome\_label.setFont(font)          exit\_button.setFont(font)          main\_menu\_button.setFont(font)          manager\_button.setFont(font)          analysis\_button.setFont(font)          report\_button.setFont(font)          patterns\_button.setFont(font)          logout\_button.setFont(font)          # Layouts          main\_layout = QVBoxLayout()          # Top Section          top\_layout = QHBoxLayout()          top\_layout.addWidget(exit\_button, 10)          top\_layout.addWidget(welcome\_label, 90)          top\_groupbox = QGroupBox("SYS-SELL")          top\_groupbox.setLayout(top\_layout)          main\_layout.addWidget(top\_groupbox, 10)          # Middle Section          middle\_layout = QHBoxLayout()          left\_groupbox = QGroupBox("Инструкция и правила")          left\_layout = QVBoxLayout()          left\_layout.addWidget(QLabel("Правила использования:"))          left\_layout.addWidget(QLabel("1. Не передавайте свои учетные \nданные другим лицам."))          left\_layout.addWidget(QLabel("2. Используйте приложение только в \nрамках предназначенных целей."))          left\_layout.addWidget(QLabel("3. При возникновении проблем или \nвопросов обращайтесь к администратору."))          left\_groupbox.setLayout(left\_layout)          center\_groupbox = QGroupBox("Главное меню")          center\_layout = QVBoxLayout()          center\_layout.addWidget(main\_menu\_button)          center\_layout.addWidget(manager\_button)          center\_layout.addWidget(analysis\_button)          center\_layout.addWidget(report\_button)          center\_groupbox.setLayout(center\_layout)          right\_groupbox = QGroupBox("Мини-справочник")          right\_layout = QVBoxLayout()          right\_layout.addWidget(QLabel("Добро пожаловать!"))          right\_layout.addWidget(QLabel("Выйдите из аккаунта, нажав кнопку \n'Выход'."))          right\_layout.addWidget(QLabel("Навигируйтесь по разделам, \nиспользуя кнопки в центре."))          right\_layout.addWidget(QLabel("Если вы забыли пароль, \nобратитесь к администратору."))          right\_groupbox.setLayout(right\_layout)          middle\_layout.addWidget(left\_groupbox, 35)          middle\_layout.addWidget(center\_groupbox, 30)          middle\_layout.addWidget(right\_groupbox, 35)          # Set overall layout          main\_layout.addLayout(middle\_layout, 80)          # Bottom Section          bottom\_layout = QHBoxLayout()          bottom\_layout.addWidget(logout\_button, 10)          bottom\_groupbox = QGroupBox("Желаете сменить аккаунт?")          bottom\_groupbox.setLayout(bottom\_layout)          main\_layout.addWidget(bottom\_groupbox, 10)          # Signals and slots          exit\_button.clicked.connect(self.exit\_program)          main\_menu\_button.clicked.connect(self.go\_to\_main\_menu)          manager\_button.clicked.connect(self.go\_to\_management)          analysis\_button.clicked.connect(self.go\_to\_analysis)          report\_button.clicked.connect(self.go\_to\_report)          # patterns\_button.clicked.connect(self.go\_to\_patterns)          logout\_button.clicked.connect(self.logout)          self.setLayout(main\_layout)      def exit\_program(self):          sys.exit(app.exec\_())      def go\_to\_main\_menu(self):          permissions = self.get\_permissions()          if permissions == "Full" or permissions == "Limited" or permissions == "Average" or permissions == "Полный доступ" or permissions == "Средний доступ":              self.info\_page = SalesManagementWindow(self, self.login\_window, self.username)              self.info\_page.show()              self.hide()          else:              QMessageBox.warning(self, "Отказано в доступе", "У вас недостаточно прав, обратитесь к администратору.", QMessageBox.Ok)      def go\_to\_analysis(self):          self.info\_page = SalesAnalysisWindow(self, self.login\_window, self.username)          self.info\_page.show()          self.hide()      def go\_to\_report(self):          self.info\_page = ReportAnalysisWindow(self, self.login\_window, self.username)          self.info\_page.show()          self.hide()      # def go\_to\_patterns(self):      #     self.info\_page = SalesAnalysisWindow(self, self.username)      #     self.info\_page.show()      #     self.hide()      def go\_to\_management(self):          permissions = self.get\_permissions()          if permissions == "Full" or permissions == "Полный доступ":              self.info\_page = UserManagementWindow(self, self.login\_window, self.username)              self.info\_page.show()              self.hide()          else:              QMessageBox.warning(self, "Отказано в доступе", "У вас недостаточно прав, обратитесь к администратору.", QMessageBox.Ok)      def logout(self):          self.login\_page = LoginWindow()          self.login\_page.show()          self.hide()      def get\_permissions(self):          query = "SELECT permissions FROM users WHERE username = ?"          self.cursor.execute(query, (self.username,))          result = self.cursor.fetchone()          return result[0] if result else None      def center\_on\_screen(self):          screen\_geometry = QDesktopWidget().screenGeometry()          center\_point = screen\_geometry.center()          new\_x = center\_point.x() - self.width() // 2          new\_y = center\_point.y() - self.height() // 2          self.move(new\_x, new\_y)  if \_\_name\_\_ == '\_\_main\_\_':      app = QApplication(sys.argv)      login\_window = LoginWindow()      login\_window.show()      app.setStyleSheet(qdarkstyle.load\_stylesheet\_from\_environment())      sys.exit(app.exec\_())  **report\_window.py**  import openpyxl  import sys  from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QPushButton, QTableWidget, QTableWidgetItem, QGroupBox, QFormLayout, QLineEdit, QDateTimeEdit, QDoubleSpinBox, QDesktopWidget, QFileDialog, QRadioButton  from PyQt5.QtCore import Qt, QFileInfo  from PyQt5.QtGui import QTextDocumentWriter  import pandas as pd  import sqlite3  import matplotlib.pyplot as plt  class ReportAnalysisWindow(QWidget):      def \_\_init\_\_(self, previous\_window, login\_window, username):          super().\_\_init\_\_()            self.setWindowTitle("Анализ продаж")          self.showFullScreen()          self.previous\_window = previous\_window          self.login\_window = login\_window          self.username = username          self.init\_ui()          self.init\_db()          self.populate\_sales\_table()      def setFullScreen(self):          desktop = QApplication.desktop()          screen\_geometry = desktop.screenGeometry()          self.setGeometry(screen\_geometry)      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store sales information if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS sales (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  product\_name TEXT NOT NULL,                  saler\_name TEXT NOT NULL,                  region TEXT NOT NULL,                  date TEXT NOT NULL,                  price REAL NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Top GroupBox with buttons and labels          top\_layout = QHBoxLayout()          back\_button = QPushButton("Назад")          window\_name\_label = QLabel("Анализ продаж")          username\_label = QLabel(f"Username: {self.username}")          top\_layout.addWidget(back\_button)          top\_layout.addWidget(window\_name\_label)          top\_layout.addWidget(username\_label)          top\_groupbox = QGroupBox()          top\_groupbox.setLayout(top\_layout)          # Left GroupBox: Table from DB (including sales information)          left\_layout = QVBoxLayout()          table\_label = QLabel("Таблица из базы данных (Продажи):")          self.sales\_table = QTableWidget()          self.sales\_table.setColumnCount(6)          self.sales\_table.setHorizontalHeaderLabels(["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])          left\_layout.addWidget(table\_label)          left\_layout.addWidget(self.sales\_table)          left\_groupbox = QGroupBox()          left\_groupbox.setLayout(left\_layout)          # Right GroupBox: Form with inputs and buttons for analysis          right\_layout = QVBoxLayout()          form\_layout = QFormLayout()          # self.input\_id = QLineEdit()            # form\_layout.addRow("Название файла:", self.input\_id)          button\_save\_file = QPushButton("Скачать")              # Radio buttons for choosing report format          self.radio\_img = QRadioButton("Image")          self.radio\_excel = QRadioButton("Excel")          self.radio\_excel.setChecked(True)  # PDF selected by default            report\_format\_layout = QHBoxLayout()          report\_format\_layout.addWidget(self.radio\_img)          report\_format\_layout.addWidget(self.radio\_excel)          right\_layout.addLayout(form\_layout)          right\_layout.addLayout(report\_format\_layout)          right\_layout.addWidget(button\_save\_file)          right\_groupbox = QGroupBox()          right\_groupbox.setLayout(right\_layout)          # Main Layout: Horizontal arrangement of left and right GroupBoxes          main\_layout = QHBoxLayout()          main\_layout.addWidget(left\_groupbox, 5)  # 50%          main\_layout.addWidget(right\_groupbox, 5)  # 50%          # Overall Layout: Vertical arrangement of top and main layouts          overall\_layout = QVBoxLayout()          overall\_layout.addWidget(top\_groupbox)          overall\_layout.addLayout(main\_layout)          self.setLayout(overall\_layout)          # Signals and slots for buttons (placeholder functions)          back\_button.clicked.connect(self.go\_back)          button\_save\_file.clicked.connect(self.save\_report)      def go\_back(self):          self.previous\_window.show()          self.hide()        def save\_report(self):          # Get the selected report format          if self.radio\_img.isChecked():              file\_extension = "png"          elif self.radio\_excel.isChecked():              file\_extension = "xlsx"          else:              return  # No format selected          # Open a file dialog to choose the file path and name          file\_path, \_ = QFileDialog.getSaveFileName(self, "Сохранить отчет", "", f"{file\_extension.upper()} files (\*.{file\_extension})")          print(file\_path)          if file\_path:              # Check if the file already exists              file\_info = QFileInfo(file\_path)              if file\_info.exists():                  return  # File already exists              # Get the data from the table              num\_rows = self.sales\_table.rowCount()              num\_cols = self.sales\_table.columnCount()              data = []              for row in range(num\_rows):                  row\_data = []                  for col in range(num\_cols):                      item = self.sales\_table.item(row, col)                      if item:                          row\_data.append(item.text())                      else:                          row\_data.append("")                  data.append(row\_data)              # Convert the data to a DataFrame              df = pd.DataFrame(data, columns=["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])              # Perform sales analysis              df["Price"] = pd.to\_numeric(df["Price"], errors="coerce")              # Count occurrences of each product              product\_counts = df["Product Name"].value\_counts()              # Calculate average occurrences              average\_occurrences = product\_counts.mean()              # Apply conditional formatting to highlight occurrences below and above the average              df\_style = df.style.apply(lambda x: ['background-color: red' if product\_counts.get(x['Product Name'], 0) < average\_occurrences else 'background-color: green' if product\_counts.get(x['Product Name'], 0) > average\_occurrences else '' for i in x], axis=1)              # Save the styled DataFrame to a file based on the selected format              if file\_extension == "png":                  df["Price"] = pd.to\_numeric(df["Price"], errors="coerce")                  product\_counts.plot(kind='bar')                  plt.xlabel("Product Name")                  plt.ylabel("Total Sales")                  plt.title("Total Sales per Product")                  plt.tight\_layout()                  # Save the plot to a temporary file                  temp\_plot\_file = file\_path                  print(temp\_plot\_file)                  plt.savefig(temp\_plot\_file)                  plt.close()              elif file\_extension == "xlsx":                  # Save the DataFrame to an Excel file                  df\_style.to\_excel(file\_path, index=False)                  # Additional code to save product sales plot to the same Excel file                  # Assuming you have the necessary imports and setup for plotting                  product\_sales = df.groupby("Product Name")["Price"].sum().reset\_index()                  # Save the DataFrame and the plot to the report file                  with pd.ExcelWriter(file\_path, mode="a", engine="openpyxl") as writer:                      df.to\_excel(writer, index=False, sheet\_name="Sales Data")                      product\_sales.to\_excel(writer, index=False, sheet\_name="Product Sales")      # def save\_report(self):      #     # Get the selected report format      #     if self.radio\_pdf.isChecked():      #         file\_extension = "pdf"      #     elif self.radio\_excel.isChecked():      #         file\_extension = "xlsx"      #     else:      #         return  # No format selected      #     # Open a file dialog to choose the file path and name      #     file\_path, \_ = QFileDialog.getSaveFileName(self, "Сохранить отчет", "", f"{file\_extension.upper()} files (\*.{file\_extension})")      #     if file\_path:      #         # Check if the file already exists      #         file\_info = QFileInfo(file\_path)      #         if file\_info.exists():      #             return  # File already exists      #         # Get the data from the table      #         num\_rows = self.sales\_table.rowCount()      #         num\_cols = self.sales\_table.columnCount()      #         data = []      #         for row in range(num\_rows):      #             row\_data = []      #             for col in range(num\_cols):      #                 item = self.sales\_table.item(row, col)      #                 if item:      #                     row\_data.append(item.text())      #                 else:      #                     row\_data.append("")      #             data.append(row\_data)      #         # Convert the data to a DataFrame      #         df = pd.DataFrame(data, columns=["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])      #         # Save the DataFrame to a file based on the selected format      #         if file\_extension == "pdf":      #             writer = QTextDocumentWriter(file\_path)      #             writer.setFormat("pdf")      #             writer.write(df.to\_html())      #         elif file\_extension == "xlsx":      #             df["Price"] = pd.to\_numeric(df["Price"], errors="coerce")      #             product\_sales = df.groupby("Product Name")["Price"].sum().reset\_index()      #             # Example: Plot bar chart for product sales          #             product\_sales.plot(kind="bar", x="Product Name", y="Price", rot=45)      #             plt.xlabel("Product Name")      #             plt.ylabel("Total Sales")      #             plt.title("Total Sales per Product")      #             plt.tight\_layout()      #             # Save the plot to a temporary file      #             temp\_plot\_file = "temp\_plot.png"      #             plt.savefig(temp\_plot\_file)      #             plt.close()      #             # Save the DataFrame and the plot to the report file      #             with pd.ExcelWriter(file\_path) as writer:      #                 df.to\_excel(writer, index=False, sheet\_name="Sales Data")      #                 product\_sales.to\_excel(writer, index=False, sheet\_name="Product Sales")          def populate\_sales\_table(self):          query = "SELECT \* FROM sales"          self.cursor.execute(query)          sales\_data = self.cursor.fetchall()          self.sales\_table.setRowCount(len(sales\_data))          for row, sale in enumerate(sales\_data):              for col, value in enumerate(sale):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)                  self.sales\_table.setItem(row, col, item)  if \_\_name\_\_ == '\_\_main\_\_':      app = QApplication(sys.argv)      username = "YourUsername"      sales\_analysis\_window = ReportAnalysisWindow('x', username)      sales\_analysis\_window.show()      sys.exit(app.exec\_())  **sales\_analysis.py**  import sys  from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QPushButton, QTableWidget, QTableWidgetItem, QGroupBox, QFormLayout, QLineEdit, QDateTimeEdit, QDoubleSpinBox, QDesktopWidget, QFileDialog  from PyQt5.QtCore import Qt  from PyQt5.QtCore import QDateTime  import sqlite3  import matplotlib.pyplot as plt  class SalesAnalysisWindow(QWidget):      def \_\_init\_\_(self, previous\_window, login\_window, username):          super().\_\_init\_\_()          self.setWindowTitle("Анализ продаж")          self.showFullScreen()          self.previous\_window = previous\_window          self.login\_window = login\_window          self.username = username          self.init\_ui()          self.init\_db()          self.populate\_sales\_table()      def setFullScreen(self):          desktop = QApplication.desktop()          screen\_geometry = desktop.screenGeometry()          self.setGeometry(screen\_geometry)      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store sales information if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS sales (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  product\_name TEXT NOT NULL,                  saler\_name TEXT NOT NULL,                  region TEXT NOT NULL,                  date TEXT NOT NULL,                  price REAL NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Top GroupBox with buttons and labels          top\_layout = QHBoxLayout()          back\_button = QPushButton("Назад")          window\_name\_label = QLabel("Анализ продаж")          username\_label = QLabel(f"Username: {self.username}")          top\_layout.addWidget(back\_button)          top\_layout.addWidget(window\_name\_label)          top\_layout.addWidget(username\_label)          top\_groupbox = QGroupBox()          top\_groupbox.setLayout(top\_layout)          # Left GroupBox: Table from DB (including sales information)          left\_layout = QVBoxLayout()          table\_label = QLabel("Таблица из базы данных (Продажи):")          self.sales\_table = QTableWidget()          self.sales\_table.setColumnCount(6)          self.sales\_table.setHorizontalHeaderLabels(["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])          left\_layout.addWidget(table\_label)          left\_layout.addWidget(self.sales\_table)          left\_groupbox = QGroupBox()          left\_groupbox.setLayout(left\_layout)          # Right GroupBox: Form with inputs and buttons for analysis          right\_layout = QVBoxLayout()          form\_layout = QFormLayout()          self.input\_id = QLineEdit()          self.input\_region = QLineEdit()          self.input\_start\_date = QDateTimeEdit()          self.input\_start\_date.setDisplayFormat("dd.MM.yyyy")          self.input\_start\_date.setDateTime(QDateTime())          self.input\_end\_date = QDateTimeEdit()          self.input\_end\_date.setDisplayFormat("dd.MM.yyyy")          self.input\_end\_date.setDateTime(QDateTime())            form\_layout.addRow("ID товара:", self.input\_id)          form\_layout.addRow("Регион :", self.input\_region)          form\_layout.addRow("Начальная дата анализа:", self.input\_start\_date)          form\_layout.addRow("Конечная дата анализа:", self.input\_end\_date)          button\_run\_analysis = QPushButton("Запуск")          button\_clear = QPushButton("Очистить")          right\_layout.addLayout(form\_layout)          right\_layout.addWidget(button\_run\_analysis)          right\_layout.addWidget(button\_clear)          right\_groupbox = QGroupBox()          right\_groupbox.setLayout(right\_layout)          # Main Layout: Horizontal arrangement of left and right GroupBoxes          main\_layout = QHBoxLayout()          main\_layout.addWidget(left\_groupbox, 5)  # 50%          main\_layout.addWidget(right\_groupbox, 5)  # 50%          # Overall Layout: Vertical arrangement of top and main layouts          overall\_layout = QVBoxLayout()          overall\_layout.addWidget(top\_groupbox)          overall\_layout.addLayout(main\_layout)          self.setLayout(overall\_layout)          # Signals and slots for buttons (placeholder functions)          back\_button.clicked.connect(self.go\_back)          button\_run\_analysis.clicked.connect(self.run\_analysis)          button\_clear.clicked.connect(self.clear\_form)      def go\_back(self):          self.previous\_window.show()          self.hide()      def run\_analysis(self):          # Get the search criteria from the input fields          search\_id = self.input\_id.text()          start\_date = self.input\_start\_date.dateTime().toString("dd.MM.yyyy")          end\_date = self.input\_end\_date.dateTime().toString("dd.MM.yyyy")          search\_region = self.input\_region.text()          # Construct the query based on the search criteria          query = "SELECT \* FROM sales WHERE "          params = []          if search\_id:              query += "id = ?"              params.append(search\_id)          if start\_date and end\_date:              if search\_id:                  query += " AND "              query += "date BETWEEN ? AND ?"              params.extend([start\_date, end\_date])          if search\_region:              if search\_id or start\_date or end\_date:                  query += " AND "              query += "region = ?"              params.append(search\_region)          # Execute the query and fetch the results          if params:              self.cursor.execute(query, params)              search\_results = self.cursor.fetchall()              # Populate the table with the search results              self.sales\_table.setRowCount(len(search\_results))              for row, result in enumerate(search\_results):                  for col, value in enumerate(result):                      item = QTableWidgetItem(str(value))                      item.setFlags(Qt.ItemIsEnabled)                      self.sales\_table.setItem(row, col, item)          else:              # No search criteria provided              # Display a message or handle it as needed              pass          self.generate\_pie\_chart(search\_results)      def generate\_pie\_chart(self, sales\_data):          # Extract product names and their frequencies from the sales data          product\_counts = {}          for sale in sales\_data:              product\_name = sale[1]              if product\_name in product\_counts:                  product\_counts[product\_name] += 1              else:                  product\_counts[product\_name] = 1          # Plot the pie chart          plt.figure(figsize=(6, 6))          plt.pie(product\_counts.values(), labels=product\_counts.keys(), autopct='%1.1f%%')          plt.title('Product Distribution')          plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle          plt.show()        def clear\_form(self):          query = "SELECT \* FROM sales"          self.cursor.execute(query)          search\_results = self.cursor.fetchall()          self.sales\_table.setRowCount(len(search\_results))          for row, result in enumerate(search\_results):              for col, value in enumerate(result):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)                  self.sales\_table.setItem(row, col, item)      def populate\_sales\_table(self):          query = "SELECT \* FROM sales"          self.cursor.execute(query)          sales\_data = self.cursor.fetchall()          self.sales\_table.setRowCount(len(sales\_data))          for row, sale in enumerate(sales\_data):              for col, value in enumerate(sale):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)                  self.sales\_table.setItem(row, col, item)  if \_\_name\_\_ == '\_\_main\_\_':      app = QApplication(sys.argv)      username = "YourUsername"      sales\_analysis\_window = SalesAnalysisWindow(username)      sales\_analysis\_window.show()      sys.exit(app.exec\_())  **sales\_manager\_window.py**  import sys  import pandas as pd  from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QPushButton, QTableWidget, QTableWidgetItem, QGroupBox, QFormLayout, QLineEdit, QDateTimeEdit, QDoubleSpinBox, QDesktopWidget, QFileDialog, QMessageBox  from PyQt5.QtCore import Qt  # Add this import  from PyQt5.QtWidgets import QPushButton  import sqlite3  class SalesManagementWindow(QWidget):      def \_\_init\_\_(self, previous\_window,login\_window, username):          super().\_\_init\_\_()          self.setWindowTitle("Управление продажами")          self.showFullScreen()            self.previous\_window = previous\_window          self.login\_window = login\_window          self.username = username          self.search\_warning\_label = QLabel()          self.init\_db()          self.init\_ui()      def setFullScreen(self):          desktop = QApplication.desktop()          screen\_geometry = desktop.screenGeometry()          self.setGeometry(screen\_geometry)      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store sales information if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS sales (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  product\_name TEXT NOT NULL,                  saler\_name TEXT NOT NULL,                  region TEXT NOT NULL,                  date TEXT NOT NULL,                  price REAL NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Top GroupBox with buttons and labels          top\_layout = QHBoxLayout()          back\_button = QPushButton("Назад")          # logout\_button = QPushButton("Назад")          window\_name\_label = QLabel("Управление продажами")          username\_label = QLabel(f"Username: {self.username}")          top\_layout.addWidget(back\_button)          top\_layout.addWidget(window\_name\_label)          top\_layout.addWidget(username\_label)          top\_groupbox = QGroupBox()          top\_groupbox.setLayout(top\_layout)          # Left GroupBox: Table from DB (including sales information)          left\_layout = QVBoxLayout()          table\_label = QLabel("Таблица из базы данных (Продажи):")          self.sales\_table = QTableWidget()          self.sales\_table.setColumnCount(6)          self.sales\_table.setHorizontalHeaderLabels(["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])          # Fetch and display sales information          query = "SELECT \* FROM sales"          self.cursor.execute(query)          sales\_data = self.cursor.fetchall()          self.sales\_table.setRowCount(len(sales\_data))          for row, sale in enumerate(sales\_data):              for col, value in enumerate(sale):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)  # Make cells read-only                  self.sales\_table.setItem(row, col, item)          left\_layout.addWidget(table\_label)          left\_layout.addWidget(self.sales\_table)          left\_groupbox = QGroupBox()          left\_groupbox.setLayout(left\_layout)          # Center GroupBox: Buttons only          center\_layout = QVBoxLayout()          button\_add = QPushButton("Добавить товар")          button\_search = QPushButton("Поиск товара")          button\_edit = QPushButton("Редактирование")          button\_export = QPushButton("Экспорт в Excel")          button\_import = QPushButton("Импорт из Excel")          button\_clear\_all = QPushButton("Полная очистка проданных товаров")          center\_layout.addWidget(button\_add)          center\_layout.addWidget(button\_search)          center\_layout.addWidget(button\_edit)          center\_layout.addWidget(button\_export)  # Add export button          center\_layout.addWidget(button\_import)  # Add import button          center\_layout.addWidget(button\_clear\_all)          center\_groupbox = QGroupBox()          center\_groupbox.setLayout(center\_layout)          # Right GroupBox: Form with inputs and buttons          self.right\_layout = QVBoxLayout()          self.form\_layout = QFormLayout()          # input\_product\_name = QLineEdit()          # input\_saler\_name = QLineEdit()          # input\_region = QLineEdit()          # input\_date = QDateTimeEdit()          # input\_date.setDisplayFormat("dd.MM.yyyy")          # input\_price = QDoubleSpinBox()          # input\_price.setMaximum(999999)          # self.form\_layout.addRow("Название товара:", input\_product\_name)          # self.form\_layout.addRow("ФИО:", input\_saler\_name)          # self.form\_layout.addRow("Регион:", input\_region)          # self.form\_layout.addRow("Дата:", input\_date)          # self.form\_layout.addRow("Стоимость товара:", input\_price)          self.button\_save = QPushButton("Сохранить")          self.button\_clear = QPushButton("Очистить")            # self.button\_load = QPushButton("Загрузить")          # self.right\_layout.addLayout(self.form\_layout)          # self.right\_layout.addWidget(self.button\_save)          # self.right\_layout.addWidget(self.button\_clear)          # self.right\_layout.addWidget(self.button\_load)          self.right\_groupbox = QGroupBox()          self.right\_groupbox.setLayout(self.right\_layout)          # Main Layout: Horizontal arrangement of left, center, and right GroupBoxes          self.main\_layout = QHBoxLayout()          self.main\_layout.addWidget(left\_groupbox, 6)  # 60%          self.main\_layout.addWidget(center\_groupbox, 2)  # 20%          self.main\_layout.addWidget(self.right\_groupbox, 2)  # 20%          # Overall Layout: Vertical arrangement of top and main layouts          self.overall\_layout = QVBoxLayout()          self.overall\_layout.addWidget(top\_groupbox)          self.overall\_layout.addLayout(self.main\_layout)          self.setLayout(self.overall\_layout)          # Signals and slots for buttons (placeholder functions)          back\_button.clicked.connect(self.go\_back)          # logout\_button.clicked.connect(self.logout)          # self.button\_save.clicked.connect(self.save\_data)          # self.button\_clear.clicked.connect(self.clear\_form)          # self.button\_load.clicked.connect(load\_data)          button\_add.clicked.connect(self.add\_product)          button\_search.clicked.connect(self.search\_product)          button\_edit.clicked.connect(self.edit\_product)          button\_export.clicked.connect(self.export\_to\_excel)          button\_import.clicked.connect(self.import\_from\_excel)          button\_clear\_all.clicked.connect(self.clear\_all\_sales)        def clear\_all\_sales(self):          # Ask for confirmation          confirmation = QMessageBox.question(self, "Подтверждение",                                              "Вы уверены, что хотите очистить все продажи?",                                              QMessageBox.Yes | QMessageBox.No)          if confirmation == QMessageBox.Yes:              # Clear all sales records from the database              query = "DELETE FROM sales"              self.cursor.execute(query)              self.conn.commit()              # Refresh the sales table to reflect the changes              self.refresh\_sales\_table()      # Placeholder functions for button actions      def save\_data(self):          print('SAVING')          product\_name = self.input\_product\_name.text()          saler\_name = self.input\_saler\_name.text()          region = self.input\_region.text()          date = self.input\_date.text()          price = self.input\_price.value()          # Validate the input data if needed          # Insert new sale data into the database          query = "INSERT INTO sales (product\_name, saler\_name, region, date, price) VALUES (?, ?, ?, ?, ?)"          self.cursor.execute(query, (product\_name, saler\_name, region, date, price))          self.conn.commit()          # Refresh the sales table          self.refresh\_sales\_table()          # Clear the input fields          self.clear\_form()      def clear\_form(self):          self.input\_product\_name.clear()          self.input\_saler\_name.clear()          self.input\_region.clear()          self.input\_date.clear()          self.input\_price.clear()      # def load\_data():      #     # Placeholder for loading data from Excel      #     file\_dialog = QFileDialog()      #     file\_path, \_ = file\_dialog.getOpenFileName(self, "Выберите файл Excel", "", "Excel Files (\*.xlsx \*.xls)")      #     if file\_path:      #         # Implement loading logic from Excel here      #         pass      def add\_product(self):          self.main\_layout.removeWidget(self.right\_groupbox)          self.right\_groupbox.deleteLater()            right\_layout = QVBoxLayout()          form\_layout = QFormLayout()            self.input\_product\_name = QLineEdit()          self.input\_saler\_name = QLineEdit()          self.input\_region = QLineEdit()          self.input\_date = QDateTimeEdit()          self.input\_date.setDisplayFormat("dd.MM.yyyy")          self.input\_price = QDoubleSpinBox()          self.input\_price.setMaximum(999999)          form\_layout.addRow("Название товара:", self.input\_product\_name)          form\_layout.addRow("ФИО:", self.input\_saler\_name)          form\_layout.addRow("Регион:", self.input\_region)          form\_layout.addRow("Дата:", self.input\_date)          form\_layout.addRow("Стоимость товара:", self.input\_price)          self.button\_save = QPushButton("Сохранить")          self.button\_clear = QPushButton("Очистить")          self.button\_save.clicked.connect(self.save\_data)          self.button\_clear.clicked.connect(self.clear\_form)          # self.button\_load = QPushButton("Загрузить")            form\_layout.addWidget(self.button\_save)          right\_layout.addWidget(self.button\_clear)          right\_layout.addLayout(form\_layout)            # right\_layout.addWidget(self.button\_load)          self.right\_groupbox = QGroupBox()          self.right\_groupbox.setLayout(right\_layout)          self.main\_layout.addWidget(self.right\_groupbox)      def search\_product(self):          # print('Pressed search')          self.main\_layout.removeWidget(self.right\_groupbox)          self.right\_groupbox.deleteLater()          # Create a new layout for the search form          new\_layout = QVBoxLayout()          # Form with inputs          form\_layout = QFormLayout()          self.input\_search\_id = QLineEdit()          self.input\_search\_product\_name = QLineEdit()          self.input\_search\_saler\_name = QLineEdit()          form\_layout.addRow("ID:", self.input\_search\_id)          form\_layout.addRow("Название товара:", self.input\_search\_product\_name)          form\_layout.addRow("ФИО:", self.input\_search\_saler\_name)          new\_layout.addLayout(form\_layout)          # Buttons          self.button\_search\_product = QPushButton("Поиск")          self.button\_clear\_search = QPushButton("Очистить")          new\_layout.addWidget(self.button\_search\_product)          new\_layout.addWidget(self.button\_clear\_search)          self.search\_warning\_label = QLabel()          new\_layout.addWidget(self.search\_warning\_label)          self.right\_groupbox = QGroupBox()          self.right\_groupbox.setLayout(new\_layout)          self.main\_layout.addWidget(self.right\_groupbox)          self.button\_search\_product.clicked.connect(self.search\_submit)          self.button\_clear\_search.clicked.connect(self.search\_clear)      def edit\_product(self):          self.main\_layout.removeWidget(self.right\_groupbox)          self.right\_groupbox.deleteLater()          edit\_layout = QVBoxLayout()          form\_edit\_layout = QFormLayout()          self.input\_edit\_id = QLineEdit()          self.input\_edit\_product\_name = QLineEdit()          self.input\_edit\_saler\_name = QLineEdit()          self.input\_edit\_region = QLineEdit()          self.input\_edit\_date = QDateTimeEdit()          self.input\_edit\_date.setDisplayFormat("dd.MM.yyyy")          self.input\_edit\_price = QDoubleSpinBox()          self.input\_edit\_price.setMaximum(999999)          form\_edit\_layout.addRow("ID:", self.input\_edit\_id)          label\_below\_id = QLabel("Введите новые данные:")          form\_edit\_layout.addRow(label\_below\_id)          form\_edit\_layout.addRow("Название товара:", self.input\_edit\_product\_name)          form\_edit\_layout.addRow("ФИО:", self.input\_edit\_saler\_name)          form\_edit\_layout.addRow("Регион:", self.input\_edit\_region)          form\_edit\_layout.addRow("Дата:", self.input\_edit\_date)          form\_edit\_layout.addRow("Стоимость товара:", self.input\_edit\_price)          button\_edit\_save = QPushButton("Сохранить")          button\_edit\_search = QPushButton("Поиск")          button\_edit\_del = QPushButton("Удалить")          edit\_layout.addLayout(form\_edit\_layout)          edit\_layout.addWidget(button\_edit\_save)          edit\_layout.addWidget(button\_edit\_search)          edit\_layout.addWidget(button\_edit\_del)          # Add the warning label          self.edit\_warning\_label = QLabel()          edit\_layout.addWidget(self.edit\_warning\_label)          # Set the new layout to the right group box          self.right\_groupbox = QGroupBox()          self.right\_groupbox.setLayout(edit\_layout)          self.main\_layout.addWidget(self.right\_groupbox)          button\_edit\_save.clicked.connect(self.edit\_save)          button\_edit\_search.clicked.connect(self.edit\_search)          button\_edit\_del.clicked.connect(self.edit\_delete)        def edit\_delete(self):          # Placeholder function for deleting a row by ID          edit\_id = self.input\_edit\_id.text()          # Check if the ID is provided          if not edit\_id:              self.edit\_warning\_label.setText("Please enter ID for deletion.")              self.edit\_warning\_label.setStyleSheet("color: red;")              return          # Confirm deletion with user          confirm\_dialog = QMessageBox()          confirm\_dialog.setIcon(QMessageBox.Warning)          confirm\_dialog.setText("Вы действительно хотите удалить информацию о товаре?")          confirm\_dialog.setStandardButtons(QMessageBox.Yes | QMessageBox.No)          result = confirm\_dialog.exec\_()          if result == QMessageBox.Yes:              # Delete the row from the database              query = "DELETE FROM sales WHERE id=?"              self.cursor.execute(query, (edit\_id,))              self.conn.commit()              # Display a success message or handle errors              self.edit\_warning\_label.setText("Record deleted successfully.")              self.edit\_warning\_label.setStyleSheet("color: green;")              # Clear the input fields after deletion              self.clear\_edit\_form()              self.refresh\_sales\_table()          else:              # User canceled the deletion              self.edit\_warning\_label.setText("Deletion canceled.")              self.edit\_warning\_label.setStyleSheet("color: orange;")      def clear\_edit\_form(self):          # Clear the input fields in the edit form          self.input\_edit\_id.clear()          self.input\_edit\_product\_name.clear()          self.input\_edit\_saler\_name.clear()          self.input\_edit\_region.clear()          self.input\_edit\_date.clear()          self.input\_edit\_price.clear()      def edit\_search(self):          edit\_id = self.input\_edit\_id.text()          # Validate input (you can customize the validation logic)          if not edit\_id.isdigit():              # Display error message in red text              self.search\_warning\_label.setText("Invalid ID. Please enter a valid numeric ID.")              self.search\_warning\_label.setStyleSheet("color: red;")              return          # Fetch data from the database based on the search criteria          query = "SELECT \* FROM sales WHERE id=?"          self.cursor.execute(query, (edit\_id,))          search\_result = self.cursor.fetchall()          # Display the search result in the table          self.sales\_table.setRowCount(len(search\_result))          for row, result in enumerate(search\_result):              for col, value in enumerate(result):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)  # Make cells read-only                  self.sales\_table.setItem(row, col, item)          # Clear the input fields          self.clear\_edit\_form()      def edit\_save(self):          # Placeholder function for saving edited data          edit\_id = self.input\_edit\_id.text()          new\_product\_name = self.input\_edit\_product\_name.text()          new\_saler\_name = self.input\_edit\_saler\_name.text()          new\_region = self.input\_edit\_region.text()          new\_date = self.input\_edit\_date.text()          new\_price = self.input\_edit\_price.value()          # Validate the input data if needed          # Update the existing sale data in the database          query = """          UPDATE sales          SET              product\_name = CASE WHEN ? != '' THEN ? ELSE product\_name END,              saler\_name = CASE WHEN ? != '' THEN ? ELSE saler\_name END,              region = CASE WHEN ? != '' THEN ? ELSE region END,              date = CASE WHEN ? != '' THEN ? ELSE date END,              price = CASE WHEN ? != '' THEN ? ELSE price END          WHERE id=?      """          self.cursor.execute(query, (new\_product\_name, new\_product\_name,                                      new\_saler\_name, new\_saler\_name,                                      new\_region, new\_region,                                      new\_date, new\_date,                                      new\_price, new\_price,                                      edit\_id))          self.conn.commit()          # Display a success message or handle errors          self.edit\_warning\_label.setText("Changes saved successfully.")          self.edit\_warning\_label.setStyleSheet("color: green;")          # Refresh the sales table          self.refresh\_sales\_table()          # Clear the input fields          # clear\_form()      def search\_submit(self):          # Get the input values          search\_id = self.input\_search\_id.text()          search\_product\_name = self.input\_search\_product\_name.text()          search\_saler\_name = self.input\_search\_saler\_name.text()          # Clear the warning label          self.search\_warning\_label.clear()          # # Validate input (you can customize the validation logic)          # if not search\_id.isdigit():          #     self.search\_warning\_label.setText("Invalid ID. Please enter a valid numeric ID.")          #     self.search\_warning\_label.setStyleSheet("color: red;")          #     return          # Construct the query based on the search criteria          query = "SELECT \* FROM sales WHERE "          params = []          if search\_id:              query += "id = ?"              params.append(search\_id)          if search\_product\_name:              if search\_id:                  query += " AND "              query += "product\_name = ?"              params.append(search\_product\_name)          if search\_saler\_name:              if search\_id or search\_product\_name:                  query += " AND "              query += "saler\_name = ?"              params.append(search\_saler\_name)          # Execute the query and fetch the results          if params:              self.cursor.execute(query, params)              search\_results = self.cursor.fetchall()              # Populate the table with the search results              self.sales\_table.setRowCount(len(search\_results))              for row, result in enumerate(search\_results):                  for col, value in enumerate(result):                      item = QTableWidgetItem(str(value))                      item.setFlags(Qt.ItemIsEnabled)                      self.sales\_table.setItem(row, col, item)          else:              # No search criteria provided              # Display a message or handle it as needed              query = "SELECT \* FROM sales"              self.cursor.execute(query)              search\_results = self.cursor.fetchall()              self.sales\_table.setRowCount(len(search\_results))              for row, result in enumerate(search\_results):                  for col, value in enumerate(result):                      item = QTableWidgetItem(str(value))                      item.setFlags(Qt.ItemIsEnabled)                      self.sales\_table.setItem(row, col, item)          # self.refresh\_sales\_table()          # Clear the input fields          self.search\_clear()      def search\_clear(self):          print('clear')          # Implement clear search form logic here          self.input\_search\_id.clear()          self.input\_search\_product\_name.clear()          self.input\_search\_saler\_name.clear()      def refresh\_sales\_table(self):          # Fetch and display sales information          query = "SELECT \* FROM sales"          self.cursor.execute(query)          sales\_data = self.cursor.fetchall()          self.sales\_table.setRowCount(len(sales\_data))          for row, sale in enumerate(sales\_data):              for col, value in enumerate(sale):                  item = QTableWidgetItem(str(value))                  item.setFlags(Qt.ItemIsEnabled)  # Make cells read-only                  self.sales\_table.setItem(row, col, item)        def logout(self):          self.login\_window.show()          self.hide()      # Placeholder functions for button actions      def go\_back(self):          self.previous\_window.show()          self.hide()  # Replace with actual logic        def export\_to\_excel(self):      # Fetch data from the database          query = "SELECT \* FROM sales"          self.cursor.execute(query)          sales\_data = self.cursor.fetchall()          # Convert data to pandas DataFrame          df = pd.DataFrame(sales\_data, columns=["ID", "Product Name", "Saler Name", "Region", "Date", "Price"])          # Choose file path for saving the Excel file          file\_path, \_ = QFileDialog.getSaveFileName(self, "Сохранить в Excel", "", "Excel Files (\*.xlsx)")          # Check if user selected a file path          if file\_path:              # Export DataFrame to Excel              df.to\_excel(file\_path, index=False)              QMessageBox.information(self, "Экспорт завершен", "Данные успешно экспортированы в Excel.")        def import\_from\_excel(self):          # Choose file path for importing the Excel file          file\_path, \_ = QFileDialog.getOpenFileName(self, "Импорт из Excel", "", "Excel Files (\*.xls \*.xlsx)")          # Check if user selected a file path          if file\_path:              try:                  # Read Excel file into pandas DataFrame                  df = pd.read\_excel(file\_path)                  # Convert date column to the desired format                  df['Date'] = pd.to\_datetime(df['Date']).dt.strftime('%d.%m.%Y')                  # Insert DataFrame records into the database                  for index, row in df.iterrows():                      query = "INSERT INTO sales (product\_name, saler\_name, region, date, price) VALUES (?, ?, ?, ?, ?)"                      self.cursor.execute(query, (row["Product Name"], row["Saler Name"], row["Region"], row["Date"], row["Price"]))                    self.conn.commit()                  self.refresh\_sales\_table()  # Refresh table after import                  QMessageBox.information(self, "Импорт завершен", "Данные успешно импортированы из Excel.")              except Exception as e:                  QMessageBox.critical(self, "Ошибка импорта", f"Произошла ошибка при импорте данных из Excel: {str(e)}")    if \_\_name\_\_ == '\_\_main\_\_':      app = QApplication(sys.argv)      username = "YourUsername"  # Replace with the actual username      sales\_management\_window = SalesManagementWindow(username)      sales\_management\_window.show()      sys.exit(app.exec\_())  **users\_edit\_window.py**  import sys  from PyQt5.QtWidgets import (      QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QTableWidget,      QTableWidgetItem, QFormLayout, QComboBox, QGroupBox, QFileDialog, QMessageBox  )  from PyQt5.QtCore import Qt  import sqlite3  import csv  import pandas as pd  class UserManagementWindow(QWidget):      def import\_users(self):          options = QFileDialog.Options()          file\_name, \_ = QFileDialog.getOpenFileName(self, "Import Users", "", "Excel Files (\*.xls \*.xlsx)", options=options)  # Update file filter          if file\_name:              try:                  users\_df = pd.read\_excel(file\_name)              except Exception as e:                  QMessageBox.critical(self, "Ошибка", f"Ошибка при импорте пользователей: {str(e)}")                  return                if users\_df.empty:                  QMessageBox.warning(self, "Внимание", "Файл с пользователями пуст.")                  return              users\_data = users\_df.values.tolist()              for user\_data in users\_data:                  if len(user\_data) == 4:                      full\_name, username, password, permissions = user\_data                      self.cursor.execute("INSERT INTO users (full\_name, username, password, permissions) VALUES (?, ?, ?, ?)",                                          (full\_name, username, password, permissions))              self.conn.commit()              self.load\_user\_data()      def export\_users(self):          options = QFileDialog.Options()          file\_name, \_ = QFileDialog.getSaveFileName(self, "Export Users", "", "Excel Files (\*.xlsx)", options=options)  # Update file filter          if file\_name:              query = "SELECT full\_name, username, password, permissions FROM users"              self.cursor.execute(query)              users = self.cursor.fetchall()              users\_df = pd.DataFrame(users, columns=["Full Name", "Username", "Password", "Permissions"])              users\_df.to\_excel(file\_name, index=False)  # Export to xlsx format      def \_\_init\_\_(self, previous\_window, login\_window, username):          super().\_\_init\_\_()          self.setWindowTitle("Управление пользователями")          self.showFullScreen()            self.previous\_window = previous\_window          self.login\_window = login\_window          self.username = username          self.init\_db()          self.init\_ui()      def setFullScreen(self):          desktop = QApplication.desktop()          screen\_geometry = desktop.screenGeometry()          self.setGeometry(screen\_geometry)      def init\_db(self):          # Connect to SQLite database (create a new one if it doesn't exist)          self.conn = sqlite3.connect("user\_database.db")          self.cursor = self.conn.cursor()          # Create a table to store user information if it doesn't exist          self.cursor.execute('''              CREATE TABLE IF NOT EXISTS users (                  id INTEGER PRIMARY KEY AUTOINCREMENT,                  full\_name TEXT NOT NULL,                  username TEXT NOT NULL,                  password TEXT NOT NULL,                  permissions TEXT NOT NULL              )          ''')          self.conn.commit()      def init\_ui(self):          # Widgets          user\_table\_label = QLabel("База данных пользователей:")          self.user\_table = QTableWidget()          self.user\_table.setColumnCount(5)          self.user\_table.setHorizontalHeaderLabels(["ID", "Full Name", "Username", "Password", "Permissions"])          # back\_button = QPushButton("Back")          # logout\_button = QPushButton("Logout")          # clear\_db\_button = QPushButton("Clear Database")          add\_user\_label = QLabel("Добавить нового пользователя:")          self.full\_name\_input = QLineEdit()          self.username\_input = QLineEdit()          self.password\_input = QLineEdit()          self.permissions\_combo = QComboBox()          # self.permissions\_combo.setGeometry(10, 10, 200, 30)          self.permissions\_combo.addItems(["Полный доступ", "Средний доступ", "Ограниченный доступ"])          add\_button = QPushButton("Добавить пользователя")          import\_button = QPushButton("Импортировать пользователей")          export\_button = QPushButton("Экспортировать пользователей")          search\_label = QLabel("Поиск пользователя по ID:")          self.search\_id\_input = QLineEdit()          search\_button = QPushButton("Найти")          delete\_button = QPushButton("Удалить пользователя")          edit\_label = QLabel("Редактировать пользователя:")          self.edit\_id\_input = QLineEdit()          self.edit\_full\_name\_input = QLineEdit()          self.edit\_username\_input = QLineEdit()          self.edit\_password\_input = QLineEdit()          self.edit\_permissions\_combo = QComboBox()          self.edit\_permissions\_combo.addItems(["Полный доступ", "Средний доступ", "Ограниченный доступ"])          save\_edit\_button = QPushButton("Сохранить изменения")          # Left Half: User Table Section          user\_table\_layout = QVBoxLayout()          user\_table\_layout.addWidget(user\_table\_label)          user\_table\_layout.addWidget(self.user\_table)          # user\_table\_layout.addWidget(back\_button)          # user\_table\_layout.addWidget(logout\_button)          # user\_table\_layout.addWidget(clear\_db\_button)          # Right Half: Divided Vertically          right\_half\_layout = QVBoxLayout()          # GroupBox 1: User Info          user\_info\_groupbox = QGroupBox("Информация о пользователях")          user\_info\_layout = QVBoxLayout()          user\_info\_label = QLabel(              "Этот раздел предоставляет возможность просматривать, добавлять, редактировать и удалять информацию о пользователях в системе.\n\n"              "Для просмотра данных пользователей, таблица отображает полные имена, имена пользователя, пароли и разрешения доступа к системе.\n\n"              "Для добавления нового пользователя, заполните поля 'Полное имя', 'Имя пользователя', 'Пароль' и выберите разрешения доступа из выпадающего списка, "              "затем нажмите кнопку 'Добавить пользователя'.\n\n"              "Для редактирования существующих пользователей, введите их ID, затем внесите необходимые изменения в соответствующие поля и нажмите кнопку 'Сохранить изменения'.\n\n"              "Для удаления одного или нескольких пользователей, введите их ID через запятую, затем нажмите кнопку 'Удалить пользователя'.\n\n"              "Кроме того, вы можете импортировать и экспортировать данные о пользователях в формате Excel, используя кнопки 'Импортировать пользователей' и 'Экспортировать пользователей' соответственно."          )          user\_info\_layout.addWidget(user\_info\_label)          user\_info\_layout.addWidget(user\_table\_label)          user\_info\_layout.addWidget(self.user\_table)          # user\_info\_layout.addWidget(back\_button)          # user\_info\_layout.addWidget(logout\_button)          # user\_info\_layout.addWidget(clear\_db\_button)            user\_info\_groupbox.setLayout(user\_info\_layout)          # GroupBox 2: Adding Users          add\_users\_groupbox = QGroupBox("Добавление пользователей")          add\_users\_layout = QFormLayout()          add\_users\_layout.addRow(add\_user\_label)          add\_users\_layout.addRow("Полное имя:", self.full\_name\_input)          add\_users\_layout.addRow("Имя пользователя:", self.username\_input)          add\_users\_layout.addRow("Пароль:", self.password\_input)          add\_users\_layout.addRow("Разрешения доступа:", self.permissions\_combo)          add\_users\_layout.addRow(add\_button)          add\_users\_layout.addRow(import\_button)  # Add import button            # add\_users\_layout.addRow(load\_button)          # add\_users\_layout.addRow(save\_button)          add\_users\_groupbox.setLayout(add\_users\_layout)          # GroupBox 3: Editing Users          edit\_users\_groupbox = QGroupBox("Редактирование пользователей")          edit\_users\_layout = QFormLayout()          edit\_users\_layout.addRow(edit\_label)          edit\_users\_layout.addRow("ID пользователя:", self.edit\_id\_input)          edit\_users\_layout.addRow("Полное имя:", self.edit\_full\_name\_input)          edit\_users\_layout.addRow("Имя пользователя:", self.edit\_username\_input)          edit\_users\_layout.addRow("Пароль:", self.edit\_password\_input)          edit\_users\_layout.addRow("Разрешения доступа:", self.edit\_permissions\_combo)          edit\_users\_layout.addRow(save\_edit\_button)          edit\_users\_groupbox.setLayout(edit\_users\_layout)          # GroupBox 4: Load/Save Database          # load\_save\_groupbox = QGroupBox("Load/Save Database")          # load\_save\_layout = QVBoxLayout()          # load\_save\_layout.addWidget(load\_button)          # load\_save\_layout.addWidget(save\_button)          # load\_save\_groupbox.setLayout(load\_save\_layout)          # GroupBox 5: Search/Delete Users          search\_delete\_groupbox = QGroupBox("Поиск/Удаление пользователей")          search\_delete\_layout = QVBoxLayout()          search\_delete\_layout.addWidget(search\_label)          search\_delete\_layout.addWidget(self.search\_id\_input)          search\_delete\_layout.addWidget(search\_button)          search\_delete\_layout.addWidget(delete\_button)          search\_delete\_layout.addWidget(export\_button)  # Add export button          search\_delete\_groupbox.setLayout(search\_delete\_layout)          # Add GroupBoxes to Right Half Layout          right\_half\_layout.addWidget(user\_info\_groupbox)          right\_half\_layout.addWidget(add\_users\_groupbox)          right\_half\_layout.addWidget(edit\_users\_groupbox)          # right\_half\_layout.addWidget(load\_save\_groupbox)          right\_half\_layout.addWidget(search\_delete\_groupbox)          # Top Layout: Back, Logout, and Username          top\_layout = QHBoxLayout()          back\_button = QPushButton("Назад")          logout\_button = QPushButton("Выход из аккаунта")          username\_label = QLabel(f"Username: {self.username}")          top\_layout.addWidget(back\_button)          top\_layout.addWidget(logout\_button)          top\_layout.addWidget(username\_label)          # Signals and slots          back\_button.clicked.connect(self.back\_function)          add\_button.clicked.connect(self.add\_user)          logout\_button.clicked.connect(self.logout)          search\_button.clicked.connect(self.search\_user)          delete\_button.clicked.connect(self.delete\_user)          save\_edit\_button.clicked.connect(self.save\_edit\_user)          import\_button.clicked.connect(self.import\_users)          export\_button.clicked.connect(self.export\_users)          # Layout          main\_layout = QHBoxLayout()          # main\_layout.addLayout(top\_layout)          main\_layout.addLayout(user\_table\_layout)          main\_layout.addLayout(right\_half\_layout)          full\_window\_layout = QVBoxLayout()          full\_window\_layout.addLayout(top\_layout)          full\_window\_layout.addLayout(main\_layout)          self.setLayout(full\_window\_layout)            # Reload user data          self.load\_user\_data()      def logout(self):          self.login\_window.show()          self.hide()      def back\_function(self):          self.previous\_window.show()          self.hide()      def load\_user\_data(self):          # Load user data from the database and update the table          query = "SELECT \* FROM users"          self.cursor.execute(query)          users = self.cursor.fetchall()          self.user\_table.setRowCount(0)          for user in users:              row\_position = self.user\_table.rowCount()              self.user\_table.insertRow(row\_position)              for column, data in enumerate(user):                  self.user\_table.setItem(row\_position, column, QTableWidgetItem(str(data)))          # self.edit\_permissions\_combo.addItems(["Full", "Average", "Limited"])      def add\_user(self):          # Get input values          full\_name = self.full\_name\_input.text()          username = self.username\_input.text()          password = self.password\_input.text()          permissions = self.permissions\_combo.currentText()           # Проверяем, что все поля ввода заполнены          if not full\_name or not username or not password:              QMessageBox.warning(self, "Пустые поля", "Пожалуйста, заполните все поля для добавления пользователя.")              return              # Проверяем уникальность имени пользователя          query = "SELECT \* FROM users WHERE username = ?"          self.cursor.execute(query, (username,))          existing\_user = self.cursor.fetchone()          if existing\_user:              QMessageBox.warning(self, "Пользователь уже существует", "Пользователь с таким именем уже существует. Пожалуйста, выберите другое имя пользователя.")              return          # Insert the new user into the database          query = "INSERT INTO users (full\_name, username, password, permissions) VALUES (?, ?, ?, ?)"          self.cursor.execute(query, (full\_name, username, password, permissions))          self.conn.commit()          # Clear input fields          self.full\_name\_input.clear()          self.username\_input.clear()          self.password\_input.clear()          # Reload user data          self.load\_user\_data()      def save\_user\_data(self):          # Save user data to a file (or take any other necessary action)          pass          def search\_user(self):          # self.edit\_full\_name\_input.clear()          # self.edit\_username\_input.clear()          # self.edit\_password\_input.clear()          # self.edit\_permissions\_combo.clear()          # Search for a user by ID and populate the edit fields          user\_id = self.search\_id\_input.text()          print(user\_id)          query = "SELECT \* FROM users WHERE id = ?"          self.cursor.execute(query, (user\_id,))          user = self.cursor.fetchall()          if user:              self.user\_table.setRowCount(len(user))              for row, result in enumerate(user):                  for col, value in enumerate(result):                      item = QTableWidgetItem(str(value))                      item.setFlags(Qt.ItemIsEnabled)                      self.user\_table.setItem(row, col, item)          else:              # No search criteria provided              # Display a message or handle it as needed              query = "SELECT \* FROM users"              self.cursor.execute(query)              user = self.cursor.fetchall()              self.user\_table.setRowCount(len(user))              for row, result in enumerate(user):                  for col, value in enumerate(result):                      item = QTableWidgetItem(str(value))                      item.setFlags(Qt.ItemIsEnabled)                      self.user\_table.setItem(row, col, item)      def delete\_user(self):          # Delete user(s) by ID(s)          input\_ids = self.search\_id\_input.text()          if not input\_ids:              QMessageBox.warning(self, "Пустой ввод", "Пожалуйста, введите ID(-ы) пользователя(-ей) для удаления.")              return          # Remove extra spaces before splitting IDs          input\_ids = input\_ids.replace(" ", "")          user\_ids = input\_ids.split(',')          query = "DELETE FROM users WHERE id IN ({})".format(','.join(['?'] \* len(user\_ids)))          self.cursor.execute(query, user\_ids)          self.conn.commit()          # Reload user data          self.load\_user\_data()      def save\_edit\_user(self):          # Save edited user data to the database          user\_id = self.edit\_id\_input.text()          full\_name = self.edit\_full\_name\_input.text()          username = self.edit\_username\_input.text()          password = self.edit\_password\_input.text()          permissions = self.edit\_permissions\_combo.currentText()          # Проверяем, что все поля ввода заполнены          if not full\_name or not username or not password:              QMessageBox.warning(self, "Пустые поля", "Пожалуйста, заполните все поля для сохранения изменений.")              return            query = """          UPDATE users          SET              full\_name = CASE WHEN ? != '' THEN ? ELSE full\_name END,              username = CASE WHEN ? != '' THEN ? ELSE username END,              password = CASE WHEN ? != '' THEN ? ELSE password END,              permissions = CASE WHEN ? != '' THEN ? ELSE permissions END          WHERE id=?          """            self.cursor.execute(query, (full\_name, full\_name,                                      username, username,                                      password, password,                                      permissions, permissions,                                      user\_id))          self.conn.commit()          # Clear edit fields          self.edit\_id\_input.clear()          self.edit\_full\_name\_input.clear()          self.edit\_username\_input.clear()          self.edit\_password\_input.clear()          # Reload user data          self.load\_user\_data()  # Load user data again without clearing permissions combo  if \_\_name\_\_ == '\_\_main\_\_':      app = QApplication(sys.argv)      username = "YourUsername"  # Replace with the actual username      user\_management\_window = UserManagementWindow(username)      user\_management\_window.show()      sys.exit(app.exec\_()) |

**Вывод:**

Выполнив лабораторную работу, я разработал и отладил приложение на языке С++ с графическим пользовательским интерфейсом, использующим основные виджеты Qt.