Романов Вячеслав 931921

1.Постановка задачи

Разработка приложения печати графиков.

Исходные входные  данные

***Исходные данные для печати*** соответствуют некоторому типу, который определятся пользователем. Данные определенного типа могут отображаться конкретным графиком, который ориентирован на этот тип данных.

Примеры данных.

1. Данные характеризуются парой **[значение, дата]**, хранятся в БД SQLite(архив с файлами прилагается).

2. Данные представлены JSON файлом. Формат данных [**значение , дата].**

**Дано:** предложен начальный вариант архитектуры ПО, в которую требуется внести изменения с целью снижения связности архитектуры. Используется принцип внедрения зависимости. Реализация внедрения зависимости с помощью IOC контейнера.

При разработке архитектуры учесть

1. Возможность добавления новых графиков (графики отличаются видом и данными
2. Изменение визуального стиля графиков (цветной, черно белый)

**Общие требования к GUI**

1. Загружаем данные, путем выбора нужного файла. Данные в ПО не отображаем, отображаем только график, построенный относительно считанных данных.
2. При печати в pdf выбираем место сохранения графика.

**Использование предложенной реализации IOC контейнера  на с++**

Необходимо разобраться в предложенной реализации IOC контейнера.

Код сопроводить соответствующими объяснениями.

Рассмотреть необходимые темы, используемые при реализации IOC контейнера

2.Предполагаемое решение

Для реализации поставленной задачи будут использоваться IOC контейнер. Он используется для внедрения принципа инверсии управления(IOC) – это важный принцип объектно-ориентированного программирования, используемый для уменьшения связей в компьютерных программах. Также он входит в пятёрку важнейших принципов SOLID. IOC уменьшает зависимость между компонентами системы. Его идея заключается в том, чтобы сделать программу зависимой только от некоторого интерфейса. То есть программа не будет зависеть от реализации интерфейса, что позволит редактировать (добавлять или удалять) их не зависимо от самой программы. IOC контейнер служит инструментом для получения объектов реализаций интерфейса программы, которые были записаны в него ранее. В нём создается пара между интерфейсом и выбранной его реализацией. Таким образом программа может получать объекты реализаций из контейнера, зная только интерфейс. То есть зависеть от этих реализаций она не будет-ей необходима только информация об интерфейсе.

В нашем случае идея инверсии управления применяется к классу Chart. Ему необходимо использовать реализации интерфейсов I\_Print (вывод информации на диаграмму) и I\_Reader (считывание из файла). Он получает эти реализации от контейнера, таким образом их изменения не будет влиять на Chart. Помещаются эти реализации в контейнер в методах классов MainWindow и ChartWidget.

MVC – это принцип, при котором программа делится на 3 независимых друг от друга объекта (Модель, Вид, Контроллер). При таком подходе увеличивается степень использования повторного кода, также это упрощает дальнейшие модификации программного кода.

Модель – это посредник между источником данных и программой. Она содержит данные о моделируемом объекте и позволяет программе их использовать. При этом за всё, что касается дальнейшего представления этих данных в нужной форме, она не отвечает.

Вид(представление) – отвечает за представление данных, полученных из модели, в нужной форме. Для одной и той-же модели можно построить несколько различных представлений (например: файловую систему можно представить в виде таблицы, или в виде дерева).

Контроллер – это пользовательский интерфейс, который обеспечивает связь пользователя с программой. То есть, обеспечивает ввод данных пользователя и использует модель и представления для выполнения необходимой задачи.

Модель в нашем случае QFileSystemModel. Она позволяет получать информацию о файловой структуре системы. Представлением является QTableView, которое отображает содержимое некоторой папки из файловой системы. Функции контроллера содержит в себе класс MainWindow. В нём пользователь может определять корневую папку для таблицы.

Для начала я реализовал абстрактный класс I\_Print, и два его наследника Print\_Bar и Print\_Pie, которые нужны для вывода графиков на диаграмму, также в них предусмотрена чёрно-белая палитра.  
Также был написан абстрактный класс I\_Reader, от которого наследуется также два класса: Reader\_SQL\_lite, Reader\_JSON. Соответственно они нужны для считывания информации из баз данных, и если у нас файл .sqlite, то будет использоваться Reader\_SQL\_lite, а если .json то Reader\_JSON.  
Для вывода самих диаграмм и считывания из файлов разного формата был написан класс Chart, который работает на основе IOC контейнера. За счёт IOC контейнера мы сможем дописать ещё один (или несколько) класс(ов) реализующий(их) другой тип данных никак не взаимодействуя с основным интерфейсом.

Был написан класс ChartWidget, который отвечает за работоспособность окна, то есть в нём находятся объекты кнопок (принт, черно-белый график, выбор диаграммы). Также в нём находится метод, который возвращает путь для сохранения pdf, метод реализующий печати pdf файла в указанное место, метод обновления данных при смене базы данных.

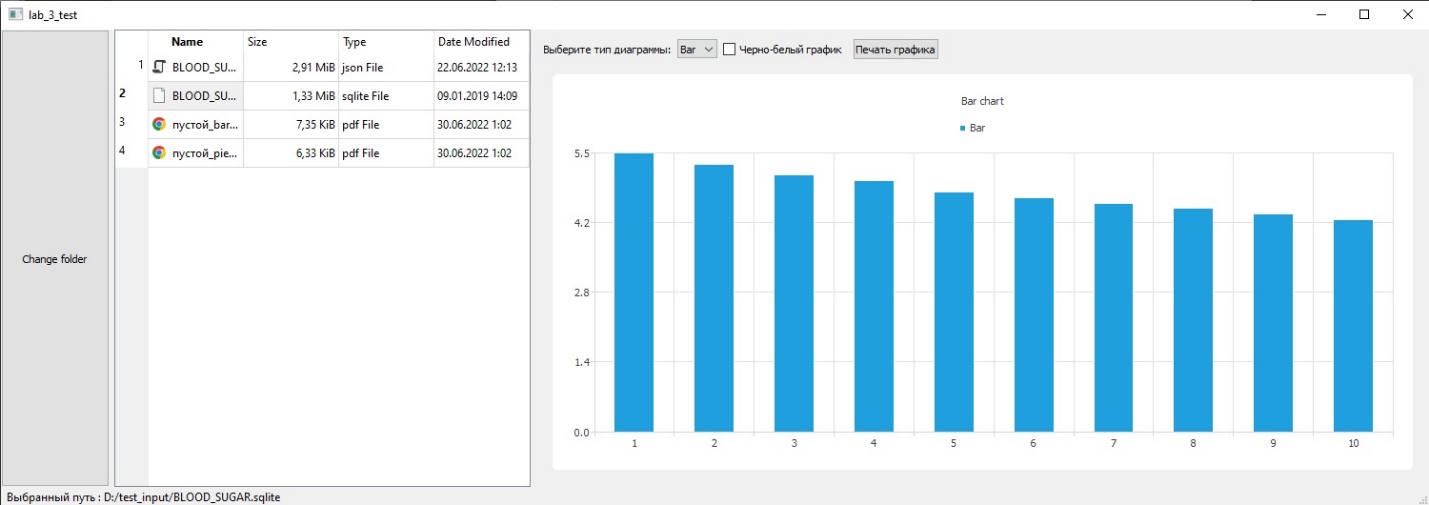
Класс MainWindow – это главное окно. Его я разделил, и добавил таблицу с файлами QTableView, а также место под сами диаграммы (класс ChartWidget). Написан метод вызова диалогового окна, через которое мы выбираем путь до файла (openFileDialogWindow()).

3. Код программы:

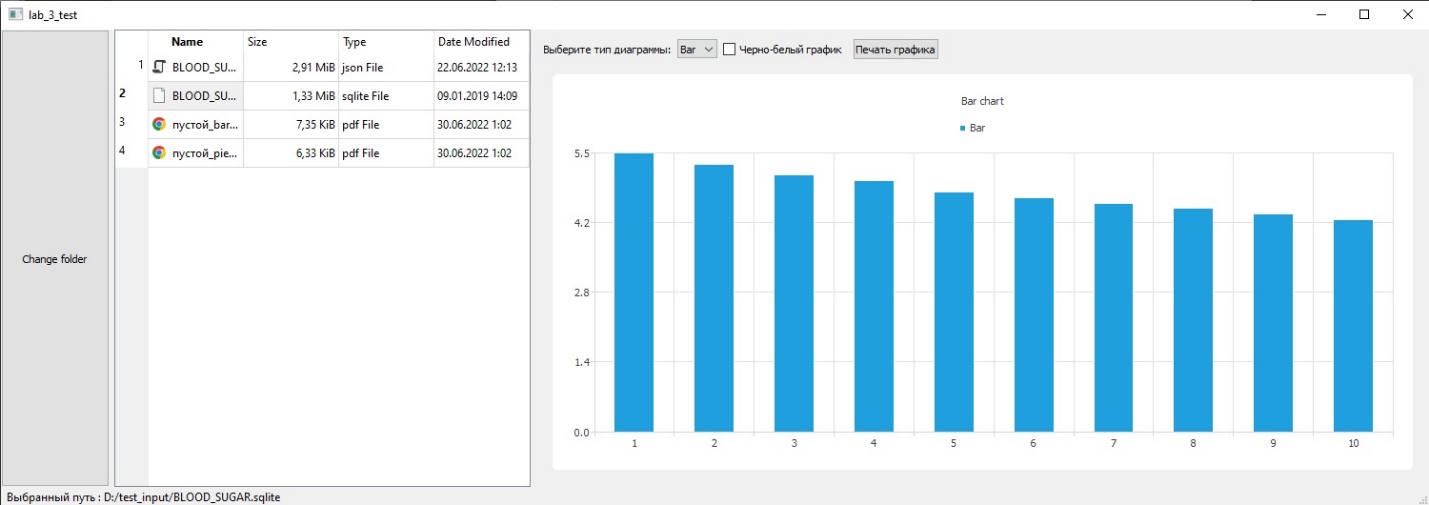
<https://github.com/Dontemon/lab3>

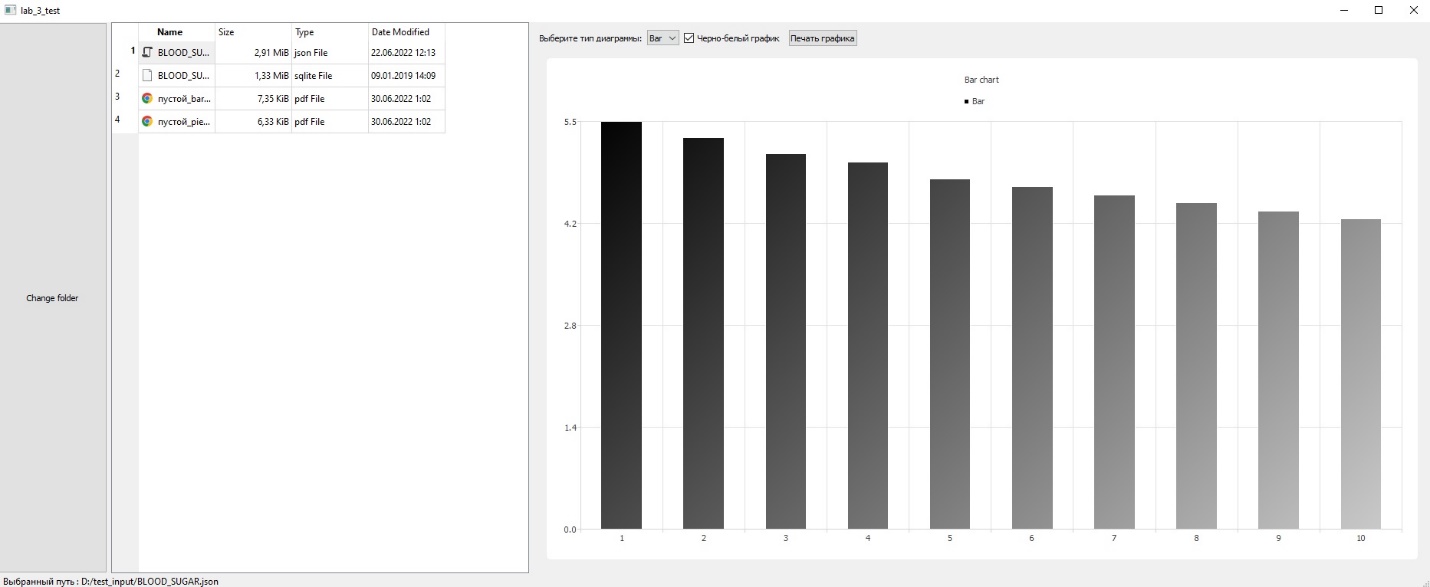
4.Руководство пользования

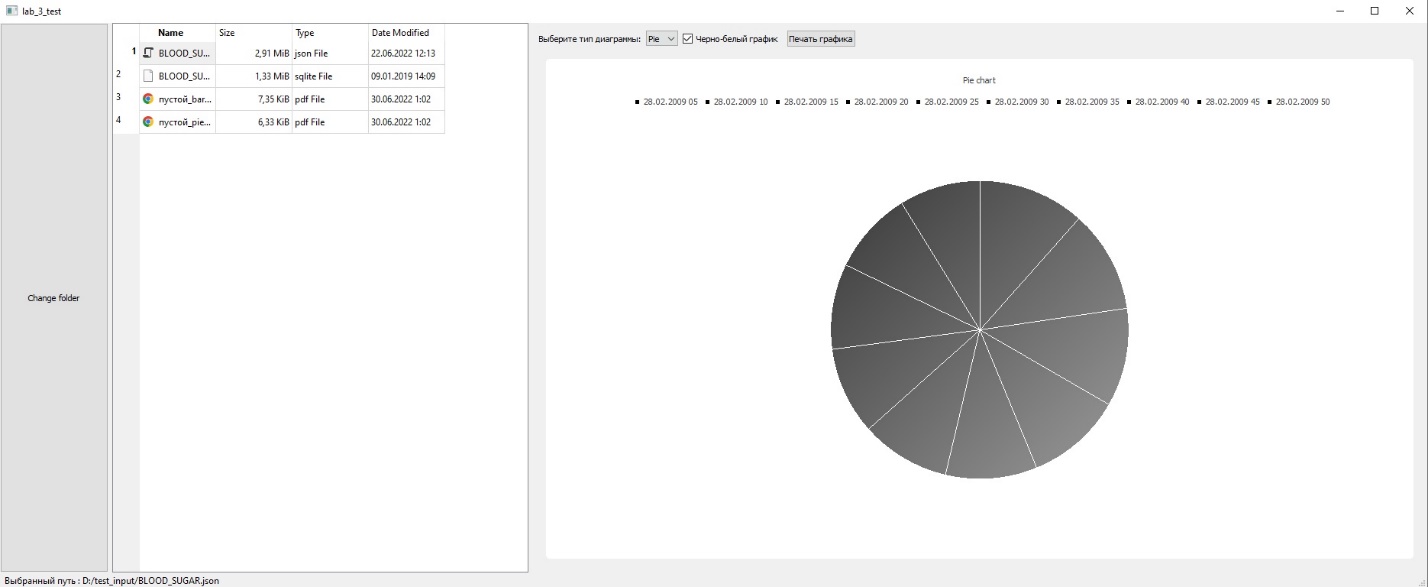
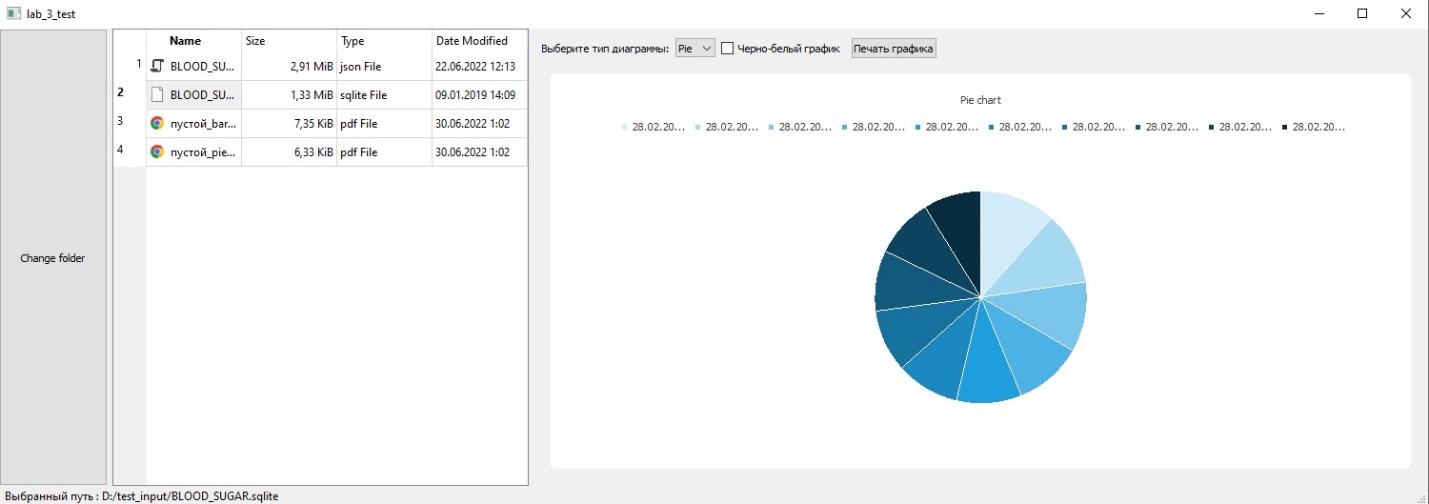
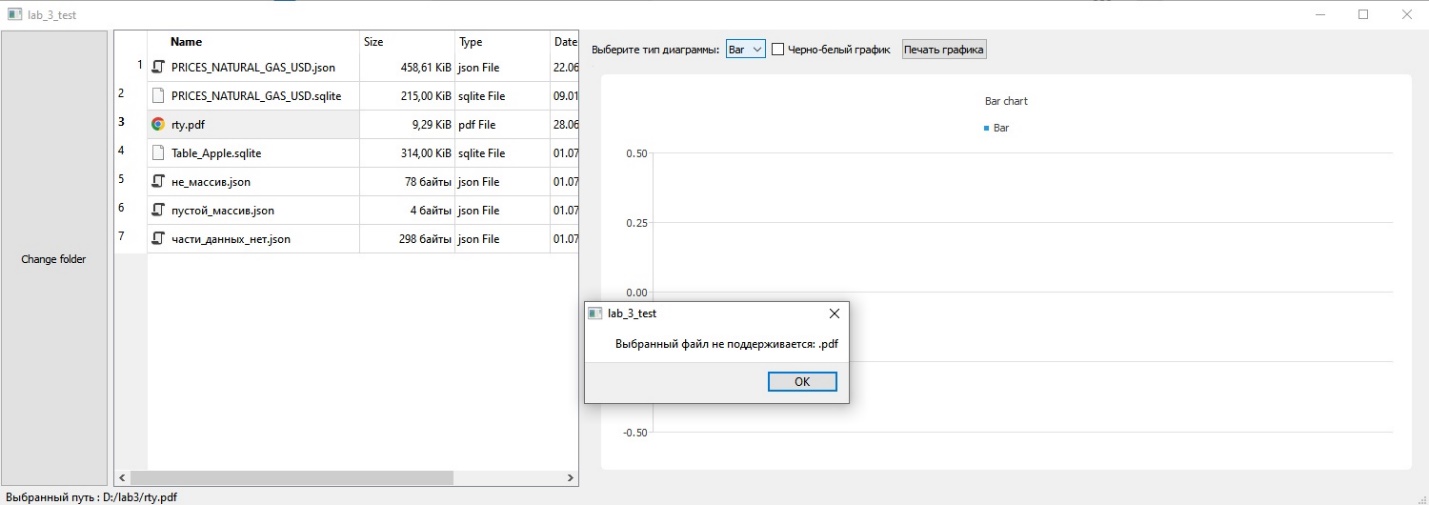
При запуске программы открывается окно, нажимая на большую кнопку Change folder можно выбрать нужную папку, где хранятся нужные базы данных. В самой папке хранятся файлы формата .json и .sqlite(если их в папке нет, то папка показывается как пустая), соответственно, если открыта нужная папка, надо кликнуть на нужную базу данных, чтобы вывелась диаграмма по выбранной базе данных.

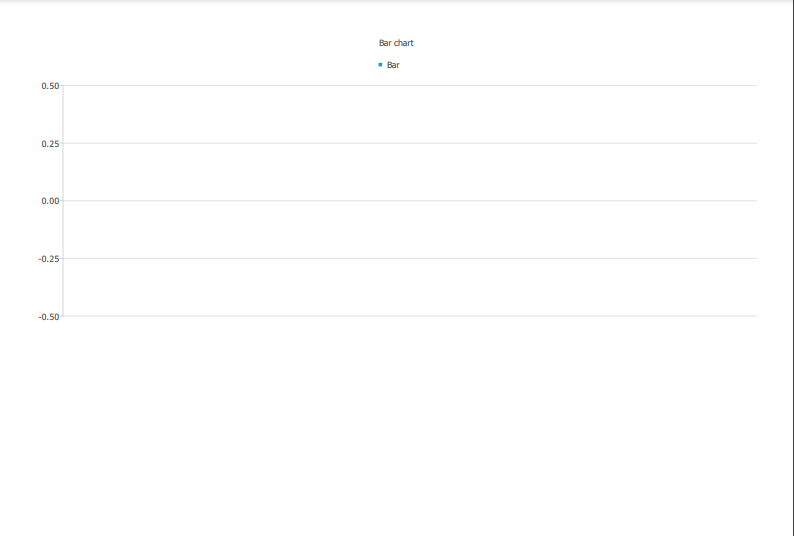
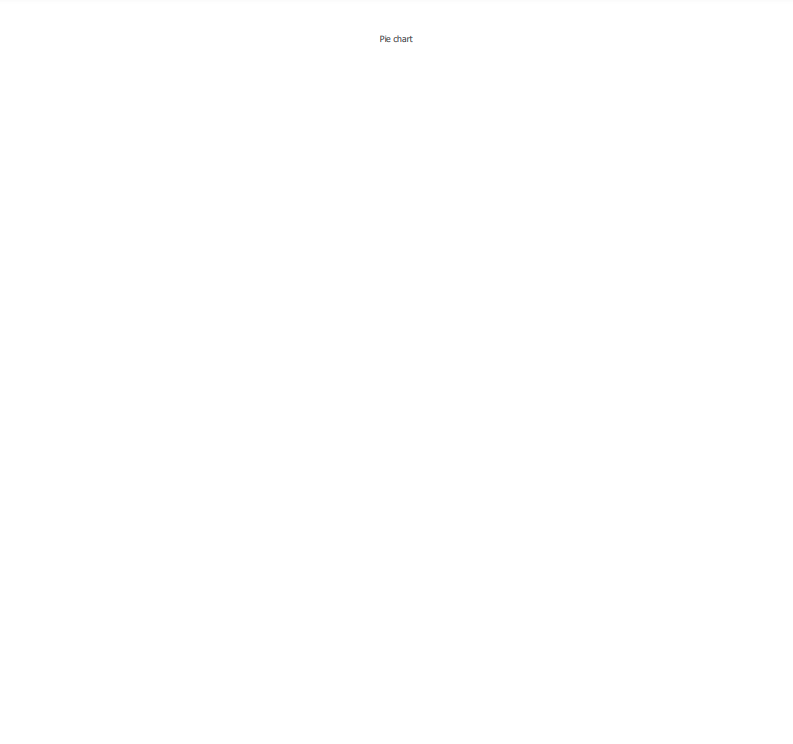
Выбрав нужный файл, в правой части окна появится диаграмма, которую можно будет поменять при необходимости. Сверху над диаграммой расположены 3 кнопки, одна из них переключает тип диаграммы(вертикальная или круговая). Также можно сделать диаграмму чёрно-белой, поставив галочку в соответствующем пункте. И есть кнопка ПЕЧАТЬ ГРАФИКА, которая позволяет выбрать место сохранения и название файла формата .pdf для выведенной диаграммы.

5.Тестирование

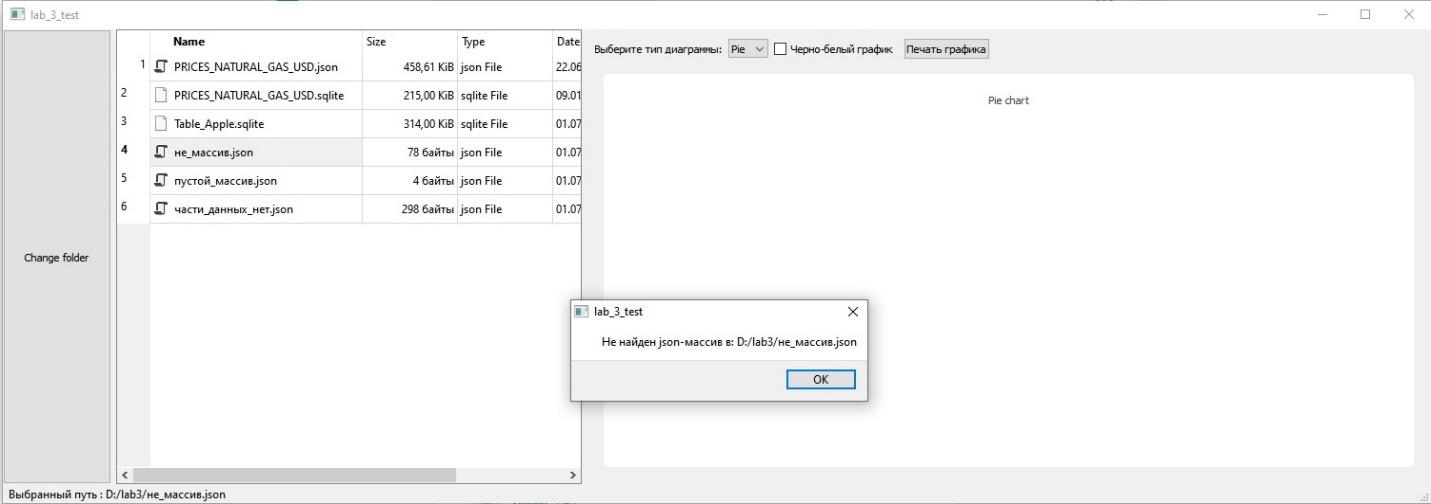
При выборе корректного файла .json или .sqlite формата будет выведено такое окно(вывод ограничен первыми десятью данными из базы)

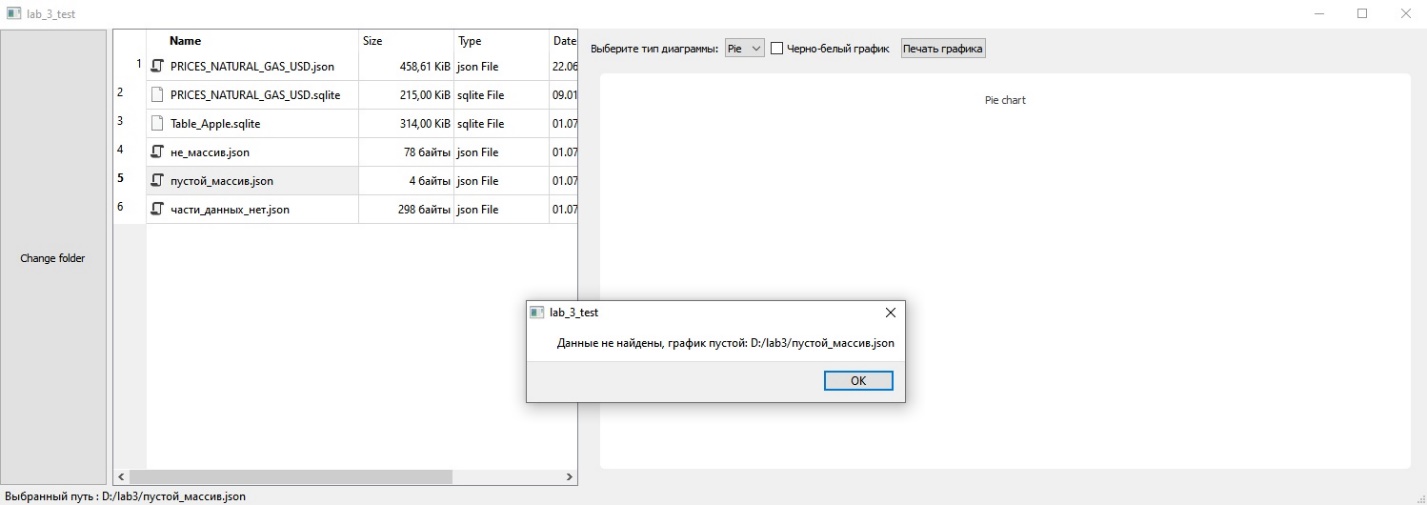
Соответственно, если поставить галочку, чтобы график стал чёрно-белый(также я поменял тут тип файла, чтобы не показывать одно и тоже дважды), то программа выведет такую диаграмму.(левая верхняя часть диаграммы всегда будет наиболее тёмной, а правая нижняя наиболее светлой, так как при написании кода был использован QLinearGradient)

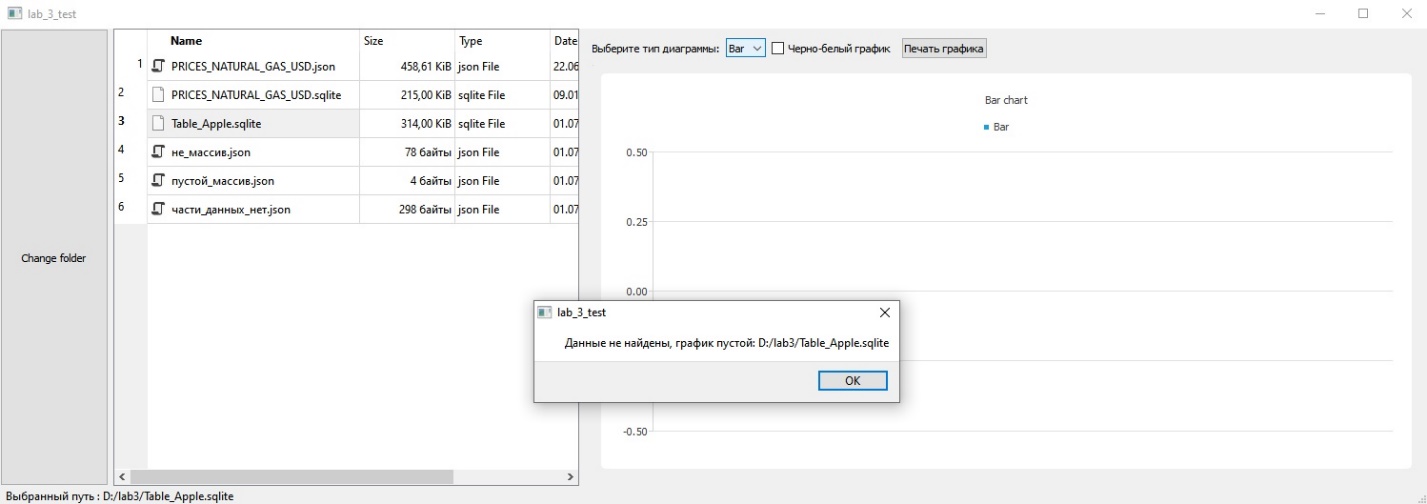
Соответственно, если поменять тип диаграммы на Pie, то правая часть окна изменится и теперь там будет круговая диаграмма  
и также другого формата и цветная диаграмма  
При этом, если выбрать неверный файл, например, файл формата .pdf, то программа выдаст окно с сообщением, что такой формат не поддерживается.

Также если в самом начала, после запуска программы, когда ни одна база данных не была выбрана, нажать на кнопку Печать графика и выбрать имя и место сохранения PDF, то программа просто сделает пдф с разметкой, которая есть с самого начала, но без диаграммы(так как собственно она не была выбрана).  


Если выбрать .json файл, но при этом он не будет сформирован как массив, то программа выдаст окно с ошибкой и местоположением файла, который пытались открыть.



Если база данных пустая, то программа также выдаст ошибку и местоположение этого файла. 



Если утеряна только часть данных, то программа выведет окно с сообщением, что часть данных не найдена, но выведет имеющиеся данные на диаграмму. 