Злобин Глеб, гр. 931920

***1. Постановка задачи***

Разработать приложение для печати графиков. Исходные данные для печати должны соответствовать некоторому типу, который определяется пользователем. Данные определенного типа могут отображаться конкретным графиком, который ориентирован на этот тип данных. Данные могут быть представлены парой [значение, дата] и храниться в БД SQLite, либо jSON-файлом с форматом [значение, дата].

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.

Необходимо разобраться в предложенной реализации IOC контейнера.

Код сопроводить соответствующими объяснениями.

Рассмотреть необходимые темы, используемые при реализации IOC контейнера

***2. Предлагаемое решение***

Во-первых стоит решить, как мы будем считывать данные, которые будем отображать в графике. Я реализовал оба предложенных в условии варианта - файл .json и файлы .sqlite, находящиеся в папке Input, которая находится в папке с проектом (путь к которой потом зададим, определив переменную PRJ\_PATH в .pro-файле). Для считывания этих данных создан класс-интерфейс ChartFileData (в файле chartfiledata.h) с единственной функцией - getData (она, конечно, виртуальная), и он имеет двух наследников: классы ChartFileDataJson и ChartFileDataSqlite, для считывания данных из обоих типов файлов. На данном этапе были использованы переменные типов QString (содержащие пути к файлам), DataTable (которые будут «принимать» данные из файлов), DataList (таблицы данных, на основании которых будем строить график), QSqlDatabase и QJsonDocument («базы данных», куда они будут считываться), QSqlQuery (для использования языка SQL при считывании из него данных), QJsonObject (сюда изначально записывается JSON-объект для чтения, полученный объект преобразуем в массив функцией toArray()), QJsonArray (сюда записываем преобразованный массив данных, состоящий из пар «ключ-значение»). Переменная Datalist должна принимать два аргумента, а нам нужен только один (название данного), поэтому считывание данных производится в две переменные: в одну (типа QString) записывается название данных, в другую (типа QPointF) записываем рандомное значение, которое не будет использоваться при построении графика. Всё это записываем в переменную типа DataTable (для этого создаём функцию CreateData), а количество данных под каждым названием определяется переменной в цикле при считывании.

Для построения графиков в первую очередь нужно было установить библиотеку QtCharts (с которой было очень много проблем, т.к. установка Qt и его компонентов заблокирована в России, и библиотека работает не на всех версиях). Затем мы стандартным образом определим класс для виджета (ThemeWidget), в котором всё будет отображаться, через наследование имеющегося среди библиотек Qt класса QWidget. В нём располагаем все необходимые элементы, используя компоновщики и разделители Splitter. Данный класс определён в файле themewidget.h, кроме него там определены вышеупомянутый метод CreateData (вызывается при соответствующем сигнале из функции on\_selectionChangedSlot), метод ReColor для изменения цветности графика (вызывается в начале построения графика (изначально он цветной) и при срабатывании слотового сигнала в функции on\_color\_chart\_slot; метод countChart для установки количества ячеек графика (вызывается только один раз в начале работы - мы ограничиваем количество ячеек, чтобы их не было слишком много и график был разборчивым). Последние 3 функции можно рассматривать как конструкторы. Также в этом файле содержатся функции из файлов moodle для построения графиков - createBarChart и createPieChart, слегка изменённые с учётом того, что мы ограничиваемся лишь 12-ю секциями в графике, чтобы он был разборчивым. Также из файлов moodle были внесены функции CreateThemeBox, CreateAnimationBox, CreateLegendBox, но они не пригодились.

Также имеется файл mainWindow.h с одноимённым классом, который задаёт пользовательский интерфейс. В нём задаётся размер окна (функцией resize), заголовок окна, путь к папке с базами данных (homePath), переменная FileModel, в которой сохраняются сами файлы баз данных, и прочие детали интерфейса, которые также упорядочиваются вертикальными и горизонтальными компоновщиками (выбор файла, выбор типа диаграммы, кнопки). Также осуществляется обход содержимого папки, в которой хранятся БД, используя программу из moodle. Там определяется переменная FileInfo, циклично проверяется, является ли текущий файл папкой, если нет, то имя и размер файла выводится на консоль. Также в этом файле устанавливаются соединения слотов и сигналов для различных функций программы (выбор графика, печать, смена цвета и т.д.), с помощью функции connect. При срабатывании этих сигналов срабатывают специальные функции (уже упомянутые ранее) - on\_color\_chart\_slot, on\_select\_dir\_slot, on\_print\_chart\_slot, on\_select\_comboboxOnChangedSlot, on\_selectionChangedSlot. Последняя функция выполняет нужные действия при выборе файла из таблицы (например, проверяет, является ли он .json-файлом или .sqlite-файлом, если нет, то возвращая исключение). Функция on\_select\_comboboxOnChangedSlot устанавливает тип графика в зависимости от сигнала, функция on\_print\_chart\_slot осуществляет печать графика, используя библиотеку QPdfWriter, функция on\_select\_dir\_slot выполняет нужные функции при выборе пользователем другой папки для выбора баз данных; функция on\_color\_chart\_slot меняет цвет графика.

***3. Коды программ***

Коды прилагаются в файлах:

themewidget.h

mainwindow.h

chartfiledata.h

themewidget.cpp

chartfiledata.cpp

mainwindow.cpp

main.cpp

Файлы с базами данных прилагаются в папке Input в папке с проектом. Путь к этой папке заранее задан в .pro-файле.

Печать графика может производиться в любую папку на компьютере.

***4. Инструкция пользователя***

Для работы нужно объявить (в файле main) переменную типа MainWindow и показать её функцией show. Т.о. отобразится окно приложения, в котором и предусмотрено выполнение всех действий.

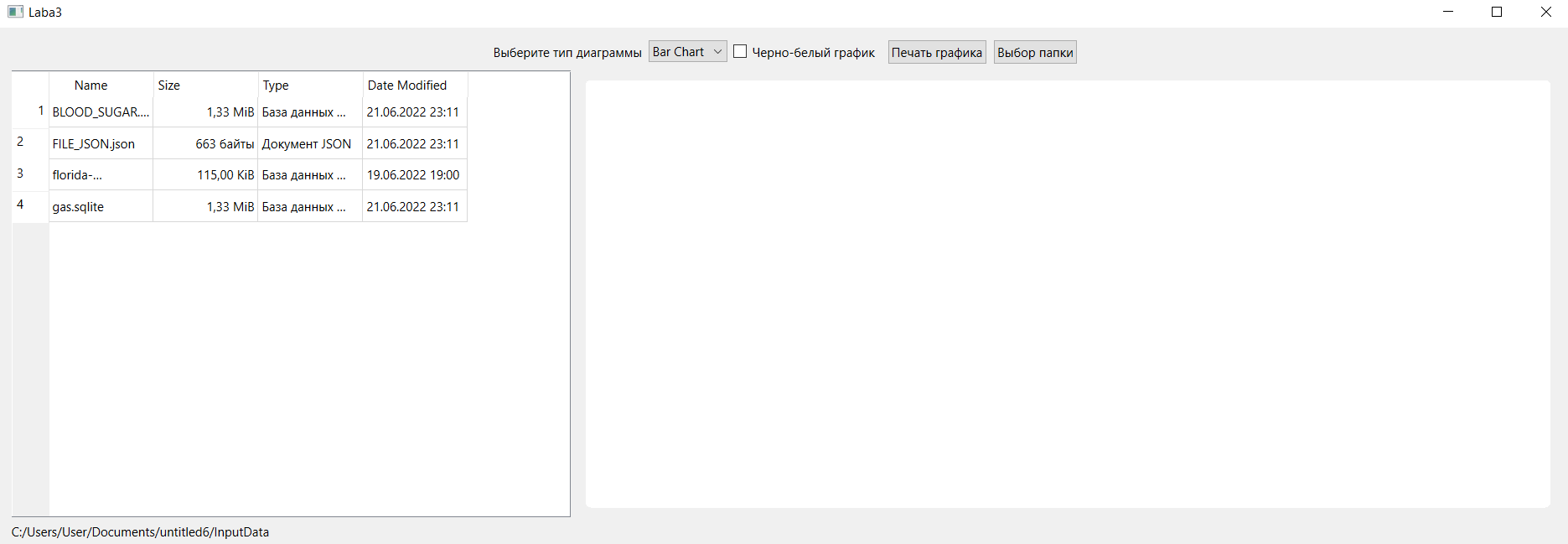
Файлы баз данных пользователь может занести любые в папку Input в папке проекта.

В интерфейсе предусмотрен выбор файла .json/.sqlite из прописанной заранее папки Input, выбор типа диаграммы, цветности графика, кнопка для печати графика, а также кнопка для смены папки с базами данных. При нажатии на файл .json/.sqlite сразу отображается график (состоящий из <=12 секций), который можно распечатать. После успешной печати графика выводится окно MessageBox с соответствующим уведомлением (это реализовано в функции on\_print\_chart\_slot).

***5. Тестирование***

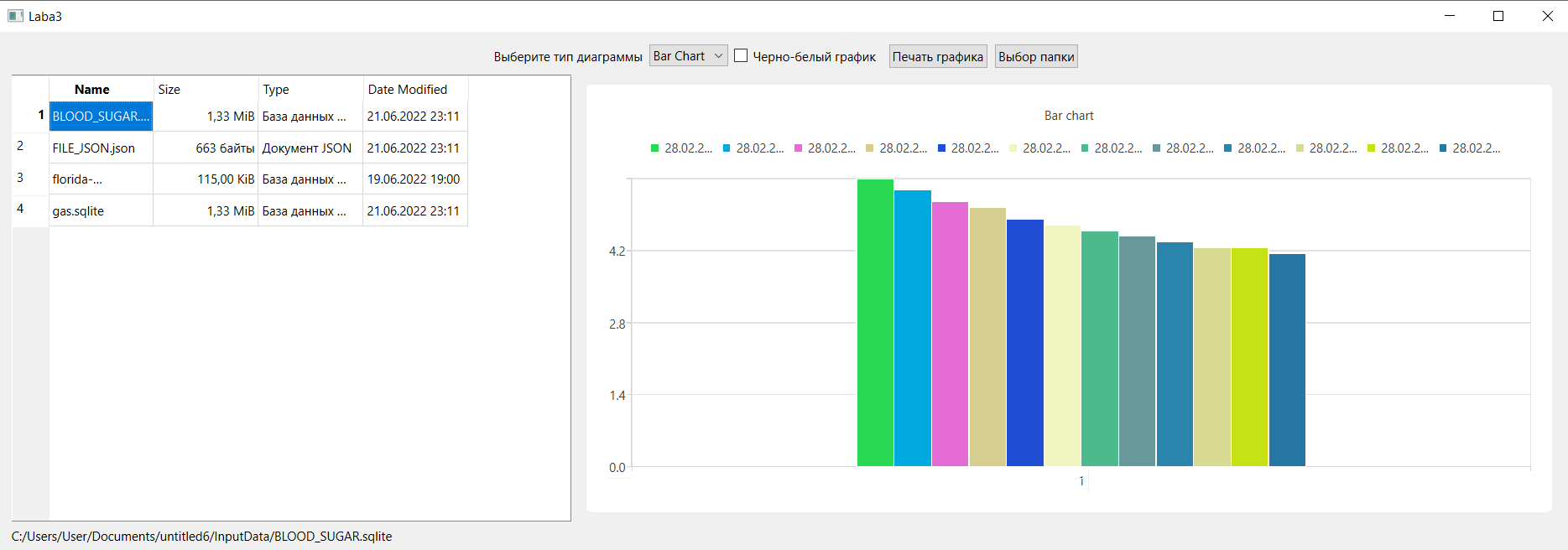
Программа протестирована на нескольких базах данных формата .sqlite и .json (они лежат в папке Input).

Так выглядит окно приложения в начале:

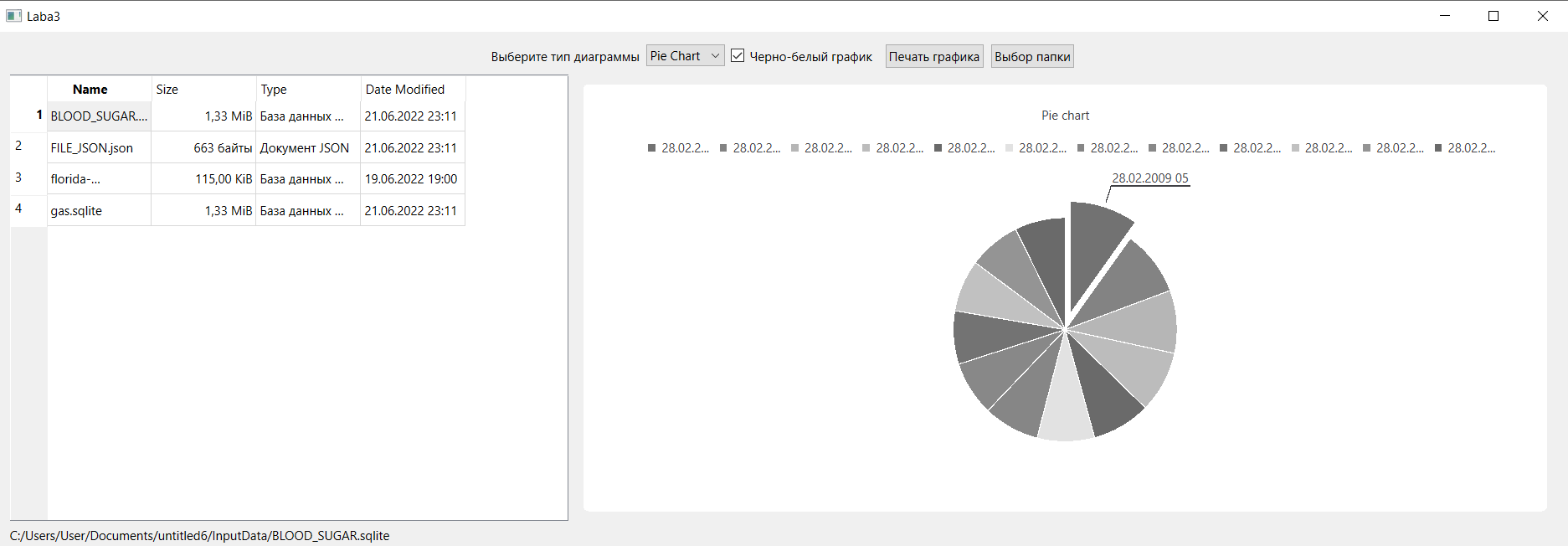


Уже видны 4 базы данных, имеющихся в папке.

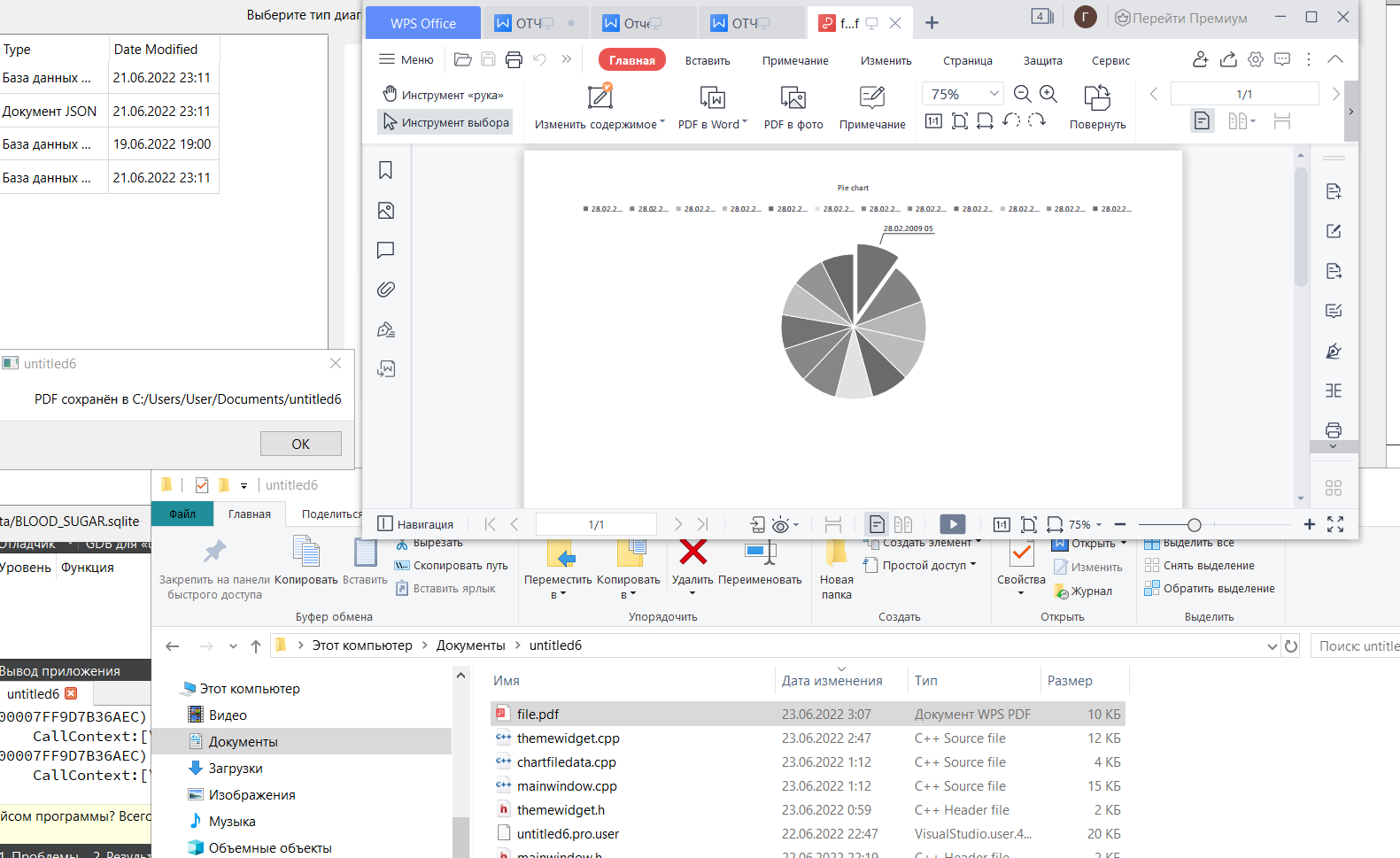
Результат при клике на первую из них:



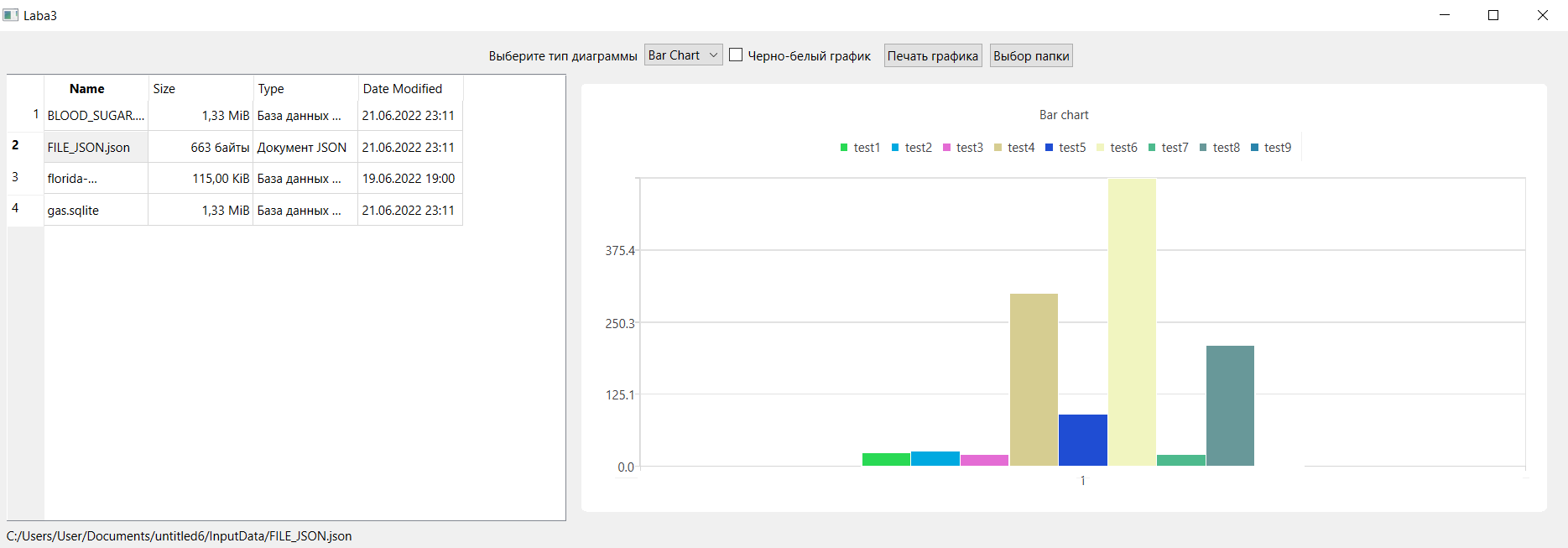
Попробуем сменить одновременно тип и цветность графика:

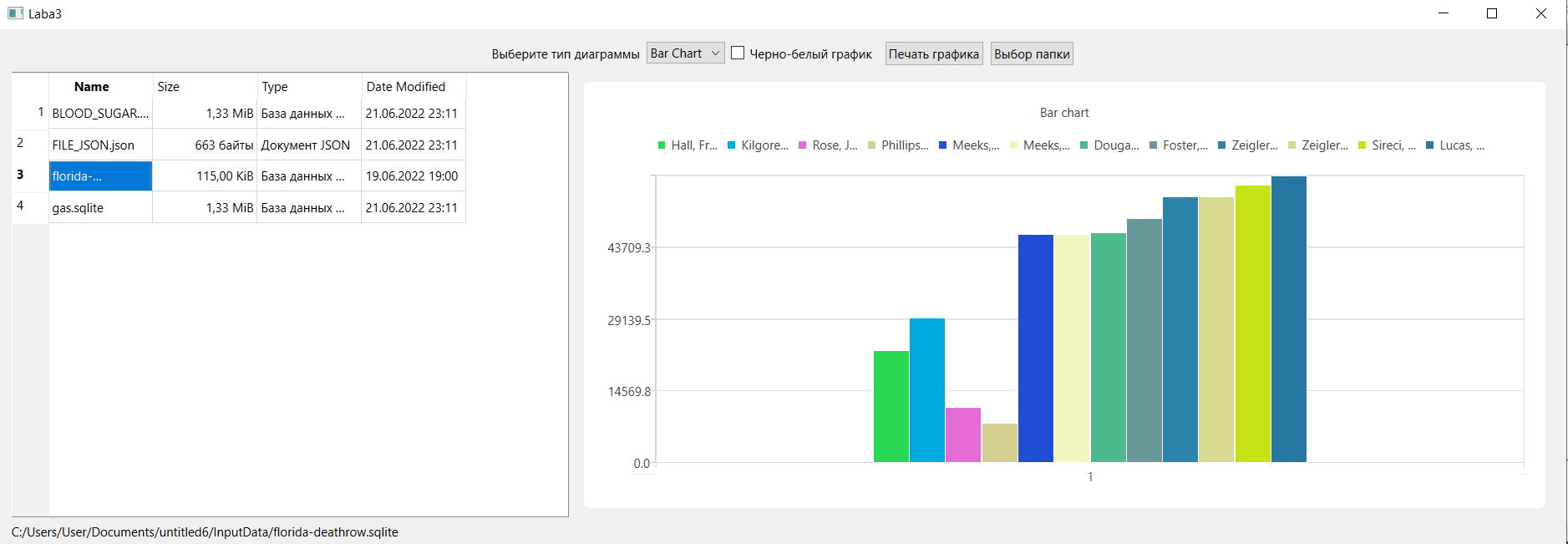


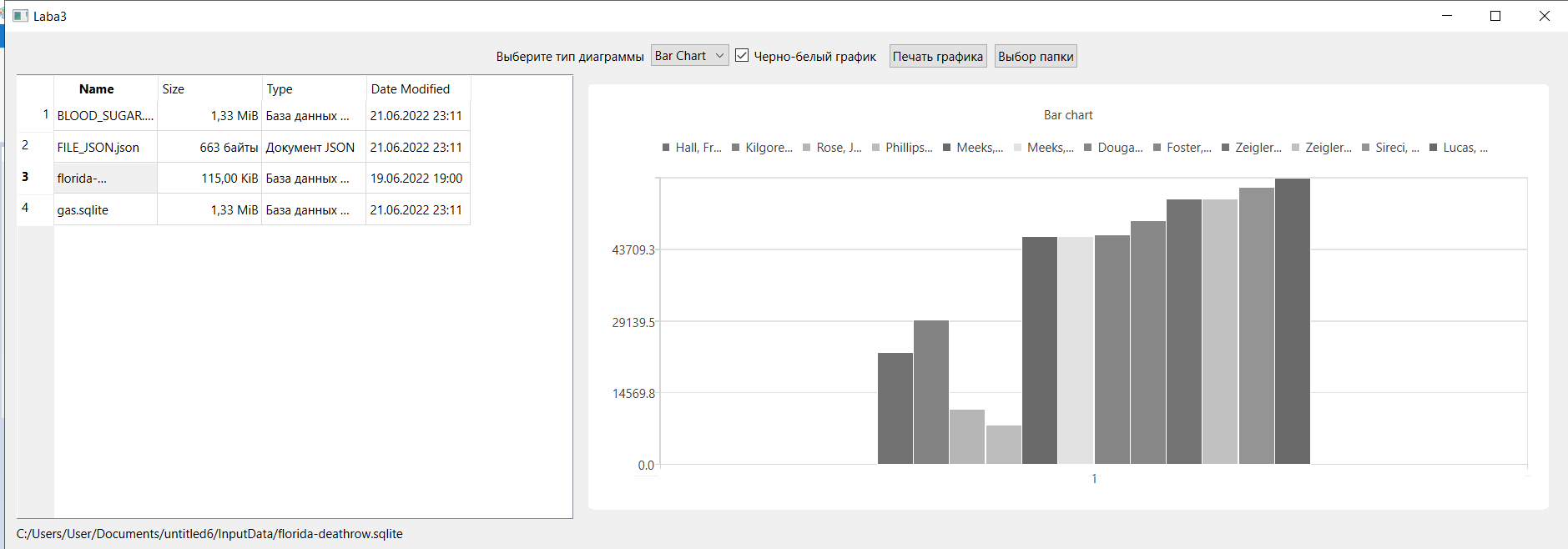
Печать также корректно работает:

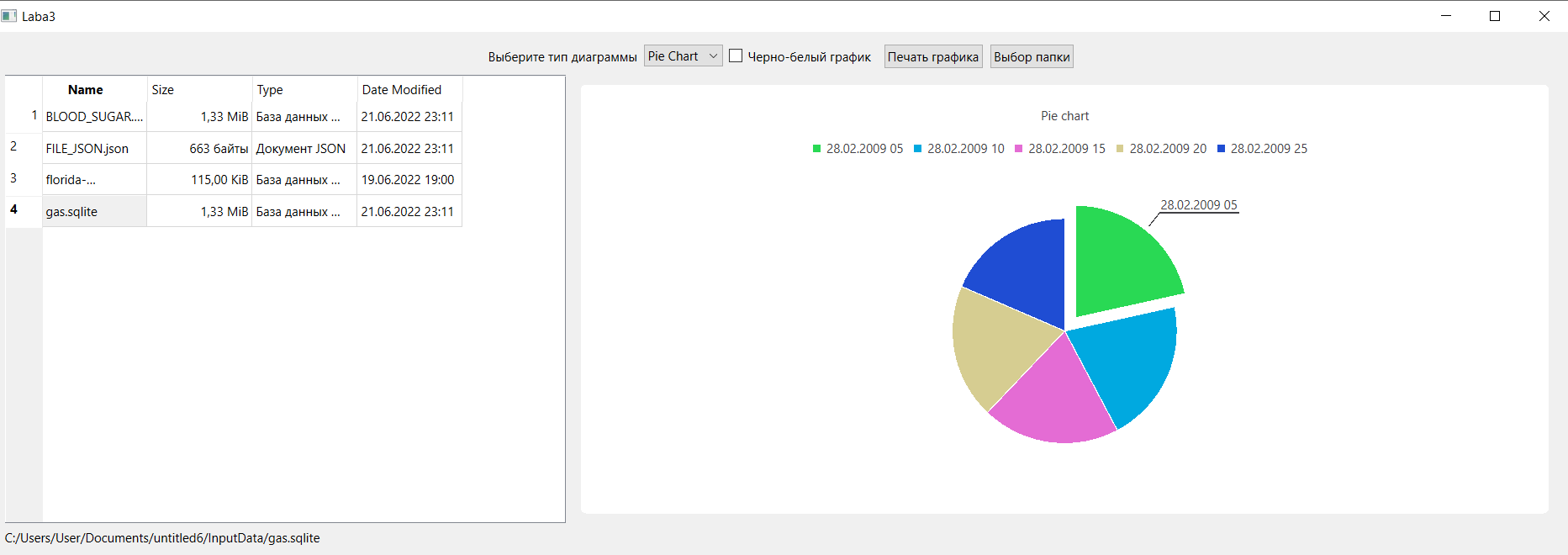


С остальными графиками также всё корректно (некоторые из них содержат <12 элементов):

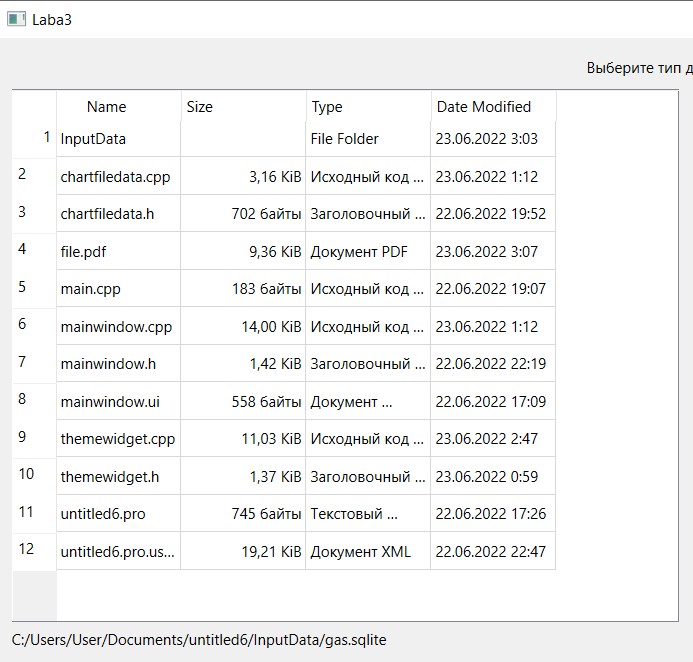
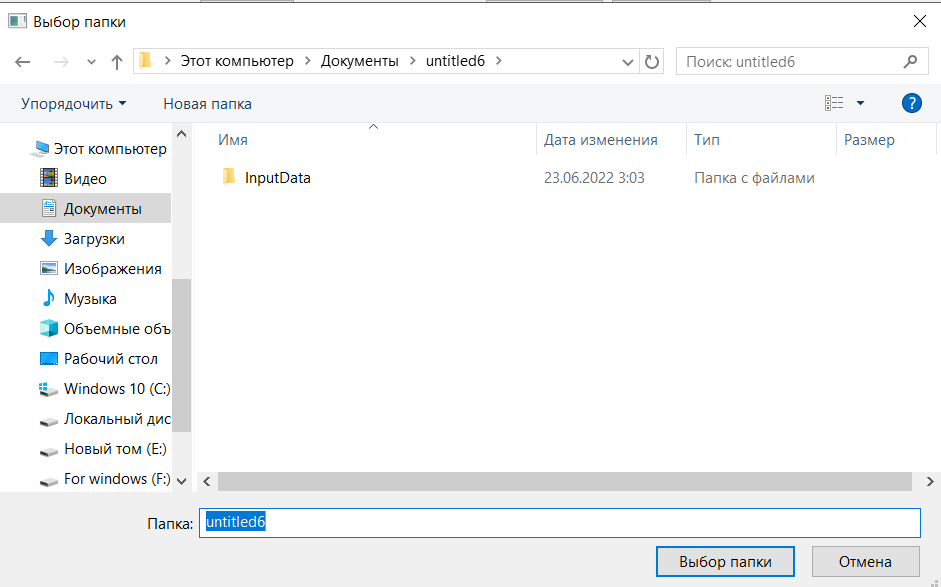








И напоследок продемонстрируем смену папки для выбора БД:



Ни одного бага в ходе выполнения данных действий обнаружено не было.