Бисенбенов Данабек, гр. 931921

***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.

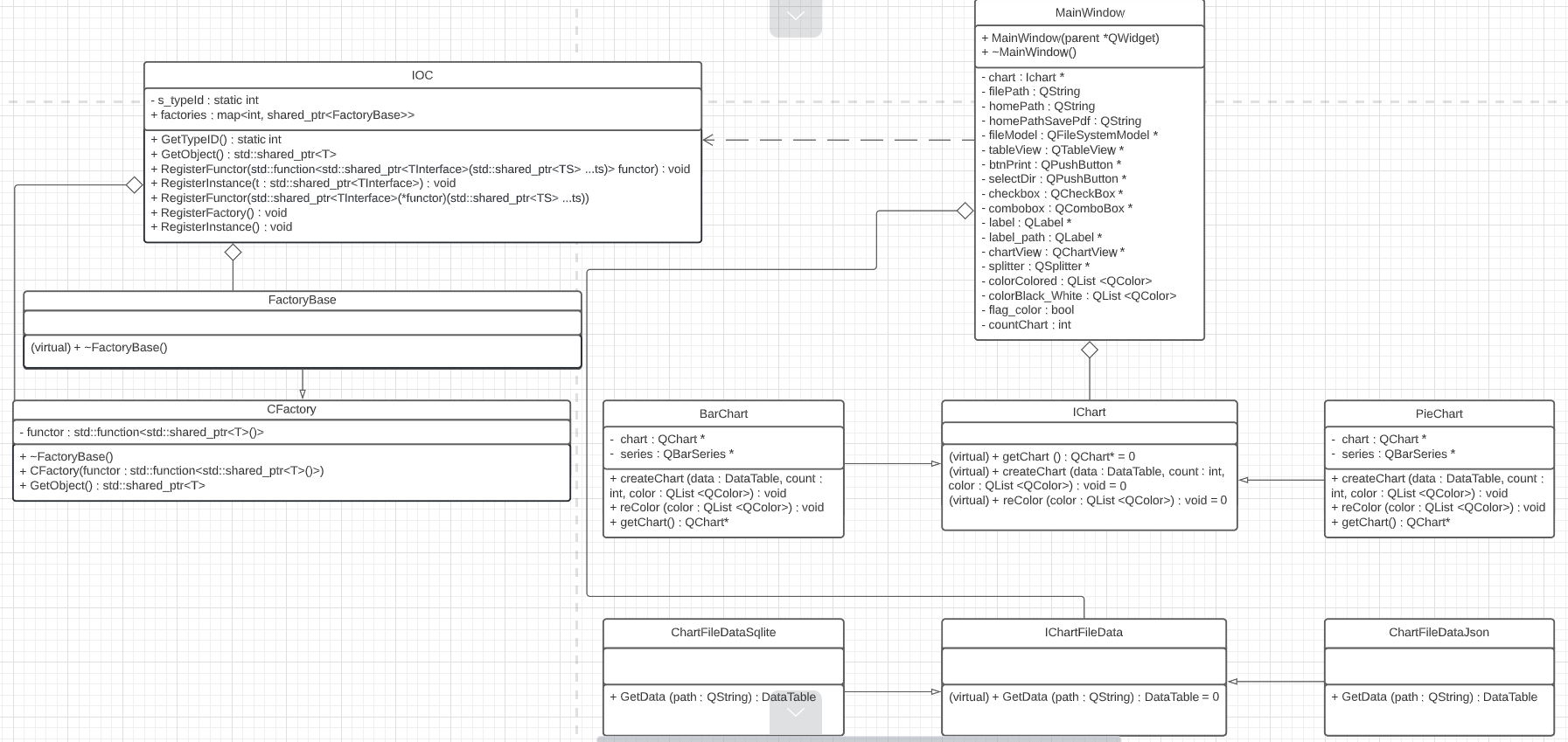
Необходимо разобраться в предложенной реализации IOC контейнера.

Код сопроводить соответствующими объяснениями.

Рассмотреть необходимые темы, используемые при реализации IOC контейнера

***2. Предлагаемое решение***

Архитектура программы:



Созданы 2 класса-интерфейса для задач, предполагающих возможные «расширения»: считывание данных для графиков с файлов различных типов (интерфейс IChartFileData; классы-наследники: ChartFileDataSqlite для считывания данных с .sqlite; ChartFileDataJson для считывания данных с .json); построение графиков разных типов (интерфейс IChart; классы-наследники: BarChart для столбчатой диаграммы; PieChart для круговой диаграммы). Они все агрегированы в класс MainWindow, использующий их информацию для построения и печати графиков, а также в котором задействован метод RegisterInstance (создание экземпляра) IOC-контейнера, определённого в отдельном классе. IOC-контейнер в данном случае не обязателен, но полезен тем, что снижает связности между компонентами: он отвечает за фактическое создание объекта, вызов конструктора и возврат только ссылочного типа, получение экземпляров любых зависимостей перед вызовом конструктора, т.о. решая задачу связности между компонентами системы. В публичной области IOC-контейнера инициализируется переменная, регистрирующая экземпляр для будущего графика. Все операции с графиками (построение/смена цвета/смена типа) производятся в классе MainWindow (в соответствующих функциях), но с использованием функции из IOC-контейнера (для регистрации экземпляра объекта), куда подставляется аргумент типа графика.

Интерфейс для смены системы цветности графиков (цветной/ЧБ) я делать не стал, т.к. двумя вариантами такая система и ограничена - расширений не предполагается. (достаточно завести булевую переменную в классе MainWindow).

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 используется библиотека QtCharts.

В классе MainWindow реализованы функции, выполняющие конкретные задачи. Функцией connect установлены соединения слотов и сигналов для выполнения этих функций (при срабатывании сигналов функции выполняются). Функция on\_color\_chart\_slot меняет цвет графика; функция on\_print\_chart\_slot печатает график, используя библиотеку QPdfWriter; функция on\_select\_dir\_slot делает необходимые действия при выборе пользователем другой папки для выбора баз данных (нужно исправить парочку надписей + внешний вид файловой системы); функция on\_select\_comboboxOnChangedSlot меняет тип графика. Функция on\_reDraw - вспомогательная, введена для того, чтобы не выполнять одни и те же действия по смене цвета дважды (цвет нужно проконтроллировать ещё и при смене типа графика). В данном классе также осуществляется обход содержимого папки, в которой хранятся БД, используя код из moodle. Там определяется переменная FileInfo, циклично проверяется, является ли текущий файл папкой, если нет, то имя и размер файла выводится на консоль. Файловая система представлена на экране в табличном представлении. Также в данном классе реализован интерфейс: задаётся размер окна (функцией resize), заголовок окна, путь к папке с базами данных (homePath), переменная FileModel, в которой сохраняются сами файлы баз данных, горизонтальные/вертикальные компоновщики (регулирующие порядок/расположение имеющихся элементов), и прочие детали интерфейса, которые в совокупности и составляют MainWindow нашей программы.

При этом именно создание графика (т.е. не печать, а прорисовка) производится в классе iChart - в нём есть 2 функции: одна - для смены цветов графика с помощью метода setBrush (она вызывается из функций для смены цветности в классе MainWindow), другая - для создания самого графика. Также при создании приложения предусмотрена плавная анимация вывода графика на экран (с использованием метода setAnimationOptions из библиотеки QCharts).

***3. Коды программ***

Коды прилагаются в файлах:

ioc.h

mainwindow.h

chartfiledata.h

Ichart.h

ichart.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.

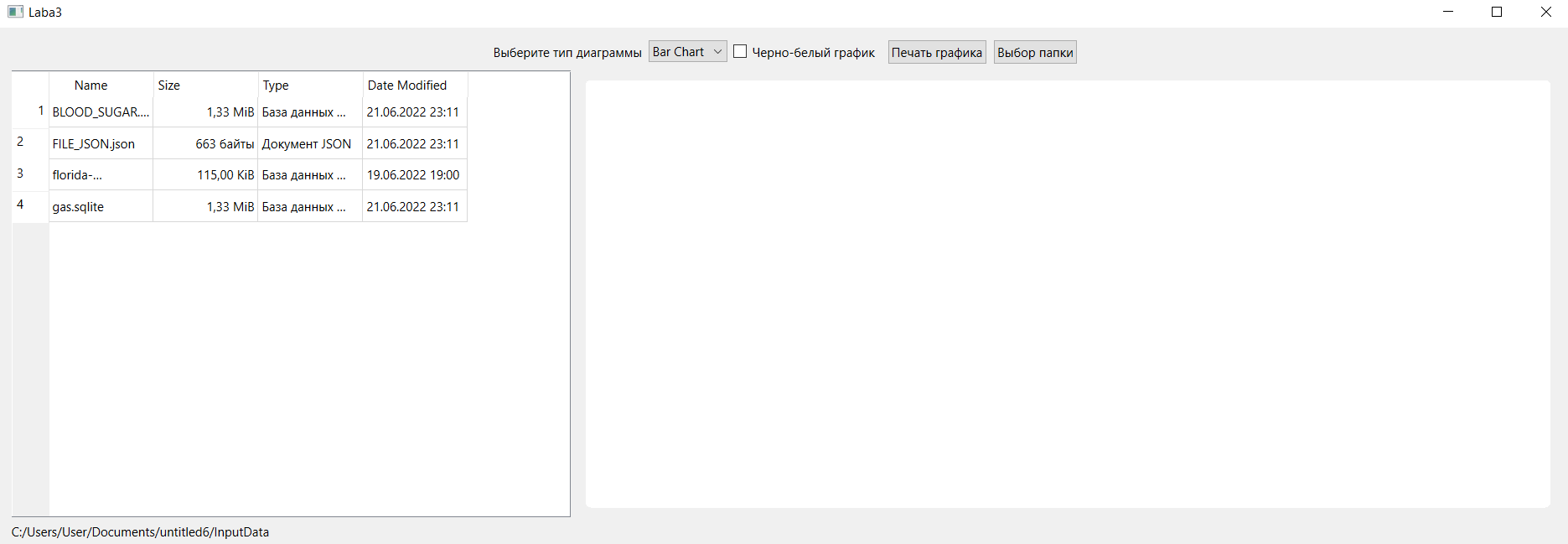
Файлы баз данных пользователь может занести любые в папку Input в папке проекта.

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***

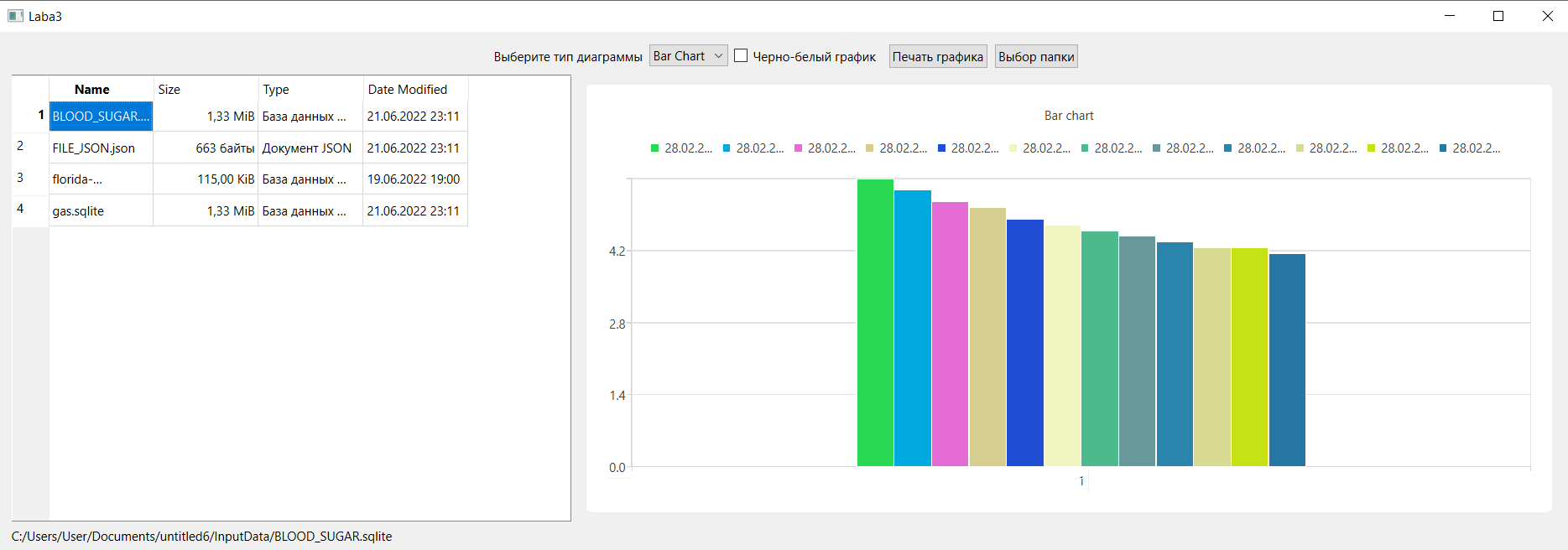
Программа протестирована на нескольких базах данных формата .sqlite и .json (они лежат в папке Input).

Так выглядит окно приложения в начале:

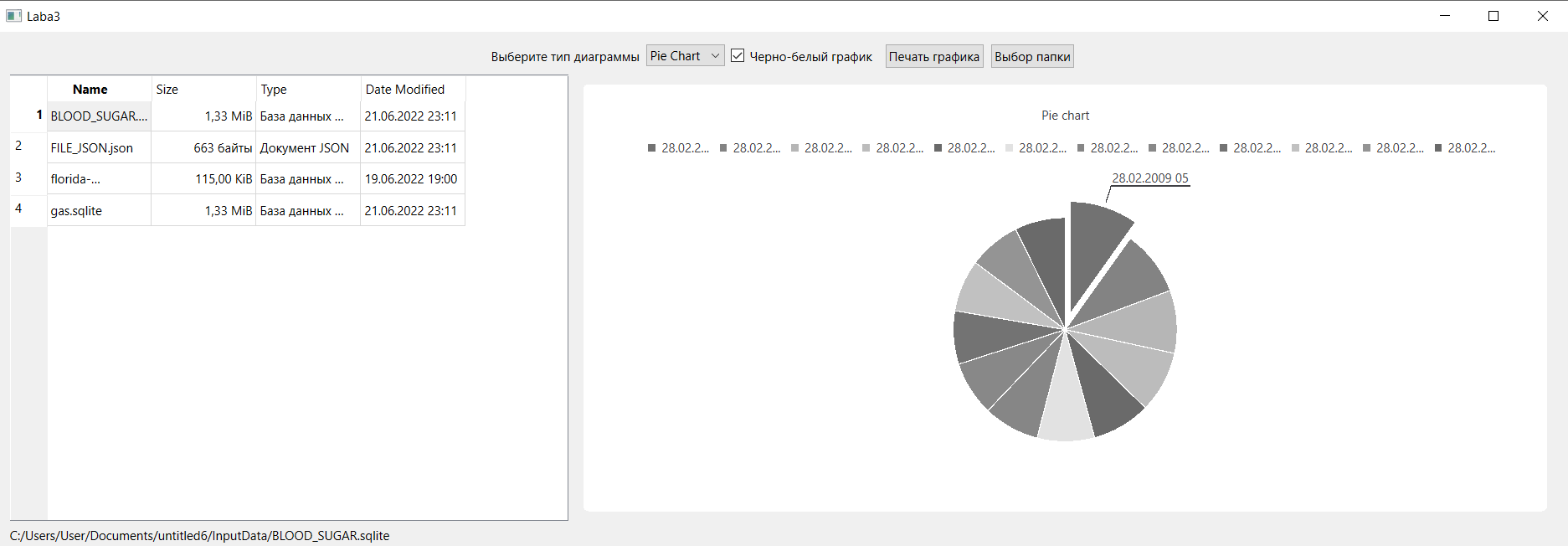


Уже видны 4 базы данных, имеющихся в папке.

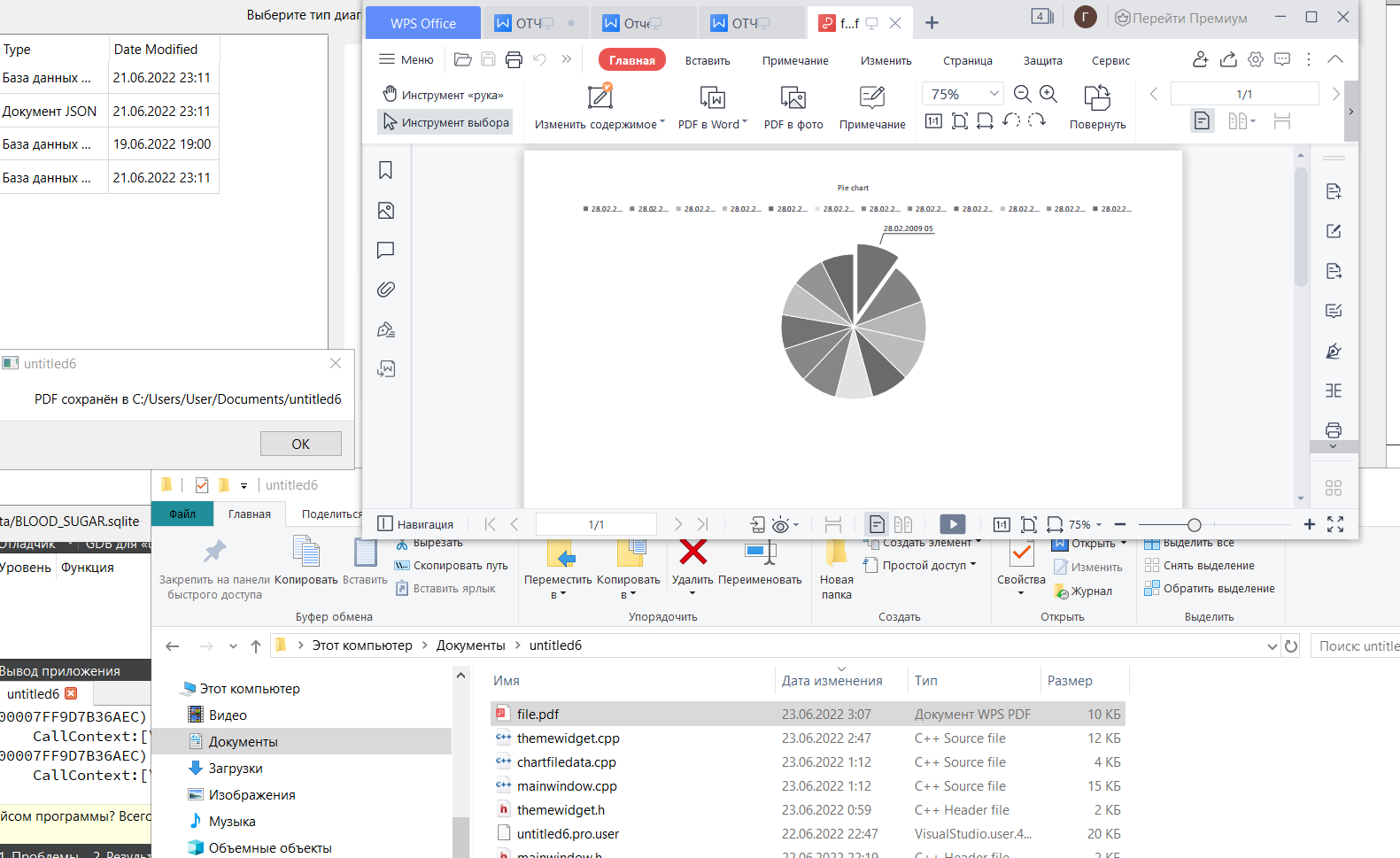
Результат при клике на первую из них:



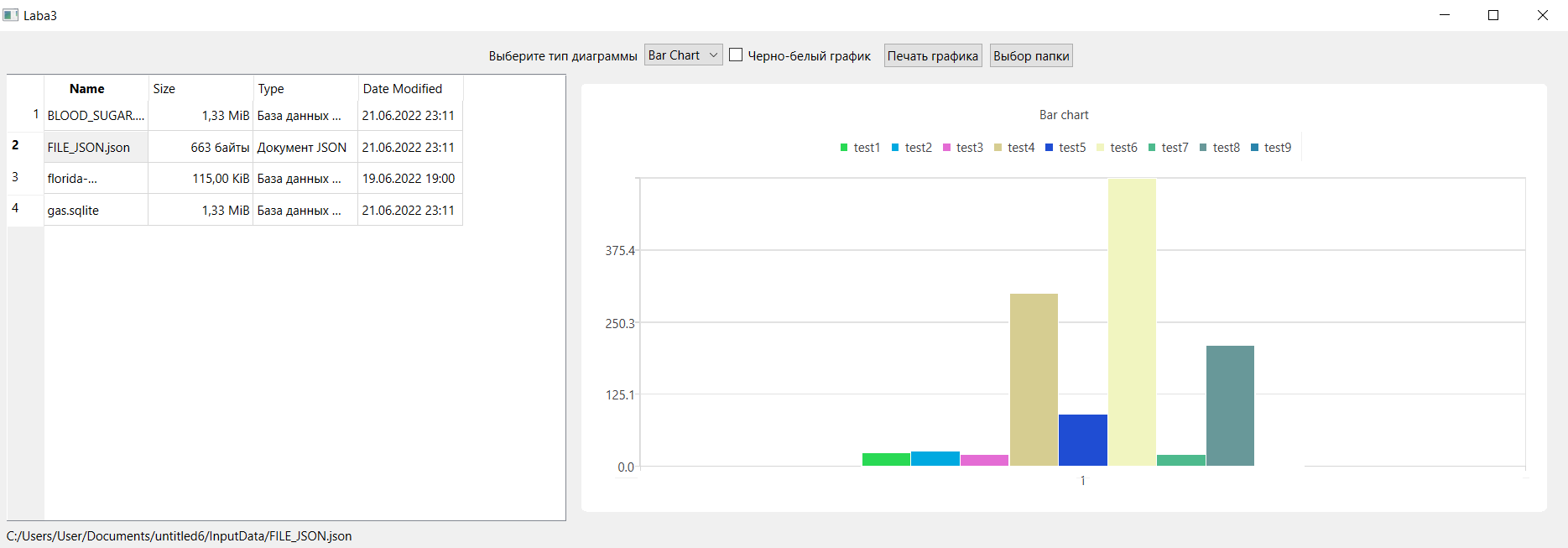
Попробуем сменить одновременно тип и цветность графика:

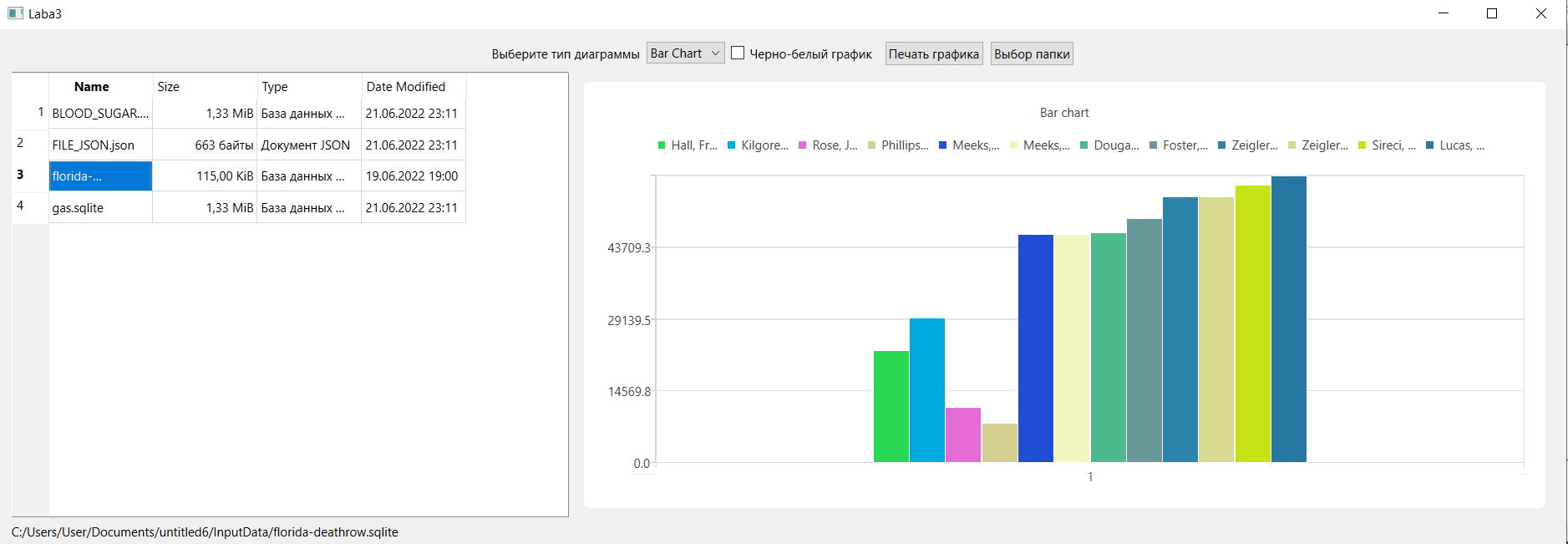


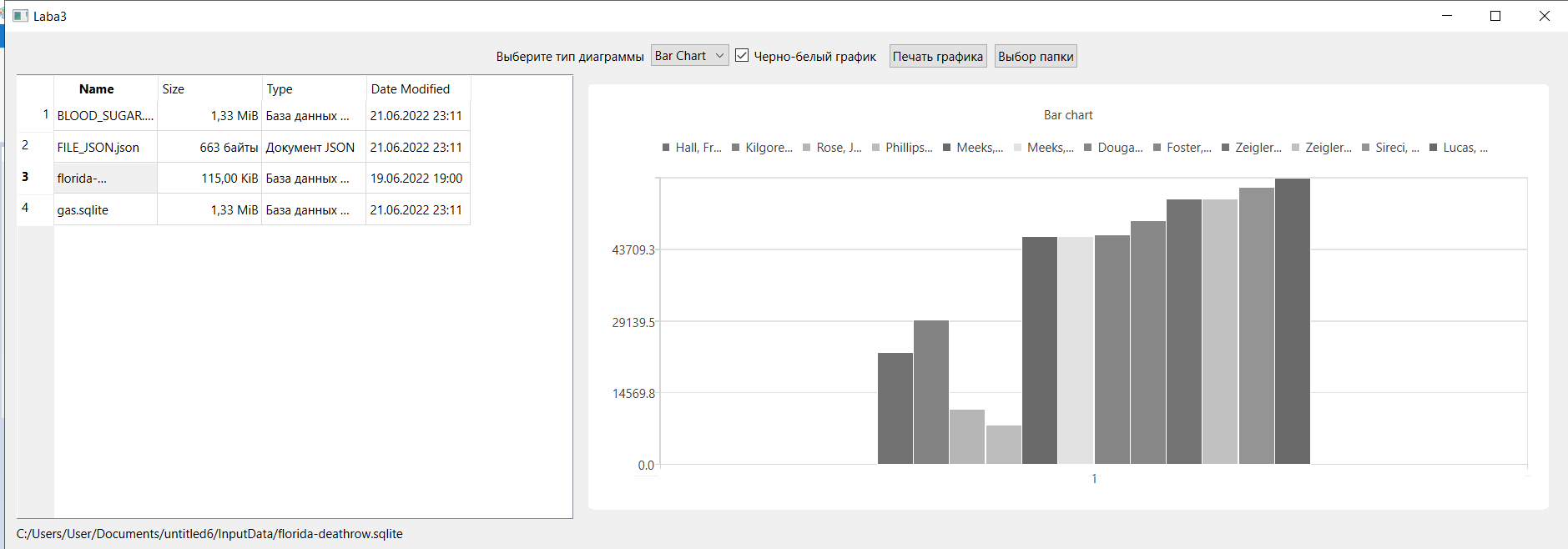
Печать также корректно работает:

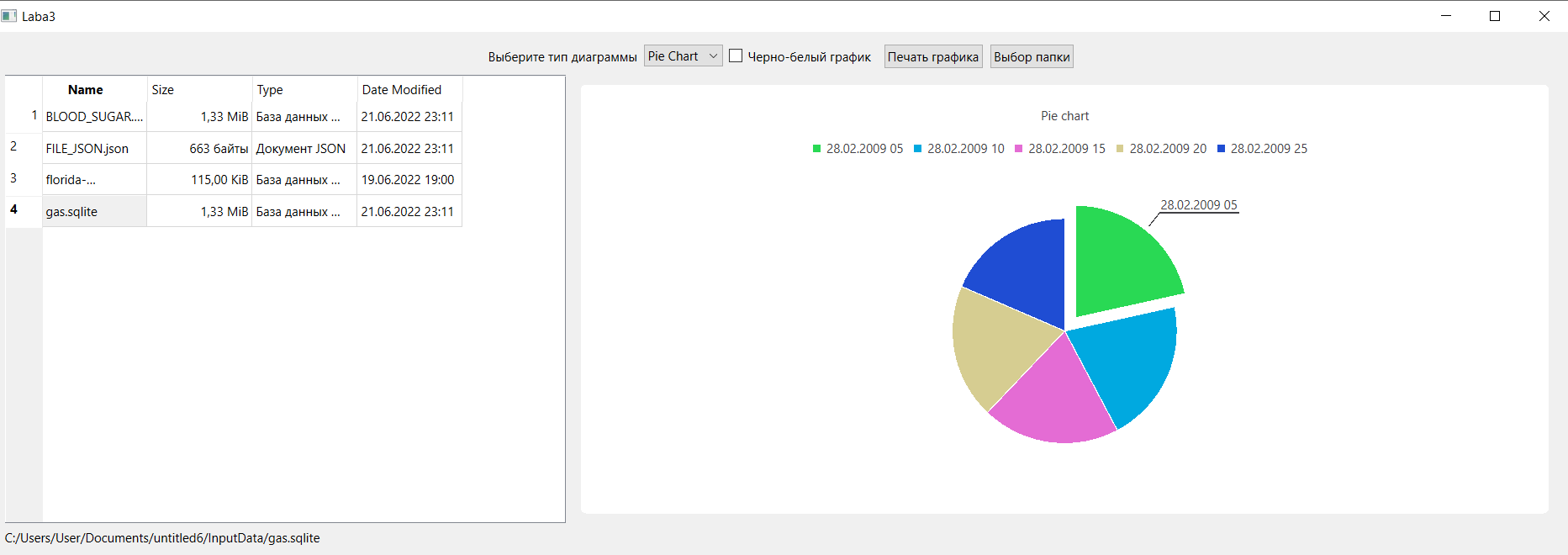


С остальными графиками также всё корректно (некоторые из них содержат <12 элементов):









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