Шарко Полина Алексеевна. Группа 931920.

1. Постановка задачи

Разработка приложения печати графиков.

Исходные входные данные

***Исходные данные для печати*** соответствуют некоторому типу, который определяется 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

Возможность добавления новых графиков (графики отличаются видом и данными)

Изменение визуального стиля графиков (цветной, черно белый).

2) Предлагаемое решение

Как сказано в задании, необходимо разработать приложение печати графиков и внести в предложенный вариант архитектуры изменения с целью снижения связности, используя принцип внедрения зависимости (dependency injection) из принципов SOLID. При внедрении Использовать IOC контейнер (IOC - inversion of control).

Inversion of Control (инверсия управления) — это абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Каждый компонент системы изолирован и не зависит от реализации других компонентов. Dependency injection(DI) - одна из реализаций IOC. IOC и DI позволяют, например, добавлять реализации интерфейса без внесения изменений в класс-интерфейс. Способ реализации контейнера - помещение в него интерфейса и реализаций интерфейса. Тогда инверсия управления будет состоять в том, что класс-интерфейс не зависит напрямую от реализаций, так как получает их из контейнера. Так код будет слабо связным, чего мы и добивались.

В реализации необходимо обеспечить связь частей, отвечающих за построение и представление графиков, с данными таблиц. Разделим программу на части, используя схему разделения данных приложения и управляющей логики MVC.

Суть Model-View-Controller (MVC) заключается в разделении на части модель, представление и контроллер таким образом, что модификация каждого компонента может осуществляться независимо.

Здесь **модель** предоставляет данные и реагирует на команды контроллера, изменяя свое состояние. В предложенной реализации в качестве модели использовалась файловая модель QFileSystemModel. Ей предоставляются данные, соответствующие полученным из файла с использованием группы классов **ChartData**.

**Представление** отвечает за отображение данных модели пользователю, реагируя на изменения модели. Для представления реализуемых в программе данных используем QTableView (представление файловой системы в виде таблицы).

**Контроллер** интерпретирует действия пользователя, оповещая модель о необходимости изменений, реализован классом виджета **MainWindow. Отвечает за демонстрацию таблицы и возможность изменения её корневой папки.**

При реализации интерфейса были использованы:

* **QFileSystemModel** - для определения файловой системы и получения пути к файлу(для отображения файлов)
* **QItemSelectionModel -** представление элементов в виде таблицы файлов
* **QCharts -** для построения диаграмм
* **QVector <DataStorage> -** хранение считанных данных. Отвечает за цвет диаграммы
* **IChartDrawing -** интерфейс для выбора типа графика. При реализации был использован IOC контейнер. В интерфейсе задана виртуальная функция рисования диаграмм, не зависящая от типа диаграммы. Тип определяется в **barChartDrawing** и **pieChartDrawing.** Есть возможность добавления новых типов графиков.
* **QChartView -**виджет для отображения диаграмм
* **QFileDialog -** выбор директории
* **QPdfWriter -** сохранение графика в формате PDF

**Реализованные классы:**

**IOC-контейнер:**

**Вспомогательные классы:**

**FactoryBase - фабрика, дает возможность хранить объект, способ создания которого неизвестен**

Методы:

virtual ~*FactoryBase*() - деструктор

**CFactory : public FactoryBase - шаблонный класс, конкретная фабрика**

Поля:

std::function<std::shared\_ptr<T>()> functor - поле хранит объект, у которого есть известный способ создания

Методы:

~*CFactory*() - деструктор

CFactory(std::function<std::shared\_ptr<T>()> functor):functor(functor) - конструктор по умолчанию. Соответствует конструктору функтора.

std::shared\_ptr<T> GetObject() - функция для получения функтора

**Контейнер:**

**IOCContainer:**

Поля:

static int s\_typeId - переменная, определяющая Id следующего выделенного типа

map<int, shared\_ptr<FactoryBase>> factories - используем map для хранения фабрик

Методы:

template<typename T> - метод, позволяющий обратиться к экземпляру локальной переменной для указанного типа

std::shared\_ptr<T> **GetObject**() - функция для получения функтора

void **RegisterFunctor**(std::function<std::shared\_ptr<TInterface>(std::shared\_ptr<TS> ...ts)> functor) - регистрация функтора

void **RegisterInstance**(std::shared\_ptr<TInterface> t) - регистрация экземпляра объекта

void **RegisterFunctor**(std::shared\_ptr<TInterface>(\*functor)(std::shared\_ptr<TS> ...ts)) - регистрация функтора с указателем на functor

void **RegisterFactory**() - Фабрика, вызывает необходимый конструктор для каждого экземпляра

void **RegisterInstance**() - фабрика, возвращает один экземпляр ообъекта для одного вызова

**Построение графика:**

**Chart:**

**Charts:**

**Класс для упрощения работы с диаграммами. Содержит информацию о данных, по которым нужно строить диаграмму, и о представлении диаграммы.**

Поля:

QChart \*chart\_ - содержит информацию о представлении диаграммы

QVector <DataStorage> data\_ - ключ-данные(дата-время) и значение данных из таблиц

bool isColored\_ - цвет диаграммы

Методы:

**Charts**(): chart\_( new QChart()), isColored\_ (true) - конструктор по умолчанию

QChart\* **getChart**() - геттер получения представления диаграммы

void **updateData**(const QString& filePath) - обновление данных диаграммы

void **reDrawChart**() const - изменение диаграмм

void **changeColor**() - смена цвета

**IOC-контейнер: интерфейс и его реализации для построения диаграммы:**

**IChartDrawing:**

**Класс - интерфейс, содержит чистую виртуальную функцию(класс является базовым, в базовом классе функция определения не имеет) для построения диаграмм по заданным данным.**

virtual void ***drawChart***(QVector <DataStorage> data, bool isColored = true, QChart\* chart\_ = new QChart()) = 0; - функция, строящая новую диаграмму QChart\* chart\_ по данным QVector <DataStorage> data

**barChartDrawing:**

**Реализация интерфейса IChartDrawing. Позволяет строить столбчатые диаграммы.**

virtual void ***drawChart***(QVector <DataStorage> data, bool isColored = true, QChart\* chart\_= new QChart()); - функция, строящая новую столбчатую диаграмму QChart\* chart\_ по данным QVector <DataStorage> data

**pieChartDrawing:**

**Реализация интерфейса IChartDrawing. Позволяет строить круговые диаграммы.**

virtual void ***drawChart***(QVector <DataStorage> data, bool isColored = true, QChart\* chart\_= new QChart()); - функция, строящая новую круговую диаграмму QChart\* chart\_ по данным QVector <DataStorage> data

Data:

**DataStorage:**

**Класс, объекты которого содержат данные из базы данных.**

Поля:

QString key - ключ-данные(дата-время)

double value - значение данных из базы данных

Методы:

**DataStorage** (QString key\_, double value\_) - конструктор по умолчанию

**IOC-контейнер: интерфейс и его реализации для хранения данных:**

**IChartData:**

**Класс - интерфейс, содержит чистую виртуальную функцию(класс является базовым, в базовом классе функция определения не имеет) для получения данных из базы данных.**

virtual QVector <DataStorage> ***getData*** (QString path\_) = 0 - функция для получения данных, сохраняемых в QVector <DataStorage>

**ChartDataSqlite:**

**Реализация интерфейса IChartData. Позволяет сохранять данные из базы данных формата .sqlite.**

QVector <DataStorage> ***getData*** (QString path\_) - функция для получения данных из файла формата .sqlite, сохраняемых в QVector <DataStorage>

**ChartDataJson:**

**Реализация интерфейса IChartData. Позволяет сохранять данные из базы данных формата .json.**

QVector <DataStorage> ***getData*** (QString path\_); - функция для получения данных из файла формата .json, сохраняемых в QVector <DataStorage>

**MainWindow:**

**MainWindow:**

Методы:

Слоты обработки

void **slotSelectionChanged**(const QItemSelection &selected, const QItemSelection &deselected); - выбор файла

void **slotChooseDirectory**(); - выбор папки для отображения всех записей

void **slotSelectionComboboxChanged**(); - выбор типа данных с помощью Combobox (выпадающего списка)

void **slotSelectionColorChanged**(); - выбора цвета: Цветной/ЧБ с помощью Checkbox (Выбор: Да/Нет)

void **slotSaveToPdf**(); - слот обработки сохранения Диаграммы в PDF

**MainWindow**(QWidget \*parent = 0) - конструктор

~***MainWindow***() - деструктор

QFileSystemModel \*fileModel - модель файловой системы

QTableView \*tableView - модель табличного представления файловой системы

QString homePath - путь к файлу

QComboBox\* boxType - выпадающий список с типами диаграмм

QCheckBox \*checkboxColor - чекбокс для смены цвета диаграммы

**Дополнительно реализована анонимная структура для взаимодействия с диаграммой**

**struct{}chartSettings**

Поля:

Charts\* chart - поле с сылкой на диаграмму

QChartView\* chartView - поле с сылкой на представление

3) Коды программ

Файлы находятся в репозитории https://github.com/PSh-2022/Lab3.git

Файлы:

* charts.h – взаимодействие с графиками
* charts.cpp - определения функций взаимодействия с графиками
* data.h – взаимодействие с данными для отображения графика
* data.cpp -определения функций взаимодействие с данными для отображения графика
* ioccontainer.h – IOC контейнер
* mainwindow.h - виджет главного окна
* mainwindow.cpp - определения функций виджета главного окна
* main.cpp - создание виджета главного окна

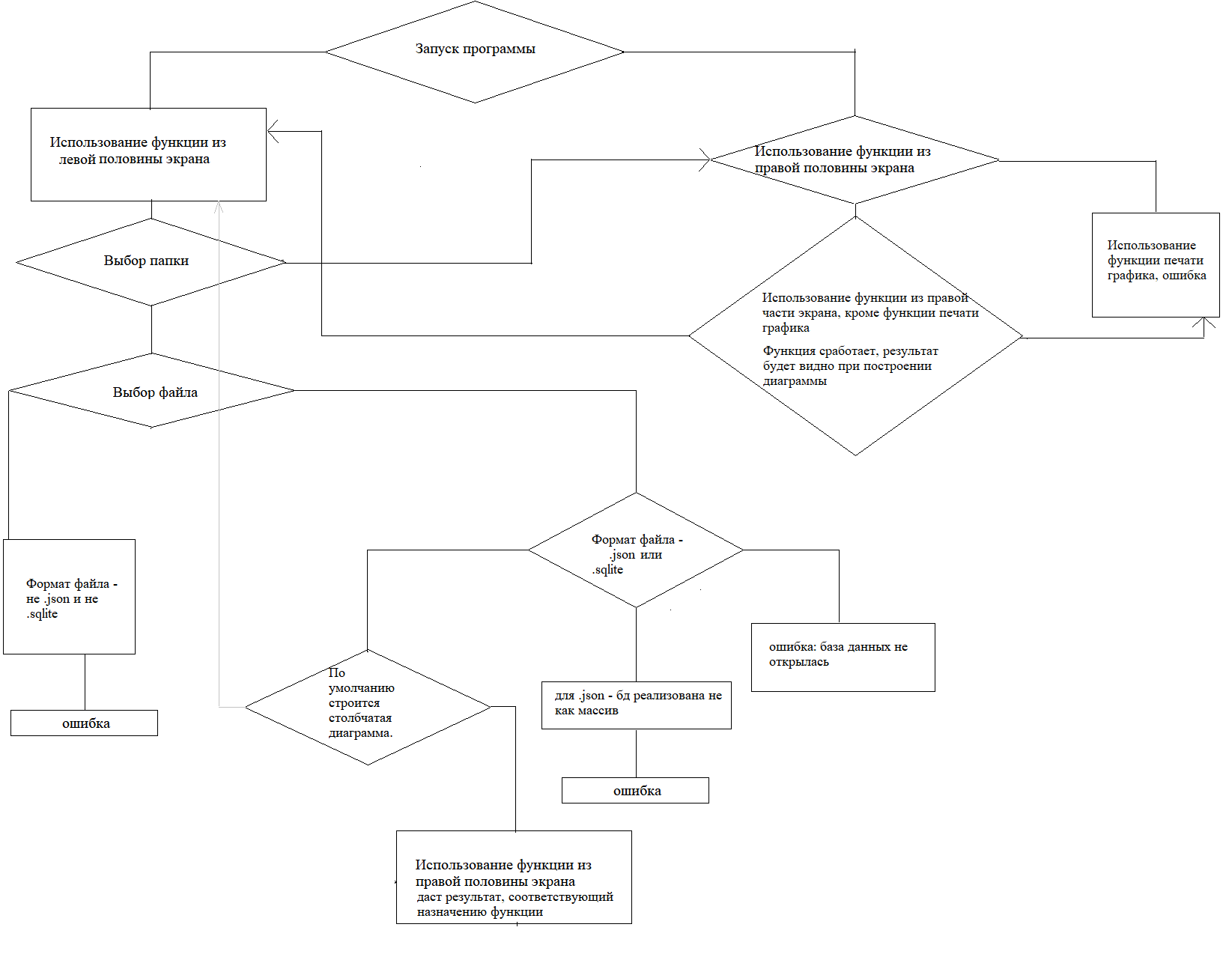
4) Инструкция пользователя

Экран разбит на две части. В левой возможен выбор базы данных для построения графика. В правой отрисовывается график. Изначально в левой половине мы видим файлы корневой папки, правая половина пуста.

Выбрать интересующую папку можно с помощью кнопки “open folder”. Из папки можно выбрать файлы формата .json и .sqlite. Для выбранной базы данных отобразится график.

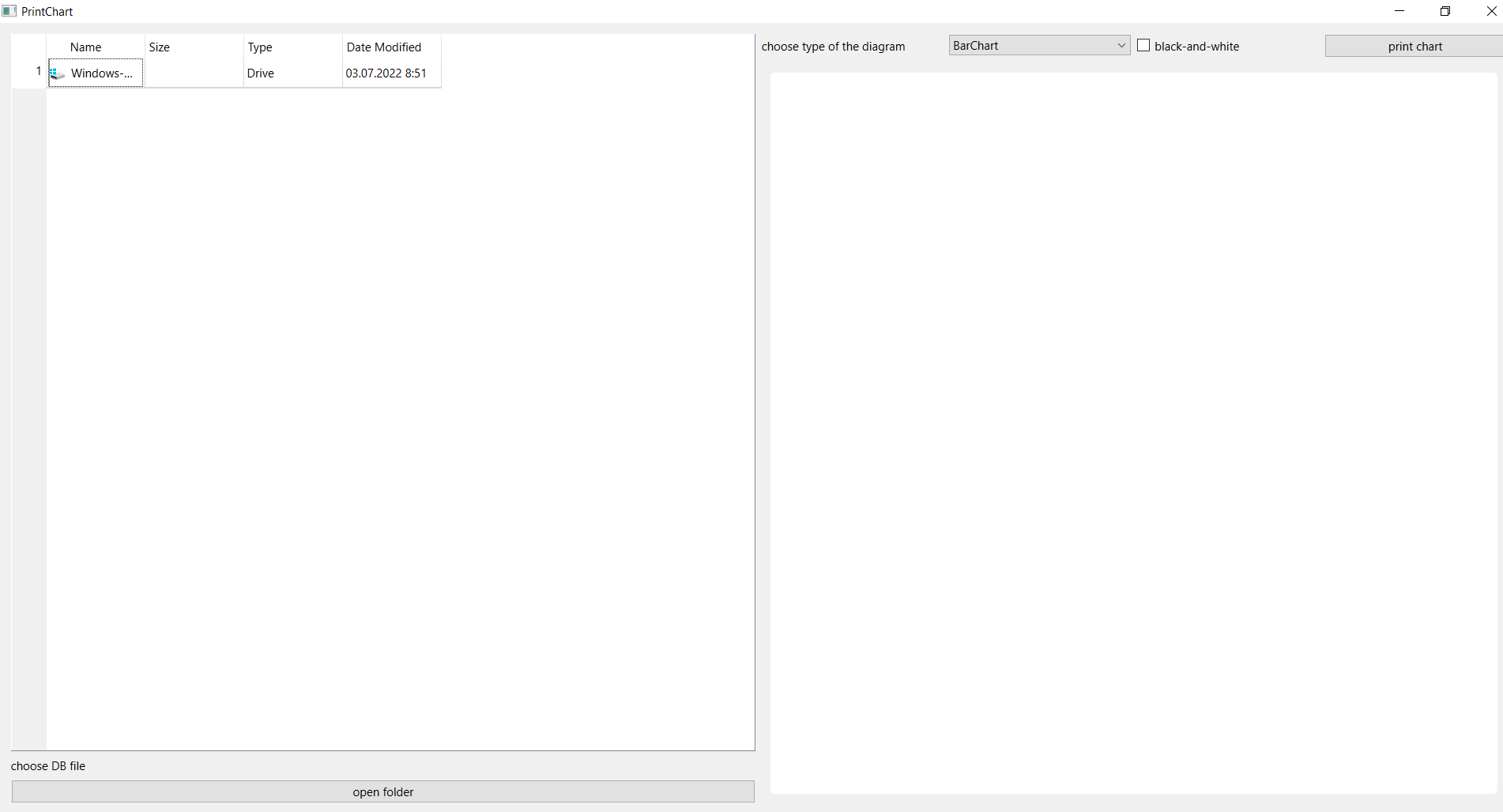
Для взаимодействия с графиком доступны выбор типа диаграммы (при наведении на “choose type of the diagram” появляется выпадающий список, предоставляющий возможность выбрать тип графика - столбчатая диаграмма при нажатии на “BarChart” или круговая диаграмма при нажатии на “PieChart” ). Есть возможность отобразить график как черно-белый при использовании чекбокса “black-and-white”. Также есть кнопка “print chart”, позволяющая сохранить диаграмму в формате pdf.

5) Тестирование

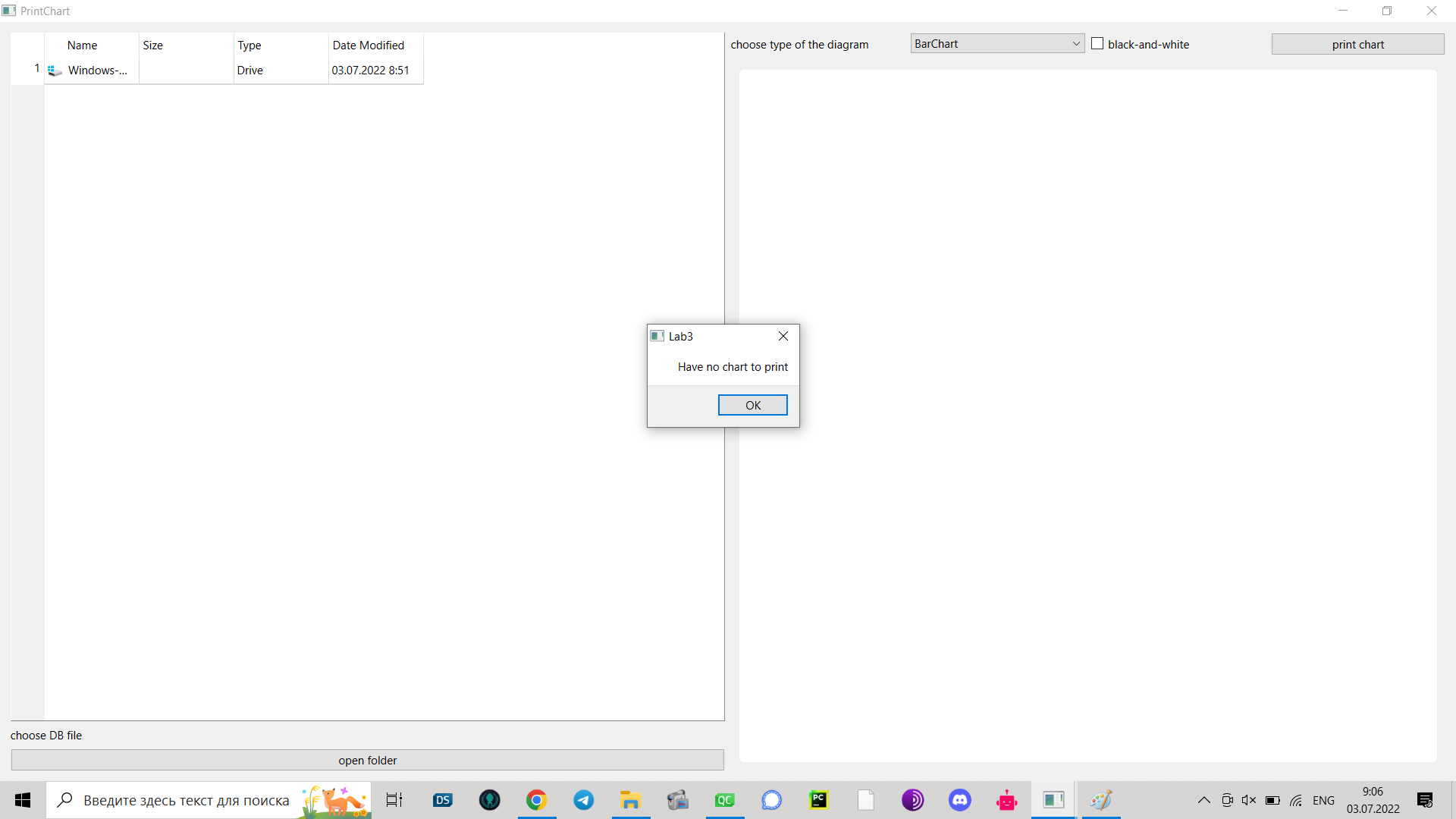
Сценарий тестирования:Сохранение диаграммы в формате PDF протестировано, результат представлен в файле printChart.pdf.

Тестирование:

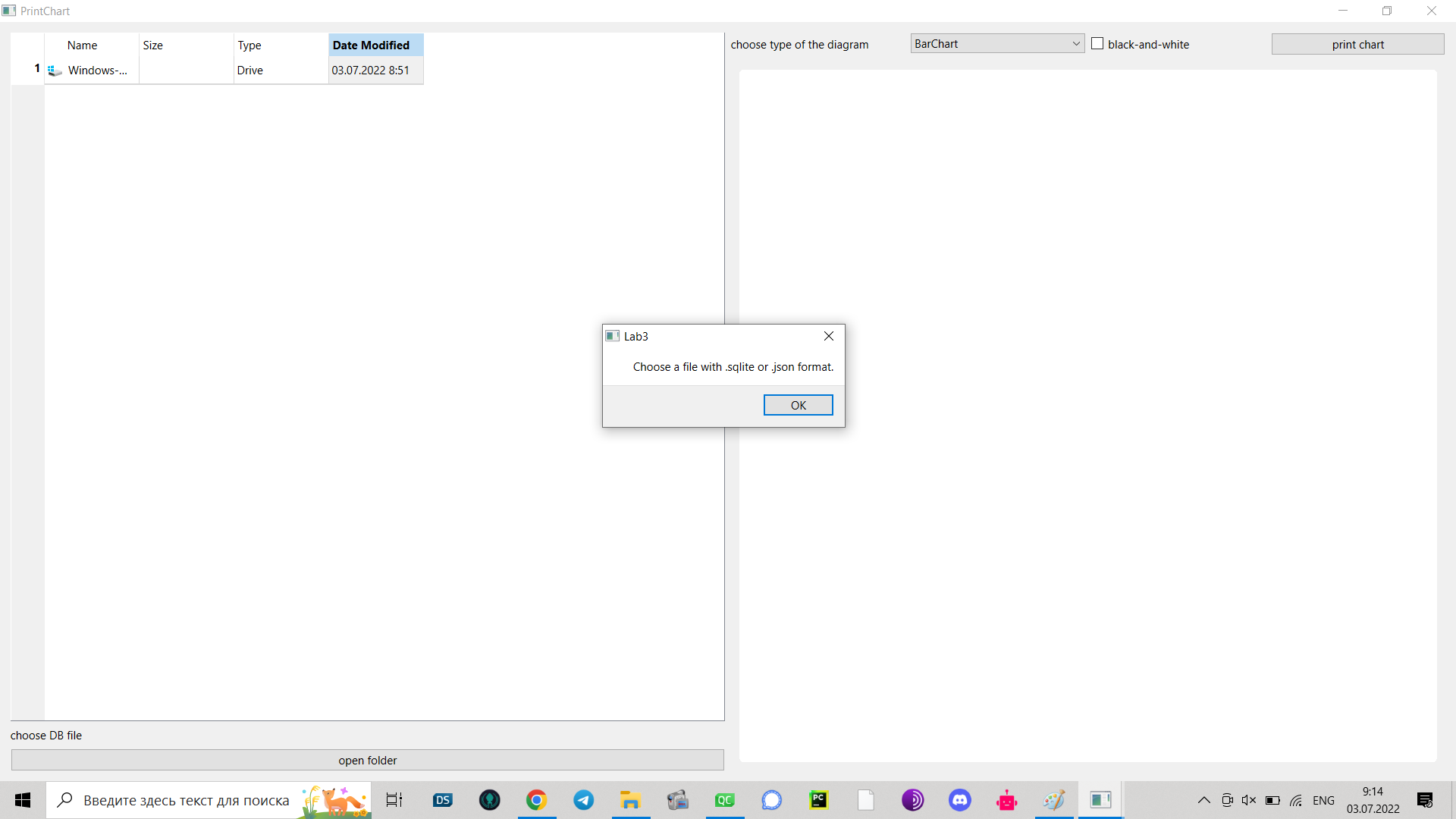
1. Запуск программы



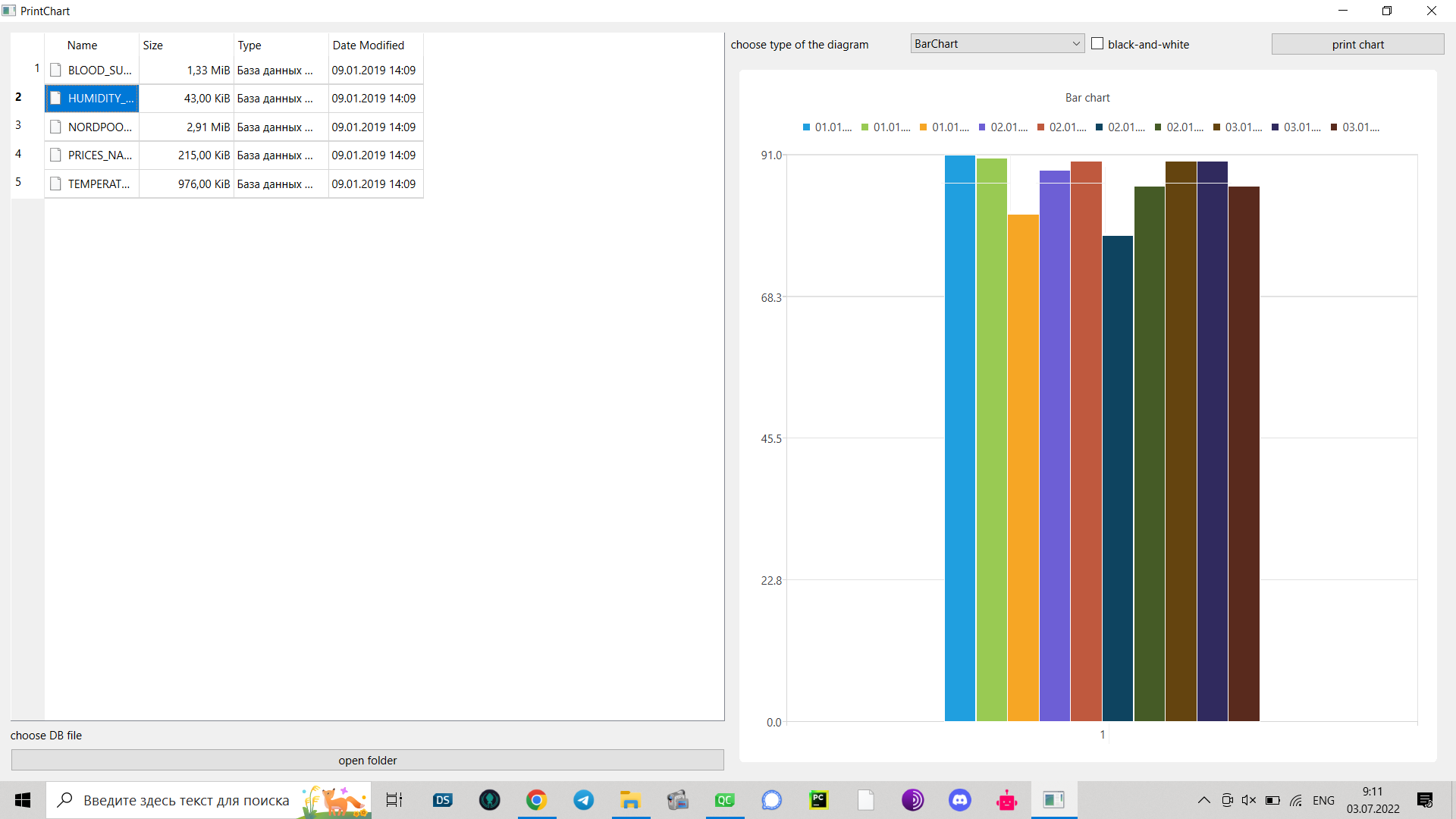
1. Использование функции печати графика при невыбранном файле с базой данных

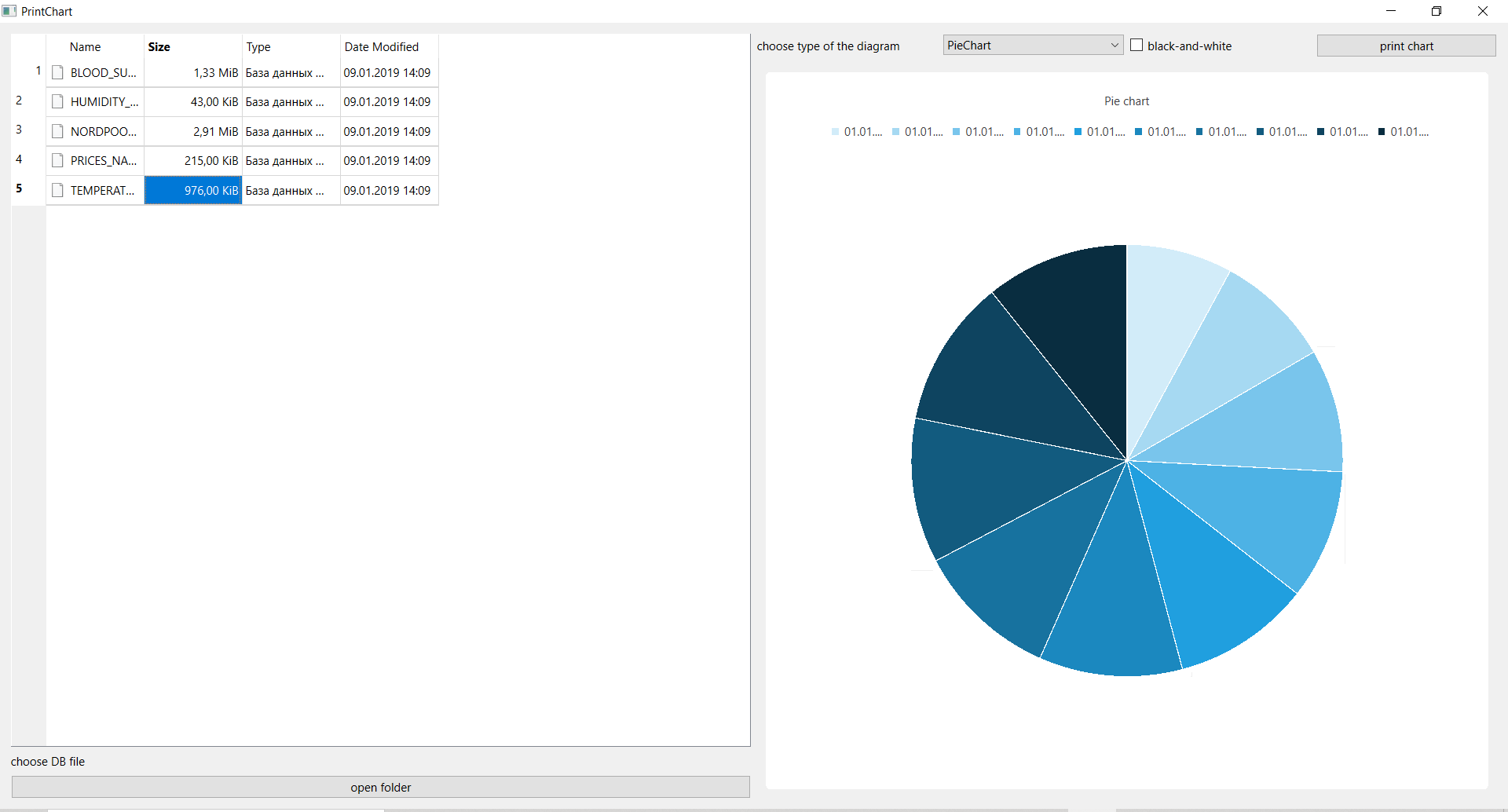


1. Выбор файла не формата json/sqlite



4.Цветная столбчатая диаграмма



5. Круговая диаграмма 

6. ЧБ столбчатая диаграмма