**Тилебалиев Самат гр. 931921**

**Лабораторная №3**1) Постановка задачи

Разработка приложения печати графиков.

***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

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).

**Общие требования к GUI**

Загружаем данные, путем выбора нужного файла. Данные в ПО не отображаем, отображаем только график, построенный относительно считанных данных.

При печати в pdf выбираем место сохранения графика.

2) Предлагаемое решение

В первую очередь у нас стоит вопрос построения правильной архитектуры, в котором зависимость между компонентами будет минимальной.  
Так как предполагается возможность добавления новых видов диаграмм и данных, то нам необходимо учесть, чтобы модули высокого уровня не зависили от внедрения модулей низкого уровня, то есть новых предоставляемых компонентов. Механизм внедрения зависимости в модуль высокого уровня называется Dependency Injection (DI).  
Для разрешения такой зависимости DI использует IOC — контейнер.

С помощью IOC — контейнера мы получаем возможность задавать новую реализацию компонента, при этом не изменяя ее интерфейс пользования в модулях высокого уровня.

Таким образом, для уменьшения зависимости между компонентами системы будем использовать IOC — контейнер.

В предложенной архитектуре использовалась концепция MVC — модель-вид-контроллер.  
Идеология MVC состоит в разделении на три отдельных модуля, которые взаимодействуют между собой, но при этом их разработка независима относительно друг друга.

В виде модели в разработке использовалась готовая файловая модель QFileSystemModel, предложенная QT, для работы с файлами. Модель в общем случае отвечает за соединение с данными.

В качестве представления использовался класс QTableView от QT для отображения файлов в табличном виде с помощью модели QFileSystemModel.

Обработка выбора файла предоставляется классом QItemSelectionModel, с помощью которого, определяя модельный индекс выбранного файла для модели, мы получаем путь к этому файлу.

Итак, для взаимодействия с диаграммой нам необходим класс, назовем его Chart, который будет извлекать данные для печати и отображать диаграмму на основе полученных данных. Нужно уточнить, что тип поддерживаемого файла и вид диаграммы задаются пользователем и они впоследствии могут расширяться. Для начала давайте наделим класс Chart объектами, которые непосредственно будут работать с диаграммой — QChart, управляющий графическим представлением диаграммы и QChartView, отображающий диаграмму. Теперь необходимо добавить базовый интерфейс пользования к нашему классу, которые не будут зависеть от каких-либо предоставляемых компонент. Это следующие методы: сохранения диаграммы в файл формат pdf, обновление диаграммы, очистки диаграммы, геттеры для QChart и QChartView и возможность сменить цвет диаграммы.  
Данные извлеченные из файла будут также храниться в классе Chart для удобства пользователя. К примеру, когда он хочет перерисовать график на другой цвет, то ему не будет необходимости передавать данные заново.  
Реализация класса Chart для полного отображения диаграммы не должна постоянно меняться при добавлении нового функционала со стороны пользователя, в данном случае - тип поддерживаемого файла и вид диаграммы.

Для этого нужно определить два интерфейса:  
интерфейс извлечения данных из файлов, назовем его IDataReader,  
и интерфейс рисования диаграммы, назовем его IChartDrawing и соотвествующие функции для получения интерфейса: getDataReader и getChartDrawing

IDataReader должен предоставить метод наследникам для извлечения данных из файла,

IChartDrawing - рисование диаграммы.

Определим реализации для заданных двух интерфейсов -

Реализации для IchartDrawing:  
barChartDrawing - для рисования диаграммы в виде гистограммы,  
pieChartDrawing — для рисования диаграммы в виде круга.

Реализации для IdataReader:  
DataReaderSql — для извлечения из файлов типа .sqlite

DataReaderJson — для извлечения из файлов типа .json

Итак, реализация функции полного отображения данных в виде диаграмм в классе Chart состоит из следующих частей:  
1. Получение интерфейса IDataReader и соответствующий вызов функции извлечения.

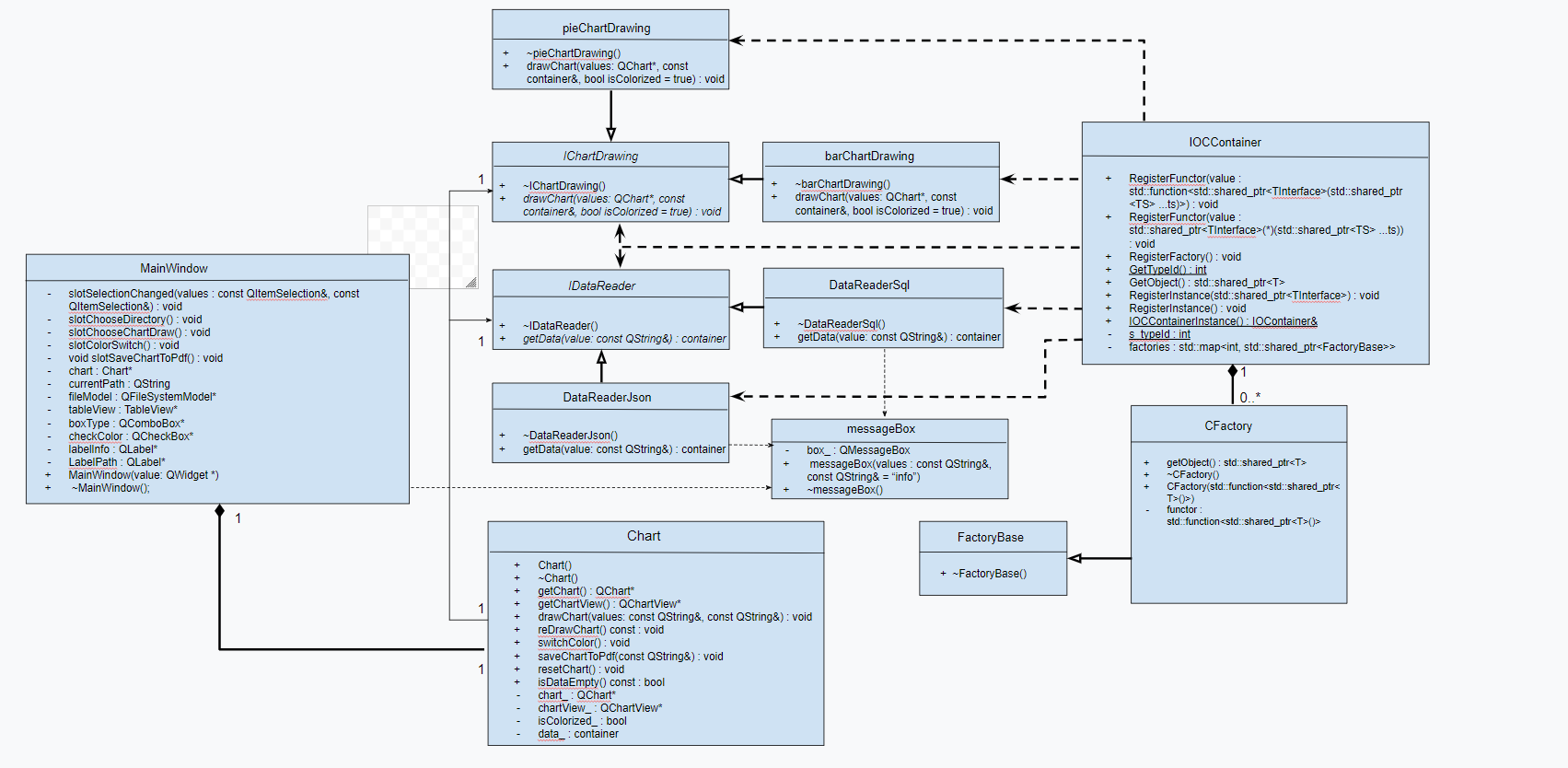
2. Получение интерфейса IChartDrawing и соответсвующий вызов фунции рисования диаграммы.

Нам скрыты реализации получения интерфейсов и реализации (их может быть бесчисленное множество).

Итак, класс Chart полностью готов к полной эксплуатации, зависимости от новых функционалов не будет.

Теперь перейдем к реализации получения интерфейсов IDataReader и IChartDrawing, которые могут использоваться в модулях высого уровня(как в случае выше).  
Эти функции для возвращения реализации применяют IOC-контейнер. Новая реализация интерфейса может быть предоставлена пользователем из вне и при этом для ее получения нет необходимости что-то менять в предложенных двух функциях getDataReader и getChartDrawing, что как раз таки приводит к снижению связности архитектуры.

Полученная UML-диаграмма:



UML – диаграмма доступна по ссылке: https://docs.google.com/drawings/d/172WHjivs8B3tKVqfbKWK1Fo5bx9Ut7EDmO4BpoigYoc/edit?usp=sharing  
Скриншот также прилагается в корневой папке проекта.

https://github.com/tmpsam22/Application-for-printing-graphics.-LAB-3

message\_box.h - представлен класс для вывода оконного сообщения

mainwindow.h – главное окно приложения

ioc\_container.h – представлен IOC-контейнер

data\_reader\_if.h - представлен интерфейс извлечения данных из файлов разного типа

data\_reader\_impls.h - представлены классы извлечения данных

common\_container.h - содержит алиасы данных для диаграммы

chart\_drawing\_if.h - представлен интерфейс рисования диаграммы

chart\_drawing\_impls.h – представлены классы рисования диаграммы

chart.h - представлен класс для взаимодействия с диаграммой

chart.cpp - представлена реализация класса для взаимодействия с диаграммой

chart\_drawing\_impls.cpp - представлены реализации классов рисования диаграммы

data\_reader\_impls.cpp - представлены реализации классов извлечения данных

ioc\_container.cpp - представлена реализация IOC-контейнера

mainwindow.cpp – реализация главного окна

4) Инструкция пользователя

Главное окно программы визуально делится на левую и правую часть.  
Левая часть окна показывает все файлы в директории. В нижней части выводится сообщения о текущей директории и о том, что программа для печати диаграмм поддерживает файлы только расширения типа .sqlite или .json.

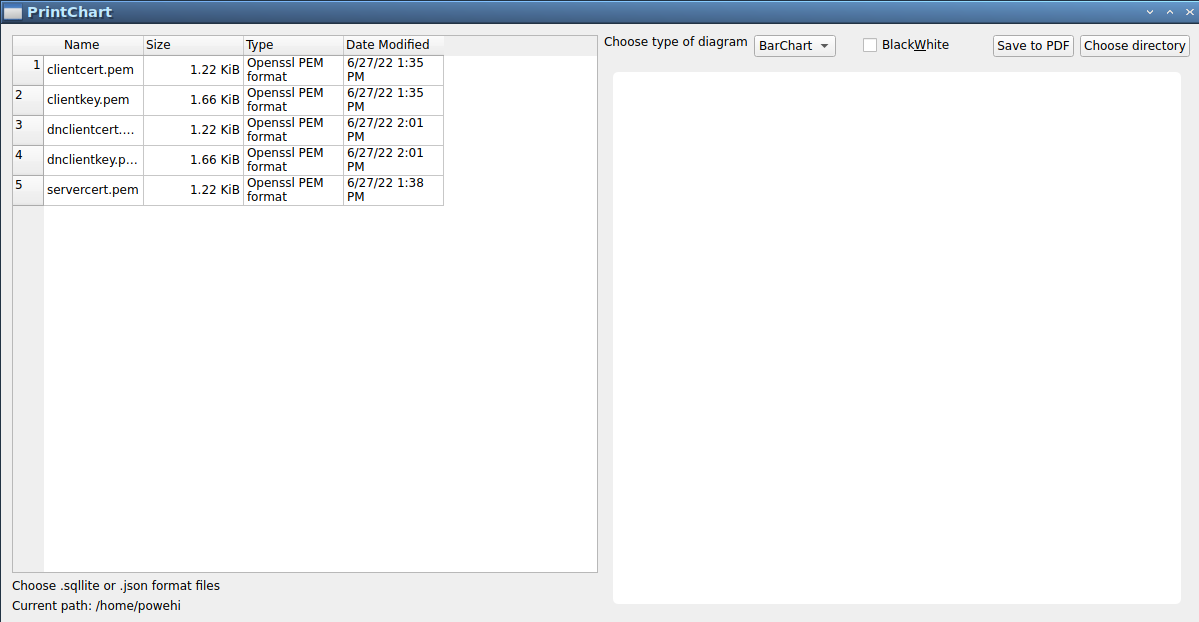
Правая часть окна отображает саму диаграмму, построенную из данных, извлеченных из файла нужного типа. Диаграмма по умолчанию отображается в виде цветной гистограммы.

В верхней части показывается меню, включающий в себя следующие команды:

* выпадающий список для выбора вывода диаграммы : в виде круга или гистограммы
* флажок для вывода диаграммы черно-белым цветом
* кнопка для сохранения диаграммы в формате pdf
* кнопка для выбора директории

Для начала, чтобы отобразить диаграмму, пользователю необходимо выбрать директорию, в котором лежат файлы[.sqlite/.json] с данными.   
Ему предоставляется возможность выбирать тип диаграммы из предлагаемых наборов в выпадающем списке и менять его цвет: из цветного в черно-белый и обратно.

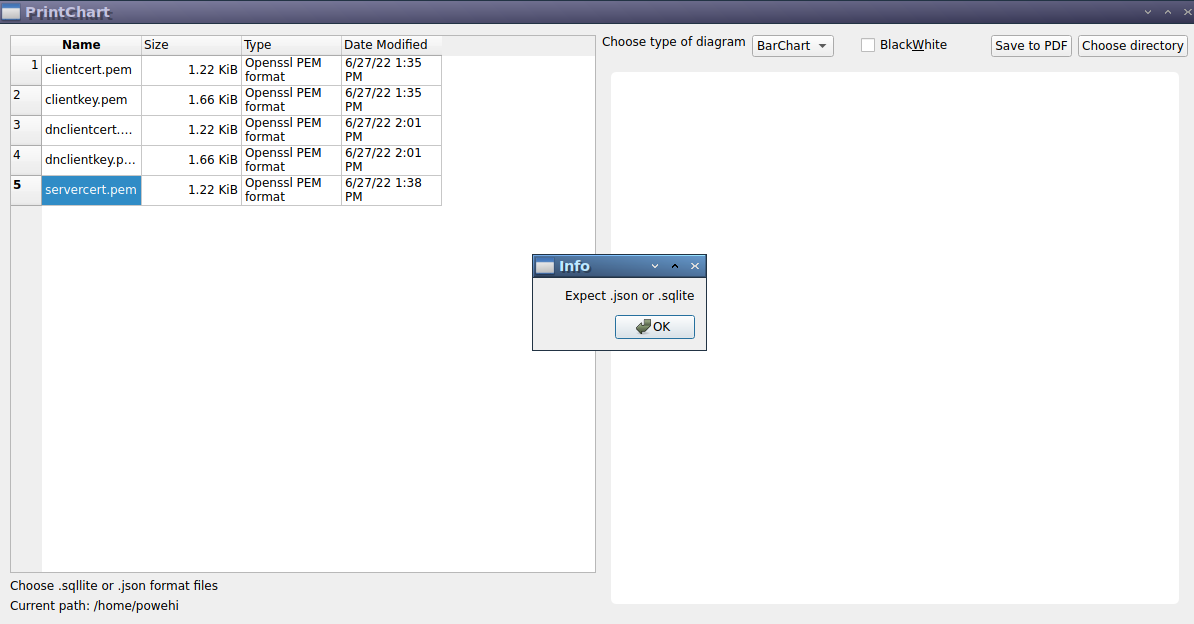
При сохранении диаграммы в файл формата pdf, по умолчанию открывается текущая директория. Выбрав нужную директорию, пользователь может сохранить диаграмму, указав желаемое имя файла.

Окно приложения:  
  


5) Тестирование

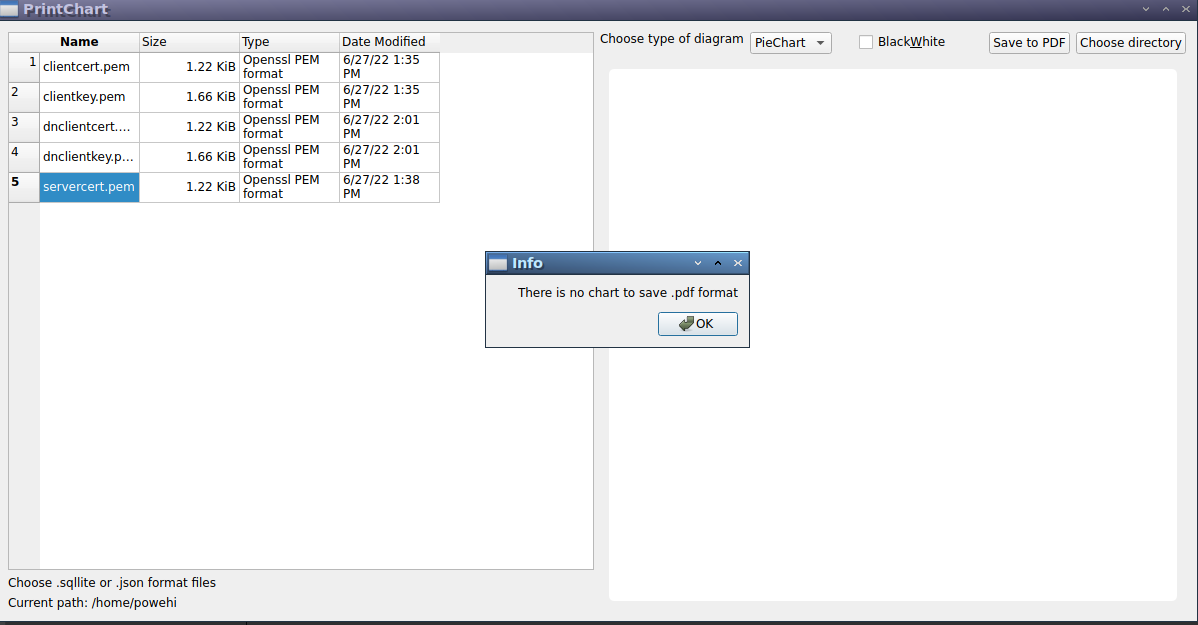
Рассмотрены все случаи взаимодействия пользователя с приложением печати диаграмм.

1. Пользователь выбирает файл в текущей директории не поддерживаемый приложением.  
Результат: выводится окно с сообщением о том, какие типы файлов программа поддерживает для извлечения из него данных.

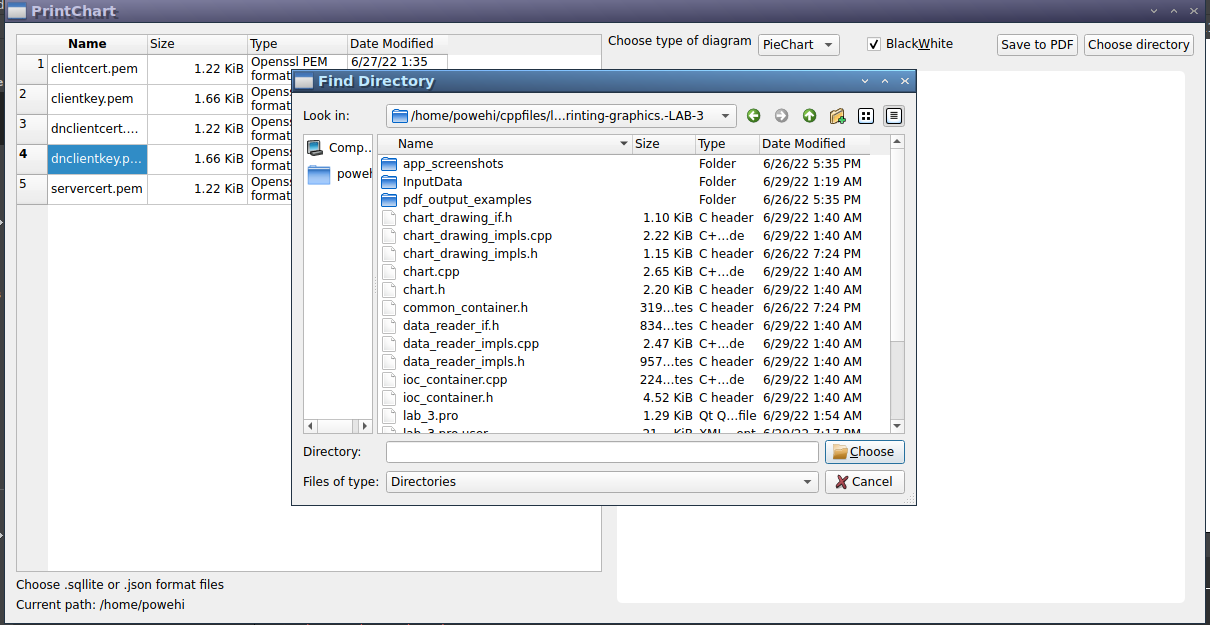


­2. Пользователь хочет сохранить в файл формата pdf пустую диаграмму.

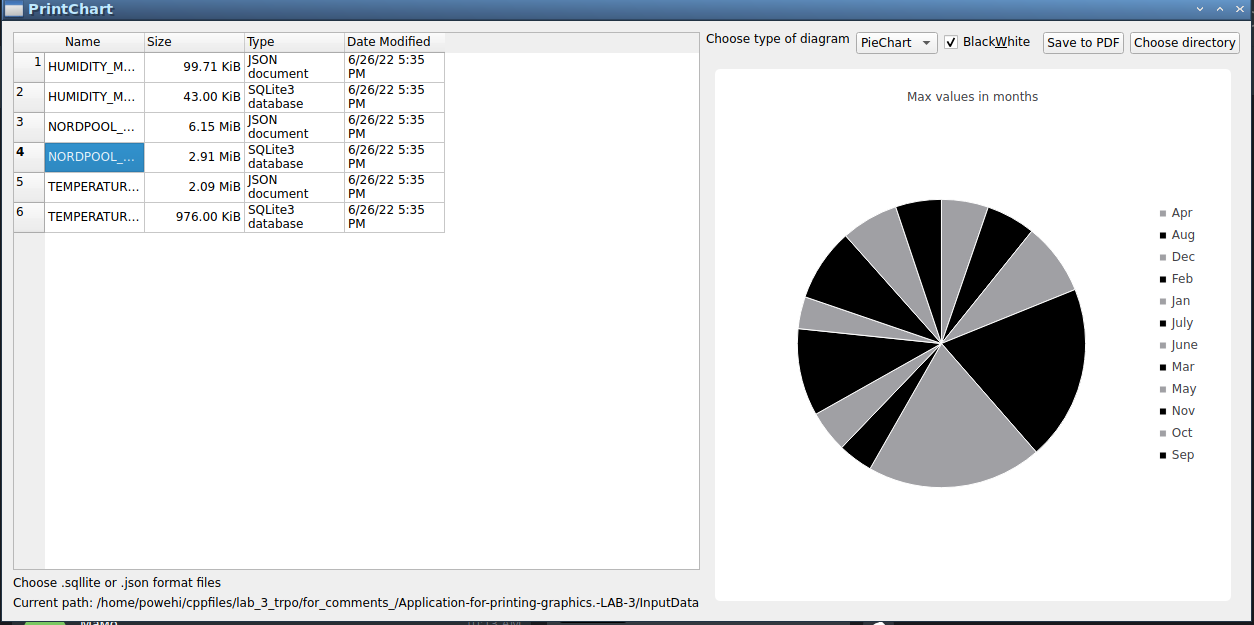
Результат: выводится окно для выбора директории.



3. Пользователь нажимает на кнопку выбора директории.  
Результат: выводится окно для выбора директории.

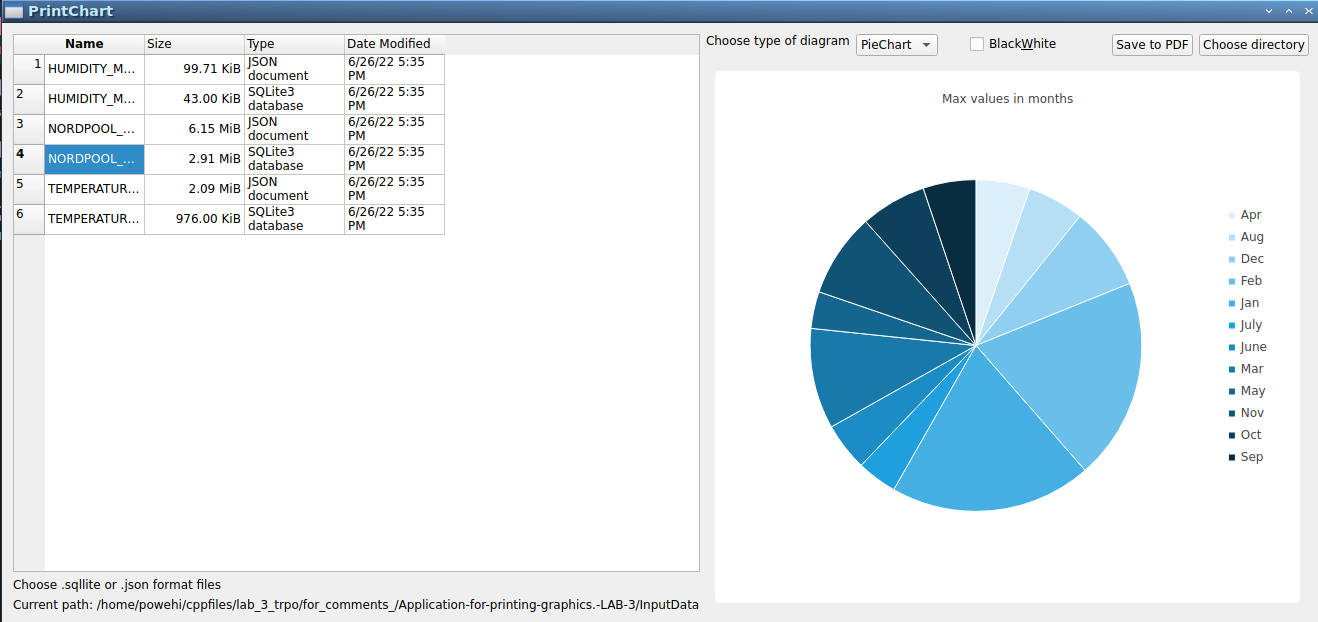


4.а Пользователь выбирает поддерживаемое приложением расширение файла с данными, указав тип диаграммы в виде круга. Цвет диаграммы – черно-белый.  
Результат: вывод круговой диаграммы в правой части окна:



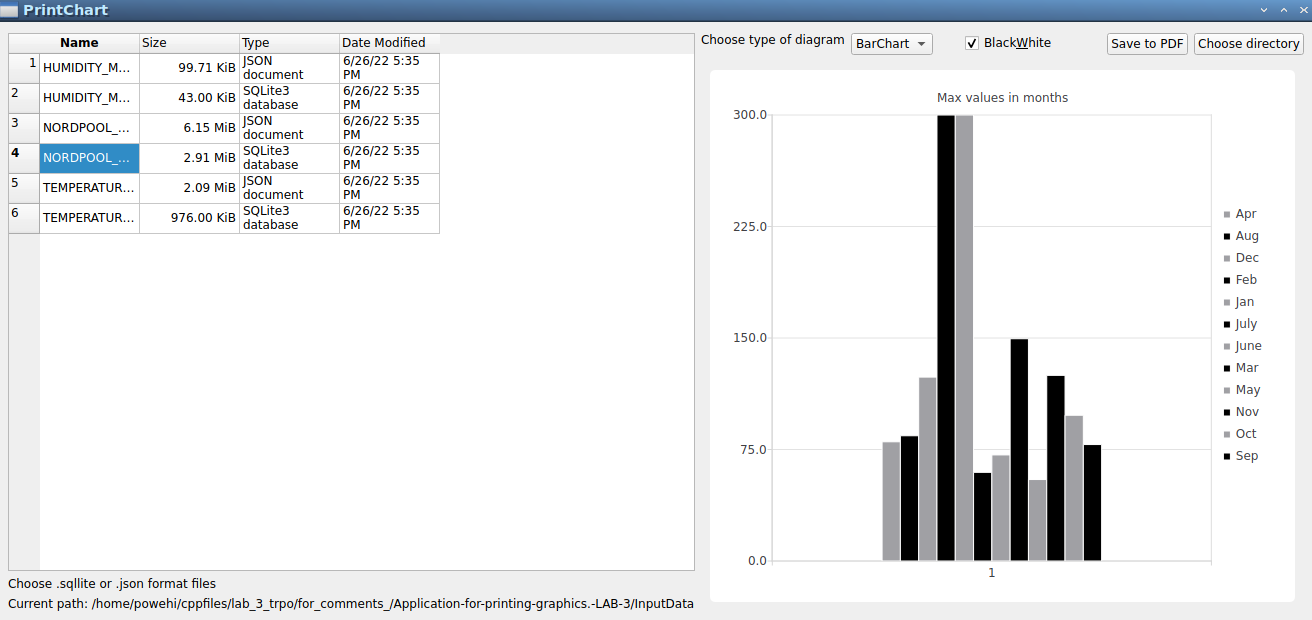
4.б Пользователь выбирает поддерживаемое приложением расширение файла с данными, указав тип диаграммы в виде круга. Цвет диаграммы – цветной.

Результат: вывод круговой диаграммы в правой части окна.



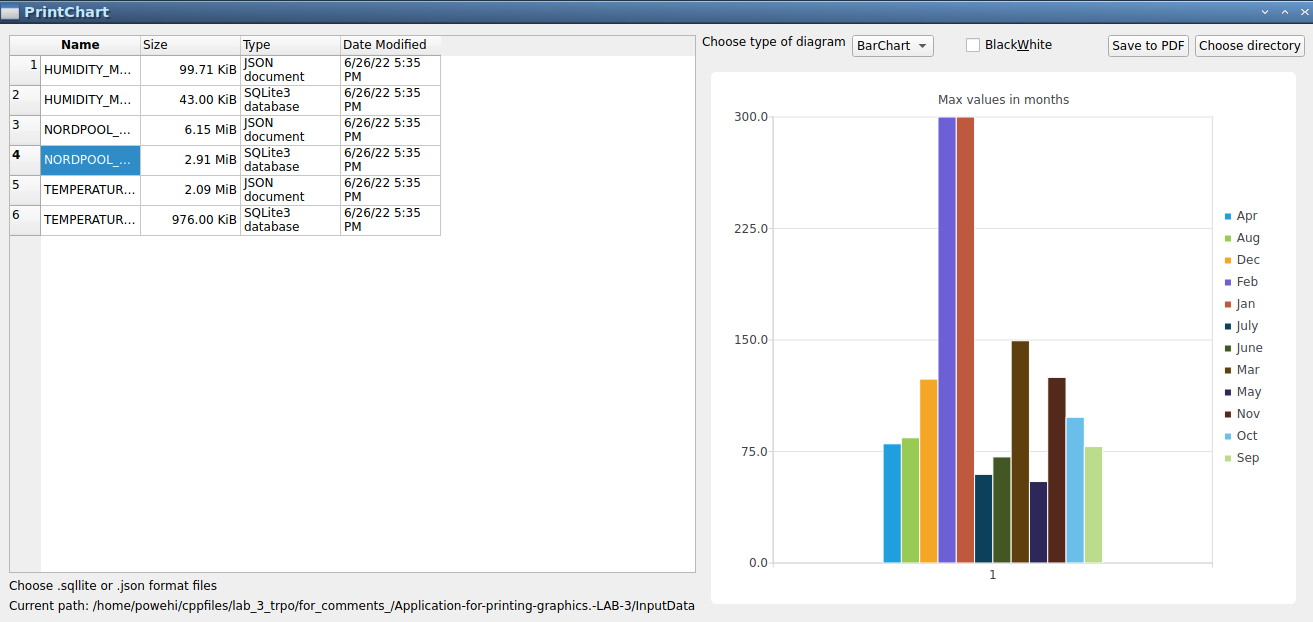
5.а Пользователь выбирает поддерживаемое приложением расширение файла с данными, указав тип диаграммы в виде гистограммы. Цвет диаграммы – черно-белый.

Результат: вывод гистограммы в правой части окна.

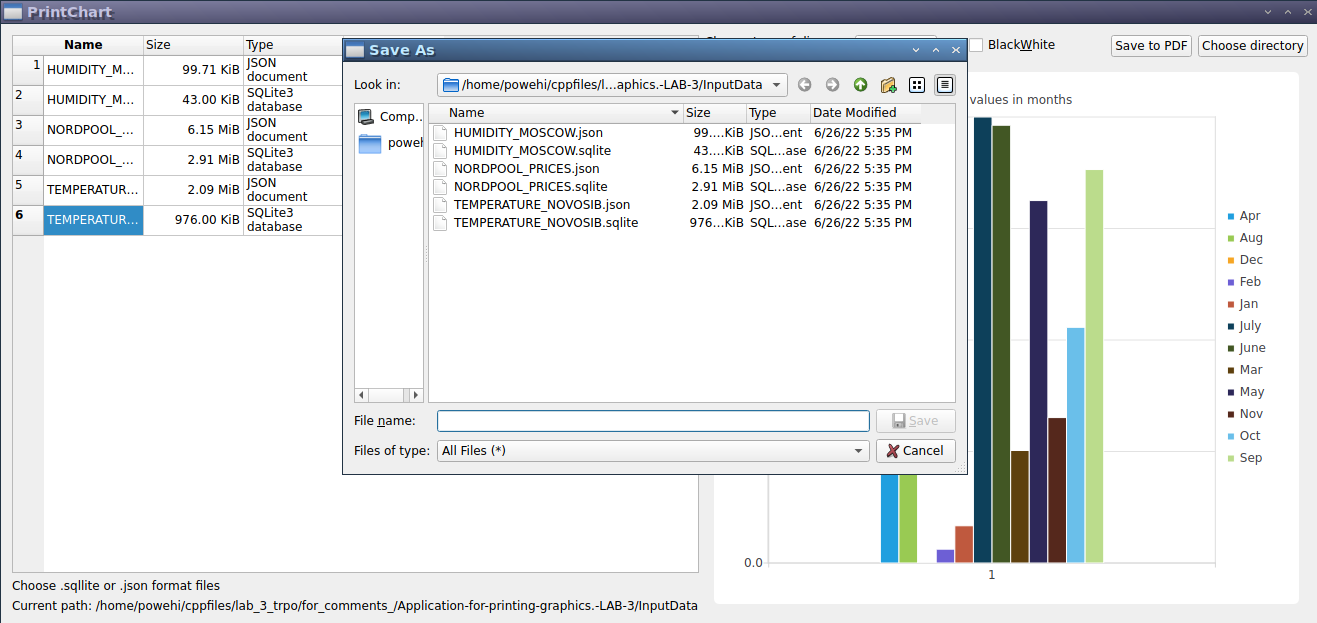


5.б Пользователь выбирает поддерживаемое приложением расширение файла с данными, указав тип диаграммы в виде гистограммы. Цвет диаграммы – цветной.

Результат: вывод гистограммы в правой части окна.



6. Пользователь хочет сохранить диаграмму в файл формата .pdf  
Результат: выводится окно с возможностью выбора директории и имени файла.



7. Пользователь выбирает не поддерживаемый приложением файл при выведенной диаграммы.  
Результат: диаграмма должна быть удалена.

