

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной
безопасности и криптографии

Система резервирования данных

ДИПЛОМНАЯ РАБОТА

студента 6 курса 631 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Старичкова Павла Александровича

Научный руководитель
старший преподаватель

22.01.2024 г.

А. А. Лобов

Заведующий кафедрой
д. ф.-м. н., доцент

22.01.2024 г.

М. Б. Абросимов

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Обзор методов резервного копирования	5
1.1 Полное резервное копирование.....	5
1.2 Инкрементальное резервное копирование	8
1.3 Дифференциальное резервное копирование	10
1.4 Другие методы	12
2 Обзор существующих решений для резервного копирования данных.....	17
2.1 Обзор Яндекс.Диск	19
3 Программная реализация.....	23
3.1 Подсистема работы с облаком.....	24
3.2 Подсистема резервного копирования	26
4 Результаты работы программы	30
4.1 Аутентификация.....	30
4.2 Подсистема работы с облаком.....	33
4.3 Подсистема резервного копирования	36
ЗАКЛЮЧЕНИЕ	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	48
ПРИЛОЖЕНИЕ А Листинг программы.....	51

ВВЕДЕНИЕ

В современном мире практически каждый человек взаимодействует с компьютерными системами, которые широко применяются в различных компаниях и предприятиях. Основное предназначение этих систем заключается в обработке и хранении обширных объемов информации.

В процессе своей деятельности компании накапливают огромные массивы данных, часть из которых представляет собой критически важную информацию для пользователей или ключевую для устойчивого функционирования компании. Такая информация включает в себя финансовые данные, коммерческие и производственные тайны, а также личную информацию. Потеря таких данных может стать непредсказуемой проблемой. В этом контексте возникает вопрос: как обеспечить защиту от потери информации?

Один из эффективных методов защиты от потери данных, вызванной различными причинами, такими как сбой оборудования или злонамеренные действия, представляет собой использование систем резервного копирования и восстановления информации. Эти системы нацелены на поддержание целостности и доступности данных. Их ключевая функция заключается в создании дубликатов важной информации в безопасном хранилище. В случае повреждения данных в основной компьютерной системе, эти системы позволяют восстановить информацию из резервной копии.

Для выполнения процедуры резервного копирования часто используются специальные программно-аппаратные системы, известные как системы резервного копирования. Они предназначены для регулярного автоматического копирования системных и пользовательских данных, а также для оперативного восстановления данных.

В рамках данной работы будут рассмотрены различные методы резервного копирования и хранения копий, их преимущества, недостатки и области

применения. Также будет проведен обзор существующих систем резервного копирования данных. В конечном итоге предполагается создание программного продукта, позволяющего эффективно создавать и восстанавливать резервные копии данных.

1 Обзор методов резервного копирования

Резервное копирование данных – это процесс сохранения избыточных копий файлов и каталогов, находящихся на локальных дисках, на сменные носители или в защищенное хранилище, которое может быть как в локальной сети, так и вне ее. Избыточные копии могут использоваться для восстановления в случае, если оригинальные файлы потеряны или повреждены. Резервное копирование чаще всего планируется на ежедневной основе. При этом любые новые файлы, или файлы, измененные с момента последнего копирования, оказываются на носителе, так что они будут доступны для восстановления на диск [1].

В зависимости от важности и частоты использования и изменения данных выбирают один из нескольких основных способов копирования:

1. полное резервное копирование (Full backup);
2. инкрементальное резервное копирование (Incremental backup);
3. дифференциальное резервное копирование (Differential backup).

1.1 Полное резервное копирование

Этот метод резервного копирования представляет собой создание полной копии всего исходного объема данных. Он выделяется своей простотой и быстротой процесса восстановления, что делает его предпочтительным вариантом для обеспечения безопасности данных. Однако его основными ограничениями являются высокие требования к объему доступной памяти, поскольку каждая копия равна размеру резервируемых данных. Также метод чрезмерно времязатратен при обработке больших объемов данных, и его выполнение создает значительную нагрузку на сеть, что может мешать нормальному функционированию инфраструктуры во время выполнения резервирования [2].

Полная резервная копия

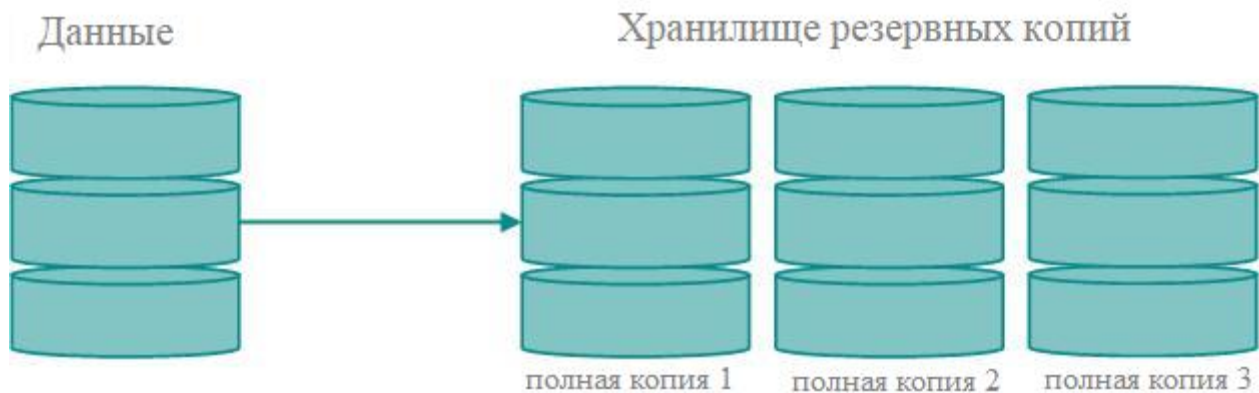


Рисунок 1 – Полное резервное копирование

В связи с вышеуказанными характеристиками полное резервное копирование часто применяется с длинными интервалами между выполнениями и дополняется другими методами копирования. Полная копия, созданная при каждом выполнении, служит отправной точкой для последующих резервных операций.

Существуют усовершенствования этого метода, такие как механизм дедупликации и сжатие данных.

Механизм дедупликации представляет собой технологию, направленную на эффективное управление объемом данных путем выявления и удаления дублирующихся блоков или фрагментов информации в хранилище. В контексте резервного копирования, это позволяет существенно сократить объем необходимого хранилища, уменьшив дублирование данных. Процесс дедупликации начинается с выделения уникальных блоков данных в хранилище. Когда обнаруживается дублирование, система сохраняет только одну копию данного блока и создает ссылки на нее вместо хранения множественных копий. Это особенно полезно в тех случаях, когда одни и те же данные повторяются в различных частях системы или в разных версиях резервных копий [3].

Механизм сжатия данных представляет собой процесс уменьшения объема информации путем удаления избыточных или ненужных бит данных, сохраняя при этом основную структуру и содержание информации. Включает в себя следующие шаги: анализ данных, кодирование и сжатие, хранение или передача. При помощи сжатия можно снизить как требования к хранилищу, так и нагрузку на сеть.

При выборе метода полного резервного копирования важно учитывать специфику данных и потребности организации, чтобы найти оптимальный баланс между уровнем защиты и затратами на хранение и обработку информации.

Преимущества метода:

- 1) быстрое восстановление данных;
- 2) удобное управление хранилищем, поскольку весь объем данных сохраняется в одном файле копии.

Недостатки метода:

- 1) затраты времени на создание копии;
- 2) требование большого объема хранилища;
- 3) высокая нагрузка на сеть при использовании удаленных хранилищ для копий.

1.2 Инкрементальное резервное копирование

Инкрементальное резервное копирование исходит из полной копии данных в качестве отправной точки. При последующих операциях копирования сохраняются только измененные данные, произошедшие с момента последнего резервирования. Через определенные большие промежутки времени создается новая полная копия.



Рисунок 2 – Инкрементальное резервное копирование

При полном восстановлении системы необходимо начать с восстановления из последней полной копии, а затем последовательно восстанавливать данные из инкрементальных копий в порядке их создания. Однако этот процесс может занимать значительное время, так как требуется сначала вернуться к изначальной полной копии, а затем последовательно восстановить все инкрементальные изменения. В случае отсутствия или повреждения хотя бы одной инкрементальной копии, полное восстановление данных становится невозможным. Инкрементальное резервное копирование применяется с целью оптимизации использования хранилища.

Инкрементальное копирование довольно простой и эффективный метод резервного копирования. Он может использоваться как в больших и малых предприятиях, так и в домашних условиях. Для больших предприятий, где объемы

данных могут быть огромны, данный метод позволяет существенно снизить нагрузку на инфраструктуру, а также минимизировать потери в случае сбоев, т.к. создание инкрементальных копий менее затратно и их можно проводить чаще. Для малых предприятий, имеющих ограниченные ресурсы, данный метод предоставляет возможность экономии объема хранилища. И для дома это довольно удобный способ сохранять важные личные данные.

Преимущества:

- 1) быстрое выполнение резервного копирования, поскольку в копию включаются только измененные данные;
- 2) требуется меньший объем хранилища, так как сохраняются лишь изменения;
- 3) гибкость в выборе частоты выполнения копирования, при этом каждый инкремент предоставляет отдельную точку восстановления;
- 4) меньшая нагрузка на сеть по сравнению с полным копированием.

Недостатки:

- 1) медленный процесс полного восстановления, требующий последовательного восстановления изначальной полной копии и всех последующих инкрементальных изменений;
- 2) успешное восстановление данных зависит от целостности всех инкрементов в цепочке, и отсутствие или повреждение хотя бы одной из них может привести к невозможности полного восстановления данных.

1.3 Дифференциальное резервное копирование

Дифференциальное резервное копирование представляет собой средний вариант между полным и инкрементным методами. Стартовой точкой служит полная копия данных, после чего периодически создаются копии только измененных данных. В отличие от инкрементного метода, где изменения измеряются относительно предыдущей копии, в дифференциальном методе изменения всегда отсчитываются от полной копии. Этот метод совмещает преимущества и недостатки полного и инкрементного копирования, представляя их в более сбалансированной форме.

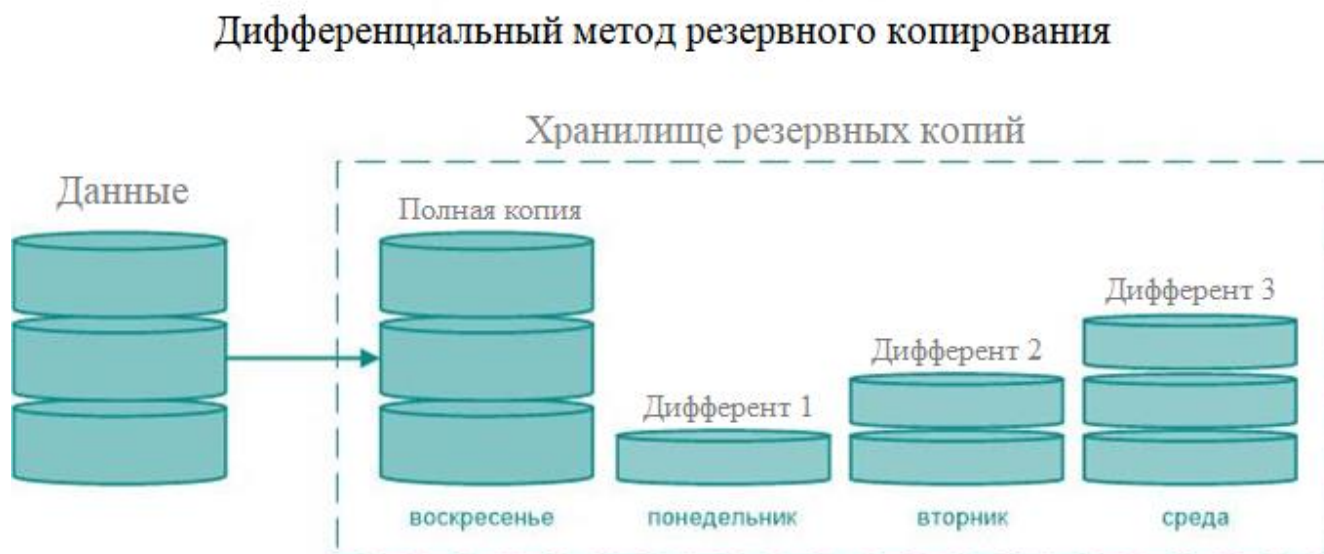


Рисунок 3 – Дифференциальное резервное копирование

Дифференциальное резервное копирование обеспечивает более быстрое восстановление данных по сравнению с инкрементным методом, поскольку для восстановления требуются всего две части: полная резервная копия и последняя дифференциальная копия. Скорость выполнения резервного копирования и восстановления занимает промежуточное положение между полным и инкрементным методами. Резервное копирование выполняется быстрее, чем полное, но медленнее, чем инкрементное. Восстановление занимает больше времени, чем у полного метода, но меньше, чем у инкрементного. Объем памяти,

требуемый для дифференциального резервного копирования, на протяжении определенного периода меньше, чем для полного копирования, но больше, чем для инкрементного [4].

Преимущества:

- 1) резервное копирование быстрее, чем полное, но медленнее, чем инкрементное;
- 2) восстановление быстрее, чем инкрементное, но медленнее чем полное;
- 3) надежность восстановления: для восстановления требуется только полная и последняя резервная копия.

Недостатки:

- 1) увеличение времени и объема: Каждое последующее копирование требует больше времени и занимает больше объема хранилища по сравнению с предыдущими.

Если подвести итог обзора основных методов, то можно сказать, что:

- полное копирование: Эффективно для создания базовой точки, но может быть неудобным для регулярных резервных копий;
- инкрементное копирование: Экономично по ресурсам, но требует внимательного управления для обеспечения полной восстанавливаемости;
- дифференциальное копирование: Обеспечивает баланс между временной эффективностью и экономией места, что делает его привлекательным выбором для многих сценариев.

1.4 Другие методы

1) Обратное инкрементальное резервное копирование.

Этот метод резервного копирования начинается с создания полной копии данных. В ходе последующих резервных операций все данные из предыдущей полной копии переносятся в новую резервную копию, и предыдущая полная копия заменяется инкрементом. Таким образом, каждая новая резервная копия содержит все изменения, произошедшие с момента предыдущего резервного копирования.

Принцип работы:

1) начальная точка: начало процесса обратного инкрементального резервного копирования представляет собой полную копию данных. Это служит базовой точкой для всех последующих резервных операций;

2) перенос данных: при каждом новом резервном копировании все данные из предыдущей полной копии перемещаются в новую резервную копию. Таким образом, актуальные данные оказываются в новой полной копии, а предыдущая полная копия становится инкрементом;

3) замещение предыдущей копии: после переноса данных предыдущая полная копия заменяется инкрементальной копией. Это позволяет поддерживать структуру резервных копий в порядке, а также минимизировать использование пространства хранения.

Обратное инкрементальное резервное копирование может быть особенно полезным в сценариях, где важно обеспечивать структурированное и эффективное управление хранилищем данных, при этом необходимость быстрого восстановления является приоритетной.

Метод обратного инкрементального резервного копирования

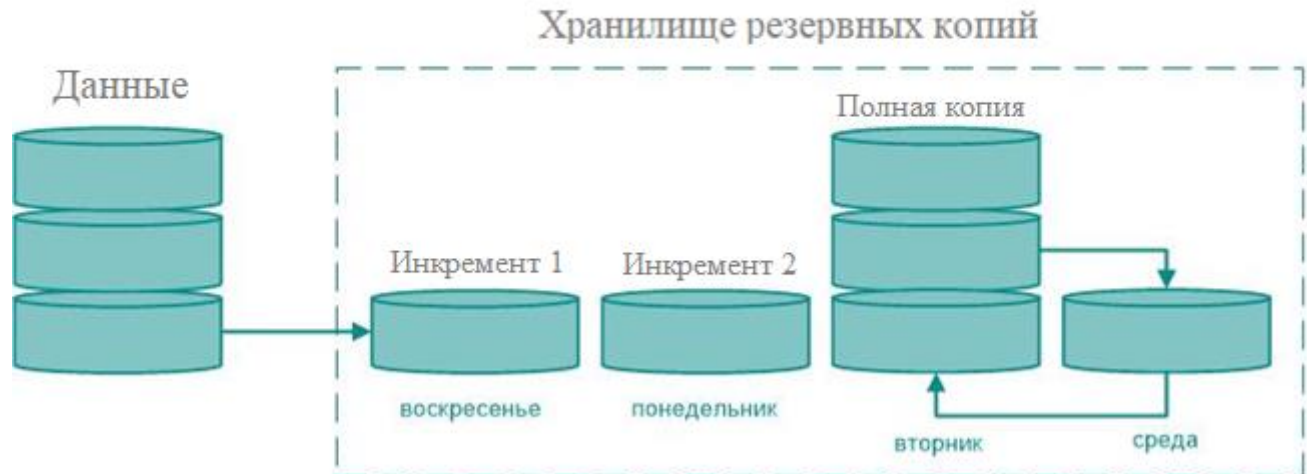


Рисунок 4 – Обратное инкрементальное резервное копирование

Преимущества:

1) эффективное использование места: обратное инкрементальное резервное копирование обеспечивает оптимальное использование пространства хранения, так как каждая новая резервная копия содержит только актуальные изменения;

2) относительная быстрота восстановления: поскольку все необходимые данные для восстановления находятся в последней полной копии, процесс восстановления может быть относительно быстрым;

3) структурированное хранение данных: метод обеспечивает четкую структуру хранения, где каждая полная копия является актуальным снимком данных на момент ее создания.

Недостатки:

1) увеличение времени переноса: перенос всех данных из предыдущей полной копии в новую может быть времязатратным процессом, особенно при наличии больших объемов информации;

2) зависимость от предыдущих копий: успешное восстановление данных требует наличия всех предыдущих полных копий, что делает процесс более уязвимым к потере одного из инкрементов.

2) Дельта-копирование.

Разновидность инкрементального метода. При таком методе в копию записываются только изменения, происходящие в файлах, а не переписываются полностью изменяемые данные. Его отличает высокая скорость создания, крайняя экономия места и значительно меньшее количество избыточных данных. Недостатком является крайне длительное восстановление данных, т.к. нужно собрать воедино все записанные измененные части файлов.

Принцип работы:

1) инкрементальные изменения: в процессе дельта-копирования резервной копии фиксируются только те изменения, которые произошли в файлах с момента последнего копирования. Это позволяет значительно сэкономить пространство хранения, поскольку записываются лишь дельты изменений;

2) экономия места: поскольку дельта-копирование не требует полной перезаписи файлов, оно обеспечивает высокую экономию места, особенно в случаях, когда изменения в файлах невелики по сравнению с их общим размером;

3) скорость создания: процесс создания дельта-копии происходит быстро, так как он фокусируется исключительно на записи измененных частей файлов, минимизируя время, необходимое для этого этапа.

3) Метод бинарных патчей.

Метод бинарных патчей, аналогично дельта-копированию, ориентирован на копирование измененных частей файлов. Однако в отличие от дельта-копирования, в методе бинарных патчей работа ведется на уровне битов информации.

Принцип работы:

1) изменения на битовом уровне: метод бинарных патчей фокусируется на записи измененных битов данных в файлах. Это позволяет точно отслеживать каждое изменение, даже на самом мельчайшем уровне, внутри файловой структуры.

2) применение бинарных патчей: при создании резервной копии выделяются и сохраняются только те биты, которые изменились с момента последнего копирования. Это позволяет значительно сократить объем передаваемой и хранимой информации.

3) эффективное использование ресурсов: метод бинарных патчей обеспечивает высокую экономию места и ресурсов, так как он оперирует на уровне измененных битов, минимизируя избыточные данные.

4) Зеркалирование.

Зеркалирование — это метод создания точной копии исходных данных, схожий с полным копированием. Используется на аппаратном уровне в массивах RAID1 или при создании сайтов-зеркал.

В отличие от полного копирования, зеркалирование не включает в себя процессы архивирования и систематизации изменений в данных за определенный период. Исходные данные сначала «зеркалируются» в копию. После этого в зеркало копируются только те файлы, которые претерпели изменения, минимизируя объем передаваемых данных.

Одной из ключевых особенностей зеркалирования является возможность прямого доступа к данным без необходимости выполнения процесса восстановления. Это делает метод особенно привлекательным для сценариев, где требуется быстрый доступ и высокая отказоустойчивость.

Однако следует учитывать, что зеркалирование требует дополнительного места для хранения точной копии данных. В отсутствие архивирования и систематизации измененных файлов может возникнуть неоптимальное использование ресурсов. Тем не менее, его эффективность в обеспечении быстрого доступа и отказоустойчивости делает его важным методом для обеспечения целостности и доступности данных.

Зеркалирование также находит применение в сфере веб-хостинга, особенно при создании зеркальных сайтов. Этот метод обеспечивает высокую степень надежности, так как при отказе одного сервера данные могут быть мгновенно переключены на зеркальный сервер без простоев в доступе к сайту.

Стоит отметить, что зеркалирование может быть не только физическим, но и географическим. Географическое зеркалирование предполагает хранение зеркальных копий данных в разных географически удаленных местах. Это увеличивает уровень отказоустойчивости, так как данные остаются доступными даже в случае крупных сбоев, таких как естественные бедствия или серьезные сетевые проблемы.

Несмотря на свою затратность в плане использования места, зеркалирование остается одним из наиболее надежных методов обеспечения доступности данных и уменьшения риска потери информации.

2 Обзор существующих решений для резервного копирования данных

Решения для резервного копирования можно условно разделить на три типа:

- онлайн сервисы резервного копирования;
- программное обеспечение синхронизации файлов;
- программное обеспечение для резервного копирования;

Однако, зачастую продукты можно отнести к двум или всем трем группам.

Онлайн (удаленные, облачные) сервисы резервного копирования – это сервисы, которые предоставляют пользователю систему для резервного копирования, хранения и восстановления данных. Зачастую работают с сервисами самого разработчика продукта.

Существуют такие подходы к периодичности копирования, как: ручное копирование, когда пользователь вручную запускает процесс копирования, запланированное копирование, когда процесс запускается по определенному графику, и копирование, основанное на событиях компьютерной системы, таких как завершение работы с базой данных, выключение и т.д. Онлайн системы резервного копирования обычно создаются для клиентской программы, которая выполняется по заданному графику. Некоторые работают раз в день, обычно ночью, когда компьютеры не используются. Другие, более новые облачные службы резервного копирования, работают непрерывно, фиксируя изменения в системах пользователей практически в реальном времени [23].

Онлайн система резервного копирования обычно собирает, сжимает, шифрует и передает данные на серверы удаленного поставщика услуг резервного копирования.

Представителями данного типа являются:

- iDrive;
- BackBlaze;
- Acronis True Image [6].

Программное обеспечение синхронизации файлов. Синхронизация файлов представляет собой процесс обеспечения актуальности компьютерных файлов в двух или более местоположениях согласно определенным правилам. Бывает односторонняя и двусторонняя синхронизация. При односторонней синхронизации файлы из источника копируются в одно или несколько местоположений. В двусторонней синхронизации файлов обновленные файлы копируются в обоих направлениях, обычно с целью поддержания полной идентичности двух местоположений друг к другу. То есть такое программное обеспечение не создает резервные копии в полном понимании этого слова. Они только синхронизируют актуальные данные в нескольких местах [7].

Представителями данного типа являются:

- Dropbox;
- Google Drive;
- Яндекс Диск;
- iCloud [6].

Программное обеспечение для резервного копирования. Ориентируется на работу локально или в рабочей сети организации. Некоторые продукты нацелены на работу с томами или физическими дисками. Другие позволяют сохранять отдельные папки и файлы. В большинстве своем поддерживают синхронизацию файлов и создание резервных копий с контролем версий.

Представителями данного типа являются:

- Amanda;
- Rsync и основанные на нем графические реализации;
- FreeFileSync;
- Comodo Backup [6].

Основными возможностями любого продукта являются:

- создание резервных копий с использованием методов резервного копирования или без (синхронизация);
- контроль версий;
- сжатие данных;
- шифрование при передаче и хранении данных;
- поддержка целостности копий и обнаружение ошибок;
- анализ файлов и передача только измененных блоков файлов [8].

Некоторые организации даже предлагают отправить им жесткий диск для создания первоначальной полной копии данных. Такой подход удобен при наличии больших объемов данных, которые не составляют какой-либо тайны.

Т.к. разрабатываемая в данной работе программа взаимодействует с Яндекс Диск, то рассмотрим настольную реализацию данного приложения.

2.1 Обзор Яндекс.Диск

Программа доступна бесплатно на официальном сайте. Устанавливается посредством стандартного инсталлятора Windows, занимает 5-10 секунд.

При запуске программы аккаунт подтянулся автоматически из браузера. Пользователя встречает главное окно программы (рис. 5).

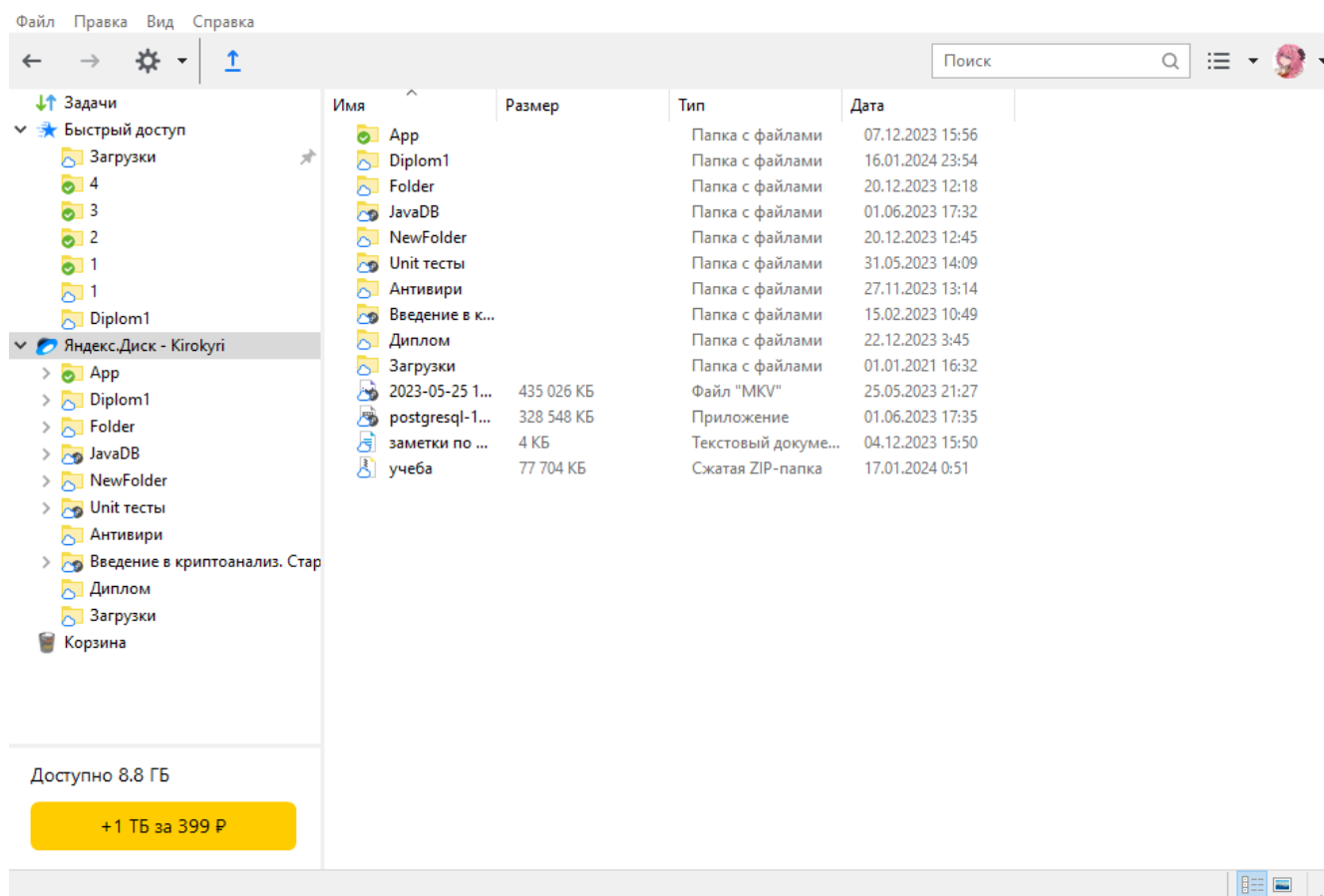


Рисунок 5 – Главное окно программы

В левой части находится компактный проводник, в котором отображаются только папки. Папку можно развернуть, нажав на стрелку слева, и увидеть подпапки. Если же выбрать папку, то на большой панели справа появится ее полное содержимое.

Данное окно поддерживает базовый функционал Яндекс Диска – возможность загружать или скачивать файлы, перетаскивая их из локального проводника в окно и наоборот, удалять файлы в хранилище. Также это можно делать из контекстного меню при нажатии правой кнопки мыши.

По умолчанию программа имеет одну папку для синхронизации (рис. 6).

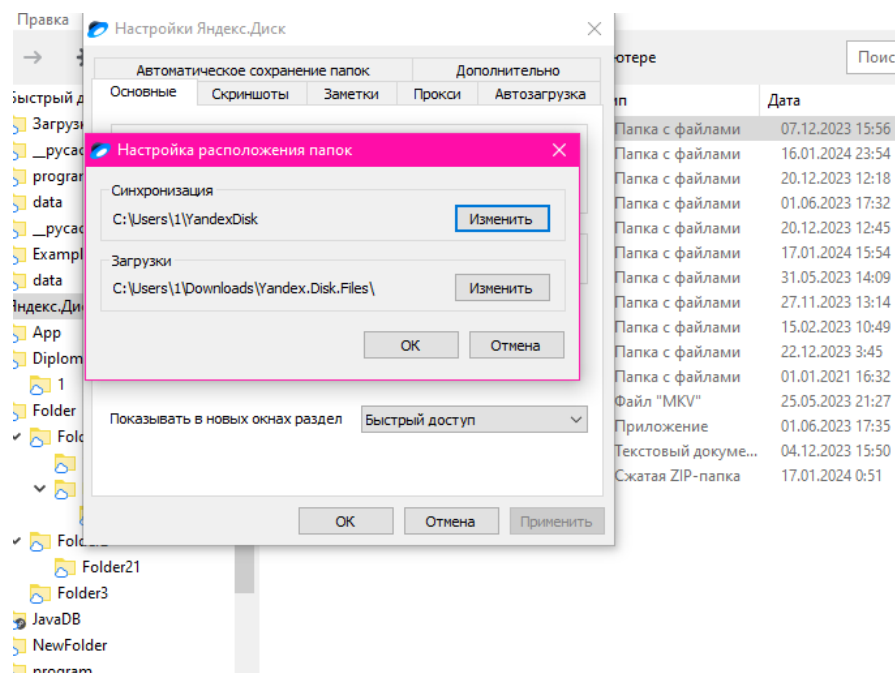


Рисунок 6 – Настройки

Любые файлы и папки, помещенные в данную папку, автоматически загружаются в хранилище в корень диска и синхронизируются. Синхронизация всегда двухсторонняя. Находящиеся в данной папке файлы имеют удобную индикацию статуса синхронизации в виде зеленой галочки, добавляющейся на иконку файла (рис. 7).

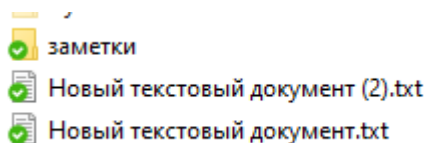


Рисунок 7 – Индикация статуса синхронизации

Программа имеет довольно низкую скорость загрузки, как показывает сама программа, скорость редко превышает 1 МБ/с. Скорость скачивания при этом удовлетворительная.

Программа также имеет возможность добавления любых других папок для синхронизации (рис. 8), помимо единственной доступной (рис. 6), однако для этого требуется оплатить подписку.

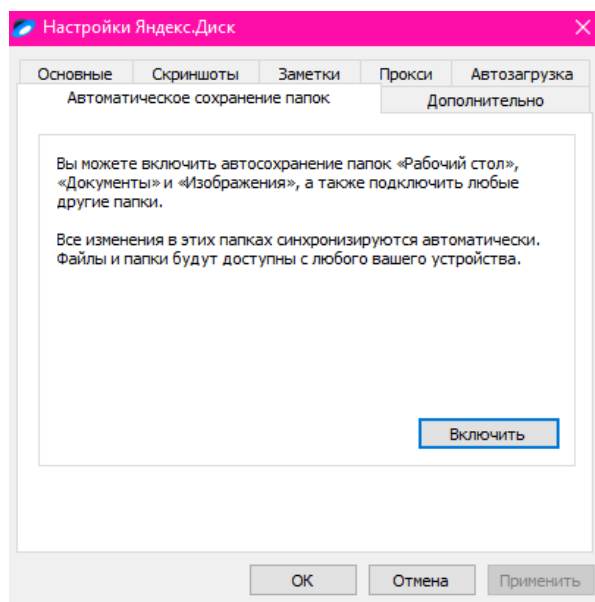


Рисунок 8 – Подключение дополнительных папок

В целом программа удобна для домашнего использования неподготовленным пользователем и справляется со своими функциями.

3 Программная реализация

В результате проделанной работы была разработана и реализована программа на языке Python с использованием среды разработки Visual Studio Code.

Программа собрана в исполняемый .exe файл при помощи модуля pyinstaller. Для своей работы требует дополнительные файлы, находящиеся в папке data, в том же каталоге, что и программа. Т.е. пользователю достаточно разархивировать программу и запустить.

Программа предоставляет интерфейс для работы с облачным хранилищем Яндекс.Диск (далее облако) и инструмент создания и восстановления резервных копий данных с автоматическим тихим режимом выполнения копирования.

В качестве основных библиотек используются PyQt5 – для реализации интерфейса и yadisk – для работы с облаком. В качестве вспомогательных библиотек используются sys – для обработки закрытия программы, tempfile – для работы с временными папками, os – для навигации в файловой системе, zipfile – для работы с zip архивами, hashlib – для функции вычисления хэш-суммы sha256 и datetime – для работы с форматом даты и времени.

При запуске программы пользователь видит окно аутентификации, в котором предлагается перейти по ссылке, аутентифицироваться в сервисе яндекс, а затем ввести код, который появится после аутентификации на яндекс (рис. 9-10).

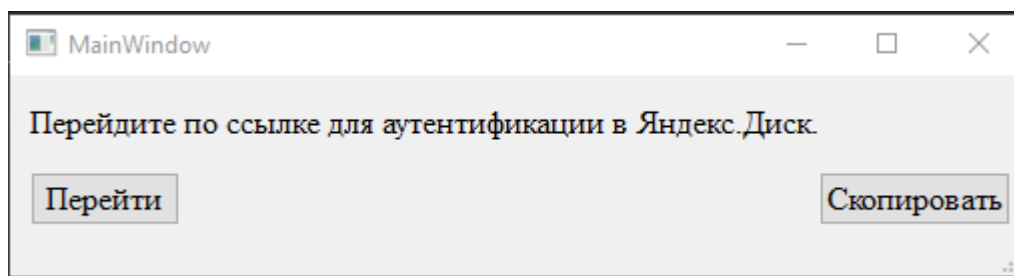


Рисунок 9 – Окно аутентификации

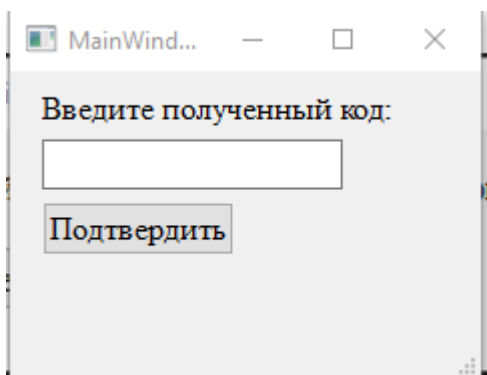


Рисунок 10 - Окно аутентификации

Программу можно условно поделить на две подсистемы: подсистема работы с облаком и подсистема резервного копирования. По завершении аутентификации пользователь попадает на главное окно программы, которое и является репрезентацией подсистемы работы с облаком.

3.1 Подсистема работы с облаком

Подсистема работы с облаком предоставляет функционал, аналогичный десктоп приложению Яндекс.Диск и позволяет просматривать, скачивать и загружать файлы, удалять файлы, создавать папки. Файлы отображаются в удобном файловом древе (рис. 11).

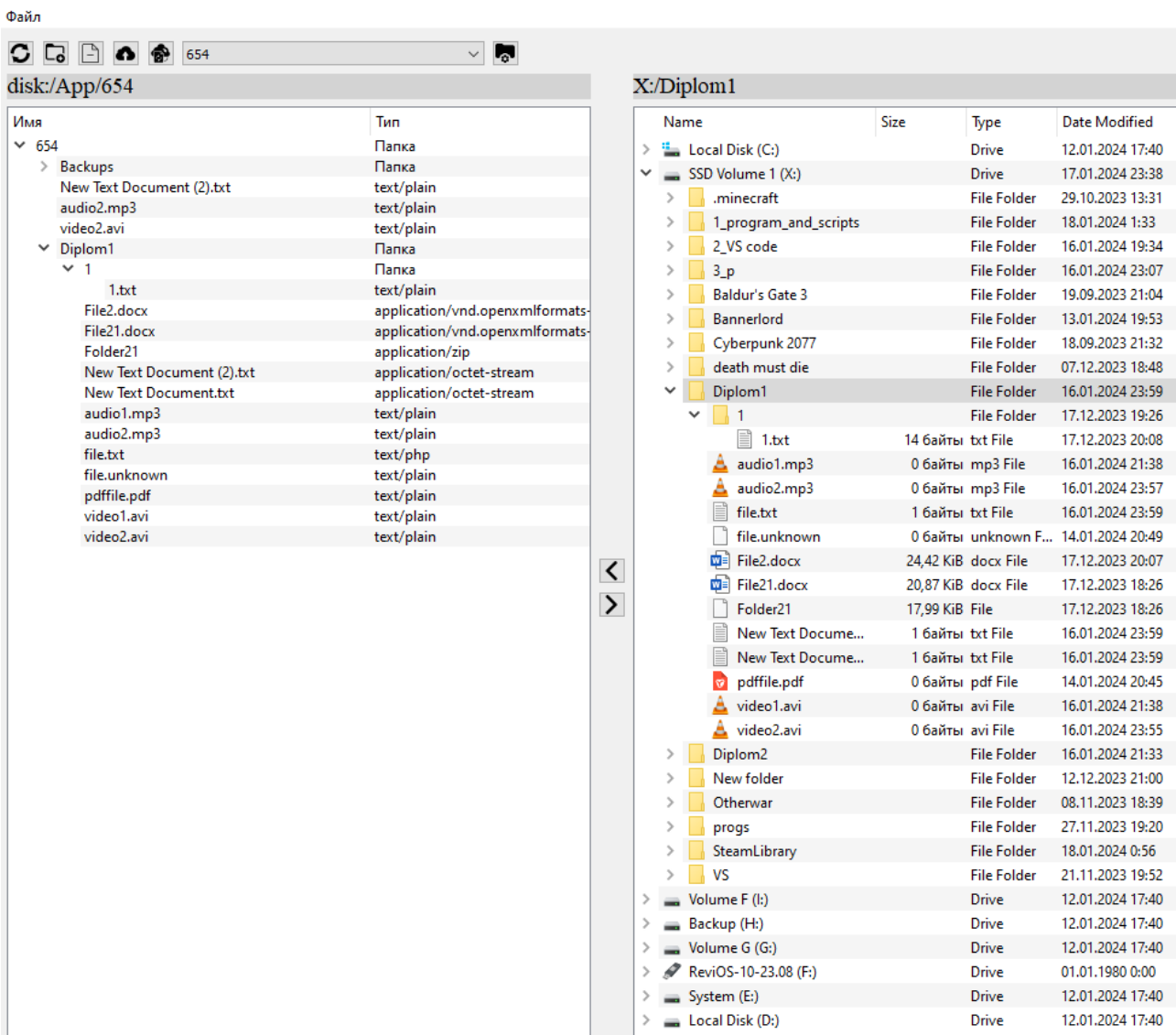


Рисунок 11 – Основное окно программы

Основное окно разделено на две панели: левая панель – соответствует файловой структуре облака, правая панель – файловой структуре локальной системы. Между панелями находятся кнопки скачивания и загрузки файлов. Над панелями находятся строки, отображающие путь выбранного в панели файла. Выше находится панель с кнопками управления программой: перезагрузка файлового древа облака, создание папки, удаление файлов, создание резервной копии, восстановление резервной копии. После всех кнопок расположен выпадающий список, в котором можно выбрать рабочую папку для работы с рабочей папкой или

корень облака для работы со всем облаком. И в конце – кнопка для настройки рабочей папки.

3.2 Подсистема резервного копирования

Подсистема резервного копирования предоставляет инструменты резервного копирования и реализует метод инкрементального резервного копирования. Также есть возможность автоматического периодического запуска резервного копирования, которая реализуется при помощи .bat скрипта запуска программы с параметром /auto_backup и планировщика задач.

Главным механизмом данной подсистемы являются “рабочие папки”. Рабочая папка – это отдельная папка, с которой пользователь может работать как с любой другой на облаке. При выборе рабочей папки, на экране отображается файловое древо именно этой папки. Для добавления и удаления рабочих папок вызывается отдельное окно, в котором отображаются все имеющиеся рабочие папки и соответствующие кнопки для их добавления и удаления (рис.12).

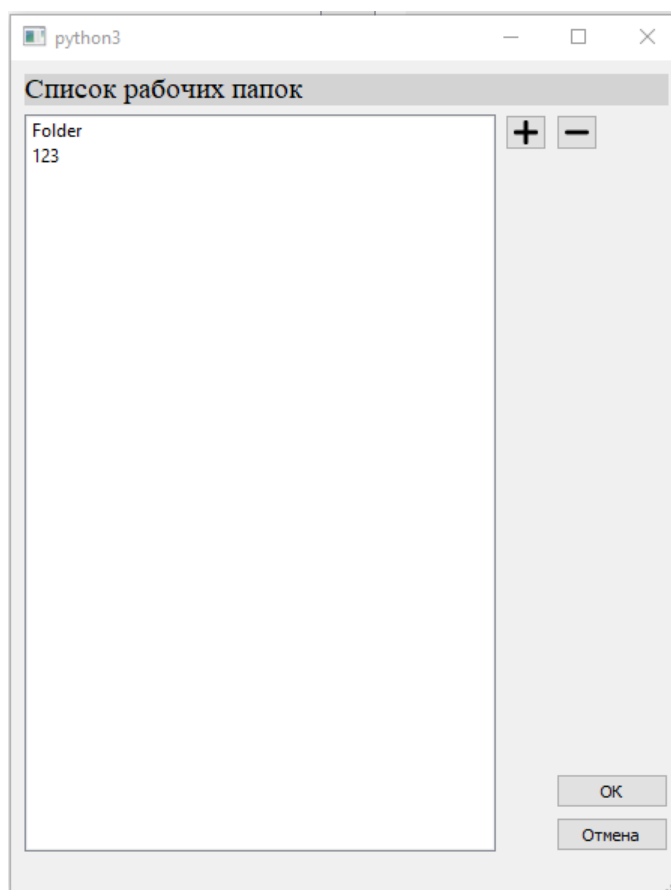


Рисунок 12 – Окно настройки рабочих папок

Цель данного механизма состоит в том, что рабочая папка запоминает все загруженные в нее файлы и позволяет создавать резервные копии этих файлов. Сама рабочая папка считается полной резервной копией. Нажатием на кнопку создания резервной копии, пользователь инициирует процесс проверки файлов и создания инкрементальной копии.

Восстановление резервной копии также применяется нажатием соответствующей кнопки. При ее нажатии вызывается новое окно, в котором пользователю предлагается выбрать, какую копию он хочет восстановить. Он может выбрать как полную копию, так и любую инкрементальную копию (рис.13).

Создание и восстановление резервной копии выполняется только для выбранной рабочей папки.

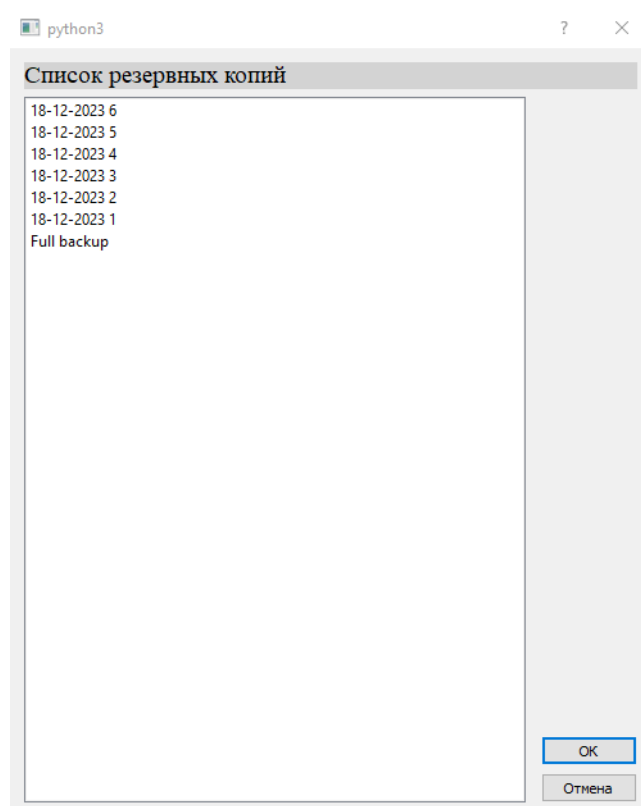


Рисунок 13 – Окно выбора резервной копии для восстановления

Для рабочей папки можно выполнить настройку. В окне настройки (рис. 14) представлены 4 предустановленных набора форматов и один набор, который формируется путем сканирования папок, загруженных в рабочую папку, и составления списка форматов, которых нет в предустановленных наборах.

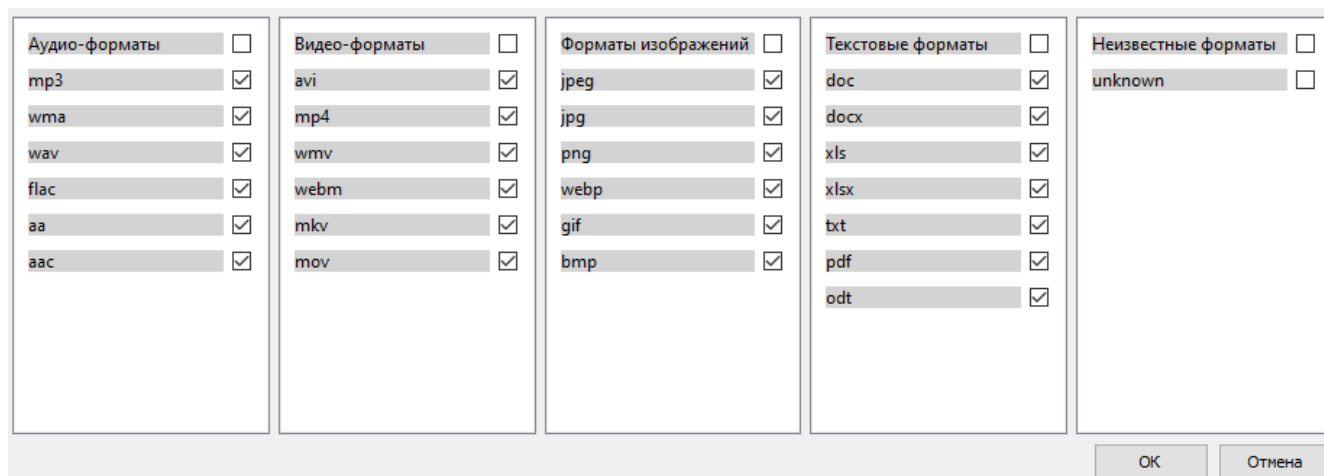


Рисунок 14 – Окно настройки рабочей папки

Суть данной настройки состоит в том, что при создании резервной копии программа также ищет в папках, добавленных в рабочую папку, файлы, имеющие выбранные расширения, но не добавленные в рабочую папку. Т.е. программа автоматически загружает на облако новые файлы выбранных типов, выполняет одностороннюю синхронизацию.

4 Демонстрация работы программы

Проведем тестирование программы и рассмотрим некоторые примеры работы в ней.

4.1 Аутентификация

Запуская программу, попадаем на экран аутентификации (рис. 15).

Аутентификация в сервисах яндекс реализована с использованием OAuth2.0 и результатом аутентификации является полученный OAuth-токен (далее токен), позволяющий приложению работать с сервисами яндекс от имени пользователя, используя полученные разрешения. Для тестирования приложения яндекс позволяет получить отладочный токен, который неизменен со временем и дает приложению доступ к аккаунту автора приложения, но другим пользователем нужно получать свой токен [9].

Для данного приложения требуются следующие разрешения:

- доступ к папке приложения на Диске;
- доступ к информации о Диске;
- чтение всего Диска;
- запись в любом месте на Диске;

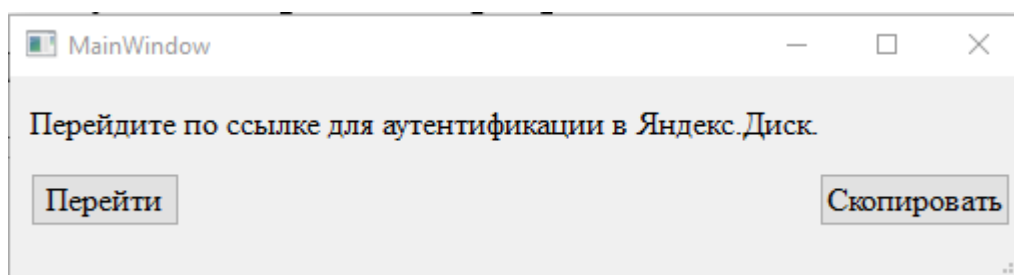


Рисунок 15 – Переход по ссылке

При переходе по ссылке, она открывается в браузере по-умолчанию. Также есть возможность скопировать ссылку и открыть вручную. Ссылка ведет на форму аутентификации Яндекс (рис. 16).

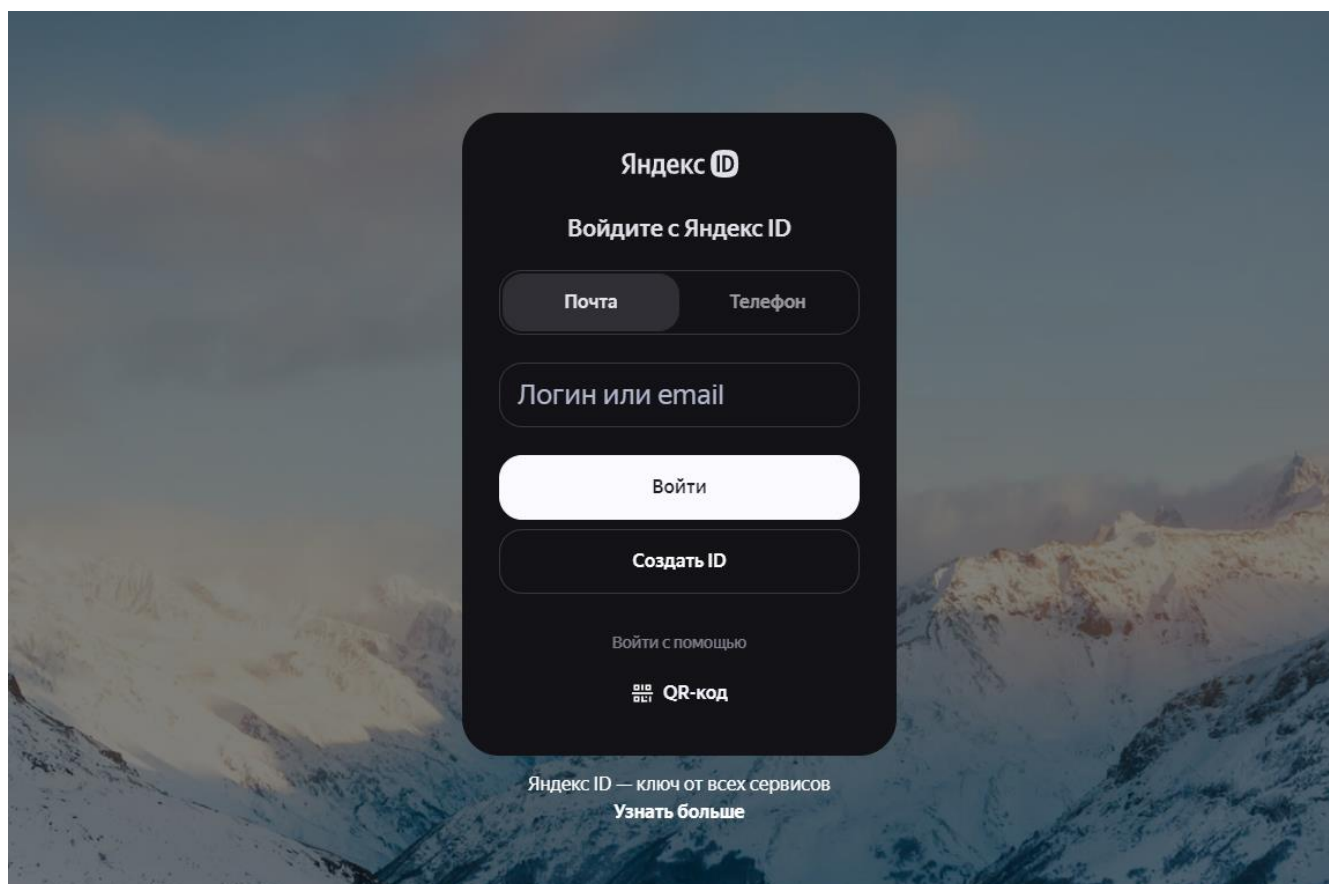


Рисунок 16 – Форма аутентификации

После входа в аккаунт, Яндекс предлагает дать разрешения приложению (рис. 17).

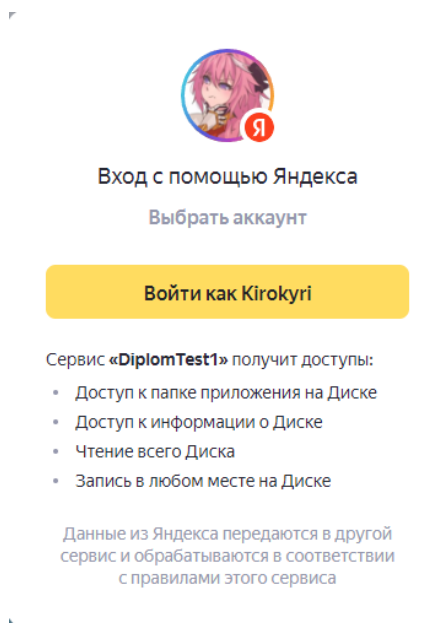
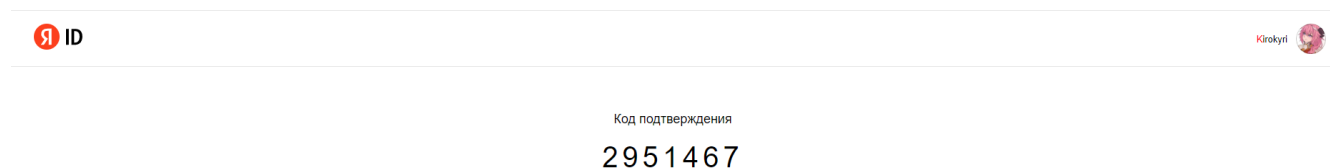


Рисунок 17 – Разрешения для приложения

После этого нас перенаправляет на форму, в которой записан код подтверждения (рис. 18).



Я ID

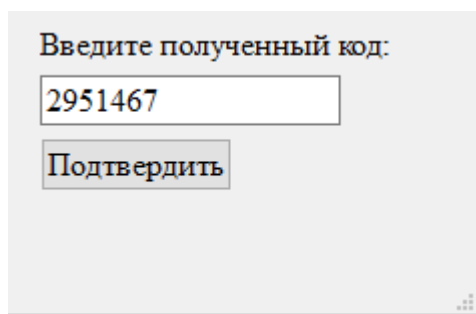
Kirokyt

Код подтверждения

2951467

Рисунок 18 – Форма с кодом подтверждения

Вводим код и попадаем в главное окно программы.



Введите полученный код:

2951467

Подтвердить

Рисунок 19 – Окно ввода кода подтверждения

4.2 Подсистема работы с облаком

В главном окне в левой панели видим корень облака, а в правой панели локальную файловую систему (рис. 20).

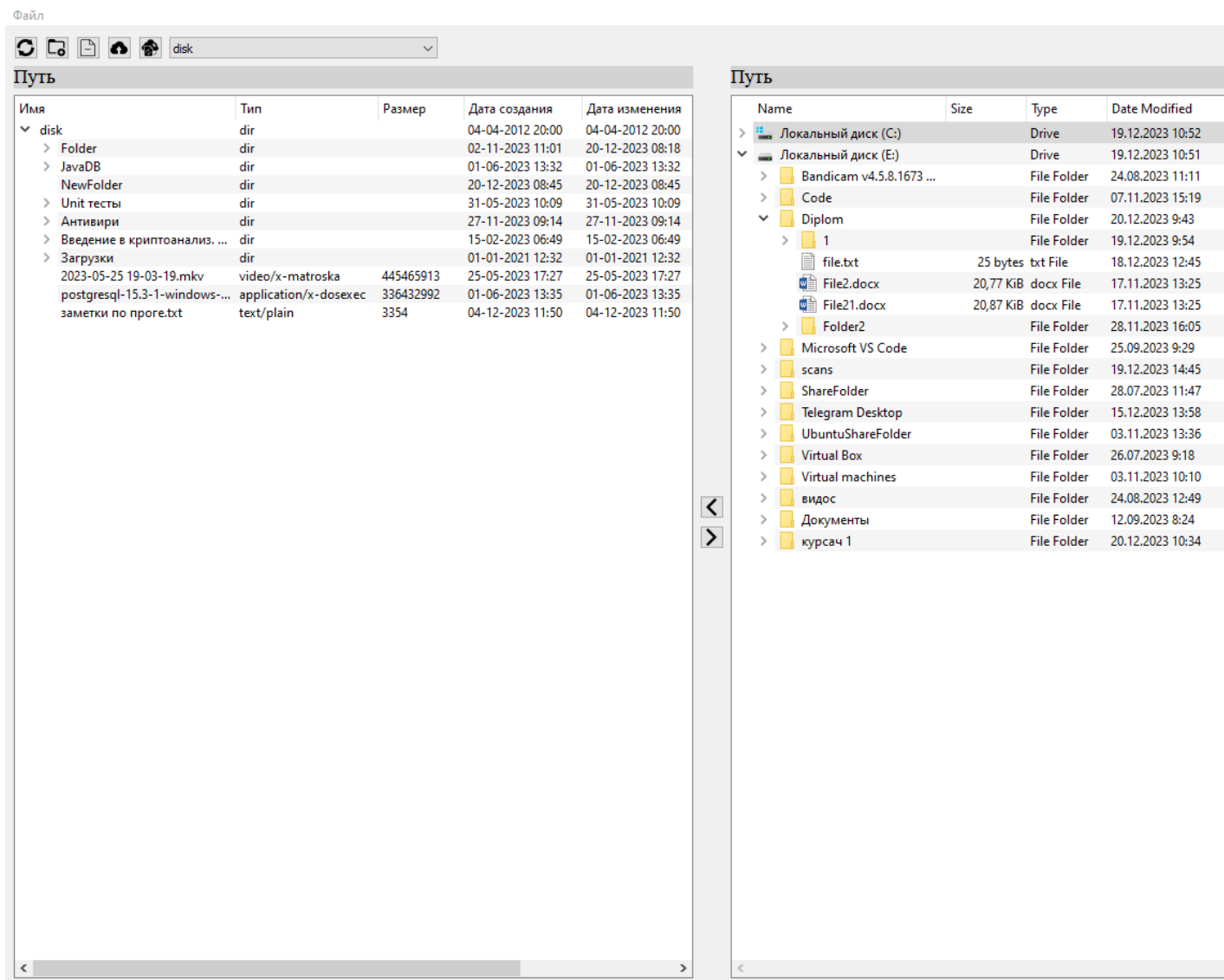


Рисунок 20 – Главное окно

Для начала создадим папку NewFolder в корне облака и попробуем загрузить туда некоторые файлы. Выбираем корень, папку disk, и нажимаем соответствующую кнопку. Вписываем имя новой папки и нажимаем ОК (рис. 21).

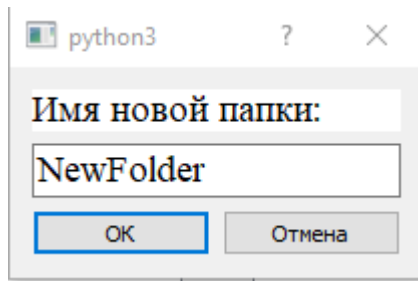


Рисунок 21 – Выбор имени новой папки

В древе появилась новая папка с заданным именем (рис. 22). Внутри программы, при создании папки, конструируется новый путь на облаке, включающий в себя введенное название, создается папка по новому пути, а затем обновляется отображение древа.

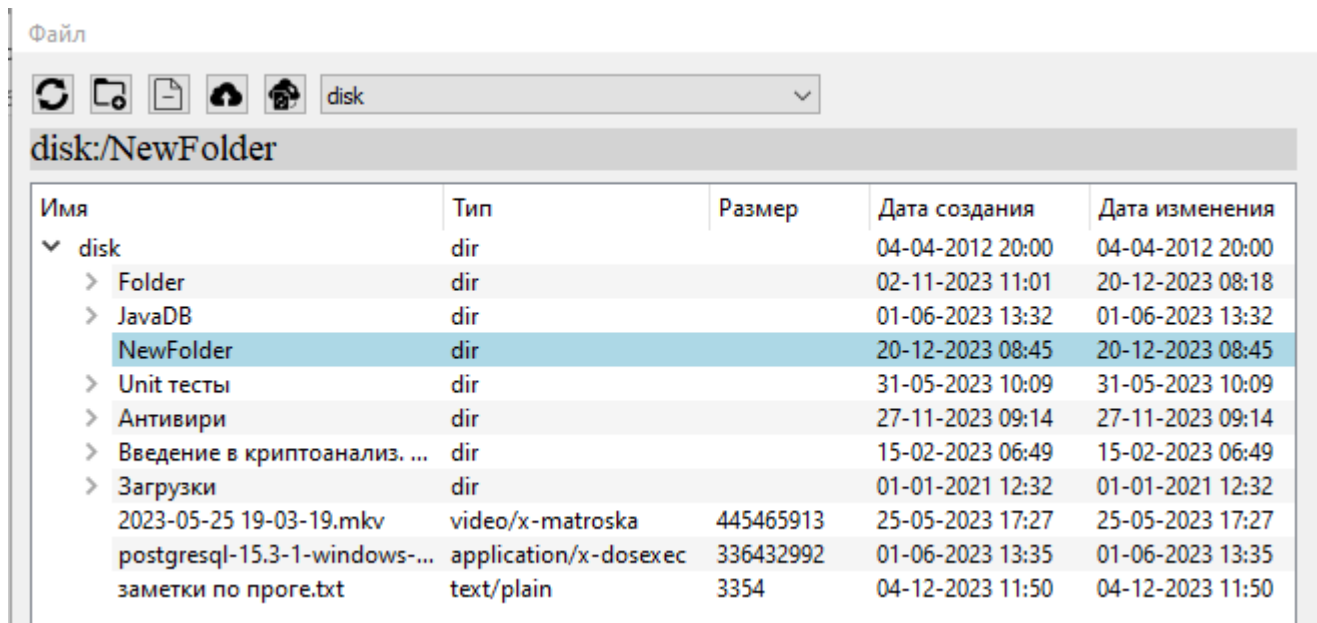


Рисунок 22 – Созданная папка

Далее загрузим в нее некоторые файлы и папки (рис. 23). В левой панели выберем созданную папку, а в правой – папку, в которой находятся файл и вложенная папка с файлом, а также просто файл. В строчках над панелями также отображаются выбранные пути.

Видим, что в нее успешно загрузились файлы, причем они повторяют иерархию в локальной файловой системе.

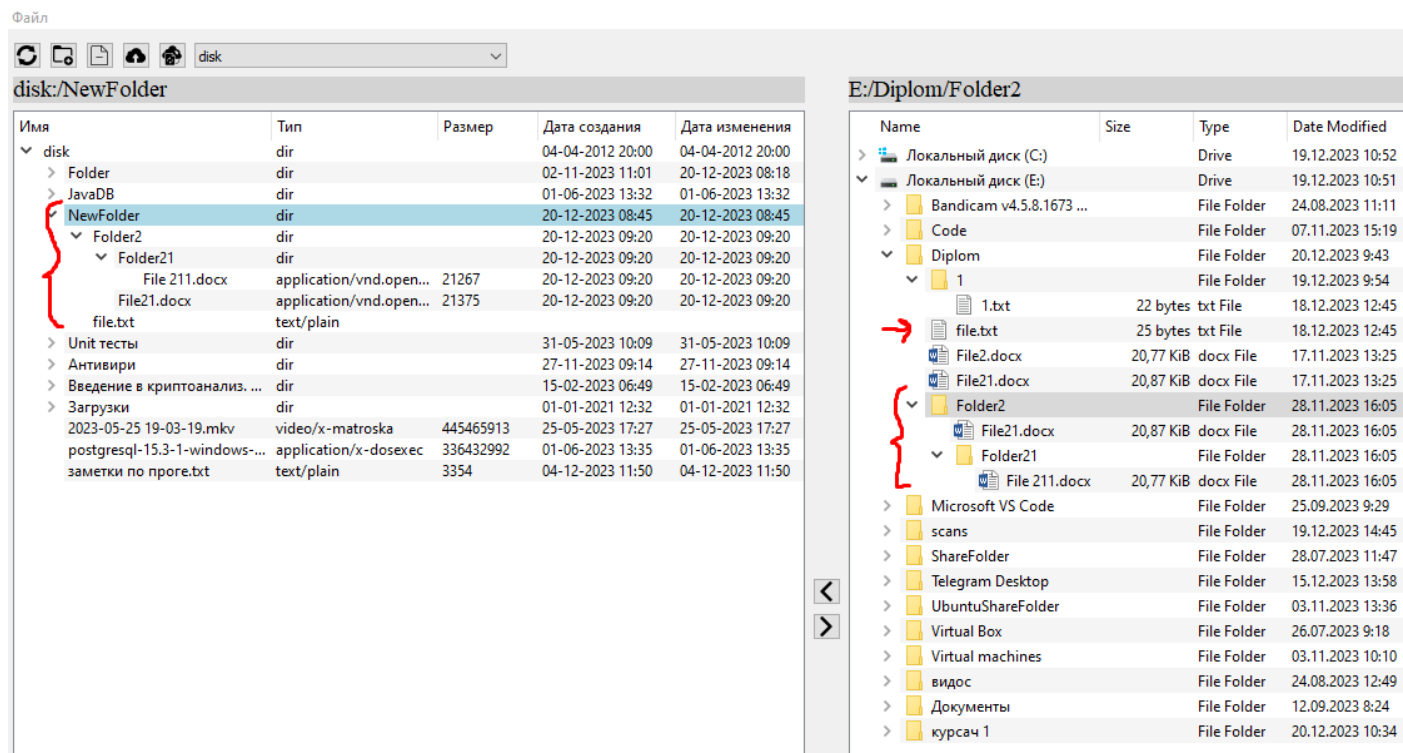


Рисунок 23 – Загрузка файлов и папок

Можно убедиться, что в веб версии облака файлы и папка также появились (рис. 24).

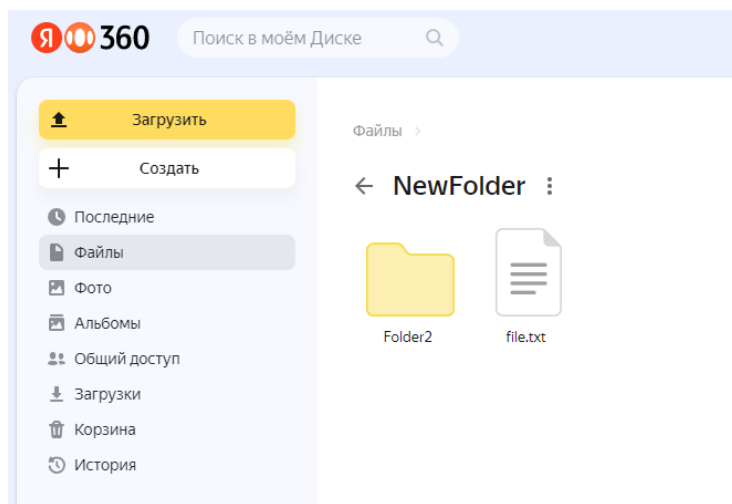


Рисунок 24 – Проверка наличия загруженных файлов

Теперь скачаем созданную папку с облака в корень диска E: (рис. 25). С облака папки обычно скачиваются в .zip архиве, поэтому скачанные папки разархивируются, а архив удаляется.

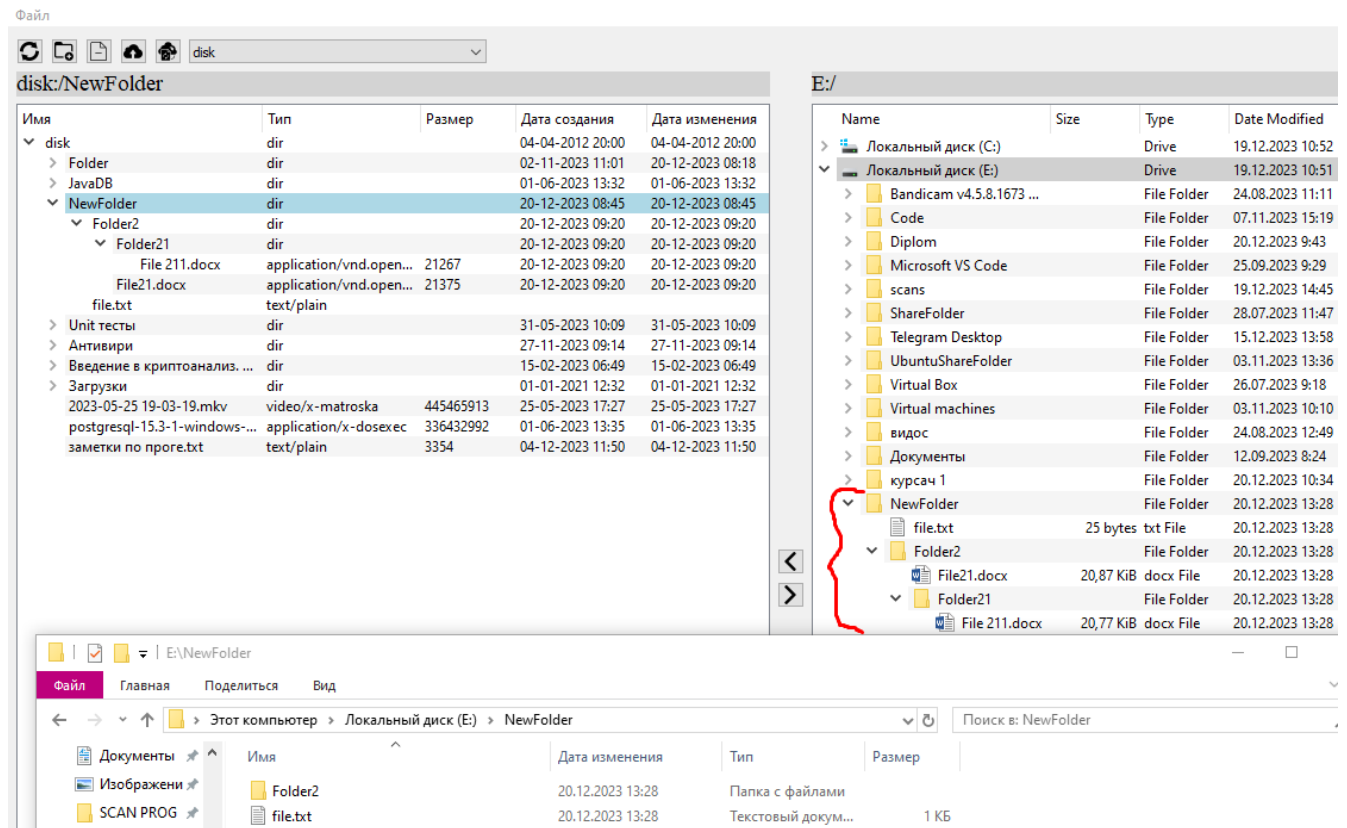


Рисунок 25 – Скачивание папки с файлами

4.3 Подсистема резервного копирования

Далее рассмотрим работу подсистемы резервного копирования. Данная подсистема реализует метод инкрементального резервного копирования. Она работает с использованием «рабочих папок». Пользователь создает рабочую папку, выбирает ее и загружает в нее свои файлы. В качестве полной копии выступает сама рабочая папка со всеми загруженными в нее файлами. После загрузки файлов появляется возможность создавать инкрементальные копии загруженных файлов. Суть инкрементальной копии в том, что в резервную копию добавляются только файлы, которые имеют изменения в сравнении с этим же файлом в прошлых копиях. Рабочих папок может быть много и для каждой из них существуют свои отдельные копии.

При выявлении изменений в файлах программа руководствуется хэш-суммами этих файлов. Хэш-суммы копий файлов на облаке вычисляются автоматически сервером облака, а хэш-суммы локальных файлов вычисляются в программе. В качестве алгоритма используется sha256, выбранный ввиду того, что на облаке также используется этот метод [10]. Таким образом хэш-суммы одного и того же файла на облаке и на локальном диске будут совпадать.

Создадим новую рабочую папку, загрузим в нее файлы и протестируем подсистему резервного копирования.

По выбору в меню «Файл» пункта «Создание и удаление рабочих папок» появляется окно для работы с рабочими папками. Данное окно поддерживает функции создания и удаления рабочих папок. Создадим папку с именем NewWorkFolder (рис. 26).

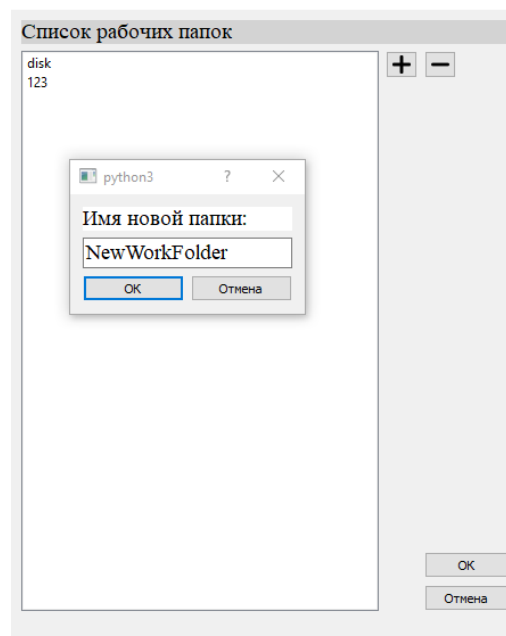


Рисунок 26 – Создание новой рабочей папки

После нажатия на ОК, новая рабочая папка появляется сверху в выпадающем списке. Выбираем ее и видим, пустую новую папку (рис. 27). Все рабочие папки создаются в папке приложения /App.

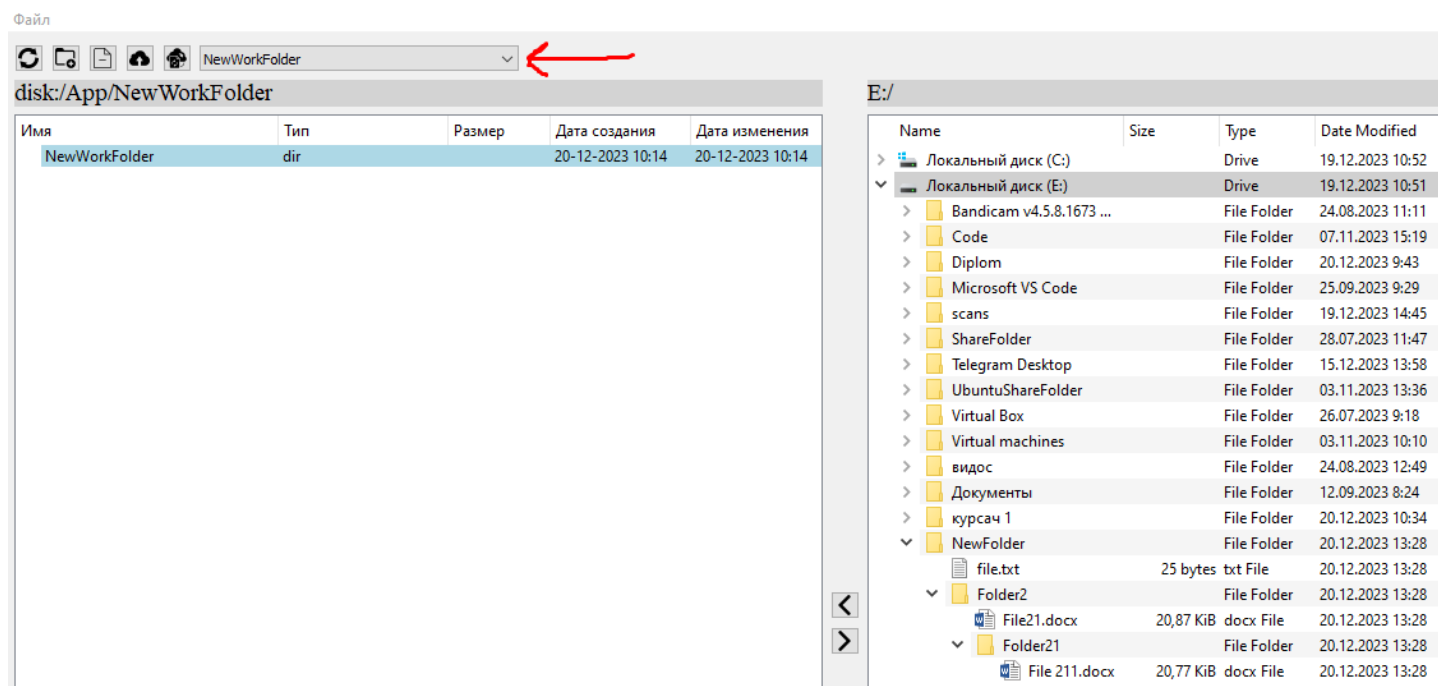


Рисунок 27 – Новая рабочая папка

Далее загрузим в нее три текстовых файла и создадим несколько резервных копий. При создании каждой копии будем в одном или нескольких файлах менять данные так, чтобы было понятно, какая копия восстановлена.

Перед загрузкой в каждый файл внесли число 0 и загрузили папку с этими файлами в рабочую папку (рис. 28).

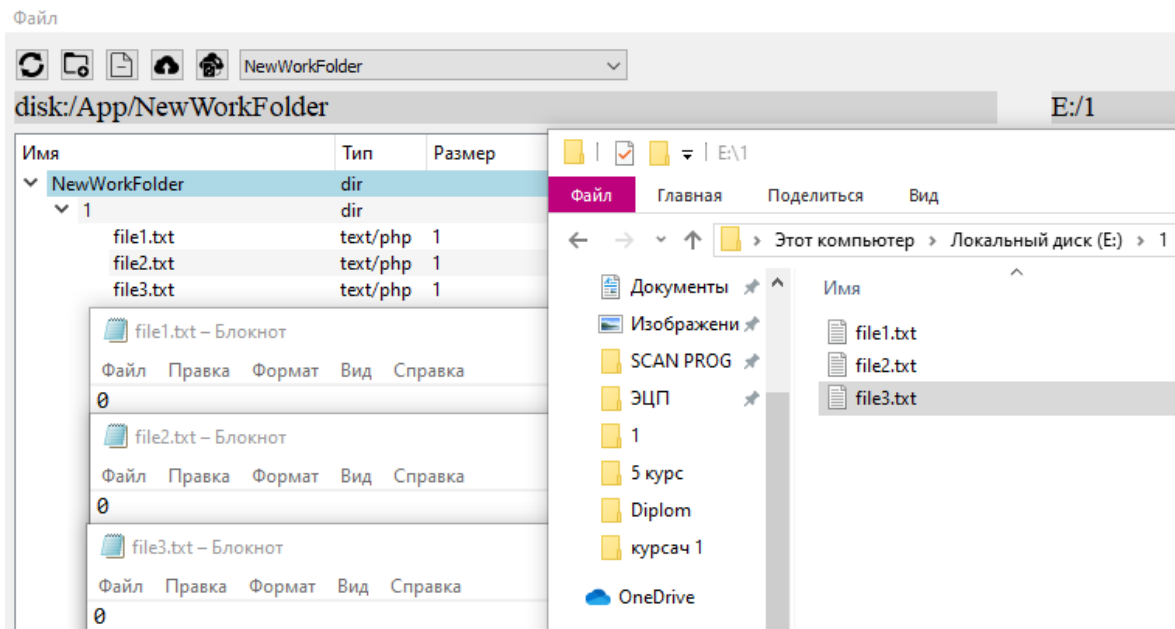


Рисунок 28 – Файлы перед загрузкой

Внесем в файлы 1 и 2 значение 1 и создадим первую копию. В списке копий появилась новая копия, в чем также можно убедиться, проверив веб версию облака (рис. 29).

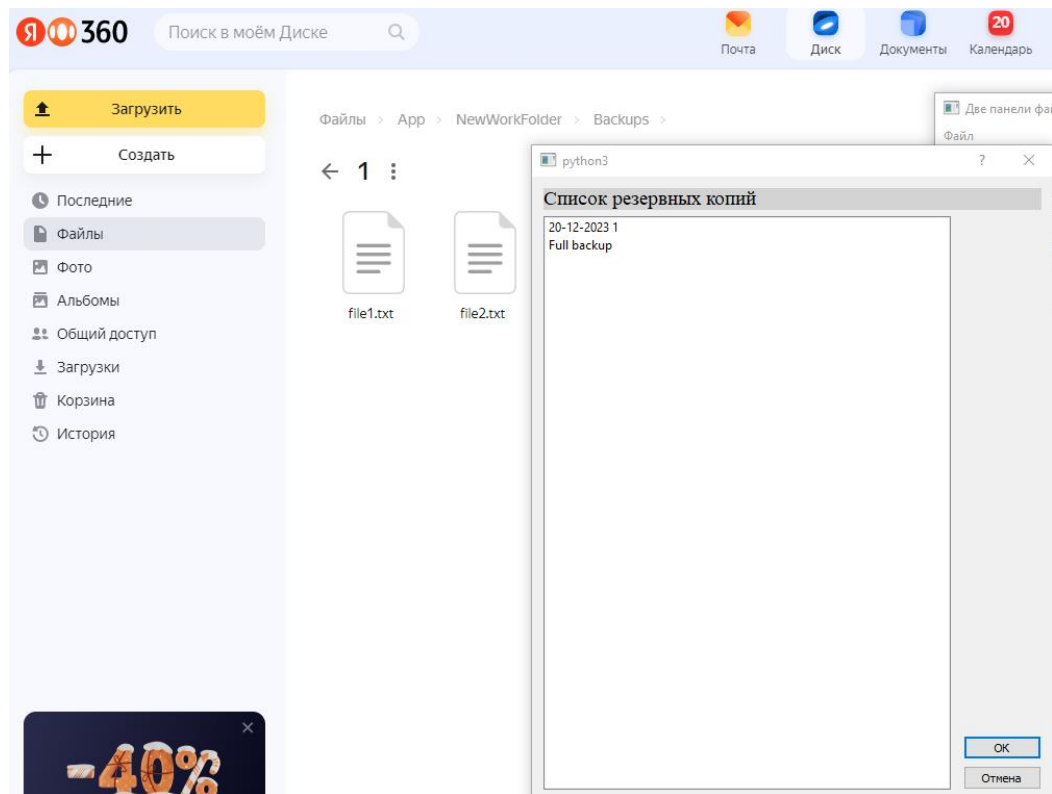


Рисунок 29 – Первая резервная копия

Меняя значения в файлах и создавая копии, получили пять инкрементальных копий (рис. 30). Значения в файлах соответствуют номеру копии, в которую он записан.

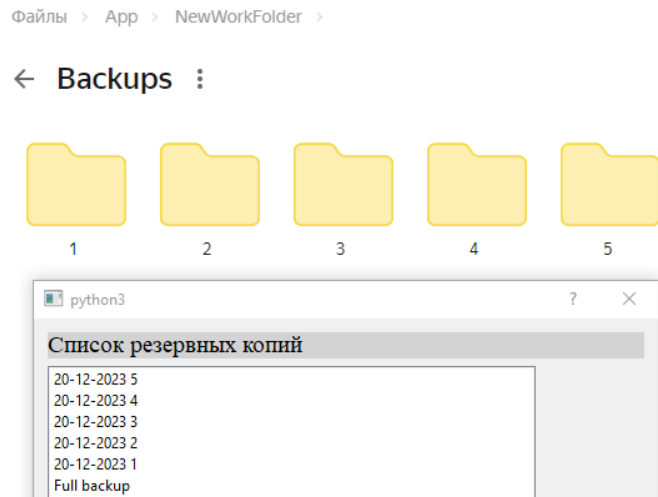


Рисунок 30 – Результат создания копий

Теперь приступим к восстановлению. Восстановим копию 3, в которой находятся файлы 2 и 3 со значениями 3. Ближайшая копия, в которой находится файл 1 – копия 2. Выбираем из списка копию 3, нажимаем ОК и проверяем файлы. Видим, что файлы 2 и 3 восстановились из выбранной копии 3, а файл 1 из копии 2.

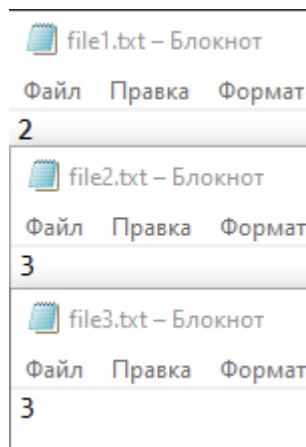


Рисунок 31 – Восстановленная копия 3

Восстановим копию 5, в которой все три файла имеют значение 5 (рис. 32).

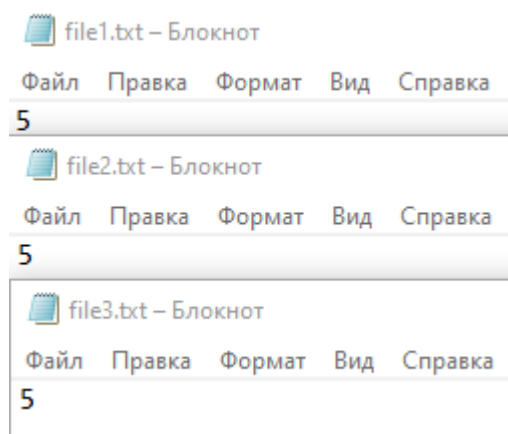


Рисунок 32 – Восстановленная копия 5

И наконец восстановим изначальную полную копию (рис. 33).

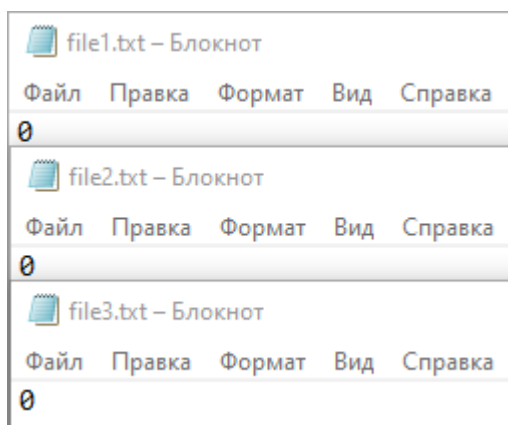


Рисунок 33 – Восстановленная полная копия

При закрытии программа сохраняет все данные о рабочих папках и резервных копиях в XML файл, а при запуске считывает их.

Далее протестируем механизм автоматического создания копий и добавления новых файлов. Для этого создадим новую рабочую папку, загрузим в нее папку с тремя файлами (рис. 34) и в настройках рабочей папки выберем все текстовые форматы и все найденные форматы (рис. 35). Видим, что в правой колонке отображается .bat формат.

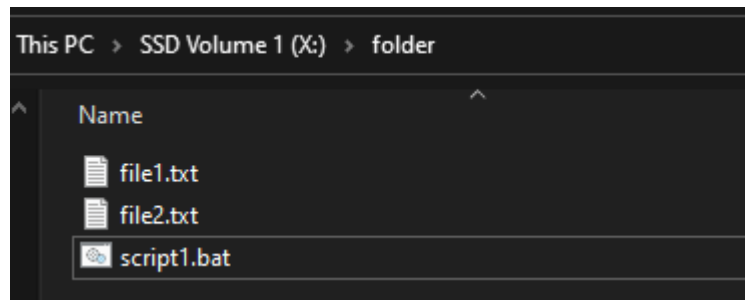


Рисунок 34 – Загружаемая папка с файлами

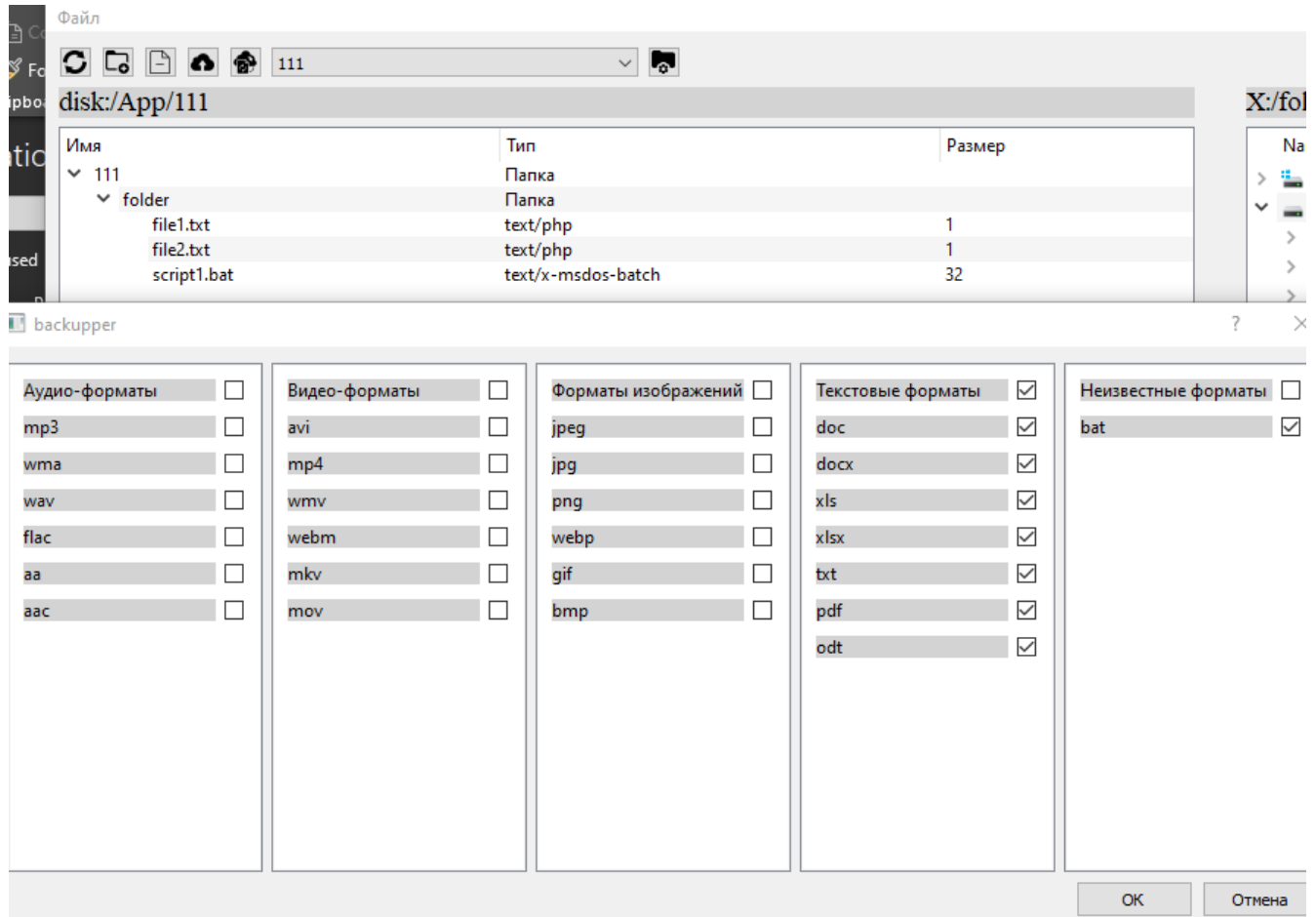


Рисунок 35 – Выбор форматов

Сохраним комбинацией клавиш Ctrl+S и закроем программу.

Запустим скрипт из папки с программой, который добавляет в планировщик задачу запуска программы. В планировщике появилась новая задача, которая запускает скрипт start.bat каждый день в 20:00 (рис. 36).

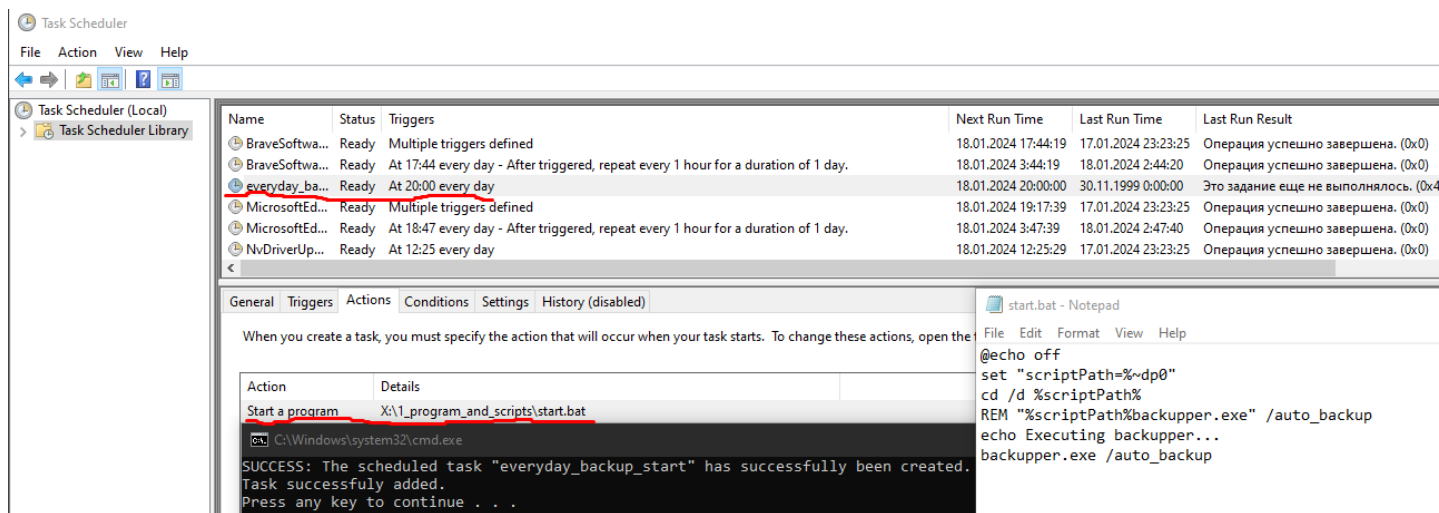


Рисунок 36 – Добавление задачи периодического резервного копирования

В целях тестирования будем запускать задачу вручную.

Изменим file1.txt и создадим новый файл docfile1.docx и выполним создание резервной копии. Видим, что появилась новая резервная копия с файлом file1.txt (рис. 37) и в рабочую папку на облаке был загружен новый файл docfile1.docx (рис. 38).

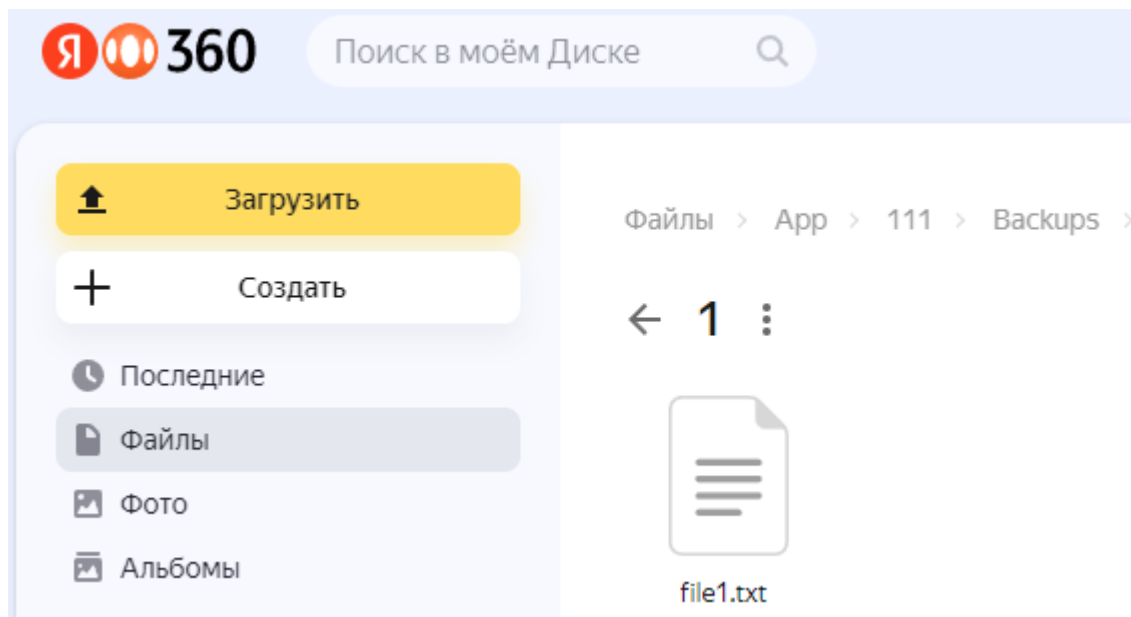


Рисунок 37 – Созданная копия

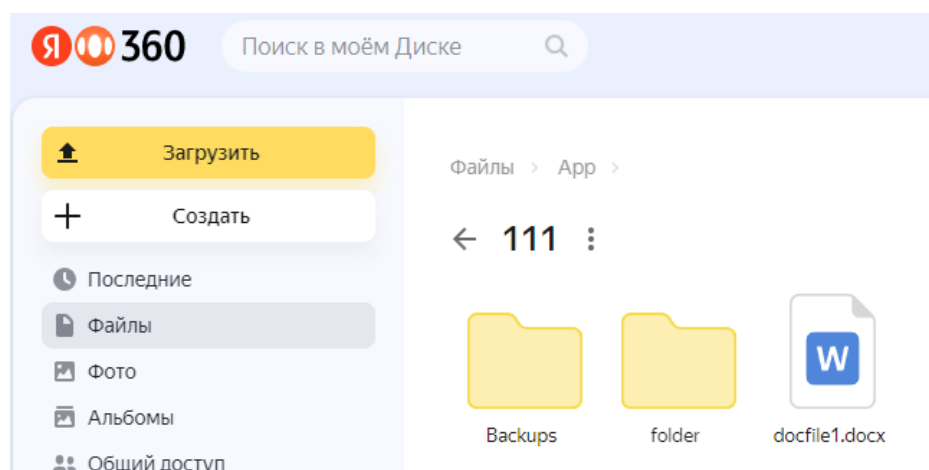


Рисунок 38 – Загружен новый файл

Теперь изменим файлы `docfile1.docx` и `script1.bat`, создадим новый файл `script2.bat` и выполним создание резервной копии. Аналогично предыдущей проверке была создана новая резервная копия, содержащая измененные файлы (рис. 39) и загружен новый файл (рис. 40).

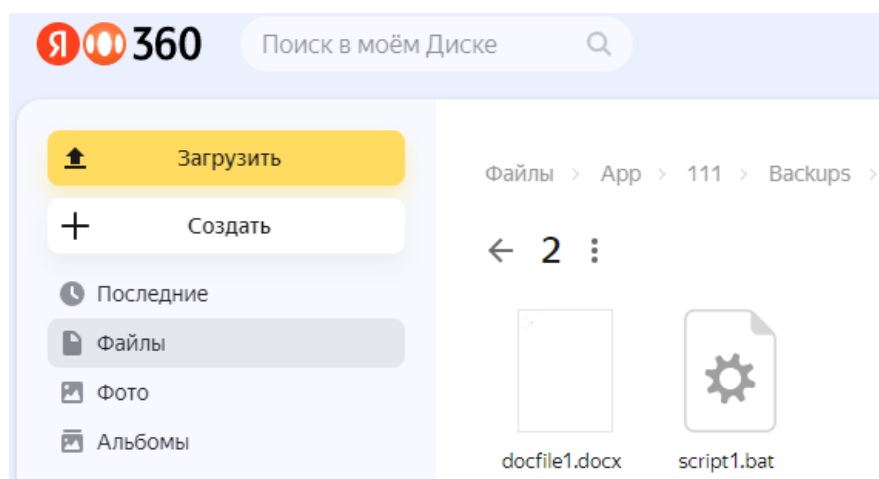


Рисунок 39 – Созданная копия

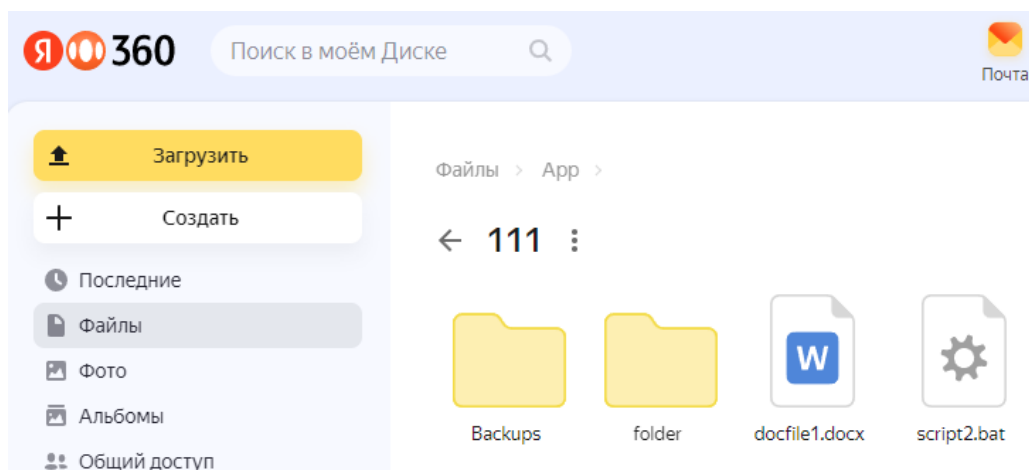


Рисунок 40 – Загружен новый файл

В заключение хочется отметить некоторые особенности программы:

- не требует установки дополнительных пакетов;
- простой и удобный интерфейс;
- односторонняя синхронизация файлов;
- контроль версий резервных копий.

Также программа совместима с Linux системами. Тестирование выполнялось на дистрибутиве Ubuntu 20.04.6 LTS Focal Fossa (рис. 41-42).

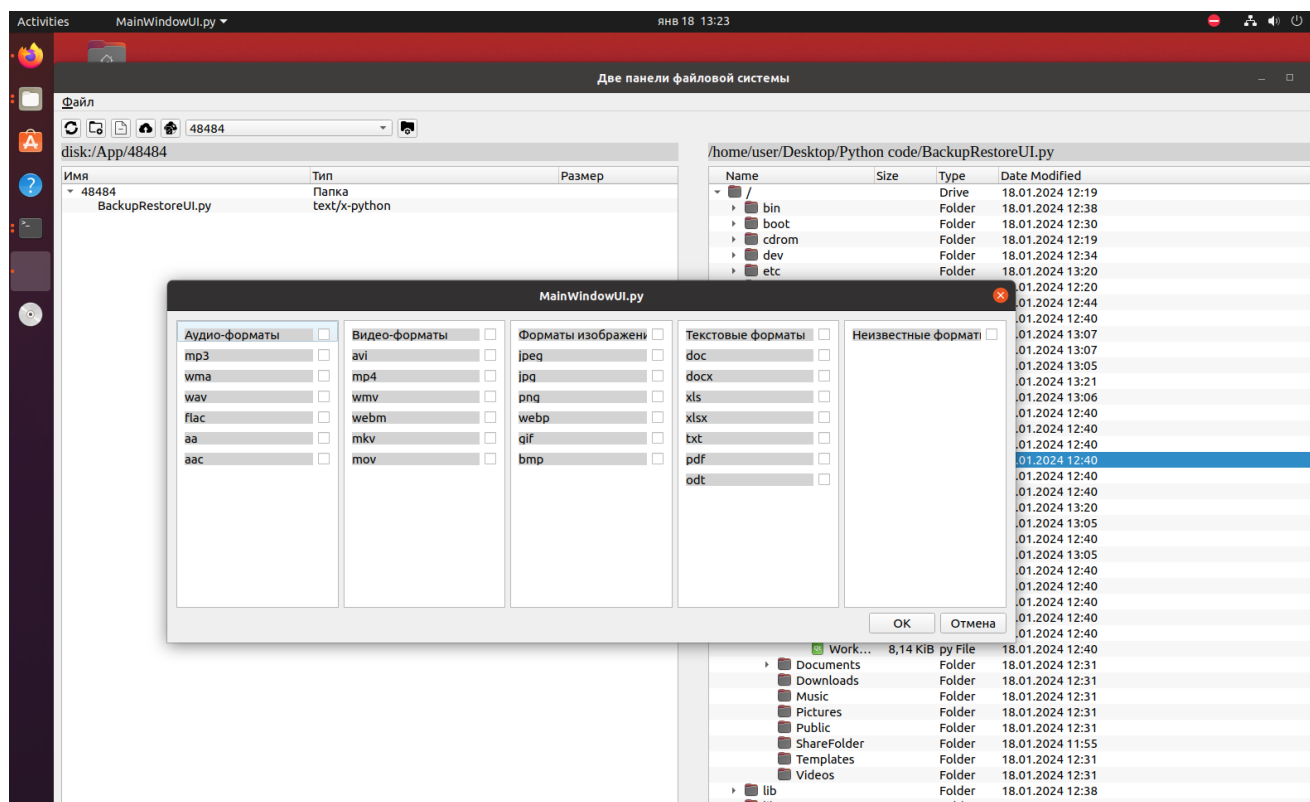


Рисунок 41 – Окно настройки рабочих папок

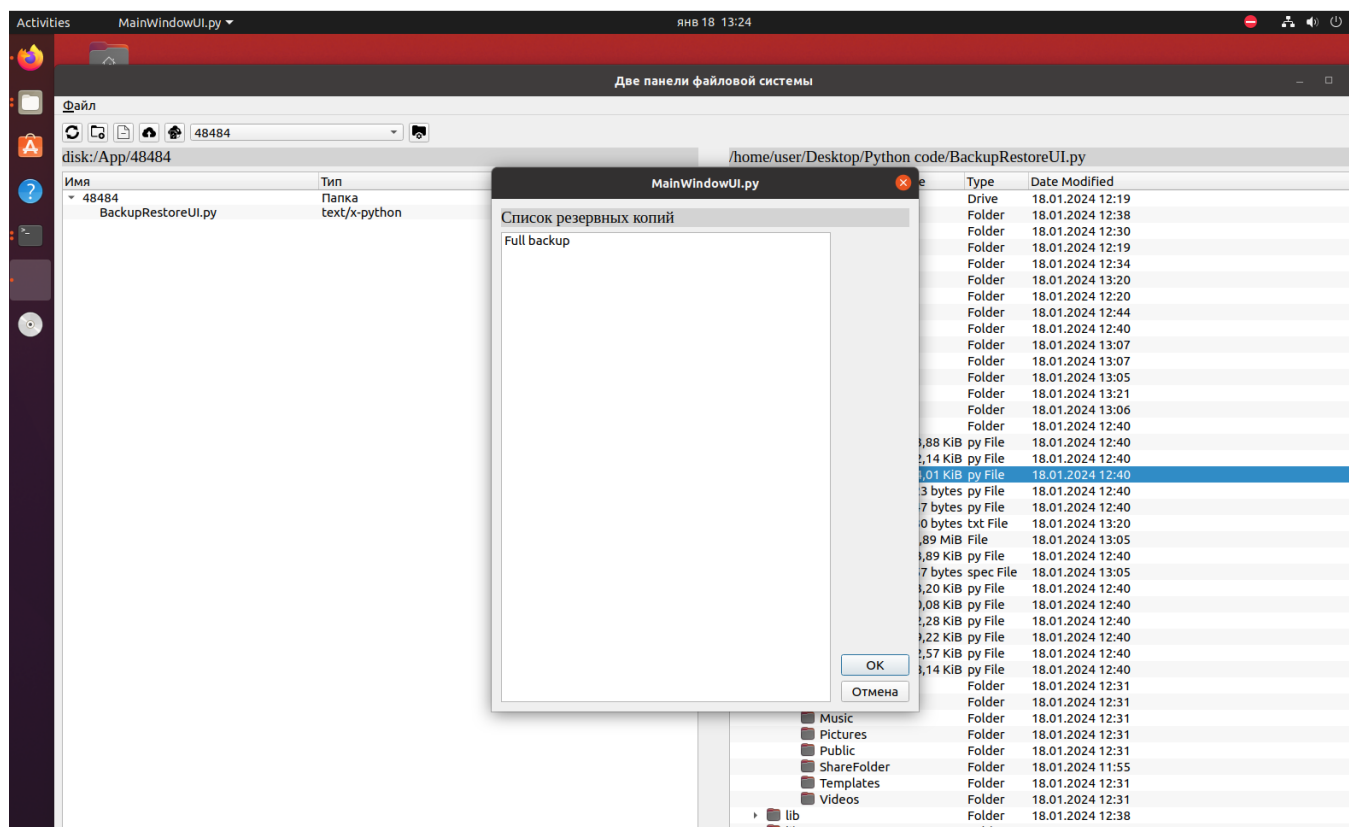


Рисунок 42 – Окно восстановления резервных копий

ЗАКЛЮЧЕНИЕ

Этот исследовательский процесс включал в себя анализ различных методов резервного копирования данных, в их числе исследование преимуществ и недостатков каждого из них.

В ходе работы были достигнуты следующие результаты:

- 1) изучены способы создания резервных копий;
- 2) реализована программа, ориентированная на домашнее использование, которая:
 - 2.1) позволяет работать с облачным хранилищем Яндекс Диск;
 - 2.2) позволяет создавать и восстанавливать резервные копии
 - 2.3) позволяет задать форматы сохраняемых файлов и автоматически загружать их в хранилище;
 - 2.3) имеет возможность создания резервных копий по расписанию;
 - 2.4) имеет графический интерфейс;
 - 2.5) совместима с системами Windows 10 и Linux Ubuntu 22.04.

Следует отметить, что данная работа сконцентрирована на практической реализации и применении изученных методов в реальных сценариях. В ходе исследования мы не только рассмотрели теоретические аспекты, но и успешно применили полученные знания для создания и тестирования программного продукта.

Таким образом, основные поставленные задачи были решены, и цель данного исследования успешно достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Системы резервного копирования [Электронный ресурс] / URL: <https://www.jetinfo.ru/sistemy-rezervnogo-kopirovaniya/> (дата обращения: 13.09.2023). – Загл. с экрана. – Яз. Рус.

2 Backup Types Explained: Full, Incremental, Differential, Synthetic, and Forever-Incremental [Электронный ресурс] / URL: <https://www.nakivo.com/blog/backup-types-explained-full-incremental-differential-synthetic-and-forever-incremental/> (дата обращения: 15.09.2023). – Загл. с экрана. – Яз. Англ.

3 Wang, Y. Efficient and Secure Deduplication for Cloud-based Backups / Y. Wang. – 2015 г.

4 Виды резервного копирования [Электронный ресурс] / URL: <https://backupsolution.ru/backup-types/> (дата обращения: 17.09.2023). – Загл. с экрана. – Яз. Рус.

5 Nelson, S. Pro Data Backup and Recovery / S. Nelson. – Springer Science+Business Media, 2011. – 280 с.

6 Wikipedia. Comparison of backup software [Электронный ресурс] / URL: https://en.wikipedia.org/wiki/Comparison_of_backup_software (дата обращения: 12.01.2024). – Загл. с экрана. – Яз. Англ.

7 Tridgell, A. Efficient Algorithms for Sorting and Synchronization / A. Tridgell. – Australian National University, 1999. – 105 с.

8 Preston, W. C. Backup & Recovery: Inexpensive Backup Solutions for Open Systems / W. C. Preston. – O'Reilly Media, 2007. – 768 с.

9 Подключение к API Яндекс ID. Получение кода подтверждения от пользователя [Электронный ресурс] / URL: <https://yandex.ru/dev/id/doc/ru/codes/screen-code> (дата обращения: 27.09.2023). – Загл. с экрана. – Яз. Рус.

10 Яндекс.Диск Полигон [Электронный ресурс] / URL: <https://yandex.ru/dev/disk/poligon/> (дата обращения: 27.09.2023). – Загл. с экрана. – Яз. Рус.

11 Habr. Четыре способа написать Hello world, или инструменты для создания GUI на Python [Электронный ресурс] / URL: <https://habr.com/ru/companies/selectel/articles/750146/> (дата обращения: 19.09.2023). – Загл. с экрана. – Яз. Рус.

12 Python GUI: создаём простое приложение с PyQt и Qt Designer [Электронный ресурс] / URL: <https://tproger.ru/translations/python-gui-pyqt> (дата обращения: 19.09.2023). – Загл. с экрана. – Яз. Рус.

13 Riverbank Computing. PyQt5 Documentation [Электронный ресурс] / URL: <https://www.riverbankcomputing.com/static/Docs/PyQt5/index.html> (дата обращения: 20.09.2023). – Загл. с экрана. – Яз. Англ.

14 Qt for Python Documentation. Editable Tree Model Example [Электронный ресурс] / URL: https://doc.qt.io/qtforpython-6/examples/example_widgets_itemviews_editabletreemodel.html (дата обращения: 22.09.2023). – Загл. с экрана. – Яз. Англ.

15 RussianBlogs. Понимание сигналов и слотов в pyqt5. URL: <https://russianblogs.com/article/75191190117/> (дата обращения: 22.09.2023). – Загл. с экрана. – Яз. Рус.

16 Qt for Python Documentation. PySide Documentation contents [Электронный ресурс] / URL: <https://pyside.github.io/docs/pyside/contents.html> (дата обращения: 25.09.2023). – Загл. с экрана. – Яз. Рус.

17 Работа с REST API Яндекс.Диска через Python Requests [Электронный ресурс] / <https://ramziv.com/article/8> (дата обращения: 26.09.2023). – Загл. с экрана. – Яз. Рус.

18 Yandex.Disk Documentation. Документация YaDisk [Электронный ресурс] / URL: <https://yadisk.readthedocs.io/ru/latest/index.html> (дата обращения: 28.09.2023). – Загл. с экрана. – Яз. Рус.

19 GitHub Repository: ivknv/yadisk [Электронный ресурс] / URL: <https://github.com/ivknv/yadisk> (дата обращения: 28.09.2023). – Загл. с экрана. – Яз. Рус.

20 Qt/C++ - Lesson 038. Switching between windows in the Qt [Электронный ресурс] / URL: <https://evileg.com/en/post/112/> (дата обращения: 13.10.2023). – Загл. с экрана. – Яз. Англ.

21 Let's Node Blog. Деплой Python программ на Linux [Электронный ресурс] / URL: https://blog.letsnode.io/Python_deploy (дата обращения: 26.11.2023). – Загл. с экрана. – Яз. Рус.

ПРИЛОЖЕНИЕ А

Листинг программы

Содержание файла MainWindowUI.py

```
# Разные библиотеки
import sys
import tempfile
import os
import yadisk
from zipfile import ZipFile
from hashlib import sha256
from datetime import datetime

# PyQt
from PyQt5.QtGui import QIcon, QFont, QCloseEvent
from PyQt5.QtCore import pyqtSlot, pyqtSignal, QModelIndex, Qt, QDir
from PyQt5.QtWidgets import (QAbstractItemView, QTreeView, QApplication,
                              QMainWindow,
                              QHeaderView, QDialog, QPushButton, QLineEdit,
                              QComboBox,
                              QTreeWidget, QTreeWidgetItem, QVBoxLayout,
                              QWidget, QToolButton, QLabel, QHBoxLayout,
                              QFileSystemModel, QSpacerItem)

# Мои файлы
from myTreeWidgetItem import TreeItem
from workFolder import WorkFolder, WorkFolderFile,
load_workfolders_from_xml, save_workfolders_to_xml
from workFolderTree import WorkFolderTree
from errorHandler import ErrorHandler
from WorkFolderUI2 import WorkFolderUI, ChangeList
from AuthenticationScreen import Authenticate
from backup import Backup
from BackupRestoreUI import BackupRestoreUI
from WorkFolderConfigureUI import ConfigureWorkFolderUI

class CustomFileSystemModel(QFileSystemModel):
    def __init__(self):
        super(QFileSystemModel, self).__init__()

        self.horizontalHeaders = [''] * 4
        self.setHeaderData(0, Qt.Horizontal, "Column 0")
        self.setHeaderData(1, Qt.Horizontal, "Column 1")

    def data(self, index, role):
        if role == Qt.DisplayRole and index.column() == 2: # Предполагаем,
что размер в байтах
            file_info = self.fileInfo(index)
            return file_info.size()
        return super().data(index, role)
```

```

    def setHeaderData(self, section, orientation, data, role=Qt.EditRole):
        if orientation == Qt.Horizontal and role in (Qt.DisplayRole,
Qt.EditRole):
            try:
                self.horizontalHeaders[section] = data
                return True
            except:
                return False
        return super().setHeaderData(section, orientation, data, role)

    def headerData(self, section, orientation, role=Qt.DisplayRole):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            try:
                return self.horizontalHeaders[section]
            except:
                pass
        return super().headerData(section, orientation, role)

class CustomDialog(QDialog):
    def __init__(self, parent=None):
        super(CustomDialog, self).__init__(parent)
        layout = QVBoxLayout()

        buttonLayout = QHBoxLayout()

        label = QLabel(text="Имя новой папки:")
        self.main_font = QFont('Times New Roman', 14)
        self.setStyleSheet("QLabel { background-color: white }")
        label.setFont(self.main_font)

        self.line_edit = QLineEdit(self)
        self.line_edit.setFont(self.main_font)
        layout.addWidget(label)
        layout.addWidget(self.line_edit)

        okButton = QPushButton("OK", self)
        cancelButton = QPushButton("Отмена", self)

        okButton.clicked.connect(self.okButtonClicked)
        cancelButton.clicked.connect(self.cancelButtonClicked)

        buttonLayout.addWidget(okButton)
        buttonLayout.addWidget(cancelButton)
        layout.addLayout(buttonLayout)

        self.setLayout(layout)

    def okButtonClicked(self):
        self.accept()

    def cancelButtonClicked(self):
        self.reject()

```

```

def get_input_value(self):
    return self.line_edit.text()

class FileTreeViewer(QMainWindow):
    item_selected_signal = pyqtSignal()
    update_view_signal = pyqtSignal()
    update_on_upload_signal = pyqtSignal(str)
    errorHandler = ErrorHandler()
    workfolder_deleted_signal = pyqtSignal()
    cloud_path_root = ''
    app_folder = 'disk:/App'
    work_folder_list_file = 'data/WorkFolders.xml'

    def __init__(self, y: yadisk.YaDisk):
        super().__init__()
        self.show()
        self.y = y

        self.root = TreeItem(y.get_meta(self.cloud_path_root, limit=0))
        build_hierarchy(self.root, self.app_folder)
        self.app_folder_root = TreeItem(y.get_meta(self.app_folder,
limit=0))
        self.root_work_folder_tree =
WorkFolderTree(WorkFolder(self.cloud_path_root, self.root.get_name()),
self.root)
        self.work_folder_tree_map = {}          # Соответствие между пунктами
списка папок и объектами папок.
        self.work_folder_tree_map[self.root_work_folder_tree.get_name()] =
self.root_work_folder_tree
        self.work_folder_tree_list = []
        self.current_selected_work_folder_tree = self.root_work_folder_tree
        self.item_map = {}

        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Две панели файловой системы')
        self.setGeometry(100, 100, 1600, 800)

        # Левое дерево
        self.tree_widget_left = QTreeWidget(self)
        self.tree_widget_left.setHeaderLabels(['Имя', 'Тип', 'Размер',
'Дата создания', 'Дата изменения'])
        self.tree_widget_left.setRootIsDecorated(True)
        self.tree_widget_left.setAlternatingRowColors(True)
        self.tree_widget_left.setStyleSheet("QTreeWidget::item { border-
bottom: 1px solid black; border-right: 1px dark grey; }")

        self.tree_widget_left.itemSelectionChanged.connect(self.handle_item_selecti
on_left)

```

```

        self.item_selected_signal.connect(self.handle_item_selected_signal)
## При выборе строки в дереве возвращается объект TreeItem

self.update_on_upload_signal.connect(self.handle_update_on_upload_signal)


self.tree_widget_left.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)

self.tree_widget_left.setHorizontalScrollMode(QAbstractItemView.ScrollMode.ScrollPerPixel)

self.tree_widget_left.setVerticalScrollMode(QAbstractItemView.ScrollMode.ScrollPerPixel)

self.tree_widget_left.setHorizontalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAlwaysOn)
    left_scroll_bar = self.tree_widget_left.horizontalScrollBar()
    left_scroll_bar.setSingleStep(20)
    self.tree_widget_left.setAnimated(True)
    self.tree_widget_left.setAllColumnsShowFocus(False)

    self.tree_widget_left.setStyleSheet("""
        QTreeWidget::item:selected {
            background-color: lightblue;
            color: black;
        }
    """)

# Настройка ширины столбцов
for column in range(self.tree_widget_left.columnCount()):
    self.tree_widget_left.setColumnWidth(column, 300)

# Построение дерева
self.build_tree(self.tree_widget_left, self.root)

# Правое дерево
self.file_model = QFileSystemModel()
file_model_root_index =
self.file_model.setRootPath(QDir.rootPath())
    self.file_model.setHeaderData(4, Qt.Horizontal, "Новый столбец")
    self.file_model.setFilter(QDir.Filter.AllEntries |
QDir.Filter.NoDotAndDotDot)

    self.tree_widget_right = QTreeView(self)
    self.tree_widget_right.setModel(self.file_model)

    header = QHeaderView(Qt.Orientation.Vertical,
self.tree_widget_left)

    self.tree_widget_right.header().selectAll()

```

```

        self.tree_widget_right.setRootIndex(self.file_model.index('')) #
Установка корневой директории
        self.tree_widget_right.setRootIsDecorated(True)
        self.tree_widget_right.setAlternatingRowColors(True)
        self.tree_widget_right.setStyleSheet("QTreeWidgetItem { border-
bottom: 1px solid black; }")
        self.tree_widget_right.setAllColumnsShowFocus(False)
        self.tree_widget_right.setStyleSheet("""
            QTreeWidgetItem:selected {
                background-color: lightblue;
                color: black;
            }
        """)

self.tree_widget_right.setSelectionBehavior(QAbstractItemView.SelectionBeha
vior.SelectRows)

self.tree_widget_right.setHorizontalScrollMode(QAbstractItemView.ScrollMode
.ScrollPerPixel)

self.tree_widget_right.setVerticalScrollMode(QAbstractItemView.ScrollMode.S
crollPerPixel)

self.tree_widget_right.setHorizontalScrollBarPolicy(Qt.ScrollBarPolicy.Scro
llBarAlwaysOn)
        ridget_scroll_bar = self.tree_widget_right.horizontalScrollBar()
        ridget_scroll_bar.setSingleStep(20)
        selection_model = self.tree_widget_right.selectionModel()

selection_model.selectionChanged.connect(self.on_item_selected_right)


        self.tree_widget_right.setColumnWidth(0, 200)
        for column in range(1,
self.file_model.columnCount(file_model_root_index)):
            self.tree_widget_right.setColumnWidth(column, 75)


        # Строка пути
        self.main_font = QFont('Times New Roman', 14)
        self.setStyleSheet("QLabel { background-color: lightgrey }")
# Все label имеют свето-серый цвет
        self.address_label_left = QLabel("Путь")
        self.address_label_left.setFont(self.main_font)
        self.address_label_right = QLabel("Путь")
        self.address_label_right.setFont(self.main_font)


        # Кнопки со стрелками
        arrow_left_button = QToolButton()
        arrow_left_button.setIcon(QIcon('data/icons/left_arrow.png'))

```

```

arrow_left_button.setToolTip("Загрузить")
arrow_left_button.clicked.connect(self.upload_to_cloud)

arrow_right_button = QToolButton()
arrow_right_button.setIcon(QIcon('data/icons/right_arrow.png'))
arrow_right_button.setToolTip("Скачать")
arrow_right_button.clicked.connect(self.download_from_cloud)

# Кнопка обновления
reload_view_button = QToolButton()
reload_view_button.setIcon(QIcon('data/icons/reload_button.png'))
reload_view_button.setToolTip("Обновить      | F5")
reload_view_button.clicked.connect(self.reload_tree)
reload_view_button.setShortcut("F5")

# Кнопка новой папки
new_dir_button = QToolButton()
new_dir_button.setIcon(QIcon('data/icons/add_folder.png'))
new_dir_button.setToolTip("Создать папку      | Ctrl+N")
new_dir_button.clicked.connect(self.add_folder)
new_dir_button.setShortcut("Ctrl+N")

# Кнопка удаления папки
remove_dir_button = QToolButton()
remove_dir_button.setToolTip("Удалить      | Ctrl+D")
remove_dir_button.setIcon(QIcon("data/icons/delete_icon.png"))
remove_dir_button.clicked.connect(self.remove_file_or_folder)
remove_dir_button.setShortcut("Ctrl+D")

# Кнопка создания бэкапа
make_backup_button = QToolButton()
make_backup_button.setToolTip("Создать резервную копию      | Ctrl+P")
make_backup_button.setIcon(QIcon("data/icons/create_backup.png"))
make_backup_button.setText('Создать копию')
make_backup_button.clicked.connect(self.create_incremental_backup)
make_backup_button.setShortcut("Ctrl+P")

# Кнопка восстановления бэкапа
restore_backup_button = QToolButton()
restore_backup_button.setToolTip("Восстановить резервную копию      | Ctrl+R")
restore_backup_button.setIcon(QIcon("data/icons/restore_backup.png"))
restore_backup_button.setText('Восстановить копию')
restore_backup_button.clicked.connect(self.restore_backup)
restore_backup_button.setShortcut("Ctrl+R")

# Кнопка настройки рабочей папки
configure_workfolder_button = QToolButton()
configure_workfolder_button.setToolTip("Настройка рабочей папки")

```



```

configure_workfolder_button.setIcon(QIcon("data/icons/configure_folder.png"
))
    configure_workfolder_button.setText('Настройка рабочей папки')

configure_workfolder_button.clicked.connect(self.configure_workfolder)


# Выпадающий список рабочих папок
self.workFolderBox = QComboBox()
self.workFolderBox.setObjectName("workFolderBox")
self.workFolderBox.setFixedSize(250, 20)
self.workFolderBox.setToolTip("Список рабочих папок")

self.workFolderBox.addItem(self.root_work_folder_tree.get_name())
self.initial_load_work_folders()
self.workFolderBox.setCurrentIndex(0)

self.workFolderBox.currentIndexChanged.connect(self.handle_work_folder_box_
selection)


# Меню бар
menubar = self.menuBar()
file_menu = menubar.addMenu("&Файл")
self.exit_action = file_menu.addAction("Выход")
self.exit_action.setShortcut("Ctrl+Q")
self.exit_action.triggered.connect(self.close)

self.reload_action = file_menu.addAction("Обновить")
self.reload_action.triggered.connect(self.reload_tree)

self.save_folder_action = file_menu.addAction("Сохранить")
self.save_folder_action.setShortcut("Ctrl+S")
self.save_folder_action.triggered.connect(self.saveFolders)

self.folder_setting_action = file_menu.addAction("Создание и
удаление рабочих папок")

self.folder_setting_action.triggered.connect(self.init_workfolder_UI)


# Настройка выкладки
verticalMainLayout = QVBoxLayout()

hLayout1 = QHBoxLayout()
hLayout1.addWidget(reload_view_button)
hLayout1.addWidget(new_dir_button)
hLayout1.addWidget(remove_dir_button)
hLayout1.addWidget(make_backup_button)
hLayout1.addWidget(restore_backup_button)
hLayout1.addWidget(self.workFolderBox)
hLayout1.addWidget(configure_workfolder_button)
hLayout1.setAlignment(Qt.AlignLeft)

```

```

leftLayout = QVBoxLayout()
leftLayout.addWidget(self.address_label_left)
leftLayout.addWidget(self.tree_widget_left)

buttonLayout = QVBoxLayout()
buttonLayout.addSpacing(400)
buttonLayout.addWidget(arrow_left_button)
buttonLayout.addWidget(arrow_right_button)
buttonLayout.addSpacing(400)

rightLayout = QVBoxLayout()
rightLayout.addWidget(self.address_label_right)
rightLayout.addWidget(self.tree_widget_right)

hLayout2 = QHBoxLayout()
hLayout2.addLayout(leftLayout)
hLayout2.addLayout(buttonLayout)
hLayout2.addLayout(rightLayout)

verticalMainLayout.addLayout(hLayout1)
verticalMainLayout.addLayout(hLayout2)

central_widget = QWidget()
central_widget.setLayout(verticalMainLayout)
self.setCentralWidget(central_widget)

# ===== BACKUP BLOCK
=====

@pyqtSlot()
def create_incremental_backup(self, dialog_call = False):
    if self.current_selected_work_folder_tree ==
self.root_work_folder_tree:
        self.errorHandler.handle('Error making backup: workfolder must
be selected.')
        return

    if dialog_call:
        new_name = 'Получаем имя из диалога'
    else:
        new_name = datetime.__format__(datetime.now(), "%d-%m-%Y")

    workFolder =
self.current_selected_work_folder_tree.get_work_folder()
    workFolderFiles = workFolder.get_files().copy()

    files_to_add = []    # пути файлов, которые нужно добавить
    deleted_files = []  # пути удаленных файлов
    last_number = 0

```

```

        for backup in workFolder.get_sorted_backups_list():
            if backup.get_number() > last_number: last_number =
backup.get_number()

            backup_files = backup.get_files()
            backup_deleted_files = backup.get_deleted_files()

            files_to_remove = []
            for file in workFolderFiles:
                local_path = file.get_local_path()
                cloud_path =
f'{workFolder.get_backups_path()}/{backup.get_number()}/{os.path.basename(l
ocal_path)}'
                if not os.path.exists(local_path):
                    if local_path in backup_deleted_files:
                        files_to_remove.append(file)
                    if local_path in deleted_files:
                        deleted_files.remove(local_path)
                    continue
                else:
                    if not local_path in deleted_files:
                        deleted_files.append(local_path)
                    continue

                if local_path not in backup_files:
                    continue

                cloud_hash = self.y.get_meta(cloud_path) ['sha256']
                local_hash = calculate_sha256(local_path)

                if not local_hash == cloud_hash:
                    files_to_add.append(local_path)
                    files_to_remove.append(file)

            for file in files_to_remove:
                workFolderFiles.remove(file)

# Сверка оставшихся файлов с полным бэкапом.
if not len(workFolderFiles) == 0:
    for file in workFolderFiles:
        local_path = file.get_local_path()
        cloud_path = file.get_cloud_path()

        cloud_hash = self.y.get_meta(cloud_path) ['sha256']
        local_hash = calculate_sha256(local_path)
        if not local_hash == cloud_hash:
            files_to_add.append(local_path)

if len(files_to_add) == 0:
    self.errorHandler.handle('New backup is not needed.')
    return

```

```

        last_number += 1
        new_backup = Backup(name=new_name + ' ' + str(last_number),
path=f'{workFolder.get_backups_path()}/{last_number}', number=last_number,
creation=datetime.now())
        # Получаем список файлов, которые нужно добавить в новый бэкап и
список файлов, которые были удалены.
        for file in files_to_add:
            new_backup.add_file(file)
        for file in deleted_files:
            new_backup.add_deleted_file(file)

        self._upload_backup(new_backup)

def _upload_backup(self, backup: Backup):
    try:
        self.y.mkdir(backup.get_path())
    except Exception as e:
        self.errorHandler.handle(f'Error making backup dir: {e}')
        return

    self.current_selected_work_folder_tree.add_backup(backup)
    for file in backup.get_files():
        name = os.path.basename(file)
        try:
            self.y.upload(file, backup.get_path() + f'/{name}')
        except Exception as e:
            self.errorHandler.handle(f'Error uploading file to backup:
{e}')
```

```

def _restore_files(self, files):
    for file in files:
        local_path, cloud_path = file
        if os.path.exists(local_path):
            try:
                os.remove(local_path)
            except Exception as e:
                self.errorHandler.handle(f'Error removing file: {e}')

        try:
            self.y.download(cloud_path, local_path)
        except Exception as e:
            self.errorHandler.handle(f'Error downloading file: {e}')
    self.errorHandler.handle('Backup restoring complete.')
```

```

@pyqtSlot()
def restore_backup(self):
    '(Восстановление туда же, откуда сохранялось и восстановление в
определенную папку)'
    '(Функция получения списка удаленных файлов до нужной даты) (Функция
удаления файла и скачивания)'
```

```

        '1) Берем бэкап, который нужно восстановить.'
        '2) Восстанавливаем из него файлы. Записываем восстановленные файлы
в список. Записываем список удаленных файлов.'
        '3) Проверяем файлы из следующего бэкапа. Если они не были уже
восстановлены и их нет в списке удаленных, то восстанавливаем. Добавляем к
списку восстановленных. Добавляем список удаленных'
        '4) Дошли до полной копии. Восстанавливаем только те, которые не
были восстановлены или удалены.'

        workFolder =
self.current_selected_work_folder_tree.get_work_folder()
        workFolderFiles = workFolder.get_files().copy()

        dialog =
BackupRestoreUI(self.current_selected_work_folder_tree.get_work_folder().g
et_sorted_backups_list(), self)
        result = dialog.exec_()

        if result == QDialog.Rejected:
            return

        selected_backup = dialog.get_input_value()
        files_to_restore = []          # хранит пары (локальный путь, облачный
путь)
        deleted_files = []
        tmp_filepath_list = []
        if not selected_backup.get_name() == 'Full backup':
            for backup in
workFolder.get_sorted_backups_list(until_date=datetime.strptime(selected_ba
ckup.get_creation_time(), Backup.datetime_format)):
                deleted_files += backup.get_deleted_files()
                for file in backup.get_files():
                    local_path = file
                    filename = os.path.basename(local_path)
                    cloud_path =
f'{workFolder.get_backups_path()}/{backup.get_number()}/{filename}'

                    if not local_path in tmp_filepath_list and not
local_path in deleted_files:
                        tmp_filepath_list.append(local_path)
                        files_to_restore.append((local_path, cloud_path))

        # Просмотрели все бэкапы. Имеем список файлов для восстановления
        # Смотрим оставшийся полный бэкап
        for file in workFolderFiles:
            local_path = file.get_local_path()
            cloud_path = file.get_cloud_path()

            if not local_path in tmp_filepath_list and not local_path in
deleted_files:
                tmp_filepath_list.append(local_path)
                cloud_hash = self.y.get_meta(cloud_path)['sha256']

```

```

        local_hash = calculate_sha256(local_path)
        if not local_hash == cloud_hash:
            files_to_restore.append((local_path, cloud_path))

    self._restore_files(files_to_restore)

# ===== MISC BLOCK
=====
    def closeEvent(self, event = QCloseEvent()):
        self.saveFolders()
        ErrorHandler.handle("===== Session end
=====")
        # Вызов метода closeEvent базового класса
        super().closeEvent(event)

# ===== SELECTION AND SIGNAL HANDLING BLOCK
=====
    def handle_work_folder_box_selection(self):
        workFolderTree =
self.work_folder_tree_map.get(self.workFolderBox.currentText())
        workFolder = workFolderTree.get_work_folder()
        workFolderRoot = workFolderTree.get_root()
        self.current_selected_work_folder_tree = workFolderTree

        self.tree_widget_left.clear()
        self.build_tree(self.tree_widget_left, workFolderRoot)

        self.current_selected_treeWidget: QTreeWidgetItem =
self.tree_widget_left.topLevelItem(0)
        self.current_selected_cloud_item: TreeItem =
self.item_map.get(id(self.current_selected_treeWidget))

self.address_label_left.setText(self.current_selected_cloud_item.get_path()
)

    @pyqtSlot()
    def handle_item_selection_left(self):
        selected_items = self.tree_widget_left.selectedItems()
        if selected_items:
            selected_item = selected_items[0]
            item_id = id(selected_item)

            # Используем идентификатор для получения объекта TreeItem из
словаря
            tree_item = self.item_map.get(item_id)
            self.current_selected_treeWidget: QTreeWidgetItem =
selected_item
            self.tmp_selected_treeWidget: QTreeWidgetItem = selected_item
            self.current_selected_cloud_item: TreeItem = tree_item
            self.tmp_selected_cloud_item: TreeItem = tree_item

```

```

        if tree_item:
            self.item_selected_signal.emit()

    def handle_item_selected_signal(self):
## Signal handler

self.address_label_left.setText(self.current_selected_cloud_item.get_path()
)

    @pyqtSlot()
    def on_item_selected_right(self):
        index = self.tree_widget_right.currentIndex()
        self.current_selected_local_item: QModelIndex = index
        self.tmp_selected_local_item: QModelIndex = index
        full_path = self.file_model.filePath(index)
        self.address_label_right.setText(full_path)

# ===== VIEW BLOCK
=====
    @pyqtSlot()
    def reload_tree(self):
        self.tree_widget_left.clear()
        self.item_map = {}

        root = TreeItem(self.y.get_meta(self.cloud_path_root, limit=0))
        if not self.current_selected_work_folder_tree ==
self.root_work_folder_tree:
            build_hierarchy(root, self.app_folder,
self.current_selected_work_folder_tree.get_backups_path())
        else:
            build_hierarchy(root, self.app_folder)
        self.root = root

        self.build_tree(self.tree_widget_left, self.root)

self.tree_widget_left.expandItem(self.tree_widget_left.itemFromIndex(self.t
ree_widget_left.rootIndex()))

    def build_tree(self, tree_widget, tree_item: TreeItem):
        datetime_format = "%d-%m-%Y %H:%M"
        if tree_item.get_type() == 'file':
            item = QTreeWidgetItem(tree_widget, [tree_item.get_name(),
tree_item.get_mime_type(),
tree_item.get_size(),
tree_item.get_created().strftime(datetime_format),
tree_item.get_modified().strftime(datetime_format)])
        else:
            item = QTreeWidgetItem(tree_widget, [tree_item.get_name(),
'Папка',

```

```

tree_item.get_created().strftime(datetime_format),
tree_item.get_modified().strftime(datetime_format)])

    self.item_map[id(item)] = tree_item
    # Рекурсивно добавляем дочерние элементы
    for child_item in tree_item.child_items:
        self.build_tree(item, child_item)

def update_on_file_upload(self, cloud_path):
    new_item = TreeItem(self.y.get_meta(cloud_path, limit=0))
    new_tree_widget = QTreeWidgetItem(None, [new_item.get_name(),
new_item.get_mime_type()])

    parent_item = self.current_selected_cloud_item
    tree_widget = self.current_selected_treeWidget

    if parent_item.is_file():
        parent_item = parent_item.get_parent()
        tree_widget = tree_widget.parent()

    parent_item.add_child(new_item)
    tree_widget.addChild(new_tree_widget)

    self.item_map[id(new_tree_widget)] = new_item

def update_on_folder_upload(self, cloud_path):
    parent_item = self.current_selected_cloud_item
    tree_widget = self.current_selected_treeWidget
    if parent_item.is_file():
        parent_item = parent_item.get_parent()
        tree_widget = tree_widget.parent()

    new_folder_item = TreeItem(self.y.get_meta(cloud_path, limit=0))

    build_hierarchy(new_folder_item)
    self.build_tree(tree_widget, new_folder_item)
    parent_item.add_child(new_folder_item)

# Connected to update_on_upload_signal. Emitted in def upload_to_cloud
def handle_update_on_upload_signal(self, cloud_path):
    if self.y.is_dir(cloud_path):
        self.update_on_folder_upload(cloud_path)
    else:
        self.update_on_file_upload(cloud_path)

def handle_update_on_delete_signal(self):
    print('deleted')

# ===== CLOUD BLOCK
=====
def _get_folder_children(self, path):
    dir_model = QDir()

```



```

        dir_model.cd(path)
        children_list = [child for child in dir_model.entryInfoList() if
child.absolutePath() == dir_model.path()]
        return children_list

    def _construct_paths(self):
        local_path =
self.file_model.filePath(self.current_selected_local_item)
        file_name =
self.file_model.fileName(self.current_selected_local_item)
        cloud_path = self.current_selected_cloud_item.get_path()

        if self.current_selected_cloud_item.is_file():
            parent_item = self.current_selected_cloud_item.get_parent()
            cloud_path = parent_item.get_path() + f'/{file_name}'
        else:
            cloud_path += f'/{file_name}'
        return local_path, cloud_path

    def _upload_file_to_cloud(self, local_path, cloud_path):
        try:
            self.y.upload(local_path, cloud_path)
            if not self.current_selected_work_folder_tree ==
self.root_work_folder_tree:

self.current_selected_work_folder_tree.add_file(WorkFolderFile(local_path,
os.path.basename(local_path), cloud_path))
        except Exception as e:
            self.errorHandler.handle(f"Error uploading file: {e}")

    def _upload_folder_to_cloud(self, local_path, cloud_path):
        try:
            self.y.mkdir(cloud_path)
        except Exception as e:
            self.errorHandler.handle(f"Error mkdir: {e}")
        local_children = self._get_folder_children(local_path)
        for child in local_children:
            new_cloud_path = cloud_path + f'/{child.fileName()}'
            if
self.file_model.isDir(self.file_model.index(child.absoluteFilePath())):
                self._upload_folder_to_cloud(child.absoluteFilePath(),
new_cloud_path)
            else:
                self._upload_file_to_cloud(child.absoluteFilePath(),
new_cloud_path)

    @pyqtSlot()
    def upload_to_cloud(self):
        local_path, cloud_path = self._construct_paths()
        print(f'Uploading {local_path} to {cloud_path}.')
        if self.file_model.isDir(self.current_selected_local_item):

```

```

        if not self.current_selected_work_folder_tree ==
self.root_work_folder_tree:

self.current_selected_work_folder_tree.add_folder(local_path)
    self._upload_folder_to_cloud(local_path, cloud_path)
    else:
        self._upload_file_to_cloud(local_path, cloud_path)
    print("Upload complete.")
    self.update_on_upload_signal.emit(cloud_path)

@pyqtSlot()
def download_from_cloud(self):
    local_path = self.address_label_right.text()
    cloud_path = self.address_label_left.text()

    if self.current_selected_cloud_item.is_file():
        if self.file_model.isDir(self.current_selected_local_item):
            local_path +=
f'/{self.current_selected_cloud_item.get_name()}'
        else:
            local_path =
self.file_model.filePath(self.current_selected_local_item.parent()) +
f'/{self.current_selected_cloud_item.get_name()}'
        else:
            if self.file_model.isDir(self.current_selected_local_item):
                local_path =
self.file_model.filePath(self.current_selected_local_item)
            else:
                local_path =
self.file_model.filePath(self.current_selected_local_item.parent())

    temp_folder = tempfile.mkdtemp()

    # Скачиваем файл
    download_path = os.path.join(temp_folder, 'downloaded_file.zip')
    print(download_path)

    cloud_path = _cloud_path_fixer(cloud_path)

    try:
        self.y.download(cloud_path, download_path)
    except Exception as e:
        self.errorHandler.handle(f"Error downloading file: {e}")

    with ZipFile(download_path, 'r') as zip_ref:
        zip_ref.extractall(local_path)

    os.remove(download_path)
    os.rmdir(temp_folder)

@pyqtSlot()

```

```

def add_folder(self):
    new_folder_name = ''
    cloud_path = ''

    dialog = CustomDialog(self)
    result = dialog.exec_()

    if result == QDialog.Accepted:
        new_folder_name = dialog.get_input_value()

    if new_folder_name == '':
        return
    if self.current_selected_cloud_item.is_dir():
        cloud_path = self.current_selected_cloud_item.get_path() +
f'/{new_folder_name}'
    else:
        cloud_path =
self.current_selected_cloud_item.get_parent().get_path() +
f'/{new_folder_name}'

    cloud_path = _cloud_path_fixer(cloud_path)

    try:
        self.y.mkdir(cloud_path)
        self.update_on_upload_signal.emit(cloud_path)
    except Exception as e:
        self.errorHandler.handle(f'Failed mkdir: {e}. {cloud_path}')

@pyqtSlot()
def remove_file_or_folder(self):
    cloud_path = self.current_selected_cloud_item.get_path()
    treeItem = self.current_selected_treeWidget

    if self.current_selected_work_folder_tree ==
self.root_work_folder_tree:
        try:
            self.y.remove(cloud_path)
            treeItem.parent().removeChild(treeItem)
        except Exception as e:
            self.errorHandler.handle(f'Error removing dir: {e}')

# ===== WORKFOLDER BLOCK
=====
def initial_load_work_folders(self):
    if not os.path.exists(self.work_folder_list_file):
        print(f"File {self.work_folder_list_file} doesn't exist")
        return False

    if os.path.getsize(self.work_folder_list_file) == 0:
        print(f"File {self.work_folder_list_file} is empty")
        return False

```

```

        work_folder_list =
load_workfolders_from_xml(self.work_folder_list_file)
        name_list = []
        for workFolder in work_folder_list:
            workFolder = WorkFolder(work_folder=workFolder)

            workFolderRoot =
TreeItem(self.y.get_meta(workFolder.get_path(), limit=0))
            build_hierarchy(workFolderRoot)
            workFolderTree = WorkFolderTree(workFolder, workFolderRoot)

            name_list.append(workFolder.get_name())
            self.work_folder_tree_map[workFolder.get_name()] =
workFolderTree
            self.work_folder_tree_list.append(workFolderTree)

        self.workFolderBox.addItem(name_list)
        return True

    def saveFolders(self):
        folder_list = []
        for treeFolder in self.work_folder_tree_list:
            folder_list.append(treeFolder.get_work_folder())

        try:
            save_workfolders_to_xml(folder_list,
self.work_folder_list_file)
        except Exception as e:
            print(f'Error saving workfolders: {e}')

    def init_workfolder_UI(self):
        workFolderList = list(self.work_folder_tree_map.keys())
        workFolderUI = WorkFolderUI(workFolderList=workFolderList,
parent=self)
        workFolderUI.sendDataSignal.connect(self._recieveData)
        workFolderUI.show()

    def _recieveData(self, status: int, changeList: ChangeList):      # ()
        if status == 0:
            return

        toCreateList = changeList.return_data().get('Create')
        toDeleteList = changeList.return_data().get('Delete')

        for item in toCreateList:
            path = f'{self.app_folder}/{item}'
            try:
                self.y.mkdir(path)
                self.workFolderBox.addItem(item)
                folderTree = WorkFolderTree(WorkFolder(path, item),
TreeItem(self.y.get_meta(path, limit=0)))
                self.y.mkdir(folderTree.get_backups_path())

```

```

        self.work_folder_tree_list.append(folderTree)
        self.work_folder_tree_map[item] = folderTree

    except Exception as e:
        self.errorHandler.handle(f'Error making directory: {e}')

    for item in toDeleteList:
        self.remove_work_folder(item)

# Удаление всех упоминаний папки из списков, в которых она хранится
def _delete_work_folder(self, work_folder: WorkFolderTree):
    try:
        self.work_folder_tree_map.pop(work_folder.get_name())

    self.work_folder_tree_list.pop(self.work_folder_tree_list.index(work_folder
    ))

    self.workFolderBox.removeItem(self.workFolderBox.findText(work_folder.get_n
    ame()))

        self.tree_widget_left.clear()
        self.workFolderBox.setCurrentIndex(0)

        self.build_tree(self.tree_widget_left,
        self.root_work_folder_tree.get_root())

        self.current_selected_treeWidget: QTreeWidgetItem =
        self.tree_widget_left.topLevelItem(0)
        self.current_selected_cloud_item: TreeItem =
        self.item_map.get(id(self.current_selected_treeWidget))
    except Exception as e:
        self.errorHandler.handle(f'Error removing workfolder: {e}')

# Используется при обработке изменений в списке рабочих папок
def remove_work_folder(self, folder_name):
    folder = self.work_folder_tree_map.get(folder_name)

    try:
        self.y.remove(folder.get_path())
        self._delete_work_folder(folder)

    except Exception as e:
        self.errorHandler.handle(f'Error removing workfolder: {e}')

def configure_workfolder(self):
    if self.current_selected_work_folder_tree ==
    self.root_work_folder_tree:
        self.errorHandler.handle('Only workfolder configurable.')
        return
    ext_list = []
    for folder in self.current_selected_work_folder_tree.get_folders():

```

```

        extensions = get_file_formats(folder)
        ext_list = list(set(ext_list + extensions))

        dialog =
ConfigureWorkfolderUI(self.current_selected_work_folder_tree.get_config(),
ext_list, self)
        result = dialog.exec_()

        if result == QDialog.Rejected:
            return

        new_config = dialog.get_data()
        self.current_selected_work_folder_tree.set_config(new_config)

    @classmethod
    def get_token(self, y: yadisk.YaDisk):
        self.y = y

def get_file_formats(folder_path):
    formats = []
    for _, _, files in os.walk(folder_path):
        for file in files:
            _, file_extension = os.path.splitext(file)
            file_extension = file_extension.lstrip('.').rstrip()
            if str(file_extension) == '' or str(file_extension) == ' ':
                continue
            formats.append(file_extension)
    return formats

def build_hierarchy(root:TreeItem, appFolder = '', backups_folder = ''):
    x = y.listdir(root.get_path())
    for item in list(x):
        if (not appFolder == '' and item['path'] == appFolder) or (not
backups_folder == '' and item['path'] == backups_folder):
            continue
        treeItem = TreeItem(item, parent=root)
        root.add_child(treeItem)
        if treeItem.get_type() == 'dir':
            build_hierarchy(treeItem)

def calculate_sha256(file_path):
    sha256_hash = sha256()

    with open(file_path, "rb") as file:
        # Считываем блоки данных и обновляем хэш-сумму
        for byte_block in iter(lambda: file.read(4096), b''):
            sha256_hash.update(byte_block)

    # Получаем окончательное значение хэш-суммы
    return sha256_hash.hexdigest()

```

```

def _cloud_path_fixer(cloud_path: str):
    return cloud_path.replace('//', '/')

# ===== AUTOBACKUP BLOCK
=====
def get_files_with_exact_formats(workfolder: WorkFolder) -> list:
    file_list = [file.get_local_path() for file in workfolder.get_files()]
    folders = workfolder.get_folders()
    ext_list = workfolder.get_config().get_formats()

    result_files = []
    for folder in folders:
        for root, dirs, files in os.walk(folder):
            for file in files:
                local_path = os.path.join(root, file).replace('\\', '/')
                cloud_path = workfolder.get_path() + '/' + file
                _, file_ext = os.path.splitext(file)
                if file_ext.lstrip('.').rstrip() in ext_list and not
local_path in file_list:
                    result_files.append(WorkFolderFile(local_path, file,
cloud_path))

    return result_files

def create_incremental_backups_by_command(workfolder_list):

    new_name = datetime.__format__(datetime.now(), "%d-%m-%Y")
    for workFolder in workfolder_list:
        config = workFolder.get_config()
        if not config.get_autobackup():
            continue
        workFolderFiles = workFolder.get_files().copy()

        files_to_add = []    # пути файлов, которые нужно добавить
        deleted_files = []   # пути удаленных файлов
        last_number = 0
        for backup in workFolder.get_sorted_backups_list():
            if backup.get_number() > last_number: last_number =
backup.get_number()

            backup_files = backup.get_files()
            backup_deleted_files = backup.get_deleted_files()

            files_to_remove = []
            for file in workFolderFiles:
                local_path = file.get_local_path()
                cloud_path =
f'{workFolder.get_backups_path()}/{backup.get_number()}/{os.path.basename(l
ocal_path)}'

                if not os.path.exists(local_path):
                    if local_path in backup_deleted_files:
                        files_to_remove.append(file)

```

```

        if local_path in deleted_files:
            deleted_files.remove(local_path)
            continue
        else:
            if not local_path in deleted_files:
                deleted_files.append(local_path)
            continue

    if local_path not in backup_files:
        continue

    cloud_hash = y.get_meta(cloud_path)['sha256']
    local_hash = calculate_sha256(local_path)

    if not local_hash == cloud_hash:
        files_to_add.append(local_path)
        files_to_remove.append(file)

    for file in files_to_remove:
        workFolderFiles.remove(file)

    # Сверка оставшихся файлов с полным бэкапом.
    if not len(workFolderFiles) == 0:
        for file in workFolderFiles:
            local_path = file.get_local_path()
            cloud_path = file.get_cloud_path()

            cloud_hash = y.get_meta(cloud_path)['sha256']
            local_hash = calculate_sha256(local_path)
            if not local_hash == cloud_hash:
                files_to_add.append(local_path)

    if len(files_to_add) == 0:
        ErrorHandler.handle(f'New backup for {workFolder.get_name()} is
not needed.')
        continue

    last_number += 1
    new_backup = Backup(name=new_name + ' ' + str(last_number),
path=f'{workFolder.get_backups_path()}/{last_number}', number=last_number,
creation=datetime.now())
    # Получаем список файлов, которые нужно добавить в новый бэкап и
список файлов, которые были удалены.
    for file in files_to_add:
        new_backup.add_file(file)
    for file in deleted_files:
        new_backup.add_deleted_file(file)

    _upload_backup_by_command(new_backup, workFolder)

def _upload_backup_by_command(backup: Backup, workFolder: WorkFolder):
    try:

```



```

        y.mkdir(backup.get_path())
    except Exception as e:
        ErrorHandler.handle(f'Error making backup directory: {e}')
        return

    workFolder.add_backup(backup)
    for file in backup.get_files():
        name = os.path.basename(file)
        try:
            y.upload(file, backup.get_path() + f'/{name}')
        except Exception as e:
            ErrorHandler.handle(f'Error uploading file to backup: {e}')

def _upload_file_to_cloud(local_path, cloud_path):
    try:
        y.upload(local_path, cloud_path)
        return True
    except Exception as e:
        ErrorHandler.handle(f"Error uploading file: {e}")
        return False

def initial_load_work_folders_by_command() -> (bool, list):
    if not os.path.exists(FileTreeViewer.work_folder_list_file):
        return (False, [])

    if os.path.getsize(FileTreeViewer.work_folder_list_file) == 0:
        return (False, [])

    work_folder_list =
load_workfolders_from_xml(FileTreeViewer.work_folder_list_file)
    res = []
    for workFolder in work_folder_list:
        res.append(WorkFolder(work_folder=workFolder))

    return (True, res)

def saveFolders_by_command(workfolder_list: list):
    try:
        save_workfolders_to_xml(workfolder_list,
FileTreeViewer.work_folder_list_file)
    except Exception as e:
        ErrorHandler.handle(f'Error saving workfolders: {e}')

def make_auto_backup():
    ErrorHandler.handle('Autobackup start')
    status, workfolder_list = initial_load_work_folders_by_command()
    if not status:
        ErrorHandler.handle('No workfolders that need backup.')
        return False
    for workFolder in workfolder_list:
        new_files_list = get_files_with_exact_formats(workFolder)
        for file in new_files_list:

```

```

        if _upload_file_to_cloud(file.get_local_path(),
file.get_cloud_path()):
            workFolder.add_file(file)
            with open('log.txt', 'a') as f:
                ErrorHandler.handle(f'File {file.get_local_path()}
added.')
```

```

        create_incremental_backups_by_command(workfolder_list)
        saveFolders_by_command(workfolder_list)

@pyqtSlot()
def get_token(y: yadisk.YaDisk):
    return y

if __name__ == '__main__':
    ErrorHandler.handle("===== Session start
=====")
    y = Authenticate.authenticate()

    if len(sys.argv) > 1 and sys.argv[1] == "/auto_backup":
        make_auto_backup()
        ErrorHandler.handle("===== Session end
=====")
        sys.exit()
    else:
        # Запустить основное приложение
        app = QApplication(sys.argv)
        window = FileTreeViewer(y)
        sys.exit(app.exec_())
```

Содержание файла AuthenticationScreen.py

```

import sys
import yadisk
from PyQt5 import uic
from PyQt5.QtWidgets import *
from PyQt5 import QtWidgets
from PyQt5.QtGui import QDesktopServices, QClipboard
from PyQt5.QtCore import QUrl

class Authenticate():
    token_file = 'data/token.txt'
    client_id = "236b60cb77e94e8b8aef1781bd53ea75"
    client_secret = "c7f803ee6a0243689c8efd9ea4f92e08"
    redirect_uri = "https://oauth.yandex.ru/verification_code"

    @classmethod
    def authenticate(self):
        self.y = yadisk.YaDisk()
        _token = read_token_file(self.token_file)
        if not _token == '':
            self.y.token = _token
```

```

        if self.y.check_token():
            make_app_dir(self.y)
            return self.y
        else:
            self.y = yadisk.YaDisk()

        self.y, app = authenticate_with_oauth(self.client_id,
self.client_secret)
        make_app_dir(self.y)
        make_token_file(self.token_file, self.y.token)
        app.quit()
        return self.y

def authenticate_with_oauth(client_id, client_secret):
    def loadLoginUi():
        Form, Window = uic.loadUiType("data/LoginScreenFinal.ui")
        app = QApplication(sys.argv)
        window = Window()
        form = Form()
        form.setupUi(window)
        window.show()
        window.resize(507, 100)

        form.proceedBtn.setText('Скопировать')
        form.openLinkBtn.clicked.connect(open_link_onClickBtn)
        form.okBtn.clicked.connect(enterCode_OnClickBtn)
        form.proceedBtn.clicked.connect(proceed_onClickBtn)
        return window, form, app

    def open_link_onClickBtn():
        nonlocal auth_url
        QDesktopServices.openUrl(QUrl(auth_url))
        form.stackedWidget.setCurrentIndex(1)
        window.resize(235, 153)

    def proceed_onClickBtn():
        nonlocal auth_url
        form.stackedWidget.setCurrentIndex(1)
        window.resize(235, 153)
        clipboard = QApplication.clipboard()
        clipboard.setText(auth_url)
        QMessageBox.information(window, 'Оповещение', 'Ссылка скопирована в
буфер.')

    def enterCode_OnClickBtn():
        code = form.lineEdit.text()
        try:
            # Обмен кода авторизации на токен доступа
            y.token = y.get_token(code).access_token
        except Exception as e:
            print(f"Аутентификация не удалась: {e}")
            QMessageBox.information(window, 'Оповещение', 'Аутентификация
не удалась.')

```

```

        window.close()

y = yadisk.YaDisk(client_id, client_secret)
auth_url = y.get_code_url()
window, form, app = loadLoginUi()
app.exec()

if y.check_token():
    return y, app
else:
    QMessageBox.information(window, 'Оповещение', 'Неверный код.')

def make_app_dir(y: yadisk.YaDisk):
    try:
        if not y.exists('/App'):
            y.mkdir('/App')
    except Exception as e:
        print(e)

def make_token_file(path: str, token: str):
    with open(path, 'w') as file:
        file.write(token)
        file.close()
    return

def read_token_file(path: str) -> str:
    try:
        with open(path, 'r') as file:
            token = file.read()
            file.close()
            return token
    except FileNotFoundError:
        return ''
    except Exception as e:
        print(f'Error: {e}')
        return ''

def main():
    y = Authenticate.authenticate()

if __name__ == '__main__':
    main()

```

Содержание файла myTreeItem.py

```

from yadisk.objects.resources import ResourceObject

class TreeItem:
    def __init__(self, data : ResourceObject, parent: 'TreeItem' = None,
compressed:bool = False):

```

```

        self.data = data
        self.compressed = compressed
        self.parent_item = parent
        self.child_items = []
        self.column_cnt = 2 # file, type
        self.columns_to_show = []

    def get_path(self):
        return self.data['path']
    def set_path(self, path):
        self.data['path'] = path

    def get_name(self):
        return self.data['name']
    def set_name(self, name):
        self.data['name'] = name

    def get_type(self):
        return self.data['type']
    def set_type(self, type):
        self.data['type'] = type

    def get_mime_type(self):
        return self.data['mime_type']
    def set_mime_type(self, mime):
        self.data['mime_type'] = mime

    def get_modified(self):
        return self.data['modified']
    def get_created(self):
        return self.data['created']
    def get_size(self):
        size = str(self.data['size'])
        if size == 'None':
            size = ' '
        return size

    def is_dir(self):
        if self.get_type() == 'dir':
            return True
        return False

    def is_file(self):
        if not self.is_dir():
            return True
        return False

    def add_child(self, child: 'TreeItem'):
        self.child_items.append(child)

    def set_parent(self, parent: 'TreeItem'):
        self.parent_item = parent

```

```

def delete_children(self):
    self.child_items.clear()

def child(self, number: int) -> 'TreeItem':
    if number < 0 or number >= len(self.child_items):
        return None
    return self.child_items[number]

def last_child(self):
    return self.child_items[-1] if self.child_items else None

def child_count(self) -> int:
    return len(self.child_items)

def child_number(self) -> int:
    if self.parent_item:
        return self.parent_item.child_items.index(self)
    return 0

def column_count(self) -> int:
    return self.column_cnt

def insert_children(self, position: int, count: int, columns: int) ->
bool:
    if position < 0 or position > len(self.child_items):
        return False

    for row in range(count):
        data = [None] * columns
        item = TreeItem(data.copy(), self)
        self.child_items.insert(position, item)

    return True

def get_parent(self):
    return self.parent_item

def set_data(self, column: int, value):
    if column < 0 or column >= len(self.item_data):
        return False

    self.item_data[column] = value
    return True

def __repr__(self) -> str:
    return(self.child_items)

```

Содержание файла workFolder.py

```

import xml.etree.ElementTree as ET
from backup import Backup
from datetime import datetime

```

```

from config import Config

class WorkFolder():
    def __init__(self, cloud_path: str = None, name: str = None,
work_folder: 'WorkFolder' = None):
        if work_folder:
            self.cloud_path = work_folder.get_path()
            self.name = work_folder.get_name()
            self.files = work_folder.get_files()
            self.backups = work_folder.get_backups()
            self.folders = work_folder.get_folders()
            self.config = work_folder.get_config()
        else:
            self.cloud_path = cloud_path
            self.name = name
            self.files = []
            self.backups = []
            self.folders = []
            self.config = Config()

        self.backups_path = f'{self.cloud_path}/Backups'

    def get_path(self):
        return self.cloud_path
    def set_path(self, path):
        self.cloud_path = path

    def get_name(self):
        return self.name
    def set_name(self, name):
        self.name = name

    def get_files(self) -> list:
        return self.files
    def add_file(self, file: 'WorkFolderFile'):
        self.files.append(file)
    def del_file(self, file: 'WorkFolderFile'):
        self.files.remove(file)

    def get_config(self) -> Config:
        return self.config
    def set_config(self, config: Config):
        self.config = config
    def add_format(self, format: str):
        self.config.add_format(format)
    def remove_format(self, format: str):
        self.config.remove_format(format)

    def get_backups(self) -> list:
        return self.backups
    def add_backup(self, backup: Backup):
        self.backups.append(backup)
    def del_backup(self, backup: Backup):

```

```

        self.backups.remove(backup)

    def get_folders(self) -> list:
        return self.folders
    def add_folder(self, folder_path: str):
        self.folders.append(folder_path)
    def remove_folder(self, folder_path: str):
        self.folders.remove(folder_path)

    def get_backups_path(self):
        return self.backups_path
    def set_backups_path(self, path: str):
        self.backups_path = path

    def get_backup(self, backupNumber: int) -> Backup:
        return next((backup for backup in self.backups if
backup.get_number() == backupNumber), None)

    def get_backup_name_list(self) -> list:
        res = []
        res = [backup.get_name() for backup in self.backups]
        return res

    # Отсортированный список бэкапов. Самые новые в начале
    def get_sorted_backups_list(self, until_date = None) -> list:
        mylist = self.backups.copy()
        if until_date is not None:
            mylist = filter(lambda x: x.creation_datetime <= until_date,
mylist)
        return sorted(mylist, key=lambda x: x.creation_datetime,
reverse=True)

    def get_last_backup(self) -> Backup:
        return max(self.backups, key=lambda x: x.creation_datetime,
default=None)

    def to_xml(self):
        folder_element = ET.Element('WorkFolder')

        cloud_path_element = ET.SubElement(folder_element, 'CloudPath')
        cloud_path_element.text = self.cloud_path

        name_element = ET.SubElement(folder_element, 'Name')
        name_element.text = self.name

        files_element = ET.SubElement(folder_element, 'Files')
        for file in self.files:
            file_element = ET.SubElement(files_element, 'File')

            local_path_element = ET.SubElement(file_element, 'LocalPath')
            local_path_element.text = file.local_path

```



```

        file_name_element = ET.SubElement(file_element, 'FileName')
        file_name_element.text = file.name

        cloud_path_element = ET.SubElement(file_element, 'CloudPath')
        cloud_path_element.text = file.cloud_path

    backups_element = ET.SubElement(folder_element, 'Backups')
    for backup in self.backups:
        backup_element = ET.SubElement(backups_element, 'Backup')

        name_element = ET.SubElement(backup_element, 'Name')
        name_element.text = backup.name

        path_element = ET.SubElement(backup_element, 'Path')
        path_element.text = backup.path

        number_element = ET.SubElement(backup_element, 'Number')
        number_element.text = str(backup.number)

        creation_element = ET.SubElement(backup_element, 'Creation')
        creation_element.text =
backup.creation_datetime.strftime(Backup.datetime_format)

        deleted_files_element = ET.SubElement(backup_element,
'DeletedFiles')
        for file in backup.deleted_files:
            deleted_file_element = ET.SubElement(deleted_files_element,
'DeletedFile')
            deleted_file_element.text = file

        backup_files_element = ET.SubElement(backup_element,
'BackupFiles')
        for file in backup.files:
            backup_file_element = ET.SubElement(backup_files_element,
'BackupFile')
            backup_file_element.text = file

    folders_element = ET.SubElement(folder_element, 'Folders')
    for folder in self.folders:
        _folder_element = ET.SubElement(folders_element, 'Folder')
        _folder_element.text = folder

    config_element = ET.SubElement(folder_element, 'Config')
    for ext in self.config.get_formats():
        ext_element = ET.SubElement(config_element, 'Extension')
        ext_element.text = ext
    auto_backup_element = ET.SubElement(config_element, 'Auto_backup')
    auto_backup_element.text = str(self.config.get_autobackup())

    return folder_element

@classmethod
def from_xml(cls, xml_element):

```

```

cloud_path = xml_element.find('CloudPath').text
name = xml_element.find('Name').text

work_folder = cls(cloud_path, name)

files_element = xml_element.find('Files')
for file_element in files_element.findall('File'):
    local_path = file_element.find('LocalPath').text
    file_name = file_element.find('FileName').text
    cloud_path = file_element.find('CloudPath').text

    work_folder.add_file(WorkFolderFile(local_path, file_name,
cloud_path))

backups_element = xml_element.find('Backups')
for backup_element in backups_element.findall('Backup'):
    backup_name = backup_element.find('Name').text
    path = backup_element.find('Path').text
    number = backup_element.find('Number').text
    creation = backup_element.find('Creation').text

    backup = Backup(backup_name, path, int(number),
datetime.strptime(creation, Backup.datetime_format))
    deleted_files_element = backup_element.find('DeletedFiles')
    for file in deleted_files_element.findall('DeletedFile'):
        backup.add_deleted_file(file.text)
    backup_files_element = backup_element.find('BackupFiles')
    for file in backup_files_element.findall('BackupFile'):
        backup.add_file(file.text)

    work_folder.add_backup(backup)

folders_element = xml_element.find('Folders')
for folder in folders_element.findall('Folder'):
    folder_path = folder.text
    work_folder.add_folder(folder_path)

config = Config()
config_element = xml_element.find('Config')
for ext in config_element.findall('Extension'):
    config.add_format(ext.text)
auto_backup_element = config_element.find('Auto_backup')
config.set_autobackup(bool(auto_backup_element.text))

work_folder.set_config(config)
return work_folder

class WorkFolderFile():
    def __init__(self, local_path: str, name: str, cloud_path: str = None):
        self.local_path = local_path
        self.name = name
        if cloud_path:

```

```

        self.cloud_path = cloud_path
    else:
        self.cloud_path = ''

    def get_local_path(self):
        return self.local_path
    def set_local_path(self, path):
        self.local_path = path

    def get_name(self):
        return self.name
    def set_name(self, name):
        self.name = name

    def get_cloud_path(self):
        return self.cloud_path
    def set_cloud_path(self, path):
        self.cloud_path = path

def save_workfolders_to_xml(workfolders, filename):
    root = ET.Element('Root')

    for folder in workfolders:
        folder_element = folder.to_xml()
        root.append(folder_element)

    tree = ET.ElementTree(root)
    tree.write(filename)

def load_workfolders_from_xml(filename):
    tree = ET.parse(filename)
    root = tree.getroot()

    workfolders = []
    for folder_element in root.findall('WorkFolder'):
        workfolders.append(WorkFolder.from_xml(folder_element))

    return workfolders

```

Содержимое файла workFolderTree.py

```

from workFolder import WorkFolder, WorkFolderFile
from myTreeItem import TreeItem
from backup import Backup
from config import Config

class WorkFolderTree():
    def __init__(self, workFolder: WorkFolder, rootTreeItem: TreeItem):
        self.workFolder = workFolder
        self.root = rootTreeItem

```

```

def get_work_folder(self):
    return self.workFolder
def set_work_folder(self, workFolder: WorkFolder):
    self.workFolder = workFolder

def get_root(self):
    return self.root
def set_root(self, root: TreeItem):
    self.root = root

def get_path(self):
    return self.workFolder.get_path()
def set_path(self, path):
    self.workFolder.set_path(path)

def get_name(self):
    return self.workFolder.get_name()
def set_name(self, name):
    self.workFolder.set_name(name)

def get_files(self) -> list:
    return self.workFolder.get_files()
def add_file(self, file: 'WorkFolderFile'):
    self.workFolder.add_file(file)
def del_file(self, file: 'WorkFolderFile'):
    self.workFolder.del_file(file)

def get_backups(self) -> list:
    return self.workFolder.get_backups()
def add_backup(self, backup: Backup):
    self.workFolder.add_backup(backup)
def del_backup(self, backup: Backup):
    self.workFolder.del_backup(backup)

def get_config(self) -> Config:
    return self.workFolder.get_config()
def set_config(self, config: Config):
    self.workFolder.set_config(config)
def add_format(self, format: str):
    self.workFolder.add_format(format)
def remove_format(self, format: str):
    self.workFolder.remove_format(format)

def get_folders(self) -> list:
    return self.workFolder.get_folders()
def add_folder(self, folder_path: str):
    self.workFolder.add_folder(folder_path)
def remove_folder(self, folder_path: str):
    self.workFolder.remove_folder(folder_path)

```

```

def get_backup(self, backupNumber: int) -> Backup:
    return self.workFolder.get_backup(backupNumber)

def get_backup_name_list(self) -> list:
    return self.workFolder.get_backup_name_list()

def get_last_backup(self) -> Backup:
    return self.workFolder.get_last_backup()

def get_backups_path(self):
    return self.workFolder.get_backups_path()
def set_backups_path(self, path: str):
    self.workFolder.set_backups_path(path)

```

Содержимое файла WorkFolderUI2.py

```

import sys
import os
import yadisk

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout,
                              QHBoxLayout, QComboBox,
                              QPushButton, QTextEdit, QMainWindow, QDialog,
                              QLabel, QLineEdit, QFileDialog)
from PyQt5.QtGui import QIcon, QFont
from PyQt5.QtCore import pyqtSlot, pyqtSignal

class ChangeList():
    def __init__(self):
        self.operation_types = ['Create', 'Delete']
        self.changeList = {} # Содержит список имен рабочих
        папок с соответствующим типом операции
        for operation in self.operation_types:
            self.changeList[operation] = []

    def addItem(self, operation:str, item:str):
        self.changeList[operation].append(item)

    def addItems(self, operation:str, items:list):
        for item in items:
            self.addItem(operation, item)

    def return_data(self):
        return self.changeList

    def print(self):
        for operation in self.operation_types:
            print(self.changeList.get(operation))

class CustomDialog(QDialog):
    def __init__(self, parent=None):

```

```

super(CustomDialog, self).__init__(parent)
layout = QVBoxLayout()

buttonLayout = QHBoxLayout()

label = QLabel(text="Имя новой папки:")
self.main_font = QFont('Times New Roman', 14)
self.setStyleSheet("QLabel { background-color: white }")
label.setFont(self.main_font)

self.line_edit = QLineEdit(self)
self.line_edit.setFont(self.main_font)
layout.addWidget(label)
layout.addWidget(self.line_edit)

okButton = QPushButton("OK", self)
cancelButton = QPushButton("Отмена", self)

okButton.clicked.connect(self.okButtonClicked)
cancelButton.clicked.connect(self.cancelButtonClicked)

buttonLayout.addWidget(okButton)
buttonLayout.addWidget(cancelButton)
layout.addLayout(buttonLayout)

self.setLayout(layout)

def okButtonClicked(self):
    self.accept()

def cancelButtonClicked(self):
    self.reject()

def get_input_value(self):
    return self.line_edit.text()

class WorkFolderUI(QMainWindow):
    sendDataSignal = pyqtSignal(int, ChangeList)
    def __init__(self, workFolderList: list, parent = None):
        super().__init__(parent=parent)
        self.work_folder_list = workFolderList
        self.change_list = ChangeList()
        self.status = 0 # 0 - отмена, 1 - OK
        self.init_ui()

    def init_ui(self):
        self.setObjectName("MainWindow")
        self.resize(447, 557)
        self.centralwidget = QtWidgets.QWidget(self)
        self.centralwidget.setObjectName("centralwidget")

        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)

```

```

self.gridLayout.setObjectName("gridLayout")

sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)

# Кнопка добавления папки
self.addFolderButton = QtWidgets.QPushButton(self.centralwidget)
self.addFolderButton.setSizePolicy(sizePolicy)
self.addFolderButton.setObjectName("addFolderButton")
self.addFolderButton.setIcon(QIcon('data/icons/plus_icon.png'))
self.addFolderButton.clicked.connect(self.add_work_folder)
self.gridLayout.addWidget(self.addFolderButton, 1, 2, 1, 1)

# Кнопка удаления папки
self.removeFolderButton = QtWidgets.QPushButton(self.centralwidget)
self.removeFolderButton.setSizePolicy(sizePolicy)
self.removeFolderButton.setObjectName("removeFolderButton")
self.removeFolderButton.setIcon(QIcon('data/icons/minus_icon.png'))
self.removeFolderButton.clicked.connect(self.remove_work_folder)
self.gridLayout.addWidget(self.removeFolderButton, 1, 3, 1, 1)

# Кнопка ОК
self.confirmButton = QtWidgets.QPushButton(self.centralwidget)
self.confirmButton.setObjectName("confirmButton")
self.confirmButton.setText('OK')
self.confirmButton.clicked.connect(self.confirm)
self.gridLayout.addWidget(self.confirmButton, 3, 3, 1, 1)

# Кнопка Отмена
self.cancelButton = QtWidgets.QPushButton(self.centralwidget)
self.cancelButton.setObjectName("cancelButton")
self.cancelButton.setText('Отмена')
self.cancelButton.clicked.connect(self.cancel)
self.gridLayout.addWidget(self.cancelButton, 4, 3, 1, 1)

# Панель информации
self.workFolderInfoList = QtWidgets.QListWidget(self.centralwidget)
self.workFolderInfoList.setObjectName("workFolderInfoList")
self.gridLayout.addWidget(self.workFolderInfoList, 1, 1, 4, 1)
self.setCentralWidget(self.centralwidget)
self.workFolderInfoList.addItem(self.work_folder_list)

# Label
self.label = QtWidgets.QLabel(self.centralwidget)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label.sizePolicy().hasHeightForWidth())
self.label.setSizePolicy(sizePolicy)

```

```

font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(14)
self.label.setFont(font)
self.label.setObjectName("label")
self.gridLayout.addWidget(self.label, 0, 1, 1, 3)
self.label.setText("Список рабочих папок")

self.menubar = QtWidgets.QMenuBar(self)
self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))
self.menubar.setObjectName("menubar")
self.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(self)
self.statusbar.setObjectName("statusbar")
self.setStatusBar(self.statusbar)

QtCore.QMetaObject.connectSlotsByName(self)

@pyqtSlot()
def confirm(self):
    self.status = 1
    self.sendDataSignal.emit(self.status, self.change_list)
    self.close()

@pyqtSlot()
def cancel(self):
    self.status = 0
    self.sendDataSignal.emit(self.status, self.change_list)
    self.close()

# Создание рабочей папки
@pyqtSlot()
def add_work_folder(self):
    dialog = CustomDialog(self)
    result = dialog.exec_()
    new_folder_name = ''

    if result == QDialog.Accepted:
        new_folder_name = dialog.get_input_value()

    if new_folder_name == '':
        return

    # Здесь просто накапливаются изменения. Все изменения
    # обрабатываются в основном классе.
    self.workFolderInfoList.addItem(new_folder_name)
    self.change_list.addItem('Create', new_folder_name)

@pyqtSlot()
def remove_work_folder(self):
    item = self.workFolderInfoList.currentItem()
    name = item.text()
    row = self.workFolderInfoList.row(item)

```



```

        self.workFolderInfoList.takeItem(row)
        self.change_list.addItem('Delete', name)

def get_full_file_name(file_path):
    file_name, file_extension =
os.path.splitext(os.path.basename(file_path))
    return f"{file_name}{file_extension}"

if __name__ == '__main__':
    token = "y0_AgAAAABFABmLAARAjwAAAADwrN9yWawqWRH8Sp2wcpW6h6sv_4QzKhs"
    y = yadisk.YaDisk(token=token)
    app = QApplication(sys.argv)
    ex = WorkFolderUI(y)
    sys.exit(app.exec_())

```

Содержимое файла WorkfolderConfigureUI.py

```

import sys
import os
import yadisk
from backup import Backup

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout,
                              QHBoxLayout, QComboBox,
                              QPushButton, QTextEdit, QMainWindow, QDialog,
                              QLabel, QLineEdit, QFileDialog)
from PyQt5.QtGui import QIcon, QFont
from PyQt5.QtCore import pyqtSlot, pyqtSignal
from PyQt5.QtWidgets import QApplication, QWidget, QListWidget,
                              QListWidgetItem, QHBoxLayout, QCheckBox, QVBoxLayout, QToolButton,
                              QAbstractItemView
from PyQt5.QtCore import QSize, Qt

from config import Config

# Вызывается по кнопке 'Настройка рабочей папки' и после создания новой
# папки
# Принимает на вход список настроек папки.
# Содержит списки форматов файлов, которые можно выбрать для сохранения.
# Имеет кнопки "Выбрать все", ОК и Отмена. "Выбрать все" выбирает все или
# отменяет выбор всего.

class CheckableItem(QWidget):
    stateChangeSignal = pyqtSignal(str, bool)

    def __init__(self, label_text, parent=None):
        super().__init__(parent)

        self.checkbox = QCheckBox()

```

```

        self.checkbox.setFixedSize(QSize(15, 15))
        self.label = QLabel(label_text)
        self.label.setMinimumWidth(50)
        self.label.setFixedHeight(15)

        self.checkbox.stateChanged.connect(self.handle_state_changed)

        layout = QHBoxLayout(self)
        layout.addWidget(self.label)
        layout.addWidget(self.checkbox)

    def isChecked(self):
        return self.checkbox.isChecked()

    def checkItem(self):
        self.checkbox.setChecked(True)

    def uncheckItem(self):
        self.checkbox.setChecked(False)

    def setState(self, state):
        self.checkbox.setChecked(state)

    def handle_state_changed(self, state: bool):
        self.stateChangeSignal.emit(self.label.text(), state)

class CheckBoxListWidget(QListWidget):
    def __init__(self, title: str, parent=None):
        super().__init__(parent)
        self.title = title
        self.setSelectionMode(QAbstractItemView.SelectionMode.NoSelection)

    def addItem(self, label_text):
        item = QListWidgetItem(self)
        widget = CheckableItem(label_text)
        #item.setSizeHint(widget.sizeHint())
        item.setSizeHint(QSize(40, 25))
        self.setItemWidget(item, widget)
        super().addItem(item)

    def get_last_item(self) -> CheckableItem:
        count = self.count()
        if count > 0:
            return self.itemWidget(self.item(count - 1))

    def check_last_item(self):
        count = self.count()
        if count > 0:
            self.itemWidget(self.item(count - 1)).checkItem()

```

```

class ConfigureWorkfolderUI(QDialog):
    def __init__(self, config: Config, found_formats: list, parent=None) ->
None:
        super(ConfigureWorkfolderUI, self).__init__(parent)
        self.config = config

        audio_formats = ['mp3', 'wma', 'wav', 'flac', 'aa', 'aac']
        video_formats = ['avi', 'mp4', 'wmv', 'webm', 'mkv', 'mov']
        image_formats = ['jpeg', 'jpg', 'png', 'webp', 'gif', 'bmp']
        text_formats = ['doc', 'docx', 'xls', 'xlsx', 'txt', 'pdf', 'odt']

        self.titles = []
        self.formats_checkbox_list = []
        self.formats_checkbox_list = self.init_lists(audio_formats,
video_formats, image_formats, text_formats, found_formats)

        self.init_ui()

    def init_lists(self, audio, video, image, text, found):
        audio_checkbox = CheckBoxLayoutWidget('Аудио-форматы')
        self.titles.append(audio_checkbox.title)
        audio_checkbox.addItem(audio_checkbox.title)

audio_checkbox.get_last_item().stateChangeSignal.connect(self.select_all_ch
anged)
        for ext in audio:
            audio_checkbox.addItem(ext)
            if ext in self.config.get_formats():
                audio_checkbox.check_last_item()

        video_checkbox = CheckBoxLayoutWidget('Видео-форматы')
        self.titles.append(video_checkbox.title)
        video_checkbox.addItem(video_checkbox.title)

video_checkbox.get_last_item().stateChangeSignal.connect(self.select_all_ch
anged)
        for ext in video:
            video_checkbox.addItem(ext)
            if ext in self.config.get_formats():
                video_checkbox.check_last_item()

        image_checkbox = CheckBoxLayoutWidget('Форматы изображений')
        self.titles.append(image_checkbox.title)
        image_checkbox.addItem(image_checkbox.title)

image_checkbox.get_last_item().stateChangeSignal.connect(self.select_all_ch
anged)
        for ext in image:
            image_checkbox.addItem(ext)
            if ext in self.config.get_formats():

```

```

        image_checkbox.check_last_item()

        text_checkbox = CheckBoxListWidget('Текстовые форматы')
        self.titles.append(text_checkbox.title)
        text_checkbox.addItem(text_checkbox.title)

    text_checkbox.get_last_item().stateChangeSignal.connect(self.select_all_cha
nged)
        for ext in text:
            text_checkbox.addItem(ext)
            if ext in self.config.get_formats():
                text_checkbox.check_last_item()

        found_checkbox = CheckBoxListWidget('Неизвестные форматы')
        self.titles.append(found_checkbox.title)
        found_checkbox.addItem(found_checkbox.title)

    found_checkbox.get_last_item().stateChangeSignal.connect(self.select_all_ch
anged)
        for ext in found:
            if not ext in audio and not ext in video and not ext in image
and not ext in text:
                found_checkbox.addItem(ext)
                if ext in self.config.get_formats():
                    found_checkbox.check_last_item()

        for ext in self.config.get_formats():
            if not ext in audio and not ext in video and not ext in image
and not ext in text and not ext in found:
                found_checkbox.addItem(ext)
                found_checkbox.check_last_item()

        return([audio_checkbox, video_checkbox, image_checkbox,
text_checkbox, found_checkbox])

def init_ui(self):
    self.resize(1024, 400)

    main_layout = QVBoxLayout(self)
    upper_layout = QHBoxLayout()
    lower_layout = QHBoxLayout()
    lower_layout.setAlignment(Qt.AlignRight)

    main_layout.addLayout(upper_layout)
    main_layout.addLayout(lower_layout)

    for checkbox in self.formats_checkbox_list:
        upper_layout.addWidget(checkbox)

    button1 = QToolButton()
    button1.setText('OK')
    button1.setFixedSize(QSize(80, 25))

```

```

        button1.clicked.connect(self.confirm)
        button2 = QPushButton()
        button2.setText('Отмена')
        button2.setFixedSize(QSize(80, 25))
        button2.clicked.connect(self.cancel)

        lower_layout.addWidget(button1)
        lower_layout.addWidget(button2)

        self.show()

    def confirm(self):
        self.accept()

    def cancel(self):
        self.reject()

    def select_all_changed(self, title, state):
        print(f'Title {title} selected.')
        if title in self.titles:
            for checkbox in self.formats_checkbox_list:
                if title == checkbox.title:
                    count = checkbox.count()
                    for index in range(0, count):
                        widget = checkbox.itemWidget(checkbox.item(index))
                        widget.setState(state)

    def get_data(self) -> Config:
        new_config = Config()
        for checkbox in self.formats_checkbox_list:
            count = checkbox.count()
            for index in range(0, count):
                widget = checkbox.itemWidget(checkbox.item(index))
                if widget.isChecked() and not widget.label.text() in
self.titles:
                    new_config.add_format(widget.label.text())
        return new_config

if __name__ == "__main__":
    app = QApplication(sys.argv)
    conf = Config()
    conf.add_format('wav')
    conf.add_format('txt')
    conf.add_format('unknown')
    ex = ConfigureWorkfolderUI(conf, ['ppp', 'rrr', 'ddd'])
    sys.exit(app.exec_())

```

Содержание файла backup.py

```
from datetime import datetime
```

```

class Backup():
    # Папка бэкапа
    datetime_format = "%d-%m-%Y %H:%M:%S"
    def __init__(self, name: str, path: str, number: int, creation:
datetime):
        self.name = name
        # Название бэкапа (дается
пользователем)
        self.path = path
        # Путь к папке бэкапа на
облаке
        self.number = number
        # Номер бэкапа для этой
рабочей папки
        self.creation_datetime = creation
        # Дата создания бэкапа
        self.deleted_files = []
        # Список файлов, удаленных
с момента прошлого бэкапа. Содержит локальные пути
        self.files = []
        # Список файлов, которые
содержатся в данном бэкапе. Содержит локальные пути

    def get_name(self):
        return self.name
    def set_name(self, name: str):
        self.name = name

    def get_path(self):
        return self.path
    def set_path(self, path: str):
        self.path = path

    def get_number(self):
        return self.number
    def set_number(self, number: int):
        self.number = number

    def get_creation_time(self) -> str:
        return datetime.strftime(self.creation_datetime,
self.datetime_format)
    def set_creation_time(self, creation: datetime):
        self.creation_datetime = creation

    def get_deleted_files(self) -> list:
        return self.deleted_files
    def add_deleted_file(self, file: str):
        self.deleted_files.append(file)
    def remove_deleted_file(self, file: str):
        self.deleted_files.remove(file)

    def get_files(self) -> list:
        return self.files
    def add_file(self, file: str):
        self.files.append(file)
    def add_files(self, files: list):
        for file in files:

```

```

        self.add_file(file)
def remove_file(self, file: str):
    self.files.remove(file)

```

Содержание файла BackupRestoreUI.py

```

import sys
import os
import yadisk
from backup import Backup

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout,
                              QHBoxLayout, QComboBox,
                              QPushButton, QTextEdit, QMainWindow, QDialog,
                              QLabel, QLineEdit, QFileDialog)
from PyQt5.QtGui import QIcon, QFont
from PyQt5.QtCore import pyqtSlot, pyqtSignal

# Вызывается по кнопке "Восстановить данные"
# Получает список бэкапов выбранной рабочей папки.
# Предоставляет выбор бэкапа для восстановления.
# Отображает имеющиеся бэкапы
# Имеет кнопка ОК и Отмена.

class BackupResotoreUI(QDialog):
    sendDataSignal = pyqtSignal(Backup)
    def __init__(self, backups_list: list, parent=None) -> None:
        super(BackupResotoreUI, self).__init__(parent)
        self.backups = backups_list
        self.full_backup = Backup('Full backup', '', 0, '')
        self.backups.append(self.full_backup)
        self.init_ui()

    def init_ui(self):
        self.resize(500, 600)

        self.gridLayout = QtWidgets.QGridLayout(self)
        self.gridLayout.setObjectName("gridLayout")

        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
        QtWidgets.QSizePolicy.Fixed)

        # Кнопка ОК
        self.confirmButton = QtWidgets.QPushButton(self)
        self.confirmButton.setObjectName("confirmButton")
        self.confirmButton.setText('OK')
        self.confirmButton.clicked.connect(self.confirm)
        self.gridLayout.addWidget(self.confirmButton, 3, 3, 1, 1)

        # Кнопка Отмена
        self.cancelButton = QtWidgets.QPushButton(self)

```

```

self.cancelButton.setObjectName("cancelButton")
self.cancelButton.setText('Отмена')
self.cancelButton.clicked.connect(self.cancel)
self.gridLayout.addWidget(self.cancelButton, 4, 3, 1, 1)

# Панель информации
self.backupList = QListWidget(self)
self.backupList.setObjectName("backupList")
self.gridLayout.addWidget(self.backupList, 1, 1, 4, 1)
self.backupList.addItem([x.get_name() for x in self.backups])

# Label
self.label = QLabel(self)
sizePolicy = QSizePolicy(QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label.sizePolicy().hasHeightForWidth())
self.label.setSizePolicy(sizePolicy)
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(14)
self.label.setFont(font)
self.label.setObjectName("label")
self.gridLayout.addWidget(self.label, 0, 1, 1, 3)
self.label.setText("Список резервных копий")

QtCore.QMetaObject.connectSlotsByName(self)

self.show()

def confirm(self):
    self.accept()

def cancel(self):
    self.reject()

def get_input_value(self) -> Backup:
    for backup in self.backups:
        if backup.get_name() == self.backupList.currentItem().text():
            return backup

if __name__ == "__main__":
    app = QApplication(sys.argv)
    ex = BackupResotoreUI([])
    sys.exit(app.exec_())

```

Содержимое файла config.py


```

class Config():
    def __init__(self) -> None:
        self.chosen_formats = []
        self.auto_backup = False

    def get_formats(self) -> list:
        return self.chosen_formats
    def get_autobackup(self) -> bool:
        return self.auto_backup

    def add_format(self, format: str):
        if not format in self.chosen_formats:
            self.chosen_formats.append(format)
    def remove_format(self, format: str):
        if format in self.chosen_formats:
            self.chosen_formats.remove(format)

    def set_autobackup(self, flag: bool):
        self.auto_backup = flag

```

Содержимое файла errorHandler.py

```

from datetime import datetime

class ErrorHandler():

    @classmethod
    def handle(self, errorMessage: str):
        with open('log.txt', 'a') as log:
            cur_date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
            log.write(f"{cur_date} || {errorMessage}")
            log.write('\n')

```