МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

Кафедра программного обеспечения

|  |  |
| --- | --- |
|  | РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК  Заведующий кафедрой  К.т.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С. Воробьева  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

бакалаврская работа

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ И ВОССТАНОВЛЕНИЯ ДАННЫХ

02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль «Технологии программирования»

|  |  |  |
| --- | --- | --- |
| Выполнил работу  студент 4 курса  очной формы обучения |  | Поляков  Игорь  Андреевич |
| Руководитель  доцент |  | Ялдыгин  Валерий  Борисович |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc107498702)

[ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc107498703)

[1.1 ОСНОВНЫЕ ВИДЫ РЕЗЕРВНОГО КОПИРОВАНИЯ 8](#_Toc107498704)

[1.1.1 ПОЛНОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ 8](#_Toc107498705)

[1.1.2 ИНКРЕМЕНТНОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ 10](#_Toc107498706)

[1.1.3 ДИФФЕРЕНЦИАЛЬНОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ 11](#_Toc107498707)

[1.1.4 СРАВНЕНИЕ ВИДОВ РЕЗЕРВНОГО КОПИРОВАНИЯ 13](#_Toc107498708)

[1.2 ХАРАКТЕРИСТИКИ СИСТЕМ РЕЗЕРВНОГО КОПИРОВАНИЯ 14](#_Toc107498709)

[1.3 ХРАНЕНИЕ РЕЗЕРВНЫХ КОПИЙ 15](#_Toc107498710)

[1.4 ВОССТАНОВЛЕНИЕ ДАННЫХ ИЗ РЕЗЕРВНЫХ КОПИЙ 17](#_Toc107498711)

[1.4.1 ВОССТАНОВЛЕНИЕ ДАННЫХ НА ЧИСТОМ КОМПЬЮТЕРЕ 17](#_Toc107498712)

[1.4.2 ПРОВЕРКА АКТУАЛЬНОСТИ РЕЗЕРВНЫХ КОПИЙ 17](#_Toc107498713)

[1.5 КЛИЕНТ-СЕРВЕРНЫЕ МОДЕЛИ РЕЗЕРВНОГО КОПИРОВАНИЯ 18](#_Toc107498714)

[1.6 СРАВНЕНИЕ СУЩЕСТВУЮЩИХ ПРОГРАММНЫХ РЕШЕНИЙ 20](#_Toc107498715)

[ВЫВОД ПО ГЛАВЕ 25](#_Toc107498716)

[ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ И ВОССТАНОВЛЕНИЯ ДАННЫХ 26](#_Toc107498717)

[2.1 РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОЙ МОДЕЛИ ПРИЛОЖЕНИЯ 26](#_Toc107498718)

[2.2 АРХИТЕКТУРА СИСТЕМЫ 28](#_Toc107498719)

[2.3 РЕАЛИЗОВАННЫЕ МОДУЛИ, КЛАССЫ И МЕТОДЫ 30](#_Toc107498720)

[2.4 УСТАНОВКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 41](#_Toc107498721)

[2.5 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 42](#_Toc107498722)

[ВЫВОД ПО ГЛАВЕ 46](#_Toc107498723)

[ЗАКЛЮЧЕНИЕ 47](#_Toc107498724)

[СПИСОК ЛИТЕРАТУРЫ 48](#_Toc107498725)

[ПРИЛОЖЕНИЕ 1. РЕАЛИЗАЦИЯ УПРАВЛЯЮЩЕЙ СЛУЖБЫ 50](#_Toc107498726)

[ПРИЛОЖЕНИЕ 2. РЕАЛИЗАЦИЯ ПОДЧИНЕННОЙ СЛУЖБЫ 56](#_Toc107498728)

[ПРИЛОЖЕНИЕ 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ 66](#_Toc107498730)

[ПРИЛОЖЕНИЕ 4. РЕАЛИЗАЦИЯ КЛАССОВ ДАННЫХ ДЛЯ РАБОТЫ ОСТАЛЬНЫХ КОМПОНЕНТОВ ПРОГРАММЫ 82](#_Toc107498732)

# ВВЕДЕНИЕ

На сегодняшний день информация является одним из важнейших ресурсов и одной из движущих сил прогресса и цивилизации. Информационные технологии активно используются во многих сферах жизнедеятельности человека.

С момента появления разумного человека были придуманы различные методы хранения информации, например зарубки на деревьях, расположение предметов. Существенным развитием этих способов стала письменность – графическое изображение символов на камне, глине, папирусе, бумаге. Большое значение в развитии этого направления имело изобретение книгопечатания. За свою историю человечество приобрело большое количество информации в библиотеках, архивах, периодических изданиях и других разнообразных письменных документах.

В настоящее время объем информации, используемой человеком, настолько велик и разнообразен, что бумажные носители не обеспечивают ее эффективное накопление, хранение и использование. С момента появления первых вычислительных машин получило широкое распространение хранение информации в двоичном коде. Для ее хранения используются различные запоминающие устройства.

Большой объем важной и конфиденциальной информации хранится в цифровом виде. Для ее защиты разработаны различные инструменты шифрования, ограничения прав доступа и резервного копирования.

Резервное копирование в последнее время стало синонимом защиты данных. Защита данных от потери, различных повреждений и других проблем является одной из приоритетных задач для IT компаний.

Разработка приложений для резервного копирования направлена на обеспечение безопасности данных при непредвиденных ситуациях, сбоях аппаратных средств, программного обеспечения.

В настоящее время деятельность значительного большинства предприятий связана с обработкой и хранением данных, а потеря даже части данных может привести к серьезным убыткам и сбоям в работе (вплоть до приостановки деятельности). В связи с этим возникает задача резервирования данных (и их восстановления в случае необходимости). Сегодня существует множество различных видов услуг резервного копирования данных, которые помогают предприятиям, организациям и частным лицам обеспечить безопасность данных и не терять важную информацию в случае стихийных бедствий, кражи или других чрезвычайных ситуаций. Основные причины потерь информации [1] представлены на рисунке 1.



Рис. 1. Основные причины потерь информации

Как правило, в организациях есть несколько источников данных для резервирования, например:

* базы данных;
* файлы и папки с файловых серверов;
* файлы и папки данные с серверов автоматизированных систем;
* данные с компьютеров пользователей.

В связи с резервным копированием, помимо самой задачи создания резервных копий, возникают задачи хранения, управления жизненным циклом и восстановления резервных копий данных. Важным аспектом резервного копирования является вопрос управления и оптимального использования дискового пространства.

При наличии большого количества разнородных источников, для которых требуется резервирование, повышается сложность работы системного администратора по организации резервного копирования и по управлению дисковым пространством.

Для повышения эффективности работы системного администратора по организации и контроля выполнения резервного копирования в случае наличия большого количества и контроля корректности его выполнения целесообразно разработать программное обеспечение, осуществляющее централизованное управление процессом резервирования.

Цель: разработка программного обеспечения для управления процессами создания, хранения, управления жизненным циклом и восстановления резервных копии данных. Целью создания системы является предотвращение потери критически важной информации в случае сбоев или выхода из строя аппаратуры, ошибок программных средств или пользователей, а также злонамеренного уничтожения информации, чем обеспечивается непрерывность технологических процессов и бизнес-процессов предприятия.

Для достижения цели, необходимо решить следующие задачи:

* изучить существующие программные решения по резервному копированию;
* разработать модель клиент-серверного приложения;
* реализовать резервирование и восстановление баз данных Miscrosoft SQL Server и PostgreSQL;
* разработать программное обеспечение, реализующее резервирования и восстановление для наиболее востребованных типов источников (файлы, каталоги, базы данных SQL Server) и позволяющее добавлять новые типы источников.

Для успешной подготовки и защиты выпускной квалификационной работы использовались средства и методы физической культуры и спорта с целью поддержания должного уровня физической подготовленности, обеспечивающую высокую умственную и физической работоспособность. В режим рабочего дня включались различные формы организации занятий физической культурой (физкультпаузы, физкультминутки, занятия избранным видом спорта) с целью профилактики утомления, появления хронических заболеваний и нормализации деятельности различных систем организма.

В рамках подготовки к защите выпускной квалификационной работы автором созданы и поддерживались безопасные условия жизнедеятельности, учитывающие возможность возникновении чрезвычайных ситуаций.

# ГЛАВА 1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 ОСНОВНЫЕ ВИДЫ РЕЗЕРВНОГО КОПИРОВАНИЯ

Перейдем к определению термина резервного копирования по версии SNIA (Storage Networking Industry Association) [2] Резервная копия (англ. backup copy) – данные, хранящиеся на энергонезависимых носителях, чаще удаленно и предназначенные для восстановления информации, в случаях, если оригинальные данные копии потеряны или недоступны. Резервное копирование (англ. backup) – процесс создания резервных копий.

Существует несколько распространенных операций копирования, в зависимости от важности хранимой на компьютере информации и частоты её использования, используют несколько видов резервирования данных [3]:

* полное резервное копирование,
* дифференциальное резервное копирование,
* инкрементное резервное копирование.

Первым этапом решения задачи резервного копирования является структурирование и анализ информации, также ее эффективная обработка и определение объекта хранения информации, приоритет и частота использования, оценка количественных характеристик: общий объем данных.

В жизненный цикл информации входит создание, копирование, использование, хранение (краткосрочное/долгосрочное).

Резервное копирование обеспечивает защиту данных вне зависимости от времени или места обработки.

На сегодняшний день большая часть программных продуктов использует хранение резервных копий в облаке, сетевых хранилищах информации, что позволяет восстановить информацию полностью при потере данных на локальных машинах.

### 1.1.1 ПОЛНОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ

Главный и основополагающий метод создания резервных копий информации, резервная копия содержит всю исходную информацию о файлах, выбранных для резервирования, при таком способе их проще восстановить. Данный способ резервирования данных самый надежный, и к тому же самый затратный [4]. При необходимости резервирования нескольких копий, общий объем в хранилище будет повышаться прямо пропорционально их количеству. Для устранения проблемы расхода памяти используется архивирование, а также комбинация данного метода с другими способами резервирования информации: дифференциальным и инкрементным. Также полное резервирование данных нужно если возникает необходимость создания резервной копии информации для быстрого восстановления чистой системы.

Плюсы способа:

1. Быстрый поиск файлов – так как происходит резервирование всех данных на носителе, то во время поиска файла нет необходимости затрачивать время на просмотр нескольких носителей.
2. Резервная копия данных находится на одном носителе - при возникновении необходимости восстановления системы, вся информация будет содержаться в последней созданной резервной копии.

Минусы способа:

1. Избыточность данных – большая часть файлов системы модифицируются редко, и каждая новое полное резервирование является копией данных, созданных во время первого полного резервного копирования. Этот метод резервирования занимает большие объемы носителей.
2. Полное резервирование занимает много времени – на резервирование данных может уйти достаточно много времени, в случае если используются сетевые устройства для хранения.

Полное резервирование чаще ограничивается ежедневным или еженедельным графиком, но при достаточно большой скорости и емкости носителя появляется возможность значительно снизить временное окно копирования.

Также есть проблема безопасности, полные резервные копии содержат копию всех данных, потому нужно убедиться в использовании шифрования для защиты резервных копий.

### 1.1.2 Инкрементное резервное копирование

Основное отличие от метода полного резервирования данных в том что при инкрементном резервировании происходит копирование не всех данных, а лишь той части, что была модифицирована с даты последнего копирования [5]. Для определения времени резервирования информации используются разные способы, например, в операционной системе Windows применяется подходящий флаг файла (архивный бит), его установка происходит если документ был модифицирован, и обнуляется программой резервирования информации. В других операционных системах применяется время модификации файла. Способ с использованием данного вида резервирования будет неполным, если не делать полное резервирование данных. Во время восстановления данных возникает необходимость сделать восстановление из последней резервной копии, созданной полным резервированием, а затем последовательно восстанавливать информацию из инкрементных копий в порядке их создания. Этот способ применяется для уменьшения расхода памяти на носителях информации, (уменьшить количество ленточных носителей) при создании архивных копий файлов. Данный способ позволяет уменьшить время затрачиваемое на выполнение задачи резервирования, что имеет существенное значение в условиях, если компьютер работает постоянно. Но при инкрементном резервировании есть минус: пошаговое восстановление вернет и удаленные файлы за период резервирование. В качестве примера, по выходным дням используется полное резервирование, а по будням инкрементное. Пользователь создает файл в понедельник, происходит его модификация во вторник, переименование в среду, удаление в четверг. Это значит, что при последовательном восстановлении информации за неделю получится два файла: со старым именем за вторник до модификации, и с новым именем (после переименования), созданным в среду. Это происходит из-за того что в каждой из инкрементной копии хранится различная версия того же файла что и в предыдущей копии, как итог восстановлению будут подлежать все файлы. Потому при последовательном восстановлении информации из архивов нужно резервировать побольше дискового объема памяти, хватило места, в том числе и удаленным файлам, потому как при инкрементном копировании обрабатываются файлы, модифицированные со времени последнего резервирования данных, новая резервная копия будет являться приращением предыдущей.

При восстановлении информации после инкрементного резервирования данных уйдет больше времени по сравнению с остальными методами резервирования, так как нужна полная резервная копия, и все инкрементные, сделанная после последнего полного резервного копирования.

плюсы метода:

* лучшее применение носителей информации – происходит сохранение только файлов модифицированных со времени последнего полного или инкрементного резервирования, резервные копии данных занимают меньшие объемы.
* Меньшее время создания копий – на инкрементное резервное копирование уходит меньше времени, чем на дифференциальное или полное из-за сохранения только модифицированных файлов.

Минусы метода:

* Восстановление файлов происходит более медленно если сравнивать с другими видами резервирования ( возникает необходимость наличия полной резервной копий и последующих инкрементных).

### 1.1.3 Дифференциальное резервное копирование

Главная разница между дифференциальным резервированием и инкрементным резервированием данных [6] в том, что происходит копирование данных модифицированных со времени работы последнего полного резервирования данных. Зарезервированные копии данных перемещаются в архив “нарастая”. В операционных системах Windows такой эффект получается из-за того, что во время дифференциального копирования не происходит сбрасывания архивного бита, в итоге модифицированные данные отправляются в копию архива, пока полное резервирование не делает обнуление архивных битов. Плюсы данного метода в том что, создаваемая таким способом копия, содержит в себе данные из предыдущей копии, это более оптимально по времени для полного восстановления данных. Такому способу нужны всего две резервные копии: последняя дифференциальная и полная резервная копия, потому восстанавливать данные возможно намного быстрее, в сравнении со способом поэтапного восстановления файлов при инкрементном копировании, что является поэтапным. Дифференциальное копирование лишено вышеперечисленных минусов инкрементного, например когда удаленные файлы также возвращаются при полном восстановлении. Это вызывает меньше путаницы с файлами. Есть и минусы - дифференциальное резервирование в экономии требуемого объема носителей значительно проигрывает инкрементному. Это происходит потому как в каждой из новых копий хранятся те же данные что и в предыдущих, суммарно объем резервируемых данных может превзойти или быть равен объему при полном копировании. И при определении расписания резервирования возникает необходимость учета затрат времени на резервирование дифференциальной копии, размеры которой могут превосходить полную копию изначальных данных.

Плюсы:

1. Быстрый поиск зарезервированных файлов - для восстановления системы, зарезервированной с использованием метода дифференциального резервного копирования нужно иметь лишь две зарезервированные копии данных – последнюю дифференциальную резервную копию и последнюю полную резервную копию. Время затрачиваемое на восстановление данных получается намного меньше если сравнивать с видами резервирования, для которых используются полная резервная копия и последующие инкрементные, созданные со времени последнего полного резервирования.
2. Уменьшено время резервирования и восстановления - Дифференциальное резервирование занимает меньше времени, чем если бы использовалось полное резервирование. При полном восстановлении данных существует необходимость только последней дифференциальной копии и полной изначальной копии, быстрее происходит восстановление.

Минусы:

1. Избыточность данных – происходит сохранение всех файлов, модифицированных со времени последнего полного резервирования. Это приводит к тому, что создаются лишние резервные копии данных, созданных при предыдущих дифференциальных резервированиях.

### 1.1.4 Сравнение видов резервного копирования

В Таблице 1 представлен принцип работы основных видов резервного копирования. Происходят 4 процесса резервного копирования. Первый процесс может быть только полным резервным копированием, так как без него инкрементное и дифференциальное резервное копирование невозможно.

Таблица 1

Сравнение видов резервного копирования

|  |  |  |  |
| --- | --- | --- | --- |
| Номер резервного копирования | Полное резервное копирование | Инкрементное копирование | Дифференциальное копирование |
| Backup 1 | Все данные | – | – |
| Backup 2 | Все данные | Изменения после  Backup1 | Изменения после  Backup1 |
| Backup 3 | Все данные | Изменения после  Backup2 | Изменения после  Backup1 |
| Backup 4 | Все данные | Изменения после  Backup3 | Изменения после  Backup1 |

## 1.2 Характеристики систем резервного копирования

Важную роль при выборе систем резервного копирования играют характеристики процессов хранения и резервирования. Их необходимо оговаривать заранее в контракте с поставщиком программного обеспечения. Требования к данным характеристикам описываются в отдельном соглашении об уровне услуг SLA (англ. Service Level Agreement).

При выдвижении требований к характеристикам обычно используются такие термины, как RPO, RTO, Backup Window [7].

Backup Window (окно резервного копирования) – количество времени, затрачиваемое для выполнения операций резервирования данных на целевой системе.

RPO (англ. Recovery Point Objective) – момент времени для восстановления данных. RPO определяет, как часто требуется производить операции резервирования данных и какое количество резервных копий следует хранить.

RTO (англ. Recovery Time Objective) – время для восстановления системы в случае необходимости. RTO имеет низкое значение при существовании копии необходимых данных. Использование технологии CDP тоже характеризуется низким RTO.

RTA (англ. Recovery Time Actual) – действительное время восстановления. Используется в сочетании со временем RTO. Определяется эмпирически, при проведении тестирования.

Data security (безопасность данных) – уровень защиты от неавторизованного доступа к хранимой информации. Это подразумевает как защиту от физического доступа к системам хранения, так и шифрование необходимых данных.

## 1.3 Хранение резервных копий

При создании резервной копии возникает необходимость ее сохранения. Но нет очевидности в том, что конкретно следует хранить и где [8]. Для правильного выбора места сохранения резервной копии, сперва нужно учитывать обстоятельства, в которых в дальнейшем будут использоваться резервируемые данные. Стоит выделить следующие ситуации:

* Восстановление файлов будет происходить по запросу пользователей.
* Полное восстановление в случае какой-либо чрезвычайной ситуации.
* Архивное хранение (возможно не потребуется совсем).

Первые две ситуации вступают в несовместимые противоречия. Если пользователь случайно удалит файл, он захочет восстановить его сразу. А значит носитель с резервной копией данных должен быть в пределах десятка метров от компьютера на котором возникла необходимость восстановления. Но при наступлении чрезвычайной ситуации возникнет необходимость выполнения полного восстановления возможно даже не одного компьютера в используемом центре данных, при том если случившийся сбой носит физический характер, то возможно разрушение не только компьютеров, но и всех резервных копий, что сохранялись на машинах. Вариант архивного хранения файлов менее спорный - вероятность использования системным администратором, довольно низка, потому если резервный носитель хранится далеко от центра данных, это не станет проблемой. Практика показывает, объем редко запрашиваемых данных в архиве составляет примерно 80% от объема всех данных хранящихся на оперативном носителе, при этом около 20% архивных данных не будут востребованы. Отправляя эти данные в архивный носитель, можно освободить до 80% объема на оперативном носителе, что уменьшает объем и окно резервирования данных. Для решения данных задач используются разные методы, зависящие только от потребностей организации. Первый возможный метод заключается в хранении копий за несколько дней локально, а затем в переносе этих копий в более безопасный удаленный носитель информации, после создания новых ежедневных копий. Другой метод заключается содержании пары наборов носителей:

* Набор носителей в центре данных, используется только для восстановления конкретных данных по запросу.
* Набор носителей для удалённого хранения и восстановления в случае наступления чрезвычайных ситуаций.

Наличие двух наборов носителей подразумевает необходимость делать резервирование данных дважды или выполнять их копирование. Это возможно сделать, но применение двойного резервирования данных может занимать больше времени, а для копирования резервных копий могут потребоваться несколько дополнительных носителей для работы с резервируемыми данными (или выделение отдельного компьютера). Сложность для системного администратора здесь заключена в выдерживании баланса между удовлетворением нужд пользователей и наличием резервных копий на случай чрезвычайных ситуаций.

## 1.4 Восстановление данных из резервных копий

Чаще всего резервирование данных выполняется каждый день, а восстановление выполняется реже. Но восстановление файлов неизбежно, в нем точно будет необходимость, потому к нему лучше хорошо подготовиться. В этой задаче имеется важность в анализе двух ситуаций, что возникают при восстановлении зарезервированных данных:

* восстановление данных на чистом компьютере,
* проверка актуальности резервных копий.

### 1.4.1 Восстановление данных на чистом компьютере

Восстановление данных на чистом компьютере - это процесс при котором происходит восстановление полной копии системы на машине, на которой больше нет данных. Здесь выделяется два главных метода к восстановлению на компьютере:

Переустановка, после чего происходит восстановление, здесь базовая операционная система устанавливается также, как и на чистый компьютер. После установки операционной системы и ее правильной настройки, возможно подключение и форматирование оставшихся дисков, после чего восстановить нужные копии с резервных носителей информации.

Диск для восстановления системы - это загрузочный носитель определенного типа (чаще CD-ROM), содержащий минимальное системное окружение (пакет приложений) и позволяющий выполнение самых основных административных задач. Окружение восстановления в основном содержит системные приложения необходимые для работы с дисками вроде разбиения на разделы и форматирования, драйвера устройств, выполняющие обращения к устройству с зарезервированными данными, и программы, выполняющие восстановление данных с резервных носителей информации.

### 1.4.2 Проверка актуальности резервных копий

Все типы копий следует периодически проверять, чтобы убедиться в том, что эти копии можно прочитать и что они являются актуальными на настоящее время. Действительно, иногда копии, по той или иной причине, могут не читаться, чаще всего это обнаруживается только при потере данных, когда требуется резервная копия. Причины этого могут быть самыми разными, например: смещение головки стримера, неправильно настроенная программа резервного копирования и ошибка оператора. Но какова бы не была причина, не проводя периодических проверок, администратор не может быть уверен в том, что действительно есть резервные копии, с которых когда-нибудь позже можно будет восстановить данные.

## 1.5 Клиент-серверные модели резервного копирования

На сегодняшнее время выделяются две клиент-серверные модели для резервирования данных:

* Централизованная модель,
* Децентрализованная модель.

При централизованной модели используется четкая иерархическая модель, изображенная на рисунке 2, на каждый компьютер устанавливается клиентское приложение-агент, а на сервер приложение-сервер для управления заданиями клиентов. По такому принципу работает большая часть популярных систем резервирования данных на сегодняшний день.

Принцип работы: в серверном приложении настраивается план резервирования, и дополнительные параметры, затем сервер передает агентам клиентов команды для выполнения резервного копирования, а агенты, следуя командам выполняют задания резервирования и отправки на сервер, восстановления данных, передают серверу сообщения об ошибках.

Преимущества:

* масштабируемость,
* отказоустойчивость,
* удобное управление.

Недостатки:

* сложность реализации.

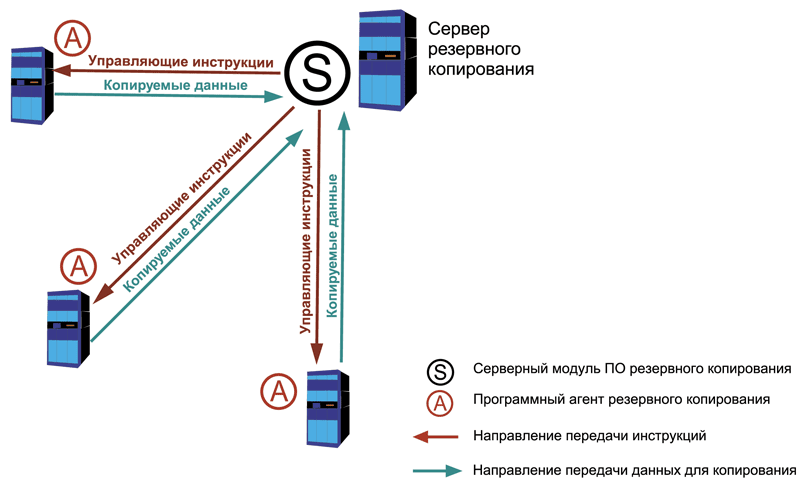


Рис. 2. Централизованная модель резервирования данных.

Децентрализованная модель построена так, что сервером модели является сетевой общий ресурс, как пример сетевая папка или FTP-сервер, рисунок 3. Клиентом является программа для резервного копирования, которая создает резервные копии данных и отправляет их на сервер. Используется в небольших сетях организаций, и при отсутствии возможности реализации централизованной модели резервного копирования имеющимися средствами.

Преимущества:

* простота реализации,
* низкая стоимость.

Недостатки:

* нет масштабируемости,
* сложность администрирования.

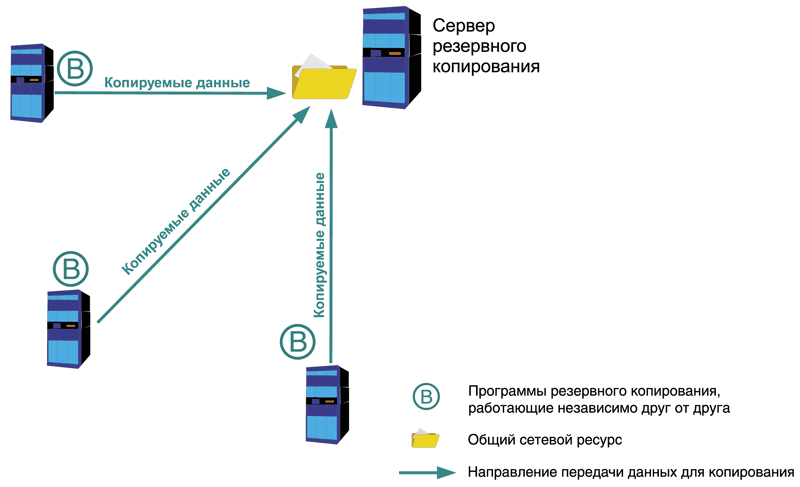


Рис. 3. Децентрализованная модель резервирования данных.

При рассмотрении данных моделей можно сделать вывод, что следует применять централизованную модель если возможна необходимость дальнейшего расширения, в остальных случаях можно ограничиться децентрализованной.

## 1.6 Сравнение существующих программных решений

Чтобы в случае необходимости была возможность быстро восстановить файлы, существует программы для резервирования. Они имеют функции архивирования, генерации образов диска или виртуальных машин. Созданные образы можно хранить локально на компьютере или на FTP-сервере, в облачном хранилище. Рынком представлено множество продуктов, обеспечивающих резервное копирование и восстановление данных, кратко ознакомимся с функциями каждой из программ. К рассмотрению было выбрано несколько популярных решений:

* Acronis True Image 2021,
* Veeam Backup & Replication Community Edition,
* EaseUS Todo Backup 2022.

**Acronis True Image 2021**

Acronis True Image 2021 – интегрированный пакет программ, обеспечивает безопасность данных компьютера. Есть возможности создания резервных копий файлов, электронной почты, разделов или диска. Функция резервного копирования в облако позволяет хранить файлы в Acronis Cloud. Также поддерживает файловые системы Linux. При создании копий следит за версиями файлов и предотвращается создание одинаковых объектов. Возможно создать изолированную область для хранилища, есть поддержка шифрования. Интерфейс изображен на рисунке 4.

Стоимость: 1700.

Системные требования:

* Процессор Intel CORE 2 Duo (2 ГГц) или аналогичный ЦП должен поддерживать инструкции SSE,
* 2 ГБ ОЗУ,
* 7 ГБ свободного пространства на системном жестком диск,
* дисковод CD-RW/DVD-RW или USB-накопитель для создания загрузочного носителя (для Linux требуется около 660 МБ свободного пространства, для Windows требуется около 700 МБ свободного пространства),
* разрешение экрана 1024 x 768.

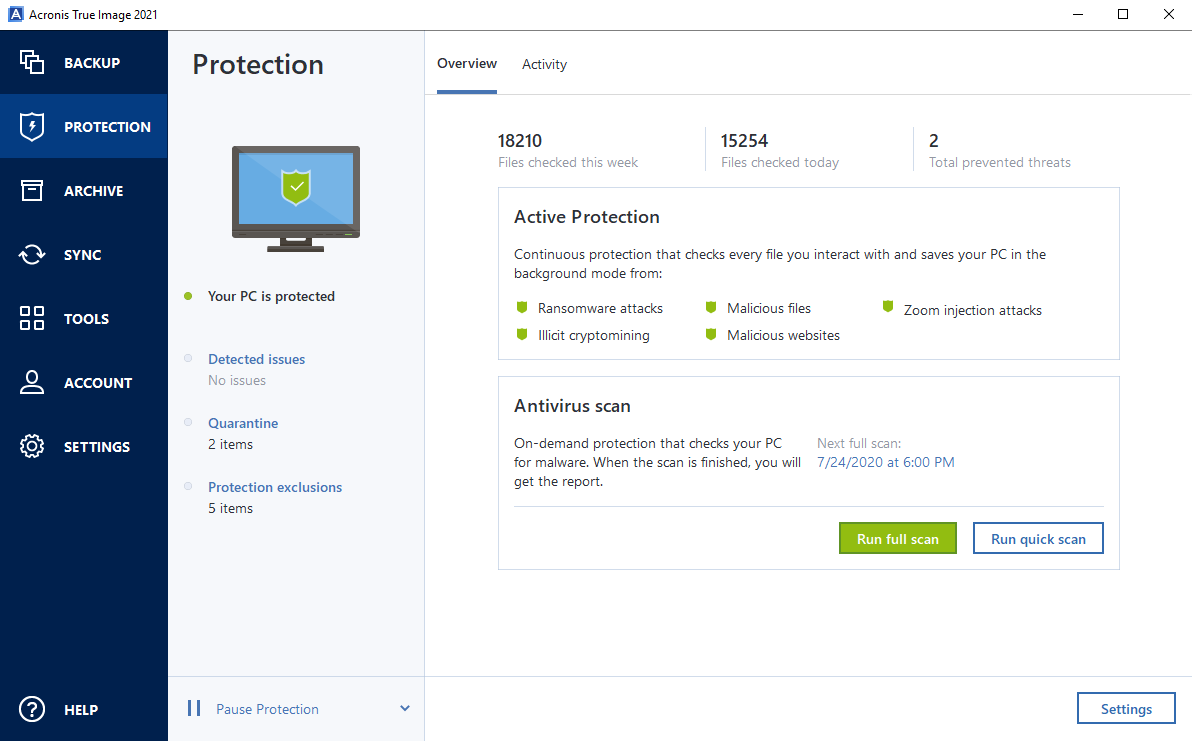


Рис. 4. Интерфейс Acronis True Image 2021

**Veeam Backup & Replication Community Edition**

Veeam Backup & Replication Community Edition – бесплатная программа резервного копирования и восстановления данных локальных и удаленных компьютеров, ориентирована на корпоративных пользователей. Есть функция создания резервных копий баз данных SQL и Oracle. Поддерживается резервирование отдельных пользовательских папок. Управление заданиями происходит через консоль администрирования. Изображение интерфейса на рисунке 5.

Стоимость: бесплатно.

Системные требования:

* процессор: х86-64,
* память: 4Гб и 500Мб для каждого выполняемого задания,
* жесткий диск: 2ГБ для установки и 10ГБ на каждые 100 виртуальных машин,
* сеть: Ethernet 1Гбит/с.

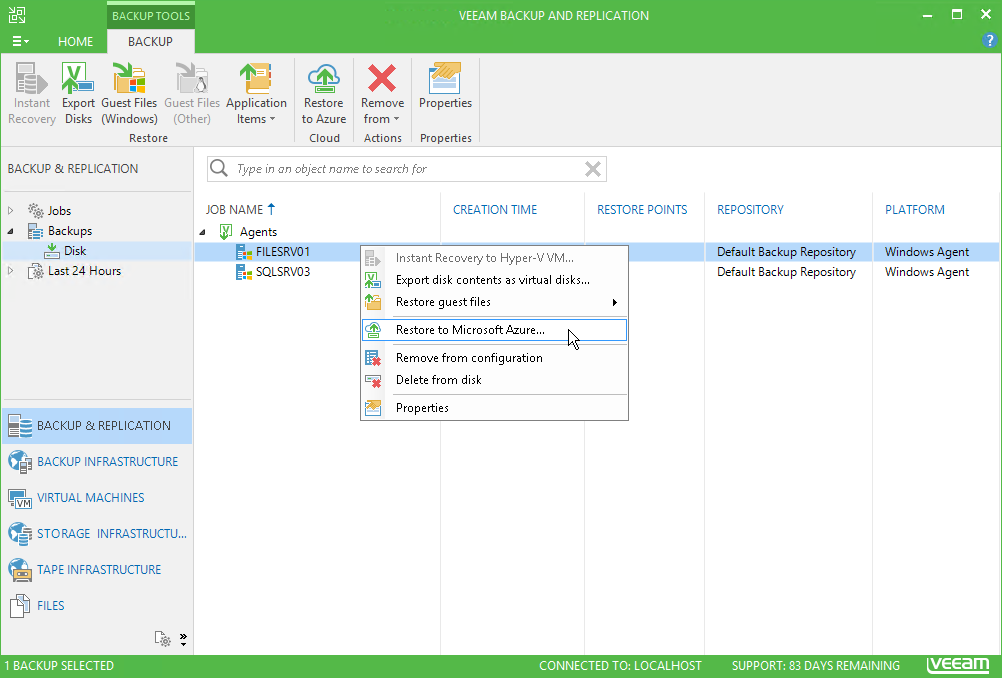


Рис. 5. Интерфейс Veeam Backup & Replication Community Edition

**EaseUS Todo Backup 2022**

EaseUS Todo Backup 2022 – популярная программа резервирования и восстановления данных. Позволяет переносить данные системы на другие диски, поэтому часто используется для миграции с использованием твердотельных SSD накопителей. Есть функция создания автозагрузочного образа и копий отдельных объектов. Выше перечислены возможности бесплатной версии, в ней нет интегрированного планировщика. Изображение интерфейса на рисунке 6.

В расширенной версии программы появляются возможности хранения резервных данных на FTP-серверах, архивирования почтовой базы, уведомления о выполнении заданий на почту.

Стоимость: бесплатно с возможностью приобрести расширенные версии.

Системные требования:

* тактовая частота процессора не менее 500 МГц,
* не менее 1 ГБ ОЗУ для ОС MS Windows XP/Vista, не менее 2 ГБ ОЗУ для ОС MS Windows 8/8.1/10,
* 1 ГБ свободного пространства на жестком диске для установки программы,
* привод CD-RW/DVD-RW или накопитель USB для создания загрузочных носителей.

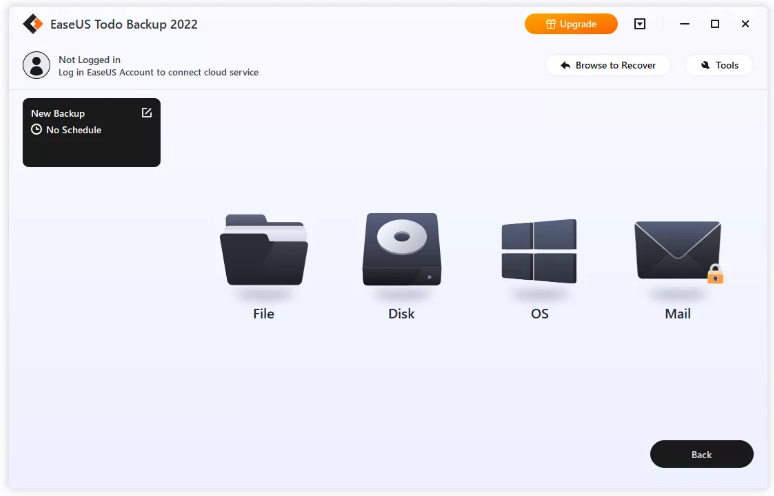


Рис. 6. Интерфейс EaseUS Todo Backup 2022

В таблице 2 приведено сравнение программных продуктов

Таблица 2

Сравнение программ резервирования данных

|  |  |  |  |
| --- | --- | --- | --- |
| Возможности программы | Acronis True Image 2021 | Veeam Backup & Replication Community Edition | EaseUS Todo Backup 2022 |
| Расписание резервного копирования | + | + | ＋ |
| Сетевое резервное копирование | + | + | + |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| Русскоязычный интерфейс | + | - | - |
| Лицензия | Пробная | Бесплатно | Бесплатно, есть расширенные версии |
| Поддержка FTP-сервера | + | + | + |
| Резервирование баз данных | - | Частично | - |

На основе рассмотренных характеристик и описания, оптимальным выбором будет использование Veeam Backup & Replication Community Edition, функциональность аналогична Acronis True Image 2021, при этом распространение бесплатно.

## Вывод по главе

В данной главе были рассмотрены технологии резервного копирования и хранения данных, приведены их преимущества и недостатки.

На основании исследованных технологий, проведено сравнение трех распространенных продуктов для резервного копирования данных. Рассмотрение вышеперечисленных продуктов показывает целесообразность использования бесплатного ПО для резервирования данных.

В главе приведено сравнение трех популярных приложений для резервирования файлов, успешно использующихся организациями для защиты данных.

# ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ И ВОССТАНОВЛЕНИЯ ДАННЫХ

## 2.1 Разработка клиент-серверной модели приложения

На основе обзора решений и технологий в области резервирования данных в Главе 1, мною разработана централизованная модель приложения, осуществляется полное резервное копирование.

Выбор модели аргументирован следующим:

* Централизованная модель обладает большей масштабируемостью, появляется возможность продолжения улучшениями программы.
* Полное резервное копирование позволяет проводить быстрое восстановление данных и имеет простое управление благодаря содержанию в одной резервной копии, что снижает количество возможных ошибок при копировании.

В качестве среды разработки используется Visual Studio 2022, в качестве языка программирования используется объектно-ориентированный язык программирования C#.

Для создания интерфейса приложения администратора используется технология Windows Forms (WinForms).

Целевая платформа – NET 6.0.

При разработке исходный код проекта сохранялся на GitHub (адрес репозитория: <https://github.com/polyakovGit/BackupSystem>).

Решение включает 5 проектов:

* ServerService – управляющая служба, реализована в виде службы Windows.
* ClientService – подчиненная служба, также реализована в виде службы Windows.
* DesktopClient – приложение с графическим интерфейсом. С его помощью пользователь может добавлять, редактировать, удалять, а также мониторить список задач.
* SharedData – библиотека классов, компилируемая в DLL и используемая остальными компонентами системы.
* PgStore – Библиотека используемая для работы с базами данных PostgreSQL.

Серверная служба обладает следующими функциями:

* Управление расписанием резервирования для каждого из клиентов.
* Прием резервных копий от клиентов и сохранение в общее хранилище.
* Восстановление данных.
* Присвоение резервной копии временной метки.
* Хранение и обработка ошибок с записью в лог.

Подчиненная служба управляет выполнением заданий, полученных от серверной службы, и передает данные на сервер для резервирования или принимает для восстановления.

Клиентское приложение обладает следующими функциями:

* Создание заданий для резервирования данных и управление их жизненным циклом.
* Получение отчетов о статусе выполнения от подчиненных служб резервирования.

Взаимодействие между приложениями осуществляется по сети по протоколу TCP/IP. Для этих целей используется библиотека Network [9]. Библиотека позволяет посредством сети получать и отправлять пакеты с различными данными. Для передачи списка заданий используется пакет ‘tasks’. Передача файлов осуществляется с помощью пакета ‘backup’.

Управляющая служба сохраняет список заданий в файле Tasks.json. Для сохранения списка в формате JSON применяется библиотека Newtonsoft.Json [10].

## 2.2 Архитектура системы

Архитектура разрабатываемой системы приведена на рисунке 7.

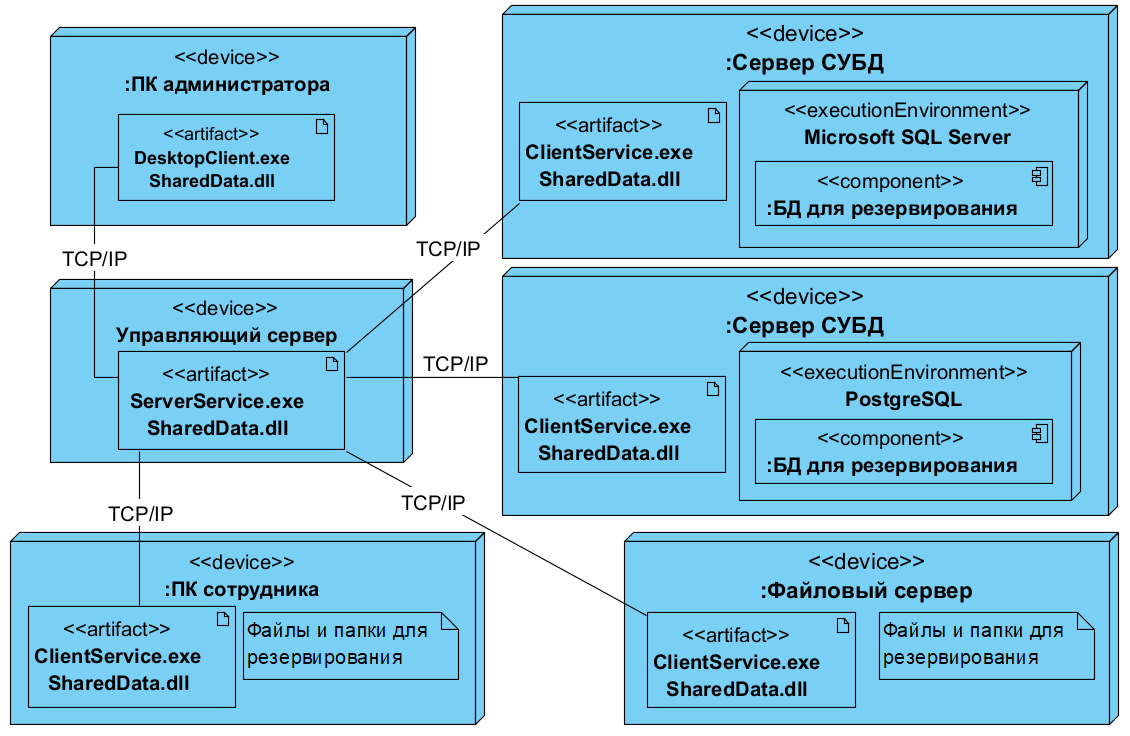


Рис. 7. Архитектура системы

Центральным звеном системы является управляющий сервер резервного копирования, на котором запущена управляющая служба. Управляющая служба прослушивает порт, который задается при установке службы (по умолчанию порт 1708). В качестве управляющего сервера можно использовать выделенный сервер или использовать сервер, уже задействованный под другие задачи (если вычислительные мощности позволяют).

К управляющему серверу подключено хранилище (например, RAID-массив или система хранения данных). При выполнении резервного копирования производится запись в хранилище, при восстановлении и проверке наличия резервных копий производится чтение из хранилища. Кроме того, периодически производится удаление устаревших резервных копий из хранилища.

На серверах и персональных компьютерах, данные которых следует резервировать, устанавливаются подчиненные службы (порт также задается при установке, по умолчанию также используется порт 1708). Подчиненные службы взаимодействуют только с управляющей службой, при этом от управляющей службы к подчиненным службам передаются *задания на резервирование* (информация о заданиях включает идентификатор, информацию о резервируемом объекте и расписании). Управляющая служба передает подчиненным службам подмножества заданий, относящиеся к соответствующим подчиненным службам. От подчиненных служб к управляющей службе передается *информация о выполнении заданий (и произошедших при этом ошибках)*. Сами резервные копии передаются от подчиненной службы к управляющей при резервировании и в обратном направлении (от управляющей службы к подчиненным) при восстановлении.

Управление заданиями на управляющем сервере (и, следовательно, заданиями на подчиненных службах) выполняется из приложения, запускаемого на ПК системного администратора. Для удобства данная программа может быть установлена на нескольких ПК, также целесообразно для удобства установить ее на управляющий сервер. При подключении к управляющей службе производится идентификация и аутентификация пользователя (данные для идентификации и аутентификации хранятся на управляющем сервере). Из АРМ администратора можно создавать задания, просматривать и редактировать параметры созданных заданий, просматривать информацию о результатах выполнения заданий, а также удалять задания. После изменения заданий информация об изменениях сразу отправляется на подчиненные службы. Информация о выполнении, изменении или удалении задания, сохраняется на сервере (действия пользователей записываются в лог). Информация о задания хранится в файле Tasks.json.

## 2.3 Реализованные модули, классы и методы

При разработке приложения были реализованы следующие модули и классы:

Модуль ClientConfig. Библиотека для работы с параметрами подключения клиентских приложений.

Класс Config. Класс для работы с параметрами подключения клиентов. Описание методов в таблице 2.

Таблица 2

Описание методов класса Config.

|  |  |
| --- | --- |
| Методы | Описание |
| public Config () | конструктор по умолчанию |
| public void SaveToFile(string filename) | Сохраняет параметры подключения в указанный файл в формате JSON |
| public async Task SaveToFileAsync(string filename) | Асинхронно сохраняет параметры подключения в указанный файл в формате JSON |
| public static Config LoadFromFile(string filename) | Загружает параметры подключения из указанного файла |
| public async static Task<Config> LoadFromFileAsync(string filename) | Асинхронно загружает параметры подключения из указанного файла |

Модуль ClientService. Служба, работает на клиентском компьютере и выполняет задания по резервированию.

Класс WinService. Описание методов в таблице 3.

Таблица 3

Описание методов класса WinService.

|  |  |
| --- | --- |
| Методы | Описание |
| public WinService() | конструктор по умолчанию |
| protected override async void OnStart(string[] args) | асинхронный метод, выполняется при запуске службы |
| protected override void OnStop() | выполняется при остановке службы |
| private async Task Connect() | выполняет подключение к серверу |
| private async void RecvHandler(SharedRequest packet, Connection connection) | обработчик пакетов, полученных от сервера по сети |
| private async Task Handler() | основной рабочий цикл службы. Проверяет задания и выполняет их |

Модуль DesktopClient. Графическое приложение для просмотра и управления заданиями.

Класс Globals. Содержит глобальные данные приложения и выполняет действия, не связанные с графическим интерфейсом. Описание методов в таблице 4.

Таблица 4

Описание методов класса Globals.

|  |  |
| --- | --- |
| Методы | Описание |
| public static bool Init() | выполняет инициализацию приложения (запуск службы и подключение к серверу) |
| public static bool Connect() | подключение к серверу |
| private static async void RecvHandler(SharedRequest packet, Connection connection) | обработчик входящих от сервера пакетов |

Продолжение таблицы 4

|  |  |
| --- | --- |
| public static async void SendLogin(string username, string password) | отправляет пакет Login на сервер по сети. |
| public static async void SendTasks() | отправляет пакет tasks на сервер по сети |
| public static async void SendRestore(int id) | отправляет пакет restore на сервер по сети. |
| public static void LoadConfig() | загружает параметры подключения из файла конфигурации |
| public static void SaveConfig() | сохраняет параметры подключения в файл конфигурации |

Класс Login. Форма с параметрами подключения. Описание методов в таблице 5.

Таблица 5

Описание методов класса Login.

|  |  |
| --- | --- |
| Методы | Описание |
| public Login() | конструктор по умолчанию |
| private void buttonConnection\_Click(object sender, EventArgs e) | обработчик нажатия на кнопку ‘Подключиться’ |
| public void Clear() | выполняет очистку полей с логином и паролем в форме |
| private void Login\_Load(object sender, EventArgs e) | выполняется при загрузке формы |
| private void Login\_FormClosing(object sender, FormClosingEventArgs e) | выполняется при закрытии формы |

Класс Main. Основное окно приложения. Описание методов в таблице 6.

Таблица 6

Описание методов класса Main.

|  |  |
| --- | --- |
| Методы | Описание |
| public Main() | конструктор по умолчанию |
| public void UpdateTable(TasksInfo tasks) | обновляет список заданий в окне согласно переданному параметру |
| private void buttonAddFile\_Click(object sender, EventArgs e) | Открывает диалог добавления резервирования файла. Создает задание по резервированию |
| private void buttonEdit\_Click(object sender, EventArgs e) | Открывает диалог для редактирования задания |
| private void buttonDelete\_Click(object sender, EventArgs e) | Удаляет выбранное задание |
| private void listView1SelectedIndexChanged(object sender, EventArgs e) | При изменении выбранного создания в списке активирует или деактивирует часть кнопок |
| private void Main\_Load(object sender, EventArgs e) | Выполняется при загрузке окна. Запускает диалог с формой Login |
| private void buttonAddDb\_Click(object sender, EventArgs e) | Открывает диалог для добавления задания по резервированию БД |
| private void buttonRestore\_Click(object sender, EventArgs e) | Отправляет на сервер команду для восстановления файла или БД |
| private void buttonDisable\_Click(object sender, EventArgs e) | Включает или отключает выбранное задание |
| private void buttonQuota\_Click(object sender, EventArgs e) | Открывает диалог настройки квоты |

Продолжение таблицы 6

|  |  |
| --- | --- |
| private void buttonHistory\_Click(object sender, EventArgs e) | Открывает диалог отображения истории выбранного задания |

Класс Quota. Форма для настройки параметров квоты. Описание методов в таблице 7.

Таблица 7

Описание методов класса Quota.

|  |  |
| --- | --- |
| Методы | Описание |
| public Quota() | конструктор по умолчанию. Заполняет поля формы необходимыми значениями |

Класс TaskDatabaseEdit. Форма для добавления и редактирования задания по резервированию БД. Описание методов в таблице 8.

Таблица 8.

Описание методов класса TaskDatabaseEdit.

|  |  |
| --- | --- |
| Методы | Описание |
| public TaskDatabaseEdit() | конструктор по умолчанию |
| private void buttonConnect\_Click(object sender, EventArgs e) | обработчик нажатия на кнопку ‘Подключиться’. Выполняет соединение с БД и заполняет выпадающий список именами баз данных |
| public DbBackupTask GetTask() | Создает класс задания из параметров полей в форме |
| public void SetTask(DbBackupTask task) | Заполняет поля на форме параметрами из класса DbBackupTask |

Класс TaskFileEdit. Форма для добавления и редактирования задания по резервированию файлов. Описание методов в таблице 9.

Таблица 9

Описание методов класса TaskFileEdit.

|  |  |
| --- | --- |
| Методы | Описание |
| public TaskFileEdit() | конструктор по умолчанию |
| private void buttonSelect\_Click(object sender, EventArgs e) | открывает стандартный диалог выбора файла |
| public FileBackupTask GetTask() | Создает экземпляр класса FileBackupTask из полей формы |
| public void SetTask(FileBackupTask task) | Заполняет поля на форме параметрами из класса FileBackupTask |

Класс TaskHistory. Форма для отображения истории задания. Описание методов в таблице 10.

Таблица 10.

Описание методов класса TaskHistory.

|  |  |
| --- | --- |
| Методы | Описание |
| public TaskHistory() | конструктор по умолчанию |
| public void UpdateHistoty(BackupTask task) | заполняет параметры таблицы данными из истории задания |

Модуль ServerService. Служба сервера.

Класс Server. Описание методов в таблице 11.

Таблица 11

Описание методов класса Server.

|  |  |
| --- | --- |
| Методы | Описание |
| public async Task Listen() | принимает входящие подключения от клиентов |
| async void LoadUsers() | загружает данные по пользователям из файла |
| private async void HandlerCommand(SharedRequest packet, Connection connection) | обработчик входящих по сети пакетов |
| void CheckBackups() | Проверяет папку с бекапами. Удаляет лишние бекапы |
| long GetFolderSize(DirectoryInfo di) | вычисляет размер папки |
| void SendLoginState(bool logged, Connection connection) | отправляет состояние авторизации по сети клиенту |
| protected override async Task ExecuteAsync(CancellationToken stoppingToken) | выполняется при запуске службы |
| protected internal void Disconnect() | выполняется при остановке службы |

Модуль SharedData. Содержит классы данных, которые пересылаются по сети между модулями.

Класс FilesInfo. Служит для передачи файлов по сети. Содержит список с информацией по файлам. Описание методов в таблице 12.

Таблица 12

Описание методов класса FilesInfo.

|  |  |
| --- | --- |
| Методы | Описание |
| public FilesInfo() | конструктор по умолчанию |
| private FilesInfo(List<FileStruct> binFiles) | конструктор, создает экземпляр класса из списка файлов |
| public void Add(int id, DateTime date, string nameFile, byte[] bin) | добавляет файл и информацию в список |
| public byte[] ToArray() | преобразовывает список в массив байт для передачи по сети |
| public static FilesInfo FromBin(byte[] bin) | создает экземпляр класса из массива байт |

Класс SharedRequest – класс для отправки запроса по сети.

Класс SharedResponse – класс для отправки ответа по сети.

Класс BackupTask – абстрактный класс задания по резервированию. Описание методов в таблице 13.

Таблица 13

Описание методов класса BackupTask.

|  |  |
| --- | --- |
| Методы | Описание |
| public void UpdateNextBackupTime() | вычисляет и обновляет дату и время следующего бекапа |
| public string GetStatusString() | возвращает текстовую строку на основе статуса задания |
| public string GetScheduleString() | – возвращает текстовую строку на основе расписания задания |
| public void AddAction(TaskAction action) | добавляет действие в историю задания |

Класс TaskInfo – содержит список всех заданий по резервированию. Описание методов в таблице 14.

Таблица 14

Описание методов класса TaskInfo.

|  |  |
| --- | --- |
| Методы | Описание |
| public byte[] ToArray() | преобразовывает содержимое класса в массив байт для передачи по сети |
| public static TasksInfo FromArray(byte[] array) | создает экземпляр класса из массива байт |
| public void SaveToFile(string filename) | сохраняет содержимое класса в файл в формате JSON |
| public async Task SaveToFileAsync(string filename) | асинхронно сохраняет содержимое класса в файл в формате JSON |
| public static TasksInfo LoadFromFile(string filename) | создает экземпляр класса на основе содержимого файла в формате JSON |

Продолжение таблицы 14

|  |  |
| --- | --- |
| public int GetNextId() | возвращает Id для новой задачи |

В качестве базового класса для заданий определен абстрактный класс BackupTask. Данный класс содержит общие свойства для всех задач:

* числовой идентификатор задания (поле Id);
* адрес сервера, на котором находится подчиненная служба (поле Address);
* порт, на котором находится подчиненная служба (поле Port);
* дата и время последнего резервирования (поле LastBackupTime);
* дата и время следующего резервирования (поле NextBackupTime);
* тип бекапа (поле TypeTimeBackup);
* статус бекапа (поле Status, тип – перечисление TaskStatus);

Существуют следующие типы статуса (перечисление TaskStatus):

* TaskStatus.New – новое задание;
* TaskStatus.Working – задание выполняется;
* TaskStatus.Error\_NoFile – указанного файла нет.
* TaskStatus.Error\_DbConnect – ошибка соединения с сервером SQL.
* TaskStatus.Restored – восстановлено.

Для конкретных типов резервирования определены потомки базового класса. В настоящее время реализовано 3 типа резервирования: резервирование файлов (класс FileBackupTask) и резервирование баз данных SQL Server (класс SQLBackupTask), PostgreSQL (класс PGBackupTask).

В случае добавления нового типа резервирования необходимо создать еще одного потомка для класса BackupTask, при этом общий функционал будет работать сразу.

Класс FileBackupTask (прямой потомок класса BackupTask) определяет задание резервирования файловой системы. В данном классе дополнительно определено поле FileName – путь к резервируемому файлу или папке (если путь заканчивается на слеш, то это папка, иначе файл).

Класс SQLBackupTask (также прямой потомок класса BackupTask) определяется задание резервирования базы данных SQL Server. В данном классе дополнительно определены следующие поля:

* Server – имя или ip-адрес сервера;
* DbName – название базы данных;
* Login – логин для доступа к базе данных;
* Password – пароль для доступа к базе данных.

Класс PGBackupTask (потомок класса BackupTask) определяется задание резервирования базы данных PostgreSQL. В данном классе дополнительно определены следующие поля:

* Host – имя или ip-адрес сервера;
* DbName – название базы данных;
* UserId – логин для доступа к базе данных;
* Password – пароль для доступа к базе данных;
* Port – порт подключения сервера PostgreSQL.

Диаграмма классов заданий резервирования приведена на рисунке 8:

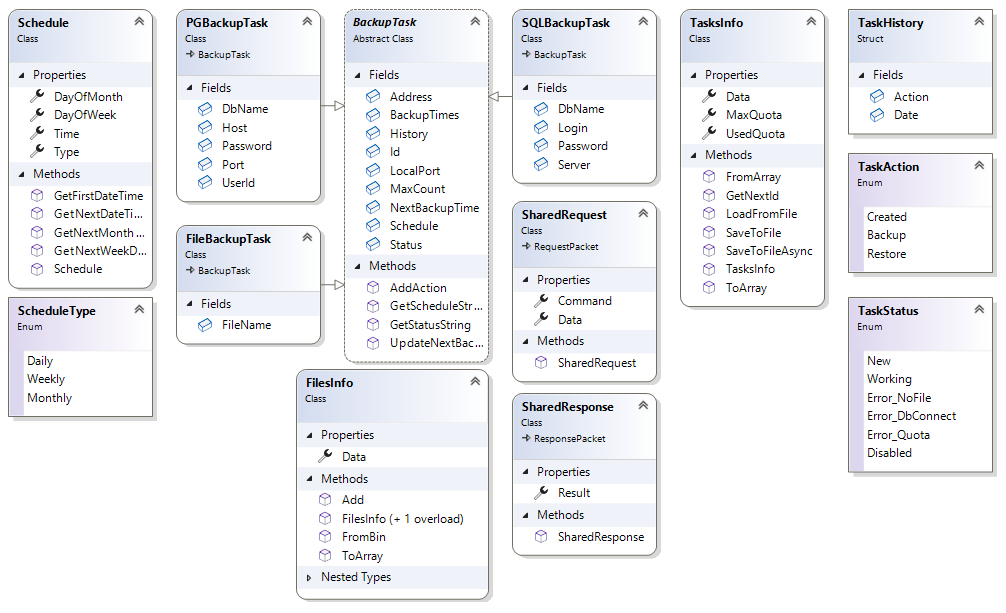


Рис. 8. Диаграмма классов заданий резервирования

В приложении 1 приведен основной код, реализующий управляющую службу.

В приложении 2 приведен основной код, реализующий подчиненную службу.

В приложении 3 приведен основной код, реализующий АРМ администратора.

В приложении 4 приведен основной код, отвечающий за передачу данных на сервер.

## 2.4 Установка программного обеспечения

Для установки системы необходимо выполнить установку следующих компонентов.

* Управляющая служба. Необходимо скопировать на управляющий сервер каталог ServerService и выполнить установку службы запуском командного файла MainService.bat из данного каталога.
* Подчиненная служба. Устанавливается на серверах и персональных компьютерах, данные которых необходимо резервировать. Необходимо скопировать на каждый из таких объектов каталог ClientService и выполнить установку службы запуском командного файла MainService.bat из данного каталога.
* АРМ администратора. Устанавливается на ПК системных администраторов и на управляющем сервере. Установка не требуется, необходимо скопировать каталог DesktopClient.

Удаление компонентов производится следующим образом:

* Управляющая служба. Для удаления необходимо от имени администратора в режиме командной строки выполнить команду sc delete ServerService

Далее следует удалить каталог ServerService.

* Подчиненная служба. Для удаления необходимо от имени администратора в режиме командной строки выполнить команду

sc delete ClientService

Далее следует удалить каталог ClientService.

* АРМ администратора. Для удаления необходимо удалить каталог DesktopClient.

## 2.5 Описание программного обеспечения

Управление заданиями выполняется из АРМ администратора. Для работы АРМ администратора необходимо, чтобы была запущена управляющая служба. Кроме того, на сетевом оборудовании и на брандмауэрах должно быть разрешено взаимодействие (отправитель – АРМ администратора, динамический порт, диапазон 49152 – 65535, согласно [11]; получатель – управляющий сервер, порт, настроенный при установке, по умолчанию порт 1708).

При запуске приложения откроется окно, изображенное на рисунке 9.



Рис. 9. Начальное окно программы

В поле «Сервер» необходимо указать имя или ip-адрес управляющего сервера, порт (по умолчанию 1708), логин и пароль. Если сервер или номер порта указаны неверно, будет выдано сообщение об ошибке (рисунок 10):

Изображение выглядит как текст

Автоматически созданное описание

Рис. 10. Сообщение об ошибке при подключении к серверу

Если возникнет ошибка при аутентификации, будет выдано следующее окно (рисунок 11):



Рис. 11. Сообщение об ошибке при вводе логина / пароля.

Если подключение выполнено успешно, значения, указанные в полях «Сервер», «Порт» и «Логин» сохраняются (при следующем запуске потребуется ввести только пароль).

В случае успешного соединения с сервером пользователь видит интерфейс со списком задач, полученный от управляющей службы. Интерфейс представлен на рисунке 12.

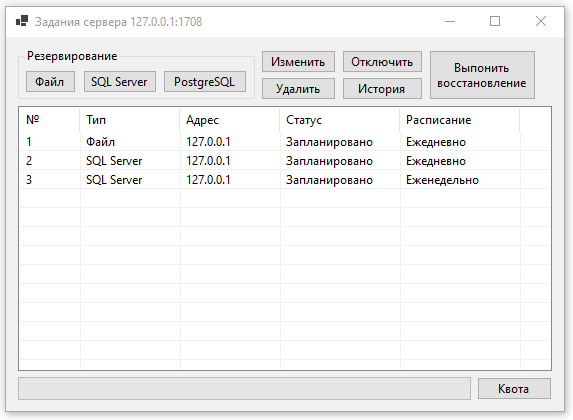


Рис. 12. Окно управления заданиями на резервирование

В данном окне можно создать, редактировать или удалить задание. В списке отображается тип задачи, адрес сервера, статус выполнения и расписание, где указан период выполнения.

Для создания задания необходимо нажать на кнопку создания соответствующего задания.

При создании задания резервирования файла нужно указать путь к файлу или каталогу, который будет сохраняться, также следует выбрать один из трех вариантов периодичности выполнения задания (ежедневно, еженедельно или ежемесячно) и указать время выполнения. Интерфейс создания и редактирования задачи изображен на рисунке 13.

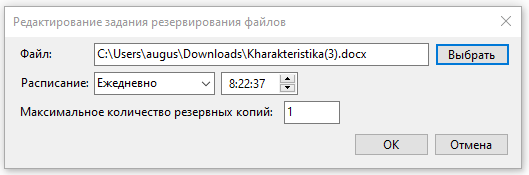


Рис. 13. Интерфейс создания и редактирования задачи файла.

В случае создания задания резервирования базы данных нужно ввести имя сервера, где находится необходимая для резервирования база данных, и указать имя пользователя, пароль. Если подключение успешно, то надпись о статусе соединения сменится и появится список баз данных. Интерфейс создания и редактирования задачи изображен на рисунке 14. Также есть настройка количества хранимых копий.

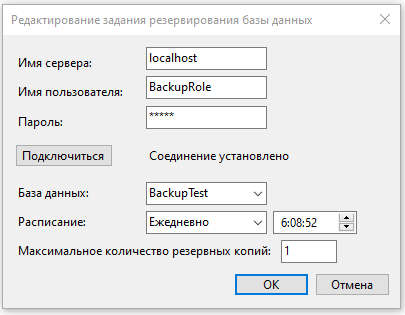


Рис. 14. Интерфейс создания и редактирования задачи базы данных.

Для восстановления базы данных или файла нужно выбрать из списка задачу и нажать кнопку восстановления, после чего появится диалоговое окно, где будет предложено ввести имя файла (по умолчанию то же имя что было при резервировании) и путь. Аналогично происходит восстановление баз данных, если ничего не менять, то восстановление в ту же базу данных, из которой была создана резервная копия (перезапись), иначе будет создана новая база данных. При восстановлении будут также созданы папки из имени файла, если они были удалены.

Доступен просмотр истории действий с заданием по резервированию, рисунок 15 и отключение задания.

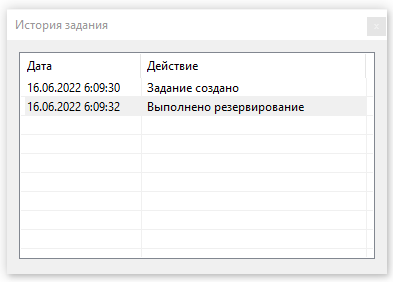


Рис. 15. Просмотр истории действий с заданием.

Также есть возможность управления квотой резервируемых данных на сервере, рисунок 16.

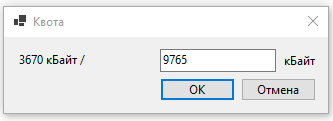


Рис. 16. Редактирование допустимого объема для резервирования.

## Вывод по главе

На основе информации, рассмотренной в теоретической части данной работы была разработана централизованная модель клиент-серверного приложения. На основе созданной модели представлен программный комплекс, разработанный при выполнении ВКР.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения ВКР были решены все задачи, необходимые для создания клиент-серверного приложения резервирования и восстановления данных:

* Рассмотрены существующие программные решения резервирования данных;
* Разработана централизованная модель клиент-серверного приложения;
* реализовано резервирование и восстановление баз данных Microsoft SQL Server и PostgreSQL
* Разработано программное обеспечение, реализующее резервирование и восстановление наиболее востребованных типов источников (файлы, базы данных SQL server) и позволяющее добавлять новые типы источников.

Такие образом, цель ВКР достигнута.

# СПИСОК ЛИТЕРАТУРЫ

1. Абидарова А. А. Резервное копирование и хранение данных [Электронный ресурс]. // URL: https://cyberleninka.ru/article/n/rezervnoe-kopirovanie-i-hranenie-dannyh (дата обращения: 12.06.2022).
2. Storage Networking Industry Association (SNIA). A Dictionary of Storage Networking Terminology // [Электронный ресурс]. URL: <http://www.snia.org/education/dictionary/> (дата обращения: 01.06.2022).
3. Виды резервирования // [Электронный ресурс]. URL: https://www.sim-networks.com/ru/blog/backup-full-increment-differential (дата обращения: 21.05.2022).
4. Microsoft Corporation. Description of Full, Incremental, and Differential Backups. // [Электронный ресурс]. URL: <http://support.microsoft.com/kb/136621> (дата обращения: 01.06.2022).
5. Dorion P. Backup and Disaster Recovery. Incremental/differential backup differences URL: [http://searchstorage.techtarget.com/tip/ 1,289483,sid5\_gci1119483,00.html](http://searchstorage.techtarget.com/tip/%201,289483,sid5_gci1119483,00.html) (дата обращения: 12.06.2022).
6. What are the differences between Differential and Incremental backups? Symantec Enterprise Technical Support. Article: TECH7665. // URL: <https://www.veritas.com/support/en_US/article.TECH7665> (дата обращения: 12.06.2022).
7. Coughlin T. Coughlin Associates, Inc. Data Protection for Small Businesses and Homes. // [Электронный ресурс]. URL: [www.tomcoughlin.com](http://www.tomcoughlin.com) (дата обращения: 01.06.2022).
8. Резервное копирование // [Электронный ресурс]. URL: http://www.rhd.ru/docs/manuals/enterprise/RHEL-4-Manual/admin-guide/s1-disaster-backups.html (дата обращения: 14.05.2022).
9. Работа с библиотекой Network // [Электронный ресурс]. URL: https://github.com/Toemsel/Network (дата обращения: 14.05.2022).
10. Работа с библиотекой Newtonsoft.Json на реальном примере // [Электронный ресурс]. URL: <https://habr.com/ru/post/481514/> (дата обращения: 14.05.2022).
11. The default dynamic port range for TCP/IP has changed since Windows Vista and in Windows Server 2008 // [Электронный ресурс]. URL: <https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/default-dynamic-port-range-tcpip-chang> (дата обращения: 01.06.2022).

# Приложение 1

# Реализация управляющей службы

using ServerService;

var pathBackup =Path.Combine(AppDomain.CurrentDomain.BaseDirectory,"BackupFiles");

if (!Directory.Exists(pathBackup))

Directory.CreateDirectory(pathBackup);

IHost host = Host.CreateDefaultBuilder(args)

.ConfigureServices(services =>

{

services.AddHostedService<Server>();

})

.UseWindowsService(options =>

{

options.ServiceName = "MainService";

})

.Build();

namespace ServerService

{

public class UserStruct

{

public string Username { get; set; }

public string Password { get; set; }

}

}

using Network;

using SharedData;

using System.Collections.ObjectModel;

using System.Text;

using Newtonsoft.Json;

namespace ServerService

{

public class Server : BackgroundService

{

private ServerConnectionContainer \_server;

private const string TASKS\_FILENAME = "Tasks.json";

private const string BACKUP\_FOLDER = "BackupFiles";

private TasksInfo \_tasks;

string exePath = AppDomain.CurrentDomain.BaseDirectory;

public static ObservableCollection<UserStruct> userDB { get; set; } = new ObservableCollection<UserStruct>();

public async Task Listen()

{

LoadUsers();

\_server = ConnectionFactory.CreateServerConnectionContainer(1708, false);

\_server.AllowUDPConnections = false;

\_server.ConnectionEstablished += (conn, type) =>

{

File.AppendAllText(Path.Combine(exePath, "log.txt"), $"-> New connection\n");

conn.TIMEOUT = 600000;

conn.RegisterPacketHandler<SharedRequest>(HandlerCommand, this);

conn.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

});

};

await \_server.Start();

}

async void LoadUsers()

{

if (File.Exists(Path.Combine(exePath, "users.json")))

{

userDB = JsonConvert.DeserializeObject<ObservableCollection<UserStruct>>(File.ReadAllText(Path.Combine(exePath, "users.json")));

File.AppendAllText(Path.Combine(exePath, "log.txt"), $"Loaded {userDB.Count.ToString()} users\n");

}

else

{

UserStruct userStruct = new UserStruct() { Username = "root", Password = new Random().Next(99999, 999999).ToString() };

userDB.Add(userStruct);

File.AppendAllText(Path.Combine(exePath, "log.txt"), $"No find user! Creating main user: {userStruct.Username}:{userStruct.Password}\n");

File.AppendAllText(Path.Combine(exePath, "users.json"), JsonConvert.SerializeObject(userDB));

}

}

private async void HandlerCommand(SharedRequest packet, Connection connection)

{

var result = "Error";

switch (packet.Command)

{

case "tasks":

{

await File.AppendAllTextAsync(Path.Combine(exePath, "log.txt"), "->Tasks updated\n");

\_tasks = TasksInfo.FromArray(packet.Data);

await \_tasks.SaveToFileAsync(TASKS\_FILENAME);

//TODO: send to other and locks

var request = new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

};

foreach (TcpConnection tcpConnection in \_server.TCP\_Connections)

{

if (tcpConnection != connection)

{

await tcpConnection.SendAsync<SharedResponse>(request);

}

}

Task.Run(() => CheckBackups());

result = "OK";

break;

}

case "backup":

{

await File.AppendAllTextAsync(Path.Combine(exePath, "log.txt"), "->Get files for backup\n");

var files = FilesInfo.FromBin(packet.Data);

foreach (var file in files.Data)

{

var dir = Path.Combine(exePath,BACKUP\_FOLDER, file.Id.ToString(), file.Date.ToString("yyMMdd\_HHmmss"));

await Task.Run(() => Directory.CreateDirectory(dir));

await File.WriteAllBytesAsync(Path.Combine(dir, Path.GetFileName(file.NameFile)), file.Bin);

}

result = "OK";

break;

}

case "Login":

{

string[] logData = Encoding.UTF8.GetString(packet.Data).Split(new string[] { " &\*&\*& " }, StringSplitOptions.None);

string username = logData[0];

string password = logData[1];

lock (userDB)

{

var user = userDB.FirstOrDefault(x => x.Username == username && x.Password == password);

if (user != null)

SendLoginState(true, connection);

else

SendLoginState(false, connection);

}

break;

}

case "restore":

{

RestoreTask restoreTask = RestoreTask.FromArray(packet.Data);

if (\_tasks.Data.ContainsKey(restoreTask.Id))

{

var task = \_tasks.Data[restoreTask.Id];

string filename = "";

if (task is FileBackupTask)

{

filename = (task as FileBackupTask).FileName;

}

else if (task is SQLBackupTask)

{

filename = (task as SQLBackupTask).DbName + ".bak";

}

else if (task is PGBackupTask)

{

filename = (task as PGBackupTask).DbName + ".backup";

}

else

{

break;

}

var fullPath = Path.Combine(exePath,

BACKUP\_FOLDER,

restoreTask.Id.ToString(),

task.BackupTimes.Max().ToString("yyMMdd\_HHmmss"),

Path.GetFileName(filename));

if (File.Exists(fullPath))

{

await File.AppendAllTextAsync(Path.Combine(exePath, "log.txt"), $"->Restore task {restoreTask.Id}\n");

var restoreFile = new FilesInfo();

restoreFile.Add(restoreTask.Id, DateTime.MinValue, restoreTask.Path, await File.ReadAllBytesAsync(fullPath));

foreach (TcpConnection tcpConnection in \_server.TCP\_Connections)

{

if (tcpConnection != connection)

{

await tcpConnection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "restore",

Data = restoreFile.ToArray()

});

}

}

}

}

result = "OK";

break;

}

default:

await File.AppendAllTextAsync(Path.Combine(exePath, "log.txt"), $"-> Command not found! ({packet.Command})\n");

break;

}

connection.Send(new SharedResponse(result, packet));

}

void CheckBackups()

{

bool tasksChanged = false;

var dirInfo = new DirectoryInfo(Path.Combine(exePath,BACKUP\_FOLDER));

foreach (var dir in dirInfo.GetDirectories())

{

int numDir = 0;

if (!Int32.TryParse(dir.Name, out numDir) || !\_tasks.Data.ContainsKey(numDir))

{

tasksChanged = true;

Directory.Delete(dir.FullName, true);

}

}

List<BackupTask> backupTasks = new List<BackupTask>(\_tasks.Data.Values);

foreach (var task in backupTasks)

{

if (task.BackupTimes.Count > task.MaxCount)

{

while (task.BackupTimes.Count > task.MaxCount)

{

var olderDateTime = task.BackupTimes.Min();

var dir = Path.Combine(BACKUP\_FOLDER, task.Id.ToString(), olderDateTime.ToString("yyMMdd\_HHmmss"));

Directory.Delete(dir, true);

task.BackupTimes.Remove(olderDateTime);

tasksChanged = true;

}

\_tasks.Data[task.Id] = task;

}

}

if (tasksChanged)

{

\_tasks.UsedQuota = GetFolderSize(dirInfo);

\_tasks.SaveToFile(TASKS\_FILENAME);

foreach (TcpConnection tcpConnection in \_server.TCP\_Connections)

{

tcpConnection.Send(new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

});

}

}

}

long GetFolderSize(DirectoryInfo di)

{

long size = 0;

// Добавляем размер файлов

FileInfo[] files = di.GetFiles();

foreach (FileInfo fi in files)

{

size += fi.Length;

}

// Добавляем размер подкаталогов

DirectoryInfo[] dirs = di.GetDirectories();

foreach (DirectoryInfo dir in dirs)

{

size += GetFolderSize(dir);

}

return size;

}

void SendLoginState(bool logged, Connection connection)

{

File.AppendAllTextAsync(Path.Combine(exePath, "log.txt"), $"Login state...{logged.ToString()}\n");

connection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "Login",

Data = Encoding.UTF8.GetBytes(logged.ToString())

});

}

protected internal void Disconnect() => \_server.Stop();

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

\_tasks = TasksInfo.LoadFromFile(TASKS\_FILENAME);

while (!stoppingToken.IsCancellationRequested)

{

try

{

await Listen();

}

catch (Exception ex)

{

// обработка ошибки однократного неуспешного выполнения фоновой задачи

}

await Task.Delay(1000);

}

}

}

}

await host.RunAsync();

# Приложение 2.

# Реализация подчиненной службы

using Newtonsoft.Json;

namespace ClientConfig

{

public class Config

{

public string ServerIp { get; set; }

public int ServerPort { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public Config()

{

ServerIp = "127.0.0.1";

ServerPort = 1708;

Login = "root";

Password = "";

}

public void SaveToFile(string filename)

{

File.WriteAllText(filename, JsonConvert.SerializeObject(this));

}

public async Task SaveToFileAsync(string filename)

{

await File.WriteAllTextAsync(filename, JsonConvert.SerializeObject(this));

}

public static Config LoadFromFile(string filename)

{

if (File.Exists(filename))

{

Config clientConfig = JsonConvert.DeserializeObject<Config>(File.ReadAllText(filename));

if (clientConfig != null)

{

return clientConfig;

}

}

return new Config();

}

public async static Task<Config> LoadFromFileAsync(string filename)

{

if (File.Exists(filename))

{

Config clientConfig = JsonConvert.DeserializeObject<Config>(await File.ReadAllTextAsync(filename));

if (clientConfig != null)

{

return clientConfig;

}

}

return new Config();

}

}

}

using ClientService;

#pragma warning disable CA1416

System.ServiceProcess.ServiceBase.Run(new WinService());

#pragma warning restore CA1416

using System.Data;

using System.Data.SqlClient;

using System.ServiceProcess;

using Network;

using SharedData;

using ClientConfig;

namespace ClientService;

#pragma warning disable CA1416

public class WinService : ServiceBase

{

private const string SERVICE\_NAME = "ClientService";

private Config \_config = new Config();

private bool \_isWork = false;

private TasksInfo \_tasks;

private TcpConnection? \_client;

private string \_address = "";

public WinService()

{

this.ServiceName = SERVICE\_NAME;

this.CanStop = true;

this.CanPauseAndContinue = false;

this.AutoLog = false;

\_tasks = new TasksInfo();

\_client = null;

}

protected override async void OnStart(string[] args)

{

var configFilename = Path.Combine(Environment.CurrentDirectory, "Config.json");

\_config = await Config.LoadFromFileAsync(configFilename);

await Connect();

\_isWork = true;

await Task.Run(Handler);

}

protected override void OnStop()

{

base.OnStop();

\_isWork = false;

}

private async Task Connect()

{

var result = await ConnectionFactory.CreateTcpConnectionAsync(\_config.ServerIp, \_config.ServerPort);

if (result.Item2 == ConnectionResult.Connected)

{

\_client = result.Item1;

\_client.RegisterPacketHandler<SharedRequest>(RecvHandler, this);

\_client.TIMEOUT = 600000;

\_address = \_client.IPLocalEndPoint.Address.MapToIPv4().ToString();

}

}

private async void RecvHandler(SharedRequest packet, Connection connection)

{

string result = "Error";

switch (packet.Command)

{

case "tasks":

{

\_tasks = TasksInfo.FromArray(packet.Data);

result = "OK";

break;

}

case "restore":

{

var files = FilesInfo.FromBin(packet.Data);

foreach (var file in files.Data)

{

if (!\_tasks.Data.ContainsKey(file.Id))

continue;

var task = \_tasks.Data[file.Id];

if (task.Address != \_address)

continue;

if (task is FileBackupTask)

{

await Task.Run(() => Directory.CreateDirectory(Path.GetDirectoryName(file.NameFile)));

await File.WriteAllBytesAsync(file.NameFile, file.Bin);

result = "OK";

}

else if (task is (SQLBackupTask))

{

var dbTask = task as SQLBackupTask;

string fullPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, file.NameFile + ".bak");

await File.WriteAllBytesAsync(fullPath, file.Bin);

try

{

SqlConnectionStringBuilder connStringBuilder = new SqlConnectionStringBuilder();

connStringBuilder.DataSource = dbTask.Server;

connStringBuilder.UserID = dbTask.Login;

connStringBuilder.Password = dbTask.Password;

using (SqlConnection conn = new SqlConnection(connStringBuilder.ConnectionString))

{

string dataFile = $"C:\\{file.NameFile}.mdf";

string logFile = $"C:\\{file.NameFile}\_log.ldf";

await conn.OpenAsync();

var query = @"RESTORE FILELISTONLY FROM DISK = @localDatabasePath";

var sqlCommand = new SqlCommand(query, conn);

sqlCommand.Parameters.AddWithValue("@localDatabasePath", fullPath);

var sqlDataReader = await sqlCommand.ExecuteReaderAsync();

while (await sqlDataReader.ReadAsync())

{

if (sqlDataReader["Type"].ToString() == "D")

dataFile = sqlDataReader["PhysicalName"].ToString().Replace(dbTask.DbName, file.NameFile);

else if (sqlDataReader["Type"].ToString() == "L")

logFile = sqlDataReader["PhysicalName"].ToString().Replace(dbTask.DbName, file.NameFile);

}

await sqlDataReader.CloseAsync();

query = @"RESTORE DATABASE @databaseName

FROM DISK = @localDatabasePath

WITH REPLACE,

MOVE @TemplateDatabase TO @NewDatabaseData,

MOVE @TemplateDatabaseLog TO @NewDatabaseLog;";

sqlCommand = new SqlCommand(query, conn);

sqlCommand.Parameters.AddWithValue("@databaseName", file.NameFile);

sqlCommand.Parameters.AddWithValue("@localDatabasePath", fullPath);

sqlCommand.Parameters.AddWithValue("@TemplateDatabase", dbTask.DbName);

sqlCommand.Parameters.AddWithValue("@TemplateDatabaseLog", dbTask.DbName+"\_log");

sqlCommand.Parameters.AddWithValue("@NewDatabaseData", dataFile);

sqlCommand.Parameters.AddWithValue("@NewDatabaseLog", logFile);

await sqlCommand.ExecuteNonQueryAsync();

}

result = "OK";

}

catch (Exception ex)

{

await File.AppendAllTextAsync(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), ex.Message + "\n");

}

finally

{

FileInfo fi = new FileInfo(fullPath);

await fi.DeleteAsync();

}

}

else if (task is (PGBackupTask))

{

var pgTask = task as PGBackupTask;

string fullPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, file.NameFile + ".backup");

await File.WriteAllBytesAsync(fullPath, file.Bin);

try

{

PgStore.Config pgConfig = new PgStore.Config();

pgConfig.ServerName = pgTask.Host;

pgConfig.Port = pgTask.Port;

pgConfig.UserName = pgTask.UserId;

pgConfig.Password = pgTask.Password;

pgConfig.DataBase = file.NameFile;

PgStore.Control.CurrentConfig = pgConfig;

PgStore.Control.ResultChanged += Control\_ResultChanged;

await Task.Factory.StartNew(() => PgStore.Control.Restaure(fullPath, true));

result = "OK";

}

catch (Exception ex)

{

await File.AppendAllTextAsync(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), ex.Message + "\n");

}

finally

{

FileInfo fi = new FileInfo(fullPath);

await fi.DeleteAsync();

}

}

if (result == "OK")

{

task.AddAction(TaskAction.Restore);

\_tasks.Data[task.Id] = task;

}

}

break;

}

default:

result = "Unknown command";

break;

}

connection.Send(new SharedResponse(result, packet));

if (result == "OK")

{

await connection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

});

}

}

private async Task<long> FileBackup(FileBackupTask fileTask, FilesInfo filesForBackup,

List<BackupTask> updatedTasks, List<string> filesForDelete, long quotaAddBytes)

{

if (File.Exists(fileTask.FileName))

{

FileInfo fi = new FileInfo(fileTask.FileName);

if (fi.Length + quotaAddBytes + \_tasks.UsedQuota <= \_tasks.MaxQuota)

{

quotaAddBytes += fi.Length;

var backupTime = DateTime.Now;

filesForBackup.Add(fileTask.Id, backupTime, fileTask.FileName, await File.ReadAllBytesAsync(fileTask.FileName));

FileBackupTask updatedTask = fileTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.AddAction(TaskAction.Backup);

updatedTask.UpdateNextBackupTime();

updatedTask.BackupTimes.Add(backupTime);

updatedTasks.Add(updatedTask);

}

else

{

FileBackupTask updatedTask = fileTask;

updatedTask.Status = SharedData.TaskStatus.Error\_Quota;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

}

else

{

FileBackupTask updatedTask = fileTask;

updatedTask.Status = SharedData.TaskStatus.Error\_NoFile;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

return quotaAddBytes;

}

private async Task<long> SqlServerBackup(SQLBackupTask dbTask, FilesInfo filesForBackup,

List<BackupTask> updatedTasks, List<string> filesForDelete, long quotaAddBytes)

{

string fileName = $"{dbTask.DbName}.bak";

string fullPath = Path.Combine(Path.GetTempPath(), fileName);

try

{

SqlConnectionStringBuilder connStringBuilder = new SqlConnectionStringBuilder();

connStringBuilder.DataSource = dbTask.Server;

connStringBuilder.UserID = dbTask.Login;

connStringBuilder.Password = dbTask.Password;

using (SqlConnection connection = new SqlConnection(connStringBuilder.ConnectionString))

{

await connection.OpenAsync();

var formatMediaName = $"DatabaseToolkitBackup\_{dbTask.DbName}";

var formatName = $"Full Backup of {dbTask.DbName}";

string query = "BACKUP DATABASE @databaseName TO DISK = @localDatabasePath WITH FORMAT, MEDIANAME = @formatMediaName, NAME = @formatName";

var sqlCommand = new SqlCommand(query, connection);

sqlCommand.Parameters.AddWithValue("@databaseName", dbTask.DbName);

sqlCommand.Parameters.AddWithValue("@localDatabasePath", fullPath);

sqlCommand.Parameters.AddWithValue("@formatMediaName", formatMediaName);

sqlCommand.Parameters.AddWithValue("@formatName", formatName);

await sqlCommand.ExecuteNonQueryAsync();

}

FileInfo fi = new FileInfo(fullPath);

if (fi.Length + quotaAddBytes + \_tasks.UsedQuota <= \_tasks.MaxQuota)

{

quotaAddBytes += fi.Length;

var backupTime = DateTime.Now;

filesForBackup.Add(dbTask.Id, backupTime, fullPath, await File.ReadAllBytesAsync(fullPath));

SQLBackupTask updatedTask = dbTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.AddAction(TaskAction.Backup);

updatedTask.UpdateNextBackupTime();

updatedTask.BackupTimes.Add(backupTime);

updatedTasks.Add(updatedTask);

}

else

{

SQLBackupTask updatedTask = dbTask;

updatedTask.Status = SharedData.TaskStatus.Error\_Quota;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

}

catch (Exception ex)

{

await File.AppendAllTextAsync(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), ex.Message + "\n");

SQLBackupTask updatedTask = dbTask;

updatedTask.Status = SharedData.TaskStatus.Error\_DbConnect;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

return quotaAddBytes;

}

void Control\_ResultChanged(object sender, EventArgs e)

{

File.AppendAllText(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), sender.ToString() + "\n");

}

private async Task<long> PgSqlBackup(PGBackupTask pgTask, FilesInfo filesForBackup,

List<BackupTask> updatedTasks, List<string> filesForDelete, long quotaAddBytes)

{

string fileName = $"{pgTask.DbName}.backup";

string fullPath = Path.Combine(Path.GetTempPath(), fileName);

try

{

PgStore.Config pgConfig = new PgStore.Config();

pgConfig.ServerName = pgTask.Host;

pgConfig.Port = pgTask.Port;

pgConfig.UserName = pgTask.UserId;

pgConfig.Password = pgTask.Password;

pgConfig.DataBase = pgTask.DbName;

PgStore.Control.CurrentConfig = pgConfig;

PgStore.Control.ResultChanged += Control\_ResultChanged;

await Task.Factory.StartNew(() => PgStore.Control.Backup(Path.GetTempPath(), fileName, true));

FileInfo fi = new FileInfo(fullPath);

if (fi.Length + quotaAddBytes + \_tasks.UsedQuota <= \_tasks.MaxQuota)

{

quotaAddBytes += fi.Length;

var backupTime = DateTime.Now;

filesForBackup.Add(pgTask.Id, backupTime, fullPath, await File.ReadAllBytesAsync(fullPath));

PGBackupTask updatedTask = pgTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.AddAction(TaskAction.Backup);

updatedTask.UpdateNextBackupTime();

updatedTask.BackupTimes.Add(backupTime);

updatedTasks.Add(updatedTask);

}

else

{

PGBackupTask updatedTask = pgTask;

updatedTask.Status = SharedData.TaskStatus.Error\_Quota;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

}

catch (Exception ex)

{

await File.AppendAllTextAsync(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), ex.Message + "\n");

PGBackupTask updatedTask = pgTask;

updatedTask.Status = SharedData.TaskStatus.Error\_DbConnect;

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

return quotaAddBytes;

}

private async Task Handler()

{

while (\_isWork)

{

//Если нет подключения, то пытаемся подключиться к серверу

if (\_client == null || !\_client.IsAlive)

{

await Connect();

}

else

{

try

{

long quotaAddBytes = 0;

var filesForBackup = new FilesInfo();

var updatedTasks = new List<BackupTask>();

var filesForDelete = new List<string>();

foreach (var task in \_tasks.Data.Values.Where(task =>

task.NextBackupTime <= DateTime.Now

&& task.Status != SharedData.TaskStatus.Disabled

&& task.Address == \_address))

{

if (task == null)

continue;

if (task is FileBackupTask)

{

quotaAddBytes = await FileBackup(task as FileBackupTask, filesForBackup, updatedTasks,

filesForDelete, quotaAddBytes);

}

else if (task is SQLBackupTask)

{

quotaAddBytes = await SqlServerBackup(task as SQLBackupTask, filesForBackup, updatedTasks,

filesForDelete, quotaAddBytes);

}

else if (task is PGBackupTask)

{

quotaAddBytes = await PgSqlBackup(task as PGBackupTask, filesForBackup, updatedTasks,

filesForDelete, quotaAddBytes);

}

}

if (filesForBackup.Data.Count > 0)

{

//send backup files

await \_client.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "backup",

Data = filesForBackup.ToArray()

});

}

if (updatedTasks.Count > 0)

{

\_tasks.UsedQuota += quotaAddBytes;

foreach (var task in updatedTasks)

{

\_tasks.Data[task.Id] = task;

}

//send tasks list

await \_client.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

});

}

if (filesForDelete.Count > 0)

{

foreach (var file in filesForDelete)

{

FileInfo fi = new FileInfo(file);

await fi.DeleteAsync();

}

}

}

catch { }

}

await Task.Delay(1000);

}

}

}

public static class FileExtensions

{

public static Task DeleteAsync(this FileInfo fi)

{

return Task.Factory.StartNew(() => fi.Delete());

}

}

#pragma warning restore CA1416

# Приложение 3.

# Реализация приложения с графическим интерфейсом

using Network;

using System.Text;

using ClientConfig;

using SharedData;

namespace DesktopClient;

public static class Globals

{

public static Config Config = new Config();

private static string \_configFilename = Path.Combine(Environment.CurrentDirectory, "Config.json");

public static TasksInfo Tasks;

private static TcpConnection? \_connection;

public static Main? MainWindow;

public static string IpAddress = "";

public static int Port = 0;

public static Login Login = null;

public static bool connected = false;

public static bool Init()

{

return Connect();

}

public static bool Connect()

{

ConnectionResult result = ConnectionResult.TCPConnectionNotAlive;

\_connection = ConnectionFactory.CreateTcpConnection(Config.ServerIp, Config.ServerPort, out result);

if (result == ConnectionResult.Connected)

{

connected = true;

\_connection.RegisterStaticPacketHandler<SharedRequest>(RecvHandler);

\_connection.TIMEOUT = 600000;

IpAddress = \_connection.IPLocalEndPoint.Address.MapToIPv4().ToString();

Port = \_connection.IPLocalEndPoint.Port;

return true;

}

return false;

}

private static async void RecvHandler(SharedRequest packet, Connection connection)

{

string result = "Error";

switch (packet.Command)

{

case "tasks":

{

Tasks = TasksInfo.FromArray(packet.Data);

if (MainWindow != null)

MainWindow.BeginInvoke((Action)(() => MainWindow.UpdateTable(Globals.Tasks)));

result = "OK";

break;

}

case "Login":

{

bool isLogged = Convert.ToBoolean(Encoding.UTF8.GetString(packet.Data));

try

{

if (isLogged)

{

Login.Invoke(new Action(() =>

{

Login.isLogin = true;

Login.Close();

}));

}

else

{

Login.Invoke(new Action(() =>

{

MessageBox.Show("Login of password invalid!", "Login Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

Login.Clear();

}));

}

}

catch { }

break;

}

default:

result = "Unknown command";

break;

}

connection.Send(new SharedResponse(result, packet));

}

public static async void SendLogin(string username, string password)

{

try

{

await \_connection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "Login",

Data = Encoding.UTF8.GetBytes($"{username} &\*&\*& {password}")

});

}

catch { }

}

public static async void SendTasks()

{

if (\_connection == null)

return;

await \_connection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "tasks",

Data = Tasks.ToArray()

});

}

public static async void SendRestore(RestoreTask restoreTask)

{

if (\_connection == null)

return;

await \_connection.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "restore",

Data = restoreTask.ToArray()

});

}

public static void LoadConfig()

{

Config = Config.LoadFromFile(\_configFilename);

}

public static void SaveConfig()

{

Config.SaveToFile(\_configFilename);

}

}

namespace DesktopClient

{

public partial class Login : Form

{

public Login()

{

InitializeComponent();

textBoxServerAddress.Text = Globals.Config.ServerIp;

textBoxServerPort.Text = Globals.Config.ServerPort.ToString();

textBoxLogin.Text = Globals.Config.Login;

textBoxPass.Text = Globals.Config.Password;

}

private void buttonConnection\_Click(object sender, EventArgs e)

{

Globals.Config.ServerIp = textBoxServerAddress.Text;

int port = 0;

int.TryParse(textBoxServerPort.Text, out port);

Globals.Config.ServerPort = port;

Globals.Config.Login = textBoxLogin.Text;

Globals.Config.Password = textBoxPass.Text;

Globals.SaveConfig();

if (Globals.Init())

{

Globals.SendLogin(Globals.Config.Login, Globals.Config.Password);

}

else if (Globals.connected)

{

Globals.SendLogin(Globals.Config.Login, Globals.Config.Password);

}

else

{

MessageBox.Show("Не удалось соединиться с сервером");

}

}

public bool isLogin = false;

public void Clear()

{

textBoxLogin.Clear();

textBoxPass.Clear();

}

private void Login\_Load(object sender, EventArgs e)

{

Globals.Login = this;

}

private void Login\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!isLogin)

Environment.Exit(0);

}

}

}

using System.Data;

using SharedData;

namespace DesktopClient;

public partial class Main : Form

{

public Main()

{

InitializeComponent();

}

public void UpdateTable(TasksInfo tasks)

{

this.Text = $"Задания сервера {Globals.Config.ServerIp}:{Globals.Config.ServerPort}";

listView1.Items.Clear();

foreach (var task in tasks.Data.Values.OrderBy(t => t.Id))

{

ListViewItem lvi = new ListViewItem();

lvi.Tag = task;

lvi.Text = task.Id.ToString();

lvi.Name = "Task";

string taskType;

if (task is FileBackupTask)

taskType = "Файл";

else if (task is SQLBackupTask)

taskType = "SQL Server";

else if (task is PGBackupTask)

taskType = "PostgreSQL";

else

taskType = "Неизвестно";

ListViewItem.ListViewSubItem subItem = new ListViewItem.ListViewSubItem(lvi, taskType);

lvi.SubItems.Add(subItem);

subItem = new ListViewItem.ListViewSubItem(lvi, $"{task.Address}:{task.LocalPort}");

lvi.SubItems.Add(subItem);

subItem = new ListViewItem.ListViewSubItem(lvi, task.GetStatusString());

lvi.SubItems.Add(subItem);

subItem = new ListViewItem.ListViewSubItem(lvi, task.GetScheduleString());

lvi.SubItems.Add(subItem);

listView1.Items.Add(lvi);

}

progressBarQuota.Value = tasks.UsedQuota < tasks.MaxQuota

? (int)(tasks.UsedQuota \* 100 / tasks.MaxQuota)

: 100;

}

private void buttonAddFile\_Click(object sender, EventArgs e)

{

var taskEditDlg = new TaskFileEdit();

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

if (string.IsNullOrEmpty(newTask.FileName))

return;

newTask.Id = Globals.Tasks.GetNextId();

newTask.Address = Globals.IpAddress;

newTask.LocalPort = Globals.Port;

newTask.AddAction(TaskAction.Created);

Globals.Tasks.Data[newTask.Id] = newTask;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

private void buttonEdit\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count <= 0)

return;

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task == null)

return;

if (task is FileBackupTask)

{

var taskEditDlg = new TaskFileEdit();

taskEditDlg.SetTask(task as FileBackupTask);

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

task.NextBackupTime = newTask.NextBackupTime;

task.Schedule = newTask.Schedule;

task.MaxCount = newTask.MaxCount;

(task as FileBackupTask).FileName = newTask.FileName;

Globals.Tasks.Data[task.Id] = task;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

else if (task is SQLBackupTask)

{

var dbTask = task as SQLBackupTask;

var taskEditDlg = new TaskDatabaseEdit();

taskEditDlg.SetTask(dbTask);

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

dbTask.NextBackupTime = newTask.NextBackupTime;

dbTask.Schedule = newTask.Schedule;

dbTask.MaxCount = newTask.MaxCount;

dbTask.Server = newTask.Server;

dbTask.Login = newTask.Login;

dbTask.Password = newTask.Password;

dbTask.DbName = newTask.DbName;

Globals.Tasks.Data[dbTask.Id] = dbTask;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

else if (task is PGBackupTask)

{

var pgTask = task as PGBackupTask;

var taskEditDlg = new TaskPgSqlEdit();

taskEditDlg.SetTask(pgTask);

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

pgTask.NextBackupTime = newTask.NextBackupTime;

pgTask.Schedule = newTask.Schedule;

pgTask.MaxCount = newTask.MaxCount;

pgTask.Host = newTask.Host;

pgTask.Port = newTask.Port;

pgTask.UserId = newTask.UserId;

pgTask.Password = newTask.Password;

pgTask.DbName = newTask.DbName;

Globals.Tasks.Data[pgTask.Id] = pgTask;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

}

private void buttonDelete\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count != 1)

return;

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task == null)

return;

Globals.Tasks.Data.Remove(task.Id);

Globals.SendTasks();

UpdateTable(Globals.Tasks);

listView1\_SelectedIndexChanged(this, EventArgs.Empty);

}

private void listView1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (listView1.SelectedIndices.Count == 1)

{

buttonEdit.Enabled = true;

buttonDelete.Enabled = true;

buttonRestore.Enabled = true;

buttonDisable.Enabled = true;

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task != null)

{

buttonDisable.Text = task.Status == SharedData.TaskStatus.Disabled ? "Включить" : "Отключить";

}

buttonHistory.Enabled = true;

}

else

{

buttonEdit.Enabled = false;

buttonDelete.Enabled = false;

buttonRestore.Enabled = false;

buttonDisable.Enabled = false;

buttonHistory.Enabled = false;

}

}

private void Main\_Load(object sender, EventArgs e)

{

Login s = new Login();

s.ShowDialog();

}

private void buttonAddDb\_Click(object sender, EventArgs e)

{

var taskEditDlg = new TaskDatabaseEdit();

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

newTask.Id = Globals.Tasks.GetNextId();

newTask.Address = Globals.IpAddress;

newTask.LocalPort = Globals.Port;

newTask.AddAction(TaskAction.Created);

Globals.Tasks.Data[newTask.Id] = newTask;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

private void buttonRestore\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count == 1)

{

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task == null)

return;

if (task is FileBackupTask)

{

FileBackupTask fileTask = task as FileBackupTask;

var dialog = new RestoreFile();

dialog.textBoxFilename.Text = fileTask.FileName;

if (dialog.ShowDialog() != DialogResult.OK)

return;

RestoreTask restoreTask = new RestoreTask(fileTask.Id,

string.IsNullOrEmpty(dialog.textBoxFilename.Text) ? fileTask.FileName : dialog.textBoxFilename.Text);

Globals.SendRestore(restoreTask);

}

else if (task is SQLBackupTask)

{

SQLBackupTask sqlTask = task as SQLBackupTask;

var dialog = new RestoreDb();

dialog.textBoxDbName.Text = sqlTask.DbName;

if (dialog.ShowDialog() != DialogResult.OK)

return;

RestoreTask restoreTask = new RestoreTask(sqlTask.Id,

string.IsNullOrEmpty(dialog.textBoxDbName.Text) ? sqlTask.DbName : dialog.textBoxDbName.Text);

Globals.SendRestore(restoreTask);

}

else if (task is PGBackupTask)

{

PGBackupTask pgTask = task as PGBackupTask;

var dialog = new RestoreDb();

dialog.textBoxDbName.Text = pgTask.DbName;

if (dialog.ShowDialog() != DialogResult.OK)

return;

RestoreTask restoreTask = new RestoreTask(pgTask.Id,

string.IsNullOrEmpty(dialog.textBoxDbName.Text) ? pgTask.DbName : dialog.textBoxDbName.Text);

Globals.SendRestore(restoreTask);

}

}

}

private void buttonDisable\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count != 1)

return;

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task == null)

return;

if (task.Status == SharedData.TaskStatus.Disabled)

{

task.Status = task.BackupTimes.Count == 0 ? SharedData.TaskStatus.New : SharedData.TaskStatus.Working;

}

else

{

task.Status = SharedData.TaskStatus.Disabled;

}

Globals.Tasks.Data[task.Id] = task;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

listView1\_SelectedIndexChanged(this, EventArgs.Empty);

}

private void buttonQuota\_Click(object sender, EventArgs e)

{

var quotaDlg = new Quota();

if (quotaDlg.ShowDialog() != DialogResult.OK)

return;

long newQuota = 0;

if (long.TryParse(quotaDlg.textBoxMax.Text, out newQuota))

{

Globals.Tasks.MaxQuota = newQuota \* 1024;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

}

private void buttonHistory\_Click(object sender, EventArgs e)

{

if (listView1.SelectedItems.Count <= 0)

return;

BackupTask task = (BackupTask)listView1.SelectedItems[0].Tag;

if (task == null)

return;

var historyDlg = new TaskHistory();

historyDlg.UpdateHistoty(task);

historyDlg.ShowDialog();

}

private void buttonAddPgSql\_Click(object sender, EventArgs e)

{

var taskEditDlg = new TaskPgSqlEdit();

if (taskEditDlg.ShowDialog() != DialogResult.OK)

return;

var newTask = taskEditDlg.GetTask();

newTask.Id = Globals.Tasks.GetNextId();

newTask.Address = Globals.IpAddress;

newTask.LocalPort = Globals.Port;

newTask.AddAction(TaskAction.Created);

Globals.Tasks.Data[newTask.Id] = newTask;

Globals.SendTasks();

UpdateTable(Globals.Tasks);

}

}

using System.Security.Principal;

namespace DesktopClient;

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

private static void Main()

{

var identity = WindowsIdentity.GetCurrent();

var principal = new WindowsPrincipal(identity);

if (!principal.IsInRole(WindowsBuiltInRole.Administrator))

{

MessageBox.Show("Запустите с правами администратора");

return;

}

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Globals.LoadConfig();

Globals.MainWindow = new Main();

Application.Run(Globals.MainWindow);

}

}

namespace DesktopClient

{

public partial class Quota : Form

{

public Quota()

{

InitializeComponent();

labelUsed.Text = $"{Globals.Tasks.UsedQuota/1024} кБайт /";

textBoxMax.Text = $"{Globals.Tasks.MaxQuota/1024}";

}

}

}

namespace DesktopClient

{

public partial class RestoreDb : Form

{

public RestoreDb()

{

InitializeComponent();

}

}

}

namespace DesktopClient

{

public partial class RestoreFile : Form

{

public RestoreFile()

{

InitializeComponent();

}

private void buttonSelectFile\_Click(object sender, EventArgs e)

{

var dialog = new SaveFileDialog();

dialog.InitialDirectory = !string.IsNullOrEmpty(textBoxFilename.Text)

? Path.GetDirectoryName(textBoxFilename.Text)

: Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

if (dialog.ShowDialog() != DialogResult.OK)

return;

textBoxFilename.Text = dialog.FileName;

}

}

}

using System.Data;

using System.Data.SqlClient;

using SharedData;

namespace DesktopClient

{

public partial class TaskDatabaseEdit : Form

{

public TaskDatabaseEdit()

{

InitializeComponent();

}

private void buttonConnect\_Click(object sender, EventArgs e)

{

comboBoxDatabases.Items.Clear();

try

{

SqlConnectionStringBuilder connStringBuilder = new SqlConnectionStringBuilder();

connStringBuilder.DataSource = textBoxServerName.Text;

connStringBuilder.UserID = textBoxNameUser.Text;

connStringBuilder.Password = textBoxPass.Text;

using (SqlConnection connection = new SqlConnection(connStringBuilder.ConnectionString))

{

connection.Open();

DataTable dtDatabases = connection.GetSchema("databases");

foreach (DataRow item in dtDatabases.Rows)

comboBoxDatabases.Items.Add(item["database\_name"]);

labelConnectionStatus.Text = "Соединение установлено";

}

}

catch (Exception ex)

{

labelConnectionStatus.Text = "Соединение не установлено";

}

if (comboBoxDatabases.Items.Count > 0)

comboBoxDatabases.SelectedIndex = 0;

}

public SQLBackupTask GetTask()

{

int maxCount = 1;

int.TryParse(textBoxCount.Text, out maxCount);

if (maxCount <= 0)

maxCount = 1;

var task = new SQLBackupTask()

{

Server = textBoxServerName.Text,

Login = textBoxNameUser.Text,

Password = textBoxPass.Text,

DbName = comboBoxDatabases.Text,

Schedule = scheduleControl.GetSchedule(),

MaxCount = maxCount

};

task.NextBackupTime = task.Schedule.GetFirstDateTime();

return task;

}

public void SetTask(SQLBackupTask task)

{

textBoxServerName.Text = task.Server;

textBoxNameUser.Text = task.Login;

textBoxPass.Text = task.Password;

comboBoxDatabases.Text = task.DbName;

scheduleControl.SetSchedule(task.Schedule);

textBoxCount.Text = task.MaxCount.ToString();

}

}

}

using SharedData;

namespace DesktopClient

{

public partial class TaskFileEdit : Form

{

public TaskFileEdit()

{

InitializeComponent();

}

private void buttonSelect\_Click(object sender, EventArgs e)

{

var dialog = new OpenFileDialog();

dialog.InitialDirectory = !string.IsNullOrEmpty(textBoxFilename.Text)

? Path.GetDirectoryName(textBoxFilename.Text)

: Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

if (dialog.ShowDialog() != DialogResult.OK)

return;

textBoxFilename.Text = dialog.FileName;

}

public FileBackupTask GetTask()

{

int maxCount = 1;

int.TryParse(textBoxCount.Text, out maxCount);

if (maxCount <= 0)

maxCount = 1;

var task = new FileBackupTask()

{

FileName = textBoxFilename.Text,

Schedule = scheduleControl.GetSchedule(),

MaxCount = maxCount

};

task.NextBackupTime = task.Schedule.GetFirstDateTime();

return task;

}

public void SetTask(FileBackupTask task)

{

textBoxFilename.Text = task.FileName;

scheduleControl.SetSchedule(task.Schedule);

textBoxCount.Text = task.MaxCount.ToString();

}

}

}

using SharedData;

namespace DesktopClient

{

public partial class TaskHistory : Form

{

public TaskHistory()

{

InitializeComponent();

}

public void UpdateHistoty(BackupTask task)

{

listView.Items.Clear();

foreach (var history in task.History)

{

ListViewItem lvi = new ListViewItem();

lvi.Tag = history;

lvi.Text = history.Date.ToString();

lvi.Name = "History";

string action = history.Action switch

{

TaskAction.Created => "Задание создано",

TaskAction.Backup => "Выполнено резервирование",

TaskAction.Restore => "Выполнено восстановление",

\_ => "-"

};

ListViewItem.ListViewSubItem subItem = new ListViewItem.ListViewSubItem(lvi, action);

lvi.SubItems.Add(subItem);

listView.Items.Add(lvi);

}

}

}

}

using System.Data;

using Npgsql;

using SharedData;

namespace DesktopClient

{

public partial class TaskPgSqlEdit : Form

{

public TaskPgSqlEdit()

{

InitializeComponent();

}

private void buttonConnect\_Click(object sender, EventArgs e)

{

comboBoxDatabases.Items.Clear();

try

{

NpgsqlConnectionStringBuilder connStringBuilder = new NpgsqlConnectionStringBuilder();

connStringBuilder.Host = textBoxHost.Text;

int port = 0;

int.TryParse(textBoxPort.Text, out port);

connStringBuilder.Port = port;

connStringBuilder.Username = textBoxUserId.Text;

connStringBuilder.Password = textBoxPass.Text;

using (NpgsqlConnection connection = new NpgsqlConnection(connStringBuilder.ConnectionString))

{

connection.Open();

NpgsqlDataAdapter adapter = new NpgsqlDataAdapter("SELECT datname FROM pg\_database WHERE datistemplate = false;", connection);

DataSet dataSet = new DataSet();

adapter.Fill(dataSet);

foreach(DataRow row in dataSet.Tables[0].Rows)

comboBoxDatabases.Items.Add(row[0].ToString());

labelConnectionStatus.Text = "Соединение установлено";

}

}

catch (Exception ex)

{

labelConnectionStatus.Text = "Соединение не установлено";

}

if (comboBoxDatabases.Items.Count > 0)

comboBoxDatabases.SelectedIndex = 0;

}

public PGBackupTask GetTask()

{

int port = 0;

int.TryParse(textBoxPort.Text, out port);

int maxCount = 1;

int.TryParse(textBoxCount.Text, out maxCount);

if (maxCount <= 0)

maxCount = 1;

var task = new PGBackupTask()

{

Host = textBoxHost.Text,

Port = port,

UserId = textBoxUserId.Text,

Password = textBoxPass.Text,

DbName = comboBoxDatabases.Text,

Schedule = scheduleControl.GetSchedule(),

MaxCount = maxCount

};

task.NextBackupTime = task.Schedule.GetFirstDateTime();

return task;

}

public void SetTask(PGBackupTask task)

{

textBoxHost.Text = task.Host;

textBoxPort.Text = task.Port.ToString();

textBoxUserId.Text = task.UserId;

textBoxPass.Text = task.Password;

comboBoxDatabases.Text = task.DbName;

scheduleControl.SetSchedule(task.Schedule);

textBoxCount.Text = task.MaxCount.ToString();

}

}

}

# Приложение 4.

# Реализация классов данных для работы остальных компонентов программы

using System.Runtime.Serialization.Formatters.Binary;

namespace SharedData

{

[Serializable]

public class FilesInfo

{

public FilesInfo() { Data = new List<FileStruct>(); }

private FilesInfo(List<FileStruct> binFiles) { Data = binFiles; }

[Serializable]

public struct FileStruct

{

public int Id;

public DateTime Date;

public string NameFile;

public byte[] Bin;

}

public List<FileStruct> Data { get; private set; }

public void Add(int id, DateTime date, string nameFile, byte[] bin) =>

Data.Add(new FileStruct() { Id = id, Date = date, NameFile = nameFile, Bin = bin });

public byte[] ToArray()

{

var binFormatter = new BinaryFormatter();

var mStream = new MemoryStream();

binFormatter.Serialize(mStream, Data);

return mStream.ToArray();

}

public static FilesInfo FromBin(byte[] bin)

{

var mStream = new MemoryStream();

var binFormatter = new BinaryFormatter();

mStream.Write(bin, 0, bin.Length);

mStream.Position = 0;

return new FilesInfo(binFormatter.Deserialize(mStream) as List<FileStruct>);

}

}

}

using System.Runtime.Serialization.Formatters.Binary;

namespace SharedData

{

[Serializable]

public class RestoreTask

{

public int Id = -1;

public string Path = "";

public RestoreTask(int id, string path)

{

Id = id;

Path = path;

}

public byte[] ToArray()

{

using (var ms = new MemoryStream())

{

var binFormatter = new BinaryFormatter();

binFormatter.Serialize(ms, this);

return ms.ToArray();

}

}

public static RestoreTask FromArray(byte[] array)

{

using (var ms = new MemoryStream())

{

var binFormatter = new BinaryFormatter();

ms.Write(array, 0, array.Length);

ms.Position = 0;

return binFormatter.Deserialize(ms) as RestoreTask;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SharedData

{

public enum ScheduleType

{

Daily = 0,

Weekly = 1,

Monthly = 2,

}

[Serializable]

public class Schedule

{

public Schedule()

{

Type = ScheduleType.Daily;

Time = TimeSpan.MinValue;

DayOfWeek = DayOfWeek.Sunday;

DayOfMonth = 1;

}

public ScheduleType Type { get; set; }

public TimeSpan Time { get; set; }

public DayOfWeek DayOfWeek { get; set; }

public int DayOfMonth { get; set; }

public DateTime GetFirstDateTime()

{

return Type switch

{

ScheduleType.Daily => DateTime.Today + Time,

ScheduleType.Weekly => GetNextWeekDay(DateTime.Today, DayOfWeek) + Time,

ScheduleType.Monthly => GetNextMonthDay(DateTime.Today, DayOfMonth) + Time,

\_ => DateTime.MaxValue

};

}

public DateTime GetNextDateTime(DateTime prev)

{

return Type switch

{

ScheduleType.Daily => prev.AddDays(1),

ScheduleType.Weekly => prev.AddDays(7),

ScheduleType.Monthly => GetNextMonthDay(prev.AddDays(1), DayOfMonth) + Time,

\_ => DateTime.MaxValue

};

}

public static DateTime GetNextWeekDay(DateTime start, DayOfWeek day)

{

int daysToAdd = ((int)day - (int)start.DayOfWeek + 7) % 7;

return start.AddDays(daysToAdd);

}

public static DateTime GetNextMonthDay(DateTime start, int day)

{

if (start.Day > day)

start = start.AddMonths(1);

int daysInMonth = DateTime.DaysInMonth(start.Year, start.Month);

return new DateTime(start.Year, start.Month, daysInMonth > day ? day : daysInMonth);

}

}

}

using Network.Attributes;

using Network.Packets;

namespace SharedData;

[PacketRequest(typeof(SharedRequest))]

public class SharedResponse : ResponsePacket

{

public SharedResponse(string result, SharedRequest request)

: base(request)

{

this.Result = result;

}

public string Result { get; set; }

}

public class SharedRequest : RequestPacket

{

public SharedRequest()

{

Command = "null";

Data = new byte[10];

}

public string Command { get; set; }

public byte[] Data { get; set; }

}using System;

namespace SharedData

{

public enum TaskStatus

{

New = 0,

Working,

Error\_NoFile,

Error\_DbConnect,

Error\_Quota,

Disabled

}

public enum TaskAction

{

Created = 0,

Backup,

Restore

}

[Serializable]

public struct TaskHistory

{

public DateTime Date;

public TaskAction Action;

}

[Serializable]

public abstract class BackupTask

{

public int Id = -1; //ID задания

public List<DateTime> BackupTimes = new List<DateTime>(); //Список времен текущих резервных копий

public DateTime NextBackupTime = DateTime.MaxValue; //Время следующего резервирования

public Schedule Schedule = new Schedule();

public TaskStatus Status = TaskStatus.New; //Статус

public int MaxCount = 1; //Количество хранимых резервных копий

public List<TaskHistory> History = new List<TaskHistory>();

public string Address = "";

public int LocalPort = 0;

public void UpdateNextBackupTime()

{

NextBackupTime = Schedule.GetNextDateTime(NextBackupTime);

}

public string GetStatusString()

{

return Status switch

{

TaskStatus.New => "Запланировано",

TaskStatus.Working => "Выполняется",

TaskStatus.Error\_NoFile => "Файл не найден",

TaskStatus.Error\_DbConnect => "Ошибка подключения",

TaskStatus.Error\_Quota => "Превышение квоты",

TaskStatus.Disabled => "Отключено",

\_ => "-"

};

}

public string GetScheduleString()

{

return Schedule.Type switch

{

ScheduleType.Daily => "Ежедневно",

ScheduleType.Weekly => "Еженедельно",

ScheduleType.Monthly => "Ежемесячно",

\_ => "-"

};

}

public void AddAction(TaskAction action) => History.Add(new TaskHistory { Date = DateTime.Now, Action = action });

}

[Serializable]

public class FileBackupTask : BackupTask

{

public string FileName = "";

}

[Serializable]

public class SQLBackupTask : BackupTask

{

public string Server = "";

public string Login = "";

public string Password = "";

public string DbName = "";

}

[Serializable]

public class PGBackupTask : BackupTask

{

public string Host = "";

public string UserId = "";

public string Password = "";

public string DbName = "";

public int Port = 0;

}

}

using System;

using System.Collections.Generic;

using System.Runtime.Serialization.Formatters.Binary;

using Newtonsoft.Json;

namespace SharedData

{

[Serializable]

public class TasksInfo

{

public TasksInfo() { Data = new Dictionary<int, BackupTask>(); UsedQuota = 0; MaxQuota = 10000000; }

public Dictionary<int,BackupTask> Data { get; private set; }

public long UsedQuota { get; set; }

public long MaxQuota { get; set; }

public byte[] ToArray()

{

using (var ms = new MemoryStream())

{

var binFormatter = new BinaryFormatter();

binFormatter.Serialize(ms, this);

return ms.ToArray();

}

}

public static TasksInfo FromArray(byte[] array)

{

using (var ms = new MemoryStream())

{

var binFormatter = new BinaryFormatter();

ms.Write(array, 0, array.Length);

ms.Position = 0;

return binFormatter.Deserialize(ms) as TasksInfo;

}

}

public void SaveToFile(string filename)

{

File.WriteAllText(filename, JsonConvert.SerializeObject(this,

new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto }));

}

public async Task SaveToFileAsync(string filename)

{

await File.WriteAllTextAsync(filename, JsonConvert.SerializeObject(this,

new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto }));

}

public static TasksInfo LoadFromFile(string filename)

{

if (File.Exists(filename))

{

TasksInfo taskInfo = JsonConvert.DeserializeObject<TasksInfo>(File.ReadAllText(filename),

new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.Auto });

if (taskInfo != null)

{

return taskInfo;

}

}

return new TasksInfo();

}

public int GetNextId()

{

if (Data.Count == 0)

return 1;

return Data.Keys.Max() + 1;

}

}

}