МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

ОТЧЕТ

О РЕЗУЛЬТАТАХ ИНДИВИДУАЛЬНОЙ

ПРЕДДИПЛОМНОЙ ПРАКТИКИ

ФГАОУ ВО «ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(наименование организации)

КАФЕДРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

(наименование структурного подразделения)

|  |  |  |
| --- | --- | --- |
| *ка была групповой, то указываются ФИО всех участвовавших обучаю* | | |
| Выполнил  обучающийся 4 курса,  МОиАИС-184 группы |  | Поляков И. А. |
|  | (подпись) | (ФИО) |
| Руководитель практики  от института |  | Ялдыгин В. Б. |
|  | (подпись) | (ФИО) |

Защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022

Результаты

экзамена / зачета \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись руководителя практики от института)

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc106250598)

[ГЛАВА 1 ВИЗУАЛИЗАЦИЯ УЧЕБНОГО ПЛАНА 5](#_Toc106250599)

[1.1 Описание предметной области 5](#_Toc106250600)

[1.2 Постановка задачи 5](#_Toc106250601)

[1.3 Используемые технологии 6](#_Toc106250602)

[1.4 Алгоритм расчета привязки номера недели учебного года 7](#_Toc106250603)

[1.5 Предложения по используемым технологиям 10](#_Toc106250604)

[ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ И ВОССТАНОВЛЕНИЯ ДАННЫХ 11](#_Toc106250605)

[2.1 Актуальность темы 11](#_Toc106250606)

[2.2 Постановка задачи. Методы и алгоритмы 13](#_Toc106250607)

[2.3 Архитектура системы 15](#_Toc106250608)

[2.4 Разработка программного обеспечения 17](#_Toc106250609)

[2.5 Реализованные модули, классы и методы 18](#_Toc106250610)

[ГЛАВА 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 25](#_Toc106250611)

[3.1 Установка программного обеспечения 25](#_Toc106250612)

[3.2 Описание программного обеспечения 25](#_Toc106250613)

[ЗАКЛЮЧЕНИЕ 30](#_Toc106250614)

[СПИСОК ЛИТЕРАТУРЫ 31](#_Toc106250615)

[Приложение 1 32](#_Toc106250616)

[Приложение 2 36](#_Toc106250617)

[Приложение 3 41](#_Toc106250618)

[Приложение 4 44](#_Toc106250619)

# 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.

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 разнообразные 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.

**Цели и задачи практики** – разработать эффективный механизм резервного копирования и восстановления данных с удаленным хранением копий.

Для достижения поставленной цели необходимо решить следующие задачи:

* + изучить основные принципы организации процесса резервного копирования и восстановления данных;
  + выделить оптимальные способы резервного копирования, а также выбрать место хранения резервных копий и способ их переноса;
  + разработать программное обеспечение, реализующее функции резервного копирования, синхронизации и восстановления данных;
  + централизованное автоматизированное управление процедурами резервного копирования, хранения и восстановления данных (далее - РК);
  + создание резервных копий данных на устройствах хранения данных;
  + управление расписанием создания резервных копий;
  + централизованно- управляемое восстановление данных;

Дата прохождения практики: 11.05.2022 по 04.06.2022.

# ГЛАВА 1 ВИЗУАЛИЗАЦИЯ УЧЕБНОГО ПЛАНА

## 1.1 Описание предметной области

Предметная область проекта – учебный план. Данная область была выбрана, потому что предоставленный в открытом доступе учебный план на сайте ТюмГУ сложен к зрительному восприятию студентами и преподавателями. Поскольку предоставленный материал избыточен и имеет неудобный формат документа учебного плана для быстрого поиска информации об изучаемых дисциплинах. Было насчитано более 400 дисциплин учебного плана ТюмГУ на официальном сайте. Возможно, данный формат публикации учебного плана хранит в себе какой-то смысл, но студенту для поиска нужной ему информации требуется затратить достаточно времени.

Однако, имеются не только названия дисциплин, но и объем образовательной части, контактная работа, самостоятельная работа, распределение нагрузки по периодам, количество недель, компетенции, распределение форм промежуточной аттестации по периодам. И всё это только разделы, содержащие в себе множество информации.

## 1.2 Постановка задачи

Дано: публикация учебного плана на сайте ТюмГУ в формате PDF документа, в каждом из документов имеются следующие атрибуты: ID, название дисциплины, количество лекционных часов, количество практических часов, специальность, компетенция, приобретаемые в ходе прохождения курса дисциплины, форма сдачи дисциплины.

Необходимо разработать web-приложение, которое будет отображать учебный план в распределенном виде по курсам, семестрам и доступной форме для визуального восприятия.

На основе полученных данных требуется выгрузка учебного плана из доступного ресурса, преобразование форматов из PDF в CSV файл, проектирование базы данных для хранения данных, разработать дизайн и интерфейс, далее выгрузить эти данные в WEB-приложение.

Цель проекта: реализовать веб-сайт, на котором будет размещаться учебный план по направлению обучения студента в визуально наглядной форме с возможностью подробного просмотра информации об изучаемых дисциплинах.

Основные задачи проекта:

* ознакомиться с открытой литературой по предметной области;
* спроектировать БД;
* выгрузить учебный план из доступных источников;
* разработать сайт;
* протестировать работоспособность приложения.

## 1.3 Используемые технологии

При реализации проекта использовались следующие технологии: в качестве основного языка программирования был выбран Php, данный язык предоставляет много возможностей для разработчика. Производительность языка программирования Php вполне достаточно для веб-ресурсов. Да, есть более быстрые и производительные семейства, например, «С», но конкретно для веб-разработок и приложений, мощностей и параметров языка Php более чем достаточно. Вся серверная часть была построена именно на нем.

Также использован язык программирования Javascript для написания frontend- и backend-частей сайта. К нему были использованы Webpack инструмент, позволяющий скомпилировать JavaScript модули в единый JS-файл и кроссплатформенная библиотека Jquey используемая для упрощения клиентского сценария HTML.

Для разработки интерфейса веб-приложения использовались HTML, СSS

и SCSS с библиотекой Bootstrap. HTML – это стандартизированный язык разметки веб-страниц, а SCSS, используемый для описания внешнего вида сайта. Bootstrap – набор шаблонов для создания интерфейса веб-приложений. Последние две технологии из вышеперечисленных были выбраны из-за своей лёгкости в использовании, так как не пришлось бы расписывать стили [2, с.28] элементов HTML-страниц с нуля, а только их видоизменять.

PhpStorm - удобная интегрированная среда разработки (IDE), с хорошим функционалом для сборки и тестирования проекта. Представляет собой интеллектуальный редактор для PHP, HTML и JavaScript с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для PHP и JavaScript;

Bootstrap - свободный фреймворк, используемый для быстрой вёрстки адаптивного дизайна сайта и включающий в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения;

Visual Studio Code - редактор кода, используемый при работе с сервером;

MSSQL - система управления реляционными базами данных базами данных (далее – РСУБД), используемая для хранения большого количества информации на сервере;

Open Server – портативный локальный WAMP/WNMP сервер, имеющий многофункциональную управляющую программу и большой выбор подключаемых компонентов;

Git –система контроля версий (VCS), позволяющая отслеживать и фиксировать изменения в коде: восстановить код в случае сбоя или откатить до более ранних версий. Удобный инструмент для взаимодействия нескольких разработчиков на одном проекте.

## 1.4 Алгоритм расчета привязки номера недели учебного года

Поскольку в календарном учебном графике, размещенном на сайте ТюмГУ, обозначения учебного процесса привязаны к номеру недели учебного года, а не к конкретной дате, важно правильно рассчитать номер учебной недели на выбранную дату на календаре пользователя. Для расчета номера недели на языке JavaScript реализована функция getWeek(), которая для каждого дня отрисованного календаря возвращает номер учебной недели(с учетом того, что учебная неделя начинается 1 сентября).

Date.prototype.getWeek = function () {

var target = new Date(this.valueOf());

var dayNr = (this.getDay() + 6) % 7;

target.setDate(target.getDate() - dayNr + 1

var firstThursday = target.valueOf();

target.setMonth(8, 1);//число с которого начинается первая неделя

if (target.getDay() != 1) {

target.setMonth(8, 1 + ((-1 - target.getDay()) + 7) % 7);

}

var a = 1 + Math.ceil((firstThursday - target) / 604800000);

if (a > 0) return a;

else return (52 - Math.abs(a));

}

Блок-схема алгоритма предоставлена на рисунке 1.

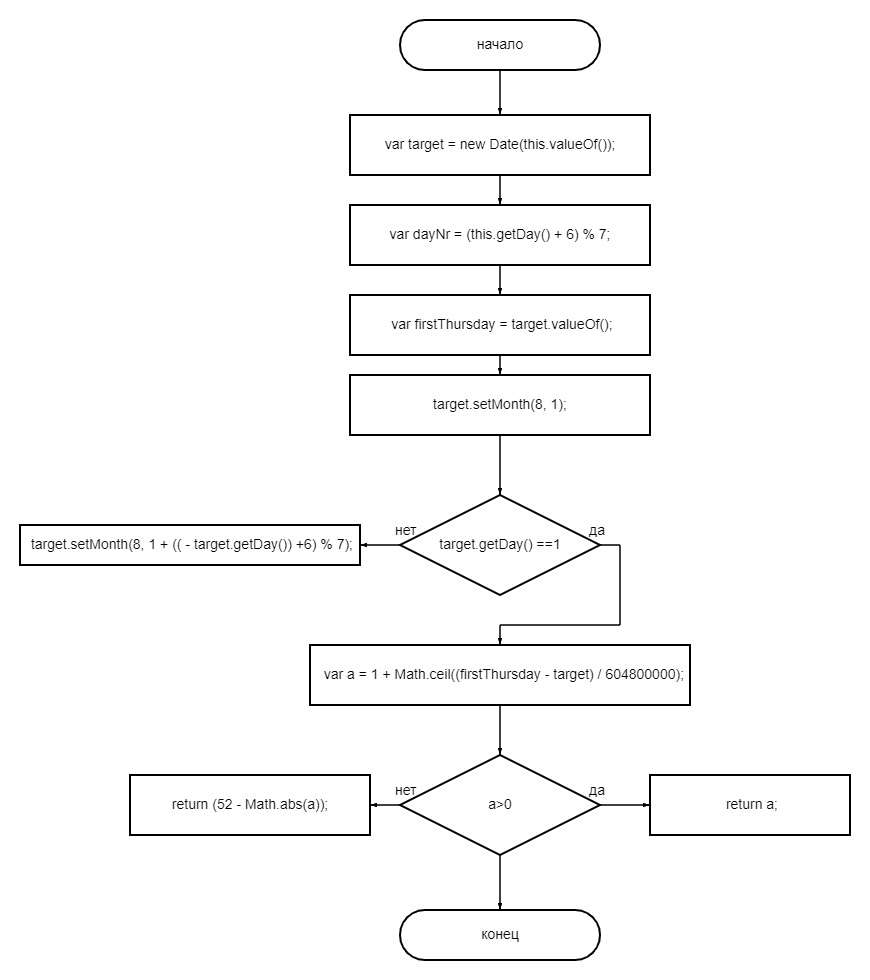


Рис. 1. Блок-схема алгоритма расчета номера учебной недели

В функции сначала инициализируется переменная со значением даты в календаре, равной количеству миллисекунд, прошедших с полуночи 1 января 1970 года. Далее высчитывается номер дня недели. В качестве начала отсчета устанавливается 1 сентября (target.SetMonth(8,1)). Далее, если номер дня недели равен единице(понедельник), то в переменной target пересчитывается день месяца по формуле 1.

где d- порядковый номер дня недели даты, с которой отсчитывается учебный год

Далее в переменную a записывается номер недели, рассчитанный по формуле 2.

где math.ceil - функция, возвращающая наименьшее целое, большее или равное аргументу,

D – дата понедельник недели, содержащей искомую дату, которая выражена в миллисекундах, прошедших с полуночи 1 января 1970 года,

t- дата первого понедельника месяца начала учебной недели (в миллисекундах).

Если переменная a отрицательной она пересчитывается по формуле 3. Далее возвращается значение переменной a, которая хранить номер недели на конкретную дату календаря.

## 1.5 Предложения по используемым технологиям

Проект разрабатывался группой разработчиков, так что было предложено использовать Github для удобства совместной разработкой. Также для повышения скорости верстки обсуждалось применение Bootstrap, а для запуска сервера на локальной машине XAMPP

На момент начала практики студенты уже определились с используемыми алгоритмами и реализовали их, поэтому помощи в этом не потребовалось, вопросы со стороны студентов в основном было связаны с представлением результатов, поэтому обсуждался вид презентации и отчета.

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

## 2.1 Актуальность темы

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 представлены на рисунке 2 [6].



Рис. 2. 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.

Задачи:

* изучить существующие аналоги;
* сформулировать требования к разрабатываемой системе;
* спроектировать разрабатываемую систему;
* определить форматы и структуру данных системы;
* разработать программное обеспечение, реализующее резервирования и восстановление для наиболее востребованных типов источников (файлы, каталоги, базы данных SQL Server) и позволяющее добавлять новые типы источников.

## 2.2 Постановка задачи. Методы и алгоритмы

Основной задачей разрабатываемой системы является надежное создание и управление жизненным циклом резервных копий, а также восстановление из них в случае утери или порчи данных, участвующих в технологических процессах и бизнес-процессах.

Программное обеспечение должно обеспечить решение следующих задач:

* централизованное автоматизированное управление процедурами резервного копирования, хранения и восстановления данных;
* создание резервных копий данных (файлов, баз данных) на устройствах хранения данных;
* централизовано-управляемое восстановление данных из резервных копий;
* управление расписанием создания резервных копий;
* восстановление данных;
* проверка прав пользователей при выполнении задач по резервному копированию и восстановлению.

Приложение поддерживает следующие схемы резервирования данных.

* Полное резервное копирование:

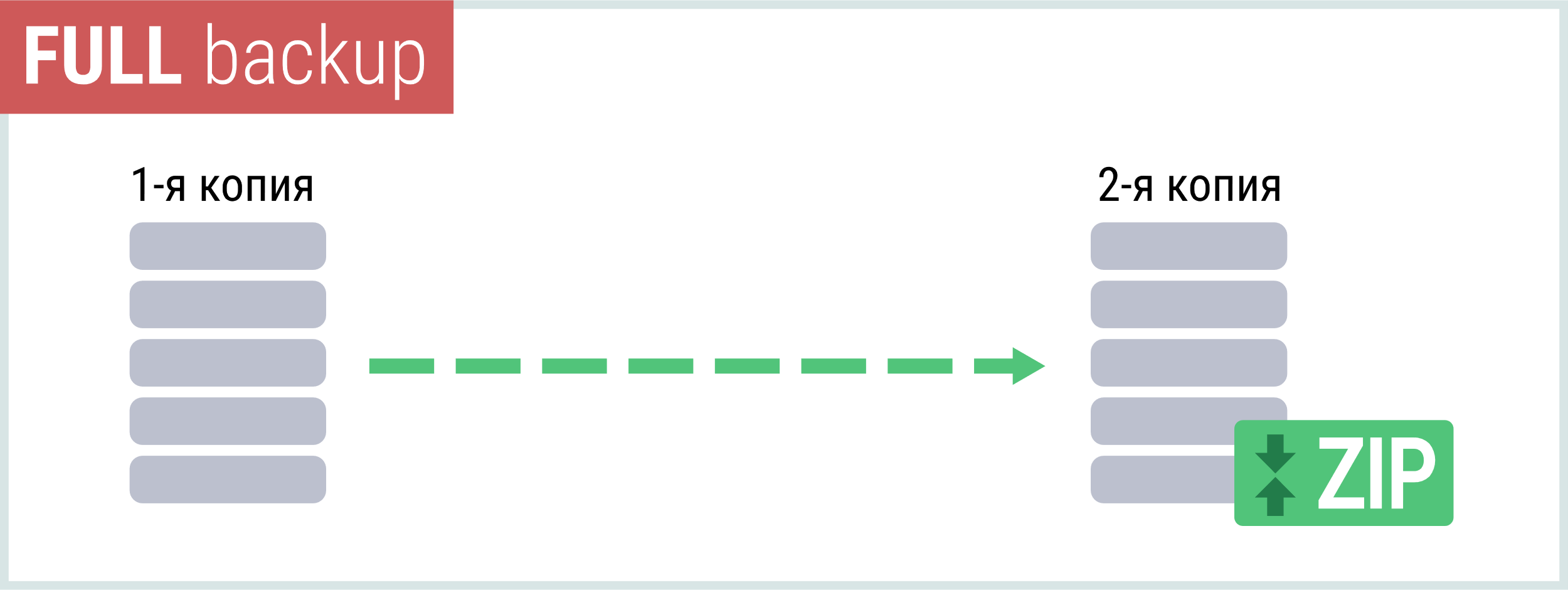


Рис. 3. Полное резервное копирование

При полном бэкапе каждый раз создается полная копия **всей системы,** точнее, всех тех данных, которые вы определили для резервного копирования при постановке задачи. Для уменьшения итогового объема резервной копии все данные сжимаются в архив. Таким образом, в вашем хранилище при полном резервном копировании с заданной периодичностью появляются архивы, где данные в основной своей массе дублируются (поскольку на протяжении долгого времени не изменяются). Это серьезный недостаток, ведь расходуется огромный объем ресурсов: место в хранилище, время создания и процессорное время, вычислительные мощности, наконец, ресурсы трафика при транспортировке архивов в удаленную СХД.

Метод полного копирования ранее был очень распространенным из-за высокой надежности, однако в чистом виде на сегодняшний день он признан малоэффективным. Например, для резервного копирования невысокой глубиной (менее двух недель) или с высокой частотой (раз в сутки, раз в несколько часов) полный бэкап чрезмерно расходует ресурсы.

Немного спасет ситуацию механизм ***дедупликации*** – выявление и удаление дублирующихся данных в полных копиях. Он также задается специальными программными средствами как на уровне СХД или сервера, так и на клиенте непосредственно. Статистика в некоторых источниках приводит впечатляющие результаты степени дедупликации – от 90% до 98%.

Преимуществом полного бэкапа можно назвать скорость восстановления: когда данные поднимаются из одного архива, это происходит быстрее, чем при дифференциальном или инкрементном бэкапе. На сегодняшний день метод полного резервного копирования, как правило, используется исключительно как базовый в сочетании с другими методами, менее ресурсоемкими [3].

## 2.3 Архитектура системы

Архитектура разрабатываемой системы приведена на рисунке 4.

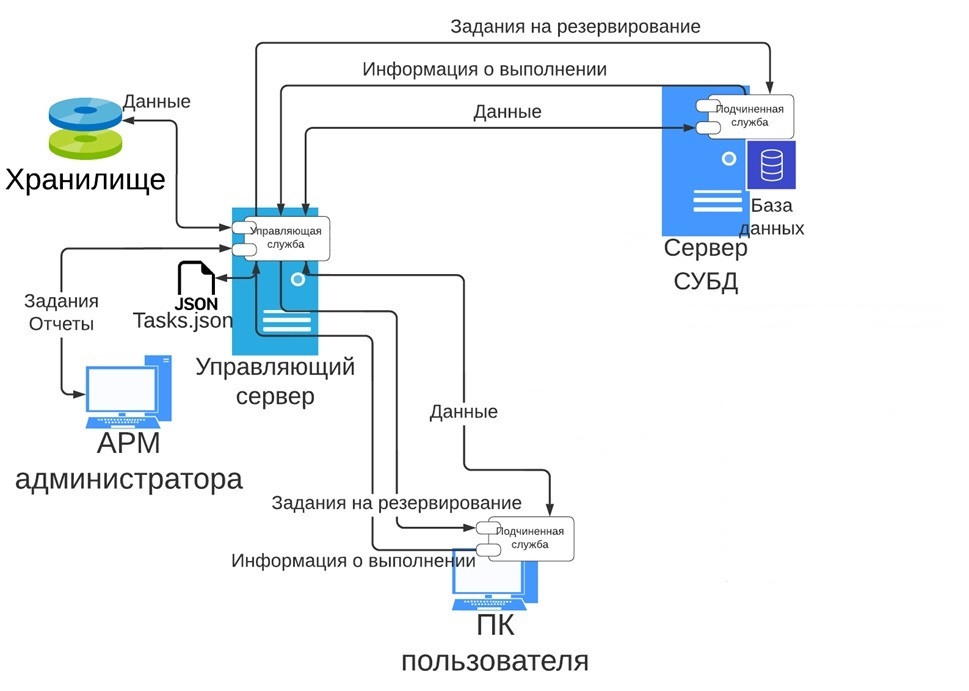


Рис. 4. Архитектура системы

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, номер версии задания, 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 созданные задания, просматривать информацию о результатах выполнения заданий, а также удалять задания. После изменения заданий информация об изменениях сразу отправляется на подчиненные службы. Информация о пользователе, выполнившем создание, изменение или удаление задания, сохраняется на сервере (действия пользователей записываются в лог, информация о пользователе, последним изменившим задачу, дополнительно сохраняется в самой задаче). Информация о задания хранится в файле Tasks.json.

## 2.4 Разработка программного обеспечения

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.
* ClientConfig – класс для хранения конфигурации подчиненной службы, используется для автономного подключения к серверу.

Взаимодействие между приложениями осуществляется по сети по протоколу TCP/IP. Для этих целей используется библиотека Network [1]. Библиотека позволяет посредством сети получать и отправлять пакеты с различными данными. Для передачи списка задач используется пакет ‘tasks’. Передача файлов осуществляется с помощью пакета ‘backup’.

Управляющая служба сохраняет список задач в файле Tasks.json. Для сохранения списка в формате JSON применяется библиотека Newtonsoft.Json [2].

## 2.5 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.

Методы:

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.

Методы:

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.

Методы:

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.

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 поключения из файла конфигурации.

public static void SaveConfig() – сохраняет параметры подключения в файл конфигурации.

Класс Login. Форма с параметрами подключения.

Методы:

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.

Методы:

public Main() – конструктор по умолчанию.

public void UpdateTable(TasksInfo tasks) – обновляет список заданий в окне согласно переданому 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 listView1\_SelectedIndexChanged(object sender, EventArgs e) – При изменении выбранного создания в списке активирует или деативирует 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.

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

Класс Quota. Форма для настройки параметров квоты.

Методы:

public Quota() – конструктор по умолчанию. Заполняет поля формы необходимыми значениями.

Класс TaskDatabaseEdit. Форма для добавления и редактирования задания по резервированию БД.

Методы:

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.

Методы:

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.

Методы:

public TaskHistory() – конструктор по умолчанию.

public void UpdateHistoty(BackupTask task) – заполняет параметры таблицы данными из истории задания.

Модуль ServerService. Служба сервера.

Класс Server.

Методы:

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 входящий по сети пакетов.

void CheckBackups() – Проверяет папку с бэкапами. Удаляет лишние бэкапы.

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.

Методы:

public FilesInfo() – конструктор по умолчанию.

private FilesInfo(List<FileStruct> binFiles) – консруктор, создает экземпляр класса из списка файлов.

public void Add(int id, DateTime date, string nameFile, byte[] bin) – добавляет файл и информаци 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.

Методы:

public void UpdateNextBackupTime() – вычисляет и обновляет дату и время следующего бэкапа.

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.

Методы:

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.

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 бэкапа (поле TypeTimeBackup);
* статус бэкапа (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 2 типа резервирования: резервирование файлов (класс FileBackupTask) и резервирование базы данных SQL Server (класс DbBackupTask).

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 Path – путь к резервируемому файлу или папке (если путь заканчивается на слеш, то это папка, иначе файл).

Класс DbBackupTask (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.

Диаграмма классов заданий резервирования приведена на рисунке 5:

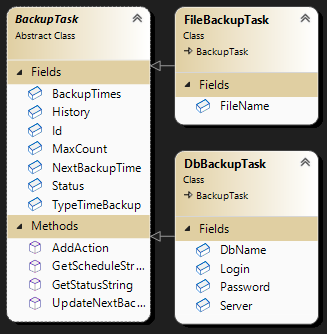


Рис. 5. Диаграмма классов заданий резервирования

В приложении 1 приведен основной код, отвечающий за передачу данных на сервер.

В приложении 2 приведен основной код, реализующий управляющую службу.

В приложении 3 приведен основной код, реализующий АРМ администратора.

В приложении 4 приведен основной код, реализующий подчиненную службу.

# ГЛАВА 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 3.1 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.

## 3.2 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 [4]; получатель – управляющий сервер, порт, настроенный при установке, по умолчанию порт 1708).

При запуске приложения откроется окно, изображенное на рисунке 6.



Рис. 6. 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 7):

Изображение выглядит как текст

Автоматически созданное описание

Рис. 7. Сообщение об ошибке при подключении к серверу

Если возникнет ошибка при аутентификации, будет выдано следующее окно (рисунок 8):



Рис. 8. 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 9.

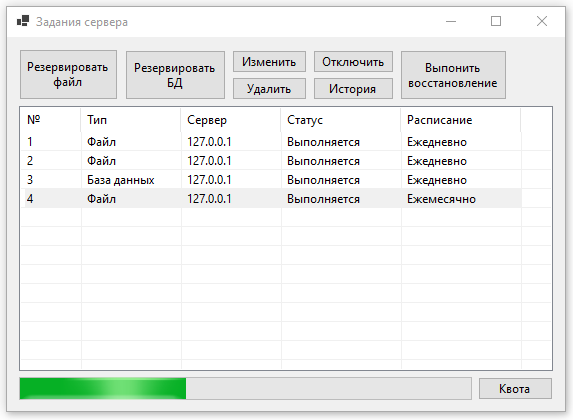


Рис. 9. Окно управления заданиями на резервирование

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 10.

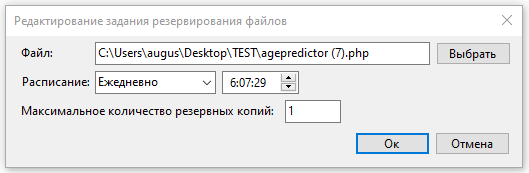


Рис. 10. Интерфейс создания и редактирования задачи файла.

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 11. Также есть настройка количества хранимых копий.

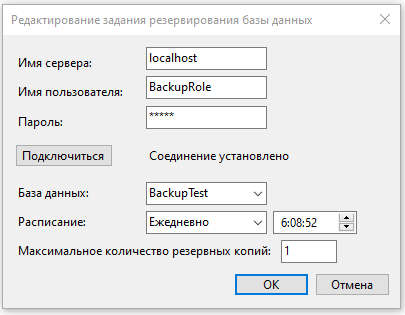


Рис. 11. Интерфейс создания и редактирования задачи базы данных.

Для восстановления базы данных или файла нужно выбрать из списка задачу и нажать кнопку восстановления.

Доступен просмотр истории действий с заданием по резервированию, рисунок 12 и отключение задания.

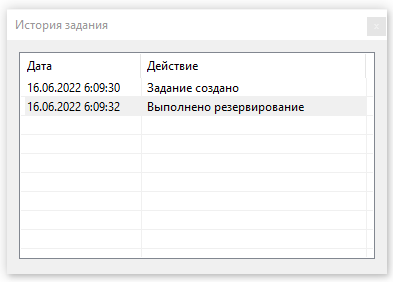


Рис. 12. Просмотр истории действий с заданием.

Также есть возможность управления квотой резервируемых данных на сервере, рисунок 13.

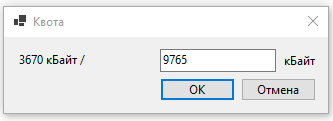


Рис. 13. Редактирование допустимого объема для резервирования.

# ЗАКЛЮЧЕНИЕ

В ходе работы были решеные следующие задачи:

* Сформулированы требования к разрабатываемой системе;
* Спроектирована разрабатывается система;
* Разработано программное обеспечение, реализующее резервирование и восстановление наиболее востребованных типов источников (файлы, базы данных SQL server) и позволяющее добавлять новые типы источников;

Во время работы получены навыки написания клиент-серверных приложений на языке C#, кроме того, закреплены навыки работы с windows forms и C#, базами данных.

# СПИСОК ЛИТЕРАТУРЫ

1. Работа с библиотекой Network // [Электронный ресурс]. URL: https://github.com/Toemsel/Network (дата обращения: 14.05.2022).
2. Работа с библиотекой Newtonsoft.Json на реальном примере // [Электронный ресурс]. URL: <https://habr.com/ru/post/481514/> (дата обращения: 14.05.2022).
3. Виды резервирования // [Электронный ресурс]. URL: https://www.sim-networks.com/ru/blog/backup-full-increment-differential (дата обращения: 21.05.2022).
4. 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).
5. Процесс в C# (класс Process) // Интернет портал dir.by [Электронный ресурс]. URL: <https://dir.by/developer/csharp/process/> (дата обращения: 18.05.2022).
6. 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).

# Приложение 1

Реализация данных для приема/передачи на сервер с помощью языка программирования C#.

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 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 int TypeTimeBackup = 0; //Тип расписания

public TaskStatus Status = TaskStatus.New; //Статус

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

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

public void UpdateNextBackupTime()

{

NextBackupTime = TypeTimeBackup switch

{

0 => NextBackupTime.AddDays(1), // Ежедневно

1 => NextBackupTime.AddDays(1), // Еженедельно

2 => NextBackupTime.AddMonths(1), //Ежемесячно

\_ => DateTime.MaxValue //Ошибочный тип

};

}

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 TypeTimeBackup switch

{

0 => "Ежедневно",

1 => "Еженедельно",

2 => "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 DbBackupTask : BackupTask

{

public string Server = "";

public string Login = "";

public string Password = "";

public string DbName = "";

}

}

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;

}

}

}

# Приложение 2

Реализация службы сервера.

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();

await host.RunAsync();

using Network;

using SharedData;

using System.IO;

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":

{

int id = BitConverter.ToInt32(packet.Data, 0);

if (\_tasks.Data.ContainsKey(id))

{

var task = \_tasks.Data[id];

string filename = "";

if (task is FileBackupTask)

{

filename = (task as FileBackupTask).FileName;

}

else if (task is DbBackupTask)

{

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

}

else

{

break;

}

var fullPath = Path.Combine(exePath,

BACKUP\_FOLDER,

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 {id}\n");

var restoreFile = new FilesInfo();

restoreFile.Add(id, DateTime.MinValue, filename, 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);

}

}

}

}

namespace ServerService

{

public class UserStruct

{

public string Username { get; set; }

public string Password { get; set; }

}

}

# Приложение 3

Реализация клиента.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

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);

}

}

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 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;

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(int id)

{

if (\_connection == null)

return;

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

{

Command = "restore",

Data = BitConverter.GetBytes(id)

});

}

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)

{

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 DbBackupTask)

taskType = "База данных";

else

taskType = "Неизвестно";

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

lvi.SubItems.Add(subItem);

subItem = new ListViewItem.ListViewSubItem(lvi, Globals.Config.ServerIp);

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.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.TypeTimeBackup = newTask.TypeTimeBackup;

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 DbBackupTask)

{

var dbTask = task as DbBackupTask;

var taskEditDlg = new TaskDatabaseEdit();

taskEditDlg.SetTask(dbTask);

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

return;

var newTask = taskEditDlg.GetTask();

dbTask.NextBackupTime = newTask.NextBackupTime;

dbTask.TypeTimeBackup = newTask.TypeTimeBackup;

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);

}

}

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.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;

Globals.SendRestore(task.Id);

}

}

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();

}

}

namespace DesktopClient

{

public partial class Quota : Form

{

public Quota()

{

InitializeComponent();

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

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

}

}

}

using System.Data;

using System.Data.SqlClient;

using SharedData;

namespace DesktopClient

{

public partial class TaskDatabaseEdit : Form

{

public TaskDatabaseEdit()

{

InitializeComponent();

comboBoxSchedule.SelectedIndex = 0;

}

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 DbBackupTask GetTask()

{

int maxCount = 1;

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

if (maxCount <= 0)

maxCount = 1;

var task = new DbBackupTask()

{

Server = textBoxServerName.Text,

Login = textBoxNameUser.Text,

Password = textBoxPass.Text,

DbName = comboBoxDatabases.Text,

NextBackupTime = dateTimePicker1.Value,

TypeTimeBackup = comboBoxSchedule.SelectedIndex,

MaxCount = maxCount

};

return task;

}

public void SetTask(DbBackupTask task)

{

textBoxServerName.Text = task.Server;

textBoxNameUser.Text = task.Login;

textBoxPass.Text = task.Password;

comboBoxDatabases.Text = task.DbName;

dateTimePicker1.Value = task.NextBackupTime;

comboBoxSchedule.SelectedIndex = task.TypeTimeBackup;

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

}

}

}

using SharedData;

namespace DesktopClient

{

public partial class TaskFileEdit : Form

{

public TaskFileEdit()

{

InitializeComponent();

comboBox1.SelectedIndex = 0;

}

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

{

var dialog = new OpenFileDialog();

if (!string.IsNullOrEmpty(textBoxFilename.Text))

{

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,

NextBackupTime = dateTimePicker1.Value,

TypeTimeBackup = comboBox1.SelectedIndex,

MaxCount = maxCount

};

return task;

}

public void SetTask(FileBackupTask task)

{

textBoxFilename.Text = task.FileName;

dateTimePicker1.Value = task.NextBackupTime;

comboBox1.SelectedIndex = task.TypeTimeBackup;

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);

}

}

}

}

# Приложение 4

Реализация службы клиента.

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;

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;

}

}

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 is FileBackupTask)

{

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

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

result = "OK";

}

else if (task is (DbBackupTask))

{

var dbTask = task as DbBackupTask;

string fullPath = Path.Combine(Path.GetTempPath(), file.NameFile);

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))

{

await conn.OpenAsync();

string query = @"RESTORE DATABASE @databaseName

FROM DISK = @localDatabasePath

WITH REPLACE";

var sqlCommand = new SqlCommand(query, conn);

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

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

await sqlCommand.ExecuteNonQueryAsync();

}

result = "OK";

}

catch (Exception ex)

{

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

}

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 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))

{

if (task == null)

continue;

if (task is FileBackupTask)

{

FileBackupTask fileTask = task as FileBackupTask;

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);

}

}

else //DbBackupTask

{

DbBackupTask dbTask = task as DbBackupTask;

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));

DbBackupTask updatedTask = dbTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.AddAction(TaskAction.Backup);

updatedTask.UpdateNextBackupTime();

updatedTask.BackupTimes.Add(backupTime);

updatedTasks.Add(updatedTask);

}

else

{

DbBackupTask 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");

DbBackupTask updatedTask = dbTask;

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

updatedTask.UpdateNextBackupTime();

updatedTasks.Add(updatedTask);

}

}

}

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 5

Реализация конфигурации.

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;

ServerPort;

Login;

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();

}

}}