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 обучающихся* | | |
| 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)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc105386691)

[ГЛАВА 1 ВИЗУАЛИЗАЦИЯ УЧЕБНОГО ПЛАНА 5](#_Toc105386692)

[1.1 Описание предметной области 5](#_Toc105386693)

[1.2 Постановка задачи 5](#_Toc105386694)

[1.3 Предложения по используемым технологиям 6](#_Toc105386695)

[ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЯ И ВОССТАНОВЛЕНИЯ ДАННЫХ 7](#_Toc105386696)

[2.1. Актуальность темы 7](#_Toc105386697)

[2.2. Постановка задачи. Методы и алгоритмы 8](#_Toc105386698)

[2.3. Архитектура системы 10](#_Toc105386699)

[2.4. Разработка программного обеспечения 11](#_Toc105386700)

[ГЛАВА 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 15](#_Toc105386701)

[3.1. Установка программного обеспечения 15](#_Toc105386702)

[3.2. Работа в АРМ администратора 15](#_Toc105386703)

[Список литературы 18](#_Toc105386704)

[Приложение 1 19](#_Toc105386705)

[Приложение 2 24](#_Toc105386706)

[Приложение 3 28](#_Toc105386707)

[Приложение 4 31](#_Toc105386708)

# 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. Студенту для поиска нужной ему информации требуется затратить достаточно много 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.

Учебный план в вузе — это документ, где указано, какие дисциплины, в каком порядке и каком объёме будет изучать студент конкретной специальности. Также из плана можно узнать, сколько практик будет у учащихся, по каким предметам и в какие семестры их ждёт защита курсовых. Удобный просмотр учебного плана позволяет не тратить много времени на поиск необходимой информации не только студентам, но и преподавателям.

Разработанное web-приложение, которое отображает учебный план в распределенном виде по курсам, семестрам и доступной форме для визуального восприятия, что значительно помогает сократить время для поиска необходимой информации по учебному плану, благодаря поиску по фразам, словам, ключевым индексам, поддерживает работу с Excel.

## 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.

## 1.3 Предложения по используемым технологиям

Bootstrap - верстка адаптивного дизайна сайта, со множеством шаблонов оформления компонентов, что позволяет сэкономить время при разработке дизайна.

Git - удобный инструмент для взаимодействия нескольких разработчиком в одном проекте.

XAMPP - кроссплатформенная сборка веб-сервера, содержащая Apache, MySQL, интерпретатор скриптов PHP, язык программирования Perl и большое количество дополнительных библиотек, позволяющих запустить полноценный веб-сервер.

# 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).

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

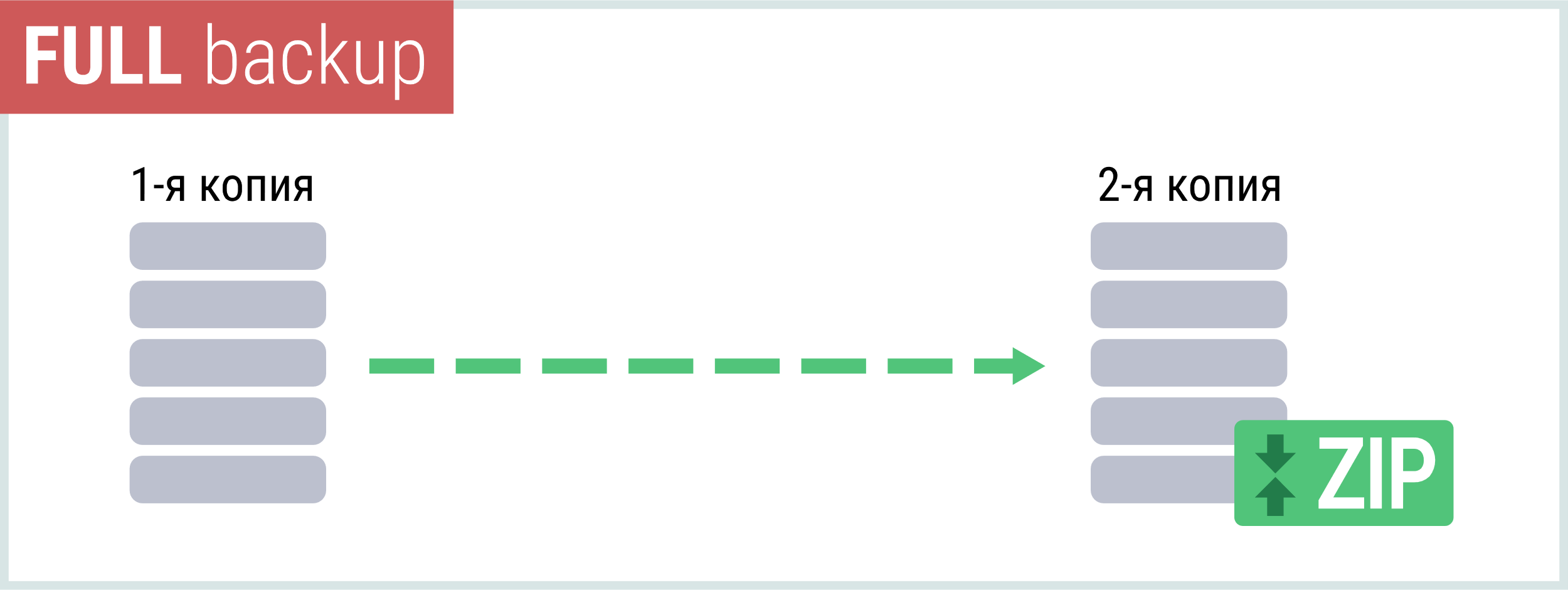
Основной задачей разрабатываемой системы является надежное создание и управление жизненным циклом резервных копий, а также восстановление из них в случае утери или порчи данных, участвующих в технологических процессах и бизнес-процессах.

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.

1. 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 разве что скорость восстановления: когда данные поднимаются из одного архива, это происходит быстрее, чем при инкрементальном или дифференцированном бэкапе. На сегодняшний день метод полного резервного копирования, как правило, используется исключительно как базовый в сочетании с другими методами, менее ресурсоемкими. Иногда такой подход называют также **смешанным** или **синтетическим** бэкапом [3].

## 2.3. Архитектура системы

Архитектура разрабатываемой системы приведена на рисунке 1.

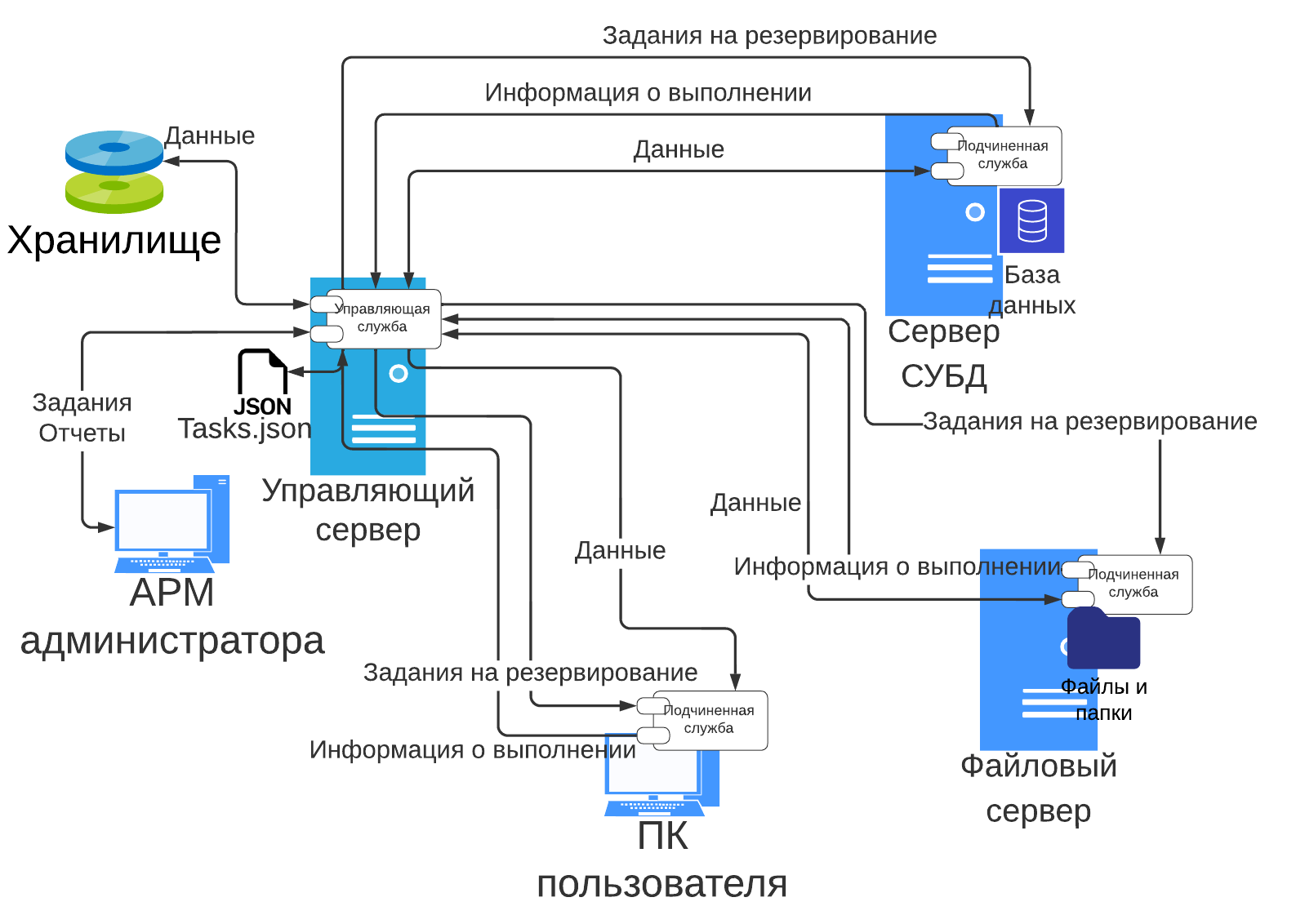


Рис. 1. Архитектура системы

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 4 проекта:

* 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.

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].

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 2:

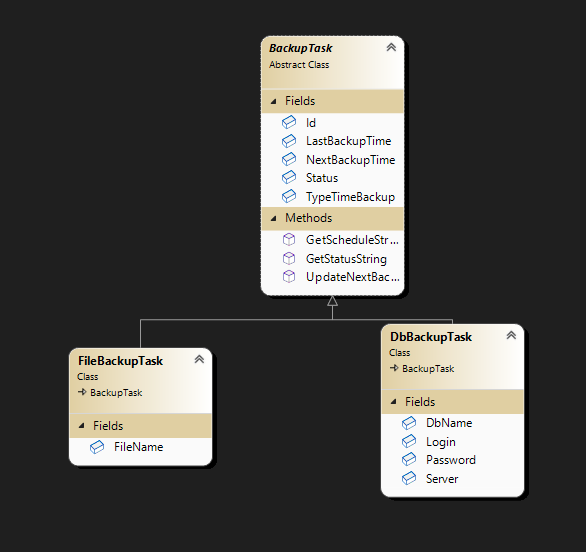


Рис. 2. Диаграмма классов заданий резервирования

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. Работа в АРМ администратора

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 3.



Рис. 3. Начально 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 4):

Изображение выглядит как текст

Автоматически созданное описание

Рис. 4. Сообщение об ошибке при подключении к серверу

Если возникнет ошибка при аутентификации, будет выдано следующее окно (рисунок 5):



Рис. 5. 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 6.

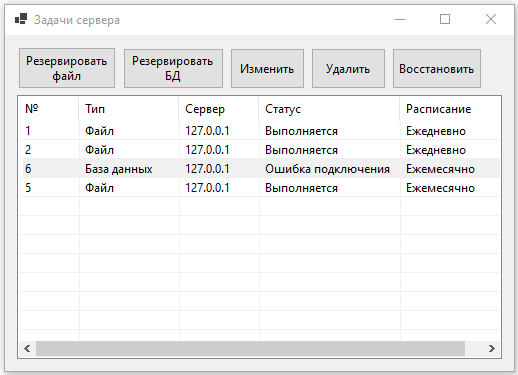


Рис. 6. Окно управления заданиями на резервирование

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 7.

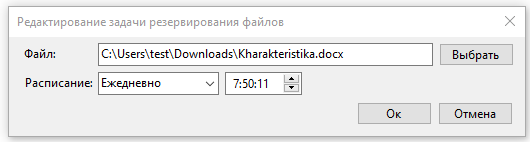


Рис. 7. Интерфейс создания и редактирования задачи файла.

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 8.

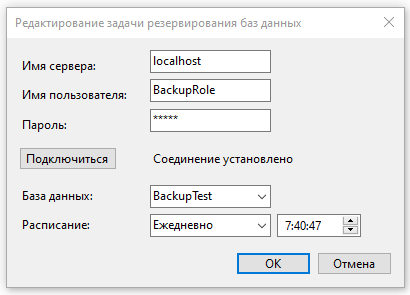


Рис. 7. Интерфейс создания и редактирования задачи базы данных.

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

# 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).

# Приложение 1

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

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 string NameFile;

public byte[] Bin;

}

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

public void Add(int id, string nameFile, byte[] bin) => Data.Add(new FileStruct() { Id = id, 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>);

}

}

}

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; }

}

namespace SharedData

{

public enum TaskStatus

{

New = 0,

Working,

Error\_NoFile,

Error\_DbConnect,

Restored

}

[Serializable]

public abstract class BackupTask

{

public int Id = -1;

public DateTime LastBackupTime = DateTime.MinValue;

public DateTime NextBackupTime = DateTime.MaxValue;

public int TypeTimeBackup = 0;

public TaskStatus Status = TaskStatus.New;

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 => "Новая",

TaskStatus.Working => "Выполняется",

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

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

TaskStatus.Restored => "Восстановлено",

\_ => "-"

};

}

public string GetScheduleString()

{

return TypeTimeBackup switch

{

0 => "Ежедневно",

1 => "Еженедельно",

2 => "Ежемесячно",

\_ => "-"

};

}

}

[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 = "";

}

}

namespace SharedData

{

[Serializable]

public class TasksInfo

{

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

private TasksInfo(Dictionary<int, BackupTask> tasksList) { Data = tasksList; }

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

public byte[] ToArray()

{

using (var ms = new MemoryStream())

{

var binFormatter = new BinaryFormatter();

binFormatter.Serialize(ms, Data);

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 new TasksInfo(binFormatter.Deserialize(ms) as Dictionary<int, BackupTask>);

}

}

public void SaveToFile(string filename)

{

File.WriteAllText(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();

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.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);

\_tasks.SaveToFile(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)

{

await Task.Run(() => Directory.CreateDirectory($@"{BACKUP\_FOLDER}\{file.Id}\"));

await File.WriteAllBytesAsync(Path.Combine(exePath, $@"{BACKUP\_FOLDER}\{file.Id}\{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}\{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, 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()

{

foreach (var dir in Directory.EnumerateDirectories(BACKUP\_FOLDER))

{

int numDir = 0;

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

{

Directory.Delete($@"{BACKUP\_FOLDER}/{dir}", true);

}

}

}

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.

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;

}

if (!Globals.Init())

{

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

return;

}

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Globals.MainWindow = new Main();

Application.Run(Globals.MainWindow);

}

}

namespace DesktopClient;

public static class Globals

{

public static string SERVER\_IP;// = "127.0.0.1";

public static int SERVER\_PORT;// = 1708;

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()

{

Process.Start(@"C:\WINDOWS\system32\sc.exe",

$"create test start=auto binPath=\"{Environment.CurrentDirectory}\\ClientService\\ClientService.exe\"");

Process.Start(@"C:\Windows\system32\sc.exe", $"start test \"{Environment.CurrentDirectory}\"");

return Connect();

}

public static bool Connect()

{

ConnectionResult result = ConnectionResult.TCPConnectionNotAlive;

\_connection = ConnectionFactory.CreateTcpConnection(SERVER\_IP, SERVER\_PORT, out result);

if (result == ConnectionResult.Connected)

{

connected = true;

\_connection.RegisterStaticPacketHandler<SharedRequest>(RecvHandler);

\_connection.TIMEOUT = 60000;

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("Имя пользователя или пароль неверен", "Ошибка входа", 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)

});

}

}

# Приложение 4

Реализация службы клиента.

namespace ClientService;

public class WinService : ServiceBase

{

private const string SERVICE\_NAME = "ClientService";

private string SERVER\_IP;

private const int SERVER\_PORT = 1708;

private string \_homePath;

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;

\_homePath = string.Empty;

\_tasks = new TasksInfo();

\_client = null;

}

protected override async void OnStart(string[] args)

{

if (args.Length == 0)

Stop();

await Connect();

\_homePath = args[0];

\_isWork = true;

await Task.Run(Handler);

}

protected override void OnStop()

{

base.OnStop();

\_isWork = false;

}

private async Task Connect()

{

var result = await ConnectionFactory.CreateTcpConnectionAsync(SERVER\_IP, SERVER\_PORT);

if (result.Item2 == ConnectionResult.Connected)

{

\_client = result.Item1;

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

\_client.TIMEOUT = 60000;

}

}

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

result = "OK";

}

else

{

//TODO:

}

}

break;

}

default:

result = "Unknown command";

break;

}

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

}

private async Task Handler()

{

while (\_isWork)

{

//Если нет подключения, то пытаемся подключиться к серверу

if (\_client == null)

{

await Connect();

}

else

{

var filesForBackup = new FilesInfo();

var updatedTasks = new List<BackupTask>();

foreach (var task in \_tasks.Data.Values.Where(task => task.NextBackupTime <= DateTime.Now))

{

if (task is FileBackupTask)

{

FileBackupTask fileTask = task as FileBackupTask;

if (File.Exists(fileTask.FileName))

{

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

FileBackupTask updatedTask = fileTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.UpdateNextBackupTime();

updatedTask.LastBackupTime = DateTime.Now;

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;

try

{

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

string fullPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, fileName);

string connString = $"Data Source = {dbTask.Server}; User ID = {dbTask.Login}; Password = {dbTask.Password}";

using (SqlConnection connection = new SqlConnection(connString))

{

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();

}

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

DbBackupTask updatedTask = dbTask;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.UpdateNextBackupTime();

updatedTask.LastBackupTime = DateTime.Now;

updatedTasks.Add(updatedTask);

}

catch (Exception ex)

{

File.AppendAllText(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt"), ex.Message);

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)

{

foreach (var task in updatedTasks)

{

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

}

//send tasks list

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

{

Command = "tasks",

Data = \_tasks.ToArray()

});

}

}

await Task.Delay(1000);

}

}

}