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 |  | Павлова Е.А. |
|  | (подпись) | (ФИО) |

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

Результаты

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

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc105049505)

[Раздел 1 (проект 2 курса) 5](#_Toc105049506)

[ГЛАВА 2. РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЯ 7](#_Toc105049507)

[2.1 Пользовательское приложение 7](#_Toc105049508)

[Список литературы 12](#_Toc105049509)

[Приложение 1 12](#_Toc105049510)

[Приложение 2 15](#_Toc105049511)

[Приложение 3 17](#_Toc105049512)

[Приложение 4 19](#_Toc105049513)

# 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 02.06.2022.

# Раздел 1 (проект 2 курса)

**Название проекта**

**Проблема**

Предоставленный учебный план на сайте ТюмГУ сложен к зрительному восприятию студентам, так как избыточен и представлен в неудобном формате для чтения.

**Актуальность**

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.

**для кого предназначен проект**

Студенты и преподаватели.

**насколько полезен**

Значительно помогает сократить время для поиска необходимой информации по учебному плану, благодаря поиску по фразам, словам, ключевым индексам, поддерживает работу с Excel.

**что нужно для реализации проекта**

Исходные данные с сайта ТюмГУ и хостинг сервера.

**описание данных и их источников**

Исходные данные с сайта ТюмГУ в форматах pdf/csv

**ваши предложения (с обоснованием) по использованию технологий и моделей обработки данных.**

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 WINDOWS-ПРИЛОЖЕНИЯ

## 2.1 Пользовательское приложение

Приложение разрабатывается с помощью языка программирования C#. В качестве среды программирования используется Visual Studio 2022. Визуальная часть программы основана на использовании Windows Forms (WinForms). Платформа – NET 6.0.

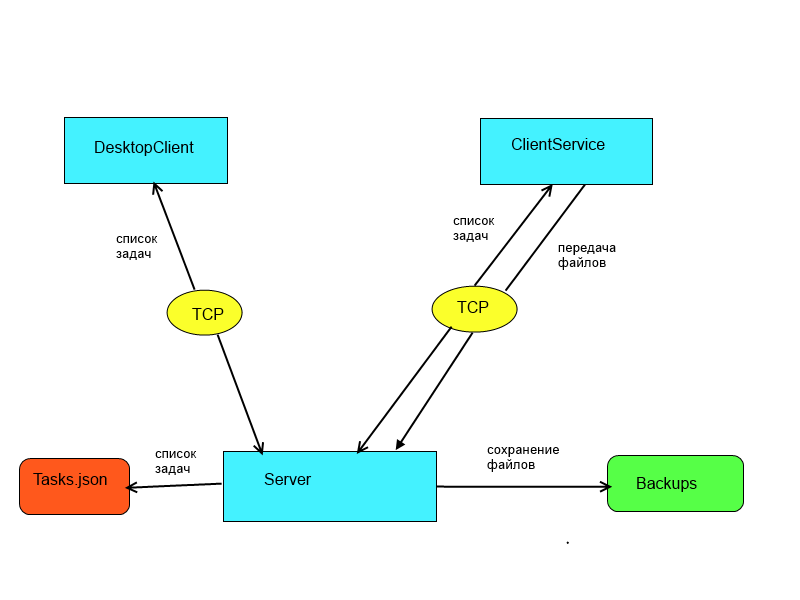
Приложение состоит из трех частей:

1. DesktopClient – приложение с графическим интерфейсом. С его помощью пользователь может добавлять, редактировать, а также мониторить список задач.
2. ClientService – работает в виде службы Windows. Получает список задач и по расписанию выполняет их (отправляет файлы на сервер).
3. Server – получает и сохраняет у себя файлы. Также хранит у себя список задач и передает его клиентам.

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

Сервер сохраняет список задач в файле Tasks.json. Для сохранения списка в формате JSON применяется библиотека Newtonsoft.Json. Запросы и ответы, перед отправкой и после приема, предварительно проходят процесс парсинга JSON.

Архитектура приложения представлена на рисунке 1.

 Рис 1. Архитектура приложения.

Пользовательское приложение включает в себя следующий функционал:

* Использование приложения без регистрации
* Создание задачи для резервирования данных
* Просмотр созданных задач
* Возможность редактирования существующих задач

Пользовательский интерфейс построен с использованием стандартных элементов Windows Forms.

При запуске приложения клиента нужно убедиться в работе службы сервера и правах запуска администратора, так как создается служба для обработки команд пользователя. Если сервер неактивен, то будет выдано сообщение с информацией, предупреждающей об этом, как и о правах. Изображения представлены на рисунках 2 и 3.

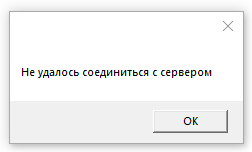


Рис. 1. Окно информации о неудачном подключении к серверу

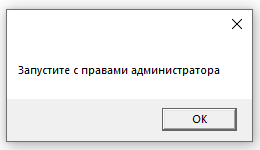


Рис. 3. Окно информации о недостаточных правах для запуска клиента

Чтобы заново попробовать запустить приложение следует получить права администратора и запустить сервер.

Если запуск программы происходит впервые, то нужно установить службу сервера, для этого запускается MainService.bat.

В случае успешного соединения с сервером, пользователь видит интерфейс со списком задач, список получен с сервера, есть функции создания новой задачи, редактирования и удаления. Интерфейс представлен на рисунке 4.

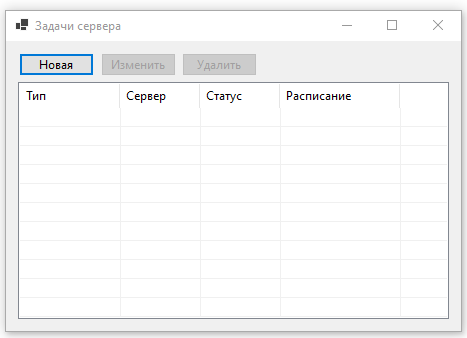


Рис. 4. Интерфейс пользователя, список задач.

Список задач содержит тип задачи, адрес сервера, статус выполнения и расписание, где указан период выполнения.

Для создания задачи необходимо выбрать новую и указать файл, который будет сохраняться, выбрать периодичность выполнения задачи из трех вариантов: ежедневно, еженедельно или ежемесячно. Также нужно указать время выполнения. Интерфейс создания и редактирования задачи изображен на рисунке 5.

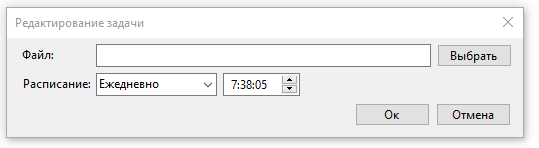


Рис. 5. Интерфейс создания и редактирования задачи.

**Используемые технологии**

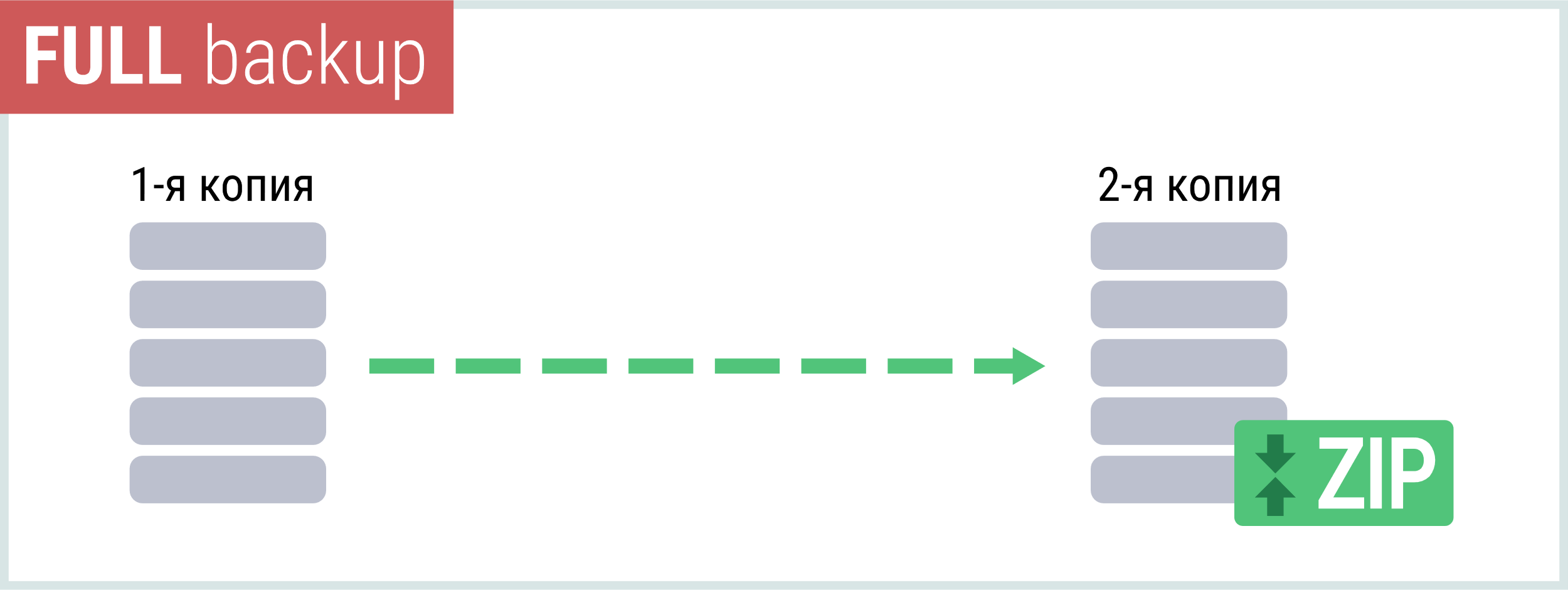
* C# — [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) для платформы [Microsoft .NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) и [.NET Core](https://ru.wikipedia.org/wiki/.NET_Core).
* Windows Forms — интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework.

**Алгоритмы**

(виды резервирования, описать подробно)

Приложение поддерживает следующие схемы резервирования данных:

* + 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 еще **смешанным** или **синтетическим** бэкапом.

- [https://www.sim-networks.com/ru/blog/backup-full-increment-differential]

# Список литературы

1. Работа с библиотекой Newtonsoft.Json на реальном примере // [Электронный ресурс]. URL: <https://habr.com/ru/post/481514/> (дата обращения: 14.05.2022).
2. Работа с библиотекой Network // [Электронный ресурс]. URL: https://github.com/Toemsel/Network (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. 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 string NameFile;

public byte[] Bin;

}

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

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

}

[Serializable]

public class BackupTask

{

public string FileName = "";

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",

\_ => "-"

};

}

public string GetScheduleString()

{

return TypeTimeBackup switch

{

0 => "Ежедневно",

1 => "Еженедельно",

2 => "Ежемесячно",

\_ => "-"

};

}

}

}

namespace SharedData

{

[Serializable]

public class TasksInfo

{

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

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

public Dictionary<string,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<string, BackupTask>);

}

}

public void SaveToFile(string filename)

{

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

}

public static TasksInfo LoadFromFile(string filename)

{

if (File.Exists(filename))

{

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

if (taskInfo != null)

{

return taskInfo;

}

}

return new TasksInfo();

}

}

}

# 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 TasksInfo \_tasks;

public async Task Listen()

{

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

\_server.AllowUDPConnections = false;

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

{

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

conn.SendAsync<SharedResponse>(new SharedRequest()

{

Command = "tasks",

Data = \_tasks.ToArray()

});

};

await \_server.Start();

}

private async void HandlerCommand(SharedRequest packet, Connection connection)

{

var exePath = AppDomain.CurrentDomain.BaseDirectory;

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);

}

}

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 File.WriteAllBytesAsync(Path.Combine(exePath,$@"BackupFiles\{Path.GetFileName(file.NameFile)}"), file.Bin);

result = "OK";

break;

}

default:

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

break;

}

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

}

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);

}

}

}

}

# 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 const string SERVER\_IP = "127.0.0.1";

public const int SERVER\_PORT = 1708;

public static TasksInfo Tasks;

private static TcpConnection? \_connection;

public static Main? MainWindow;

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)

{

\_connection.RegisterStaticPacketHandler<SharedRequest>(RecvHandler);

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;

}

default:

result = "Unknown command";

break;

}

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

}

public static async void SendTasks()

{

if (\_connection == null)

return;

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

{

Command = "tasks",

Data = Tasks.ToArray()

});

}

}

# Приложение 4

Реализация службы клиента.

namespace ClientService;

#pragma warning disable CA1416

public class WinService : ServiceBase

{

private const string SERVICE\_NAME = "ClientService";

private const string SERVER\_IP = "127.0.0.1";

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);

}

}

private async void RecvHandler(SharedRequest packet, Connection connection)

{

string result = "Error";

switch (packet.Command)

{

case "tasks":

{

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

result = "OK";

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 (File.Exists(task.FileName))

{

filesForBackup.Add(task.FileName, await File.ReadAllBytesAsync(task.FileName));

BackupTask updatedTask = task;

updatedTask.Status = SharedData.TaskStatus.Working;

updatedTask.UpdateNextBackupTime();

updatedTask.LastBackupTime = DateTime.Now;

updatedTasks.Add(updatedTask);

}

else

{

BackupTask updatedTask = task;

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

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.FileName] = task;

}

//send tasks list

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

{

Command = "tasks",

Data = \_tasks.ToArray()

});

}

}

await Task.Delay(1000);

}

}

}