

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

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

Кафедра дискретной математики и информационных технологий

**ПРИМЕНЕНИЕ МЕТОДА РАСПРЕДЕЛЕННОГО ХРАНЕНИЯ  
ДАННЫХ ДЛЯ ПОВЫШЕНИЯ ИХ ЗАЩИЩЕННОСТИ**

**БАКАЛАВРСКАЯ РАБОТА**

студента 4 курса 421 группы  
направления 09.03.01 — Информатика и вычислительная техника  
факультета КНиИТ  
Захарова Сергея Алексеевича

Научный руководитель  
старший преподаватель

\_\_\_\_\_

П. О. Дмитриев

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

\_\_\_\_\_

Л. Б. Тяпаев

Саратов 2025

## СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ .....	4
ВВЕДЕНИЕ .....	5
1 Описание предметной области .....	7
1.1 Актуальность поставленной задачи .....	7
1.2 Обзор существующих способов повышения конфиденциальности хранения данных .....	7
1.3 Обзор существующих решений .....	9
2 Теоретические сведения .....	11
2.1 Общие сведения о RAID .....	11
2.2 RAID 5 .....	12
3 Исследование возможности дополнительной защиты данных с использованием разделенного хранения .....	15
3.1 Применение метода распределенного хранения данных для повышения их защищенности .....	16
3.2 Динамический размер распределяемых частей файла .....	19
4 Разработка решения для обработки данных на облачных сервисах ....	24
4.1 Использованные технологии .....	24
4.2 Яндекс Диск .....	27
4.3 Dropbox .....	30
4.4 GoogleDrive .....	33
5 Реализация класса, распределяющего данные среди облачных хранилищ по принципу RAID 5 .....	39
5.1 Загрузка файла с распределением .....	40
5.2 Считывание данных .....	41
6 Разработка пользовательского интерфейса .....	43
7 Тестирование реализованного решения .....	52
ЗАКЛЮЧЕНИЕ .....	55
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	56
Приложение А Исходный код модуля распределения данных RAID 5 ....	58
Приложение Б Исходный код модуля восстановления данных RAID 5 ....	66
Приложение В Исходный код Program .....	84
Приложение Г Исходный код Startup .....	87
Приложение Д Исходный код YandexDiskService .....	88

Приложение Е	Исходный код DropboxService .....	92
Приложение Ж	Исходный код GoogleDriveService .....	96
Приложение З	Исходный код Raid5Service .....	102
Приложение И	Исходный код Raid5Controller .....	161
Приложение К	Исходный код App.vue .....	166

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

**RAID** — это технология объединения нескольких физических дисков в единый логический массив для повышения надежности;

**RAID 5** — 5 уровень RAID;

**JBOD** — это технология объединения дисков без избыточности

**CORS** — это технология, которая с помощью специальных HTTP-заголовков позволяет веб-браузеру получать разрешение на доступ к ресурсам с сервера, находящегося на другом домене, отличном от текущего.

**Фреймворк** — это программная платформа, которая предоставляет базовую структуру для разработки приложений. Он включает в себя готовые компоненты, библиотеки, стандарты и инструменты, упрощающие создание программного обеспечения.

**API** — это набор строго определенных правил, протоколов и инструментов, которые позволяют различным программным компонентам взаимодействовать друг с другом.

## ВВЕДЕНИЕ

Современные облачные сервисы хранения данных предоставляют пользователям доступные и отказоустойчивые решения, однако их повсеместное использование порождает новые вызовы. Основными из них являются ограничения на конфиденциальность, а также недоверие к централизованным поставщикам услуг. Отсутствие прозрачности в механизмах хранения, невозможность гарантировать, что данные доступны только пользователю, а также уязвимость при компрометации единственного провайдера — всё это подталкивает к поиску альтернатив.

На фоне этих ограничений возникла идея исследовать методы распределённого хранения данных, как способ повысить защищённость и конфиденциальность при использовании облачных сервисов. В частности, цель заключалась в построении модели, при которой данные пользователя не хранятся целиком на одном облачном ресурсе, что делает невозможным их полную компрометацию при утечке с одного сервера.

Исходным этапом исследования стала простая модель на базе RAID 1 — с применением шифрования и зеркалирования на три независимых хранилища. Такой подход обеспечивает отказоустойчивость, но остаётся ресурсоёмким и уязвимым: обладая доступом к одному из зеркал, злоумышленник может попытаться расшифровать полный файл.

В работе была рассмотрена более гибкая модель — RAID 5, дополненная собственным подходом к защите данных, заменяющим прямое шифрование. Отказ от классического шифрования обусловлен стремлением уменьшить вычислительную нагрузку и сложность восстановления, а также возможностью заложить конфиденциальность непосредственно в схему распределения.

Исходя из этого, целью работы является обоснование и иллюстрация возможности распределенного хранения, повышающего отказоустойчивость, а также безопасность хранения данных. Для достижения цели были поставлены следующие задачи:

1. Исследовать методы распределенного хранения для повышения конфиденциальности хранимых данных;
2. На основании сравнения выбрать метод для программной реализации;
3. Разработать пакет программ иллюстрирующих метод хранения;
4. Реализовать взаимодействие с облачными хранилищами;

5. Реализовать веб-приложение использующее облачные сервисы в качестве хранилищ;
6. Провести тестирование и анализ результатов.

## **1 Описание предметной области**

### **1.1 Актуальность поставленной задачи**

В последние годы наблюдается устойчивая тенденция утраты доверия пользователей и компаний к централизованным облачным хранилищам. Несмотря на их удобство и доступность, всё чаще фиксируются случаи сбоев в работе сервисов, утечек данных и компрометации информации. Одним из наиболее заметных примеров является массовый отказ облачных сервисов Яндекс в ноябре 2024 года, приведший к недоступности хранилищ и потере данных для ряда пользователей [1].

Утечки персональных данных, происходящие даже в крупнейших облачных платформах, свидетельствуют о недостаточном уровне информационной безопасности и уязвимости перед внешними и внутренними угрозами. Кроме того, централизованное хранение данных у одного поставщика создаёт единую точку отказа и увеличивает риски в случае технических или организационных сбоев.

Особенно остро проблема встаёт в условиях, когда пользователь не может контролировать, где и каким образом хранятся его файлы. Это делает критически важным поиск новых подходов к хранению информации, способных снизить риски утраты данных и несанкционированного доступа.

В этой связи актуальными становятся методы распределённого хранения данных, позволяющие исключить полную зависимость от одного сервиса и повысить уровень конфиденциальности за счёт распределения информации между несколькими независимыми сервисами.

### **1.2 Обзор существующих способов повышения конфиденциальности хранения данных**

В условиях растущих угроз информационной безопасности задача обеспечения конфиденциальности данных становится ключевой при проектировании систем хранения. Существуют различные подходы, направленные на снижение риска несанкционированного доступа, каждый из которых обладает своими преимуществами и ограничениями:

1. Разделение данных — один из базовых методов повышения конфиденциальности — физическое или логическое разделение информации. Данные разбиваются на части, которые хранятся на различных носителях

или в разных хранилищах. Такой подход препятствует восстановлению полной информации при компрометации одной из частей. Примером является использование RAID-массивов или схемы тайного разделения.

2. Шифрование — наиболее распространённый способ защиты — криптографическое преобразование данных. Применение симметричного или асимметричного шифрования обеспечивает невозможность прочтения содержимого без ключа. Однако эффективность метода напрямую зависит от стойкости алгоритма, безопасности хранения ключей и корректной реализации. Кроме того, в случае утраты ключа восстановление данных становится невозможным.
3. Стеганография — метод скрытия информации внутри других данных. Стеганография обеспечивает не только конфиденциальность, но и маскировку самого факта передачи или хранения важной информации. Однако объём встраиваемых данных ограничен, и метод чаще применяется в сочетании с другими средствами.
4. Отделение метаданных — метаданные — сопутствующая информация о файле, часто представляют не меньшую ценность, чем содержимое. Один из способов повышения конфиденциальности — хранение метаданных отдельно от основного содержимого или их полное удаление. Это затрудняет анализ структуры и происхождения файлов при возможной утечке.
5. Комбинированные методы — на практике для повышения стойкости к атакам часто применяются комбинации вышеперечисленных подходов. Например, данные могут быть предварительно зашифрованы, затем разделены на части, каждая из которых скрыта с помощью стеганографических методов и отправлена в разные хранилища. Такой многоуровневый подход повышает общий уровень защищённости, но требует дополнительной вычислительной мощности и усложнённой логики восстановления.
- 6.

Таким образом, выбор подхода зависит от поставленных целей: критичности защищаемой информации, допустимого уровня избыточности, вычислительных ресурсов и требуемой отказоустойчивости. В данной работе акцент делается на методах разделения и распределённого хранения с эле-



ментами повышения конфиденциальности без использования классического шифрования.

### 1.3 Обзор существующих решений

Существует множество подходов к организации хранения информации, каждый из которых ориентирован на определённые цели: надёжность, производительность, масштабируемость или конфиденциальность. Рассмотрим основные модели хранения, применяемые как в локальных, так и в распределённых системах.

1. Локальное хранение на диске с файловой системой — наиболее простой и традиционный способ, представляющий собой использование физического накопителя с файловой системой. Такой подход обеспечивает быструю работу и полное управление над данными, но не предполагает встроенных механизмов отказоустойчивости или распределения. Конфиденциальность и сохранность в этом случае полностью зависят от пользователя и среды.
2. RAID-массивы — технология **RAID** позволяет организовать логические массивы из нескольких физических дисков. Различные уровни **RAID** обеспечивают разные свойства: от увеличения производительности до обеспечения отказоустойчивости. Эти решения могут использоваться как на аппаратном уровне, так и программно.
3. Распределённое сетевое хранилище — в данной модели данные хранятся на нескольких узлах, связанных сетью. Примеры включают технологии типа Ceph, GlusterFS, HDFS. Эти системы обеспечивают масштабируемость и отказоустойчивость, автоматически реплицируя и распределяя данные. Часто они применяются в корпоративных и облачных инфраструктурах. Однако их настройка и обслуживание требуют значительных ресурсов и квалификации.
4. Объектные хранилища — в отличие от блочной или файловой модели, объектное хранилище оперирует единицами хранения — объектами, включающими данные, метаданные и уникальный идентификатор. Примеры: Amazon S3, MinIO. Такой подход упрощает масштабирование, позволяет работать с огромными объёмами данных и обеспечивает гибкое управление доступом. Однако он менее пригоден для операций с высокой частотой обновлений.

5. Использование облаков как хранилищ — облачные сервисы предоставляют пользователю интерфейс для хранения и извлечения данных, обычно через HTTP API, реализующий операции CRUD. Пользователь работает не с физическим диском, а с абстрактной сущностью — файлом, объектом, записью. Примеры — Google Drive, Dropbox, Яндекс Диск. Эти сервисы удобны в использовании и обеспечивают высокую доступность, но ограничены по гибкости конфигурации.

С точки зрения защиты данных, использование облаков требует дополнительных мер: данные проходят через открытые каналы, хранятся на инфраструктуре провайдера и обрабатываются по его правилам. Это порождает риски утечки и потери контроля. В таких условиях особенно актуальны подходы, сочетающие распределение, частичную независимость и отказоустойчивость на уровне прикладной логики — например, программная реализация RAID с использованием нескольких облачных провайдеров.

## 2 Теоретические сведения

### 2.1 Общие сведения о RAID

Технология RAID появилась в эпоху, когда жесткие диски были значительно дороже и менее надежными, чем сейчас. Изначально аббревиатура RAID имела расшифровку "Избыточный массив недорогих дисков" (Redundant Array of Inexpensive Disks), но со временем ее значение изменилось на "Избыточный массив независимых дисков" (Redundant Array of Independent Disks). Системы, поддерживающие RAID, часто называют RAID-массивами [2].

Основные задачи RAID:

1. Повышение производительности за счет чередования (striping) – данные распределяются по нескольким дискам, что снижает нагрузку на каждый из них.
2. Увеличение отказоустойчивости благодаря избыточности (redundancy) – даже при отказе одного диска система продолжает работу за счет резервных данных.

Отдельный жесткий диск имеет ограниченную скорость и срок службы, но объединение нескольких дисков в RAID-массив позволяет значительно улучшить надежность и производительность системы в целом.

RAID-контроллер объединяет физические диски в виртуальный жесткий диск, который сервер воспринимает как единое устройство. При этом реальное распределение данных между дисками остается скрытым и видно только администратору.

Существуют разные уровни RAID, определяющие способы распределения данных. Почти все они предусматривают избыточное хранение информации, что позволяет восстановить данные при отказе диска. Восстановление данных происходит параллельно с работой сервера, что может временно снизить производительность.

Один из распространенных подходов — использование зеркалирования RAID 1, при котором зашифрованный файл дублируется на три независимых диска. Такое решение обладает определенными достоинствами: простота реализации и высокая надёжность при отказе одного из носителей. Однако его недостатки включают в себя низкую эффективность использования пространства и вероятность компрометации данных при получении доступа к одному из хранилищ [3].

Альтернативным методом является применение RAID 5 — распределения данных с расчётом контрольных сумм, обеспечивающим отказоустойчивость и более эффективное использование ресурсов. Основная идея — разбить исходный файл на части и распределить их между хранилищами, при этом контрольная сумма каждой пары блоков позволяет восстановить один из них в случае утраты.

Выбор метода RAID 5 обусловлен его преимуществами перед RAID 1: RAID 5 требует меньшего объёма хранилищ за счёт хранения только одного блока четности на группу данных, при этом обеспечивая возможность восстановления утерянной части. Кроме того, благодаря данному подходу файл распределяется частями среди носителей данных, что повышает степень их защищённости за счёт того, что невозможно получить полную информацию, имея только одно хранилище.

## 2.2 RAID 5

RAID 5 уровня является отказоустойчивым массивом хранения данных на нескольких хранилищах данных, в данном случае на облачных сервисах, представляющим собой распределение данных на определенное количество чередований с вычислением для них контрольных сумм. Контрольная сумма представляет собой  $A \oplus B = Parity$ , где  $A$  и  $B$  представляют собой два массива байтов разделённого файла [4]. Таким образом, достигается возможность восстановить данные при потере  $A$  или  $B$  ввиду следующих равенств:

- $A \oplus Parity = B$
- $Parity \oplus B = A$

Как правило, файл разделяется по одному из двух правил, в зависимости от размера данных, с которыми предстоит работать:

1. Весь файл целиком делится на 6 равных по размеру массивов байтов, данные массивы разделяются на 3 пары и для каждой пары вычисляется чётность.
2. Задается максимальный размер массива и данные распределяются до тех пор, пока весь файл не будет разделен и их количество не будет кратно 6.

Например, имеется файл, который требуется распределить вторым методом, для этого файл поочередно разделяется  $n$  раз на части в формате массивов байт  $A_i, B_i, C_i, D_i, E_i, F_i$ . Из данных массивов формируются пары

$A_i, B_i, C_i, D_i, E_i, F_i$ , для каждой из них формируются массивы чётности:

- $A_i \oplus B_i = Parity_{i1}$
- $C_i \oplus D_i = Parity_{i2}$
- $E_i \oplus F_i = Parity_{i3}$

Затем данные распределяются между дисками так, чтобы на диске сохранилась ровно одна чётность данного чередования, а на двух других соответствующие данные пары. Процесс повторяется до тех пор, пока файл не будет исчерпан.

На рисунке 1 продемонстрирована модель распределения данных с применением трёх дисков.

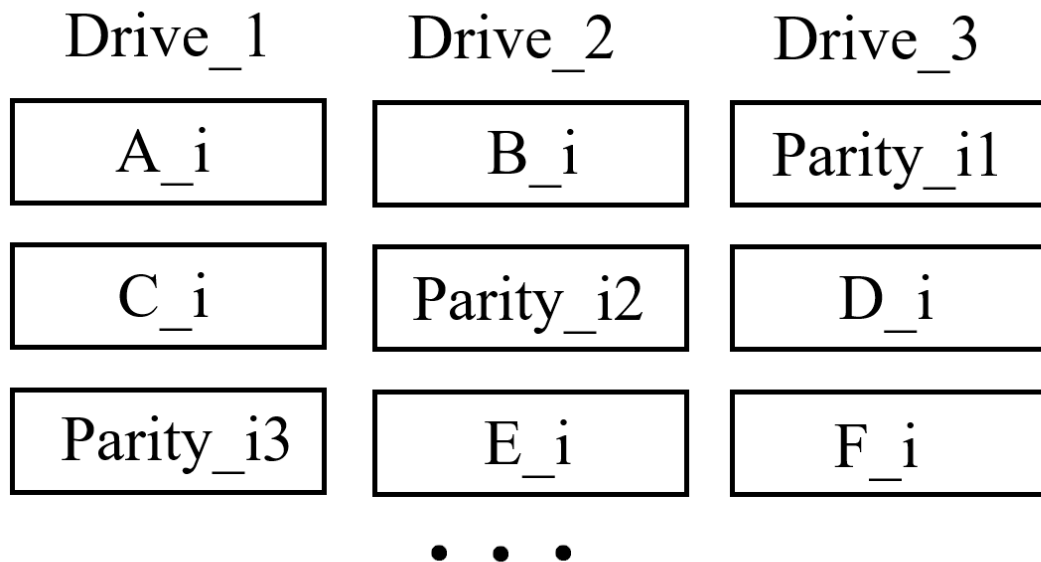


Рисунок 1 – Модель распределения данных

Таким образом, при выводе из строя одного из дисков, например, диска 2, все утраченные данные можно восстановить до первоначальных массивов с байтами следующим образом:

- $A_i \oplus Parity_{i1} = B_i$
- $Parity_{i3} \oplus F_i = E_i$

Данный метод предполагает, что будет использоваться как минимум 3 диска, при этом имея возможность масштабируемости на необходимое большее количество. Потери в общем объеме хранилища будут составлять объем

одного диска, при этом данные подлежат восстановлению при потере одного хранилища.

### 3 Исследование возможности дополнительной защиты данных с использованием разделенного хранения

Разработка модели RAID 5 программным путем содержит в себе ряд нюансов, отличных от её реализации на физических дисках, которые сводятся к необходимости подготовить собственную модель файловой системы. В данном решении файл будет представляться в виде нескольких массивов байт для организации RAID, они распределяются вместе с массивами четности среди трех файлов. В двух файлах хранятся 6 наборов байт, принадлежащих первоначальному файлу, данные байты хранят в себе полную информацию о файле, за счет их сбора в первоначальном порядке файл восстанавливается до первоначального вида. В третьем файле хранятся массивы четности, за счет которых восстанавливаются потерянные данные, хранящиеся в остальных файлах. Модель распределения данных представлена на рисунке 2.

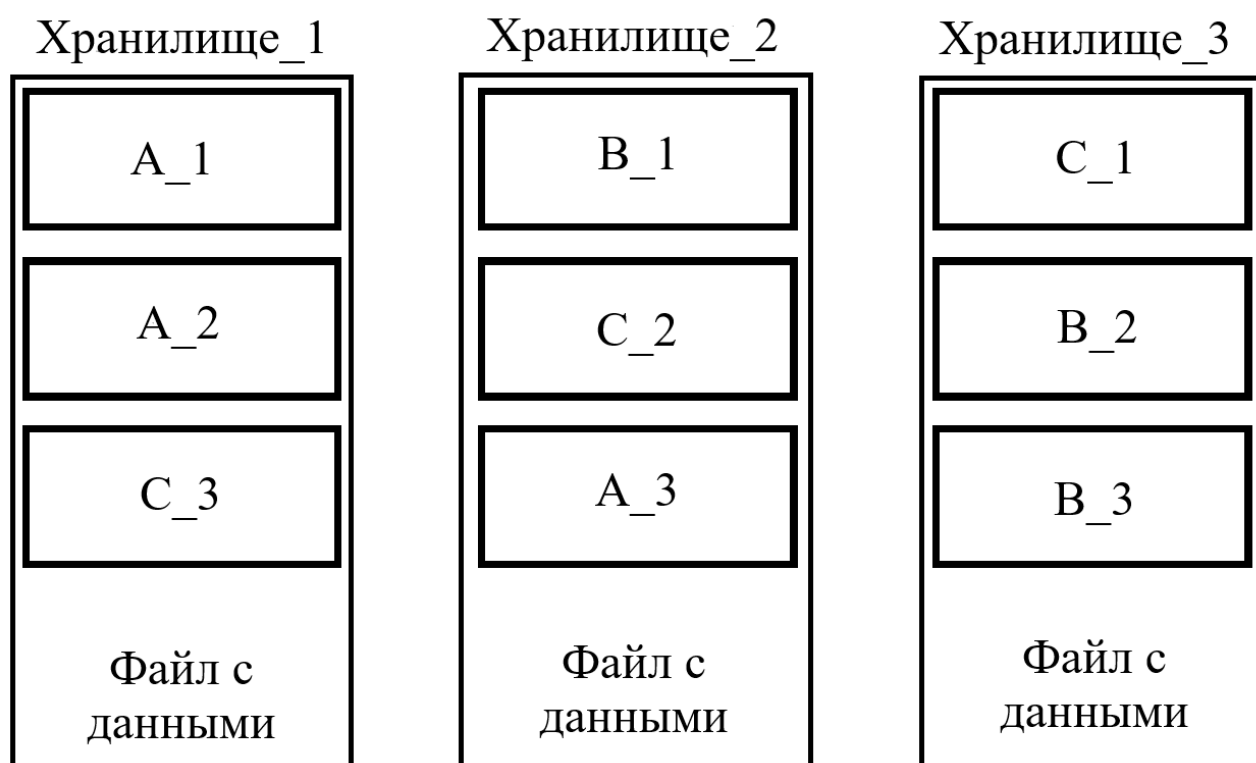


Рисунок 2 – Модель распределения данных среди файлов

В данной модели обозначения  $A_1, B_1, A_2, B_2, A_3, B_3$  представляют собой наборы байтов первоначального файла,  $C_1, C_2, C_3$  наборы массивов четности. Эти данные распределяются среди файлов для дальнейшего применения.

Восстановление данных при потере одного из файлов производится за счет применения операции XOR. Применение данной операции соотносится с

описанной моделью следующим образом:

- $A_1 \oplus C_1 = B_1$
- $B_1 \oplus C_1 = A_1$
- $A_2 \oplus C_2 = B_2$
- $B_2 \oplus C_2 = A_2$
- $A_3 \oplus C_3 = B_3$
- $B_3 \oplus C_3 = A_3$

Таким образом, при потере одной тройки данных присутствует возможность восстановить её за счёт пары других. Так же данная организация повышает безопасность хранения данных за счёт того, что даже если злоумышленник попытается восстановить данные, он не сможет получить гарантированно верный результат, если не знает модель, по которой производится вычисление четности.

### 3.1 Применение метода распределенного хранения данных для повышения их защищенности

Первым этапом разработки сервиса является подготовка модели для её последующего внедрения в систему. Для достижения данной цели было разработано два консольных приложения на языке **C#**, реализующее представление файла в виде массива байт, разделяя их на части для распределения по дискам.

Первое приложение выполняет функцию разделения данных на составляющие, расчета массивов четности а также подготовку трех файлов в формате, готовом для извлечения и взаимодействия с данными. В качестве входных данных приложение принимает путь к файлу, находящемуся на устройстве, после завершения работы сохраняются три файла, сформированные согласно описанной модели.

Одним из вызовов в вопросе программной реализации **RAID 5** становится тот факт, что в реализации на аппаратном уровне, файловые системы хранят данные блочно, а также они хранят блоки с дополнительными данными, отдельно от данных файла. В следствии чего поднимается вопрос о способах определения окончания файла в ситуации, когда в последней части данных остается меньше байт, чем в остальных. Из-за того, что файл представляется в виде шести массивов одинакового размера, хранящих его байты, в случае, если изначальный размер файла не кратен шести, будет происходить потеря



данных, поэтому при определении размера каждой части производится деление с округлением вверх, а лишние байты, не принадлежащие изначальному файлу, записываются в виде нулей. За счет данного решения сохраняется возможность применения операции XOR для расчёт массивов четности. Пример данной реализации предоставлен на листинге ниже:

```
int reduce = 0;
int size = (int)Math.Ceiling(fileContent.Length / 6.0);
if(fileContent.Length % 6 != 0)
{
    reduce = 6 - fileContent.Length % 6.0;
}
```

Данное значение позволяет определить, сколько лишних байт было добавлено при расширении частей. Для его хранения и дальнейшего применения после завершения формирования файла данное значение помещается в конец сформированного файла в виде набора байтов:

```
for (int i = 0; i < byteArray.Length; i++)
{
    data_1_final[j] = byteArray[i];
    // Данная переменная является счетчиком
    // свидетельствующем о завершении основной
    // части набора данных
    j++;
}
```

Формирование массивов четности производится за счет отдельного метода. Данный метод принимает в качестве аргумента два массива данных, производит в цикле логическую операцию XOR для каждой пары элементов из предоставленных массивов. По завершении работы данный метод возвращает готовый для применения массив четности:

```
byte[] SolveParity(byte[] data_1, byte[] data_2)
{
    byte[] parity = new byte[data_1.Length];
```

```

for (int i = 0; i < parity.Length; i++)
    parity[i] = (byte)(data_1[i] ^ data_2[i]);
return parity;
}

```

В процессе работы приложения формируются три основных массива с байтами. Данные массивы заполняются данными в соответствии с моделью, и по завершению работы приложения из них формируются подготовленные файлы. Результат работы приложения в формате ответа в консоли и полученных файлов представлен на рисунке 3.

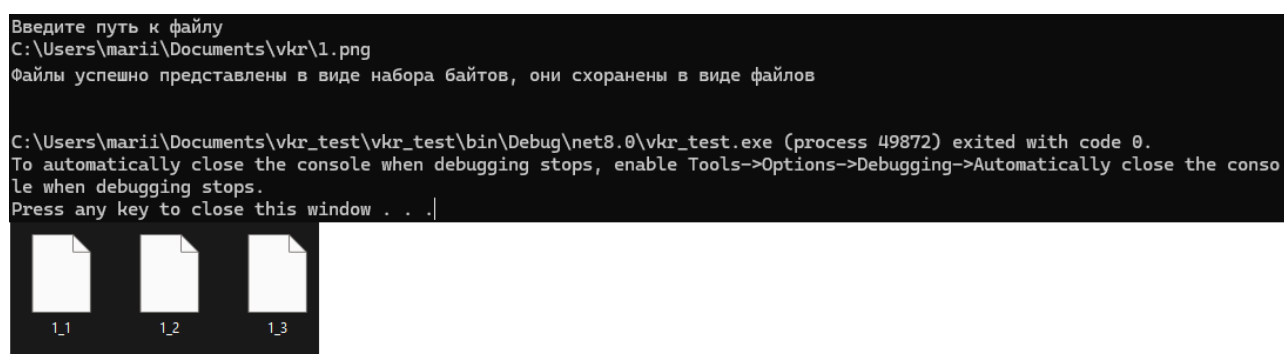


Рисунок 3 – Результат работы первого приложения на файле 1.png

Второе приложение выполняет функцию восстановления данных из полученных файлов. Данное приложение запрашивает у пользователя распределённые файлы, а также ключ, по завершении работы собирает изначальный файл.

После получения необходимых файлов производится определение необходимых действий, требуется ли восстанавливать файл при помощи массива четности либо все данные присутствуют, и файл может быть собран при помощи первоначальных байт. Полученные файлы представляются в виде массива байт и распределяются по частям в порядке, по которому они были представлены в изначальном файле. В случае необходимости восстановить утраченные блоки производится операция *XOR* с имеющимися байтами. При помощи сохранённого значения, отображающего количество лишних байт, размер массива с последним блоком данных уменьшается. При помощи полученных данных формируется и сохраняется изначальный файл. Пример работы приложения в формате сообщения от программы в консоли и полученного файла представлен на рисунке 4.

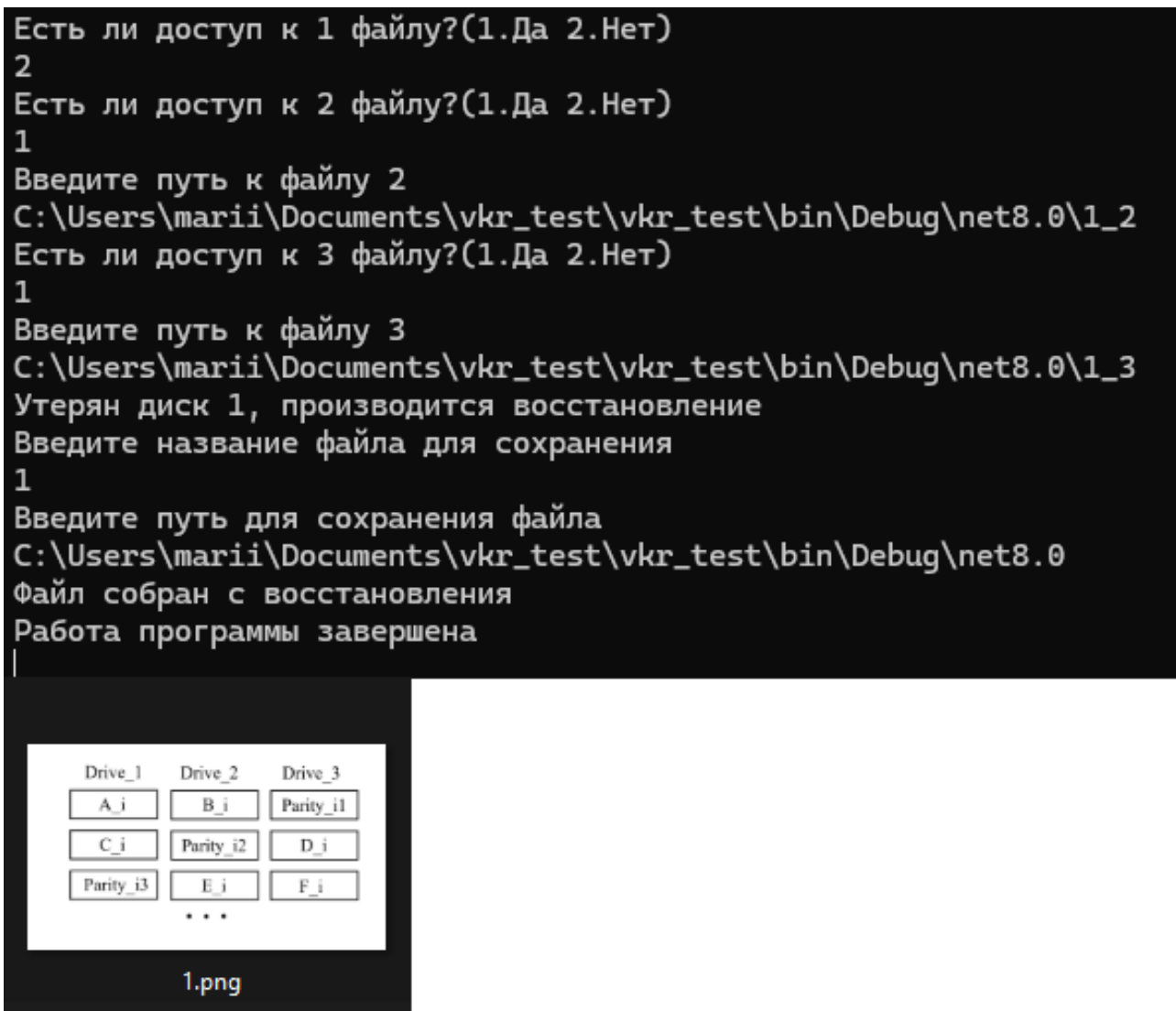


Рисунок 4 – Результат работы второго приложения на подготовленных файлах

### 3.2 Динамический размер распределяемых частей файла

После приготовления общей модели, для повышения степени защищенности хранимых данных ее необходимо модифицировать. Для этого предлагается сделать размер каждой тройки данных задаваемым, а сами параметры хранить в формате ключа, значение которого можно будет задавать индивидуально. При таком подходе, при компрометации данных, сложность перебора для ее извлечения не имея ключа возрастает в зависимости от количества предоставленных вариантов. За значение, которое может быть изменено для динамического распределения применяется процентное соотношение от размера всего файла.

Для достижения поставленной задачи были модифицированы описанные ранее решения. Данное приложение запрашивает у пользователя ключ,

при помощи которого производится сборка файла. В данном ключе содержится конфигурация, по которой производится разделение данных файла на составляющие с отличающимся размером. Установленный размер применяется ко всем частям тройки данных изначального файла. В прошлой реализации все части имели фиксированный размер, например: части  $A_1$  и  $B_1$  всегда были равны  $\frac{1}{6}$  от изначального размера файла, однако теперь пользователю предоставляется возможность установить фиксированный размер, например: первая часть будет содержать в себе 20% от объема байт файла. Остальные части также будут отличаться по размеру, второй набор данных также задается при помощи ключа, а третий дополняется всеми оставшимися данными файла.

Программная реализация первого приложения отличается от предыдущей тем, что теперь байты файла распределяются по закону, установленному пользователем. Размер для последней тройки данных рассчитывается при помощи вычисления недостающих до полного объема изначального файла байтов:

```
if (part_size_a + part_size_b < fileContent.Length)
{
    part_size_c += (int)Math.Ceiling((fileContent.Length
        - (part_size_a + part_size_b) * 2) / 2.0);
}
```

Однако, при таком подходе возникает проблема: при восстановлении файла необходимо рассчитать размер частей. При вычислении процентного соотношения при помощи распределенных частей, не зная размер изначального файла происходит потеря данных при округлении после деления. Для решения данной проблемы было решено передавать вместе с частями файлов дополнительные 4 байта, хранящиеся на их окончании. Такой подход не влияет на защищенность файла за счет того, что знание только размера изначального файла не способствует расшифровыванию данных, однако зная ключ при таком подходе можно гарантированно точно рассчитать размер частей. Пример метода хранения дополнительных данных в части файла продемонстрирован на листинге ниже:

```
byte[] originalSize = BitConverter.GetBytes(fileContent.Length);
```

```
// j является счетчиком, указывающим на то,
// что все основные данные уже заполнены
for (int i = 0; i < 4; i++)
{
    data_1_final[j] = originalSize[i];
    j++;
}
```

Результат работы приложения представлен на рисунке 5.



Рисунок 5 – Результат работы первого приложения с примененным ключем

Реализация модификации для второго приложения отличается тем, что теперь необходимо предварительно извлечь из части файла массив байт, который является размером изначального файла. Для этого организуется цикл по последним байтам распределенной части файла и считывается информация:

```
byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_2_final.Length - 4;
    i < data_2_final.Length; i++)
{
    originalSizeBytes[counter] = data_2_final[i];
    counter++;
}
int originalSize = BitConverter.ToInt32(originalSizeBytes, 0);
```

Таким образом производится точный размер частей, совпадающий со значением полученным в первом приложении. За счет этого при помощи ключа производится распознавание каждой части для дальнейшей работы с данными. Остальной функционал второй программы остается аналогичным с предыдущей версией. Результат работы приложения продемонстрирован на рисунке 6.

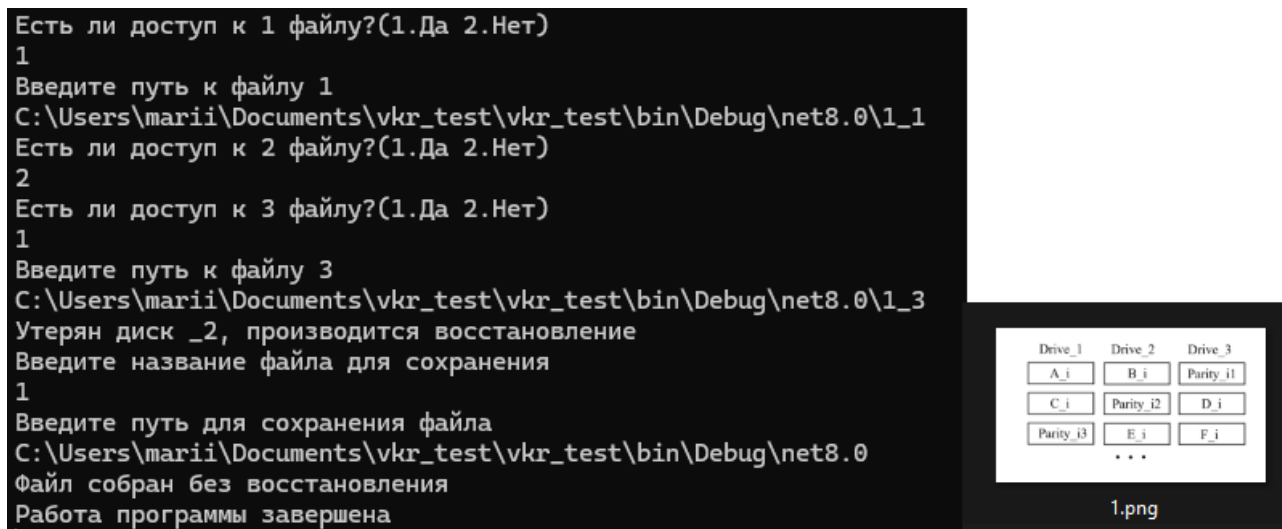


Рисунок 6 – Результат работы второго приложения с заданным размером частей

Полный код представленных модулей предоставлен в Приложениях А, Б.

Также в целях продолжения увеличения степени защищенности данных было решено подготовить несколько способов распределения данных среди хранилищ. Например, при описанной модели распределения данных, файл с массивом четности для частей 5 и 6 помещается на первое хранилище, если изменить его расположение на третье хранилище, а данные с третьего поместить на его место, то модель изменится увеличивая необходимое для перебора количество вариантов. Таким образом злоумышленнику становится еще тяжелее подобрать необходимую комбинацию, не зная параметры ключа.

В результате модификации модели распределения данных был реализован механизм динамического задания размеров частей файла, что значительно повысило гибкость и безопасность хранения. Пользователь получил возможность настраивать параметры распределения через ключ, что усложнило потенциальную компрометацию данных. Тестирование подтвердило, что даже при утрате одного из хранилищ файлы успешно восстанавливаются без

потери информации. Эти наработки создали основу для следующего этапа — интеграции распределенного хранения с облачными сервисами.

## 4 Разработка решения для обработки данных на облачных сервисах

### 4.1 Используемые технологии

Переходя от теоретических исследований к практической реализации, в данной главе рассматривается разработка решения для взаимодействия с облачными сервисами. Для разработки серверной части проекта был выбран разработанный компанией Microsoft фреймворк **ASP.Net Core**. Данный фреймворк представляет технологию для создания веб-приложений на платформе **.NET**. Предоставленная технология обеспечивает средства для разработки серверной части веб-приложения. Также он предоставляет инструментов для подключения дополнительных библиотек, обеспечивает подключение для взаимодействия с веб-интерфейсами. В качестве языка программирования для разработки приложений при помощи **ASP.NET Core** применяется компилируемый язык программирования **C#**.

При распределении файлов по принципу **RAID 5** требуется вычисление контрольных сумм для обеспечения отказоустойчивости. Это интенсивная операция, зависящая от вычислений на процессоре, где компилируемые языки имеют значительное преимущество перед интерпретируемыми. При вычислении операции  $A \oplus B$  компилируемый язык преобразует её в одну инструкцию для процессора, в свою очередь, интерпретируемый сначала произведёт проверку типов данных переменных, затем определит перегруженный оператор, отвечающий за данную операцию, и в конце начнёт выполнение соответствующего метода.

При проектировании структуры проекта был применен принцип трёхуровневой архитектуры. Это классический подход к проектированию программных систем, в котором приложение делится на три логических слоя:

1. Уровень представления — Этот слой отвечает за взаимодействие с пользователем и отображение данных. Он обеспечивает коммуникацию между пользователем и системой. В данном случае за него отвечает пользовательский интерфейс. Он выполняет следующие функции:
  - Загрузка файла на облачные сервисы с распределением по принципу **RAID 5**
  - Скачивание файла с облачных сервисов
  - Возможность восстановить потерянные данные, в случае вывода



из строя одного из дисков

2. Уровень логики приложения — Этот слой отвечает за бизнес-логику и правила работы системы. Он обрабатывает запросы, поступающие с уровня представления, и выполняет всю логику, необходимую для обработки данных, взаимодействия с базами данных и других операций.

Он реализует следующие функции:

- Распределение данных, хранящихся в выбранном файле в виде массива байтов, готового для отправления.
- Подготовка массива чётности, реализующего восстановление данных, в случае отказа в доступе к одному из облачных хранилищ.
- Сборка массивов байтов, распределенных по принципу RAID 5 в виде файла

3. Слой данных — Этот слой отвечает за управление данными и их хранение. Он взаимодействует с базами данных, файлами, облачными хранилищами или внешними сервисами данных. Слой данных получает запросы от бизнес-логики для чтения или изменения данных. Данный слой реализует взаимодействие с представленными облачными хранилищами:

- Яндекс Диск
- Dropbox
- Google Drive

Данный подход к проектированию предоставляет преимущества для реализации распределённого облачного хранилища. Чёткое разделение на уровни позволяет независимо модифицировать интерфейс пользователя, бизнес-логику обработки файлов и механизмы взаимодействия с облачными сервисами. Кроме того, трёхуровневая структура упрощает тестирование каждого компонента системы по отдельности и способствует поддержанию чистоты кода.

Основной класс, отвечающий за сборку проекта, представляет собой точку входа **ASP.NET Core** приложения и содержит конфигурацию веб-сервера. Установка конфигурации производится путём передачи параметров в экземпляр класса, отвечающего за инициализацию. При сборке проекта были добавлены два инструмента:

- Swagger

**Swagger** представляет собой набор инструментов для тестирования и визуализации запросов [5]. Основная часть данного инструмента при работе с API — предоставляет возможности интерактивного тестирования разработанных сервисов. Он обеспечивает доступ к проведению запросов напрямую из браузера, без использования вручную разработанного интерфейса и показывает разработчику все необходимые поля с типами данных.

Также данный инструментарий предоставляет возможность создания описаний работы API: информации о ресурсах, параметрах запросов, возвращаемых данных и сведениях о результатах запросов. Чтобы автоматизировать это описание, сделать его структурированным и прозрачным.

Следующим подключенным параметром является Cross-Origin Resource Sharing (CORS) — это технология, которая с помощью специальных HTTP-заголовков позволяет веб-браузеру получать разрешение на доступ к ресурсам с сервера, находящегося на другом домене, отличном от текущего. Если веб-страница пытается запросить данные с другого источника (cross-origin HTTP-запрос), это означает, что домен, протокол или порт запрашиваемого ресурса отличаются от тех, что указаны в исходном документе [6].

В целях безопасности браузеры накладывают ограничения на кросс-доменные запросы, выполняемые через **JavaScript**, следуя политике одинакового источника. Это означает, что веб-приложение может запрашивать ресурсы только с того домена, с которого оно было загружено, если сервер явно не разрешит доступ через **CORS**. Данная технология обеспечивает безопасный обмен данными между браузерами и серверами при кросс-доменных запросах.

**ASP.NET Core** предоставляет возможность настройки разрешений для данной технологии. Для настройки политики доступа необходимо передать в метод адрес, с которого будут приходить запросы, а также указать название для текущей конфигурации [7]. Пример использования продемонстрирован на листинге ниже:

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowVueApp",
        policy => policy.WithOrigins("http://localhost:5173")
```

```
.AllowAnyHeader()  
.AllowAnyMethod());  
});
```

## 4.2 Яндекс Диск

Яндекс Диск является одним из выбранных облачных хранилищ для разработки. Он предоставляет набор инструментов для разработчиков, позволяющий получать данные пользователя с его согласия, а также взаимодействовать с его личным хранилищем. Для применения данного диска в разработке **RAID** можно выделить следующие этапы:

1. Создание приложения в сервисе Яндекс ID;
2. Получение токена пользователя;
3. Разработка методов для взаимодействия с хранилищем пользователя.

При создании приложения необходимо указать его название, логотип и выбрать тип устройств, на которых оно будет предположительно исполняться. Далее следует указать адрес, на который пользователь будет перенаправлен для авторизации. Выбор данного адреса зависит от метода получения токена, для применения в веб-сервисе, как правило, используются технологии мгновенной авторизации. Для этого адресом используется ссылка, по которой создаётся вспомогательная страница для приёма токена [8].

Стандартное приложение, созданное в Яндекс ID предоставляет возможность запрашивать следующие данные аккаунта при регистрации:

- Логин, имя и фамилия, пол.
- Портрет пользователя.
- Адрес электронной почты.
- Номер телефона.
- Дата рождения.

Однако, данной информации недостаточно для достижения поставленной цели, в следствие чего к сервису необходимо подключить **REST API**, значительно расширяющий набор предоставляемых данных. Для реализации данного приложения были выбраны пункты чтения всего диска и записи в любое место диска.

В завершение работы с данным сервисом необходимо получить **OAuth** токен пользователя, необходимый в **http** запросах с сервера. Существует несколько методов, однако в данной работе будет рассмотрен метод полу-

чения отладочного токена [9]. Для этого необходимо выполнить следующий ряд действий:

1. Пользователь должен предоставить приложению доступ по указанному адресу перенаправления.
2. Изменить указанный адрес, добавив в конце `client_id=<идентификатор приложения>`. Данное значение можно узнать на панели управления сервисом.
3. Затем откроется страница с подтверждением доступа и будет выдан текущий токен.

Для отправления запросов использовался класс `HttpClient` — это класс в .NET, предназначенный для отправки HTTP-запросов и получения HTTP-ответов от ресурсов, идентифицируемых URI. Основные его функции:

- Отправка HTTP-запросов (GET, POST, PUT, DELETE)
- Работа с заголовками запросов и ответов
- Управление временем ожидания и политиками повторов
- Поддержка асинхронных операций
- Возможность обработки различных форматов данных (JSON, XML и др.)

Выгрузка файла осуществляется при помощи запроса с переданными полями в виде: OAUTH токен, путь к файлу на облачном сервисе и массив байт файла для отправки. Изначально запрашивается ссылка для загрузки файла. Полученный ответ хранится в JSON формате, из-за чего его требуется десериализовать. Далее формируется массив байтов с файлом и вместе с полученной ранее ссылкой формируется запрос для выгрузки. Пример данного кода приведён в листинге ниже:

```
public async Task<string> UploadFile
    (string filePath, string localFilePath)
{
    var uploadUrlResponse = await _httpClient
        .GetAsync($"{_apiUrl}resources/upload?path={filePath}
            &overwrite=true&fields=name,_embedded.items.path");
    // URL для загрузки файла
    uploadUrlResponse.EnsureSuccessStatusCode(); // Статус запроса
```

```

// Десериализация ответа с адресом для загрузки
var uploadUrl = JsonConvert
    .DeserializeObject<YandexDiskUploadResponse>(await
        uploadUrlResponse.Content.ReadAsStringAsyncAsync()).Href;

byte[] fileContent = File.ReadAllBytes(localFilePath);

var content = new ByteArrayContent(fileContent);
// Загрузка файла по полученному URL
var uploadResponse = await _httpClient
    .PutAsync(uploadUrl, content);
uploadResponse.EnsureSuccessStatusCode();
// Проверка, что загрузка прошла успешно

return await uploadResponse.Content.ReadAsStringAsyncAsync();
// Результат загрузки
}

```

Следующей подзадачей является скачивание файла с облачного сервиса. В виде параметров для запроса передаются OAuth токен, путь к файлу, расположенному в хранилище. В данном случае файл будет храниться в корневой папке хранилища, в связи с чем путём будет являться название файла. Формируется запрос с путём к файлу на облачном сервисе. Полученная ссылка десериализуется из формата JSON и применяется в запросе для загрузки файла. Полученный файл конвертируется из массива байтов в изначальный вид. Пример продемонстрирован на листинге ниже:

```

public async Task<byte[]> DownloadFile(string filePath)
{
    var downloadUrlResponse = await _httpClient
        .GetAsync($"{{_apiUrl}}resources/download?path={{filePath}}");
    // URL для загрузки файла
    downloadUrlResponse.EnsureSuccessStatusCode();
    // Статус запроса
}

```

```

var downloadUrl = JsonConvert
    .DeserializeObject<YandexDiskDownloadResponse>
    (await downloadUrlResponse.Content
        .ReadAsStringAsyncAsync()).Href;
// Десериализация ответа с адресом для скачивания
var downloadResponse = await _httpClient
    .GetAsync(downloadUrl);
// Скачивание файла по полученному URL
downloadResponse.EnsureSuccessStatusCode();
// Проверка результата запроса

return await downloadResponse.Content
    .ReadAsByteArrayAsync(); // Содержимое файла
}

```

### 4.3 Dropbox

Dropbox является облачным сервисом, позволяющим получать данные пользователя и взаимодействовать с его хранилищем данных [10]. Для применения данного сервиса были поставлены следующие цели:

1. Регистрация сервиса для облака;
2. Получение токена пользователя;
3. Подготовка класса для взаимодействия с диском пользователя.

Для создания приложения необходимо перейти в консоль разработчика. Выбрав соответствующую опцию, будут запрошены следующие параметры:

- Тип API
- Уровень доступа к диску
- Название регистрируемого в консоли приложения

При выборе уровня доступа к пользовательским данным присутствуют две опции. Папка для приложения — для приложения будет использоваться конкретное место хранения данных в облачном хранилище. Полный доступ — сервис будет иметь доступ ко всей информации, хранящейся на диске. После регистрации сервиса нужно указать, какие конкретно права будут предоставлены. Для данной реализации были выбраны следующие параметры:

- files.metadata.write — Редактирование файлов и папок на диске;
- files.metadata.read — Просмотр файлов и папок на диске;

- `files.content.write` — Редактирование данных, связанных с файлами на диске;
- `files.content.read` — Просмотр данных, связанных с файлами на диске;

Генерация токена доступа может осуществляться двумя способами. Он может быть создан вручную в меню приложения, данный токен будет действителен в течение двух недель. Также возможна автоматическая генерация при помощи запроса на сервере. В данном случае пользователю необходимо подтвердить предоставление прав на использование данных приложением, перейдя на специальный адрес [11].

Для формирования запроса необходимо иметь токен доступа, а также уникальный адрес, идентифицирующий текущий запрос. В данной работе будут применены два запроса: скачивание файла и загрузка на диск. Запрос для скачивания файла принимает только его путь на облачном сервисе, загрузка файла имеет следующие параметры:

- `path` — Путь в Dropbox пользователя для сохранения файла.
- `mode` — Определяет модель поведения, если файл уже существует. По умолчанию для этого параметра — добавить файл.
- `autorename` — Является булевым значением. Если возникает конфликт (согласно настройке параметра `mode`), сервер Dropbox попытается автоматически переименовать файл, чтобы избежать конфликт. По умолчанию - `False`.
- `client_modified` — Является необязательным полем, позволяет указать фактическое время создания файла, помимо автоматически записываемого значения времени загрузки.
- `mute` — Булево значение, отвечает за получение пользователем уведомлений об изменениях файлов. Если установлено значение `True`, то клиент не будет получать оповещение. По умолчанию значение `False`.
- `property_groups` — Необязательное поле, позволяет добавить файлу пользовательские свойства.
- `strict_conflict` — Является булевым значением. Представляет собой более строгую проверку каждой записи вызывает конфликт, даже если целевой путь указывает на файл с идентичным содержимым.
- `content_hash` — Не обязательный параметр. Хеш содержимого загружаемого файла. Если указан и загруженное содержимое не соответствует

этому хешу, будет возвращена ошибка.

Реализация серверной части представляет собой конструктор и набор методов, совершающих запросы для скачивания и загрузки файлов. Конструктор класса представляет собой инициализацию экземпляра `httpClient`, токена доступа и шаблона адреса.

Метод загрузки файла принимает адрес файла, хранящегося на устройстве, а также путь к файлу на удалённом хранилище. В данной реализации все файлы будут храниться в корневой папке, в следствие чего путь к файлу будет обозначаться как строка `"/` объединённая с названием файла. Файл представляется в виде массива байтов. При формировании запроса используется метод `POST` с используемым для загрузки адресом. Также в запросе используются следующие поля:

- `path`
- `mode`
- `autorename`
- `mute`

Затем к запросу добавляется содержимое файла. Производится его выполнение и проверка статуса. На сервер возвращается ответ от сервиса. Пример данной реализации предоставлен на листинге ниже:

```
public async Task<string> UploadFile
(string localFilePath, string dropboxPath){
    byte[] fileContent = File.ReadAllBytes(localFilePath);
    \\Массив байтов файла
    var request = new HttpRequestMessage \\Тело запроса, POST,
    (HttpMethod.Post, $"{_apiUrl}upload"); \\тип загрузка файла
    request.Headers.Add("Dropbox-API-Arg", \\Параметры запроса
        JsonConvert.SerializeObject(new
        {
            path = dropboxPath, \\Путь к файлу
            mode = "overwrite", \\Перезапись если файл существует
            autorename = true, \\Разрешение конфликта названий
            mute = false \\Сохранение уведомлений
        }));
    request.Content = content; \\Передача файла в тело запроса
```



```

var response = await _httpClient.SendAsync(request);
response.EnsureSuccessStatusCode(); \\Исполнение запроса
return await response.Content.ReadAsStringAsync();
\\Ответ сервиса
}

```

Для скачивания файла передаётся путь, по которому он хранится на **Dropbox**. Формируется запрос типа **GET** с адресом для загрузки файла, затем формируются параметры, в которые указывается путь к файлу. После этого выполняется запрос, полученные данные десериализуются и преобразовываются в виде файла из массива байт. Пример данной реализации предоставлен на листинге ниже:

```

public async Task<byte[]> DownloadFile(string dropboxPath)
{
    var request = new HttpRequestMessage \\Формирование запроса
        (HttpMethod.Post, $"{_apiUrl}download");
    request.Headers.Add("Dropbox-API-Arg",
        JsonConvert.SerializeObject(new \\Настройка параметров
        {
            path = dropboxPath \\Путь к файлу
        }));
    var response = await _httpClient.SendAsync(request);
    \\Выполнение запроса
    response.EnsureSuccessStatusCode(); \\Статус запроса
    var result =
        JsonConvert.DeserializeObject<DropboxFileMetadata>(
            response.Headers.GetValues("Dropbox-API-Result").First());
    \\Дессериализация данных в массив байт
    return await response.Content.ReadAsByteArrayAsync();
}

```

## 4.4 GoogleDrive

Третьим выбранным облачным хранилищем является **Google Drive**. Данный сервис предоставляет полноценную библиотеку для **.NET**, реализующую взаимодействие с диском при помощи зарегистрированного приложения.

Установка данного пакета в среде Visual Studio производилась при помощи NuGet Package Manager. Данный инструмент позволяет устанавливать необходимые для проекта библиотеки, отслеживать изменения в версиях, а также производить изменения в установленных пакетах, отключать и переносить их [12]. Для применения были установлены следующий набор библиотек:

- Google.Apis — API для взаимодействия с сервисами Google, формирования запросов.
- Google.Apis.Drive.v3 — Является библиотекой для работы с диском.

Для взаимодействия с диском пользователя необходимо зарегистрировать клиент в сервисе Google Auth Platform. При создании приложения необходимо указать его название и платформу, для которой ведётся разработка. Среди них предоставляется выбор:

- Веб-приложение
- Приложение на систему Android
- Расширение для браузера
- Приложение на систему IOS
- Приложение для телевизоров и устройств с ограниченной гарнитурой
- Оконное приложение

Также необходимо указать адрес, на который будет перенаправлен пользователь для авторизации в приложении. Данный адрес должен совпадать с адресом, на котором работает сервер. Используемая библиотека при авторизации запроса запускает отдельный клиент, выбирая случайный свободный порт, в результате чего он должен оставаться пустым для автоматического определения.

Google Drive разделяет запрашиваемые данные на три уровня безопасности. В зависимости от максимального среди затронутых уровней на приложение могут накладываться ограничения. При авторизации пользователь будет уведомлен о том, насколько безопасно использовать данное приложение. В случае, если при разработке запрашиваются данные ограниченной области действия, пользователей можно добавлять только напрямую через консоль клиента до подтверждения безопасности приложения [13]. Под особо чувствительные запросы попадают:

- Просмотр и скачивание файлов
- Изменение содержимого данных

- Просмотр информации о файлах

Формирование запроса происходит при помощи файла с учетными данными пользователя. Данный файл может быть сформирован в меню приложения. Он хранит в себе следующие поля:

- `client_id` — Уникальный идентификатор приложения
- `project_id` — Идентификатор проекта в Google Cloud Platform, к которому привязаны учетные данные
- `auth_uri` — URL-адрес, куда направляется пользователь для авторизации
- `token_uri` — URL-адрес, на который направляется запрос для обмена `authorization code` на `access token` и `refresh token`
- `auth_provider_x509_cert_url` — Ссылка на сертификаты x509, используемые для проверки подлинности OAuth 2.0
- `client_secret` — Закрытый ключ
- `redirect_uris` — Список URI, на которые Google может перенаправлять пользователя после авторизации
- `javascript_origins` — Домены, с которых разрешены JavaScript-запросы к API Google

Класс, реализующий логику взаимодействия с облачным сервисом, имеет поля, отвечающие за путь к файлу с учётными данными пользователя, а также идентификатор папки, которая будет просматриваться для взаимодействия. Конструктор класса представляет собой инициализацию всех объявленных полей. Для применения данных пользователя в запросах применяется класс `UserCredential`, предоставленный библиотекой [14].

Метод загрузки файлов принимает путь к файлу на устройстве пользователя. При помощи него определяется название файла, его содержимое записывается в виде массива байтов. Используя полученные пользовательские данные, формируется экземпляр класса `GoogleClientSecrets`, при помощи которого запускается клиент для авторизации пользователя и дальнейшего взаимодействия с диском [15]:

```
UserCredential credentials;  
var clientSecrets = await  
    GoogleClientSecrets.FromFileAsync(credentialsPath);
```

```

credentials = await
    GoogleWebAuthorizationBroker.AuthorizeAsync(
        clientSecrets.Secrets,
        new[] { DriveService.ScopeConstants.DriveFile },
        "user",
        CancellationToken.None);

```

Затем создается экземпляр сервиса для формирования запроса, он хранит в себе поля с названием созданного приложения, а также клиент для формирования запроса. Загрузка файла требует указания определенной информации о нём, а именно его название и папку, к которой он принадлежит, поэтому создается отдельная переменная, хранящая поля с данными:

```

var service = new DriveService(new BaseClientService.Initializer()
{
    HttpClientInitializer = credentials,
    ApplicationName = "Raid5"
});
var fileMetaData = new Google.Apis.Drive.v3.Data.File()
{
    Name = Path.GetFileName(localFilePath),
    Parents = new List<string> { folderId }
};

```

В итоге открывается поток для чтения файла, который передается в метод **Create**, ожидающий поля с данными о файле, его содержимом в виде потока и типа данных. Данный метод подготавливает запрос с файлом для отправки. Файл асинхронно загружается на диск и по завершении возвращает статус операции:

```

using (var stream = new FileStream(localFilePath, FileMode.Open))
{
    request = service.Files.Create(fileMetaData, stream, "");
    request.Fields = "id";
    var progress = await request.UploadAsync();
    if (progress.Status != UploadStatus.Completed)

```

```

        throw new Exception($"Upload failed:
        {progress.Exception?.Message}");
    }

```

Для реализации загрузки файла с облачного диска необходимо получить его идентификатор. В данном случае формируется запрос, запрашивающий список всех файлов в данной папке на диске, после чего файлы фильтруются по указанному названию и проверяются, чтобы они не считались удалёнными. В итоге он исполняется и возвращает первое подходящее под условия вхождение:

```

var request = service.Files.List();
request.Q = $"name = '{fileName}' and trashed = false";
request.Fields = "files(id, name)";

var result = await request.ExecuteAsync();
return result.Files.FirstOrDefault()?.Id;

```

Скачивание файлов производится аналогичным образом. Производится подготовка клиента для формирования запроса, однако в этот раз для него необходимо получить идентификатор. Для этого применяется метод, подготовленный `FindFileIdByName`, он принимает на вход название и подготовленный экземпляр сервиса. Данный метод возвращает уникальный идентификатор файла, находя его по названию в выбранной папке. Затем открывается поток для сохранения файла на устройстве пользователя, в теле которого исполняется сам запрос. Пример приведён на листинге ниже:

```

var fileId = await FindFileIdByName(fileName, service);
var fileInfo = await service.Files.Get(fileId).ExecuteAsync();
if (fileInfo == null)
    throw new Exception("File not found in Google Drive");
using (var fileStream = new
FileStream(localSavePath, FileMode.Create, FileAccess.Write))
{
    var request = service.Files.Get(fileId);
    await request.DownloadAsync(fileStream);
}

```

В данном классе был подготовлен метод для получения названий всех файлов, который необходим для корректного отображения информации в пользовательском интерфейсе. При выполнении запроса собирается список из названий всех файлов, хранимых в указанной папке. Пример приведён на листинге ниже:

```
var request = service.Files.List();
request.Q = $"''{folderId}' in parents and trashed = false";
request.Fields = "files(name)";

var result = await request.ExecuteAsync();
return result.Files.Select(file => file.Name).ToList();
```

## 5 Реализация класса, распределяющего данные среди облачных хранилищ по принципу RAID 5

После того, как все методы для взаимодействия с облачными сервисами были описаны можно начинать разработку реализации для организации RAID 5. Данная реализация представляет собой класс, содержащий в себе набор методов для взаимодействия со всеми предоставленными облачными хранилищами. Список всех реализованных методов:

- Загрузка данных с распределением на все диски
- Скачивание файла, при условии что все диски доступны
- Скачивание данных, если не доступен Яндекс Диск
- Скачивание данных, если не доступен Dropbox
- Скачивание данных, если не доступен Google Drive
- Расчет чётности для двух массивов байт

RAID 5 уровня подразумевает распределение файлов среди трёх и более дисков с подготовкой чётности для их элементов. В данном решении файл будет представлен в виде нескольких массивов байт для организации RAID, они распределяются вместе с массивами четности среди трех файлов. Операция преобразования в файл добавляет дополнительные вычисления, ввиду чего был применён способ, в котором все представленные части данных распределяются среди трех файлов, что позволяет сократить количество лишних вычислений. Данный подход минимизирует повторение операций при работе с облачными хранилищами.

Данная модель может быть модифицирована рядом способов. Была реализована возможность различных способов распределения данных среди хранилищ. За счет данного подхода модель хранения может быть усложнена пользователем, повышая степень конфиденциальности данных, определить какие части файла на текущем хранилища становится сложнее.

Также в данном решении был реализован метод, при котором части изначальных данных могут быть заданы динамически. Подготовленный ключ представляет собой конфигурацию, при помощи которой пользователь может модифицировать размер подготавливаемых данных. Благодаря данному подходу данные сложнее скомпрометировать, не зная уникальный ключ, набор байтов не получится разделить.

Подготовленная модель предлагает распределение байтов файла по ча-

стям с разными размерами. Например, размер первой части с массивами  $A_1$ ,  $B_1$ ,  $Parity_{A_1B_1}$  могут составлять 10% изначального файла, размер второй части  $A_2$ ,  $B_2$ ,  $Parity_{A_2B_2}$  может составлять 20%, а третья часть заполняется всем оставшимися байтами. Для реализации данного пользователь должен хранить файл с ключом и предоставлять его для получения файла.

По скольку в данном решении необходимо множество раз пересчитывать чётность для каждого массива байт, был реализован отдельный метод, принимающий на вход два набора данных. Данный алгоритм проходит по каждой паре элементов, производя логическую операцию  $A \oplus B = Parity$ , возвращая подготовленные данные:

```
public byte[] SolveParity(byte[] data_1, byte[] data_2)
{
    byte[] parity = new byte[data_1.Length];
    for(int i = 0; i < parity.Length; i++)
        parity[i] = (byte)(data_1[i] ^ data_2[i]);
    return parity;
}
```

### 5.1 Загрузка файла с распределением

Для распределения файла по принципу RAID 5 необходимо разбить его данные, представленные в виде набора байтов. Это достигается путем считывания выбранного файла и записывания его содержимого в массив в виде набора байтов. Распределение производится в массивы заданного размера. Для обработки ситуации, когда файл завершается до полного распределения по фиксированному заданному размеру, оставшийся объем массива байтов заполняется нулями для корректности выполнения операции XOR. Пример предоставлен на листинге ниже:

```
int j = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    if (j < fileContent.Length)
    {
        data_1[i] = fileContent[j];
```



```

        j++;
    }
    else
    {
        data_1[i] = 0;
        j++;
    }
}

```

Данные массивы преобразуются в виде файлов, хранящих в себе набор подготовленных байтов, они в свою очередь загружаются на диск. Далее были созданы массивы для хранения четности пар данных, все три массива представлены в виде одного отдельного файла. Для загрузки собранного файла из байтов необходимо его создать на сервере и после отправки удалить. Загрузка производится за счёт ранее реализованных сервисов для взаимодействия с облачными хранилищами. Пример предоставлен на листинге ниже:

```

string fileName = Path.GetFileNameWithoutExtension(filePath);
string cur_fileName = fileName + "_A";
await File.WriteAllBytesAsync(
    Path.Combine(dir, cur_fileName), data_a);

await _yandexdiskService.UploadFile(
    cur_fileName, Path.Combine(dir, cur_fileName));
File.Delete(Path.Combine(dir, cur_fileName));

```

Таким образом, производится загрузка 3 файлов данных вместе с подготовленными для каждой из пар массивами четности. В конце метода производится обработка ответов от сервисов. В случае возникновения ошибок или конфликтов производится обработка их уведомлений.

## 5.2 Считывание данных

Скачивание данных с дисков производится за счёт поиска на них двух файлов, хранящих данные об изначальном файле, либо считывания чётности, в случае, если один из дисков в данный момент недоступен. Для загрузки данных создаётся массив, который будет хранить в себе набор байтов всех его

частей. Загрузка производится за счёт подготовленных ранее сервисов, они возвращают набор байтов данного файла. Пример предоставлен на листинге ниже:

```
counter = 0;
for(int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
```

Для обработки случаев, в которых один из дисков недоступен, подготовлено 3 метода, реализующих восстановление данных при помощи массивов чётности. Каждый метод реализует загрузку данных, в зависимости от того, какая информация была утеряна. В данном случае производится предварительная загрузка массивов чётности для потерянных данных, а также файла, хранящего в себе 2, 4 и 6 части файла и производится операция  $data\_2 \oplus Parity = data\_1$  для восстановления недоступного элемента пары:

```
counter = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_12_parity[i] = data_parity[counter];
    counter++;
}
byte[] data_1 = SolveParity(data_12_parity, data_2);
```

## 6 Разработка пользовательского интерфейса

Для визуального представления работы программы был разработан пользовательский интерфейс. Он был написан при помощи фреймворка `Vue.js`. Данный фреймворк имеет встроенные инструменты для разработки, такие как `Vue DevTools`, которые позволяют в реальном времени отслеживать состояние компонентов, их данные и события, что значительно упрощает процесс отладки.

Помимо вышеописанного фреймворка, основой одностраничного приложения стали технологии `HTML` и `CSS`. `HTML` (HyperText Markup Language) — это стандартный язык разметки, используемый для создания веб-страниц. Браузеры загружают `HTML`-документы с сервера используя протоколы `HTTP/HTTPS` либо с локального устройства, а затем преобразуют код в визуальный интерфейс, отображаемый на экране [16].

Основу `HTML` составляют элементы, которые формируют структуру страницы. С их помощью можно добавлять текстовые блоки, изображения, формы, ссылки и другие интерактивные компоненты. Элементы обозначаются тегами, заключёнными в угловые скобки. Хотя сами теги не отображаются в браузере, они определяют, как будет выглядеть и функционировать контент веб-страницы.

`CSS` (Cascading Style Sheets) — это язык стилей, который определяет внешний вид и оформление веб-документов, написанных на `HTML` или `XML` (включая `XHTML`, `SVG` и другие форматы). С помощью `CSS` можно задавать стили для элементов страницы, контролируя их отображение на экране, при печати, в голосовых интерфейсах и других медиа [17].

`CSS` работает по принципу каскадных правил, позволяя управлять макетом, шрифтами, цветами, анимацией и другими визуальными аспектами веб-страницы. В отличие от `HTML`, который отвечает за структуру контента, `CSS` сосредоточен на его презентации, обеспечивая разделение содержания и дизайна.

При разработке интерфейса был задействован `HTML` элемент `button`. Данный элемент может быть использован в любой части страницы, выполняет функцию кнопки, при нажатии на которую будет производиться заранее заготовленное событие. Её внешний вид может быть отредактирован при помощи `CSS`. Этот компонент имеет следующие поля для настройки [18]:

- autofocus — Этот атрибут определяет, получит ли кнопка фокус автоматически при загрузке страницы. Фокус останется на элементе, пока пользователь вручную не переключится на другой объект. В пределах одной формы только один элемент может иметь этот атрибут.
- autocomplete — Этот атрибут поддерживается только в браузере Firefox. В отличие от других браузеров, Firefox сохраняет состояние отключенной кнопки (установленное динамически) даже после перезагрузки страницы.
- disabled — Данный атрибут отключает взаимодействие пользователя с кнопкой. Когда ни у кнопки, ни у её контейнера нет атрибута disabled, элемент остаётся активным и доступным для действий.
- form — Это поле позволяет указать элемент, с которым связана кнопка.
- formaction — Определяет URL-адрес, на который будет отправлена форма при нажатии кнопки.
- formenctype — Определяет тип контента, отправляемого на сервер.
- formmethod — Определяет метод HTTP-запроса для отправки данных на сервер.
- formnovalidate — Указывает, что данные формы не будут валидироваться при отправке.
- formtarget — Указывает, где отображать ответ, полученный после отправки формы.
- name — Хранит название элемента.
- type — Устанавливает тип кнопки. Доступны следующие параметры:
  - submit — Кнопка отправляет данные формы на сервер.
  - reset — Кнопка сбрасывает все элементы управления к их начальным значениям.
  - button — Кнопка не имеет поведения по умолчанию.
  - menu — Кнопка открывает всплывающее меню.
- value — Исходное значение элемента.

В работе данный элемент применялся в нескольких местах. Например, кнопка, отвечающая за загрузку файла, вызывает соответствующий метод и имеет поле, делающее её недоступной до выбора пользователем файла:

```
<button
  @click="downloadFiles()"
```

```
class="download-btn"
:disabled="!selectedServerFile"
>
  Скачать
</button>
```

На рисунке 7 продемонстрировано применение данного компонента на пользовательском интерфейсе в состоянии, когда он недоступен.



Рисунок 7 – Элемент `button` в неактивном состоянии с полем `disabled`

На рисунке 8 предоставлен элемент `button` при активном состоянии поля `disabled`.



Рисунок 8 – Элемент `button` в активном состоянии с полем `disabled`

`V-if` и `V-for` — это специальные директивы в Vue.js, которые используются для управления отображением элементов и итерации по массивам или объектам соответственно.

- `V-if` — это директива условного рендеринга, которая позволяет отображать или скрывать элементы на основе логического условия. Если условие истинно (`true`), элемент будет отрендерен, если ложно (`false`) — он не будет присутствовать в DOM [19].
- `V-for` — это директива для циклического рендеринга, которая позволяет итеративно отрисовывать элементы на основе массива, объекта или числа. Каждый элемент списка рендерится как отдельный DOM-элемент [20].

Компонент **V-if** в данной работе применялся для отображения элементов при соблюдении определённых условий. **V-for** позволяет отображать список элементов массива. Всплывающий список выбора элемента для загрузки отображается при условии нажатия соответствующей кнопки. При нажатии происходит переключение булевой переменной для отслеживания текущего состояния. Названия отображаются циклично, сортируясь по их индексу, при нажатии на название вызывается метод запоминающий выбранный файл и переключающий состояние отображения:

```
<button @click="showFileDropdown = !showFileDropdown"
class="select-btn">
    Выбрать файл для скачивания
</button>
<div v-if="showFileDropdown" class="dropdown">
    <div
        v-for="(file, index) in state.files"
        :key="index"
        class="dropdown-item"
        @click="selectServerFile(file)"
    >
        {{ file }}
    </div>
</div>
<div v-if="selectedServerFile" class="selected-file">
    Выбран файл: {{ selectedServerFile }}
</div>
```

Пример отображения данного элемента на пользовательском интерфейсе продемонстрирован на рисунке 9.

## Выбрать файл для скачивания

presentation

sql

IMG\_3074

Рисунок 9 – Всплывающее окно выбор файла

Элемент для выбора способа загрузки файла был реализован в виде `radio button`. Данный компонент позволяет выбирать пользователю один из предложенных вариантов пользователю. Для определения его параметров объявляется отдельный список:

```
const options = ref([
  { value: 'option1', label: 'Загрузка со всех дисков' },
  { value: 'option2', label: 'Загрузка без Яндекс Диска' },
  { value: 'option3', label: 'Загрузка без Dropbox' },
  { value: 'option4', label: 'Загрузка без Google Drive' },
  { value: 'option5', label: 'Загрузка случайным методом' }
]);
```

Отображение происходит при помощи элемента `V-for` сортируя по указанному в списке значению. Поле `type` отвечает за тип компонента, в данном случае это `radio button`, `v-model` позволяет синхронизировать данное значение с кодом JavaScript для применения его в остальных методах:

```
<label v-for="option in options"
```

```
:key="option.value" class="radio-label">
<input
  type="radio"
  v-model="selectedOption"
  :value="option.value"
>
```

В зависимости от выбранного метода пользователь может восстановить утерянные данные с неработающего диска. При выборе опции со всеми дисками загрузка будет производиться быстрее за счёт отсутствия дополнительных операций для восстановления данных при помощи хранящихся массивов четности. Пример отображения данного элемента продемонстрирован на рисунке 10.

### **Выберите метод загрузки:**

- ☒ **Загрузка со всех дисков**
- ☐ **Загрузка без Яндекс Диска**
- ☐ **Загрузка без Dropbox**
- ☐ **Загрузка без Google Drive**
- ☐ **Загрузка случайным методом**

Рисунок 10 – Окно выбора метода загрузки

Также были реализованы элементы интерфейса, позволяющие пользователю редактировать метод распределения данных для конкретных задач. Выбранные параметры передаются в виде полей запроса для обработки на



серверной части. Пример данного элемента продемонстрирован на рисунке 11.

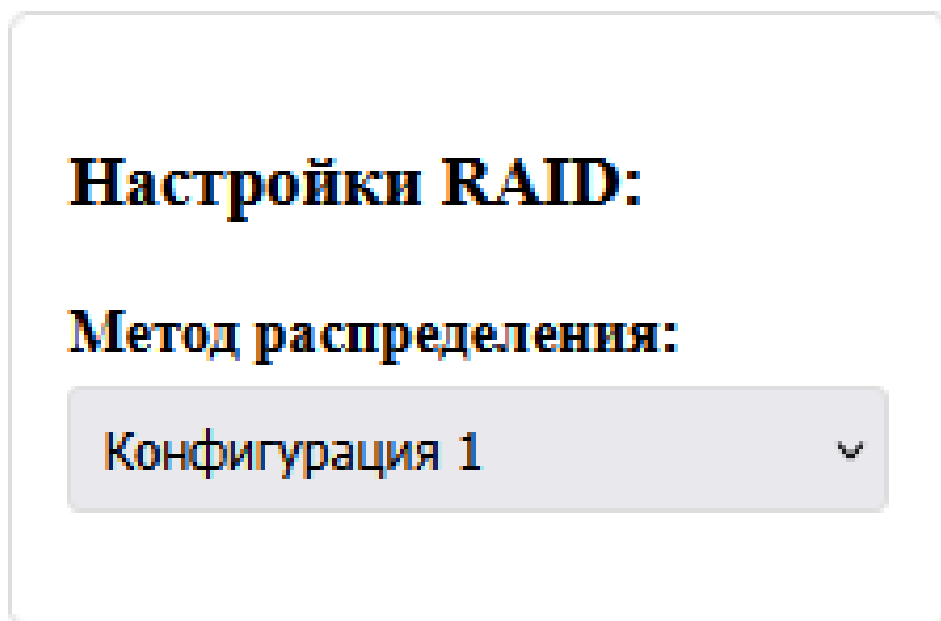


Рисунок 11 – Окно выбора метода загрузки

В левой части экрана расположен список всех имеющихся в данный момент на облаке файлов. Данный список реализован посредством циклического отображения всех элементов. Вызов метода для получения набора файлов производится при инициализации страницы. Он производится путем запроса данных с сервера:

```
const getFilesNames = async () => {  
  const response = await fetch(  
    'https://localhost:7229/api/GoogleDrive/files';  
  return await response.json();  
}
```

Метод скачивания файлов реализует запрос к back-end части приложения. Изначально производится проверка, был ли выбран файл для скачивания, в случае если нет, метод приостанавливает свою работу, уведомляя пользователя о необходимости выбора. Метод загрузки определяется за счет значения, полученного от элемента `radio button`:

```
var responseUrl = ''
```

```

switch(selectedOption.value) {
  case 'option1':
    responseUrl = 'https://localhost:7229/api/RAID5/
download/${selectedServerFile.value}';
    break;
  case 'option2':
    responseUrl = 'https://localhost:7229/api/RAID5/
downloadwithoutya/${selectedServerFile.value}';
    break;
  case 'option3':
    responseUrl = 'https://localhost:7229/api/RAID5/
downloadwithoutgoogle/${selectedServerFile.value}';
    break;
  case 'option4':
    responseUrl = 'https://localhost:7229/api/RAID5/
downloadwithoutdropbox/${selectedServerFile.value}';
    break;
  case 'option5':
    const randomOption = Math.floor(Math.random() * 4) + 1;
    responseUrl = 'https://localhost:7229/api/RAID5/${
      randomOption === 1 ? 'download' :
      randomOption === 2 ? 'downloadwithoutya' :
      randomOption === 3 ? 'downloadwithoutgoogle'
      : 'downloadwithoutdropbox'
    }/${fileName}?distributionMethod=
      ${selectedDistributionMethod.value}';
    break;
}
const response = await fetch(responseUrl);

```

При определении метода в отдельную переменную сохраняется адрес для проведения запроса. При помощи данной переменной производится запрос к контроллеру по сохраненному адресу с текущим значением пути к файлу. По завершении запроса начинается сборка файла и в последствии его скачивание для применения.

Загрузка файла на диски производится при помощи формирования POST запроса. При помощи параметров запроса в контроллер передаётся строка, отвечающая за местонахождение файла на физическом носителе.

В случае возникновения конфликта во всплывающем окне отображается причина ошибки.

При загрузке файла picture.png на облачное хранилище он представляется в виде 3 файлов, хранящих в себе распределённые по облачным сервисам наборы байт.

Пример полного интерфейса для работы с приложением предоставлен на рисунке 12.

The interface consists of several components:

- File Type Selection:** Three input fields labeled "text", "picture", and "archive".
- Buttons:**
  - "Выбрать ключ" (Select key) - blue button.
  - "Выбрать файл для загрузки" (Select file for upload) - blue button.
  - "Загрузить файл" (Upload file) - grey button.
  - "Выбрать файл для скачивания" (Select file for download) - blue button.
  - "Скачать" (Download) - grey button.
- Download Method Selection:** A section titled "Выберите метод скачивания:" (Choose download method:) with five radio button options:
  - ☒ Скачать со всех дисков (Download from all disks)
  - ☐ Скачать без Яндекс Диска (Download without Yandex Disk)
  - ☐ Скачать без Dropbox
  - ☐ Скачать без Google Drive
  - ☐ Скачать случайным методом (Download by random method)
- RAID Settings:** A section titled "Настройки RAID:" (RAID settings:) with a sub-section "Метод распределения:" (Distribution method:) containing a dropdown menu currently set to "Конфигурация 1" (Configuration 1).

Рисунок 12 – Полная версия интерфейса

Полный код программной реализации предоставлен в Приложениях В, Г, Д, Е, Ж, З, И, К.

## 7 Тестирование реализованного решения

Для проверки корректности работы алгоритма восстановления данных по принципу RAID 5 используется следующий подход:

1. Пользователь загружает файл  $F$ , который разбивается на блоки и распределяется между облачными хранилищами с вычислением контрольных блоков чётности.
2. В дальнейшем файл восстанавливается из двух блоков и одного блока чётности, находящихся в разных хранилищах.
3. Полученный восстановленный файл  $F'$  сравнивается с исходным  $F$  двумя способами:
  - побайтовое сравнение содержимого;
  - сравнение хеш-сумм:

$$\text{SHA256}(F) \stackrel{?}{=} \text{SHA256}(F').$$

4. При совпадении считается, что восстановление выполнено корректно.

Схема данного алгоритма представлена на рисунке 13.

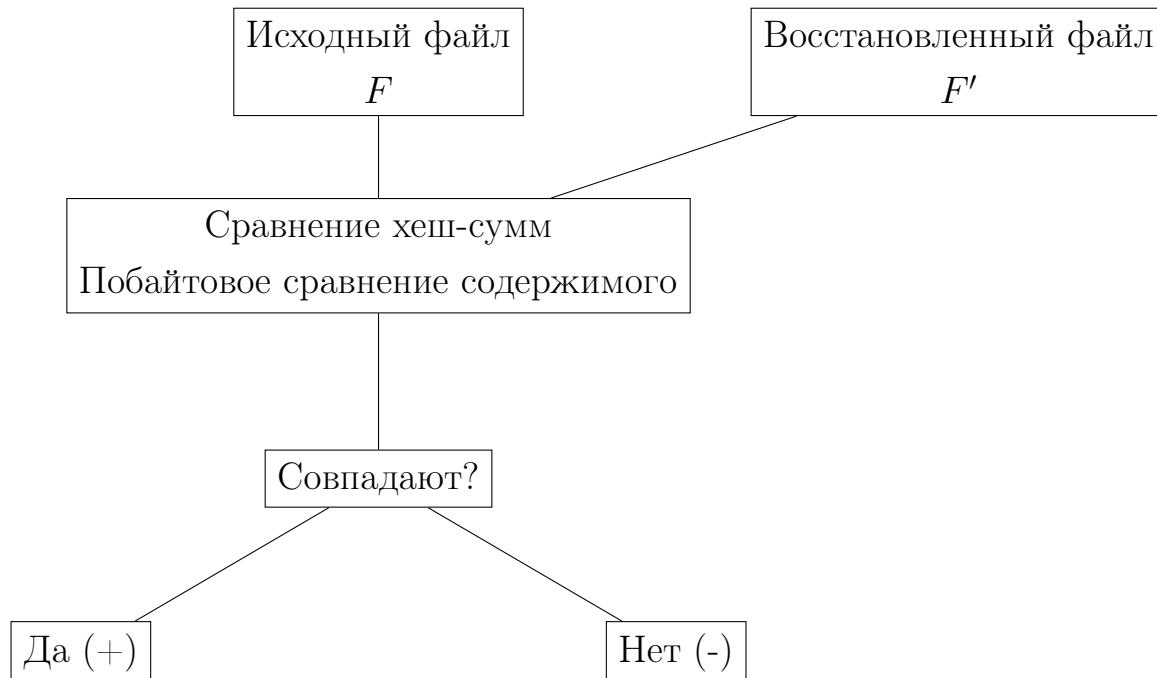


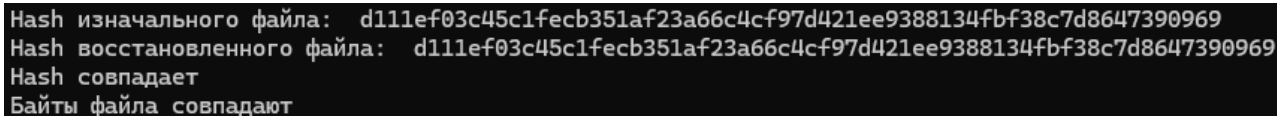
Рисунок 13 – Схема сравнения исходного и восстановленного файла

Для реализации данного тестирования была написана вспомогательная программа, производящая сравнение хеш-сумм, а также байтов двух выбранных файлов. Сравнение байтов производится путём представления файла

в виде массивов байтов и последующего сравнения значений. Для сравнения хеша создаётся экземпляр класса SHA256. Данный класс имеет метод для вычисления хеша файла, полученные значения сравниваются и результат сравнения выводится в консоль. Пример реализации метода вычисления хеш-суммы предоставлен на листинге ниже:

```
static string GetHash(string filePath)
{
    using var sha256 = SHA256.Create();
    using var stream = File.OpenRead(filePath);
    var hash = sha256.ComputeHash(stream);
    return BitConverter.ToString(hash)
        .Replace("-", "").ToLowerInvariant();
}
```

Тестирование было произведено при помощи сравнения изначального файла и файла, полученного после скачивания с применением разработанного решения. Результат данного теста представлен на рисунке 14.



```
Hash изначального файла: d111ef03c45c1fecb351af23a66c4cf97d421ee9388134fbf38c7d8647390969
Hash восстановленного файла: d111ef03c45c1fecb351af23a66c4cf97d421ee9388134fbf38c7d8647390969
Hash совпадает
Байты файла совпадают
```

Рисунок 14 – Результат сравнения изначального файла и восстановленного

Также было проведено тестирование производительности. Размер файла составляет 1,5 Мбайта на устройстве с процессором 12th Gen Intel(R) Core(TM) i7-12700H частота 2.30 МГц. Объем оперативной памяти 16 ГБайт с частотой 4800 МГц. По результатам тестирования прослеживается следующая динамика:

- Среднее время чтения — 5.6 с.
- Среднее время записи — 11.5 с.
- Среднее время восстановление данных Яндекс Диска — 7.3 с.
- Среднее время восстановление данных Google Drive — 7.3 с.
- Среднее время восстановление данных Dropbox — 4.4 с.

Результаты тестов демонстрируют стабильность работы системы: отклонения в времени выполнения операций незначительны (например, для записи — от 5.13 до 6.32 секунд). Восстановление данных при потере Google

Drive происходит быстрее, чем для других сервисов в силу того, что применённая библиотека затрачивает дополнительные ресурсы. Полученные в ходе тестирования значения предоставлены в таблице 1.

Таблица 1 – Результаты тестов производительности

№	Запись, с	Чтение, с	Восстановление Яндекс Диска, с	Восстановление Google Drive, с	Восстановление Dropbox, с
1	11.32	5.65	7.30	4.55	7.52
2	12.54	5.84	7.43	4.62	7.48
3	11.84	6.32	8.15	4.21	7.05
4	11.02	5.55	7.11	4.80	7.19
5	11.21	5.87	7.62	4.17	7.41
6	12.01	5.33	8.08	4.28	7.26
7	11.52	5.61	7.17	4.11	7.39
8	11.23	5.65	7.34	4.36	7.14
9	11.41	5.22	7.53	4.15	7.66
10	11.62	5.13	7.41	5.02	7.16

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была исследована модель распределенного хранения данных для повышения защищенности. В качестве основной для практического исследования была выбрана модель на основе принципа RAID 5. Основной целью разработки стало повышение отказоустойчивости и защищённости данных при размещении в облачных сервисах. Для демонстрации работы модели был разработан пакет программ, иллюстрирующих метод хранения.

Также было разработано веб-приложение, использующее облачные сервисы в качестве хранилищ. Оно включает в себя пользовательский интерфейс, логику обработки файлов и модуль взаимодействия с облачными хранилищами.

Для повышения защищенности хранения реализован механизм динамического задания размеров частей данных при разбиении файла, что делает схему хранения менее предсказуемой и затрудняет восстановление информации при частичном доступе к данным. Также предусмотрены различные сценарии восстановления файлов при недоступности одного из облаков, что подтверждает устойчивость системы к отказам.

Проведённое тестирование подтвердило применимость модели, устойчивость к сбоям, а также корректность полученных данных. Реализация проекта показала, что подход RAID 5 может быть успешно адаптирован для распределённой облачной инфраструктуры.

Таким образом, в рамках работы создано решение, сочетающее преимущества облачного хранения и отказоустойчивости, а также повышение защищённости при хранении данных с применением подготовленной модели, что делает его актуальным инструментом для повышения защищенности при хранении данных.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Интерфакс [Электронный ресурс] URL: <https://www.interfax.ru/russia/995242> (дата обращения: 05.01.2025)
- 2 Chen P.M. RAID: High-performance, reliable secondary storage / P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, D.A. Patterson // ACM Computing Surveys (CSUR). – 1994г. – Vol. 26. – P. 145–185.
- 3 Массивы дисков в RAID [Электронный ресурс] URL: <https://1bx.host/stati/obshchie-stat/massivy-diskov-v-raid0-raid1-raid5-raid10/> (дата обращения: 19.01.2025)
- 4 Павлова А.А. Получение доступа к данным, содержащимся в RAID 5 / Павлова А.А., Молодцова Ю.В. // Вестник Алтайской академии экономики и права. - 2022г. -№6-2. -с. 356-360.
- 5 Swagger Docs [Электронный ресурс] URL: <https://swagger.io/docs/> (дата обращения: 25.01.2025)
- 6 Cross-Origin Resource Sharing (CORS) [Электронный ресурс] URL: <https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/CORS> (дата обращения: 03.02.2025)
- 7 ASP.NET Core [Электронный ресурс] URL: <https://learn.microsoft.com/ru-ru/aspnet/core/security/cors?view=aspnetcore-9.0> (дата обращения: 04.02.2025)
- 8 Яндекс ID [Электронный ресурс] URL: <https://yandex.ru/dev/id/doc/ru/register-client> (дата обращения: 13.02.2025)
- 9 Получение OAuth-токена [Электронный ресурс] URL: <https://yandex.ru/dev/id/doc/ru/access> (дата обращения: 13.02.2025)
- 10 Dropbox Documentation [Электронный ресурс] URL: [https://www.dropbox.com/developers/documentation?\\_tk=pilot\\_lp&\\_ad=topbar1&\\_camp=docs](https://www.dropbox.com/developers/documentation?_tk=pilot_lp&_ad=topbar1&_camp=docs) (дата обращения: 20.02.2025)
- 11 Dropbox OAuth guide [Электронный ресурс] URL: <https://www.dropbox.com/lp/developers/reference/oauth-guide.html> (дата обращения: 22.02.2025)



- 12 NuGet Package Manager [Электронный ресурс] URL:  
<https://learn.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio> (дата обращения: 02.03.2025)
- 13 OAuth 2.0 Scopes for Google APIs [Электронный ресурс] URL:  
<https://developers.google.com/identity/protocols/oauth2/scopes> (дата обращения: 04.03.2025)
- 14 Class UserCredential [Электронный ресурс] URL:  
<https://cloud.google.com/dotnet/docs/reference/Google.Apis/latest/Google.Apis.Auth.OAuth2.UserCredential> (дата обращения: 10.03.2025)
- 15 Class GoogleClientSecrets [Электронный ресурс] URL:  
<https://cloud.google.com/java/docs/reference/google-api-client/latest/com.google.api.client.googleapis.auth.oauth2.GoogleClientSecrets>  
(дата обращения: 12.03.2025)
- 16 mdn web docs HTML [Электронный ресурс] URL:  
<https://developer.mozilla.org/ru/docs/Web/HTML> (дата обращения: 22.03.2025)
- 17 CSS: каскадные таблицы стилей [Электронный ресурс] URL:  
<https://developer.mozilla.org/ru/docs/Web/CSS> (дата обращения: 23.03.2025)
- 18 <button> - элемент кнопки [Электронный ресурс] URL:  
<https://developer.mozilla.org/ru/docs/Web/HTML/Reference/Elements/button> (дата обращения: 28.03.2025)
- 19 Conditional Rendering [Электронный ресурс] URL:  
<https://vuejs.org/guide/essentials/conditional.html> (дата обращения: 05.04.2025)
- 20 Отрисовка списков [Электронный ресурс] URL:  
<https://ru.vuejs.org/guide/essentials/list> (дата обращения: 05.04.2025)

## ПРИЛОЖЕНИЕ А

### Исходный код модуля распределения данных RAID 5

```
using System.ComponentModel;
using System.Security.Cryptography;
using System.Text;
Console.OutputEncoding = System.Text.Encoding.UTF8;
// Запрос пути к файлу
Console.WriteLine("Введите путь к файлу");
string filePath = Console.ReadLine();
// Считывание файла в виде набора байт
byte[] fileContent = await File.ReadAllBytesAsync(filePath);
int reduce = 0;
// Поиск ключа
string configDir = Directory.GetCurrentDirectory();
string configPath = Path.Combine(configDir, "Config.txt");
string configContent = File.ReadAllText(configPath);
bool isFound = false;
string numberString = "";
for(int i = 0; i < configContent.Length; i++)
{
    if (configContent[i] == '\n')
        isFound = true;
    if(isFound == true)
    {
        numberString += configContent[i];
    }
}
// Извлечение данных из ключа
string[] partsConfig = numberString.Split(',');
// Процентное соотношение размера частей
int part_portional_a = int.Parse(partsConfig[0]);
int part_portional_b = int.Parse(partsConfig[1]);
// Проверка что их сумма не превышает размер файла
if (part_portional_a + part_portional_b >= 100)
```

```

{
    Console.WriteLine("Сумма частей превышает размер файла");
    Environment.Exit(0);
}
//Размер изначального файла
byte[] originalSize = BitConverter.GetBytes(fileContent.Length);
// Расчет размера первого набора
int part_size_a = fileContent.Length / 100 * part_portional_a / 2;
// Расчет размера второго набора
int part_size_b = fileContent.Length / 100 * part_portional_b / 2;
int part_size_c = 0;
if (part_size_a + part_size_b < fileContent.Length)
{
    part_size_c += (int)Math.Ceiling((
        fileContent.Length - (part_size_a
            + part_size_b) * 2) / 2.0);
}
if (fileContent.Length % 2 != 0)
{
    reduce = 1;
}
int j = 0;
// Представление файла в виде массивов байт
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < data_1.Length; i++)
{
    if (j < fileContent.Length)
    {
        data_1[i] = fileContent[j];
        j++;
    }
    else
    {
        data_1[i] = 0;
    }
}

```

```

        j++;
    }
}
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < data_2.Length; i++)
{
    if (j < fileContent.Length)
    {
        data_2[i] = fileContent[j];
        j++;
    }
    else
    {
        data_2[i] = 0;
        j++;
    }
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < data_3.Length; i++)
{
    if (j < fileContent.Length)
    {
        data_3[i] = fileContent[j];
        j++;
    }
    else
    {
        data_3[i] = 0;
        j++;
    }
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < data_4.Length; i++)
{

```

```

    if (j < fileContent.Length)
    {
        data_4[i] = fileContent[j];
        j++;
    }
    else
    {
        data_4[i] = 0;
        j++;
    }
}

byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < data_5.Length; i++)
{
    if (j < fileContent.Length)
    {
        data_5[i] = fileContent[j];
        j++;
    }
    else
    {
        data_5[i] = 0;
        j++;
    }
}

byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < data_6.Length; i++)
{
    if (j < fileContent.Length)
    {
        data_6[i] = fileContent[j];
        j++;
    }
}

```

```

        else
        {
            data_6[i] = 0;
            j++;
        }
    }
    // Расчет массивов четности
    // И заполнение итоговых файлов
    byte[] data_1_final = new byte[part_size_a
        + part_size_b + part_size_c + 4];
    j = 0;
    for (int i = 0; i < data_1.Length; i++)
    {
        data_1_final[j] = data_1[i];
        j++;
    }
    for (int i = 0; i < data_3.Length; i++)
    {
        data_1_final[j] = data_3[i];
        j++;
    }
    byte[] data_56_parity = SolveParity(data_5, data_6);
    for (int i = 0; i < data_56_parity.Length; i++)
    {
        data_1_final[j] = data_56_parity[i];
        j++;
    }
    for (int i = 0; i < 4; i++)
    {
        data_1_final[j] = originalSize[i];
        j++;
    }
    byte[] data_2_final = new byte[part_size_a
        + part_size_b + part_size_c + 4];

```

```

j = 0;
for (int i = 0; i < data_2.Length; i++)
{
    data_2_final[j] = data_2[i];
    j++;
}
byte[] data_34_parity = SolveParity(data_3, data_4);
for (int i = 0; i < data_34_parity.Length; i++)
{
    data_2_final[j] = data_34_parity[i];
    j++;
}
for (int i = 0; i < data_5.Length; i++)
{
    data_2_final[j] = data_5[i];
    j++;
}
for (int i = 0; i < 4; i++)
{
    data_2_final[j] = originalSize[i];
    j++;
}
byte[] data_3_final = new byte[part_size_a
    + part_size_b + part_size_c + 4];
j = 0;
byte[] data_12_parity = SolveParity(data_1, data_2);
for (int i = 0; i < data_12_parity.Length; i++)
{
    data_3_final[j] = data_12_parity[i];
    j++;
}
for (int i = 0; i < data_4.Length; i++)
{
    data_3_final[j] = data_4[i];

```

```

        j++;
    }
    for (int i = 0; i < data_6.Length; i++)
    {
        data_3_final[j] = data_6[i];
        j++;
    }
    for (int i = 0; i < 4; i++)
    {
        data_3_final[j] = originalSize[i];
        j++;
    }
    string dir = Directory.GetCurrentDirectory();
    // Сохранение разделенных файлов
    string fileName = Path
        .GetFileNameWithoutExtension(filePath);
    string cur_fileName = fileName + "_1";
    await File.WriteAllBytesAsync
        (Path.Combine(dir, cur_fileName), data_1_final);
    cur_fileName = fileName + "_2";
    await File.WriteAllBytesAsync
        (Path.Combine(dir, cur_fileName), data_2_final);
    cur_fileName = fileName + "_3";
    await File.WriteAllBytesAsync
        (Path.Combine(dir, cur_fileName), data_3_final);
    Console.WriteLine("Файлы успешно представлены
        в виде набора байтов, они сохранены в виде файлов");
    Console.ReadLine();
    // Метод расчета четности
    byte[] SolveParity(byte[] data_1, byte[] data_2)
    {
        byte[] parity = new byte[data_1.Length];
        for (int i = 0; i < parity.Length; i++)
            parity[i] = (byte)(data_1[i] ^ data_2[i]);
    }

```



```
    return parity;  
}
```

## ПРИЛОЖЕНИЕ Б

### Исходный код модуля восстановления данных RAID 5

```
using System.Security.Cryptography;
using System.Text;

Console.OutputEncoding = System.Text.Encoding.UTF8;

string cur_fileName;
string cur_filepath;
string response1 = "2";
string response2 = "2";
string response3 = "2";
string file1Path = "";
string file2Path = "";
string file3Path = "";
// Получение пути к ключу
string configDir = Directory.GetCurrentDirectory();
string configPath = Path.Combine(configDir, "Config.txt");
string configContent = "";
if (File.Exists(configPath))
{
    configContent = File.ReadAllText(configPath);
}
else
{
    Console.WriteLine("Введите путь к ключу(Config.txt)");
    string keyPath = Console.ReadLine();
    configContent = File.ReadAllText(keyPath);
}
// Извлечение данных из ключа
bool isFound = false;
string numberString = "";
for (int i = 0; i < configContent.Length; i++)
```

```

{

    if (isFound == true)
    {
        numberString += configContent[i];
    }
    if (configContent[i] == '\n')
        isFound = true;
}
string[] partsConfig = numberString.Split(',');
// Опрос пользователя на наличие/утерю файлов
Console.WriteLine("Есть ли доступ к 1 файлу?(1.Да 2.Нет)");
response1 = Console.ReadLine();
if (response1 == "1")
{
    Console.WriteLine("Введите путь к файлу 1");
    file1Path = Console.ReadLine();
}
Console.WriteLine("Есть ли доступ к 2 файлу?(1.Да 2.Нет)");
response2 = Console.ReadLine();
if (response2 == "1")
{
    Console.WriteLine("Введите путь к файлу 2");
    file2Path = Console.ReadLine();
}
Console.WriteLine("Есть ли доступ к 3 файлу?(1.Да 2.Нет)");
response3 = Console.ReadLine();
if (response3 == "1")
{
    Console.WriteLine("Введите путь к файлу 3");
    file3Path = Console.ReadLine();
}
// Проверка файлов на отсутствие
if (response1 == "2")

```

```

{
    // Завершение программы если не хватает больше одного файла
    if (response2 == "2" || response3 == "2")
    {
        Console.WriteLine("Больше одного файла утеряно");
        Console.ReadLine();
        Environment.Exit(0);
    }
    Console.WriteLine("Утерян диск 1,
        производится восстановление");
    //Подгрузка данных
    byte[] data_2_final = await File
        .ReadAllBytesAsync(file2Path);
    byte[] data_3_final = await File
        .ReadAllBytesAsync(file3Path);
    // Получение оригинального размера
    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_2_final.Length - 4; i <
        data_2_final.Length; i++)
    {
        originalSizeBytes[counter] = data_2_final[i];
        counter++;
    }
    int originalSize = BitConverter
        .ToInt32(originalSizeBytes, 0);
    string extention = "";
    // Вычисление размера первой части
    int part_size_a = originalSize / 100
        * int.Parse(partsConfig[0]) / 2;
    // Вычисление размера второй части
    int part_size_b = originalSize / 100
        * int.Parse(partsConfig[1]) / 2;
    Console.WriteLine("Размер второго слайса

```

```

        данных равен {0}", part_size_b);
int part_size_c = 0;
int reduce = 0;
// Выисление лишних байт после округления
if (data_3_final[data_3_final.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math.Ceiling((originalSize
        - (part_size_a + part_size_b) * 2) / 2.0);
}
// Собрка файла с восстановлнием
int j = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < data_2.Length; i++)
{
    data_2[i] = data_2_final[j];
    j++;
}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_34_parity[i] = data_2_final[j];
    j++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_2_final[j];
    j++;
}
j = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)

```

```

{
    data_12_parity[i] = data_3_final[j];
    j++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_3_final[j];
    j++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_3_final[j];
    j++;
}
// Расчет четности
byte[] data_1 = SolveParity(data_2, data_12_parity);
byte[] data_3 = SolveParity(data_4, data_34_parity);
byte[] data = new byte[(part_size_a +
    part_size_b + part_size_c) * 2 - reduce];
j = 0;
for (int i = 0; i < data_1.Length; i++)
{
    data[j] = data_1[i];
    j++;
}
for (int i = 0; i < data_2.Length; i++)
{
    data[j] = data_2[i];
    j++;
}
for (int i = 0; i < data_3.Length; i++)
{

```

```

        data[j] = data_3[i];
        j++;
    }
    for (int i = 0; i < data_4.Length; i++)
    {
        data[j] = data_4[i];
        j++;
    }
    for (int i = 0; i < data_5.Length; i++)
    {
        data[j] = data_5[i];
        j++;
    }
    for (int i = 0; i < data_6.Length - reduce; i++)
    {
        data[j] = data_6[i];
        j++;
    }
    // Сохранение файла
    Console.WriteLine("Введите название файла для сохранения");
    cur_fileName = Console.ReadLine() + extention;
    Console.WriteLine("Введите путь для сохранения файла");
    string path = Console.ReadLine();
    await File.WriteAllBytesAsync(
        Path.Combine(path, cur_fileName), data);
    Console.WriteLine("Файл собран с восстановления");
}
else if(response2 == "2")
{
    // Завершение программы если не хватает больше одного файла
    if (response1 == "2" || response3 == "2")
    {
        Console.WriteLine("Больше одного файла утеряно");
        Console.ReadLine();
    }
}

```

```

        Environment.Exit(0);
    }
    Console.WriteLine("Утерян диск _2,
        производится восстановление");
    // Подгрузка данных
    byte[] data_1_final = await File.ReadAllBytesAsync(file1Path);
    byte[] data_3_final = await File.ReadAllBytesAsync(file3Path);
    // Получения данных об оригинальном размере
    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_1_final.Length - 4;
        i < data_1_final.Length; i++)
    {
        originalSizeBytes[counter] = data_1_final[i];
        counter++;
    }
    int originalSize = BitConverter.ToInt32(originalSizeBytes, 0);
    string extention = "";
    // Расчет размера первой части
    int part_size_a = originalSize /
        100 * int.Parse(partsConfig[0]) / 2;
    // Расчет размера второй части
    int part_size_b = originalSize /
        100 * int.Parse(partsConfig[1]) / 2;
    int part_size_c = 0;
    int reduce = 0;
    // Определение потери байт при округлении
    if (data_3_final[data_3_final.Length - 5] == 0)
        reduce = 1;
    if (part_size_a + part_size_b < originalSize)
    {
        part_size_c += (int)Math.Ceiling((originalSize
            - (part_size_a + part_size_b) * 2) / 2.0);
    }
}

```



```

// Восстановление файла
byte[] data = new byte[(part_size_a
    + part_size_b + part_size_c) * 2 - reduce];
int j = 0;

byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_1_final[j];
    j++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_1_final[j];
    j++;
}
byte[] data_56_parity = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_56_parity[i] = data_1_final[j];
    j++;
}
j = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_12_parity[i] = data_3_final[j];
    j++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_3_final[j];

```

```

        j++;
    }
    byte[] data_6 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_6[i] = data_3_final[j];
        j++;
    }
    byte[] data_2 = SolveParity(data_1, data_12_parity);
    byte[] data_5 = SolveParity(data_6, data_56_parity);
    j = 0;
    for (int i = 0; i < data_1.Length; i++)
    {
        data[j] = data_1[i];
        j++;
    }
    for (int i = 0; i < data_2.Length; i++)
    {
        data[j] = data_2[i];
        j++;
    }
    for (int i = 0; i < data_3.Length; i++)
    {
        data[j] = data_3[i];
        j++;
    }
    for (int i = 0; i < data_4.Length; i++)
    {
        data[j] = data_4[i];
        j++;
    }
    for (int i = 0; i < data_5.Length; i++)
    {
        data[j] = data_5[i];

```

```

        j++;
    }
    for (int i = 0; i < data_6.Length - reduce; i++)
    {
        data[j] = data_6[i];
        j++;
    }
    // Сохранение файла
    Console.WriteLine("Введите название файла для сохранения");
    cur_fileName = Console.ReadLine() + extention;
    Console.WriteLine("Введите путь для сохранения файла");
    string path = Console.ReadLine();
    await File.WriteAllBytesAsync(Path
        .Combine(path, cur_fileName), data);
    Console.WriteLine("Файл собран без восстановления");

} else if(response3 == "2")
{
    // Завершение программы если не хватает больше одного файла
    if (response1 == "2" || response2 == "2")
    {
        Console.WriteLine("Больше одного файла утеряно");
        Console.ReadLine();
        Environment.Exit(0);
    }
    Console.WriteLine("Утерян диск _3,
        производится восстановление");
    // Подгрузка данных
    byte[] data_1_final = await File
        .ReadAllBytesAsync(file1Path);
    byte[] data_2_final = await File
        .ReadAllBytesAsync(file2Path);
    // Получение размера оригинального файла
    byte[] originalSizeBytes = new byte[4];

```

```

int counter = 0;
for (int i = data_2_final.Length - 4;
    i < data_2_final.Length; i++)
{
    originalSizeBytes[counter] = data_2_final[i];
    counter++;
}
int originalSize = BitConverter.ToInt32(originalSizeBytes, 0);
string extention = "";
// Расчет размера первой части
int part_size_a = originalSize
    / 100 * int.Parse(partsConfig[0]) / 2;
// Расчет размера второй части
int part_size_b = originalSize
    / 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
// Определение наличия лишних байт
int reduce = 0;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math.Ceiling((originalSize
        - (part_size_a + part_size_b) * 2) / 2.0);
}
// Восстановление файла
int j = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_1_final[j];
    j++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{

```

```

        data_3[i] = data_1_final[j];
        j++;
    }
    byte[] data_56_parity = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_56_parity[i] = data_1_final[j];
        j++;
    }
    j = 0;
    byte[] data_2 = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_2[i] = data_2_final[j];
        j++;
    }
    byte[] data_34_parity = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_34_parity[i] = data_2_final[j];
        j++;
    }
    byte[] data_5 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_5[i] = data_2_final[j];
        j++;
    }
    // Расчет четности
    byte[] data_4 = SolveParity(data_3, data_34_parity);
    byte[] data_6 = SolveParity(data_5, data_56_parity);
    if (data_6[data_6.Length - 1] == 0)
        reduce = 1;
    byte[] data = new byte[(part_size_a

```

```

        + part_size_b + part_size_c) * 2 - reduce];
j = 0;
for (int i = 0; i < data_1.Length; i++)
{
    data[j] = data_1[i];
    j++;
}
for (int i = 0; i < data_2.Length; i++)
{
    data[j] = data_2[i];
    j++;
}
for (int i = 0; i < data_3.Length; i++)
{
    data[j] = data_3[i];
    j++;
}
for (int i = 0; i < data_4.Length; i++)
{
    data[j] = data_4[i];
    j++;
}
for (int i = 0; i < data_5.Length; i++)
{
    data[j] = data_5[i];
    j++;
}
for (int i = 0; i < data_6.Length - reduce; i++)
{
    data[j] = data_6[i];
    j++;
}
// Сохранение файла
Console.WriteLine("Введите название файла для сохранения");

```

```

    cur_fileName = Console.ReadLine() + extention;
    Console.WriteLine("Введите путь для сохранения файла");
    string path = Console.ReadLine();
    await File.WriteAllBytesAsync(
        Path.Combine(path, cur_fileName), data);
    Console.WriteLine("Файл собран без восстановления");

} else if (response1 == "1" && response2 == "1" && response3 == "1")
{
    Console.WriteLine("Все файлы на месте, производится сборка ");
    // Подгрузка данных
    byte[] data_1_final = await File.ReadAllBytesAsync(file1Path);
    byte[] data_2_final = await File.ReadAllBytesAsync(file2Path);
    byte[] data_3_final = await File.ReadAllBytesAsync(file3Path);
    // Расчет размера изначального файла
    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_2_final.Length - 4;
        i < data_2_final.Length; i++)
    {
        originalSizeBytes[counter] = data_2_final[i];
        counter++;
    }
    int originalSize = BitConverter.ToInt32(originalSizeBytes, 0);
    string extention = "";
    // Расчет размера первой части
    int part_size_a = originalSize /
        100 * int.Parse(partsConfig[0]) / 2;
    // Расчет размера второй части
    int part_size_b = originalSize /
        100 * int.Parse(partsConfig[1]) / 2;
    int part_size_c = 0;
    int reduce = 0;
    // Вычисление наличия лишних байтов

```

```

if (data_3_final[data_3_final.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math.Ceiling((originalSize
        - (part_size_a + part_size_b) * 2) / 2.0);
}
byte[] data = new byte[(part_size_a +
    part_size_b + part_size_c) * 2 - reduce];
// Сборка файла из частей
int j = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_1_final[j];
    j++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_1_final[j];
    j++;
}
j = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_2_final[j];
    j++;
}
byte[] par_data_2 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    par_data_2[i] = data_2_final[j];

```



```

        j++;
    }
    byte[] data_5 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_5[i] = data_2_final[j];
        j++;
    }
    j = 0;
    byte[] par_data_3 = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        par_data_3[i] = data_3_final[j];
        j++;
    }
    byte[] data_4 = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_4[i] = data_3_final[j];
        j++;
    }
    byte[] data_6 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_6[i] = data_3_final[j];
        j++;
    }
    j = 0;
    for(int i = 0; i < data_1.Length; i++)
    {
        data[j] = data_1[i];
        j++;
    }
    for (int i = 0; i < data_2.Length; i++)

```

```

{
    data[j] = data_2[i];
    j++;
}
for (int i = 0; i < data_3.Length; i++)
{
    data[j] = data_3[i];
    j++;
}
for (int i = 0; i < data_4.Length; i++)
{
    data[j] = data_4[i];
    j++;
}
for (int i = 0; i < data_5.Length; i++)
{
    data[j] = data_5[i];
    j++;
}
for (int i = 0; i < data_6.Length - reduce; i++)
{
    data[j] = data_6[i];
    j++;
}
// Сохранение файла
Console.WriteLine("Введите название файла для сохранения");
cur_fileName = Console.ReadLine() + extention;
Console.WriteLine("Введите путь для сохранения файла");
string path = Console.ReadLine();
await File.WriteAllBytesAsync(Path
    .Combine(path, cur_fileName), data);
Console.WriteLine("Файл собран без восстановления");
}

```

```
Console.WriteLine("Работа программы завершена");  
Console.ReadLine();  
// Метод расчета четности  
byte[] SolveParity(byte[] data_1, byte[] data_2)  
{  
    byte[] parity = new byte[data_1.Length];  
    for (int i = 0; i < parity.Length; i++)  
        parity[i] = (byte)(data_1[i] ^ data_2[i]);  
    return parity;  
}
```

## ПРИЛОЖЕНИЕ В

### Исходный код Program

```
using Microsoft.OpenApi.Models;
using vkr;

// Создание билдера приложения
var builder = WebApplication.CreateBuilder(args);

// Добавление сервисов для работы с MVC
builder.Services.AddControllersWithViews();
// Настройка генерации Swagger документации
builder.Services.AddSwaggerGen(options =>
{
    // Определение информации о API
    options.SwaggerDoc("v1", new OpenApiInfo
    {
        Version = "v1",           // Версия API
        Title = "RAID5Clouds",    // Название API
        Description = "Vkr",      // Описание API
    });
});
// Настройка политики CORS
builder.Services.AddCors(options =>
{
    // Создание политики с именем "AllowVueApp"
    options.AddPolicy("AllowVueApp",
        policy => policy.WithOrigins("http://localhost:5173")
        // Разрешенный источник
        .AllowAnyHeader()
        // Разрешение любых заголовков
        .AllowAnyMethod());
    // Разрешение любых HTTP-методов
});
```

```
// Вызов метода конфигурации сервисов из класса Startup
Startup.ConfigureServices(builder.Services);
// Сборка приложения
var app = builder.Build();

// Проверка среды выполнения
if (!app.Environment.IsDevelopment())
{
    // В production: использование страницы ошибок
    app.UseExceptionHandler("/Home/Error");
    // Включение HTTP Strict Transport Security
    app.UseHsts();
}
else
{
    // В development: включение Swagger middleware
    app.UseSwagger();
    // Настройка UI Swagger
    app.UseSwaggerUI(c => c.SwaggerEndpoint
        ("/swagger/v1/swagger.json", "vkr"));
}

// Перенаправление HTTP-запросов на HTTPS
app.UseHttpsRedirection();
// Поддержка статических файлов
app.UseStaticFiles();

// Применение политики CORS
app.UseCors("AllowVueApp");

// Включение маршрутизации
app.UseRouting();

// Включение авторизации
```

```
app.UseAuthorization();

// Настройка маршрутов по умолчанию
app.MapControllerRoute(
    name: "default", // Имя маршрута
    pattern: "{controller=Home}/{action=Index}/{id?}");
// Шаблон URL

// Запуск приложения
app.Run();
```

## ПРИЛОЖЕНИЕ Г

### Исходный код Startup

```
using vkr.Services;
namespace vkr
{
    // Статический класс для настройки сервисов приложения
    public static class Startup
    {
        // Метод для регистрации сервисов в DI-контейнере
        public static void ConfigureServices
            (IServiceCollection services)
        {
            // Добавление контроллеров с поддержкой сериализации
            //JSON
            services.AddControllers().AddNewtonsoftJson();

            // Регистрация HttpClient для выполнения HTTP-запросов
            services.AddHttpClient();

            // Регистрация сервиса для работы с Яндекс.Диском
            services.AddSingleton<YandexDiskService>();

            // Регистрация сервиса для работы с Dropbox
            services.AddSingleton<DropboxService>();

            // Регистрация сервиса для работы с Google Drive
            services.AddSingleton<GoogleDriveService>();

            // Регистрация сервиса RAID5
            services.AddSingleton<RAID5Service>();
        }
    }
}
```

## ПРИЛОЖЕНИЕ Д

### Исходный код YandexDiskService

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Extensions.Configuration;
using Newtonsoft.Json;

namespace vkr.Services
{
    /// <summary>
    /// Сервис для работы с Яндекс диском
    /// Позволяет загружать и скачивать файлы
    /// </summary>
    public class YandexDiskService
    {
        private readonly HttpClient _httpClient;
        // клиент для отправки запросов
        private readonly string _apiUrl; // URL API Яндекс диска
        private readonly string _oAuthToken; // Токен авторизации
        /// <summary>
        /// Инициализация экземпляра класса
        /// </summary>
        /// <param name="configuration">Конфигурация,
        /// содержащая настройки для работы с API</param>
        public YandexDiskService(IConfiguration configuration)
        {
            _httpClient = new HttpClient();
            _apiUrl = configuration["YandexDisk:ApiUrl"];
            // URL API из конфигурации
            _oAuthToken = configuration["YandexDisk:OAuthToken"];
            // Токен из конфигурации
            _httpClient.DefaultRequestHeaders.Authorization = new
```



```

        AuthenticationHeaderValue("OAuth", _oauthToken);
        // Заголовок авторизации
    }
    /// <summary>
    /// Загрузка файла на Яндекс диск
    /// </summary>
    /// <param name="filePath">Путь к файлу на Яндекс диске
    </param>
    /// <param name="localFilePath">
    /// Содержимое файла в виде массива байтов</param>
    /// <returns>Результат загрузки</returns>
    public async Task<string> UploadFile
    (string filePath, string localFilePath)
    {
        var uploadUrlResponse = await _httpClient.GetAsync
        ($"{_apiUrl}resources/upload?path={filePath}
        &overwrite=true&fields=name,_embedded.items.path");
        // URL для загрузки файла
        uploadUrlResponse.EnsureSuccessStatusCode();
        // Статус запроса

        // Десериализация ответа с адресом для загрузки
        var uploadUrl = JsonConvert.DeserializeObject
        <YandexDiskUploadResponse>(await
        uploadUrlResponse.Content.ReadAsStringAsync()).Href;

        byte[] fileContent = File.ReadAllBytes(localFilePath);

        var content = new ByteArrayContent(fileContent);
        // Загрузка файла по полученному URL
        var uploadResponse = await _httpClient
        .PutAsync(uploadUrl, content);
        uploadResponse.EnsureSuccessStatusCode();
        // Проверка, что загрузка прошла успешно
    }

```

```

        return await uploadResponse.Content.ReadAsStringAsync();
        // Результат загрузки
    }

    /// <summary>
    /// Скачивание файла с Яндекс диска
    /// </summary>
    /// <param name="filePath">Путь к файлу на яндекс диске
    </param>
    /// <returns>Содержимое файла в виде массива байтов
    </returns>
    public async Task<byte[]> DownloadFile(string filePath)
    {

        var downloadUrlResponse = await _httpClient
            .GetAsync($"{_apiUrl}resources/download?path=
            {filePath}");
        // URL для загрузки файла
        downloadUrlResponse.EnsureSuccessStatusCode();
        // Статус запроса
        // Десериализация ответа с адресом для скачивания
        var downloadUrl = JsonConvert
            .DeserializeObject<YandexDiskDownloadResponse>
            (await downloadUrlResponse.Content
            .ReadAsStringAsync()).Href;

        var downloadResponse = await _httpClient
            .GetAsync(downloadUrl);
        // Скачивание файла по полученному URL
        downloadResponse.EnsureSuccessStatusCode();
        // Проверка результата запроса

        return await downloadResponse.Content
            .ReadAsByteArrayAsync();
    }

```

```

        // Содержимое файла
    }
    /// <summary>
    /// Класс для десериализации ответа API
    ///при получении URL для загрузки файла
    /// </summary>
    private class YandexDiskUploadResponse
    {
        public string Href { get; set; }
        // URL для загрузки файла
    }
    /// <summary>
    /// Класс для десериализации ответа API
    ///при получении URL для скачивания файла
    /// </summary>
    private class YandexDiskDownloadResponse
    {
        public string Href { get; set; }
        // URL для скачивания файла
    }
}
}

```

## ПРИЛОЖЕНИЕ Е

### Исходный код DropboxService

```
using System.Net.Http.Headers;
using Newtonsoft.Json;

namespace vkr.Services
{
    public class DropboxService
    {
        // HTTP-клиент для отправки запросов к Dropbox API
        private readonly HttpClient _httpClient;
        // Токен доступа для авторизации в Dropbox
        private readonly string _accessToken;
        // Базовый URL Dropbox API
        private readonly string _apiUrl;

        // Конструктор: инициализация сервиса с конфигурацией
        public DropboxService(IConfiguration configuration)
        {
            // Создание нового экземпляра HttpClient
            _httpClient = new HttpClient();
            // Получение токена доступа из конфигурации
            _accessToken = configuration["Dropbox:AccessToken"];
            // Получение базового URL API из конфигурации
            _apiUrl = configuration["Dropbox:ApiUrl"];
            // Установка заголовка авторизации (Bearer-токен)
            _httpClient.DefaultRequestHeaders.Authorization =
                new AuthenticationHeaderValue
                    ("Bearer", _accessToken);
        }

        // Метод для загрузки файла в Dropbox
        public async Task<string> UploadFile
            (string localFilePath, string dropboxPath)
```

```

{
    // Проверка существования локального файла
    if (!File.Exists(localFilePath))
    {
        throw new FileNotFoundException
            ("Файл не найден", localFilePath);
    }

    // Чтение содержимого файла в массив байтов
    byte[] fileContent = File.ReadAllBytes(localFilePath);

    // Создание контента запроса из байтов файла
    using (var content = new ByteArrayContent(fileContent))
    {
        // Установка типа контента
        content.Headers.ContentType =
            new MediaTypeHeaderValue
                ("application/octet-stream");

        // Создание HTTP-запроса
        var request = new HttpRequestMessage
            (HttpMethod.Post, $"{_apiUrl}upload");

        // Добавление специального заголовка
        request.Headers.Add("Dropbox-API-Arg",
            JsonConvert.SerializeObject(new
            {
                // Путь в Dropbox
                path = dropboxPath,
                // Режим перезаписи
                mode = "overwrite",
                // Автопереименование при конфликте
                autorename = true,
                // Уведомлять о действии
            }
        ));
    }
}

```

```

        mute = false
    }));

    // Установка контента запроса
    request.Content = content;

    // Отправка запроса и получение ответа
    var response = await _httpClient
        .SendAsync(request);
    // Проверка на успешность запроса
    response.EnsureSuccessStatusCode();

    // Возврат ответа в виде строки
    return await response.Content
        .ReadAsStringAsync();
}

}

// Метод для скачивания файла из Dropbox
public async Task<byte[]> DownloadFile(string dropboxPath)
{
    // Создание HTTP-запроса
    var request = new HttpRequestMessage
        (HttpMethod.Post, $"{_apiUrl}download");

    // Добавление заголовка с метаданными запроса
    request.Headers.Add("Dropbox-API-Arg",
        JsonConvert.SerializeObject(new
        {
            // Путь к файлу в Dropbox
            path = dropboxPath
        }));

    // Отправка запроса и получение ответа

```

```

var response = await _httpClient.SendAsync(request);
// Проверка на успешность запроса
response.EnsureSuccessStatusCode();

// Десериализация метаданных файла из заголовка ответа
var result = JsonConvert.DeserializeObject
    <DropboxFileMetadata>(
        response.Headers
            .GetValues("Dropbox-API-Result").First());

// Возврат содержимого файла в виде массива байтов
return await response.Content.ReadAsByteArrayAsync();
}

// Вложенный класс для десериализации метаданных
private class DropboxFileMetadata
{
    // Имя файла
    public string name { get; set; }
    // Нижний регистр пути
    public string path_lower { get; set; }
    // Размер файла в байтах
    public long size { get; set; }
}
}
}

```

## ПРИЛОЖЕНИЕ Ж

### Исходный код GoogleDriveService

```
using System.Net.Http.Headers;
using System.Text.Json;
using System.Text;
using Google.Apis.Auth.OAuth2;
using Google.Apis.Drive.v3;
using Google.Apis.Services;
using Microsoft.AspNetCore.Http.HttpResults;
using System.Net;
using Google.Apis.Upload;
using System;

namespace vkr.Services
{
    public class GoogleDriveService
    {
        // Путь к файлу с учетными данными
        private readonly string credentialsPath;
        // ID папки в Google Drive
        private readonly string folderId;

        // Конструктор: инициализация сервиса
        public GoogleDriveService(IConfiguration configuration)
        {
            // Файл с OAuth2 credentials
            credentialsPath = "credentials.json";
            // ID целевой папки
            folderId = "1sxHqWNxEft-5gVvqM_WtdUG1ZQaPFsiB";
        }

        // Метод для загрузки файла в Google Drive
        public async Task<string> UploadFile(string localFilePath)
        {
```



```

// Проверка существования файла
if (!File.Exists(localFilePath))
    throw new FileNotFoundException
        ("File not found", localFilePath);
// Получение имени файла
var fileName = Path.GetFileName(localFilePath);
// Чтение файла
var fileContent = await File
    .ReadAllBytesAsync(localFilePath);

// Аутентификация в Google API
UserCredential credentials;
var clientSecrets = await GoogleClientSecrets
    .FromFileAsync(credentialsPath);

credentials = await GoogleWebAuthorizationBroker
    .AuthorizeAsync(
        clientSecrets.Secrets,
        // Запрашиваемые разрешения
        new[] { DriveService.ScopeConstants.DriveFile },
        "user",
        CancellationToken.None);

// Создание сервиса Google Drive
var service = new DriveService(new
    BaseClientService.Initializer()
{
    // Учетные данные
    HttpClientInitializer = credentials,
    // Имя приложения
    ApplicationName = "Raid5"
});

// Метаданные файла для загрузки

```

```

var fileMetaData = new Google.Apis.Drive.v3.Data.File()
{
    // Имя файла
    Name = Path.GetFileName(localFilePath),
    // Родительская папка
    Parents = new List<string> { folderId }
};

FilesResource.CreateMediaUpload request;

// Загрузка файла
using (var stream = new FileStream
    (localFilePath, FileMode.Open))
{
    request = service.Files.Create
        (fileMetaData, stream, "");
    request.Fields = "id"; // Запрашиваем ID файла
    // Асинхронная загрузка
    var progress = await request.UploadAsync();

    if (progress.Status != UploadStatus.Completed)
        throw new Exception($"Upload failed:
            {progress.Exception?.Message}");
}
// Возвращаем имя загруженного файла
return request.ResponseBody.Name;
}

// Метод для скачивания файла из Google Drive
public async Task DownloadFile
    (string fileName, string localSavePath)
{
    // Аутентификация (аналогично UploadFile)
    UserCredential credentials;

```

```

var clientSecrets = await GoogleClientSecrets
    .FromFileAsync(credentialsPath);

credentials = await GoogleWebAuthorizationBroker
    .AuthorizeAsync(
        clientSecrets.Secrets,
        new[] { DriveService.ScopeConstants.DriveFile },
        "user",
        CancellationToken.None);

// Создание сервиса
var service = new DriveService
    (new BaseClientService.Initializer()
    {
        HttpClientInitializer = credentials,
        ApplicationName = "Raid5"
    });

// Поиск файла по имени
var fileId = await FindFileIdByName(fileName, service);
var fileInfo = await service.Files
    .Get(fileId).ExecuteAsync();

if (fileInfo == null)
    throw new Exception
        ("File not found in Google Drive");

// Скачивание файла
using (var fileStream = new FileStream
    (localSavePath, FileMode.Create, FileAccess.Write))
{
    var request = service.Files.Get(fileId);
    // Асинхронное скачивание
    await request.DownloadAsync(fileStream);
}

```

```

    }
}

// Метод для получения списка файлов в папке
public async Task<List<string>> GetFileNamesInFolder()
{
    // Аутентификация
    UserCredential credentials;
    var clientSecrets = await GoogleClientSecrets
        .FromFileAsync(credentialsPath);

    credentials = await GoogleWebAuthorizationBroker
        .AuthorizeAsync(
            clientSecrets.Secrets,
            new[] { DriveService.ScopeConstants.DriveFile },
            "user",
            CancellationToken.None);

    // Создание сервиса
    var service = new DriveService
        (new BaseClientService.Initializer()
        {
            HttpClientInitializer = credentials,
            ApplicationName = "Raid5"
        });

    // Запрос списка файлов
    var request = service.Files.List();
    // Фильтр: файлы в папке, не в корзине
    request.Q =
        $"{folderId}' in parents and trashed = false";
    // Запрашиваем только имена файлов
    request.Fields = "files(name)";
}

```

```

        var result = await request.ExecuteAsync();
        // Возвращаем список имен
        return result.Files.Select(file => file.Name).ToList();
    }

    // Вспомогательный метод для поиска файла по имени
    public async Task<string> FindFileIdByName
    (string fileName, DriveService service)
    {
        var request = service.Files.List();
        // Фильтр по имени
        request.Q =
            $"name = '{fileName}' and trashed = false";
        // Запрашиваем ID и имя
        request.Fields = "files(id, name)";

        var result = await request.ExecuteAsync();
        // Возвращаем ID первого найденного файла
        return result.Files.FirstOrDefault()?.Id;
    }
}
}

```

## ПРИЛОЖЕНИЕ 3

### Исходный код Raid5Service

```
using Microsoft.AspNetCore.DataProtection.KeyManagement;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using vkr.Controllers;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace vkr.Services
{
    /// <summary>
    /// Сервис для реализации RAID 5
    /// с распределением информации и
    /// четности по разным облачным хранилищам
    /// </summary>
    public class RAID5Service
    {
        // Сервисы для работы с различными облачными хранилищами
        private readonly DropboxService _dropboxService;
        private readonly GoogleDriveService _googledriveService;
        private readonly YandexDiskService _yandexdiskService;

        /// <summary>
        /// Конструктор сервиса, принимающий зависимости
        /// для работы с облачными хранилищами
        /// </summary>
        public RAID5Service(DropboxService dropboxService,
            GoogleDriveService googledriveService,
            YandexDiskService yandexdiskService)
        {
            _dropboxService = dropboxService;
            _googledriveService = googledriveService;
            _yandexdiskService = yandexdiskService;
        }
    }
}
```

```

}

/// <summary>
/// Метод для записи данных с использованием RAID 5
/// </summary>
/// <param name="fileContent">Байтовый массив с
///     содержимым файла</param>
/// <param name="origFileName">
/// Исходное имя файла</param>
/// <param name="option">Вариант
/// распределения данных по хранилищам (1-3)</param>
/// <param name="key">Ключ с конфигурацией
/// распределения данных</param>
/// <returns>Результат операции (
/// "Ok" при успехе или сообщение об ошибке)</returns>
public async Task<string> WriteData(byte[]
    fileContent, string origFileName, int
    option, string key)
{
    // Разбиваем ключ на части для определения
    // пропорций распределения данных
    string[] partsConfig = key.Split(',');
    int part_portional_a = int.Parse(partsConfig[0]);
    int part_portional_b = int.Parse(partsConfig[1]);

    // Получаем исходный размер файла
    //и вычисляем размеры частей
    byte[] originalSize = BitConverter
        .GetBytes(fileContent.Length);
    int part_size_a = fileContent.Length / 100
        * part_portional_a / 2;
    int part_size_b = fileContent.Length / 100
        * part_portional_b / 2;
    int part_size_c = 0;

```

```

string response = "Ok";

// Если суммарный размер частей А и В меньше
// исходного файла, вычисляем размер части С
if (part_size_a + part_size_b < fileContent.Length)
{
    part_size_c += (int)Math.Ceiling((fileContent.Length
        - (part_size_a + part_size_b) * 2) / 2.0);
}

// Коррекция для нечетных размеров файла
int reduce = 0;
if (fileContent.Length % 2 != 0)
{
    reduce = 1;
}

// Формируем строку с метаданными о
// размерах частей и расширении файла
string extension = "";
string combined = part_size_a.ToString() +
    "!" + part_size_b.ToString() + "!" +
    part_size_c.ToString() + "!"
    + extension + "!" + reduce;
byte[] byteArray = Encoding.UTF8.GetBytes(combined);

// Разделение исходных данных на 6 частей
int j = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    if (j < fileContent.Length)
    {

```



```

        data_1[i] = fileContent[j];
        j++;
    }
    else
    {
        data_1[i] = 0;
        // Заполнение нулями если данные закончились
        j++;
    }
}

```

```

byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    if (j < fileContent.Length)
    {
        data_2[i] = fileContent[j];
        j++;
    }
    else
    {
        data_2[i] = 0;
        j++;
    }
}

```

```

byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    if (j < fileContent.Length)
    {
        data_3[i] = fileContent[j];
        j++;
    }
}

```

```

        else
        {
            data_3[i] = 0;
            j++;
        }
    }

    byte[] data_4 = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        if (j < fileContent.Length)
        {
            data_4[i] = fileContent[j];
            j++;
        }
        else
        {
            data_4[i] = 0;
            j++;
        }
    }

    byte[] data_5 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        if (j < fileContent.Length)
        {
            data_5[i] = fileContent[j];
            j++;
        }
        else
        {
            data_5[i] = 0;
            j++;
        }
    }

```

```

    }
}

byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    if (j < fileContent.Length)
    {
        data_6[i] = fileContent[j];
        j++;
    }
    else
    {
        data_6[i] = 0;
        j++;
    }
}

// Формирование блока A: данные 1,
// данные 3, четность 5-6, исходный размер
j = 0;
byte[] data_a = new byte[part_size_a +
    part_size_b + part_size_c + 4];
for (int i = 0; i < part_size_a; i++)
{
    data_a[j] = data_1[i];
    j++;
}
for (int i = 0; i < part_size_b; i++)
{
    data_a[j] = data_3[i];
    j++;
}
byte[] data_56_parity = SolveParity(data_5, data_6);

```

```

for (int i = 0; i < data_56_parity.Length; i++)
{
    data_a[j] = data_56_parity[i];
    j++;
}
for (int i = 0; i < 4; i++)
{
    data_a[j] = originalSize[i];
    j++;
}

// Формирование блока В: данные 2,
// четность 3-4, данные 5, исходный размер
j = 0;
byte[] data_b = new byte[part_size_a +
    part_size_b + part_size_c + 4];
for (int i = 0; i < part_size_a; i++)
{
    data_b[j] = data_2[i];
    j++;
}
byte[] data_34_parity = SolveParity(data_3, data_4);

for (int i = 0; i < data_34_parity.Length; i++)
{
    data_b[j] = data_34_parity[i];
    j++;
}
for (int i = 0; i < data_5.Length; i++)
{
    data_b[j] = data_5[i];
    j++;
}

```

```

for (int i = 0; i < 4; i++)
{
    data_b[j] = originalSize[i];
    j++;
}

// Формирование блока четности: четность 1-2,
// данные 4, данные 6, исходный размер
j = 0;
byte[] data_parity = new byte[part_size_a +
    part_size_b + part_size_c + 4];
byte[] parity = SolveParity(data_1, data_2);
for (int i = 0; i < part_size_a; i++)
{
    data_parity[j] = parity[i];
    j++;
}
for (int i = 0; i < data_4.Length; i++)
{
    data_parity[j] = data_4[i];
    j++;
}

for (int i = 0; i < data_6.Length; i++)
{
    data_parity[j] = data_6[i];
    j++;
}

for (int i = 0; i < 4; i++)
{
    data_parity[j] = originalSize[i];
    j++;
}

```

```

// Запись данных в облачные хранилища в
// зависимости от выбранного варианта
string dir = Directory.GetCurrentDirectory();
if (option == 1)
{
    try
    {
        // Вариант 1:
        // A -> Яндекс.Диск
        // B -> Dropbox
        // Четность -> Google Drive
        string fileName = Path
            .GetFileNameWithoutExtension(origFileName);
        string cur_fileName = fileName + "_A";
        await File.WriteAllBytesAsync
            (Path.Combine(dir, cur_fileName), data_a);

        await _yandexdiskService
            .UploadFile(cur_fileName,
                Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(dir, cur_fileName));

        cur_fileName = fileName + "_B";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_b);

        await _dropboxService.UploadFile(
            Path.Combine(dir, cur_fileName),
                "/" + cur_fileName);
        File.Delete(Path.Combine(dir, cur_fileName));

        cur_fileName = fileName + "_C";
        await File.WriteAllBytesAsync(

```

```

        Path.Combine(dir, cur_fileName), data_parity);

        await _googledriveService.UploadFile(
            Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(dir, cur_fileName));

        return response;
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
else if (option == 2)
{
    try
    {
        // Вариант 2:
        // A -> Google Drive
        // B -> Dropbox
        // C -> Яндекс.Диск
        string fileName = Path
            .GetFileNameWithoutExtension(origFileName);
        string cur_fileName = fileName + "_A";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_a);

        await _googledriveService
            .UploadFile(Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(dir, cur_fileName));

        cur_fileName = fileName + "_B";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_b);
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
}

```

```

        await _dropboxService.UploadFile(
            Path.Combine(dir, cur_fileName), "/"
                + cur_fileName);
        File.Delete(Path.Combine(dir, cur_fileName));

        cur_fileName = fileName + "_C";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_parity);

        await _yandexdiskService
            .UploadFile(cur_fileName,
                Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(
            dir, cur_fileName));

        return response;
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
else if (option == 3)
{
    try
    {
        // Вариант 3:
        // A -> Google Drive
        // B -> Яндекс.Диск
        // Четность -> Dropbox
        string fileName = Path
            .GetFileNameWithoutExtension(origFileName);
        string cur_fileName = fileName + "_A";
    }
    catch { }
}

```



```

        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_a);

        await _googledriveService
            .UploadFile(Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(dir, cur_fileName));

        cur_fileName = fileName + "_B";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_b);

        await _yandexdiskService
            .UploadFile(cur_fileName,
                Path.Combine(dir, cur_fileName));
        File.Delete(Path.Combine(
            dir, cur_fileName));

        cur_fileName = fileName + "_C";
        await File.WriteAllBytesAsync(
            Path.Combine(dir, cur_fileName), data_parity);

        await _dropboxService.UploadFile(
            Path.Combine(dir, cur_fileName),
            "/" + cur_fileName);
        File.Delete(Path.Combine(dir, cur_fileName));

        return response;
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}
return response;

```

```

}

///  

/// Метод для чтения данных, распределенных по RAID 5  

///  

public async Task<byte[]> ReadData(  

string fileName, int option, string key)  

{  

    string curFileName = fileName;  
  

    if (option == 1)  

    {  

        // Вариант 1:  

        // А - Яндекс.Диск  

        // В - Dropbox  

        // Четность - Google Drive  

        curFileName = fileName + "_A";  

        byte[] data_a = await _yandexdiskService  

            .DownloadFile(curFileName);  

        curFileName = fileName + "_B";  

        byte[] data_b = await _dropboxService  

            .DownloadFile("/") + curFileName);  

        curFileName = fileName + "_C";  

        await _googledriveService.DownloadFile(  

curFileName, Directory  

.GetCurrentDirectory() + "/" + curFileName);  

        byte[] data_parity = File  

            .ReadAllBytes(Directory  

.GetCurrentDirectory() + "/" + curFileName);  
  

        // Разбираем ключ для  

        // получения конфигурации разделения  

        string[] partsConfig = key.Split(',');

```

```

// Получаем исходный размер
// файла из последних 4 байт блока B
byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_b.Length - 4;
     i < data_b.Length; i++)
{
    originalSizeBytes[counter] = data_b[i];
    counter++;
}
int originalSize = BitConverter
    .ToInt32(originalSizeBytes, 0);

// Вычисляем размеры частей на основе ключа
int part_size_a = originalSize /
    100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize /
    100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math.Ceiling((
        originalSize - (part_size_a
        + part_size_b) * 2) / 2.0);
}

// Извлекаем данные из блока A (данные 1 и 3)
counter = 0;
byte[] data_1 = new byte[part_size_a];
for(int i = 0; i < part_size_a; i++)
{

```

```

        data_1[i] = data_a[counter];
        counter++;
    }
    byte[] data_3 = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_3[i] = data_a[counter];
        counter++;
    }

    // Извлекаем данные из блока B
    counter = 0;
    byte[] data_2 = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_2[i] = data_b[counter];
        counter++;
    }
    byte[] data_par = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_par[i] = data_b[counter];
        counter++;
    }
    byte[] data_5 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_5[i] = data_b[counter];
        counter++;
    }

    // Извлекаем данные из блока четности
    counter = 0;
    byte[] data_par_a = new byte[part_size_a];

```

```

for (int i = 0; i < part_size_a; i++)
{
    data_par_a[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

// Собираем исходные данные из всех частей
byte[] data = new byte[(part_size_a
    + part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for(int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{

```

```

        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }

    return data;

} else if(option == 2)
{
    // Вариант 2:
    // A - Google Drive
    // B - Dropbox
    // Четность - Яндекс.Диск
    curFileName = fileName + "_A";
    await _googledriveService
        .DownloadFile(curFileName,
            Directory.GetCurrentDirectory()
            + "/" + curFileName);
    byte[] data_a = File.ReadAllBytes(
        Directory.GetCurrentDirectory()

```

```

+ "/" + curFileName);
curFileName = fileName + "_B";
byte[] data_b = await _dropboxService
.DownloadFile("/") + curFileName);
curFileName = fileName + "_C";
byte[] data_parity = await
_yandexdiskService.DownloadFile(curFileName);

string[] partsConfig = key.Split(',');

byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_b.Length
- 4; i < data_b.Length; i++)
{
    originalSizeBytes[counter] = data_b[i];
    counter++;
}
int originalSize = BitConverter
.ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
.Ceiling((originalSize -
(part_size_a + part_size_b) * 2) / 2.0);
}

```

```

counter = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_a[counter];
    counter++;
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_par = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_par[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}

```



```

counter = 0;
byte[] data_par_a = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_par_a[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)

```

```

    {
        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }

    return data;

} else if (option == 3)
{
    // Вариант 3:
    // A - Google Drive
    // B - Яндекс.Диск
    // C - Dropbox
    curFileName = fileName + "_A";
    await _googledriveService.DownloadFile(
    curFileName, Directory
    .GetCurrentDirectory() + "/" + curFileName);
    byte[] data_a = File.ReadAllBytes(
    Directory.GetCurrentDirectory()

```

```

+ "/" + curFileName);
curFileName = fileName + "_B";
byte[] data_b = await _yandexdiskService
.DownloadFile(curFileName);
curFileName = fileName + "_C";
byte[] data_parity = await _dropboxService
.DownloadFile("/") + curFileName);

string[] partsConfig = key.Split(',');

byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_b.Length - 4;
i < data_b.Length; i++)
{
    originalSizeBytes[counter] = data_b[i];
    counter++;
}
int originalSize = BitConverter
.ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[
    data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
        .Ceiling((originalSize - (
        part_size_a + part_size_b) * 2) / 2.0);

```

```
}
```

```
counter = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_a; i++)
{
    data_3[i] = data_a[counter];
    counter++;
}
```

```
counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_par = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_par[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}
```

```

}
counter = 0;
byte[] data_par_a = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_par_a[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

byte[] data = new byte[(part_size_a +
part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}

```

```

        for (int i = 0; i < part_size_b; i++)
        {
            data[counter] = data_3[i];
            counter++;
        }
        for (int i = 0; i < part_size_b; i++)
        {
            data[counter] = data_4[i];
            counter++;
        }
        for (int i = 0; i < part_size_c; i++)
        {
            data[counter] = data_5[i];
            counter++;
        }
        for (int i = 0; i < part_size_c - reduce; i++)
        {
            data[counter] = data_6[i];
            counter++;
        }

        return data;
    }
    byte[] datsa = new byte[1];
    return datsa;
}

/// <summary>
/// Метод для чтения данных при недоступности Яндекс.Диска
/// </summary>
public async Task<byte[]> ReadDataWithoutYandex(
    string fileName,
    int option, string key)
{

```

```

string curFileName = fileName;
if (option == 1)
{
    // Вариант 1 без Яндекс.Диска:
    // Используем B (Dropbox) и C (Google Drive)
    curFileName = fileName + "_B";
    byte[] data_b = await _dropboxService
        .DownloadFile("/") + curFileName);
    curFileName = fileName + "_C";
    await _googledriveService.DownloadFile(
        curFileName, Directory.GetCurrentDirectory()
        + "/" + curFileName);
    byte[] data_parity = File.ReadAllBytes(
        Directory.GetCurrentDirectory() +
        "/" + curFileName);

    string[] partsConfig = key.Split(',');

    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_b.Length - 4;
        i < data_b.Length; i++)
    {
        originalSizeBytes[counter]
            = data_b[i];
        counter++;
    }
    int originalSize = BitConverter
        .ToInt32(originalSizeBytes, 0);
    int part_size_a = originalSize
        / 100 * int.Parse(partsConfig[0]) / 2;
    int part_size_b = originalSize
        / 100 * int.Parse(partsConfig[1]) / 2;
    int part_size_c = 0;

```

```

int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
        .Ceiling((originalSize - (
            part_size_a + part_size_b) * 2) / 2.0);
}

// Извлекаем данные из блока B
counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_34_parity[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}

// Извлекаем данные из блока четности
counter = 0;
byte[] data_12_parity = new byte[part_size_a];

```



```

for (int i = 0; i < part_size_a; i++)
{
    data_12_parity[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

// Восстанавливаем недостающие
// данные с помощью четности
byte[] data_1 = SolveParity(data_12_parity, data_2);
byte[] data_3 = SolveParity(data_34_parity, data_4);

// Собираем исходные данные
byte[] data = new byte[(part_size_a
    + part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{

```

```

        data[counter] = data_2[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }
    return data;
} else if(option == 2)
{
    // Вариант 2 без Яндекс.Диска:
    // Используем A (Google Drive) и B (Dropbox)
    curFileName = fileName + "_A";
    await _googledriveService.DownloadFile(
        curFileName, Directory.
            GetCurrentDirectory() + "/" + curFileName);
    byte[] data_a = File.ReadAllBytes(
        Directory.GetCurrentDirectory()

```

```

+ "/" + curFileName);
curFileName = fileName + "_B";
byte[] data_b = await _dropboxService
    .DownloadFile("/") + curFileName);

string[] partsConfig = key.Split(',');

byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_b.Length
    - 4; i < data_b.Length; i++)
{
    originalSizeBytes[counter] = data_b[i];
    counter++;
}
int originalSize = BitConverter
    .ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;

if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
        .Ceiling((originalSize - (
            part_size_a + part_size_b) * 2) / 2.0);
}

// Извлекаем данные из блока A
counter = 0;
byte[] data_1 = new byte[part_size_a];

```

```

for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_a[counter];
    counter++;
}
byte[] data_56_parity = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_56_parity[i] = data_a[counter];
    counter++;
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_34_parity[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{

```

```

        data_5[i] = data_b[counter];
        counter++;
    }
    byte[] data_6 = SolveParity(
        data_56_parity, data_5);
    byte[] data_4 = SolveParity(
        data_34_parity, data_3);
    if (data_6[data_6.Length - 5] == 0)
        reduce = 1;
    byte[] data = new byte[(part_size_a
        + part_size_b + part_size_c) * 2 - reduce];
    counter = 0;
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_1[i];
        counter++;
    }
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_2[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {

```

```

        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }

    return data;

} else if(option == 3)
{
    curFileName = fileName + "_A";
    await _googledriveService
        .DownloadFile(curFileName, Directory
            .GetCurrentDirectory() + "/" + curFileName);
    byte[] data_a = File
        .ReadAllBytes(Directory.
            GetCurrentDirectory() + "/" + curFileName);
    curFileName = fileName + "_C";
    byte[] data_parity = await
        _dropboxService.DownloadFile(
            "/" + curFileName);

    string[] partsConfig = key.Split(',');

    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_a.Length - 4;
        i < data_a.Length; i++)
    {
        originalSizeBytes[counter]
            = data_a[i];
    }

```

```

        counter++;
    }
    int originalSize = BitConverter
        .ToInt32(originalSizeBytes, 0);
    int part_size_a = originalSize
        / 100 * int.Parse(partsConfig[0]) / 2;
    int part_size_b = originalSize
        / 100 * int.Parse(partsConfig[1]) / 2;
    int part_size_c = 0;
    int reduce = 0;
    if (data_parity[data_parity.Length - 5] == 0)
        reduce = 1;
    if (part_size_a + part_size_b < originalSize)
    {
        part_size_c += (int)Math
            .Ceiling((originalSize -
                (part_size_a + part_size_b) * 2) / 2.0);
    }

    counter = 0;
    byte[] data_1 = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_1[i] = data_a[counter];
        counter++;
    }
    byte[] data_3 = new byte[part_size_b];
    for (int i = 0; i < part_size_a; i++)
    {
        data_3[i] = data_a[counter];
        counter++;
    }
    byte[] data_56_parity = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)

```

```

{
    data_56_parity[i] = data_a[counter];
    counter++;
}

counter = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_12_parity[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

byte[] data_2 = SolveParity(
    data_12_parity, data_1);
byte[] data_5 = SolveParity(
    data_56_parity, data_6);

byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)

```



```

    {
        data[counter] = data_1[i];
        counter++;
    }
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_2[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }
    return data;
}

byte[] datsa = new byte[1];
return datsa;

```

```

}

public async Task<byte[]> ReadDataWithoutGoogle
(string fileName, int option, string key)
{
    string curFileName = fileName;
    if (option == 1)
    {
        curFileName = fileName + "_A";
        byte[] data_a = await
            _yandexdiskService.
            DownloadFile(curFileName);
        curFileName = fileName + "_B";
        byte[] data_b = await
            _dropboxService.DownloadFile(
            "/" + curFileName);

        string[] partsConfig = key.Split(',');

        byte[] originalSizeBytes = new byte[4];
        int counter = 0;
        for (int i = data_b.Length - 4;
            i < data_b.Length; i++)
        {
            originalSizeBytes[counter] = data_b[i];
            counter++;
        }
        int originalSize = BitConverter
            .ToInt32(originalSizeBytes, 0);
        int part_size_a = originalSize
            / 100 * int.Parse(partsConfig[0]) / 2;
        int part_size_b = originalSize
            / 100 * int.Parse(partsConfig[1]) / 2;
        int part_size_c = 0;
    }
}

```

```

int reduce = 0;

if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
        .Ceiling((originalSize
        - (part_size_a + part_size_b)
        * 2) / 2.0);
}

counter = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}

byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_a[counter];
    counter++;
}

byte[] data_56_parity = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_56_parity[i] = data_a[counter];
    counter++;
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{

```

```

        data_2[i] = data_b[counter];
        counter++;
    }
    byte[] data_34_parity = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_34_parity[i] = data_b[counter];
        counter++;
    }
    byte[] data_5 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_5[i] = data_b[counter];
        counter++;
    }
    byte[] data_6 = SolveParity(
        data_56_parity, data_5);
    byte[] data_4 = SolveParity(
        data_34_parity, data_3);
    if (data_6[data_6.Length - 5] == 0)
        reduce = 1;
    byte[] data = new byte[(part_size_a
        + part_size_b + part_size_c) * 2 - reduce];
    counter = 0;
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_1[i];
        counter++;
    }
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_2[i];
        counter++;
    }

```

```

        for (int i = 0; i < part_size_b; i++)
        {
            data[counter] = data_3[i];
            counter++;
        }
        for (int i = 0; i < part_size_b; i++)
        {
            data[counter] = data_4[i];
            counter++;
        }
        for (int i = 0; i < part_size_c; i++)
        {
            data[counter] = data_5[i];
            counter++;
        }
        for (int i = 0; i < part_size_c - reduce; i++)
        {
            data[counter] = data_6[i];
            counter++;
        }

        return data;
    }
    else if (option == 2)
    {
        curFileName = fileName + "_C";
        byte[] data_parity = await _yandexdiskService
            .DownloadFile(curFileName);
        curFileName = fileName + "_B";
        byte[] data_b = await _dropboxService
            .DownloadFile("/") + curFileName);

        string[] partsConfig = key.Split(',');
    }

```

```

byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_b.Length - 4;
i < data_b.Length; i++)
{
    originalSizeBytes[counter] = data_b[i];
    counter++;
}
int originalSize = BitConverter
.ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
.Ceiling((originalSize -
(part_size_a + part_size_b) * 2) / 2.0);
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)

```

```

{
    data_34_parity[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}

counter = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_12_parity[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

byte[] data_1 = SolveParity(
    data_12_parity, data_2);
byte[] data_3 = SolveParity(

```

```

        data_34_parity, data_4);

byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{
    data[counter] = data_3[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{
    data[counter] = data_4[i];
    counter++;
}
for (int i = 0; i < part_size_c; i++)
{
    data[counter] = data_5[i];
    counter++;
}
for (int i = 0; i < part_size_c - reduce; i++)
{
    data[counter] = data_6[i];
    counter++;
}

```



```

    }
    return data;

}
else if (option == 3)
{
    curFileName = fileName + "_B";
    byte[] data_b = await
        _yandexdiskService
        .DownloadFile(curFileName);
    curFileName = fileName + "_C";
    byte[] data_parity = await
        _dropboxService.DownloadFile(
            "/" + curFileName);

    string[] partsConfig = key.Split(',');

    byte[] originalSizeBytes = new byte[4];
    int counter = 0;
    for (int i = data_b.Length
        - 4; i < data_b.Length; i++)
    {
        originalSizeBytes
            [counter] = data_b[i];
        counter++;
    }
    int originalSize = BitConverter
        .ToInt32(originalSizeBytes, 0);
    int part_size_a = originalSize
        / 100 * int.Parse(partsConfig[0]) / 2;
    int part_size_b = originalSize
        / 100 * int.Parse(partsConfig[1]) / 2;
    int part_size_c = 0;
    int reduce = 0;

```

```

if (data_parity[data_parity
.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b
< originalSize)
{
    part_size_c += (int)Math
        .Ceiling((originalSize - (
            part_size_a + part_size_b) * 2) / 2.0);
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_34_parity[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}

counter = 0;
byte[] data_12_parity = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)

```

```

{
    data_12_parity[i] = data_parity[counter];
    counter++;
}
byte[] data_4 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_4[i] = data_parity[counter];
    counter++;
}
byte[] data_6 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_6[i] = data_parity[counter];
    counter++;
}

byte[] data_1 = SolveParity(data_12_parity, data_2);
byte[] data_3 = SolveParity(data_34_parity, data_4);

byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)

```

```

        {
            data[counter] = data_3[i];
            counter++;
        }
        for (int i = 0; i < part_size_b; i++)
        {
            data[counter] = data_4[i];
            counter++;
        }
        for (int i = 0; i < part_size_c; i++)
        {
            data[counter] = data_5[i];
            counter++;
        }
        for (int i = 0; i < part_size_c - reduce; i++)
        {
            data[counter] = data_6[i];
            counter++;
        }
    }

    byte[] datsa = new byte[1];
    return datsa;
}

// Метод загрузки при утери Dropbox
public async Task<byte[]> ReadDataWithoutDropbox
    (string fileName, int option, string key)
{
    string curFileName = fileName;
    if (option == 1)
    {
        // Опция 1 Данные А Яндекс С Гугл
        curFileName = fileName + "_A";
        byte[] data_a = await _yandexdiskService

```

```

.DownloadFile(curFileName);
curFileName = fileName + "_C";
await _googledriveService
.DownloadFile(curFileName,
Directory.GetCurrentDirectory() +
"/" + curFileName);
byte[] data_parity = File.ReadAllBytes(
Directory.GetCurrentDirectory()
+ "/" + curFileName);
string[] partsConfig = key.Split(',');

byte[] originalSizeBytes = new byte[4];
int counter = 0;
for (int i = data_a.Length -
4; i < data_a.Length; i++)
{
    originalSizeBytes[counter]
    = data_a[i];
    counter++;
}
int originalSize = BitConverter
.ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
    .Ceiling((originalSize - (

```

```

        part_size_a + part_size_b) * 2) / 2.0);
    }

    counter = 0;
    byte[] data_1 = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_1[i] = data_a[counter];
        counter++;
    }
    byte[] data_3 = new byte[part_size_b];
    for (int i = 0; i < part_size_a; i++)
    {
        data_3[i] = data_a[counter];
        counter++;
    }
    byte[] data_56_parity = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_56_parity[i] = data_a[counter];
        counter++;
    }

    counter = 0;
    byte[] data_12_parity = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_12_parity[i] = data_parity[counter];
        counter++;
    }
    byte[] data_4 = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_4[i] = data_parity[counter];

```

```

        counter++;
    }
    byte[] data_6 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_6[i] = data_parity[counter];
        counter++;
    }

    byte[] data_2 = SolveParity
        (data_12_parity, data_1);
    byte[] data_5 = SolveParity
        (data_56_parity, data_6);

    byte[] data = new byte[(part_size_a
    + part_size_b + part_size_c) * 2 - reduce];
    counter = 0;
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_1[i];
        counter++;
    }
    for (int i = 0; i < part_size_a; i++)
    {
        data[counter] = data_2[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {

```

```

        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }
    return data;
}
else if (option == 2)
{
    // Вариант 2 без Dropbox:
    // Используем А Гугл и С Яндекс
    curFileName = fileName + "_A";
    await _googledriveService
        .DownloadFile(curFileName,
            Directory.GetCurrentDirectory()
            + "/" + curFileName);
    byte[] data_a = File.ReadAllBytes(
        Directory.GetCurrentDirectory() +
        "/" + curFileName);
    curFileName = fileName + "_C";
    byte[] data_parity = await
        _yandexdiskService.DownloadFile
        (curFileName);
    string[] partsConfig = key.Split(',');

    byte[] originalSizeBytes = new byte[4];

```



```

int counter = 0;
for (int i = data_a.Length - 4;
i < data_a.Length; i++)
{
    originalSizeBytes[counter] = data_a[i];
    counter++;
}
int originalSize = BitConverter
.ToInt32(originalSizeBytes, 0);
int part_size_a = originalSize
/ 100 * int.Parse(partsConfig[0]) / 2;
int part_size_b = originalSize
/ 100 * int.Parse(partsConfig[1]) / 2;
int part_size_c = 0;
int reduce = 0;
if (data_parity[data_parity.Length - 5] == 0)
    reduce = 1;
if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math
.Ceiling((originalSize - (part_size_a
+ part_size_b) * 2) / 2.0);
}

counter = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_a; i++)
{

```

```

        data_3[i] = data_a[counter];
        counter++;
    }
    byte[] data_56_parity = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_56_parity[i] = data_a[counter];
        counter++;
    }

    counter = 0;
    byte[] data_12_parity = new byte[part_size_a];
    for (int i = 0; i < part_size_a; i++)
    {
        data_12_parity[i] = data_parity[counter];
        counter++;
    }
    byte[] data_4 = new byte[part_size_b];
    for (int i = 0; i < part_size_b; i++)
    {
        data_4[i] = data_parity[counter];
        counter++;
    }
    byte[] data_6 = new byte[part_size_c];
    for (int i = 0; i < part_size_c; i++)
    {
        data_6[i] = data_parity[counter];
        counter++;
    }

    byte[] data_2 = SolveParity(
        data_12_parity, data_1);
    byte[] data_5 = SolveParity(
        data_56_parity, data_6);

```

```

byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{
    data[counter] = data_3[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{
    data[counter] = data_4[i];
    counter++;
}
for (int i = 0; i < part_size_c; i++)
{
    data[counter] = data_5[i];
    counter++;
}
for (int i = 0; i < part_size_c - reduce; i++)
{
    data[counter] = data_6[i];
    counter++;
}

```

```

        return data;
    }
    else if (option == 3)
    {
        curFileName = fileName + "_A";
        await _googledriveService.DownloadFile(
            curFileName, Directory.GetCurrentDirectory()
            + "/" + curFileName);
        byte[] data_a = File.ReadAllBytes(
            Directory.GetCurrentDirectory() +
            "/" + curFileName);
        curFileName = fileName + "_B";
        byte[] data_b = await _yandexdiskService
            .DownloadFile(curFileName);

        string[] partsConfig = key.Split(',');

        byte[] originalSizeBytes = new byte[4];
        int counter = 0;
        for (int i = data_b.Length - 4; i
            < data_b.Length; i++)
        {
            originalSizeBytes[counter] = data_b[i];
            counter++;
        }
        int originalSize = BitConverter
            .ToInt32(originalSizeBytes, 0);
        int part_size_a = originalSize / 100
            * int.Parse(partsConfig[0]) / 2;
        int part_size_b = originalSize / 100
            * int.Parse(partsConfig[1]) / 2;
        int part_size_c = 0;
        int reduce = 0;
    }
}

```

```

if (part_size_a + part_size_b < originalSize)
{
    part_size_c += (int)Math.Ceiling
        ((originalSize - (part_size_a +
            part_size_b) * 2) / 2.0);
}

counter = 0;
byte[] data_1 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_1[i] = data_a[counter];
    counter++;
}
byte[] data_3 = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_3[i] = data_a[counter];
    counter++;
}
byte[] data_56_parity = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_56_parity[i] = data_a[counter];
    counter++;
}

counter = 0;
byte[] data_2 = new byte[part_size_a];
for (int i = 0; i < part_size_a; i++)
{
    data_2[i] = data_b[counter];
    counter++;
}

```

```

}
byte[] data_34_parity = new byte[part_size_b];
for (int i = 0; i < part_size_b; i++)
{
    data_34_parity[i] = data_b[counter];
    counter++;
}
byte[] data_5 = new byte[part_size_c];
for (int i = 0; i < part_size_c; i++)
{
    data_5[i] = data_b[counter];
    counter++;
}
byte[] data_6 = SolveParity
(data_56_parity, data_5);
byte[] data_4 = SolveParity
(data_34_parity, data_3);
if (data_6[data_6.Length - 5] == 0)
    reduce = 1;
byte[] data = new byte[(part_size_a
+ part_size_b + part_size_c) * 2 - reduce];
counter = 0;
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_1[i];
    counter++;
}
for (int i = 0; i < part_size_a; i++)
{
    data[counter] = data_2[i];
    counter++;
}
for (int i = 0; i < part_size_b; i++)
{

```

```

        data[counter] = data_3[i];
        counter++;
    }
    for (int i = 0; i < part_size_b; i++)
    {
        data[counter] = data_4[i];
        counter++;
    }
    for (int i = 0; i < part_size_c; i++)
    {
        data[counter] = data_5[i];
        counter++;
    }
    for (int i = 0; i < part_size_c - reduce; i++)
    {
        data[counter] = data_6[i];
        counter++;
    }

    return data;
}

byte[] datsa = new byte[1];
return datsa;
}

public byte[] SolveParity(byte[] data_1, byte[] data_2)
{
    byte[] parity = new byte[data_1.Length];
    for (int i = 0; i < parity.Length; i++)
        parity[i] = (byte)(data_1[i] ^ data_2[i]);
    return parity;
}
}

```

}



## ПРИЛОЖЕНИЕ И

### Исходный код Raid5Controller

```
namespace vkr.Controllers
{
    /// <summary>
    /// API-контроллер, предоставляющий конечные
    /// точки для работы с функциональностью RAID 5:
    /// загрузка файлов, скачивание с восстановлением
    /// и получение расширений файлов.
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class RAID5Controller : Controller
    {
        public readonly RAID5Service _raid5Service;

        /// <summary>
        /// Конструктор контроллера, получает RAID5Service.
        /// </summary>
        public RAID5Controller(RAID5Service raid5Service)
        {
            _raid5Service = raid5Service;
        }

        /// <summary>
        /// Метод загрузки файла, его распределения по
        /// схеме RAID5 и сохранения.
        /// </summary>
        [HttpPost("upload")]
        public async Task<IActionResult> WriteData(
            IFormFile file, [FromForm]
            int distributionMethod,
            [FromForm] string key)
```

```

{
    try
    {
        // Проверка: если файл не
        // передан или пустой - вернуть ошибку
        if (file == null || file.Length == 0)
        {
            return BadRequest("Файл не загружен");
        }

        // Чтение файла в память
        using (var memoryStream = new MemoryStream())
        {
            // Асинхронное копирование содержимого файла в поток
            await file.CopyToAsync(memoryStream);
            // Преобразование потока в массив байтов
            var fileBytes = memoryStream.ToArray();

            // Вызов бизнес-логики
            var result = await _raid5Service.WriteData(
                fileBytes,           // Содержимое файла
                file.FileName,       // Имя файла
                distributionMethod,  // Метод распределения
                key);                // Ключ шифрования

            // Возврат успешного ответа с результатом
            return Ok(new { success = true, data = result });
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { error = ex.Message });
    }
}

```

```

/// <summary>
/// Метод для скачивания и восстановления
/// файла из всех облаков согласно схеме RAID5.
/// </summary>
[HttpGet("download/{fileName}")]
public async Task<IActionResult> ReadData(
    string fileName,
    [FromQuery] int distributionMethod,
    [FromQuery] string key)
{
    // Получение содержимого файла,
    // восстановленного с учётом схемы RAID5
    var fileContent = await
        _raid5Service
        .ReadData(
            fileName,
            distributionMethod,
            key);
    return File(fileContent, "application/octet-stream");
}

/// <summary>
/// Метод восстановления и скачивания файла без
/// использования облачного хранилища Яндекс.Диск.
/// Имитирует отказ одного из дисков RAID5.
/// </summary>
[HttpGet("downloadwithoutya/{fileName}")]
public async Task<IActionResult>
ReadDataWithoutYandex(
    string fileName,
    [FromQuery] int distributionMethod,
    [FromQuery] string key)
{

```

```

        var fileContent = await
            _raid5Service.ReadDataWithoutYandex(
                fileName, distributionMethod, key);
        return File(fileContent,
            "application/octet-stream");
    }

```

```

/// <summary>
/// Метод восстановления и скачивания файл
/// без использования Google Диска.
/// </summary>
[HttpGet("downloadwithoutgoogle/{fileName}")]
public async Task<IActionResult>
    ReadDataWithoutGoogle(
        string fileName,
        [FromQuery] int distributionMethod,
        [FromQuery] string key)
    {
        var fileContent = await
            _raid5Service
                .ReadDataWithoutGoogle(
                    fileName, distributionMethod, key);
        return File(fileContent,
            "application/octet-stream");
    }

```

```

/// <summary>
/// Метод восстановления и
/// скачивания файла без использования Dropbox.
/// Аналогично, предназначен для
/// имитации потери одного диска.
/// </summary>
[HttpGet("downloadwithoutdropbox/{fileName}")]
public async Task<IActionResult>

```

```

ReadDataWithoutDropbox(
string fileName,
[FromQuery] int distributionMethod
, [FromQuery] string key)
{
    var fileContent = await
        _raid5Service
        .ReadDataWithoutDropbox(
            fileName, distributionMethod, key);
    return File(fileContent,
        "application/octet-stream");
}
}
}

```

## ПРИЛОЖЕНИЕ К Исходный код App.vue

```
<template>
  <!-- Основной контейнер страницы -->
  <div class="container">

    <!-- Блок отображения списка файлов -->
    <div class="files-list">
      <!-- Вывод уникальных имён файлов -->
      <div v-for="(file, index) in
        uniqueFilesWithoutExtension"
        :key="index" class="file-item">
        <p>{{ file }}</p>
      </div>
    </div>

    <!-- Управляющий блок -->
    <div class="controls">

      <!-- Кнопка для выбора ключевого файла -->
      <button @click="openKeyPicker"
        class="select-btn">
        Выбрать ключ
      </button>

      <!-- Скрытый input для выбора ключевого файла -->
      <input
        type="file"
        ref="keyInput"
        @change="handleKeySelect"
        style="display: none"
        accept=".txt,.key"
      >
    </div>
  </div>
</template>
```

```

<!-- Отображение имени выбранного ключевого файла -->
<div v-if="selectedKeyFile" class="selected-file">
    Название ключевого файла: {{ selectedKeyFile.name }}
</div>

<!-- Кнопка выбора обычного файла -->
<button @click="openFilePicker" class="select-btn">
    Выбрать файл для загрузки
</button>

<!-- Скрытый input для выбора загружаемого файла -->
<input
    type="file"
    ref="fileInput"
    @change="handleFileSelect"
    style="display: none"
>

<!-- Отображение информации о выбранном
для загрузки файле -->
<div v-if="selectedLocalFile" class="selected-file">
    Название файла: {{ selectedLocalFile.name }}
</div>

<!-- Кнопка для отправки файла на сервер -->
<button
    @click="sendFilePath"
    class="upload-btn"
    :disabled="!selectedLocalFile"
>
    Загрузить файл
</button>

<!-- Выпадающее меню выбора файла для скачивания -->

```

```

<div class="file-selection">
  <button @click="showFileDropdown =
    !showFileDropdown" class="select-btn">
    Выбрать файл для скачивания
  </button>
  <div v-if="showFileDropdown" class="dropdown">
    <div
      v-for="(file, index) in uniqueFiles"
      :key="index"
      class="dropdown-item"
      @click="selectServerFile(file)"
    >
      {{ file.slice(0, -2) }}
    </div>
  </div>

  <!-- Отображение выбранного файла -->
  <div v-if="selectedServerFile" class="selected-file">
    Выбран файл: {{ selectedServerFile.slice(0, -2) }}
  </div>
</div>

<!-- Кнопка для скачивания файла -->
<button
  @click="downloadFiles()"
  class="download-btn"
  :disabled="!selectedServerFile"
>
  Скачать
</button>

<!-- Группа радиокнопок для выбора метода скачивания -->
<div class="radio-group">
  <h3>Выберите метод скачивания:</h3>

```



```

<label v-for="option in options"
:key="option.value" class="radio-label">
  <input
    type="radio"
    v-model="selectedOption"
    :value="option.value"
  >
  {{ option.label }}
</label>
</div>

<!-- Настройки RAID: метод распределения -->
<div class="configuration">
  <h3>Настройки RAID:</h3>
  <div class="input-group">
    <label>Метод распределения:</label>
    <select v-model.number="selectedDistributionMethod">
      <option v-for="option in options2"
        :key="option.value" :value="option.value">
        {{ option.label }}
      </option>
    </select>
  </div>
</div>

</div>
</div>
</template>

<script setup>
/**
 * Импорт основных функций из Vue 3 Composition API
 */
import { ref, onMounted, computed } from 'vue';

```

```

// Хранилище состояния компонента
const state = ref({
  files: [] // массив имён файлов, полученных с сервера
});

/**
 * Получение уникальных имён файлов
 */
const uniqueFiles = computed(() => {
  return [...new Set(state.value.files)];
});

/**
 * Удаление последних двух символов из имени файла
 */
const uniqueFilesWithoutExtension = computed(() => {
  return uniqueFiles.value.map(file => file.slice(0, -2));
});

/**
 * Опции для выбора метода скачивания (влияние на источники: с исключени
 */
const options = ref([
  { value: 'option1', label: 'Скачать со всех дисков' },
  { value: 'option2', label: 'Скачать без Яндекс Диска' },
  { value: 'option3', label: 'Скачать без Dropbox' },
  { value: 'option4', label: 'Скачать без Google Drive' },
  { value: 'option5', label: 'Скачать случайным методом' }
]);

/**
 * Методы распределения (логика RAID5 на серверной стороне)
 */

```

```

const options2 = ref([
  { value: 1, label: 'Конфигурация 1' },
  { value: 2, label: 'Конфигурация 2' },
  { value: 3, label: 'Конфигурация 3' }
]);

// Текущее состояние выбора
const selectedDistributionMethod = ref(1);
// Выбранный метод RAID
const selectedOption = ref('option1');
// Метод скачивания
const selectedServerFile = ref(null);
// Файл, выбранный для скачивания
const selectedLocalFile = ref(null);
// Файл, выбранный для загрузки
const selectedKeyFile = ref(null);
// Файл ключа
const showFileDropdown = ref(false);
// Видимость выпадающего списка
const fileInput = ref(null);
// Ссылка на input выбора файла
const keyInput = ref(null);
// Ссылка на input выбора ключа
const keyContent = ref('');
// Содержимое ключевого файла
const keyValues = ref('10,22');
// Значения ключа (по умолчанию)

/**
 * Получение списка файлов с сервера Google Drive
 */
const getFilesNames = async () => {
  const response = await fetch(
    'https://localhost:7229/api/GoogleDrive/files');

```

```

    return await response.json();
};

/**
 * Выбор файла из выпадающего списка
 */
const selectServerFile = (fileName) => {
    selectedServerFile.value = fileName;
    showFileDropdown.value = false;
};

/**
 * Открытие системного диалога выбора файла для загрузки
 */
const openFilePicker = () => {
    fileInput.value.click();
};

/**
 * Открытие диалога выбора ключа
 */
const openKeyPicker = () => {
    keyInput.value.click();
};

/**
 * Обработка выбора файла для загрузки
 */
const handleFileSelect = (event) => {
    const file = event.target.files[0];
    if (file) {
        selectedLocalFile.value = {
            name: file.name,
            path: file.path,

```

```

        size: file.size,
        type: file.type,
        lastModified: file.lastModified,
        file: file
    };
    console.log('Selected file object:',
        selectedLocalFile.value);
}
};

/**
 * Извлечение последней строки из содержимого ключа
 */
const parseKeyContent = (content) => {
    const lines = content.split('\n');
    return lines[lines.length - 1].trim();
};

/**
 * Обработка выбора ключевого файла
 */
const handleKeySelect = async (event) => {
    const file = event.target.files[0];
    if (file) {
        selectedKeyFile.value = {
            name: file.name,
            file: file
        };
        const content = await readFileAsText(file);
        keyValues.value = parseKeyContent(content);
        console.log('Key values:', keyValues.value);
    }
};

```

```

/**
 * Чтение содержимого текстового файла
 */
const readFileAsText = (file) => {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onload = (event) => resolve(event.target.result);
    reader.onerror = (error) => reject(error);
    reader.readAsText(file);
  });
};

/**
 * Скачивание файла с сервера в зависимости от выбранного метода
 */
const downloadFiles = async () => {
  if (!selectedServerFile.value) return;

  const fileName = selectedServerFile.value.slice(0, -2);
  // удаляем суффикс '.ya', '.go' и т.д.
  let responseUrl = '';

  // Выбор API-метода в зависимости от опции
  switch(selectedOption.value) {
    case 'option1':
      responseUrl = 'https://localhost:7229/api/RAID5/download'
        /`${fileName}?distributionMethod=${
          selectedDistributionMethod.value}&key=${
            {keyValues.value}';
      break;
    case 'option2':
      responseUrl = 'https://localhost:7229/api/RAID5/'
        downloadwithoutya/${fileName}?distributionMethod=
        ${selectedDistributionMethod.value}&key=

```

```

    ${keyValues.value}';
    break;
case 'option3':
    responseUrl = 'https://localhost:7229/api/RAID5/
downloadwithoutgoogle/${fileName}?distributionMethod=
${selectedDistributionMethod.value}&key=
${keyValues.value}';
    break;
case 'option4':
    responseUrl = 'https://localhost:7229/api/RAID5/
downloadwithoutdropbox/${fileName}?
distributionMethod=${selectedDistributionMethod
.value}&key=${keyValues.value}';
    break;
case 'option5':
    const randomOption = Math.floor(Math.random() * 4) + 1;
    responseUrl = 'https://localhost:7229/api/RAID5/${
        randomOption === 1 ? 'download' :
        randomOption === 2 ? 'downloadwithoutya' :
        randomOption === 3 ? 'downloadwithoutgoogle' :
        'downloadwithoutdropbox'
    }/${fileName}?distributionMethod=
${selectedDistributionMethod.value}
&key=${keyValues.value}';
    break;
}

// Скачивание файла
const response = await fetch(responseUrl);
const blob = await response.blob();
await computeDownload(blob);
return response;
};

```

```

/**
 * Вычисление и запуск скачивания файла через временный <a>-элемент
 */
async function computeDownload(blob) {
  const url = window.URL.createObjectURL(blob);
  const link = document.createElement('a');
  const fileName = selectedServerFile.value.slice(0, -2);
  // удаляем окончание
  link.href = url;
  link.setAttribute('download', fileName);
  // задаём имя для скачивания
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  window.URL.revokeObjectURL(url);
  // освобождение ресурсов
}

/**
 * Отправка выбранного файла на сервер
 * с ключом и методом распределения
 */
const sendFilePath = async () => {
  if (!selectedLocalFile.value?.file) {
    alert('Пожалуйста, выберите файл');
    return;
  }

  const formData = new FormData();
  formData.append('file', selectedLocalFile.value.file);
  formData.append('distributionMethod',
    selectedDistributionMethod.value.toString());
  formData.append('key', keyValues.value);

```



```

try {
  const response = await fetch('https://
localhost:7229/api/RAID5/upload', {
    method: 'POST',
    body: formData
  });

  if (!response.ok) {
    throw new Error('Ошибка HTTP! статус: ${response.status}');
  }

  const result = await response.json();
  console.log('Файл успешно загружен:', result);
  alert('Файл успешно загружен');
} catch (error) {
  console.error('Ошибка при загрузке файла:', error);
  alert('Ошибка: ' + error.message);
}
};

/**
 * Получение списка файлов при инициализации компонента
 */
onMounted(async () => {
  state.value.files = await getFilesNames();
  console.log(state.value.files);
});
</script>

```

Бакалаврская работа "Применение метода распределенного хранения данных для повышения их защищенности" выполнена мною самостоятельно, и на все источники, имеющиеся в работе, даны соответствующие ссылки.

---

подпись, дата

---

инициалы, фамилия