СОДЕРЖАНИЕ

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 технологии хранения данных стремительно развиваются, предлагая пользователям всё более надёжные и эффективные решения для управления информацией. Одним из ключевых направлений в данной области является облачное хранение данных, которое обеспечивает доступ к файлам с любого устройства и высокую отказоустойчивость. Однако с ростом объёмов данных и требований к их безопасности возникает необходимость в оптимизации методов хранения, включая распределение информации между разными облачными сервисами для повышения надёжности.

Предлагаемое решение предоставление пользователю доступа к файлам, расположенным на облачных сервисах, с возможностью восстановления данных при потере доступа к одному из сервисов.

В рамках работы предполагается разработка решения для взаимодействия с облачными сервисами, а также метода организации файловой системы по принципу `RAID 5`. Основой проекта выбрана технология `ASP.NET Core`, язык программирования `C#`. Для практической реализации и демонстрации работоспособности реализуется одностраничное веб-приложение.

Также в ходе работы требуется провести тестирование разработанного приложения. Необходимо оценить как корректность реализованного функционала, так и его соответствие выдвинутым к системе требованиям.

Целью данной бакалаврской работы является разработка приложения для хранения файлов на облачных сервисах с распределением по принципу `RAID 5`. Данное приложение должно представлять собой инструмент, позволяющий пользователю взаимодействовать с хранимыми данными.

Для достижения цели были поставлены следующие задачи:

1. Изучить принцип работы `RAID 5`;
2. Изучение различных сервисов хранения и способов взаимодействия с ними;
3. Реализовать модуль распределения данных по `RAID 5`;
4. Реализовать пользовательский интерфейс.

1 Постановка задачи

Необходимо разработать одностраничное веб-приложение с распределением данных по принципу `RAID 5` на трёх облачных сервисах для скачивания и загрузки файлов, а также восстановления данных в случае отказа одного из сервисов. Для реализации backend-части приложения используется фреймворк ASP.NET Core, язык программирования C#. Front-end часть приложения реализуется при помощи языка программирования JavaScript и фреймворка Vue3.js. Для хранения данных используются сервисы Яндекс Диск, Dropbox, Google Drive.

Для реализации приложения были выделены следующие этапы:

— Реализация класса, отвечающего за сборку проекта, вносящего необходимые при сборке проекта параметры.
— Разработка класса, выполняющего инициализацию представленных в проекте сервисов
— Реализация классов, отвечающих за запросы на облачные сервисы
— Подготовка набора сервисов, отвечающих за внутреннюю логику, выполняющих запросы на пользовательский интерфейс
— Реализация контроллеров для взаимодействия с внешними приложениями

Основной функционал программы подразумевает:

— Загрузка данных — Позволяет загрузить файл в облачные сервисы с учетом распределения данных.
— Скачивание файла — Позволяет скачать разделенный файл с автоматической сборкой файла.
— Восстановление данных — Позволяет восстановить утерянные данные в случае отказа одного из дисков.
— Сборка файла из частей — При распределении при помощи `RAID 5` файл хранится в виде нескольких наборов байтов.
— Разделение массива данных файла на части — Для распределения при помощи `RAID 5` файл должен быть разделен на несколько наборов байтов.
— Применение массива чётности — Для восстановления утерянных данных используется массив чётности.
— Запрос к облачным сервисам — Воспроизведение взаимодействия при-

ложения с облачными сервисами.

Схема взаимодействия пользователя с программой предоставлена на рисунке 1.

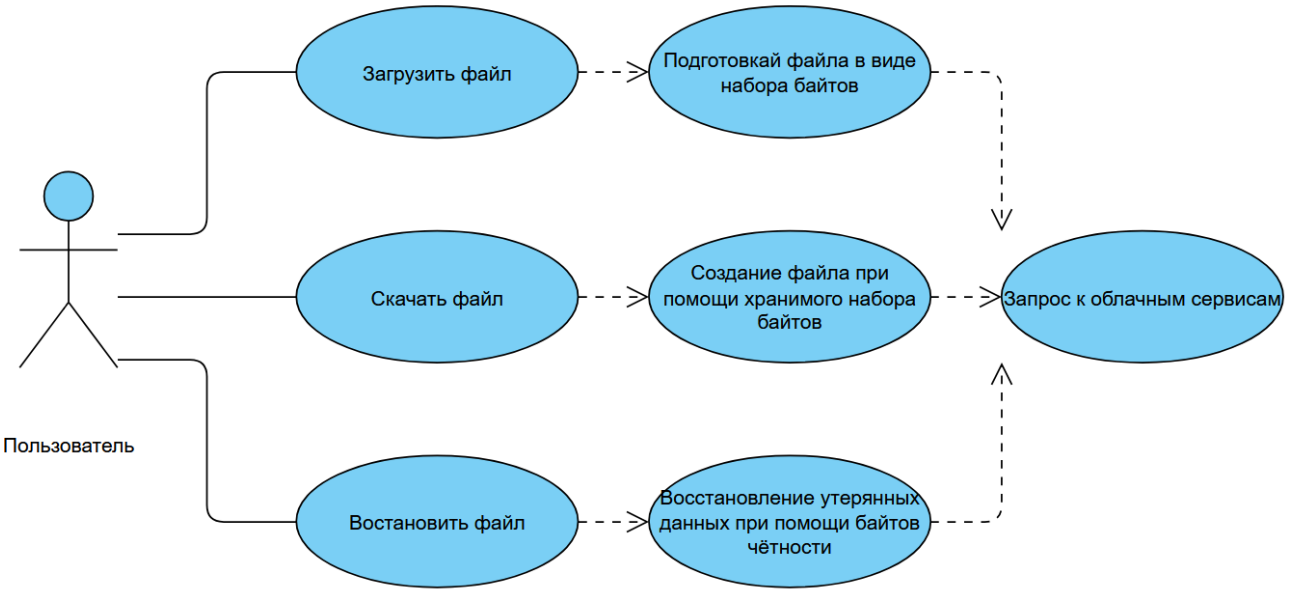

Рисунок 1 – Схема взаимодействия пользователя с программой

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 расшифровывалась как "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 [1].

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 [2].

RAID-массивы различаются по способу организации данных и уровню избыточности. Каждый уровень RAID определяет:

— метод записи (чередование, зеркалирование, контрольные суммы).
— степень отказоустойчивости.
— производительность.
— эффективность использования дискового пространства.

Далее будут рассмотрены наиболее распространённые в применении

уровни RAID

RAID 0 (Striping – чередование)

— Принцип работы: Данные разбиваются на блоки и равномерно распределяются по всем дискам массива.
— Преимущества:
  – Максимальная производительность (чтение/запись ускоряются за счет параллельной работы дисков).
  – Полное использование емкости (нет избыточности).
— Недостатки:
  – Нет отказоустойчивости — выход одного диска приводит к потере всех данных.
— Минимальное количество дисков: 2
— Применение: Для временных данных, кэширования, задач, где важна скорость, но не надежность.

RAID 1 (Mirroring – зеркалирование)

— Принцип работы: Данные полностью дублируются на двух или более дисках.
— Преимущества:
  – Высокая надежность — при отказе одного диска данные сохраняются на другом.
  – Быстрое восстановление.
— Недостатки:
  – Высокие затраты — полезная емкость равна половине общего объема (при двух дисках).
  – Скорость записи может уменьшается за счет количества дисков.
— Минимальное количество дисков: 2
— Применение: Критически важные данные.

RAID 5 (Чередование с контролем чётности)

— Принцип работы: Данные и контрольные суммы (parity) распределяются по всем дискам.
— Преимущества:
  – Оптимальный баланс между надежностью, производительностью и затратами.
  – Выдерживает отказ одного диска без потери данных.

  - Эффективное использование емкости.
- Недостатки:
  - Замедление записи из-за расчета контрольных сумм.
  - Долгое восстановление при замене диска
- Минимальное количество дисков: 3
- Применение: Файловые серверы, веб-хранилища, СУБД

RAID 6 (Двойная четность)

- Принцип работы: Данные и два набора контрольных сумм распределяются по всем дискам.
- Преимущества:
  - Выдерживает отказ двух дисков одновременно.
  - Надежнее RAID 5 для больших массивов.
- Недостатки:
  - Еще более низкая скорость записи
  - Большие потери емкости
- Минимальное количество дисков: 4
- Применение: Системы с высокими требованиями к отказоустойчивости

RAID 10 (1+0, зеркалирование + чередование)

- Принцип работы: Комбинация RAID 1 и RAID 0 — сначала диски зеркалируются, затем данные чередуются.
- Преимущества:
  - Высокая производительность
  - Хорошая отказоустойчивость
- Недостатки:
  - Высокая стоимость (только половина полезной емкости).
- Минимальное количество дисков: 4
- Применение: Высоконагруженные базы данных, виртуализация, транзакционные системы.

Исходя из предоставленных данных можно сделать вывод об эффективности методов для реализации поставленной задачи. `RAID 5` представляется наиболее сбалансированной моделью для применения на небольших данных, является экономным по занимаемому месту, резервируя только одно из предоставленных хранилищ данных, а также предоставляет исчерпывающую отказоустойчивость, позволяя восстановить данные при потере доступа к одному

из дисков.

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 [3]. 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 2 продемонстрирована модель распределения данных с применением трёх дисков.

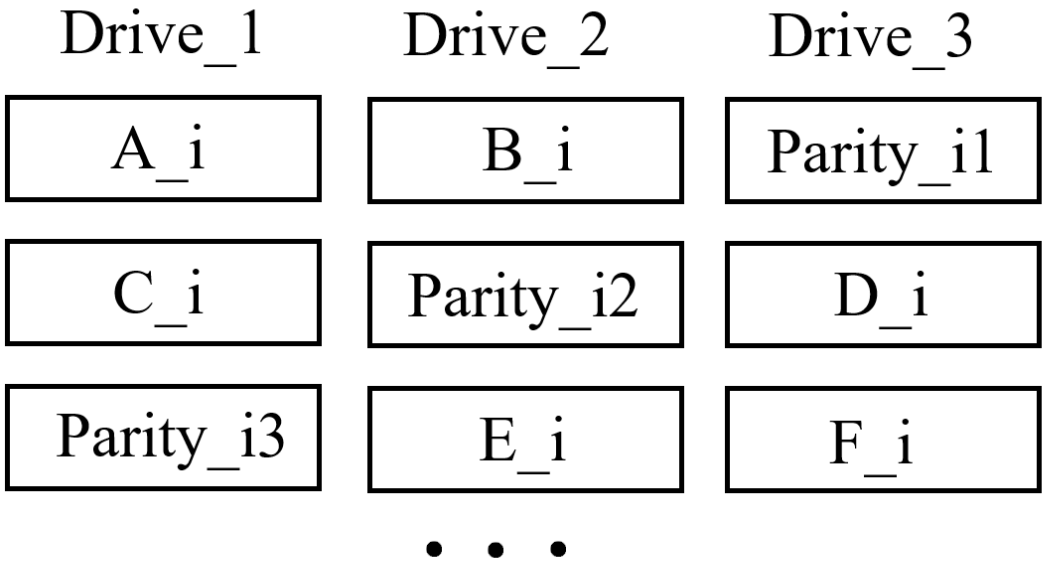

Рисунок 2 – Модель распределения данных

Таким образом, при выводе из строя одного из дисков, например, диска 2, все утерянные 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 большое количество. Потери в общем объеме хранилища будут составлять объем одного диска, при этом данные подлежат восстановлению при потере одного хранилища.

3 Разработка решения для обработки данных на облачных сервисах

3.1 Использованные технологии

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 инструментарий 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 разработке приложения был применён 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`
   - Скачивания 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`
- `CORS`

`Swagger` представляет собой набор инструментов для тестирования и

визуализации запросов [4]. 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, 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 [5].

Например, если HTML-страница с сервера http://domain-a.com запрашивает изображение <img src="http://domain-b.com/image.jpg», это будет считаться кросс-доменным запросом. Многие сайты загружают ресурсы (например, CSS, изображения, скрипты) с различных доменов, особенно при использовании CDN (Content Delivery Networks).

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.

CORS обеспечивает безопасный обмен данными между браузером и серверами при кросс-доменных запросах. Современные браузеры используют этот механизм в API (таких как XMLHttpRequest и Fetch), чтобы минимизировать риски, связанные с межсайтовыми запросами.

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 [6]. Пример использования продемонстрирован

на листинге ниже:

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

3.2 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 создается вспомогательная страница для приема токена [7].

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`, зна-

чительно расширяющий набор предоставляемых данных. Для реализации данного приложения были выбраны пункты чтение всего диска и запись 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 получения отладочного токена [8]. 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 `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 десериализация. 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.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
}
```

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
        .ReadAsStringAsync()).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
}
```

3.3 Dropbox

`Dropbox` является облачным сервисом, позволяющим получать данные пользователя и взаимодействовать с его хранилищем данных [9]. 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 [10].

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, позваляет 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 опевещение. По умолчанию значение False.
— property_groups — Необязательное поле, позволяет добавить файлу поль-

зовательские 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. Выполняется 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 дессериализуются 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();
}
```

3.4 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 [11]. 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 ведется 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 приложе-

ние. В случае, если при разработке запрашиваются данные ограниченной области действия, пользователей можно добавлять только напрямую через консоль клиента до подтверждения безопасности приложения [12]. 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 [13].

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 данные пользователя формируется экземпляр класса `GoogleClientSecrets`, при помощи

которого запускается клиент для авторизации пользователя и дальнейшего взаимодействия с диском [14]:

```
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 нем, 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 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();
```

4 Реализация класса, распределяющего данные среди облачных хранилищ по принципу 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 все расчёты будут произведены в одном потоке, ввиду чего был применён способ, в котором файл делится целиком на 6 равных частей и формируется 3 пары с массивом чётности для каждой. Данные массивы предоставляют гарантию восстановления данных в случае, если одно из облачных хранилищ недоступно. При многопоточной реализации задаётся фиксированный размер каждой части, пары формируются до тех пор, пока все байты файла не будут распределены и их общее количество не будет кратно 6.

На рисунке 3 продемонстрирован пример конкретной реализации распределения данных, представленной в программе.

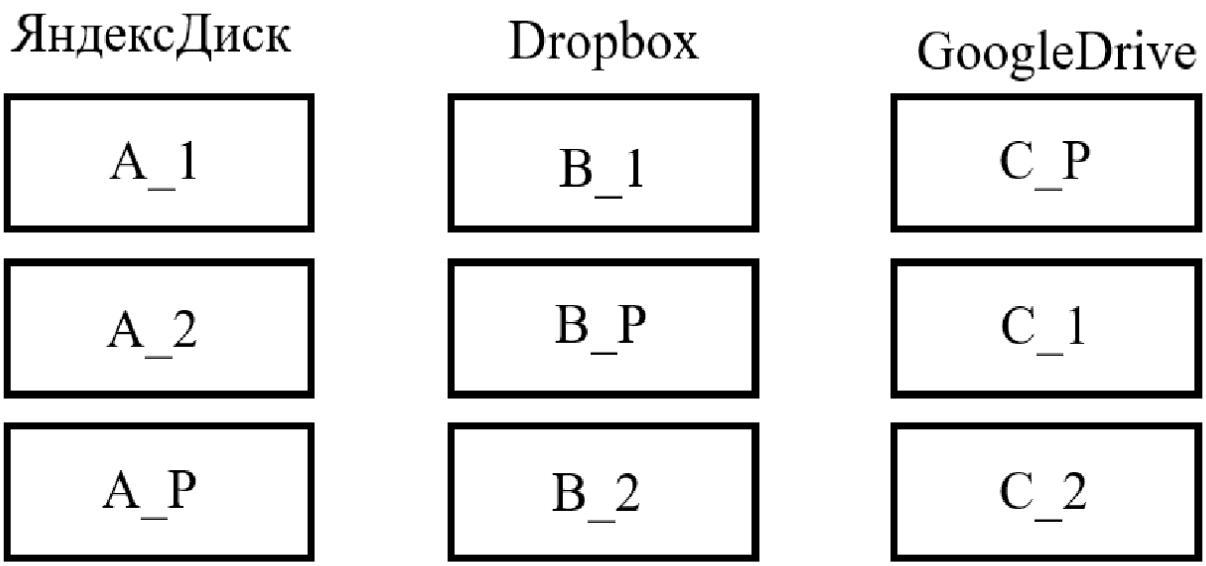

Рисунок 3 – Распределения данных среди облачных сервисов

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;
}
```

4.1 Загрузка файла с распределением

Для распределения файла по принципу `RAID 5` необходимо разбить его данные, представленные в виде набора байтов, на 6 равных частей. Для этого считывается выбранный файл и записывается в массив в виде набора байтов. Для разделения создаются 6 массивов и равномерно заполняются изначальными данными. Переменная `j` в данном случае является счетчиком для сохра-

нения позиции считывания из изначального массива. Пример предоставлен в листинге ниже:

```
byte[] fileContent = await File.ReadAllBytesAsync(filePath);
int j = 0;
byte[] data_1 = new byte[fileContent.Length / 6];
for(int i = 0; i < data_1.Length; i++)
{
    data_1[i] = fileContent[j];
    j++;
}
```

Данные массивы преобразуются в виде файлов, хранящих в себе набор подготовленных байтов, они в свою очередь загружаются на диск. Далее был создан массив для хранения текущей четности пары данных. Данный массив заполняется при отправке каждой пары элементов. 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:

```
byte[] parity = new byte[fileContent.Length / 6];
string dir = Directory.GetCurrentDirectory();
parity = SolveParity(data_1, data_2);
string fileName = Path.GetFileNameWithoutExtension(filePath);
string cur_fileName = fileName + "A_1";
await File.WriteAllBytesAsync(
    Path.Combine(dir, cur_fileName), data_1);
await _yandexdiskService.UploadFile(
    cur_fileName, Path.Combine(dir, cur_fileName));
File.Delete(Path.Combine(dir, cur_fileName));
cur_fileName = fileName + "B_1";
await File.WriteAllBytesAsync(
    Path.Combine(dir, cur_fileName), data_2);
await _dropboxService.UploadFile(
    Path.Combine(dir, cur_fileName), "/" + cur_fileName);
File.Delete(Path.Combine(dir, cur_fileName));
```

```
cur_fileName = fileName + "C_P";
await File.WriteAllBytesAsync(
    Path.Combine(dir, cur_fileName), parity);
```

Таким образом, производится загрузка 3 пар 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.

4.2 Считывание данных

Загрузка данных с дисков производится за счёт поиска на них всех 6 частей, 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:

```
string curFileName = fileName + "A_1";
byte[] data_1 = await _yandexdiskService.DownloadFile(curFileName);
byte[] data = new byte[data_1.Length * 6];
int j = 0;
for(int i = 0; i < data_1.Length; i++)
{
    data[j] = data_1[i];
    j++;
}
```

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 массива чётности для потерянных данных и производится операция $A \oplus Parity = B$ для восстановления недоступного элемента пары. Например, если недоступен Яндекс Диск и необходимо восстановить элемент `A_2` производится операция $B\_P \oplus C\_1 = A\_2$:

```
curFileName = fileName + "B_P";
byte[] data_3 = await _dropboxService.
    DownloadFile("/" + curFileName);
curFileName = fileName + "C_1";
await _googledriveService.DownloadFile(curFileName,
    Directory.GetCurrentDirectory() + "/" + curFileName);
byte[] data_4 = File.ReadAllBytes(
    Directory.GetCurrentDirectory() + "/" + curFileName);
data_3 = SolveParity(data_3, data_4);
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++;
}
```

5 Разработка пользовательского интерфейса

Пользовательский интерфейс был написан при помощи фреймворка Vue.Js. Он имеет встроенные инструменты 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 — Изначальное 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 4 продемонстрировано применение данного компонента на пользовательском интерфейсе в состоянии, когда он недоступен.

Рисунок 4 – Элемент button в неактивном состоянии с полем disabled

На рисунке 5 предоставлен элемент `button` при активном состоянии поля `disabled`.

Рисунок 5 – 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 6.

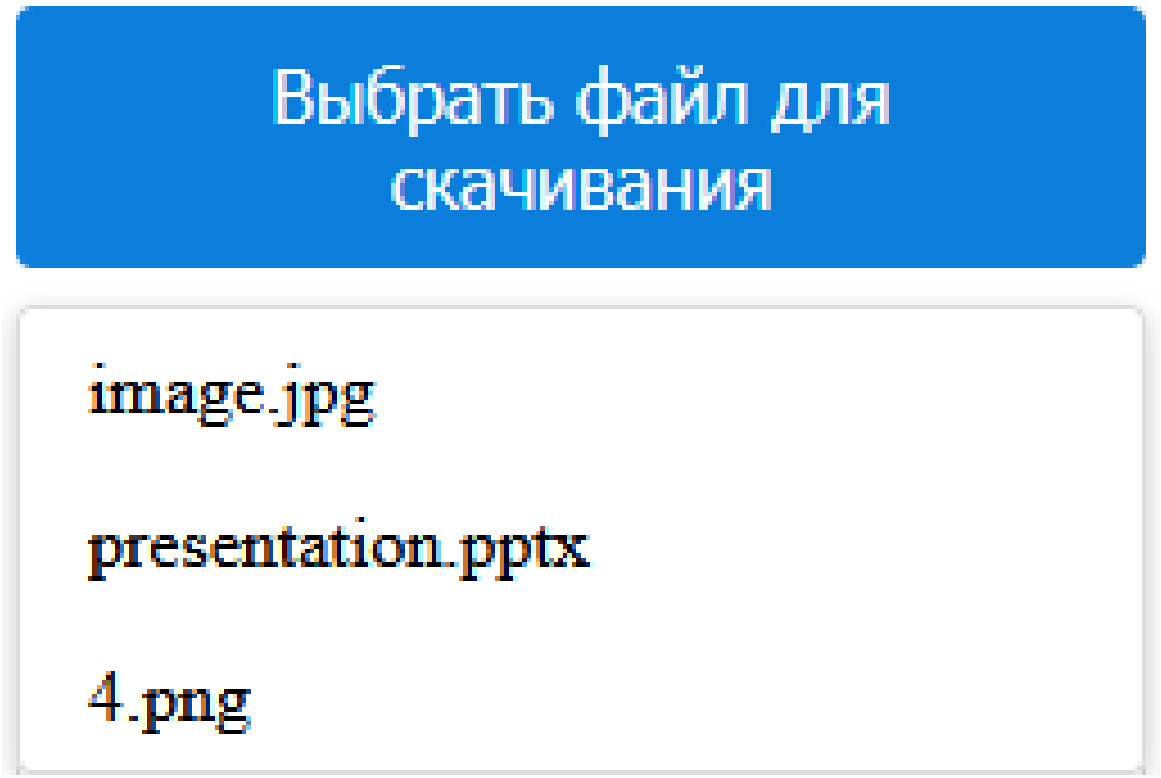

Рисунок 6 – Всплывающее окно выбор файла

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' }
]);
```

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 7.

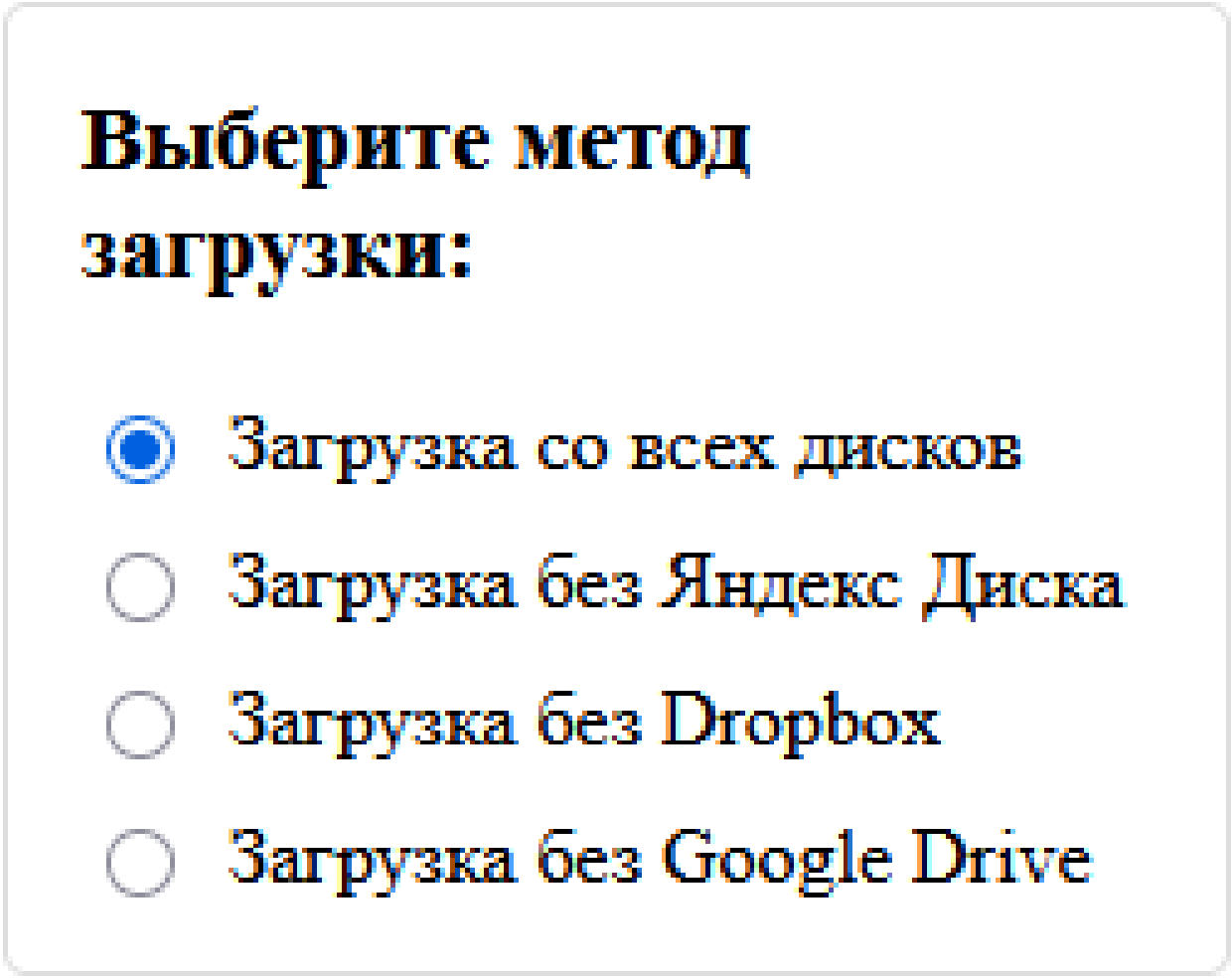

Рисунок 7 – Окно выбора метода загрузки

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();
}
```

Пример отображения всех файлов продемонстрирован на рисунке 8.

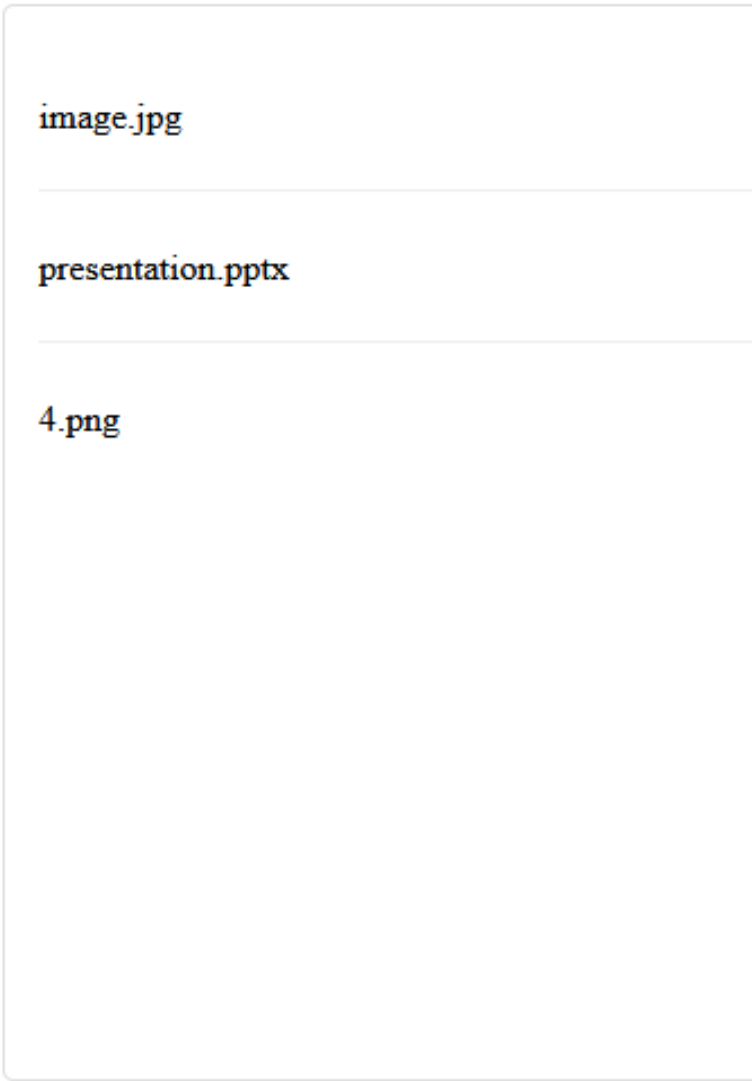

Рисунок 8 – Список доступных файлов

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;
}
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. Также за счёт поля указывается, что форматом передаваемых данных является `JSON`:

```
const response = await fetch(‘https://localhost:7229/api/
```

```
RAID5/upload/${encodeURIComponent(selectedLocalFile.value.path)}‘,
{
      method: ’POST’,
      headers: {
        ’Content-Type’: ’application/json’
      }
});
```

В случае возникновения конфликта во всплывающем окне отображается причина ошибки.

Пример полного интерфейса для работы с приложением предоставлен на рисунке 9.

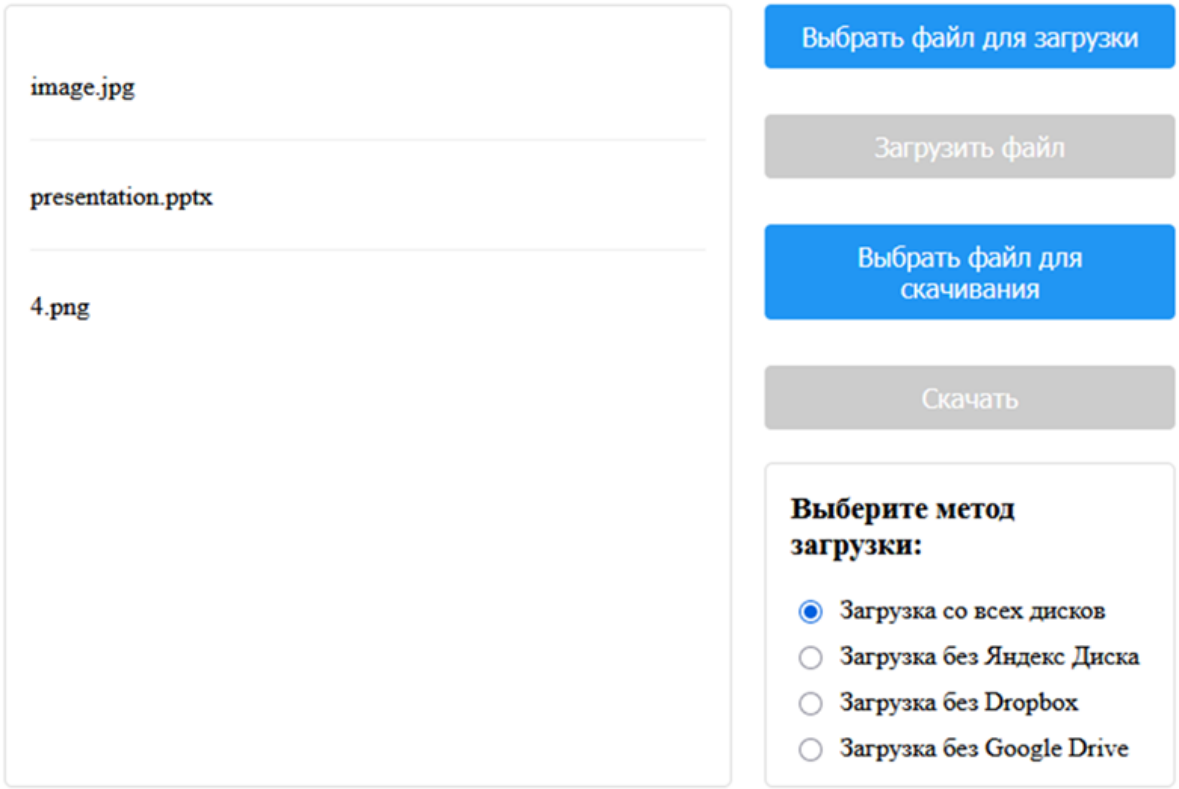

Рисунок 9 – Полная версия интерфейса

При загрузке файла picture.png на облачное хранилище он представляется в виде файлов, хранящих в себе распределённые по облачным сервисам наборы байт. Результат выполнения данной операции, в виде хранящихся данных на дисках представлен на рисунке 10.

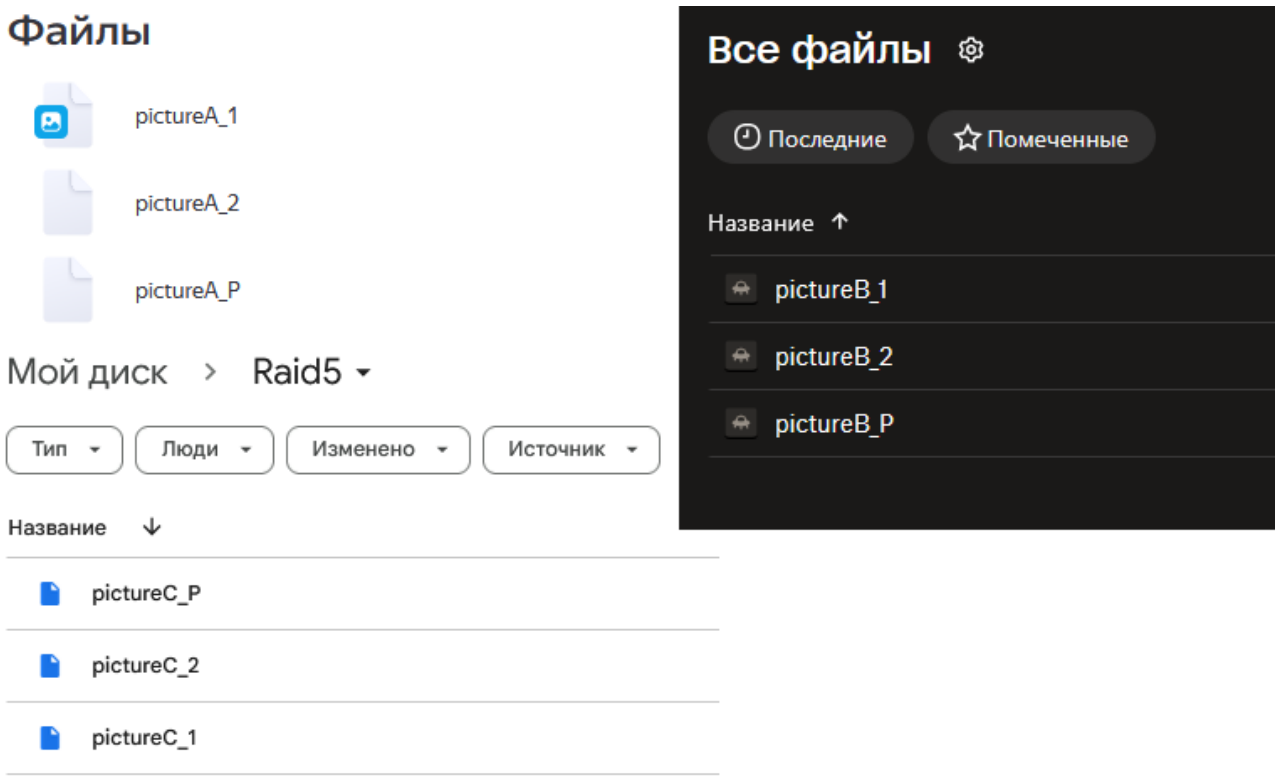

Рисунок 10 – Результат загрузки файла

Скачивание файла из облачного хранилища производится путем преобразования хранимых файлов в виде набора байт и создания из них первоначального файла. Для восстановления файла производится расчёт утерянных байтов при помощи массивов чётности. Результат выполнения данной операции представлен на рисунке 11.

Рисунок 11 – Результат скачивания файла с восстановлением данных

Полный код программной реализации предоставлен в Приложениях А, Б, В, Г, Д, Е, Ж, З.

ЗАКЛЮЧЕНИЕ

В рамках данной бакалаврской работы было разработано приложение реализующее распределение файлов в виде RAID массива 5 уровня на облачных хранилищах. Приложение реализовано при помощи фреймворков ASP.NET Core, Vue3.js а также технологий HTML, CSS и JavaScript.

В процессе разработки было разработано решение для взаимодействия с облачными сервисами, рассмотрен принцип работы и реализации RAID5. Реализация для обработки данных представленна на ряде облачных хранилищ:

— Яндекс Диск,
— Dropbox,
— GoogleDrive.

Проведённые тесты подтвердили работоспособность системы: приложение корректно распределяет данные между облачными сервисами, обеспечивает их целостность и позволяет восстанавливать файлы в случае недоступности одного из хранилищ.

Итоговое приложение представляет собой интерфейс для загрузки файлов на облачные хранилища с распределением по принципу RAID5. Также оно позволяет скачивать все файлы при помощи применения всех трех дисков, либо с учетом потери одного из них.

Разработанное приложение имеет потенциал к дальнейшему улучшению и практическому применению, так как позволяет хранить данные с повышенной отказоустойчивостью с сохранением преимуществ облачных сервисов.

Таким образом, в рамках работы создано решение, сочетающее преимущества облачного хранения и отказоустойчивости RAID 5, что делает его актуальным инструментом для безопасного и надёжного управления данными.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Chen P.M., Lee E.K., Gibson G.A., Katz R.H., Patterson D.A. 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. – 185 p.

2 Patterson D.A., Gibson G., Katz R.H. A case for redundant arrays of inexpensive disks (RAID) / D.A. Patterson, G. Gibson, R.H. Katz. – ACM SIGMOD Record, 1988г. – 116 p.

3 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.

4 Swager Docs [Электронный ресурс] URL: https://swagger.io/docs/ (дата обращения: 25.01.2025)

5 Cross-Origin Resource Sharing (CORS) [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/CORS (дата обращения: 03.02.2025)

6 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)

7 Яндекс ID [Электронный ресурс] URL: https://yandex.ru/dev/id/doc/ru/register-client (дата обращения: 13.02.2025)

8 Получение OAuth-токена [Электронный ресурс] URL: https://yandex.ru/dev/id/doc/ru/access (дата обращения: 13.02.2025)

9 Dropbox Documentation [Электронный ресурс] URL: https://www.dropbox.com/developers/documentation?_tk=pilot_lp&_ad=topbar1&_camp=docs (дата обращения: 20.02.2025)

10 Dropbox OAuth guide [Электронный ресурс] URL: https://www.dropbox.com/lp/developers/reference/oauth-guide.html (дата обращения: 22.02.2025)

11 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)

12 OAuth 2.0 Scopes for Google APIs [Электронный ресурс] URL: https://developers.google.com/identity/protocols/oauth2/scopes (дата обращения: 04.03.2025)

13 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)

14 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)

15 Rendering Mechanism [Электронный ресурс] URL: https://vuejs.org/guide/extras/rendering-mechanism (дата обращения: 20.03.2025)

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)