Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Программное средство «ОБЛАЧНОЕ ХРАНИЛИЩЕ»

|  |  |  |
| --- | --- | --- |
| Студент |  | И.В. Ермолович |
| Руководитель |  | Д.В. Низовцов |

Минск 2021

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc71651966)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc71651967)

[1.1 Анализ литературных источников 7](#_Toc71651968)

[1.2 Обзор аналогов 7](#_Toc71651969)

[1.3 Постановка задачи 10](#_Toc71651970)

[2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 12](#_Toc71651971)

[2.1 Структура программы 12](#_Toc71651972)

[2.2 Описание функциональности программного средства. 14](#_Toc71651973)

[2.3 Работа сервера 23](#_Toc71651974)

[2.4 Работа клиента 25](#_Toc71651975)

[3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 27](#_Toc71651976)

[3.1 Взаимодействие с приложением 27](#_Toc71651977)

[ЗАКЛЮЧЕНИЕ 28](#_Toc71651978)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc71651979)

[ПРИЛОЖЕНИЕ А. 30](#_Toc71651981)

[Исходный код программы 30](#_Toc71651982)

# ВВЕДЕНИЕ

В настоящий момент времени почти не осталось людей, не связанных с миром компьютерных технологий. Почти каждый человек сталкивается с работой на ПК в разных жизненных ситуациях.

Существует несколько проблем, связанных с хранением данных на жестком диске компьютера, например:

* Поломка компьютера и, как следствие, потеря доступа к данным;
* Работа на двух компьютерах, например дома и на работе, постоянно приходиться прибегать к различным переносным устройствам информации;
* Работа в большой команде, где, в идеале, доступ к данным должен быть у каждого члена команды.

Все эти проблема решаются введением в процесс работы с ПК такой технологии, как облачное хранилище.

Облачное хранилище данных — модель онлайн-хранилища, в котором данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиентам, в основном, третьей стороной. В отличие от модели хранения данных на собственных выделенных серверах, приобретаемых или арендуемых специально для подобных целей, количество или какая-либо внутренняя структура серверов клиенту, в общем случае, не видна. Данные хранятся и обрабатываются в так называемом облаке, которое представляет собой, с точки зрения клиента, один большой виртуальный сервер. Физически же такие серверы могут располагаться удалённо друг от друга географически.

Преимущества облачного хранилища:

* Возможность доступа к данным из любого компьютера, имеющего выход в интернет.
* Возможность организации совместной работы с данными.
* Высокая вероятность сохранения данных даже в случае аппаратных сбоев.
* Клиент платит только за то место в хранилище, которое фактически использует, но не за аренду сервера, все ресурсы которого он может и не использовать.
* Клиенту нет необходимости заниматься приобретением, поддержкой и обслуживанием собственной инфраструктуры по хранению данных, что, в конечном счёте, уменьшает общие издержки производства.
* Все процедуры по резервированию и сохранению целостности данных предоставляются провайдером облачного центра, который не вовлекает в этот процесс клиента.

Однако существуют и недостатки. В марте 2011 года аналитическая фирма IDC провела исследование, которое показало, что многие компании в первую очередь связывают с облачными сервисами большие проблемы по части безопасности. А независимая исследовательская организация Portio Research только подтвердила это, указав конкретные цифры: 68 % опрошенных руководителей европейских IT-компаний в целях безопасности отказываются использовать облачные технологии. Также руководитель компании G Data Security Labs и эксперт по безопасности Ральф Бенцмюллер не рекомендовал использовать облачные сервисы в связи с тем, что данные могут стать достоянием общественности.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Анализ литературных источников

Основных используемым литературным источником в данной курсовой работе является "Node.js. Разработка серверных веб-приложений на JavaScript" Хэррона Дэвида [1]. Книга учит создавать приложения при помощи Node JS. В ней рассказывается, как использовать серверные и клиентские объекты HTTP, Express и как организовывать работы баз данных на MongoDB и SQL. Чтение руководства предполагает наличие базовых знаний и использование веб-разработок и языка программирования. Практические рекомендации позволят научиться запускать и настраивать Node js и на его основе в натуральном режиме создавать клиентские и серверные HTTP-приложения вне браузера.

Следующим используемым источником является книга "Angular для профессионалов" Адама Фримена [2]. Адам Фримен начинает с описания MVC и его преимуществ, затем показывает, как эффективно использовать Angular, охватывая все этапы, начиная с основ и до самых передовых возможностей, которые кроются в глубинах этого фреймворка. Каждая тема изложена четко и лаконично, снабжена большим количеством подробностей, которые позволят вам стать действительно эффективными. Наиболее важные фичи даны без излишних подробностей, но содержат всю необходимую информацию, чтобы вы смогли обойти все подводные камни.

## 1.2 Обзор аналогов

В данной теме существуют огромное количество аналогов, но рассмотрим только наиболее схожие и популярные приложения.

Первый из рассматриваемых аналогов − Microsoft One Drive, внешний вид которого продемонстрирован на рисунке 1.1.

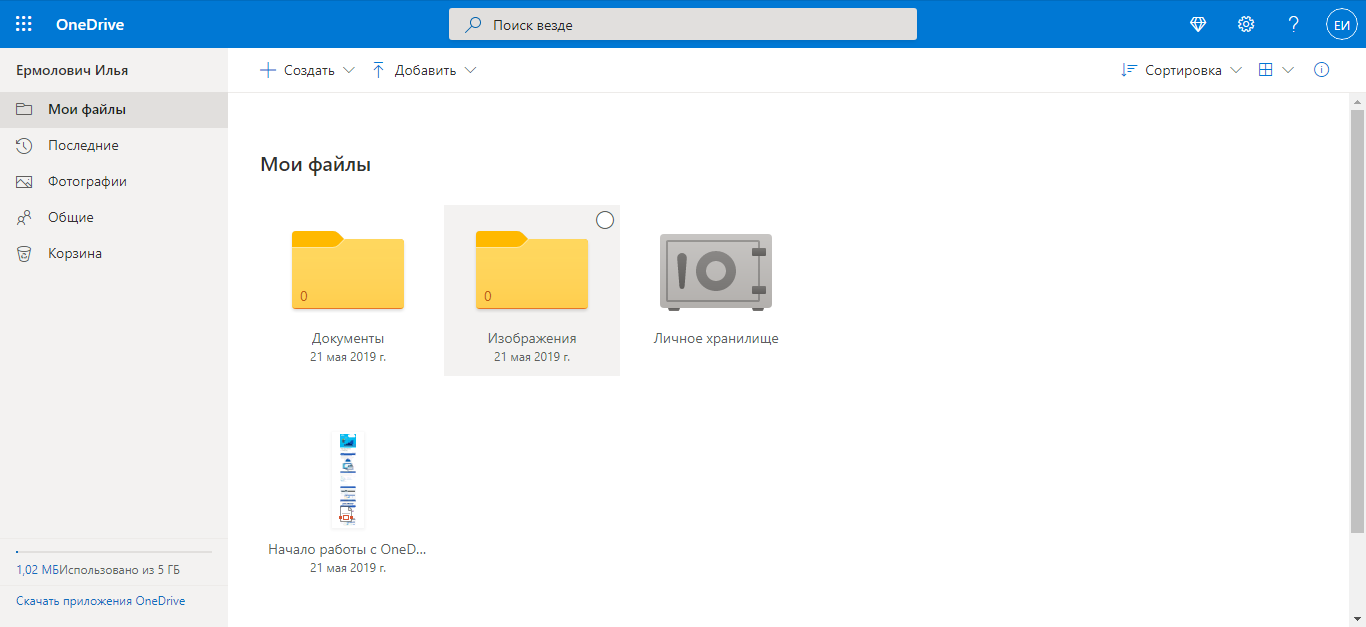


Рисунок 1.1 – программное средство «One Drive»

Данное приложение имеет несколько преимуществ:

* Офисный пакет в режиме онлайн;
* Приятный пользовательский интерфейс.

Однако у данного решения все же таки есть собственные недостатки:

* по умолчанию доступно 5 гб памяти бесплатно;
* для использования хранилища необходимо иметь учетную запись Microsoft Live;
* навязчивость, устанавливается при установке ОС Windows, тяжело удаляется;
* низкая скорость.

Данный аналог явно является не самым лучшим представителем облачных хранилищ, что свидетельствует о его непопулярности.

Следующим облачным хранилищем на очереди является Google One. Интерфейс данного программного средства представлен на рисунке 1.2

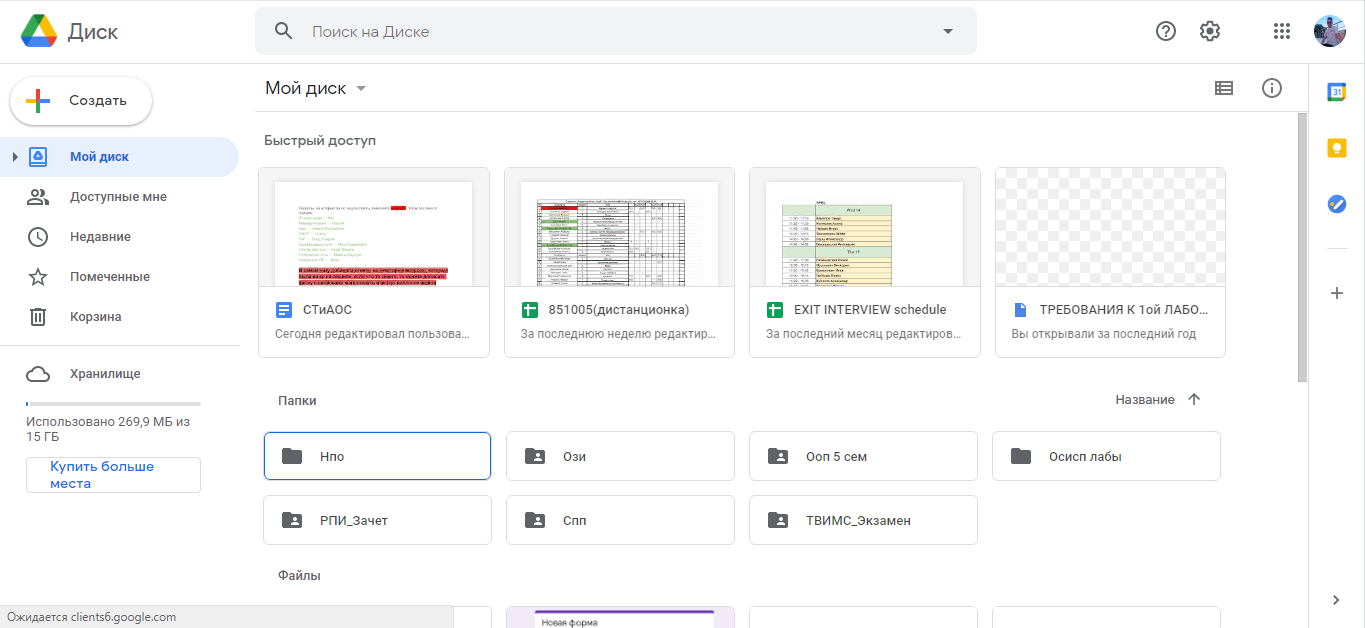


Рисунок 1.2 – программное средство «Google One»

Данное программное обеспечение является самым популярным в мире. Из преимуществ перед другими аналогами можно выделить:

* 15гб бесплатной памяти;
* собственные офисные программы, которые работают прямо в онлайн;
* лучшая для облачных хранилищ совместная работа с файлами, особенно это касается офисных приложений;
* кросс-платформенность;
* проиграть видео файл онлайн возможно прямо из облачного пространства, даже в максимальном качестве, как будто вы просто смотрите видео с YouTube.

Хоть данное программное средство и является хорошим вариантом для использования, однако оно не лишено недостатков, а именно:

* средняя скорость;
* не самый удобный пользовательский интерфейс.

Данный аналог, без сомнений, является одним из лучших облачных

сервисов в мире.

Еще один аналог рассматриваемого приложения − облачное хранилище "Яндекс Диск". Интерфейс данного программного средства представлен на рисунке 1.3.

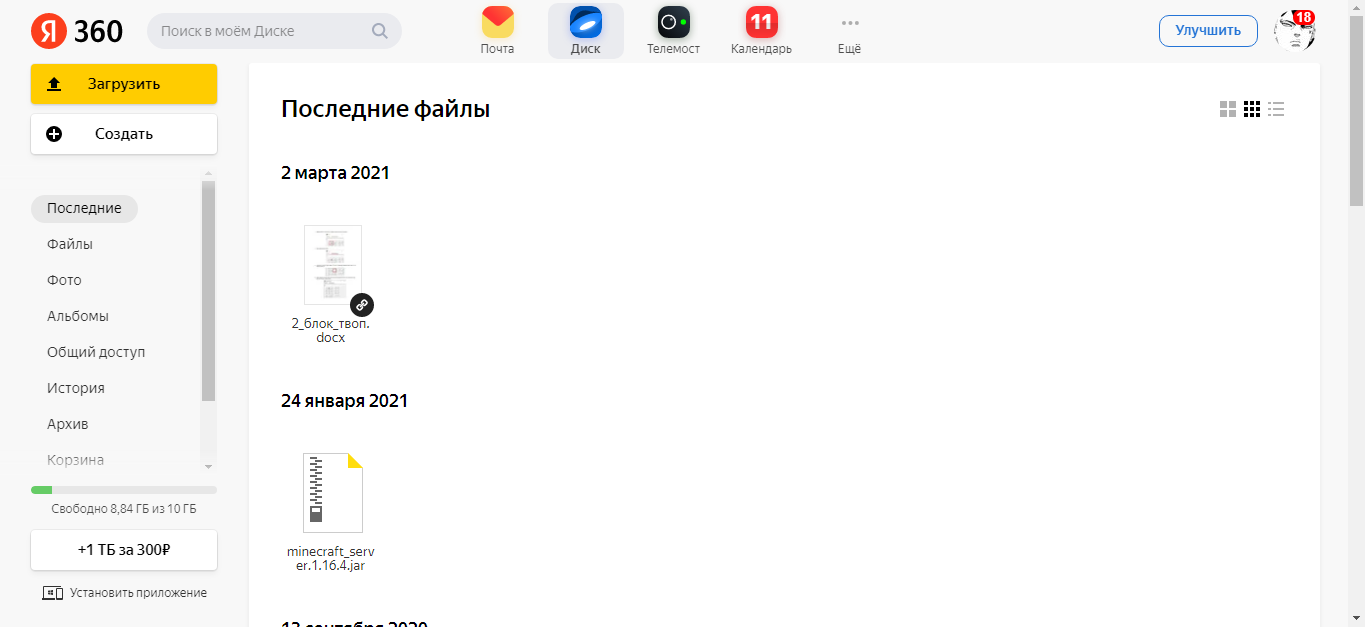


Рисунок 1.3 – программное средство «Яндекс Диск»

Данное программное средство разработано российской компанией "Яндекс", которую сравнивают, как "русский" аналог всемирно известной компании Google. Поэтому данный аналог взял многое у предыдущего, как преимущества, так и недостатки.

Рассматриваемое приложение имеет ряд плюсов:

* 10гб бесплатной памяти, а также расширение памяти путем различных акций, подарков и т.п.;
* отличная скорость;
* все файлы в хранилище, размером до 1гб, проверяются антивирусной программой Dr.Web;
* максимальный размер загружаемого файла – 50гб;
* есть возможность совместной работы с файлами.

Однако данное приложение не лишено и минусов:

* приложение для Windows очень громоздкое и неоптимизированное, тратит слишком много ресурсов ПК;
* нет возможности ограничения скорости;
* "корзина" входит в бесплатную предоставляемую память;
* очистка "корзины" происходит очень долго;

## Постановка задачи

В рамках данного курсового проекта планируется реализовать программное средство «Облачное хранилище».

В процессе реализации данного приложения планируется создать веб- приложение, которое будет построено на клиент-серверной архитектуре,

Таким образом, в ходе разработки данного программного средства планируется решить следующие задачи:

* загрузка файлов в хранилище;
* создание папок в хранилище;
* возможность скачивать файлы с хранилища;
* возможность скачивать папки с хранилища;
* логин и регистрация пользователей.

Для разработки клиентской части программного средства будет использоваться фреймворк Angular с библиотекой Angular Material. Для разработки серверной части программного средства будет использоваться язык программирования JavaScript с средой выполнения Node.js, библиотека Express для построения Http-сервера.

# 2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

Структура приложения разбита на две части: клиентскую и серверную, рисунок 2.1.

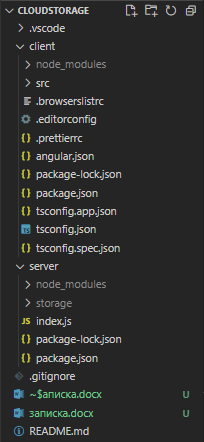


Рисунок 2.1 – Структура программного средства "Облачное хранилище"

Серверная часть приложения состоит из:

* папка node\_modules, содержит используемые npm-модули для работы сервера:
  + archiver – используется для архивации папок перед отправкой клиенту;
  + body-parser – парсит тело post-запроса;
  + cors – используется для настройки кросс-доменных запросов
  + express – используется для создания сервера;
  + express-fileupload – используется для загрузки файлов на сервер;
  + nodemon – используется для удобства отладки сервера;
  + path – используется для комбинирования строк, представляющих из себя пути;
  + rimraf – используется для удаления папок с правами администратора.
* папка storage, является непосредственным хранилищем, внутри содержаться папки каждого пользователя по его имени, внутри которых файлы каждого пользователя;
* index.js, является исполняемым файлом сервера, в нем описана работа сервера и все соответствующие запросы;

Клиентская часть приложения представляет из себя Angular-проект, рассмотрим только основные модули, компоненты и сервисы:

* shared.module, модуль, содержащий общие компоненты приложения:
  + header.component – компонент, представляющий шапку приложения;
  + error.component – компонент, представляющий диалог ошибки в приложении.
* storage.module, модуль, содержащий основные компоненты приложения:
  + auth.component – компонент, представляющий форму авторизации в приложении;
  + list.component – компонент, представляющий страницу работы с хранилищем;
  + delete.component – компонент, представляющий диалог удаления содержимого из хранилища;
* вспомогательные классы и сервисы модуля services.module:
  + user.service – сервис для работы с авторизацией пользователей;
  + storage.service – сервис для работы с хранилищем;
  + pathHelper.ts – вспомогательный класс, хранящий путь по папкам внутри хранилища;
  + progressHelper.ts – вспомогательный класс, предоставляющий прогресс загрузки файла на сервер;
  + subscriptions.service – сервис для отписки всех подписок на запросы к серверу;
  + storage.model.ts – модель элемента хранилища;

## 2.2 Описание функциональности программного средства.

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому взаимодействие приложения с пользователем необходимо организовать максимально интуитивно и просто.

При запуске приложения показывается страница логина, которая продемонстрирована на рисунке 2.2. На ней содержаться поля для ввода логина и пароля, а также кнопка логина и регистрации. При клике на кнопку логина, произойдет отправка запроса на сервер с введенными логином и паролем.

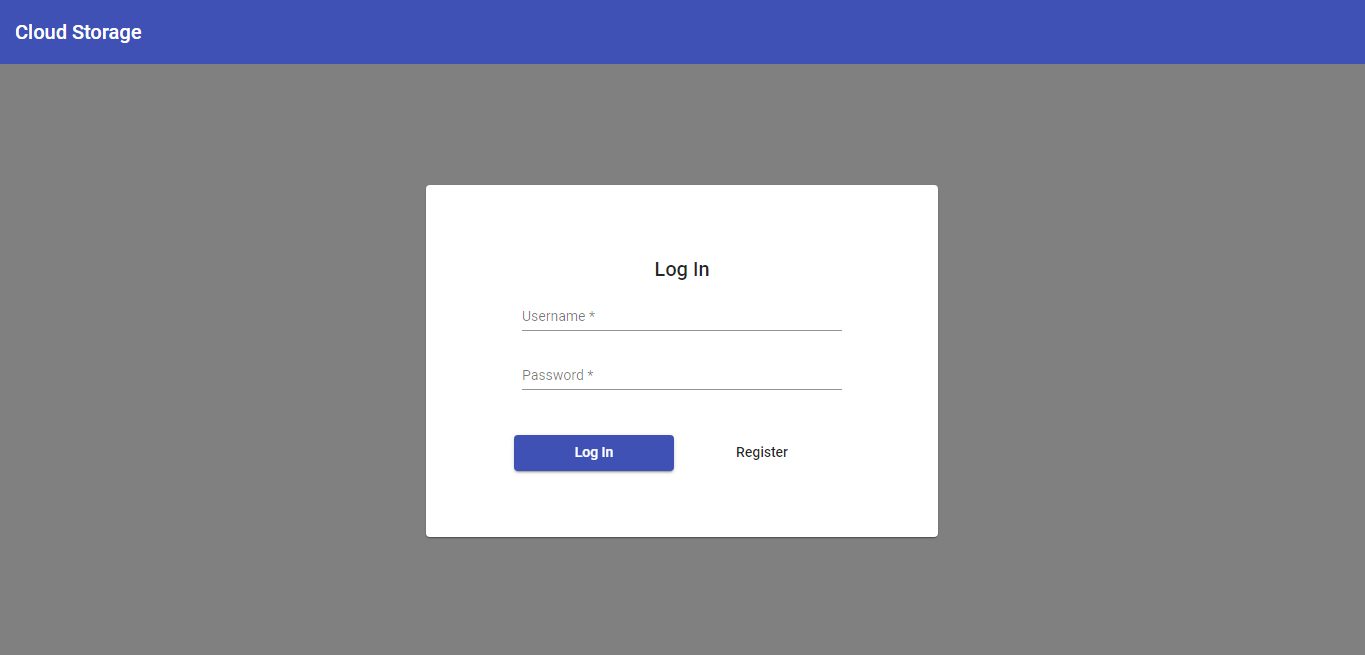


Рисунок 2.2 – Страница логина

При клике на кнопку регистрации будет открыта форма регистрации, рисунок 2.3. Форма регистрации содержит все те же самые элементы, только добавляется поле для ввода подтверждения пароля. При клике на кнопку регистрации, произойдет отправка запроса на сервер с введенными данными. При успешном запросе, откроется страница логина.

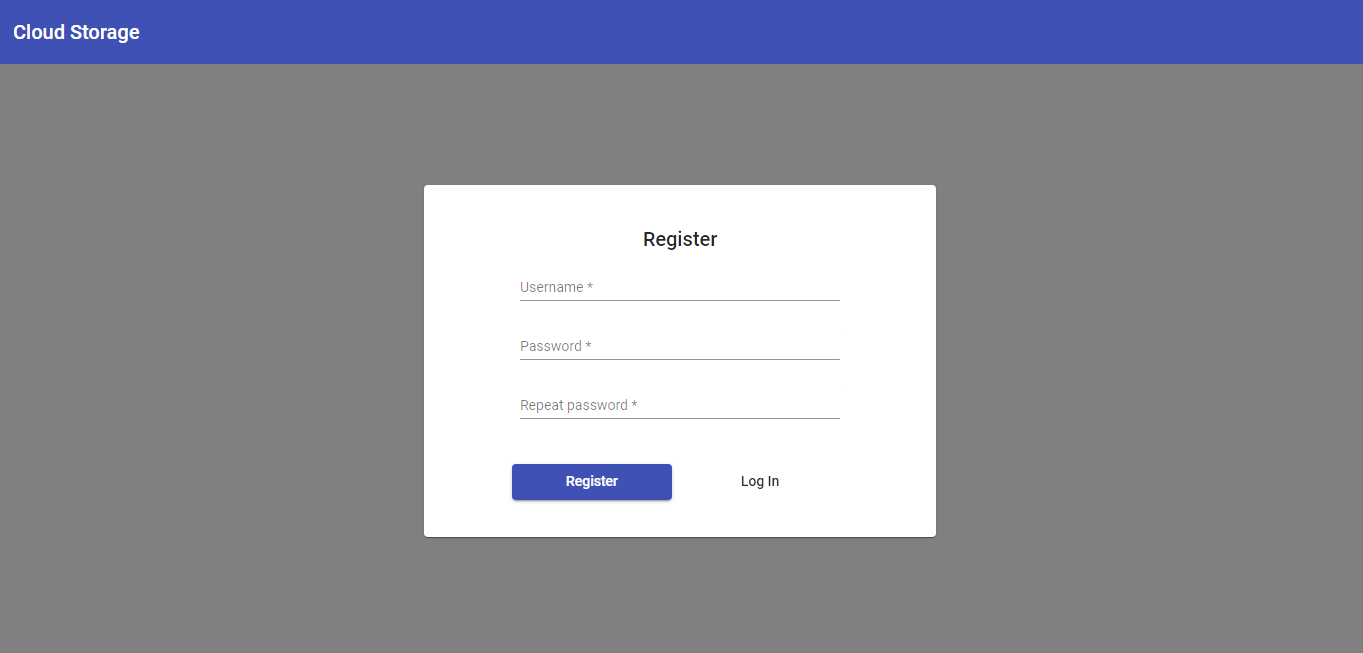


Рисунок 2.3 – Страница регистрации

Если что-то произойдет не так при попытке логина или регистрации, появиться соответствующее поле с описанием ошибки, рисунок 2.4. Ошибки могут произойти при неправильном вводе логина или пароля при логине, при несовпадении пароля и подтверждения пароля при регистрации, при занятости соответствующего логина при регистрации.

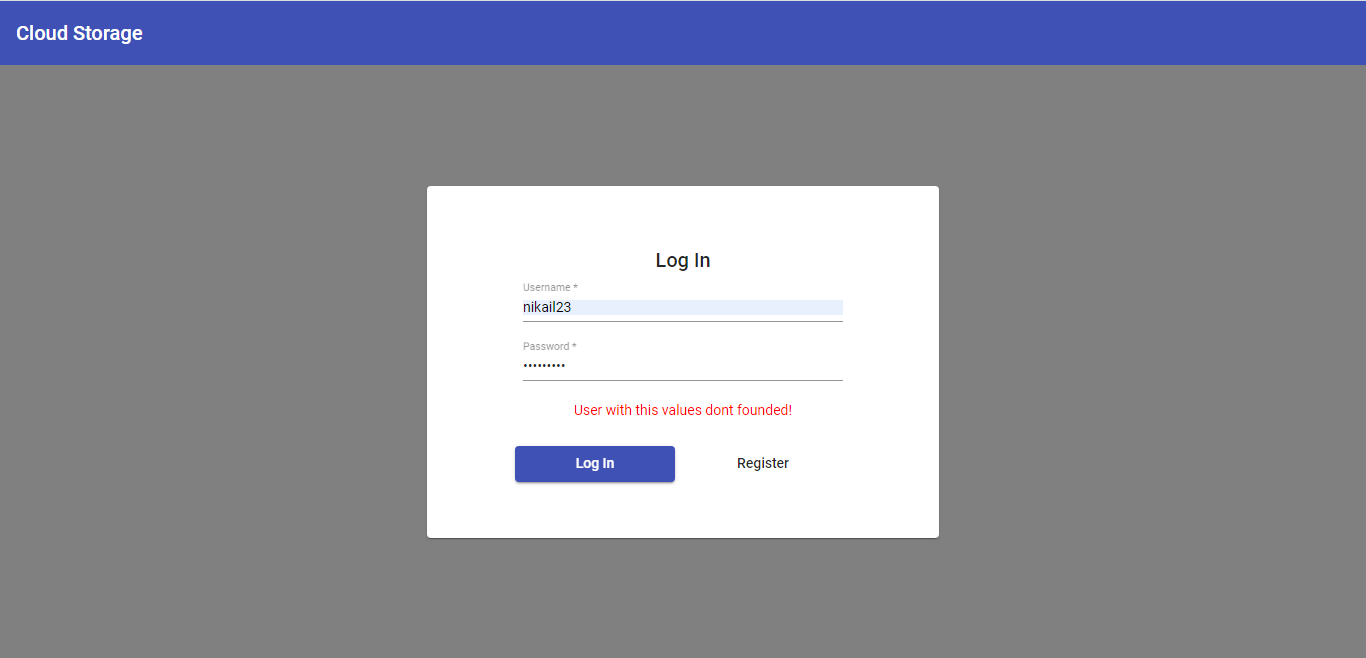


Рисунок 2.4 – Поле ошибки

При успешном логине произойдет переход на главную страницу приложения, рисунок 2.5. Главная страница приложения содержит список файлов и папок, находящихся в хранилище, а также выпадающий список с возможными действиями в хранилище.



Рисунок 2.5 – Главная страница приложения

Рассмотрим основную функциональность приложения, загрузим файл в хранилище. Для этого нажмем на кнопку "Load file" в выпадающем меню слева. Появиться стандартное окно выбора файла, рисунок 2.6.

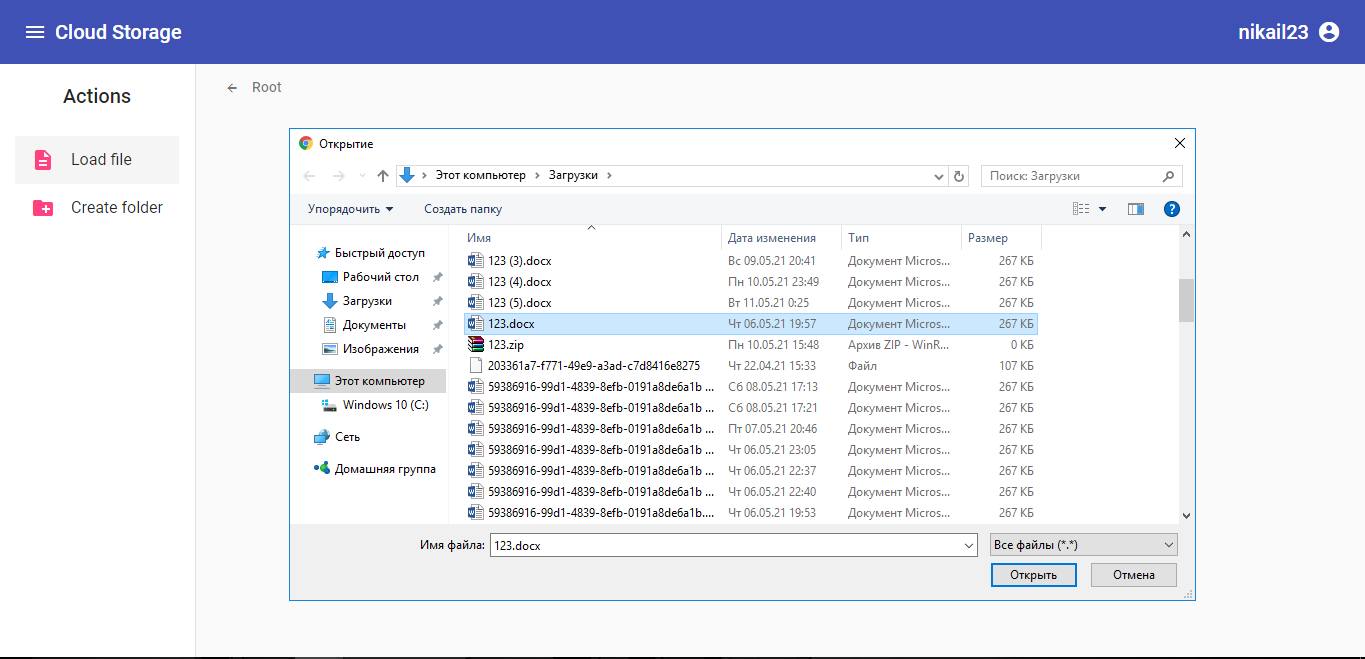


Рисунок 2.6 – Окно выбора файла

Выберем для примера файл "123.docx". После выбора файла произойдет отправка запроса на сервер с загрузкой этого файла. Загрузка файла сопровождается с отображением прогресса, в итоге в списке появиться загруженный файл и отобразиться полоса прогресса, рисунок 2.7. В списке загруженный файл отображается вместе с его именем, размером и датой загрузки в хранилище, а также с кнопка скачивания и удаления.

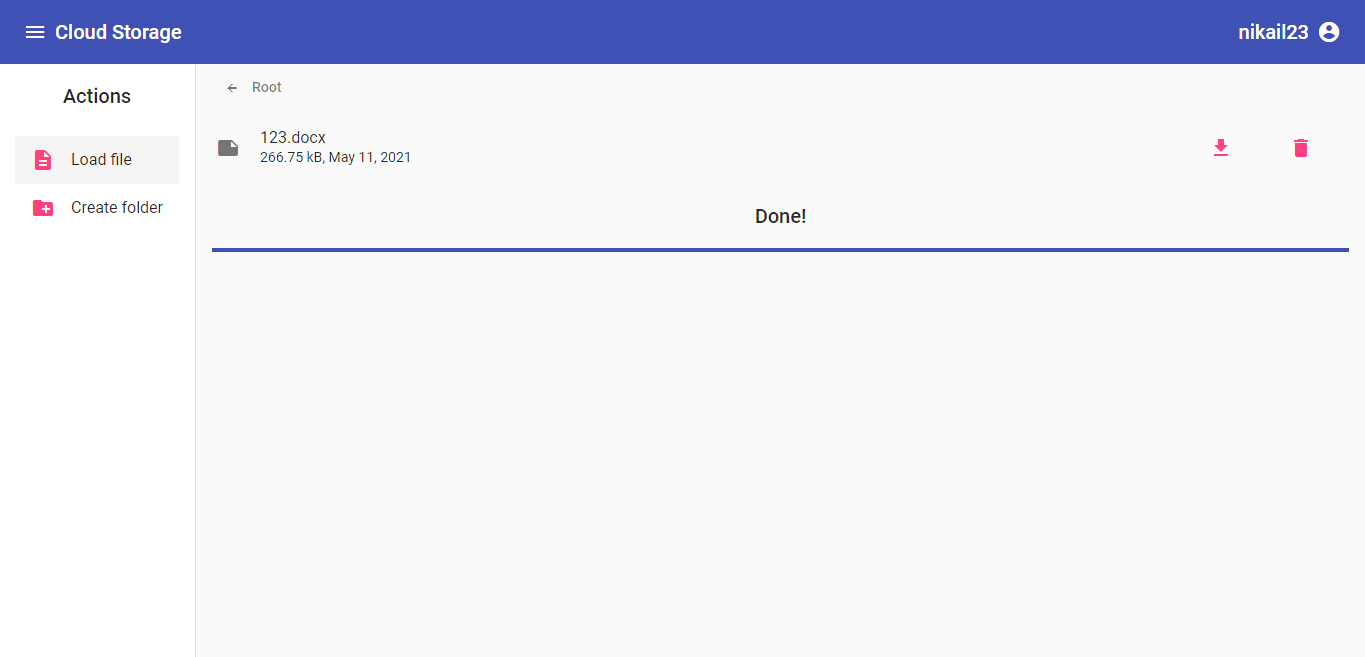


Рисунок 2.7 – Результат загрузки файла

При этом на сервере в папке "storage" будет создана папка "nikail23", соответствующая пользователю, с которого произведена авторизация. В этой папке появиться файл "123.docx", рисунок 2.8.

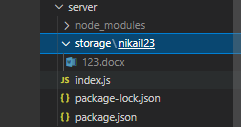


Рисунок 2.8 – Результат загрузки файла в хранилище

Попробуем теперь создать папку на сервере и загрузить в неё файл. Для этого в левом выпадающем меню нажмем на кнопку "Create folder". В списке появиться элемент для ввода имени папки и подтверждения создания или отмены, рисунок 2.9.

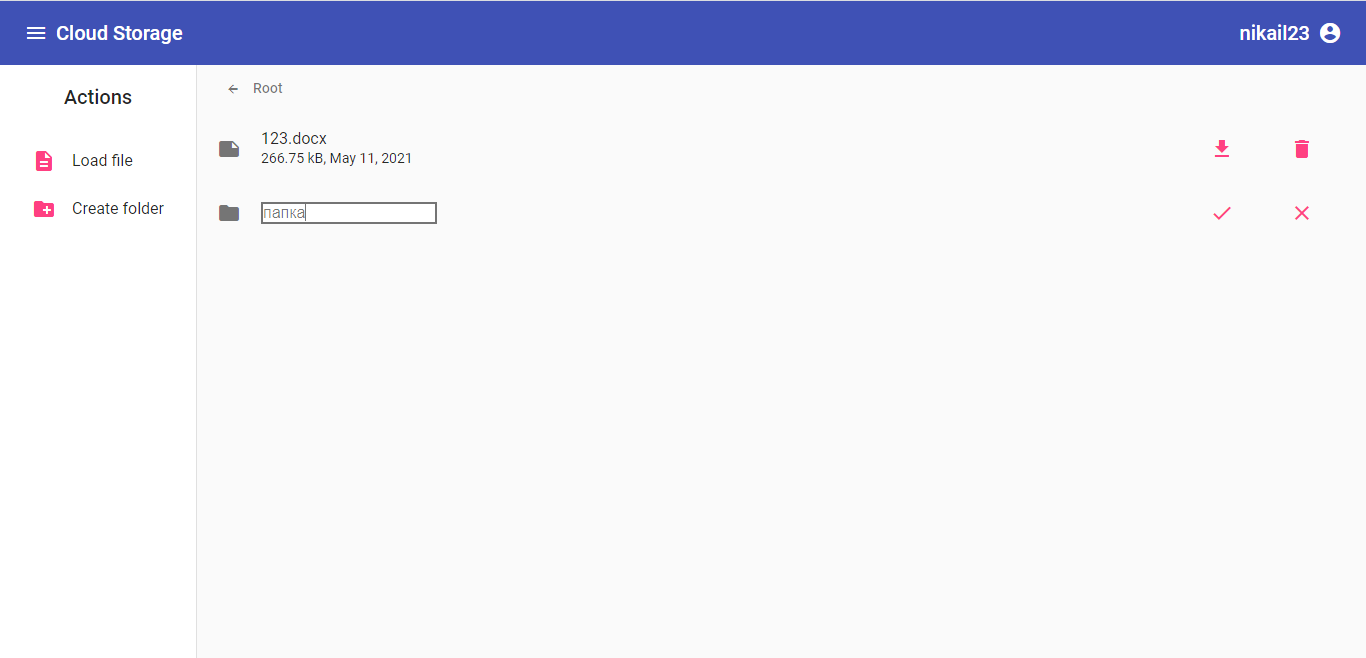


Рисунок 2.9 – Элемент создания папки

Введем в поле имя папки как "папка" и нажмем на кнопку подтверждения создания папки справа. В результате папка отобразиться в хранилище, рисунок 2.10. При клике на кнопку отмены создания папки, элемент создания папки пропадет.

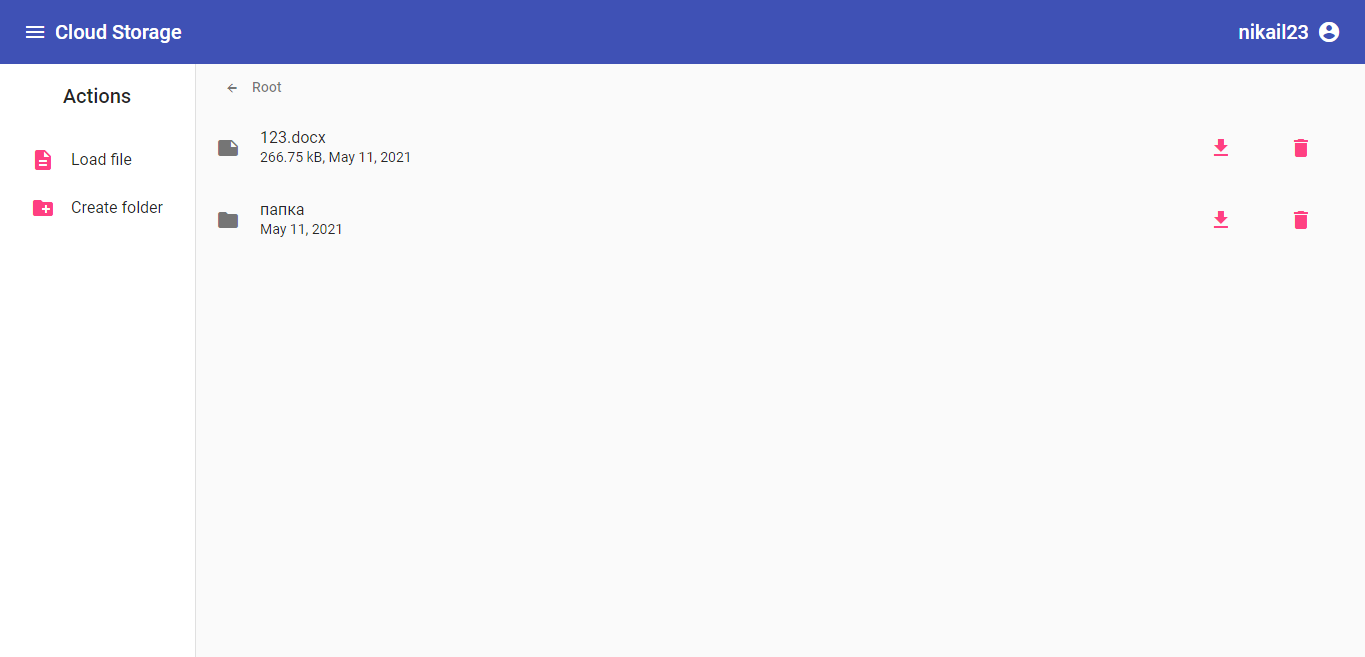


Рисунок 2.10 – Результат создания папки

Далее, чтобы загрузить файл в эту папку, нам необходимо в неё перейти. Чтобы перейти в папку, необходимо дважды кликнуть по ней в списке. В результате путь сверху станет равным "Root/папка" и список станет пустым, т.к. в папку мы еще ничего не загрузили, рисунок 2.11.



Рисунок 2.11 – Результат перехода в папку "папка"

Затем повторим действия с загрузкой файла, в результате в списке появиться новый файл, рисунок 2.12.

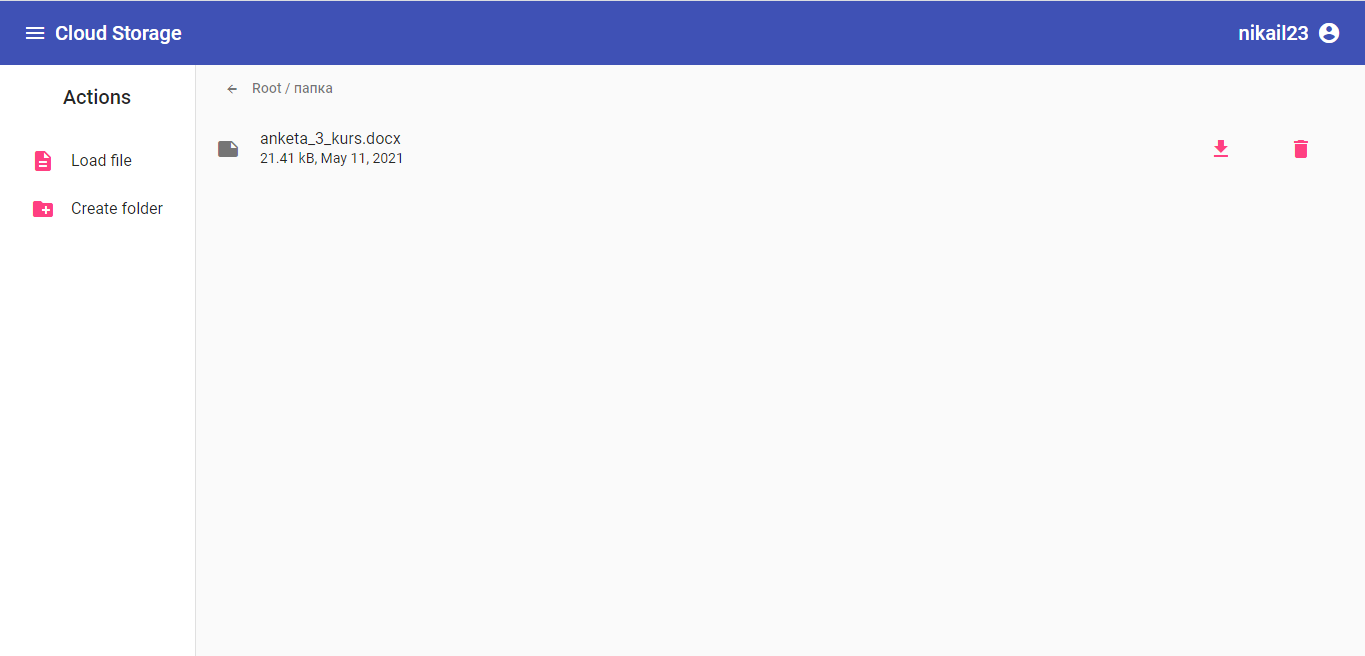


Рисунок 2.12 – Результат загрузки файла в папку "папка"

При этом на сервере мы должны увидеть, что создалась папка "папка" и в неё был загружен файл, рисунок 2.13.

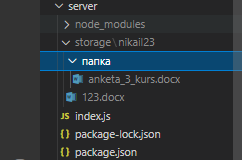


Рисунок 2.13 – Результат создания папки "папка" и загрузки в неё файла в хранилище

Чтобы выйти из соответствующей папки, необходимо нажать на кнопку "🡨", расположенную слева от отображаемого пути. В результате будет совершен переход по хранилищу на один уровень выше.

Рассмотрим функцию скачивания файла. Нажмем на кнопку скачивания файла в файле "123.docx". В результате произойдет скачивание файла в папку "Downloads", рисунок 2.14.

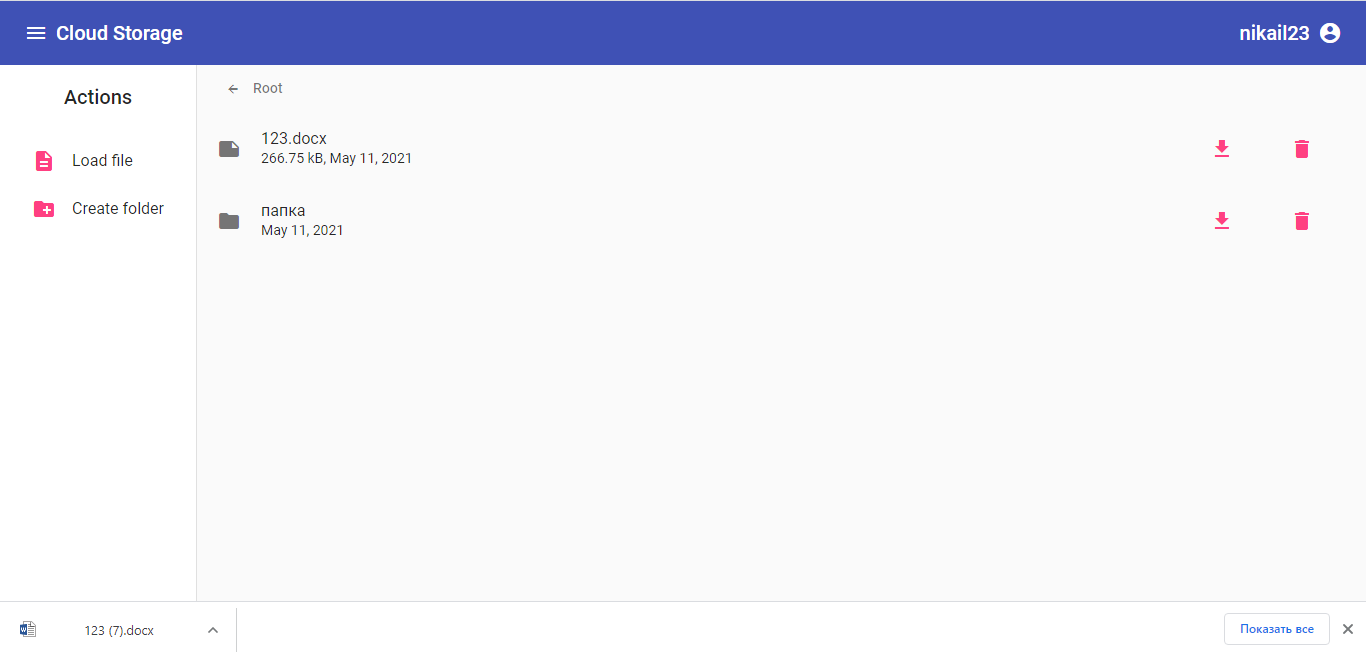


Рисунок 2.14 – Результат скачивания файла

Также мы можем скачать целую папку со всем содержимым внутри, для этого кликнем на соответствующую кнопку скачивания, только для папки "папка". В результат будет скачен архив "папка.zip" в папку "Downloads", рисунок 2.15. Архив полностью сохраняет структуру и содержимое скачиваемой папки, рисунок 2.16.

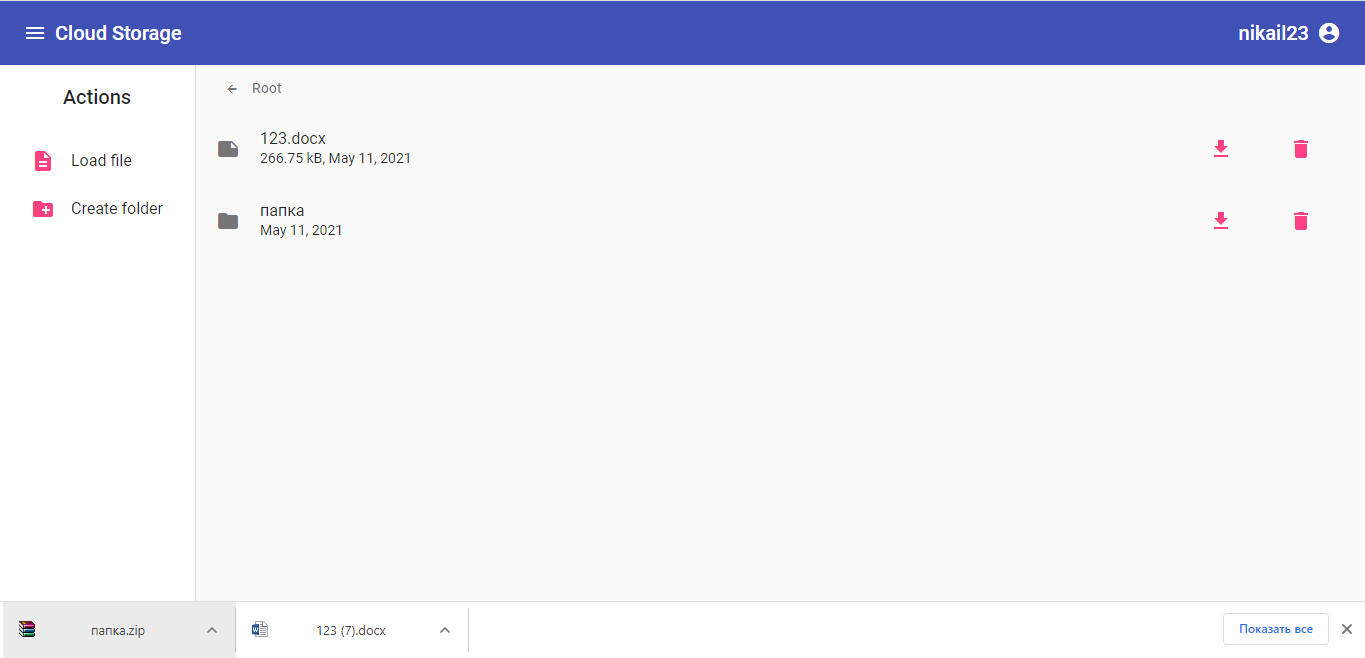


Рисунок 2.15 – Результат скачивания папки "папка"

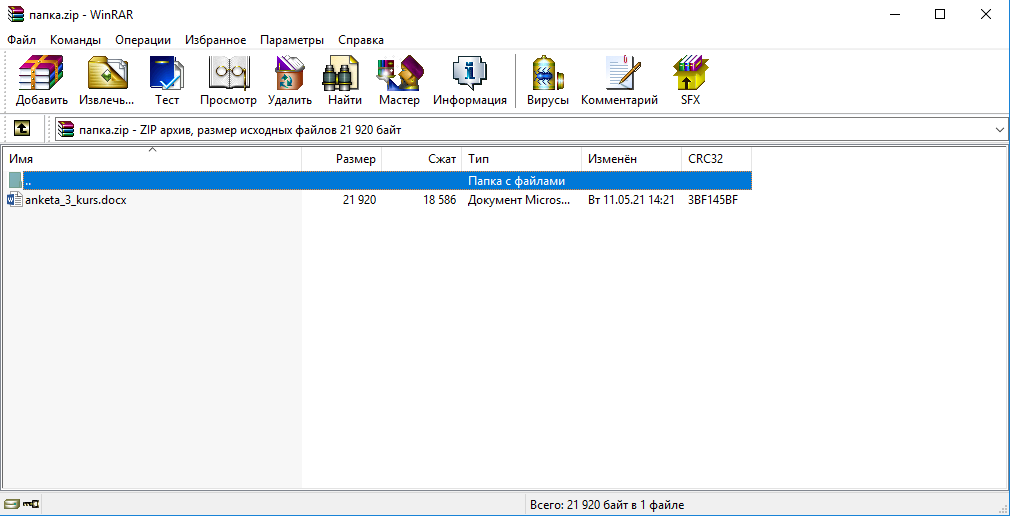


Рисунок 2.16 – Содержимое архива "папка.zip"

Далее рассмотрим функцию удаления файла. Для удаления файла необходимо кликнуть на соответствующую кнопку удаления справа. При этом, появиться диалоговое окно с информацией по удаляемому файлу. При отмене удаления, диалоговое окно закроется без удаления, при подтверждении удаления, диалоговое окно закроется, файл удалится из списка, а также из хранилища. Удалим файл "123.docx", рисунок 2.17-2.18.

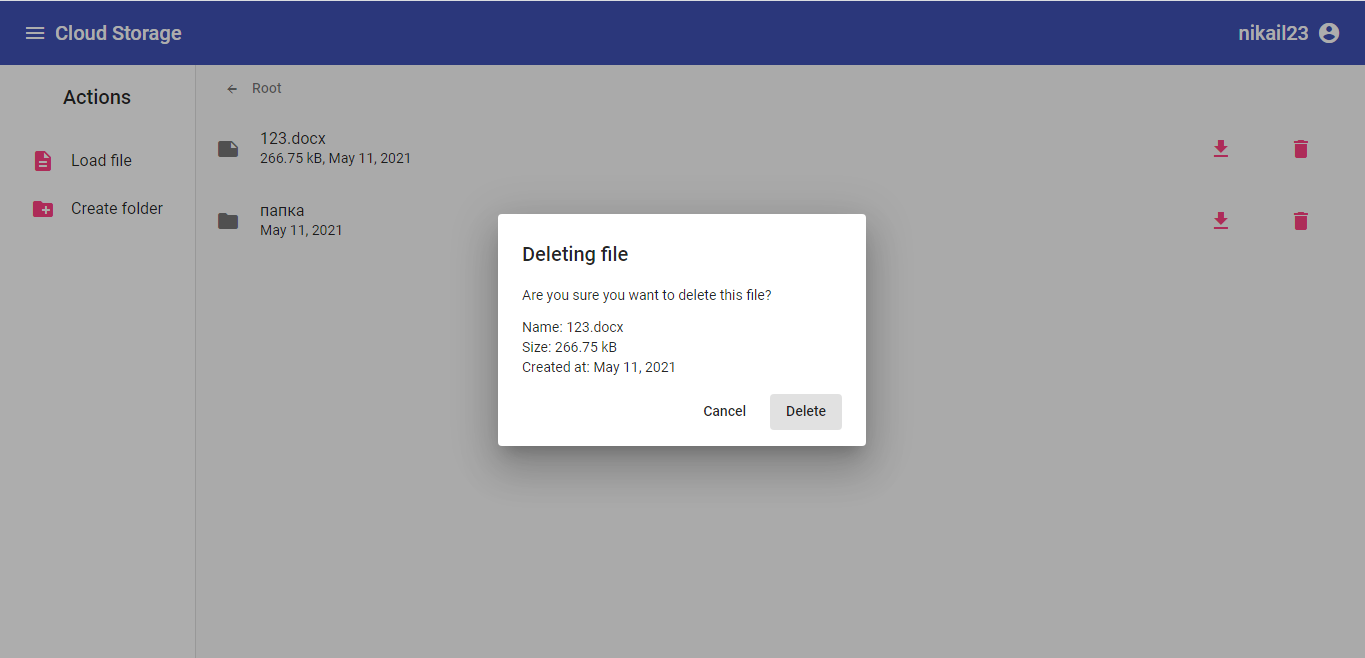


Рисунок 2.17 – Диалоговое окно удаления файла "123.docx"



Рисунок 2.18 – Результат удаления файла "123.docx"

## 2.3 Работа сервера

Работа сервера описана с помощью библиотеки Express, а также других вспомогательных библиотек.

Сервер работает по протоколу Http, по этому использует обработчики различных Htpp запросов. Также для упрощения работы были созданы вспомогательные классы Storage – сущность хранилище определенного пользователя, StorageElement – файл или папка в хранилище, StorageModel – модель данных, содержащая основные методы работы с хранилищами и список этих хранилищ, User – сущность пользователя и UserModel – модель для авторизации пользователей.

Рассмотрим обработку некоторых запросов на сервер.

Запрос на логин:

app.get("/login", (request, response) => {

setHeaders(response);

const login = request.query.login;

const password = request.query.password;

const result = users.login(login, password);

if (result) {

response.status(200).send(JSON.stringify(

{

login

}

));

} else {

response.statusCode = 404;

response.statusMessage = "User with this values dont founded!";

response.end();

}

});

Мы видим, что здесь обрабатывается GET запрос. Метод setHeaders() устанавливает заголовки ответа для CORS, без них сервис CORS не позволил бы делать запросы к серверу. Также мы видим, что запрос является запросом с параметрам, передаются два параметра – логин и пароль. Далее идет обращение к методу модели пользователей, в зависимости от его результата, клиенту вернется ответ, содержащий логин, или ответ, содержащий ошибку.

Запрос на получение содержимого хранилища пользователя:

app.get("/storage", (request, response) => {

setHeaders(response);

const userName = request.query.userName;

const result = storages.getAll(userName).map((element) => {

const children = [];

if (element.children) {

element.children.forEach(child => {

children.push(child.id);

});

}

return {

id: element.id,

name: element.name,

size: element.size,

createdAt: element.createdAt,

type: element.type,

children: children

}

});

response.status(200).send(result);

});

На сервер поступает GET запрос с параметром имени пользователя, хранилище которого необходимо получить клиенту. Также идет обращение к модели хранилища, которое возвращает хранилище пользователя по имени. После этого содержимое приводиться к нужному для клиента виду, заменяя вложенные объекты дочерних элементов на соответствующий массив их идентфикаторов.

Алгоритм метода StorageModel.getAll(userName) приведен на рисунке 2.19.

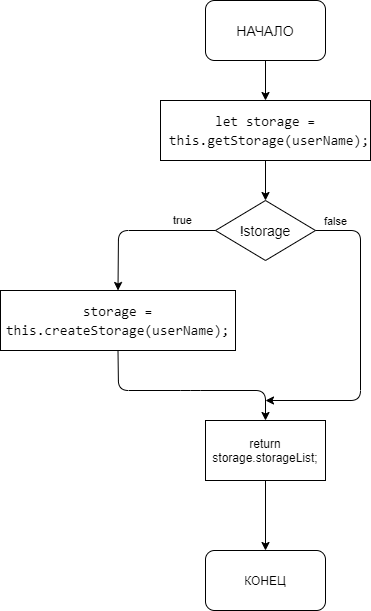


Рисунок 2.19 – Алгоритм метода StorageModel.getAll(userName)

## 2.4 Работа клиента

Клиентская часть написано на Angular. Условно можно разбить весь Angular проект на сервисы по работе с сервером, вспомогательные сервисы и классы, компоненты и модули.

Сервисы по работе с сервером используют библиотеку RxJs, для выполнения асинхронных запросов на сервер. Используя класс HttpClient выполняются все Http запросы на сервер. HttpClient возвращает объект класса Observable. Чтобы получить результат, необходимо подписаться на этот объект, вызвав метод subscribe().

В качестве примера разберем метод sendFile() сервиса storage.service.

public sendFile(file: File, path: number[], userName: string): Observable<HttpEvent<Object>> {

const uploadData = new FormData();

uploadData.append('UploadFile', file, file.name);

uploadData.append('Path', JSON.stringify(path));

uploadData.append('userName', JSON.stringify(userName));

return this.http.post('http://127.0.0.1:3000/upload', uploadData, {

reportProgress: true,

observe: 'events',

});

}

Здесь совершается POST запрос на сервер, тело запроса представляет собой объект класса FormData, который используется, как правила, для передачи файлов на сервер. В этот объект мы передаем загружаемый файл, путь из идентификаторов папок, пройденных пользователем, и имя пользователя. Возвращает метод объект класса Observable <HtppEvent <Object>>. HttpEvent позволяет отслеживать процесс отправки запросов на сервер.

# 3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 3.1 Взаимодействие с приложением

Т.к. архитектура приложения является клиент-серверной, то сначала следует запускать сервер, а лишь потом клиент. После запуска клиентской части приложения, необходимо войти в приложение. Если пользователь заходит впервые, ему необходимо сначала зарегистрироваться в приложении.

После регистрационных процедур пользователь попадает на главную страницу приложения, содержащую список элементов хранилища.

Пользователь может загружать файлы, создавать папки, скачивать файлы и папки, а также выйти из приложения.

При нажатии на кнопку добавления файла, появляется окно выбора файла. После выбора файла начнет отображаться процесс загрузки файла в хранилище, после которого файл отобразиться в списке

При нажатии на кнопку создания папки, появиться элемент создания папки, в котором нужно ввести название папки. После ввода названия и подтверждения создания, папка появиться в списке.

Чтобы перейти внутрь папки, необходимо дважды кликнуть на неё. Чтобы выйти из папки, необходимо нажать кнопку "🡨" рядом с отображением пути.

Чтобы удалить или скачать файл или папку, необходимо нажать на соответствующие кнопки рядом с информациями о файле или папке. При удалении отобразиться диалог с информацией об удаляемом элементе, а также кнопками с подтверждением или отменой. При скачивании, файлы или папки попадают в папку "Downloads". Папки предварительно архивируются.

Для того, чтобы выйти из аккаунта, необходимо нажать на иконку пользователя, справа от его имени в шапке, и нажать на кнопку "Log Out".

# ЗАКЛЮЧЕНИЕ

На сегодняшний день с развитием компьютерной техники и интернета интерес к облачным хранилищам увеличивается всё больше и больше.

В рамках данного курсового проекта было разработано программное средство «Облачное хранилище». Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* загрузка файлов в хранилище;
* создание папок в хранилище;
* возможность скачивать файлы с хранилища;
* возможность скачивать папки с хранилища;
* логин и регистрация пользователей.

Для успешной реализации вышеперечисленных функций потребовалось изучить работу с Angular, Node.js, Express, RxJS, Angular Material, Scss.

Существуют множество вариантов для улучшения данного программного обеспечения, например, возможность предоставлять другим пользователям доступ к своему хранилищу.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Node.js. Разработка серверных веб-приложений на JavaScript / Хэррон Дэвид.

[2] Angular для профессионалов / Адам Фримен

# ПРИЛОЖЕНИЕ А.

# Исходный код программы

Код сервера:

class Storage {

constructor(owner) {

this.owner = owner;

this.storageList = [];

}

}

class StorageElement {

constructor(

id,

name,

path,

type,

createdAt,

parent,

size,

children

) {

this.id = id;

this.name = name;

this.path = path;

this.type = type;

this.createdAt = createdAt;

this.parent = parent;

this.size = size;

this.children = children;

}

}

class User {

constructor(id, login, password) {

this.id = id;

this.login = login;

this.password = password;

}

}

class UserModel {

constructor() {

this.users = [];

}

check(login) {

let result = false;

this.users.forEach(user => {

if (user.login === login) {

result = true;

}

});

return result;

}

register(login, password) {

if (!this.check(login)) {

const newUser = new User(this.users.length, login, password);

this.users.push(newUser);

return true;

}

return false;

}

login(login, password) {

let result = false;

this.users.forEach(user => {

if (user.login === login && user.password === password) {

result = true;

}

});

return result;

}

get(login) {

let result = null;

this.users.forEach(user => {

if (user.login === login) {

result = user;

}

});

return result;

}

}

class StorageModel {

constructor(storagePath) {

this.storagePath = storagePath;

this.storages = [];

}

getStorage(userName) {

let result = null;

this.storages.forEach(storage => {

if (storage.owner === userName) {

result = storage;

}

});

return result;

}

createStorage(userName) {

const storage = new Storage(userName);

this.storages.push(storage);

try {

fs.mkdirSync(this.storagePath + userName);

}

catch(error) {

console.log(error);

}

return storage;

}

add(file, path, userName) {

let storage = this.getStorage(userName);

if (!storage) {

storage = this.createStorage(userName);

}

let newPath;

let lastElement;

if (path.length > 0) {

const lastElementId = path[path.length - 1]

lastElement = this.get(lastElementId, storage.storageList);

newPath = lastElement.path + `/` + file.name;

} else {

newPath = this.storagePath + `/${storage.owner}/` + file.name;

}

const storageElement = new StorageElement(

idManager.getNextId(),

file.name,

newPath,

0,

new Date(),

null,

file.size,

[]

);

if (lastElement) {

lastElement.children.push(storageElement);

storageElement.parent = lastElement;

} else {

storage.storageList.push(storageElement);

}

file.mv(newPath);

}

createFolder(name, path, userName) {

let storage = this.getStorage(userName);

if (!storage) {

storage = this.createStorage(userName);

}

let newPath;

let lastElement;

if (path.length > 0) {

const lastId = path[path.length - 1];

lastElement = this.get(lastId, storage.storageList);

newPath = lastElement.path + '/' + name;

} else {

newPath = this.storagePath + `${storage.owner}/` + name;

}

try

{

fs.mkdirSync(newPath);

const storageElement = new StorageElement(

idManager.getNextId(),

name,

newPath,

1,

new Date(),

null,

null,

[]

);

if (lastElement) {

lastElement.children.push(storageElement);

storageElement.parent = lastElement;

} else {

storage.storageList.push(storageElement);

}

return true;

}

catch (error)

{

return false

}

}

delete(id, userName) {

let storage = this.getStorage(userName);

if (!storage) {

storage = this.createStorage(userName);

}

const deletedElement = this.get(id, storage.storageList);

switch (deletedElement.type) {

case 0: fs.unlinkSync(deletedElement.path); break;

case 1: rimraf.sync(deletedElement.path); break;

}

deletedElement.children = null;

const parent = deletedElement.parent;

if (parent) {

let index;

parent.children.forEach((element, idx) => {

if (element.id === id) {

index = idx;

}

});

parent.children.splice(index, 1);

} else {

let index;

storage.storageList.forEach((element, idx) => {

if (element.id === id) {

index = idx;

}

});

storage.storageList.splice(index, 1);

}

}

getAll(userName) {

let storage = this.getStorage(userName);

if (!storage) {

storage = this.createStorage(userName);

}

return storage.storageList;

}

deepSearch(element, id) {

let result = null;

if (element.children) {

for (let i = 0; i < element.children.length; i++) {

if (result === null) {

if (element.children[i].id === id) {

result = element.children[i];

}

if (result === null && element.children[i].type === 1) {

result = this.deepSearch(element.children[i], id);

}

}

};

}

return result;

}

get(id, storageList) {

let result = null;

console.log(storageList)

for (let i = 0; i < storageList.length; i++) {

if (result === null) {

if (storageList[i].id === id) {

result = storageList[i];

}

if (result === null && storageList[i].type === 1) {

result = this.deepSearch(storageList[i], id);

}

}

};

return result;

}

}

class IdManager {

constructor() {

this.lastId = -1;

}

getNextId() {

this.lastId++;

return this.lastId;

}

}

const storages = new StorageModel("./storage/");

const users = new UserModel();

const idManager = new IdManager();

const port = 3000;

const express = require("express");

const app = express();

const cors = require("cors");

const fileUpload = require("express-fileupload");

const fs = require("fs");

const rimraf = require("rimraf");

const bodyParser = require('body-parser');

const archiver = require("archiver");

const path = require("path");

app.use(

fileUpload({

createParentPath: true,

})

);

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

app.use(cors());

function setHeaders(response) {

response.setHeader("Access-Control-Allow-Origin", "\*");

response.setHeader("Access-Control-Allow-Headers", "Content-Type");

response.setHeader(

"Access-Control-Allow-Methods",

"GET, POST, PUT, DELETE, OPTIONS"

);

}

app.get("/register", (request, response) => {

setHeaders(response);

const login = request.query.login;

const password = request.query.password;

const result = users.register(login, password);

if (result) {

response.status(200).send(result);

} else {

response.statusCode = 409;

response.statusMessage = "User with this login already exists!";

response.send(result);

}

});

app.get("/login", (request, response) => {

setHeaders(response);

const login = request.query.login;

const password = request.query.password;

const result = users.login(login, password);

if (result) {

response.status(200).send(JSON.stringify(

{

login

}

));

} else {

response.statusCode = 404;

response.statusMessage = "User with this values dont founded!";

response.end();

}

});

app.post("/upload", (request, response) => {

const file = request.files.UploadFile;

const path = JSON.parse(request.body.Path);

const userName = JSON.parse(request.body.userName);

storages.add(file, path, userName);

setHeaders(response);

response.status(200).send();

});

app.get("/get", (request, response) => {

setHeaders(response);

const id = parseInt(request.query.id);

const userName = request.query.userName;

console.log(userName);

let storage = storages.getStorage(userName);

if (!storage) {

storage = storages.createStorage(userName);

}

console.log(storage);

const element = storages.get(id, storage.storageList);

const children = [];

if (element.children) {

element.children.forEach(child => {

children.push(child.id);

});

}

const result = {

id: element.id,

name: element.name,

size: element.size,

createdAt: element.createdAt,

type: element.type,

children: children

};

response.status(200).send(result);

});

app.get("/storage", (request, response) => {

setHeaders(response);

const userName = request.query.userName;

const result = storages.getAll(userName).map((element) => {

const children = [];

if (element.children) {

element.children.forEach(child => {

children.push(child.id);

});

}

return {

id: element.id,

name: element.name,

size: element.size,

createdAt: element.createdAt,

type: element.type,

children: children

}

});

response.status(200).send(result);

});

app.get("/children", (request, response) => {

setHeaders(response);

const id = parseInt(request.query.id);

const userName = request.query.userName;

let storage = storages.getStorage(userName);

if (!storage) {

storage = storages.createStorage(userName);

}

const lastElement = storages.get(id, storage.storageList);

let result = [];

if (lastElement.children) {

lastElement.children.forEach(element => {

result.push(element);

});

}

result = result.map((element) => {

const children = [];

if (element.children) {

element.children.forEach(child => {

children.push(child.id);

});

}

return {

id: element.id,

name: element.name,

size: element.size,

createdAt: element.createdAt,

type: element.type,

children: children

}

});

response.status(200).send(result);

});

app.get("/download", (request, response) => {

const id = parseInt(request.query.id);

const userName = request.query.userName;

let storage = storages.getStorage(userName);

if (!storage) {

storage = storages.createStorage(userName);

}

const element = storages.get(id, storage.storageList);

setHeaders(response);

if (element.type === 1) {

const folderPath = path.join(\_\_dirname, element.path);

const zipName = `/${element.name}.zip`;

response.attachment(zipName);

const archive = archiver('zip', {

zlib: { level: 9 }

});

archive.on('end', function() {

console.log('Archive wrote %d bytes', archive.pointer());

});

archive.on('error', function(err) {

throw err;

});

archive.pipe(response);

archive.directory(folderPath, false);

archive.finalize();

} else {

response.download(element.path, element.name);

}

});

app.delete("/delete", (request, response) => {

const id = parseInt(request.query.id);

const userName = request.query.userName;

storages.delete(id, userName);

setHeaders(response);

response.status(200).end();

});

app.post("/createFolder", (request, response) => {

const name = request.body.name;

const path = request.body.path;

const userName = request.body.userName;

setHeaders(response);

if (storages.createFolder(name, path, userName)) {

response.status(200).end();

} else {

response.statusCode = 409;

response.statusMessage = 'Folder already exists!';

response.end();

}

});

app.listen(port);