

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Web-приложение для синхронизации и
хранения данных в облаке**

Студент _____ Платонов Данила Валерьевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Попова Н.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

_____ А.Ю. Козлов

«25» декабря 2023 г

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент _____ Платонов Данила Валерьевич _____ гр. 20ВИ1 факультета ВТ _____
<Фамилия, имя, отчество полностью>

2. Тема работы _____ Web-приложение для синхронизации и хранения данных в облаке _____

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы _____ Попова Н.А. _____

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: обеспечение возможности хранить данные в облаке и синхронизировать их между устройствами

Основные функции системы:

1. Регистрация и авторизация пользователей в системе
2. Возможность работать с различными типами файлов
3. Возможность пользователя создавать и удалять директории
4. Смена названий директорий и файлов
5. Возможность ограничивать доступ к файлами и директориям
6. Возможность пользователя предоставить доступ к хранилищу или файлу по ссылке
7. Возможность восстанавливать файлы или удалять их из корзины

Технология разработки: предметно-ориентированное программирование.

Язык программирования: JavaScript, ExpressJS

Среда программирования: React, NodeJS

4. Перечень подлежащих разработке вопросов

1. анализ предметной области и постановка задачи;
2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление ВКР.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 111 листов, 9 таблиц, 20 рисунков, 15 использованных источников и 4 приложения.

ОБЛАЧНОЕ ХРАНИЛИЩЕ, ФАЙЛЫ, NODEJS, REACT, JAVASCRIPT, EXPRESSJS, HTML, SCSS, WEB, REST API, АРХИТЕКТУРА КЛИЕНТ-СЕРВЕР, MONGODB.

Объект разработки: WEB-приложение для синхронизации и хранения данных в облаке.

Предмет разработки: разработка web-приложения для синхронизации и хранения данных в облаке.

Цель работы: разработать программное обеспечение для реализации web-приложения синхронизации и хранения данных в облаке.

Результаты работы: в процессе разработки проводилось изучение основных средств разработки и в итоге было разработано и отлажено web-приложение для синхронизации и хранения данных в облаке.

Средства разработки: React, NodeJS, ExpressJS, JavaScript, HTML, SCSS, MongoDB.

					ПГУ 09.03.02 – 1BP201.11 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.	Платонов Д.В.				Web-приложение для синхронизации и хранения данных в облаке	Лит.	Лист	Листов
Провер.	Попова Н.А.						4	111
Реценз.						группа 20ВИ1		
Н. Контр.	Такташкин Д.В.							
Утверд.	Козлов А.Ю.							

Содержание

Введение	6
1 Постановка задачи и анализ требований	7
1.1 Постановка задачи	7
1.2 Анализ программного обеспечения аналогичного назначения	8
1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению	9
2 Проектирование приложения для синхронизации и хранения данных	12
2.1 Проектирование бизнес-процессов системы	12
2.2 Проектирование базы данных	14
2.3 Проектирование интерфейса	17
3 Программная реализация	24
3.1 Программная реализация клиентской части системы	24
3.2 Программная реализация серверной части системы	32
3.3 Программная реализация базы данных	36
4 Тестирование приложения для синхронизации и хранения данных	39
4.1 Тестирование методом белого ящика	39
4.2 Расчет метрик кода	42
4.3 Тестирование интерфейса приложения	47
5 Планирование и расчет бюджета разработанной системы	48
Заключение	50
Список использованных источников	51
Приложение А (рекомендуемое) Программный код клиентской части приложения	53
Приложение Б (рекомендуемое) Программный код серверной части приложения	76
Приложение В (рекомендуемое) Результаты тестирования приложения	91
Приложение Г (рекомендуемое) Скриншоты тестирования пользовательского интерфейса	96

Введение

В современном мире создается все больше и больше различных информационных систем, работающих с большим объемом данных. Даже у простого пользователя на компьютере хранятся десятки гигабайт данных, которые необходимы ему, либо же самой операционной системе для функционирования. Будь то мобильные устройства, персональные или переносные компьютеры, для всех них есть один универсальный способ для переноса и синхронизации данных между устройствами. Таким способом является персональное облачное хранилище данных.

Облачное хранилище данных – сервис, позволяющий пользователю хранить данные на удаленном носителе, к которому у него будет доступ со своего мобильного/компьютерного устройства. Данный сервис помогает работать с файлами, передавать их между устройствами и синхронизировать между различными носителями, а также делиться с другими пользователями сервиса. Такие системы в современном мире являются довольно актуальными, так как решают важную задачу обмена данными и задачу синхронизации данных между устройствами, поэтому именно эта тема была выбрана в качестве данной дипломной работы.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Основной задачей данной дипломной работы является разработка веб приложения для синхронизации и хранения данных в облаке. Данная система не случайно разрабатывается в среде web. Интернет – удобная база, для создания подобного рода приложений, так как использует протокол http, который очень удобен при работе с файлами, так как поддерживает различные типы данных (blob, json). С помощью данного протокола можно легко передавать данные с клиента на сервер без потерь.

В качестве программных средств реализации были выбраны: rest-api (архитектура «клиент-сервер»). Для разработки серверной части программного продукта была выбрана платформа NodeJS, а также фреймворк ExpressJS. Для разработки клиентской части была выбрана библиотека React. Для хранения данных была выбрана база данных MongoDB. Выбор данных технологий обусловлен способностью обрабатывать огромное количество одновременных запросов к серверу без потери времени отклика, а также сохранять это все в базу данных с такой же скоростью, а также в более высокой скорости разработки по сравнению с аналогичными средствами разработки.

Основной набор функций, которые требуется реализовать состоит из:

- возможности работать с различными типами файлов (текстовые файлы, таблицы, изображения, видео и аудио файлы);
- возможность пользователя создавать и удалять директории;
- смена названия директорий и файлов;
- возможность ограничивать доступ к файлами и директориям, делая их публичными или приватными;
- возможность пользователя предоставить доступ к файлам либо же целому хранилищу по ссылке;
- возможность восстановить или удалить файлы из корзины.

Также у пользователя разрабатываемой системы будет присутствовать набор вспомогательных функций, напрямую не связанных с хранением и синхронизацией данных:

- возможность редактировать данные профиля в личном кабинете;
- возможность анализа хранящихся данных в виде графиков;
- возможность просмотра файлов, загруженных в хранилище без предварительного скачивания;
- возможность скачивать файлы из хранилища.

Важной особенностью разрабатываемой системы является возможность пользователя поделиться свободным количеством единиц хранения (к примеру гигабайтами) с другими пользователями, для увеличения объема их хранилища.

1.2 Анализ программного обеспечения аналогичного назначения

У разрабатываемой системы есть несколько аналогов, обладающих своим набором особенностей. Данными аналогами являются две системы: Google диск [1] и Yandex диск [2]. Данные системы были выбраны, как обладающие наибольшей популярностью варианты. Сравнение разрабатываемой системы с аналогами представлено в таблице 1.

Таблица 1 – Сравнение разрабатываемой системы с аналогичными системами

Критерий сравнения	Google диск	Yandex диск	Разрабатываемая система
Возможность работы с различными типами файлов (создавать, удалять, менять название)	+	+	+
Возможность работать с директориями (создавать, удалять, менять название)	+	+	+
Возможность сортировать файлы	+	+	+
Возможность вести историю изменений хранилища	+	+	-
Возможность анализировать хранящиеся данные в виде графиков	-	-	+
Возможность передавать свободные единицы хранения данных между пользователями	-	-	+
Возможность редактировать содержимое файлов	+	+	-

Продолжение таблицы 1

Возможность просматривать содержимое файлов без предварительного скачивания на устройство	+	+	+
Возможность настраивать и ограничивать доступ к директориям и файлам для других пользователей	+	+	+
Возможность подключить сторонних пользователей к хранилищу	+	+	-
Управление аккаунтом в личном кабинете	+	+	+
Возможность удалять и восстанавливать файлы из корзины	+	+	+

В ходе проведенного сравнения разрабатываемой системы с ее программами-аналогами, было выяснено, что разрабатываемая система обладает почти всеми основными функциями из аналогичных систем, а также имеет в своем составе уникальные функции, которые выделяют ее от аналогичных систем. К данным функциям можно отнести: визуальный анализ хранимых данных в виде графиков, а также возможность пользователя передавать свободные единицы хранения данных другим пользователям.

1.3 Формулировка и анализ требований к разрабатываемому программному обеспечению

На основе рассмотренных выше разделов, можно подвести итог и описать разрабатываемую систему и набор ее основных функций в виде диаграммы вариантов использования [3].

Диаграмма вариантов использования необходима для описания всех возможностей доступных разным пользователям системы. В данном случае приведена диаграмма вариантов использования для двух пользователей системы.

Данная диаграмма приведена на рисунке 1.

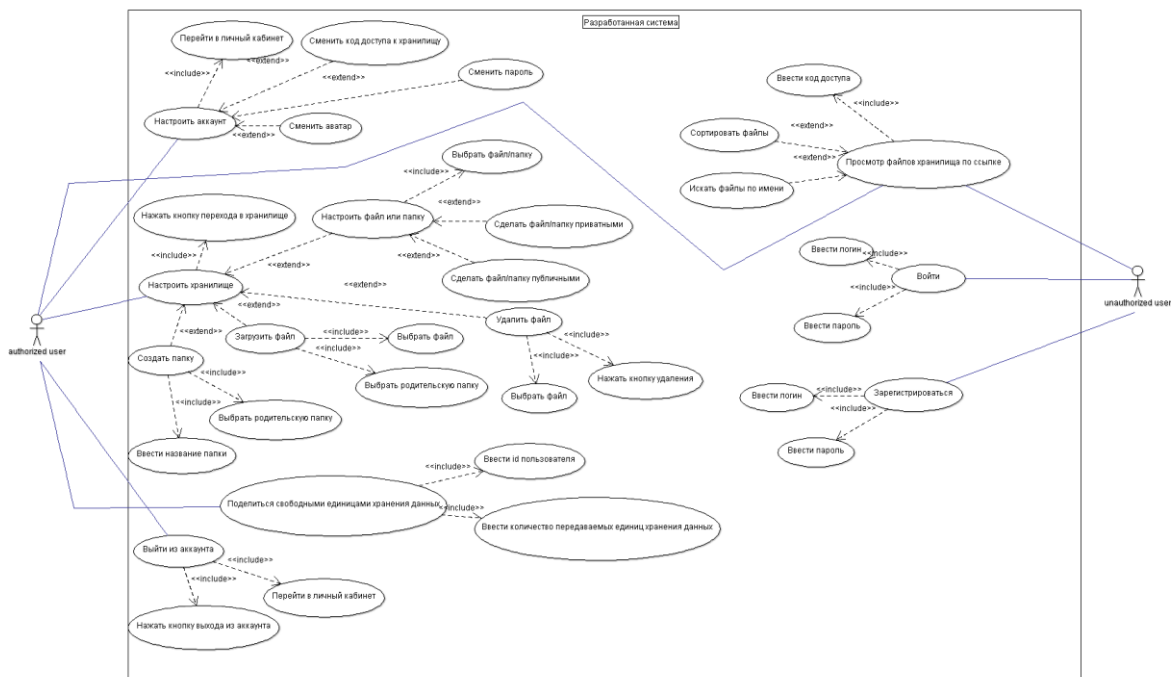


Рисунок 1 – Диаграмма вариантов использования

На данной диаграмме вариантов использования присутствует два актера: `authorized user` (авторизованный пользователь) и `unauthorized user` (неавторизованный пользователь). Каждый из них обладает своим набором доступных вариантов использования.

Так, неавторизованный пользователь имеет три основных варианта использования: «просмотреть файлы хранилища по ссылке», «войти» и «зарегистрироваться». Вариант использования «просмотреть файлы хранилища по ссылке» состоит из трех вариантов использования: ввести код доступа (`include`), сортировать файлы (`extend`) и искать файлы по имени (`extend`). Варианты использования «войти» и «зарегистрироваться» содержат в себе по два других варианта использования; ввести пароль и ввести логин.

Авторизованный же пользователь, имеет в себе так же основной вариант использования «просмотреть файлы хранилища по ссылке» как и неавторизованный пользователь, но также имеет и отличные варианты использования, а именно варианты использования: «настроить аккаунт»,

«настроить хранилище», «выйти из аккаунта» и «поделиться свободными единицами хранения данных».

Вариант использования «настроить аккаунт» состоит из: перейти в личный кабинет (include), сменить пароль (extend), сменить код доступа (extend), сменить аватар (extend).

Вариант использования «выйти из аккаунта» состоит из: перейти в личный кабинет (include), нажать кнопку выхода из аккаунта (include).

Вариант использования «поделиться свободными единицами хранения данных» состоит из: ввести id пользователя (include) и ввести количество передаваемых единиц хранения данных (include).

Вариант использования «настроить хранилище» состоит из: нажать кнопку перехода в хранилище (include), создать папку (extend), загрузить файл (extend), удалить файл (extend), настроить файл (extend).

Вариант использования «создать папку» состоит из: выбрать родительскую папку (include) и ввести название папки (include).

Вариант использования «загрузить файл» состоит из: выбрать родительскую папку (include), выбрать файл (include).

Вариант использования «удалить файл» состоит из: нажать кнопку удаления файла(include), выбрать файл (include).

Вариант использования «настроить файл» состоит из: выбрать файл (include), сменить название (extend), сделать файл публичным (extend), сделать файл приватным (extend).

2 Проектирование приложения для синхронизации и хранения данных

2.1 Проектирование бизнес-процессов системы

Для проектирования основного алгоритма, реализующего набор главных функций приложения, была выбрана диаграмма деятельности, которая описывает алгоритмы, необходимые для осуществления основной деятельности пользователя в разрабатываемой системе [4]. Диаграмма деятельности приведена на рисунке 2.

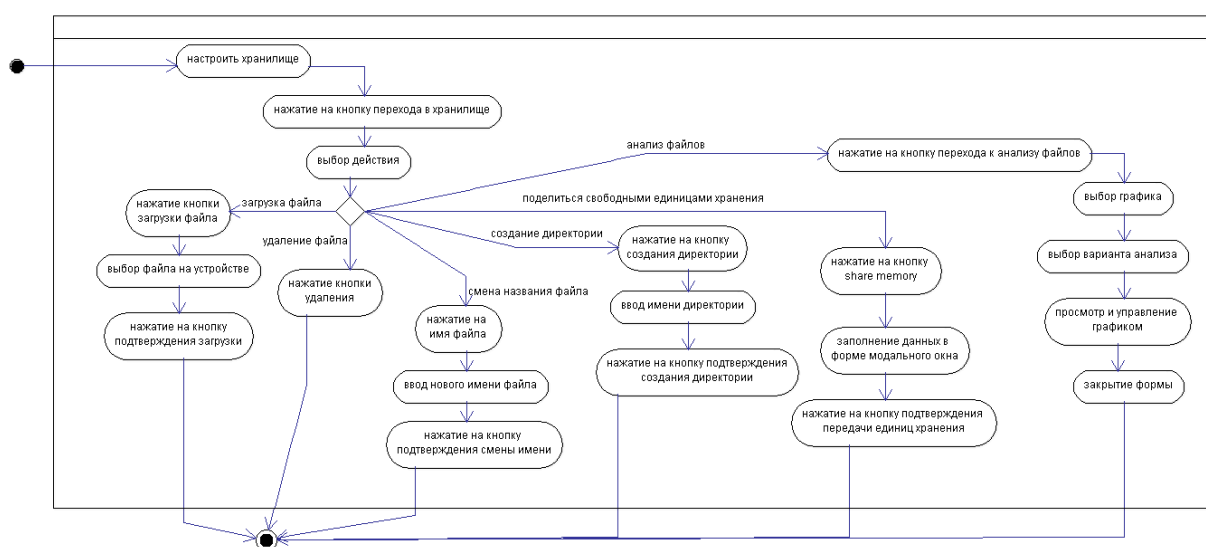


Рисунок 2 – Диаграмма деятельности

На приведенной выше диаграмме приведено описание основного алгоритма поведения пользователя. Данный алгоритм включает в себя все функции разрабатываемой системы. Описание каждой функции представлено в виде набора действий, которые необходимо совершить пользователю, чтобы получить желаемый результат. В перечень описанных на диаграмме функций входят:

- работа с файлами (создание, удаление, смена названия);
- анализ хранящихся файлов в виде графиков;
- передача единиц хранения данных другому пользователю.

Для более детального проектирования структуры приложения, была также разработана диаграмма развертывания. Данная диаграмма отображает основные компоненты системы, необходимые для реализации проектируемого функционала [5]. Данная диаграмма изображена на рисунке 3.

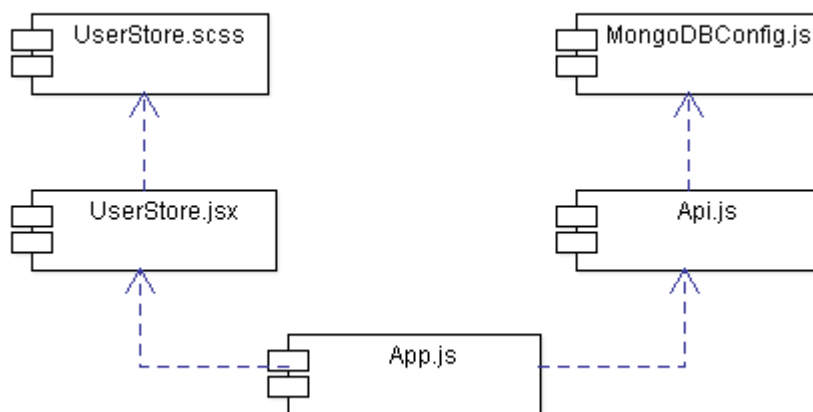


Рисунок 3 – Диаграмма развертывания

На приведенной выше диаграмме расположены основные компоненты, которые необходимы для реализации основного алгоритма приложения. В число данных компонентов входят: компонент App.js, UserStore.jsx, UserStore.scss, Api.js, MongoDBConfig.js.

Компонент App.js является корневым компонентом, в котором происходит начальная инициализация приложения. Данный компонент связан с двумя другими: UserStore.jsx и Api.js.

Компонент UserStore.jsx необходим для реализации клиентской логики для основного алгоритма приложения, в нем находится набор основных методов и разметка страницы. Данный компонент связан с компонентом UserStore.scss, в котором находится набор основных стилей для разметки внутри компонента UserStore.jsx.

Компонент Api.js необходим для хранения конфигурации axios. Это необходимо для того, чтобы можно было совершать запросы к серверу,

используя единый адрес сервера, хранящийся в данном файле и перехватывать запросы, чтобы реализовывать дополнительную функциональность (к примеру, вшить токены в заголовки запроса). Данный файл связан с компонентом MongoDBConfig.js.

Компонент MongoDBConfig.js необходим для хранения адреса удаленной базы данных. Данный компонент используется на стороне сервера для хранения информации об удаленной базе данных (ее адреса) и для возможности осуществлять запросы к самой базе данных.

2.2 Проектирование базы данных

В качестве базы данных для разрабатываемой системы была выбрана база данных MongoDB. MongoDB – noSQL база данных, основной сущностью в которой являются документы. Для хранения данных MongoDB использует такую структуру данных как JSON [6].

Для реализации основного функционала разрабатываемой предметной области в базе данных была создана коллекция документов с именем users. Данная коллекция хранит в себе информацию о всех пользователях и о файлах, внутри их хранилища. Структура данной коллекции представлена на рисунке 4.

```
_id: ObjectId('65cb2de4293be45b2596a752')
userLogin : "test1@gmail.com"
userPassword : "$2a$07$YzV4zeXXUiEAL6LFKwTJG0a2..."
balance : 5000
avatar : "http://localhost:5000/avatar.png"
▶ files : Array (1)
  secretAccessCode : "123321aaa"
▶ folders : Array (1)
  memory : 8352
▶ bucket : Array (empty)
__v : 0
```

Рисунок 4 – Структура данных коллекции users

В каждом документе данной коллекции имеется десять полей. Сам документ представлен в виде json-объекта. Подробное описание каждого поля данной коллекции приведено в таблице 2.

Таблица 2 – Описание полей документов из коллекции users

Наименование поля	Описание	Ограничения
_id	Уникальный идентификатор каждого пользователя системы	Строка. Длина 24–30 символов
userLogin	Уникальный логин каждого пользователя системы	Строка. Длина 10–255 символов
userPassword	Уникальный пароль каждого пользователя системы	Строка. Длина 10–255 символов
balance	Баланс пользователя	Число
avatar	Ссылка на файл аватара пользователя в его профиле	Строка. Длина 10–255 символов
files	Массив объектов, хранящий в себе информацию о пользовательских файлах	Массив объектов. Array<File>
secretAccessCode	Уникальный код безопасности каждого пользователя системы	Строка. Длина 10–255 символов
folders	Массив объектов, хранящий в себе информацию о пользовательских директориях внутри хранилища	Массив объектов. Array<Folder>
memory	Количество памяти, доступной пользователю	Число
bucket	Массив объектов, хранящий в себе информацию о удаленных файлах	Массив объектов. Array<File>

В приведенной выше таблице присутствует три поля, тип данных которых является массивом объектов. Так как данные типы не являются стандартными, их описание приведено в таблице 3.

Таблица 3 – Описание уникальных типов данных

Тип данных	Описание	Структура
Array<File>	Массив объектов с типом File	[{path: string, filename: string, size: string, type: string, folderId: number, status: string}]
Array<Folder>	Массив объектов с типом Folder	[{folderName: string, folderId: number, parentFolderId: number, status: string}]

Тип данных Array<File> по структуре является массивом с объектами. Каждый объект данного массива состоит из следующего набора полей: path – путь к файлу, filename – имя файла, size – размер файла, type – тип файла, folderId – идентификатор папки, в которой находится файл, status – статус файла.

Тип данных Array<Folder> по структуре является массивом с объектами. Каждый объект данного массива состоит из следующего набора полей: folderName – имя папки, parentFolderId – идентификатор родительской папки, folderId – идентификатор папки, status – статус папки.

Примеры объектов с приведенными выше типами данных отображены на рисунках 5 и 6 соответственно.

```
▼ 0: Object
  path : "http://localhost:5000/userFiles/da..."
  fileName : "back2.jpg"
  size : "0.26"
  type : "image/jpeg"
  folderId : 0
  status : "public"
```

Рисунок 5 – Объект с типом данных File

```
▼ 0: Object
  folderName : "folder"
  folderId : 1
  parentFolderId : 0
  status : "public"
```

Рисунок 6 – Объект с типом данных Folder

В ходе данного этапа была спроектирована база данных MongoDB и уникальные типы данных для разрабатываемой системы.

2.3 Проектирование интерфейса

Интерфейс разрабатываемой системы должен включать в себя следующие компоненты, необходимые для реализации основных функций:

- форма входа/регистрации;
- форма личного кабинета пользователя;
- компонент личного пользовательского хранилища;
- компонент публичного хранилища;
- вспомогательные модальные окна (в том числе окно корзины, окно анализа файлов, окно передачи свободных единиц хранения данных).

На рисунке 7 изображены основные элементы формы регистрации. Элемент под номером 1 – поле ввода логина, элемент под номером 2 – поле ввода пароля, элемент под номером 3 – кнопка перехода на форму входа, элемент под номером 4 – кнопка завершения регистрации.

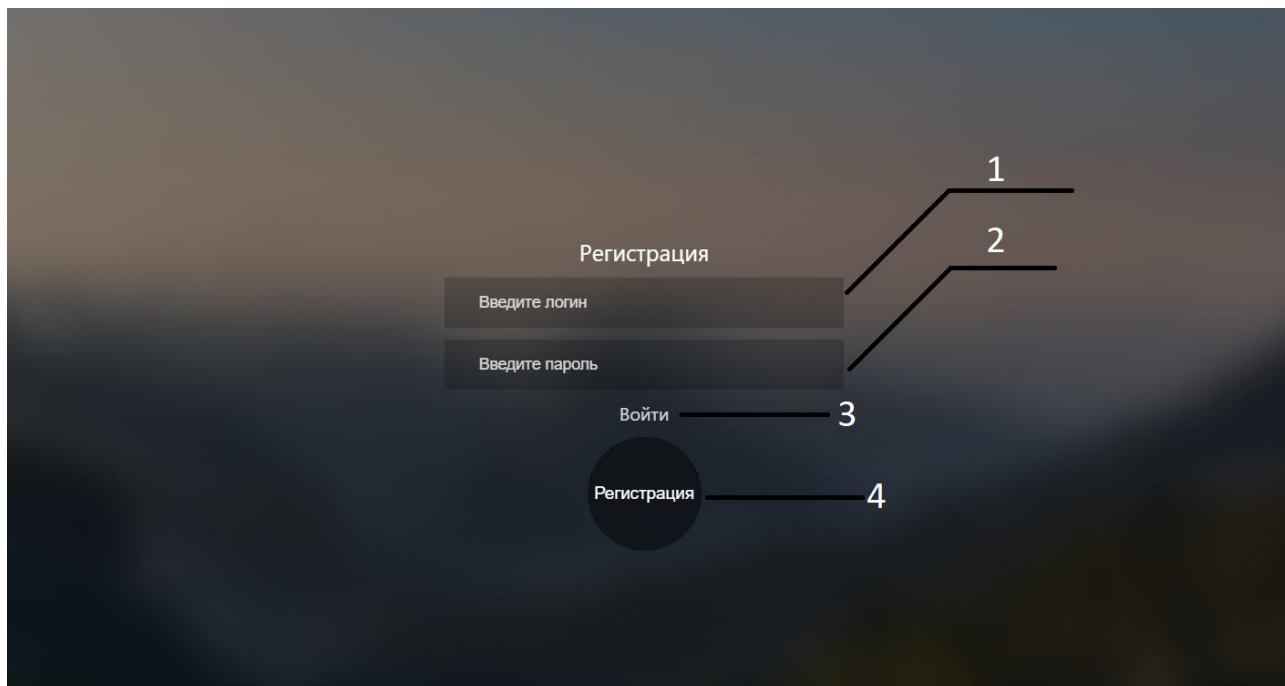


Рисунок 7 – Форма регистрации

На рисунке 8 изображены основные элементы формы входа. Элемент под номером 1 – поле ввода логина, элемент под номером 2 – поле ввода пароля, элемент под номером 3 – кнопка перехода на форму регистрации, элемент под номером 4 – кнопка завершения входа.

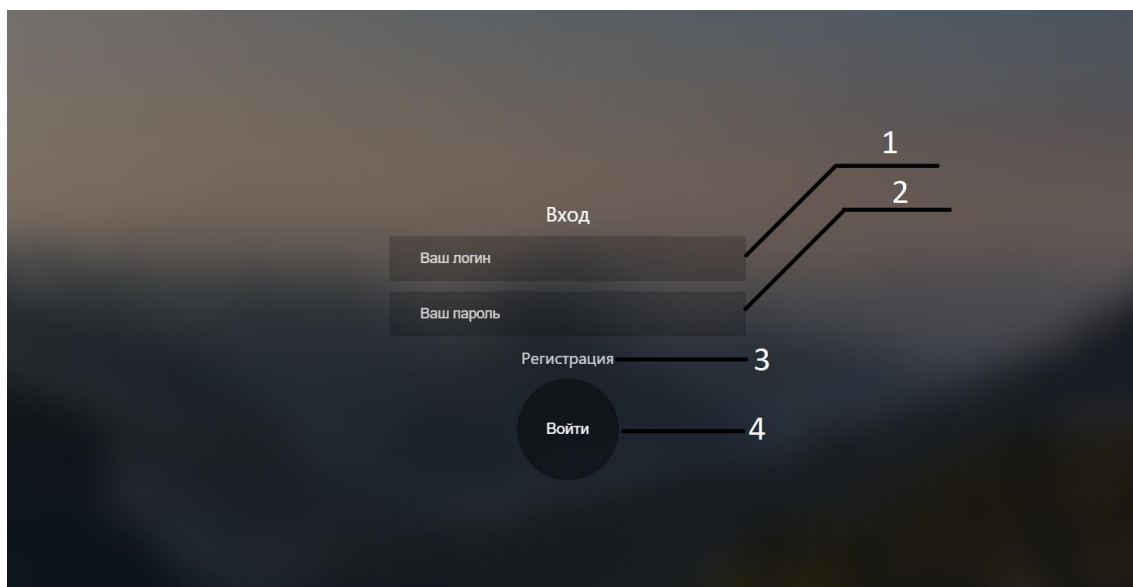


Рисунок 8 – Форма входа

На рисунке 9 изображены основные элементы формы личного кабинета. Личный кабинет необходим пользователю, чтобы иметь возможность редактировать данные своего профиля и хранилища данных. Элемент под номером 1 на рисунке 9 – аватар пользователя, элемент под номером 2 – кнопка загрузки файла аватара пользователя, элемент под номером 3 – поле отображение пользовательского логина, элемент под номером 4 – поле отображения пароля пользователя, элемент под номером 5 – кнопка отправки нового кода доступа, элемент под номером 6 – поле ввода нового кода доступа, элемент под номером 7 – кнопка открытия модального окна смены пароля, элемент под номером 8 – кнопка перехода в хранилище, элемент под номером 9 – ссылка на публичное хранилище, элемент под номером 10 – кнопка выхода из аккаунта.

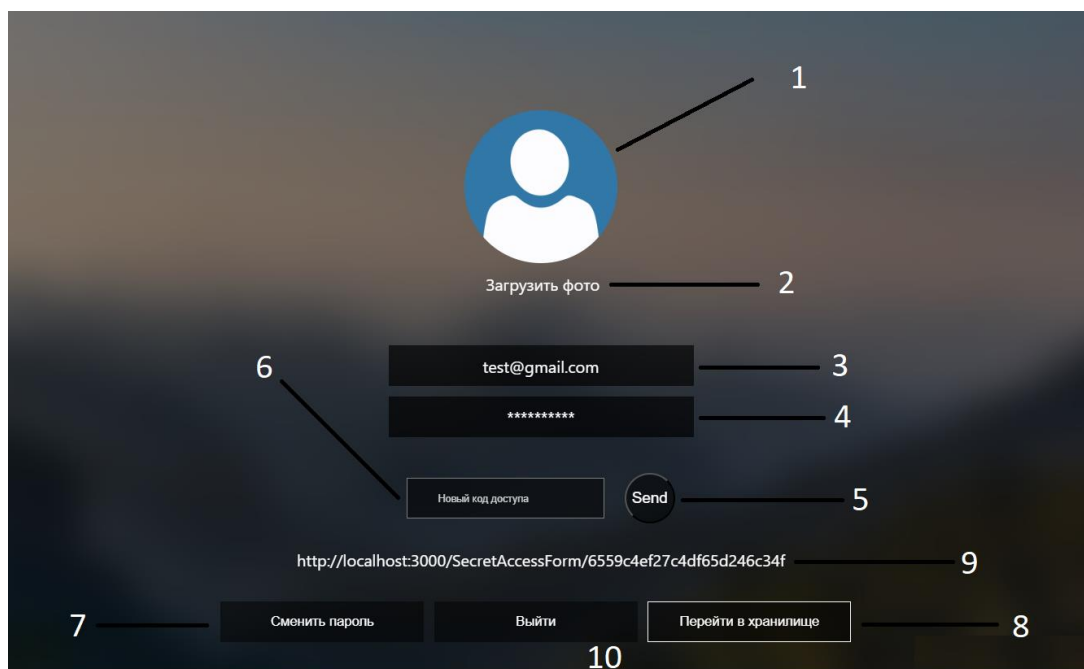


Рисунок 9 – Форма личного кабинета пользователя

На рисунке 10 изображены основные элементы хранилища пользовательских файлов. Сверху страницы элемент под номером 1 – меню выбора типа файлов для сортировки внутри текущей папки, элемент под номером 2 – поле ввода названия файла для поиска файлов по названию внутри текущей папки, элемент под номером 3 – кнопка загрузки файла в хранилище, элемент под номером 4 – кнопка перехода к предыдущей директории, элемент под номером 5 – основной блок, в котором находятся файлы хранилища.

Внутри элемента под номером 5 находятся элементы под номерами 6 – 9. Данные элементы характерны для каждого файла внутри хранилища (элементы под номерами 6, 7 появляются при наведении пользователем на файл). Элемент под номером 6 – кнопка удаления файла, элемент под номером 7 – кнопка открытия модального окна с настройками файла, элемент 8 – компонент отображающий название файла, при клике открывающий модальное окно со сменой названия файла, элемент под номером 9 – изображение файла, при клике открывается модальное окно для просмотра файла в полном размере (в случае с текстовыми файлами и архивами происходит скачивание файла).

Элемент под номером 10 – компонент отображающий progress bar с количеством занятого места в хранилище из доступного объема.

Элемент под номером 11 – кнопка перехода в личный кабинет пользователя, элемент под номером 12 – кнопка открытия модального окна с анализом файлов, элемент под номером 13 – кнопка открытия модального окна создания папки, элемент под номером 14 – кнопка открытия модального окна для передачи свободных единиц хранения информации, элемент под номером 15 – кнопка открытия корзины файлов.

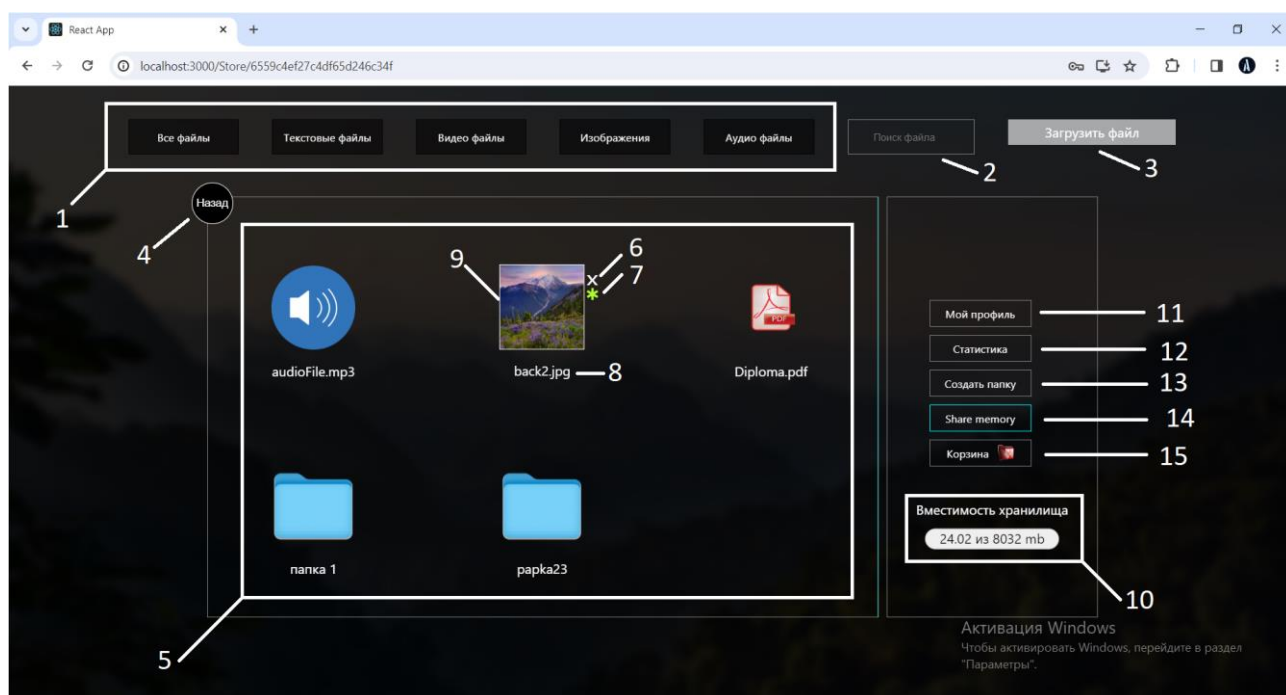


Рисунок 10 – Хранилище файлов пользователя

На рисунке 11 изображены основные элементы формы публичного хранилища. Именно это хранилище видят остальные пользователи, когда заходят по чужой ссылке на хранилище.

Элемент под номером 1 – поле ввода названия файла, элемент под номером 2 – кнопка поиска файла по названию, элемент под номером 3 – кнопка переключения в предыдущую директорию, элемент под номером 4 – изображение иконки файла, элемент под номером 5 – название файла.

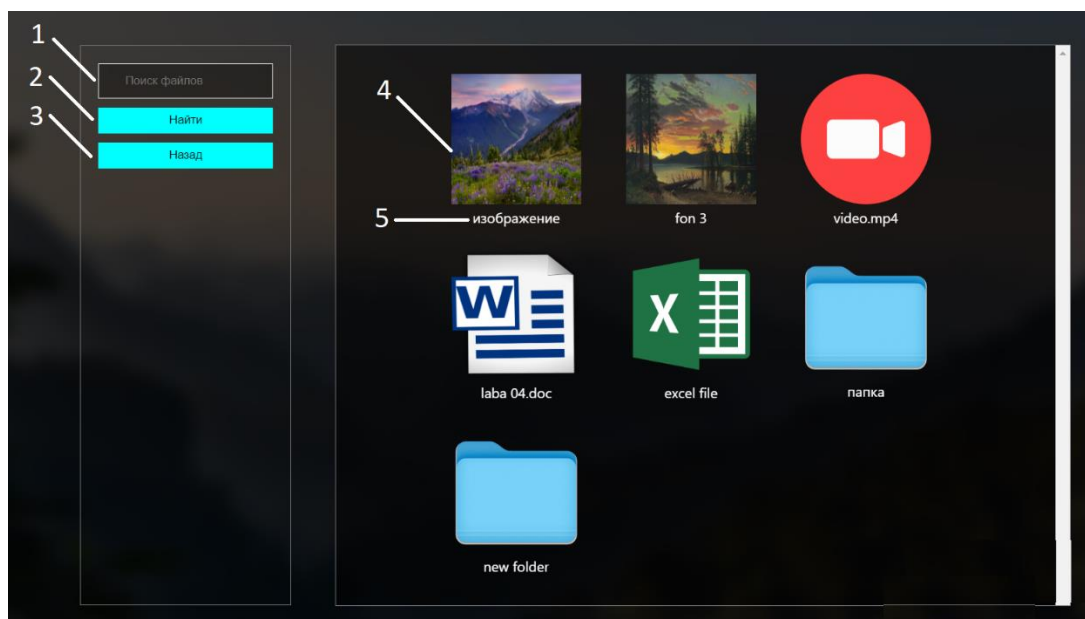


Рисунок 11 – Публичное хранилище

Элемент под номером 1 на рисунке 12 – кнопка закрытия модального окна, элемент под номером 2 – поле ввода идентификатора пользователя, элемент под номером 3 – поле ввода количества свободных единиц хранения данных, элемент под номером 4 – кнопка завершения передачи.

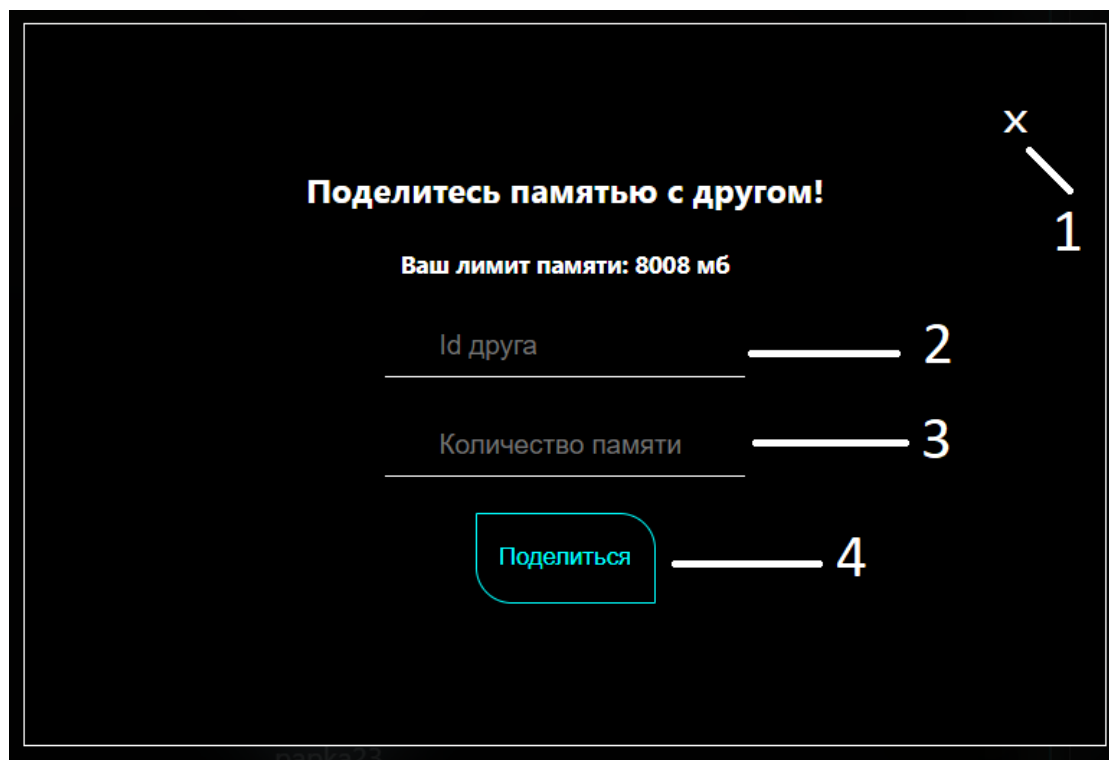


Рисунок 12 – Модальное окно передачи свободных единиц хранения данных

На рисунке 13 на модальном окне присутствует поле выбора варианта анализа данных и график, отображающий результат анализа.

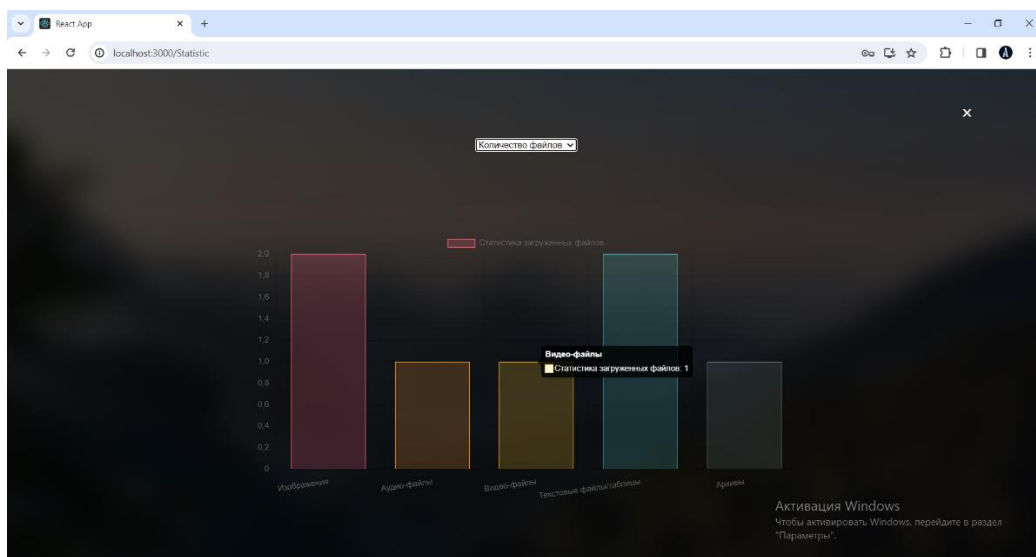


Рисунок 13 – Модальное окно с графическим анализом хранящихся файлов

На рисунке 14 на модальном окне изображены 3 элемента: кнопка закрытия окна, поле ввода названия новой папки и кнопка подтверждения создания папки.

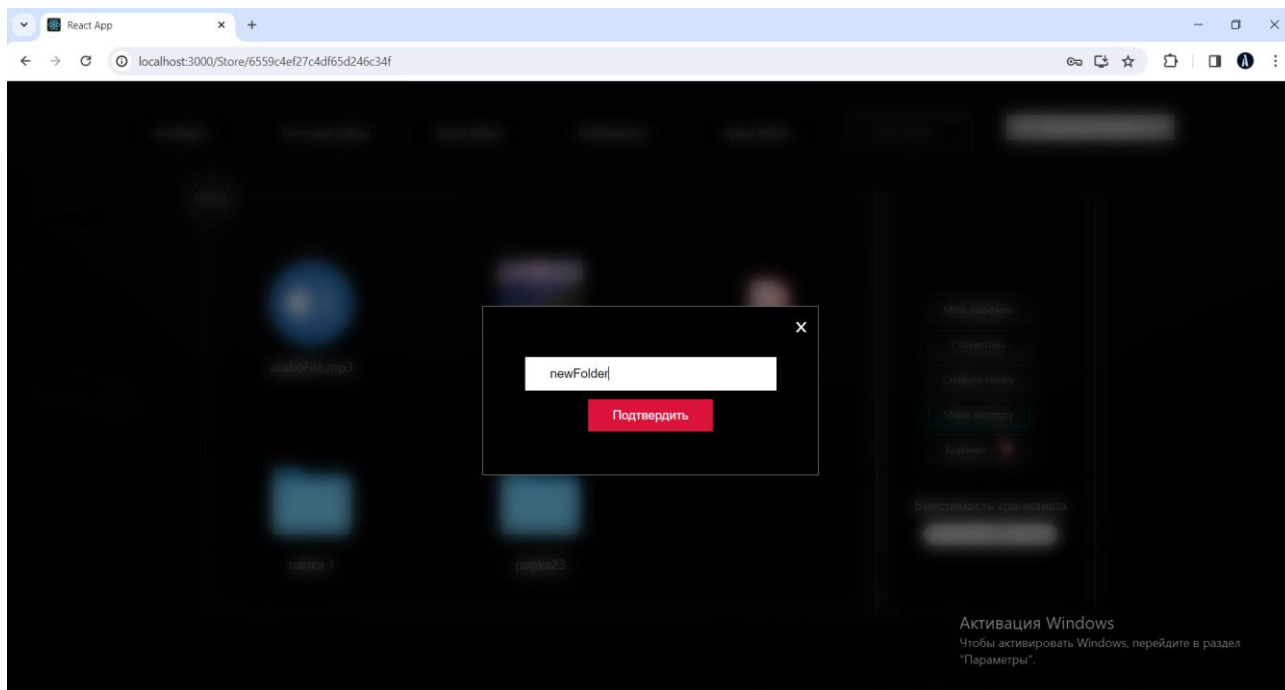


Рисунок 14 – Модальное окно смены названия папки

На рисунке 15 на модальном окне присутствует 3 элемента: кнопка закрытия окна, поле ввода нового названия файла, и кнопка подтверждения смены названия файла.

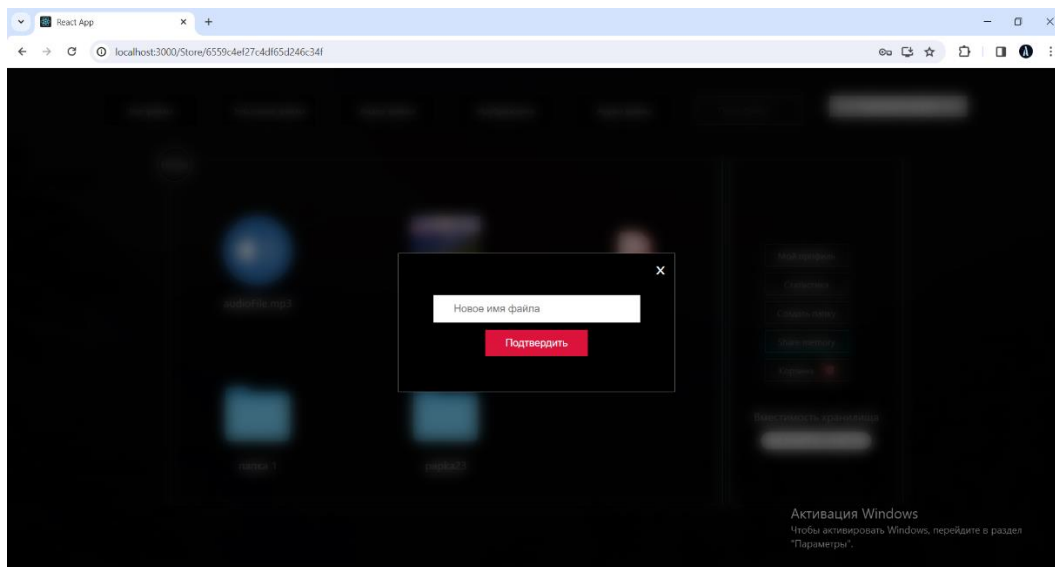


Рисунок 15 – Модальное окно смены названия файла

Результатом данного этапа стал полностью спроектированный и описанный интерфейс разработанной системы.

3 Программная реализация

3.1 Программная реализация клиентской части системы

Программная реализация клиентской части системы проводилась с помощью библиотеки React и языка программирования JavaScript, а также языка разметки HTML и препроцессора SCSS [8].

Библиотека React была выбрана потому что данная библиотека обладает высокой скоростью рендеринга содержимого компонентов на страницу за счет использования виртуальной модели дерева браузера – Virtual DOM. Компоненты, реализованные с помощью данной библиотеки пишутся на языке JSX – сочетание JS и HTML в рамках одного компонента с расширением `jsx`.

Программная реализация клиентской части системы включает в себя разработку следующих компонентов:

- рутовый компонент приложения;
- компонент авторизации;
- компонент регистрации;
- компонент личного кабинета пользователя;
- компонент хранилища данных пользователя;
- компонент публичного хранилища;
- вспомогательные компоненты (модальные окна и UI компоненты).

Ниже приведены реализации основных функций каждого компонента.

Рутовый компонент приложения – компонент, который первым запускается при загрузке приложения. В нем происходит аутентификация пользователя и редирект на последующие страницы исходя из результатов аутентификации.

Программный код данного компонента приведен в листинге 1.

Листинг 1 – Программная реализация рутового компонента системы

```
function App() {
  const history = useNavigate();
  useEffect(() => {
    if (localStorage.getItem("token")) {
      history("/Profile")
    }
    else {
      history("/Login")
    }
  })
  return (
    <div className="App">

      </div>

  );
}
export default App;
```

В текущей реализации в данном компонента присутствует хук `useEffect()`, который принимает в себя коллбэк-функцию, которая будет вызвана при каждом рендеринге рутового компонента. В данной коллбэк-функции происходит проверка авторизации пользователя и наличия у него токена. В случае, если токен имеется, то происходит редирект пользователя на страницу личного кабинета, если же токена нет, либо токен устарел, происходит редирект пользователя на страницу входа.

Редирект пользователя происходит с помощью хука `useNavigate()` и вызова функции `history()`, в которую передается маршрут, по которому следует провести редирект. В данном случае маршрута два: «/Profile» и «/Login».

Компонент входа. Данный компонент необходим, чтобы пользователь мог авторизоваться в системе. Программная реализация основных функций данного компонента представлена в листинге 2.

Листинг 2 – Программная реализация основных функций компонента входа

```
function Enter() {
  const [userLogin, setUserLogin] = useState();
  const [userPassword, setUserPassword] = useState();
  const history = useNavigate();
  const [isError, setIsError] = useState();
  const [errorText, setErrorText] = useState("");
```

```

const login = () => {
  history('/Login')
}
const showModal = () => {
  isError ? setIsError(false) : setIsError(true);
}
const enter = async () => {
  let userData = {
    userLogin: userLogin,
    userPassword: userPassword,
  }
  let rule = /^[\\w]{1}[\\w-\\.]*@[\\w-]+\\. [a-z]{2,4}$/i;
  if ((userPassword.length >= 6) && (rule.test(userLogin))) {
    await $api.post(SERVADDRESS + '/auth/login', userData)
      .then((res) => {
        localStorage.setItem("token", res.data.token);
        history("/Profile");
        console.log(res.data);
      })
      .catch((error) => {
        setErrorText("Данный пользователь не найден.
Проверьте логин и пароль");
        setIsError(true);
        console.log(error);
      })
  }
  else {
    setErrorText("Проверьте правильность email и длину
пароля (не менее 6 символов)");
    setIsError(true);
  }
}

```

В листинге 2 присутствует функция Enter(), которая выполняет валидацию данных формы регистрации и отправку данных формы на сервер, для проверки правильности введенных данных от аккаунта. В случае успешного ответа происходит редирект пользователя на страницу личного кабинета, а в случае ошибки происходит вызов функции setIsError(true). Данная функция открывает модальное окно с пояснением ошибки в правильности введенных данных.

Компонент регистрации пользователя. Данный компонент необходим для того, чтобы пользователи могли создать новый аккаунт в системе. Программная реализация основных функций данного компонента представлена в листинге 3.

Листинг 3 – Программная реализация основных функций компонента регистрации

```
function Login() {
  const [userEmail, setUserEmail] = useState();
  const [userPassword, setUserPassword] = useState();
  const [isError, setIsError] = useState(false);
  const history = useNavigate();

  const goToEnter = () => {
    history('/Enter')
  }
  const registration = () => {
    let rule = /^[\\w]{1}[\\w-\\.]*@[\\w-]+\\. [a-z]{2,4}$/i;
    const user = {
      userLogin: userEmail,
      userPassword: userPassword
    }
    if ((userPassword.length >= 6) && (rule.test(userEmail))) {
      $api.post(SERVADDRESS + '/auth/registration', user)
        .then((res) => {
          localStorage.setItem("token", res.data.token);
          history("/Profile");
          console.log(res.data);
        })
        .catch((error) => {
          console.log(error);
        })
    }
    else {
      setIsError(true);
      return;
    }
  }
}
```

В текущей реализации данного компонента присутствует основная функция registration(). Данная функция выполняет валидацию данных формы и в случае, если данные введены корректно, выполняется запрос к серверу, с попыткой зарегистрировать нового пользователя с введенными данными. Если приходит положительный ответ, происходит редирект пользователя на страницу личного кабинета, в случае ошибки будет показано модальное окно с описанием ошибки.

Реализация разметки данного компонента приведена в приложении А.

Компонент личного кабинета пользователя. Данный компонент необходим, чтобы у пользователя была возможность осуществлять управление

своим аккаунтом. Программная реализация основных функций данного компонента приведена в листинге 4.

Листинг 4 – Программная реализация основных функций компонента личного кабинета пользователя

```
function Profile() {
  const changePassword = () => {
    showModal ? setShowModal(false) : setShowModal(true);
  }
  const changeAvatar = async (e) => {
    let formData = new FormData();
    formData.append('uploadFile', e[0]);

    await $api.post(SERVADDRESS + '/upload', formData)
      .then((res) => {
        console.log(res);
        setUser({...user, avatar: res.data.avatar})
      })
      .catch((error) => {
        console.log(error);
      })
  }
  const changeSecretCode = async () => {
    if (secretCode !== "") {
      await $api.post(SERVADDRESS +
        '/profile/changeUserSecretCode', {newSecretAccessCode: secretCode})
        .then((res) => {
          inputRef.current.value = "";
          console.log(res);
        })
        .catch((error) => {
          console.log(error);
        })
    }
    else return;
  }
  useEffect(() => {
    $api.get(SERVADDRESS + '/auth/getUserData')
      .then((res) => {
        console.log(res.data.userData);
        setUser(res.data.userData);
      }).catch((error) => {
        console.log(error);
      })
  }, [])
}
```

В текущей реализации данного компонента присутствует три основных функции: `changeAvatar()`, `changeSecretCode()`, `changePassword()`.

Функция `changeAvatar` служит для возможности пользователя сменить аватар личного кабинета. Данная функция получает выбранный файл с устройства пользователя и отправляет его на сервер для сохранения.

Функция `changeSecretCode` служит для возможности пользователя сменить секретный код доступа к своему публичному хранилищу по ссылке. Данная функция получает введенный пользователем код и отправляет на сервер для сохранения.

Функция `changePassword` служит для возможности пользователя сменить пароль от аккаунта. Данная функция вызывает модальное окно для смены пароля.

Листинг модальных окон и разметки для данного компонента приведен в приложении А.

Компонент хранилища файлов пользователя. Данный компонент необходим для того, чтобы пользователь мог осуществлять основной набор функций разработанной системы.

Листинг 5 – Программная реализация функции добавления файла в хранилище

```
const uploadFile = async (file) => {
    let formData = new FormData();
    formData.append('uploadFile', file[0]);
    formData.append('folderId', chosenFolderId);

    if ((Number(sofarUserFiles) +
        Number((file[0].size/1024/1024).toFixed(2))) > 8192) {
        alert("Недостаточно места в хранилище!");
        return;
    }
    else {
        await $api.post(SERVADDRESS + '/uploadFiles', formData)
            .then((res) => {
                let files = res.data.files.filter(el => el.folderId
                    === chosenFolderId);
                setUser({...user, files: files});
                setSortedFiles(files);
                console.log(res);
            })
            .catch((err) => {
                console.log(err);
            })
    }
}
```

В текущей реализации данная функция создает объект `formData` для сохранения в нее файла в формате `blob` в поле `uploadFile` для отправки на сервер. Также в объект `formData` сохраняется идентификатор папки, в которую загружается данный файл. В случае успешного сохранения файла на сервере происходит ре-рендер страницы и обновления состояния пользователя и информации о его файлах.

Листинг 6 – Программная реализации функции удаления файла

```
const deleteFile = async (filePath) => {
  //store.dispatch({type: "DELETEDFILEPATH", payload:
  filePath})
  await $api.post(SERVADDRESS + '/files/deleteFile',
  {deletedFilePath: filePath})
  .then((res) => {
    setSortedFiles(res.data.files.filter(el => el.folderId
    === choosenFolderId));
  })
  .catch((err) => {
    console.log(err);
  })
}
```

В текущей реализации данная функция осуществляет запрос к серверу на удаление выбранного пользователем файла из хранилища, после чего в случае успешного ответа файл удаляется и осуществляется обновление данных состояния, хранящее в себе информацию о файлах пользователя.

Листинг 7 – Программная реализация функций создания и удаления директорий

```
const createFolder = async () => {
  isCreateFolder ? setIsCreateFolder(false) :
  setIsCreateFolder(true);
}
const moveFolderToBucket = async (folder) => {
  await $api.post(SERVADDRESS + '/files/moveFolderToBucket',
  {folder, parentFolderId: choosenFolderId})
  .then((res) => {
    setUser({...user, folders: res.data.folders});
    console.log(res.data);
  })
  .catch((error) => {
    console.log(error);
  })
}
```

В текущей реализации функция `createFolder()` открывает модальное окно для создания директории, а функция `moveFolderToBasket()` выполняет функцию удаления директории, в результате которого удаленная папка помещается в корзину. В случае успешного ответа от сервера обновляется состояние `user`, а именно поле `folders` объекта `user`, которое хранит в себе информацию о директориях пользователя.

Компонент публичного хранилища пользователя. Данный компонент служит для того, чтобы пользователь мог поделиться своими файлами с другими пользователями. Программная реализация основных функций данного приложения приведен в листинге 8.

Листинг 8 – Программная реализация основных функций компонента публичного хранилища

```
const searchFiles = () => {
  if (searchingName === "") {
    console.log(arrayOfFiles);
    setSortedArray(arrayOfFiles.filter((el) => el.folderId ===
currentFolderId));
  }
  else {
    const newSortedArray = sortedArray.filter(e =>
e.fileName.toLowerCase().includes(searchingName.toLowerCase()));
    setSortedArray(newSortedArray);
  }
}

const goToFolder = (folderId) => {
  setCurrentFolderId(folderId);
  setHistoryOfFolders([...historyOfFolders, folderId]);
}

const backToFolder = () => {
  if (historyOfFolders.length === 1) {
    return;
  }
  let newHistoryOfFolders = historyOfFolders;
  let lastFolder = newHistoryOfFolders[newHistoryOfFolders.length
2];
  newHistoryOfFolders.pop();
  setHistoryOfFolders(newHistoryOfFolders);
  setCurrentFolderId(lastFolder);
}
```

Текущая реализация данного компонента состоит из трех функций: `searchFile()`, `goToFolder()`, `backToFolder()`. Данные функции нужны для поиска файла по названию, перемещения вглубь папки, возвращения к предыдущей папке соответственно.

3.2 Программная реализация серверной части системы

Для программной реализации серверной части приложения была выбрана платформа NodeJS и фреймворк ExpressJS [9]. Данное решение обусловлено способностью данных технологий обрабатывать огромное количество одновременных запросов к серверу, что обеспечивает отличную производительность разработанной системы.

Программная реализация серверной части разработанной системы состоит из нескольких модулей:

- контроллер авторизации;
- контроллер по работе с файлами;
- рутовые эндпоинты серверного приложения;
- контроллер по работе с профилем пользователя.

Контроллер авторизации. Данный модель необходим для реализации функционала входа и регистрации пользователя в системе. Листинг, описывающий набор методов, имеющихся в контроллере, описан в роутере для данного контроллера. Программный код роутера приведен в листинге 9.

Листинг 9 – Программный код роутера контроллера авторизации

```
const Router = require('express');
const router = new Router();
const controller = require('./authController');

router.post('/registration', controller.registration);
router.post('/login', controller.login);
router.get('/users', controller.users);
router.get('/getUserData', controller.getUserData);

module.exports = router
```

В данном роутере присутствует набор методов, которые имеются в контроллере и выполняют основные функции по авторизации и регистрации пользователя в системе. В список данных методов входят: методом registration, доступный по адресу «/registration» и выполняющий функцию регистрации пользователя в системе, методом login, доступный по адресу «/login», выполняющий функцию входа пользователя в систему, метод users, доступный

по адресу «/users», выполняющий функцию получения данных о всех пользователях и метод `getUserData`, доступный по адресу «/getUserData», выполняющий функцию получения данных об одном пользователе по `id`.

Программный код данных методов приведен в приложении Б.

Контроллер по работе с файлами. Данный модуль необходим для реализации функционала по работе с файлами и директориями в пользовательском хранилище. Листинг, описывающий набор методов, имеющихся в контроллере, описан в роутере для данного контроллера. Программный код роутера приведен в листинге 10.

Листинг 10 – Программный код роутера контроллера по работе с файлами

```
const Router = require('express');
const router = new Router();
const controller = require('./filesController');

router.post('/uploadFile', controller.uploadFile);
router.post('/createFolder', controller.createFolder);
router.post('/changeFileName', controller.changeFileName);
router.post('/deleteFile', controller.deleteFile);
router.post('/deleteFolder', controller.deleteFolder);
router.post('/changeFolderName', controller.changeFolderName);
router.post('/changeFileStatus', controller.changeFileStatus);
router.post('/recoverUserFile', controller.recoverUserFile);
router.post('/deleteFileFromBucket',
  controller.deleteFileFromBucket);
router.post('/moveFolderToBucket', controller.moveFolderToBucket);
router.post('/recoverFolderFromBucket',
  controller.recoverFolderFromBucket);

module.exports = router
```

Данный контроллер содержит набор методов, в список которых входят: метод `uploadFile`, доступный по адресу `/uploadFile`, необходимый для загрузки файла в хранилище, метод `createFolder`, доступный по адресу `/createFolder`, необходимый для создания директории в хранилище, метод `changeFileName`, доступный по адресу `/changeFileName`, необходимый для смены названия файла в хранилище, метод `deleteFile`, доступный по адресу `/deleteFile`, необходимый для удаления файла из хранилища, метод `deleteFolder`, доступный по адресу `/deleteFolder`, необходимый для удаления директории из хранилища пользователя, метод `changeFolderName`, доступный по адресу,

/changeFolderName, необходимый для смены имени директории, метод changeFileStatus, доступный по адресу /changeFileStatus, необходимый для смены статуса файла, метод recoverUserFile, доступный по адресу /recoverUserFile, необходимый для восстановления файла из корзины, метод deleteFileFromBacKet, доступный по адресу /deleteFileFromBacKet, необходимый для удаления файла из корзины, метод moveFolderToBacKet, доступный по адресу /moveFolderToBacKet, необходимый для перемещения директории в корзину, метод deleteFolderFromBacKet, доступный по адресу /deleteFolderFromBacKet, необходимый для удаления директории из корзины.

Контроллер по работе с профилем пользователя. Данный модуль необходим для реализации функционала пользователя по управлению аккаунтом. Листинг, описывающий набор методов, имеющихся в контроллере, описан в роутере для данного контроллера. Программный код роутера приведен в листинге 11.

Листинг 11 – Программный код роутера для контроллера по работе с профилем пользователя

```
const Router = require('express');
const router = new Router();
const controller = require('./ProfileController');

router.post('/changeUserPassword', controller.changeUserPassword);
router.post('/changeUserSecretCode',
controller.changeUserSecretCode);
router.post('/getUserDataById', controller.getUserDataById);
module.exports = router
```

Данный модуль содержит набор методов, в список которых входят: метод changeUserPassword, доступный по адресу /changeUserPassword, необходимый для смены пароля пользователя, метод changeUserSecretCode, доступный по адресу /changeUserSecretCode, необходимый для смены секретного кода доступа пользователя и метод getUserDataById, доступный по адресу /getUserDataById, необходимый для получения данных пользователя по id [10].

Рутовые эндпоинты приложения. Данные эндпоинты реализуют сторонний функционал, не входящий ни в один из контроллеров, находятся в

рутовом файле приложения. Программная реализация данных методов приведена в листинге 12.

Листинг 12 – Программный код рутовых эндпоинтов серверного приложения

```
app.post("/test", cors(), async function(req, res) {
  res.status(200).json({message: "Test starting!"});
});
app.post("/shareUserMemory", async function(req, res) {
  try {
    const token = req.headers.authorization;
    const userId = jwt_decode(token);
    const user = await User.findOne({_id: userId.id});

    const userToShare = await User.findOne({_id:
req.body.userId});

    await User.updateOne({ _id: user._id }, { $inc: { memory: -
req.body.memory } });
    await User.updateOne({ _id: userToShare._id }, { $inc: {
memory: req.body.memory } });

    res.status(200).json({message: "Вы успешно передали память
другому пользователю!"});
  }
  catch (error) {
    res.status(200).json({message: "Error!"});
    console.log(error);
  }
})
async function startApp() {
  try {
    await mongoose.connect(BD_URL);
    app.listen(PORT, () => console.log('Server started at PORT'
+ " " + PORT));
  } catch (error) {
    console.log(error);
  }
}
startApp();
```

В список данных методов входят три метода: test – метод для тестового запуска сервера, startApp – метод для запуска сервера на определенном порту, с подключением к удаленной базе данных MongoDB и метод shareUserMemory, доступный по адресу /shareUserMemory, необходимый для возможности пользователя поделиться свободными единицами хранения данных с другими пользователями [11, 12].

Результатом данного этапа стал полностью разработанный сервер приложения. Листинг методов сервера приведенных в контроллерах находится в приложении Б.

3.3 Программная реализация базы данных

Для корректной работы спроектированной базы данных MongoDB необходимо программно подключить приложение к удаленной базе данных, разработать схемы данных, которые используются в методах серверной части приложения.

Схема данных – объект, описывающих типы данных, которые будут храниться в созданных коллекциях внутри базы данных MongoDB. Для корректной работы разработанной системы была создана основная схема данных User. Программный код данной схемы данных приведен в листинге 13.

Листинг 13 – Программный код схемы данных Users

```
const {model, Schema} = require('mongoose');

const User = new Schema({
  userLogin: {type: String, unique: true, required: true},
  userPassword: {type: String, required: true},
  balance: {type: Number},
  avatar: {type: String},
  files: {type: Array},
  secretAccessCode: {type: String},
  folders: {type: Array},
  memory: {type: Number},
  basket: {type: Array}
})

module.exports = model('User', User)
```

Данная схема создается с помощью импортируемого класса Schema из модуля mongoose (средство для управления базой данных MongoDB). В созданной схеме User присутствует девять полей. Описание данных полей приведено в таблице 4.

Таблица 4 – Описание полей схемы данных User

Поле	Описание	Свойства
userLogin	Логин пользователя	Уникальное, обязательное
userPassword	Пароль пользователя	Обязательное
balance	Баланс	Создается автоматически
avatar	аватар	Создается автоматически
files	Файлы пользователя	По умолчанию создается пустым
secretAccessCode	Секретный код доступа	По умолчанию создается пустым
folders	Директории пользователя	По умолчанию создается пустым
memory	Доступная память	По умолчанию создается пустым
bucket	Корзина	По умолчанию создается пустым

Автоматическое создание и заполнение полей происходит на стороне сервера, при создании объекта пользователя во время регистрации. Программный код данной процедуры приведен в листинге 14.

Листинг 14 – Программный код автоматического заполнения данных пользователя при регистрации

```

async registration(req, res) {
  try {
    const userLogin = req.body.userLogin;
    const userPassword = req.body.userPassword;
    const balance = 5000;
    const candidate = await User.find({userLogin})

    if (candidate.length !== 0) {
      res.status(400).json({message: "This email is
already used"})
    }
    else {
      const hashPassword = bcrypt.hashSync(userPassword,
7);

      const user = new User({
        userLogin,
        userPassword: hashPassword,
        balance: balance,

```

```

        avatar: "http://localhost:5000/" +
"avatar.png",
        secretAccessCode: userPassword,
        memory: 8192,
        bucket: []
    });
    const token = generateAccessToken(user._id);
    user.save();
    res.json({message: "Registration succeeded!",
token: token});
    }

    }
    catch (e) {
        console.log(e);
        res.status(400).json({message: "Registration error!"});
    }
}

```

В приведенном выше коде создается экземпляр схемы User, являющийся объектом с типом данных схемы данных User. Поле userLogin, userPassword заполняются из данных пришедших со стороны клиента, остальные поля заполняются автоматически. Так, поле balance у каждого пользователя при регистрации по умолчанию равно 5000, поле аватар ссылается на стандартный аватар для всех пользователей по адресу «http://localhost:5000/avatar.png», поле с секретным кодом по умолчанию равно паролю, память, доступная к заполнению по умолчанию равна 8192 мегабайтам, корзина, массив файлов и папок по умолчанию пусты.

4 Тестирование приложения для синхронизации и хранения данных

4.1 Тестирование методом белого ящика

В ходе данного тестирования будут проведены тесты основной функции приложения. Данной функцией является метод `uploadFile`. Метод выполняет загрузку файла в хранилище пользователя и сохранение ссылки на него в базе данных. Листинг данного метода приведен в листинге 14.

Листинг 14 – Программный код метода `uploadFile`

```
app.post («/uploadFiles», async function (req, res) {
  if (req.files && Object.keys(req.files).length !== 0) {
    const token = req.headers.authorization;
    const userId = jwt_decode(token);
    let isEmpty = await User.findOne({_id: userId.id});
    const folderId = Number(req.body.folderId);
    const uploadedFile = req.files.uploadFile;
    let uploadPath = __dirname + «/userAvatars/userFiles/» +
isEmpty.userLogin + «_» + uploadedFile.name;
    const obj = {
      path: «http://localhost:5000/userFiles/» +
isEmpty.userLogin + «_» + uploadedFile.name,
      fileName: uploadedFile.name,
      size: (uploadedFile.size/1048576).toFixed(2),
      type: uploadedFile.mimetype,
      folderId: folderId,
      status: «public»
    }
    if (fs.existsSync(uploadPath)) {
      uploadPath = __dirname + «/userAvatars/userFiles/» +
isEmpty.userLogin + «_» + «1» + uploadedFile.name;
      obj.fileName = «1»+uploadedFile.name;
      obj.path = «http://localhost:5000/userFiles/» +
isEmpty.userLogin + «_» + «1» + uploadedFile.name;
      //res.status(400).json({message: «Данный файл уже
существует в вашем хранилище»});
      await User.updateOne({ _id: isEmpty._id }, { $set: {
files: [...isEmpty.files, obj]}
});
      uploadedFile.mv(uploadPath, function (err) {
        if (err) {
          console.log(err);
          res.send(«Failed !!»);
        }
        else res.json({message: «Succesfully uploaded!»,
files: [...isEmpty.files, obj]});
      });
    }
    else res.send(«No file uploaded !!»);
  }
});
```

Блок схема описывающая алгоритм работы данного метода представлена на рисунке 16.

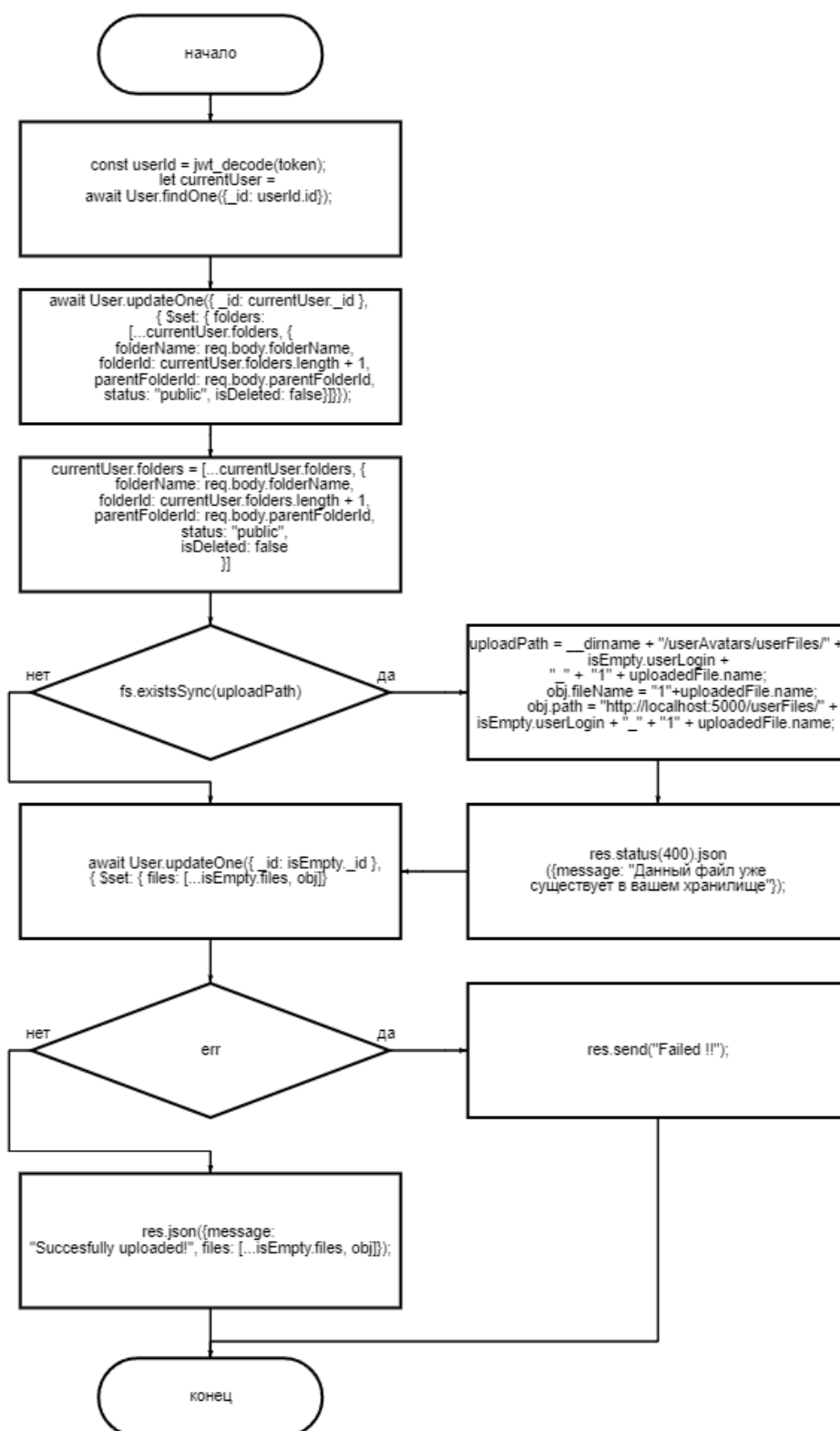


Рисунок 16 – Блок схема метода `uploadFile`

Результаты вызова данного метода при различных входных данных приведены в таблице 5.

Таблица 5 – Покрытие операторов

№	Операторы	Входные данные	Выходные данные
1	A, B, C, D, E, G, H, J, M	uploadPath = ./userAvatars/userFiles/testuser@gmail.com_lix123.png , err = true.	{«Failed!!»}
2	A, B, C, D, E, G, H, N	uploadPath = ./userAvatars/userFiles/testuser@gmail.com_lix.png, err = false.	{message:»Success fully uploaded!», files: [...]}

Исходя из проведенных выше тестов, можно сказать, что метод uploadFile работает корректно при любых входных данных.

Покрытие решений или покрытие ветвей – это метод тестирования, который направлен на то, чтобы гарантировать, что каждая из возможных ветвей из каждой точки принятия решения выполняется хотя бы один раз, и тем самым обеспечивает выполнение всего достижимого кода. То есть каждое решение принимается как истинное, так и ложное [13].

В таблице 6 представлено покрытие решений для функции uploadFile.

Таблица 6 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	F – да, K – да	uploadPath = ./userAvatars/userFiles/testuser@gmail.com_lix123.png, err = true.	{«Failed!!»}
2	F – нет, K – нет	uploadPath = ./userAvatars/userFiles/testuser@gmail.com_lix.png, err = false.	{message: «Successfully uploaded!», files: [...]}

На основе данных тестов, можно сказать, что каждая ветвь функции uploadFile принимает как истинное, так и ложно значение при данном наборе тестов, значит, функция работает правильно.

Покрытие условий – метод тестирования отвечающий на вопрос каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована; покрытие путей – все ли возможные пути через заданную часть кода были выполнены и протестированы.

Покрытие условий для функции `uploadFile` представлено в таблице 7.

Таблица 7 – Покрытие условий

№	Условие	Входные данные	Выходные данные
1	F1 – да, K1 – да	<code>uploadPath</code> = <code>./userAvatars/userFiles/testuser@gmail.com_lix123.png</code> , <code>err = true</code> .	{«Failed!!»}
2	F2 – да, K2 – да	<code>uploadPath</code> = <code>./userAvatars/userFiles/testuser@gmail.com_lix.png</code> , <code>err = false</code> .	{message: «Successfully uploaded!», files: [...]}

На основе приведенных выше тестов, можно сказать, что каждое условие в функции `uploadFile` принимает как истинное, так и ложное значение, соответственно, функция работает правильно.

4.2 Расчет метрик кода

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода [14]. Проведем подсчет данной метрики в функциях `deleteFile`, `deleteFolder`, `uploadFile`, `createFolder`, служащих для создания и удаления файлов в хранилище пользователя. Листинг данных методов приведен в приложении Б.

Физическими строками считаются все непустые строки текстового файла.

Измеряя логические строки кода, предпринимается попытка посчитать количество собственно операторов в программе, но, разумеется, их определение зависит от конкретного языка программирования.

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, узлы соединены ориентированными рёбрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла.

Граф потока управления программы для рассматриваемой функции приведен на рисунке 17.

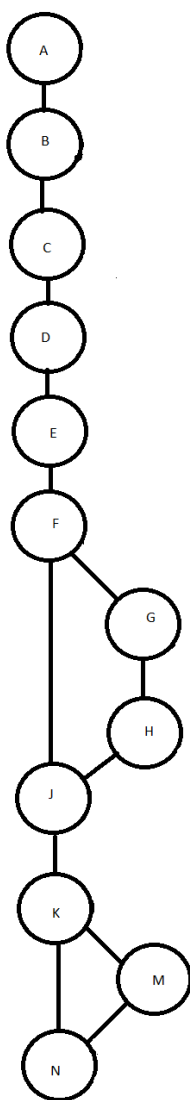


Рисунок 17 – Управляющий граф программы

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы: NUOprtr (Number of Unique Operators) – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов), NUOprnd (Number of Unique Operands) – число уникальных операндов программы (словарь операндов), Noprtr (Number of Operators) – общее число операторов в программе, Noprnd (Number of Operands) – общее число операндов в программе [15, 16].

Операторы и операнды приведены в таблицах 8 и 9 соответственно.

Таблица 8 – Операторы

Операторы	Число вхождений
=	33
!	3
!==	3
.	159
If	7
&&	1
Try	3
Catch	3
==	2
Await	8
Async	4
Jwt_decode	4
findOne()	4
updateOne()	4
...	6
+	22
===	2
>	11
{}	46

На основе данных из приведенной выше таблицы можно сказать, что $n1 = 19$, $N1 = 325$.

Таблица 9 – Операнды

Операнды	Число вхождений
currentUser	8
userId	10
Token	8
User	16
Req	13
Res	13
Err	9
folderId	22
uploadedFile	10
obj	6
Fs	4
newFoldersArray	3
newFilesArray	6
newUserBucket	3
path	19
deletedUserFile	2

На основе данных из приведенной выше таблицы можно сказать, что $n_2 = 16$, $N_2 = 152$.

Основываясь на данных, полученных в таблицах, можно посчитать длину программы, словарь программы, объем программы, потенциальный объем программы, теоретический уровень программы и сложность программы. Длина программы равна 477, словарь программы равен сумме всех операторов и операндов и равен 35, объем программы равен 2442.24, потенциальный объем программы равен 11.6. На основе данных характеристик можно посчитать сложность программы, которая в итоге равна 90.25.

Расчет метрики Чепина [17]. Суть метода состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода.

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы: Р – вводимые переменные для расчетов и для обеспечения вывода, М – модифицируемые, или создаваемые внутри программы переменные, С – переменные, участвующие в управлении работой программного модуля (управляющие переменные), Т – не используемые в программе («паразитные») переменные.

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, равный 3, имеет функциональная группа С, так как данная группа влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a_1 = 1$, $a_2 = 2$, $a_4 = 0.5$. Весовой коэффициент группы Т не равен 0, поскольку «паразитные» переменные не увеличивают сложность потока данных программы, но иногда затрудняют ее понимание. С учетом весовых коэффициентов: $Q = 62$.

Значение метрики SLOC равняется 138 физическим строкам кода и 47 логическим. Это нормальное значения для одной функции.

Значение цикломатическая сложности равняется 0. Это значение не превышает 10, следовательно, функция не является структурно ложной.

Сложность программы равна 90.25. Данный показатель для данного объема программы является нормальным значением.

Значение уровня программы согласно метрике Холстеда равняется 0.004. Уровень программы имеет значение ниже среднего по данным измерений из книги Холстеда.

Значение метрики Чепина равняется 62. Если в результате изменения рассматриваемых функций значение метрики значительно увеличиться, это будет означать низкое качество изменений.

На основе данных метрик кода, можно сделать вывод, что рассматриваемая функция написана качественно.

4.3 Тестирование интерфейса приложения

Тестирование графического интерфейса пользователя или UI-тестирование – это процесс тестирования элементов управления в приложении, который помогает убедиться, что интерфейс соответствует ожидаемой производительности и функциональности.

В процессе тестирования интерфейса данного приложения будут протестированы следующие разделы интерфейса: формы регистрации и входа в систему, личный кабинет пользователя, хранилище файлов пользователя, модальные окна, присутствующие в приложении, формы отображения файлов, аудио и видео плееры.

Тестирование интерфейса происходит по следующим критериям: корректное позиционирование элементов (1), читабельность текстовой части интерфейса (2), отзывчивость интерфейса (3).

Результаты тестирования пользовательского интерфейса приведены в приложении В.

5 Планирование и расчет бюджета разработанной системы

Для планирования и расчета бюджета была выбрана диаграмма Ганта. Разработанная диаграмма приведена на рисунке 18.

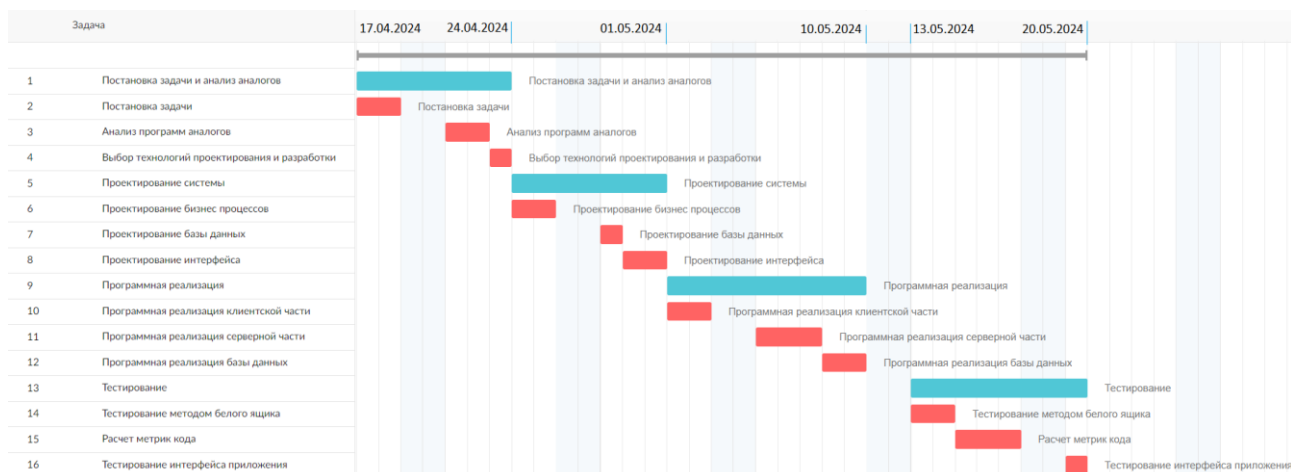


Рисунок 18 – Диаграмма Ганта

На приведенной выше диаграмме присутствует шестнадцать задач, которые были выполнены на протяжении всего периода разработки. Присутствует четыре основных задачи: «Постановка задачи и анализ аналогов», «проектирование системы», «программная реализация» и «тестирование». Каждая из этих задач в свою очередь состоит из подзадач. Так, задача «постановка задачи и анализ аналогов» состоит из трех подзадач: постановка задачи, анализ программ аналогов и выбор технологий проектирования и разработки. Задача «проектирование системы» состоит из трех подзадач: проектирование бизнес процессов системы, проектирование базы данных и проектирование интерфейса. Задача «программная реализация» состоит из трех подзадач: программная реализация клиентской части, программная реализация серверной части, и программная реализация базы данных. Задача «тестирование» состоит из трех подзадач: тестирование методом белого ящика, расчет метрик кода и тестирование интерфейса приложения.

Информация об исполнителях каждого этапа представлена на рисунке 19.

Исполнитель	Стоимость	
Архитектор	в час	300 руб.
Frontend-разработчик	в час	250 руб.
Backend-разработчик	в час	300 руб.
Тестировщик	в час	150 руб.

Рисунок 19 – Исполнители этапов проекта

Также был рассчитан бюджет разработки и проектирования системы. Таблица с результатами расчета бюджета представлена на рисунке 20.

Задача	Исполнитель	Фактическая стоимость	Статус	Затраченное время
Постановка задачи и анализ аналогов		8400 руб.		28 ч
Постановка задачи	Архитектор	3000 руб.	● Выполнено	10 ч
Анализ программ аналогов	Архитектор	3600 руб.	● Выполнено	12 ч
Выбор технологий проектирования и разработки	Архитектор	1800 руб.	● Выполнено	6 ч
Проектирование системы		10 200 руб.		34 ч
Проектирование бизнес процессов	Архитектор	4800 руб.	● Выполнено	16 ч
Проектирование базы данных	Архитектор	1200 руб.	● Выполнено	4 ч
Проектирование интерфейса	Архитектор	4200 руб.	● Выполнено	14 ч
Программная реализация		14 800 руб.		52 ч
Программная реализация клиентской части	Frontend-разработчик	4000 руб.	● Выполнено	16 ч
Программная реализация серверной части	Backend-разработчик	7200 руб.	● Выполнено	24 ч
Программная реализация базы данных	Backend-разработчик	3600 руб.	● Выполнено	12 ч
Тестирование		4950 руб.		33 ч
Тестирование методом белого ящика	Тестировщик	1200 руб.	● Выполнено	8 ч
Расчет метрик кода	Тестировщик	3000 руб.	● Выполнено	20 ч
Тестирование интерфейса приложения	Тестировщик	750 руб.	● Выполнено	5 ч

Рисунок 20 – Результаты расчета бюджета разработанной системы

По итогам расчета бюджета разработанной системы, итоговая стоимость проектирования и разработки приложения равняется 38 350 рублей.

Результатом данного этапа стал полностью спроектированный бюджет, а также сроки разработки и проектирования системы.

Заключение

В результате данной дипломной работы, было разработано веб приложение для синхронизации и хранения данных в облаке. Разработанная система была сравнена с аналогичными системами, были выявлены минусы и плюсы данных систем. Разработанная система была полностью протестирована и отлажена, начиная с клиентской и заканчивая серверной частью.

Список использованных источников

1. Google Drive // [Электронный ресурс] / Google.
URL: <https://www.google.no/intl/ru/drive/> (дата обращения: 13.04.2024)
2. Yandex диск // [Электронный ресурс] / Yandex.
URL: <https://360.yandex.ru/disk/> (дата обращения: 13.04.2024)
3. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // [Электронный ресурс] / Хабр.
URL: <https://habr.com/ru/post/566218/> (дата обращения: 13.04.2024)
4. Проектирование диаграммы деятельности // [Электронный ресурс] / National. URL: <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-deyatelnosti-uml-activity-diagram>. (дата обращения: 13.04.2024)
5. Проектирование диаграммы развертывания // [Электронный ресурс] / Guru. URL: <https://www.guru99.com/ru/deployment-diagram-uml-example.html>. (дата обращения: 13.04.2024)
6. MongoDB Documentation // [Электронный ресурс] / MongoDB.
URL: <https://www.mongodb.com/docs/>. (дата обращения: 13.04.2024)
7. ReactJS docs // [Электронный ресурс] / React.
URL: <https://legacy.reactjs.org/docs/getting-started.html>. (дата обращения: 13.04.2024)
8. ExpressJS docs // [Электронный ресурс] / ExpressJS.
URL: <https://expressjs.com/en/api.html>. (дата обращения: 13.04.2024)
9. HTML docs // [Электронный ресурс] / HTML.
URL: <https://developer.mozilla.org/ru/docs/Web/HTML>. (дата обращения: 13.04.2024)
10. SCSS docs // [Электронный ресурс] / SCSS. URL: <https://sass-lang.com/documentation/>. (дата обращения: 13.04.2024)
11. JSX docs // [Электронный ресурс] / JSX.
URL: <https://ru.react.js.org/docs/introducing-jsx.html>. (дата обращения: 13.04.2024)

12. Тестирование белым ящиком // [Электронный ресурс] / Quality.
URL: <https://tquality.ru/blog/testirovanie-metodom-belogo-yashchika/>. (дата обращения: 13.04.2024)
13. Метрики кода // [Электронный ресурс] / Microsoft.
URL: <https://learn.microsoft.com/ru>. (дата обращения: 13.04.2024)
14. Тестирование интерфейса // [Электронный ресурс] / SkyPro.
URL: <https://sky.pro/media/chto-takoe-testirovanie-polzovatelskogo-interfejsa/>. (дата обращения: 13.04.2024)
15. Метрика Чепина // [Электронный ресурс] / Habr.
URL: <https://habr.com/ru/companies/intel/articles/106082/>. (дата обращения: 13.04.2024)
16. Метрика Холстеда // [Электронный ресурс] / Turbopages.
URL: https://translated.turbopages.org/proxy_u/. (дата обращения: 13.04.2024)
17. Тестирование ПО – Методы // [Электронный ресурс] / Unetway.
URL: <https://unetway.com/tutorial/testing-software-methods>. (дата обращения: 13.04.2024)

Приложение А
(рекомендуемое)

Программный код клиентской части приложения

```

import { useNavigate } from 'react-router-dom';
import './userStore.scss';
import $api from '../..api';
import { useEffect } from 'react';
import { useState } from 'react';
import ChangeFileNameModal from './changeFileNameModal/changeFileNameModal';
import ProgressBar from './progressBar/progressBar';
import videoicon from '../..Images/video.png';
import folderImage from '../..Images/folderImage.png';
import pdficon from '../..Images/pdf.png';
import excelicon from '../..Images/excel.png';
import wordicon from '../..Images/word.png';
import bucketFile from '../..Images/bucket_file.png';
import audioIcon from '../..Images/audioIcon.png';
import ImageSlider from './imageSlider/imageSlider';
import VideoPlayer from './videoPlayer/videoPlayer';
import { useSelector } from 'react-redux';
import ChangeFolderNameModal from './changeFolderNameModal/changeFolderNameModal';
import CreatedFolderModal from './createFolderModal/createFolderModal';
import SERVADRESS from "../..servAdress";
import VideoCreator from './videoCreator/videoCreator';
import zipImage from '../..Images/allFilesIcon.png';
import FileSettingsModal from './contextMenu/fileSettingsModal';
import SharingMemoryModal from './sharingMemoryModal/sharingMemoryModal';
import FolderSettingsModal from './contextMenu/folderSettingsModal';
import UserBucket from './userBucketOfFiles/userBucket';
import AudioPlayer from './audioPlayer/audioPlayer';

function UserStore() {
  const history = useNavigate();
  const [user, setUser] = useState();
  const [isChangeFileName, setIsChangeFileName] = useState(false);
  const [previosname, setPreviosname] = useState();
  const [sizeOfUserFiles, setSizeOfUserFiles] = useState();
  const [sortedFiles, setSortedFiles] = useState();
  const [parametrOfSort, setParametrOfSort] = useState();
  const [sliderShown, setSliderShown] = useState();
  const [videoShown, setVideoShown] = useState();
  const [chosenFilePath, setChosenFilePath] = useState();
  const [findedFileName, setFindedFileName] = useState();

  const userObjectFiles = useSelector(store => store.userFiles);

  const [chosenFolderId, setChosenFolderId] = useState(0);
  const [isChangeFolderName, setIsChangeFolderName] =
  useState(false);
  const [historyOfFolders, setHistoryOfFolders] = useState([0]);
  const [historyOfLastFolder, setHistoryOfLastFolder] = useState();
  const [isCreateFolder, setIsCreateFolder] = useState(false);

  const [isVideoCreatingStart, setIsVideoCreatingStart] =
  useState(false);

```

```

//////////
    const [isFileSettingsActive, setIsFileSettingsActive] =
useState(false);
    const [isFolderSettingsActive, setIsFolderSettingsActive] =
useState(false);
    const [changedFile, setChangedFile] = useState({});
    //////////
    const [isShowingMemoryModal, setIsShowingMemoryModal] =
useState(false);
    const [userMemory, setUserMemory] = useState(0);
    //////////
    const [isUserBucketActive, setIsUserBucketActive] =
useState(false);
    //////////
    const [isAudioShown, setIsAudioShown] = useState(false);
    const [audioPath, setAudioPath] = useState();

    const uploadFile = async (file) => {
        let formData = new FormData();
        formData.append('uploadFile', file[0]);
        formData.append('folderId', choosenFolderId);

        if ((Number(sizeOfUserFiles) +
Number((file[0].size/1024/1024).toFixed(2))) > 8192) {
            alert("Недостаточно места в хранилище!");
            return;
        }
        else {
            await $api.post(SERVADDRESS + '/uploadFiles', formData)
            .then((res) => {
                let files = res.data.files.filter(el => el.folderId
=== choosenFolderId);
                setUser({...user, files: files});
                setSortedFiles(files);
                console.log(res);
            })
            .catch((err) => {
                console.log(err);
            })
        }
    }
    const sortFiles = (param) => {
        setParametrOfSort(param);
        switch(param) {
            case "image": setSortedFiles(user.files.filter((e) =>
e.type.indexOf('image') > -1));
                break;
            case "video": setSortedFiles(user.files.filter((e) =>
e.type.indexOf('video') > -1));
                break;
            case "text": setSortedFiles(user.files.filter((e) =>
e.type.indexOf('application') > -1));
                break;
            case "audio": setSortedFiles(user.files.filter((e) =>
e.type.indexOf('audio') > -1));
                break;
        }
    }

```

```

        case "all": setSortedFiles(user.files);
        break;
    }
}
const changeFileName = async (name) => {
    setPreviosname(name);
    if (isChangeFileName) {
        setIsChangeFileName(false);
    }
    else {
        setIsChangeFileName(true);
    }
}
const changeFolderName = async (folderId) => {
    setHistoryOfLastFolder(choosenFolderId);
    await setChoosenFolderId(folderId);
    isChangeFolderName ? setIsChangeFolderName(false) :
setIsChangeFolderName(true);
}
const updateFolders = async (newFolders) => {
    await setChoosenFolderId(historyOfLastFolder);
    setUser({...user, folders: newFolders.filter(el =>
el.parentFolderId == historyOfLastFolder)});
}
const updateFiles = (newFiles) => {
    user.files = newFiles;
    setSortedFiles(newFiles.filter(el => el.folderId ===
choosenFolderId));
    sortFiles(parametrOfSort);
}
const deleteFile = async (filePath) => {
    //store.dispatch({type: "DELETEDFILEPATH", payload:
filePath})
    await $api.post(SERVADDRESS + '/files/deleteFile',
{deletedFilePath: filePath})
    .then((res) => {
        setSortedFiles(res.data.files.filter(el => el.folderId
=== choosenFolderId));
    })
    .catch((err) => {
        console.log(err);
    })
}
const calculateAllFileSize = (arrOfFiles) => {
    let sum = 0;
    arrOfFiles.map((e) => {
        sum += Number(e.size);
    })
    sum = sum.toFixed(2);
    setSizeOfUserFiles(sum);
}
const openVideo = (path) => {
    setChoosenFilePath(path);
    videoShown ? setVideoShown(false) : setVideoShown(true);
}
const openSlider = (path) => {

```

```

        setChoosenFilePath(path);
        sliderShown ? setSliderShown(false) : setSliderShown(true);
    }
    const findFileByName = (e) => {
        if(e.keyCode == 13){
            setSortedFiles(sortedFiles.filter(e =>
e.fileName.toLowerCase().includes(findedFileName.toLowerCase())));
        }
        else {
            setSortedFiles(user.files);
        }
    }
    const createFolder = async () => {
        isCreateFolder ? setIsCreateFolder(false) :
setIsCreateFolder(true);
    }
    const updateUserFolders = async (folders) => {
        setUser({...user, folders: folders});
    }
    const deleteFolder = async (folderId) => {
        await $api.post(SERVADDRESS + '/files/deleteFolder',
{folderId: folderId, parentFolderId: choosenFolderId})
        .then((res) => {
            setUser({...user, folders: res.data.folders});
            console.log(res.data);
        })
        .catch((error) => {
            console.log(error);
        })
    }
    const moveFolderToBucket = async (folder) => {
        await $api.post(SERVADDRESS + '/files/moveFolderToBucket',
{folder, parentFolderId: choosenFolderId})
        .then((res) => {
            setUser({...user, folders: res.data.folders});
            console.log(res.data);
        })
        .catch((error) => {
            console.log(error);
        })
    }
    const getFilesFromFolders = async (folderId) => {
        await $api.post(SERVADDRESS + '/getFilesFromFolder',
{folderId: folderId})
        .then((res) => {
            setUser({...user, folders: res.data.folders, files:
res.data.files});
            setSortedFiles(res.data.files);
            setChoosenFolderId(folderId);
            setHistoryOfFolders([...historyOfFolders, folderId]);
            console.log(res.data);
        })
        .catch((error) => {
            console.log(error);
        })
        console.log([...historyOfFolders, folderId]);
    }

```

```

    }
    const backToFolder = async () => {
      let lastFolder;
      if (historyOfFolders.length >= 3) {
        lastFolder = historyOfFolders[historyOfFolders.length-
2];
      }
      else lastFolder = historyOfFolders[0];

      await $api.post(SERVADDRESS + '/getFilesFromFolder',
{folderId: lastFolder})
      .then((res) => {
        setUser({...user, folders: res.data.folders, files:
res.data.files});
        setSortedFiles(res.data.files);
        setChosenFolderId(lastFolder);
        if (historyOfFolders.length !== 1) {
          let newHistory = historyOfFolders;
          newHistory.pop();
          setHistoryOfFolders(newHistory);
        }
        console.log(res.data);
      })
      .catch((error) => {
        console.log(error);
      })
    }
    const createVideoFile = () => {
      isVideoCreatingStart ? setIsVideoCreatingStart(false) :
setIsVideoCreatingStart(true);
    }
    ////
    const showFileSettingsModal = (file) => {
      setChangedFile(file);
      isFileSettingsActive ? setIsFileSettingsActive(false) :
setIsFileSettingsActive(true);
    }
    const showFolderSettingsModal = (file) => {
      setChangedFile(file);
      isFolderSettingsActive ? setIsFolderSettingsActive(false) :
setIsFolderSettingsActive(true);
    }
    ////
    const showMemoryModal = () => {
      isShowingMemoryModal ? setIsShowingMemoryModal(false) :
setIsShowingMemoryModal(true);
    }
    ////
    const openUserBucket = async () => {
      isUserBucketActive ? setIsUserBucketActive(false) :
setIsUserBucketActive(true);
      if (isUserBucketActive) {
        await $api.post(SERVADDRESS + '/getFilesFromFolder',
{folderId: chosenFolderId})
        .then((res) => {

```

```

        setUser({...user, folders: res.data.folders, files:
res.data.files});
        setSortedFiles(res.data.files);
        console.log(res.data);
    })
    .catch((error) => {
        console.log(error);
    })
}
}
////
const openAudio = (path) => {
    setAudioPath(path);
    isAudioShown ? setIsAudioShown(false) :
setIsAudioShown(true);
}

useEffect(() => {
    setUser({...user, files: userObjectFiles});
    setSortedFiles(userObjectFiles);
    sortFiles(parametrOfSort);
}, [userObjectFiles])
useEffect(() => {
    $api.post(SERVADDRESS + '/getFilesFromFolder', {folderId:
choosenFolderId})
    .then((res) => {
        setUser({...user, folders: res.data.folders, files:
res.data.files});
        setSortedFiles(res.data.files);
        console.log(res.data);
    })
    .catch((error) => {
        console.log(error);
    })

    $api.get(SERVADDRESS + '/auth/getUserData')
    .then((res) => {
        setUserMemory(res.data.userData.memory);
    })
    .catch((error) => {
        console.log(error);
    })
}, [])
useEffect(() => {
    $api.get(SERVADDRESS + '/auth/getUserData')
    .then((res) => {
        calculateAllFileSize(res.data.userData.files);
    })
    .catch((error) => {
        console.log(error);
    })
})
return (
    <div className="UserStore">
        { isAudioShown ? <AudioPlayer audio={audioPath} closeModal
= {openAudio} /> : null }

```

```

        { isUserBacketActive ? <UserBacket currentFolder =
{choosenFolderId} close = {openUserBacket} /> : null }
        { isFolderSettingsActive ? <FolderSettingsModal file =
{changedFile} close = {showFolderSettingsModal} /> : null }
        { isShowingMemoryModal ? <SharingMemoryModal close =
{showMemoryModal} /> : null }
        { isFileSettingsActive ? <FileSettingsModal file =
{changedFile} close = {showFileSettingsModal} /> : null }
        { isVideoCreatingStart ? <VideoCreator
closeModal={createVideoFile} /> : null }
        { sliderShown ? <ImageSlider image = {choosenFilePath}
openSlider = {openSlider} /> : null }
        { videoShown ? <VideoPlayer image = {choosenFilePath}
openVideo = {openVideo} /> : null }
        { isChangeFileName ? <ChangeFileNameModal updateFiles =
{updateFiles} previosname = {previosname} hide = {changeFileName} />
: null }
        { isChangeFolderName ? <ChangeFolderNameModal
changeFolders = {updateFolders} folderId = {choosenFolderId} hide =
{changeFolderName} /> : null }
        { isCreateFolder ? <CreatedFolderModal updateUserFolders
= {updateUserFolders} choosenFolderId = {choosenFolderId} hide =
{createFolder} /> : null }
        <div className="store">
            <div className="store__menu">
                <button onClick={backToFolder}
className="back__to__folder__button">Назад</button>
                <div onClick={() => sortFiles("all")}
className='div__btn'>Все файлы</div>
                <div onClick={() => sortFiles("text")}
className='div__btn'>Текстовые файлы</div>
                <div onClick={() => sortFiles("video")}
className='div__btn'>Видео файлы</div>
                <div onClick={() => sortFiles("image")}
className='div__btn'>Изображения</div>
                <div onClick={() => sortFiles("audio")}
className='div__btn'>Аудио файлы</div>
                <input onKeyDown={findFileByName} onChange={(e)
=> setFindedFileName(e.target.value)} type = "text"
placeholder='Поиск файла' />

                <div className='upload__main'>
                    <input accept="image/*,video/*,.doc,.pdf,.xls,
audio/*,.zip,.rar,.7zip" onChange={(e) =>
uploadFile(e.target.files)} type = "file"></input>
                    <div className='upload__file'>Загрузить
файл</div>
                </div>
            </div>
            <div className="storage__files">
                <div className="files">
                    {sortedFiles?.map((e, id) => {
                        if (e.type.indexOf('image') > -1) {
                            return (
                                <div key={id} className="file">

```



```

        </div>
    )
}
else if (e.type.indexOf('application/pdf')
> -1) {
    return (
        <div key={id} className="file">
            <div onClick={() =>
showFileSettingsModal(e)}
                className={e.status ==
"public" ? "status": "private__status">&#10033;</div>
                <div onClick={() =>
deleteFile(e.path)} className="delete__button">x</div>
                <a target="_blank"
href={e.path}>
                    <div>
                        <img width={"100px"}
height={"100px"} src = {pdficon}/>
                    </div>
                </a>
                <span onClick={() =>
changeFileName(e.fileName)}>{e.fileName}</span>
            </div>
        )
    }
}
else if
(e.type.indexOf('application/vnd.ms-excel') > -1) {
    return (
        <div key={id} className="file">
            <div onClick={() =>
showFileSettingsModal(e)}
                className={e.status ==
"public" ? "status": "private__status">&#10033;</div>
                <div onClick={() =>
deleteFile(e.path)} className="delete__button">x</div>
                <a target="_blank"
href={e.path}>
                    <div>
                        <img width={"100px"}
height={"100px"} src = {excelicon}/>
                    </div>
                </a>
                <span onClick={() =>
changeFileName(e.fileName)}>{e.fileName}</span>
            </div>
        )
    }
}
else if (e.type.indexOf('audio') > -1) {
    return (
        <div key={id} className="file">
            <div onClick={() =>
showFileSettingsModal(e)}
                className={e.status ==
"public" ? "status": "private__status">&#10033;</div>
                <div onClick={() =>
deleteFile(e.path)} className="delete__button">x</div>

```

```

openAudio(e.path) }>
height={"100px"} src = {audioIcon}/>
changeFileName(e.fileName) }>{e.fileName}</span>
)
}
else if (e.type.indexOf('zip') > -1) {
  return (
    <div key={id} className="file">
      <div onClick={() =>
        showFileSettingsModal(e) }
        className={e.status ==
        "public" ? "status": "private__status"}>&#10033;</div>
      <div onClick={() =>
        deleteFile(e.path) } className="delete__button">x</div>
      <a href={e.path}>
        <div>
          <img width={"100px"}
            height={"100px"} src = {zipImage}/>
        </div>
      </a>
      <span onClick={() =>
        changeFileName(e.fileName) }>{e.fileName}</span>
    </div>
  )
}
}}}
{user?.folders?.map((e, id) => {
  if (e.isDeleted !== true) {
    return (
      <div key={id} className="file">
        <div onClick={() =>
          showFolderSettingsModal(e) }
          className={e.status ==
          "public" ? "status": "private__status"}>&#10033;</div>
        <div onClick={() =>
          moveFolderToBucket(e) } className="delete__button">x</div>
        <a>
          <div onClick={() =>
            getFilesFromFolders(e.folderId) }>
            <img width={"100px"}
              height={"100px"} src = {folderImage}/>
          </div>
        </a>
        <span onClick={() =>
          changeFolderName(e.folderId) }>{e.folderName}</span>
      </div>
    )
  }
  else {

```

```

        return null;
    }
    }
  }
</div>
<div className="storage__settings">
  <div className="navigation__menu">
    <div onClick={() => history("/Profile")}
className='item'>Мой профиль</div>
    <div onClick={() => history("/Statistic")}
className='item'>Статистика</div>
    <div onClick={createFolder}
className="item">Создать папку</div>
    <div onClick={showMemoryModal}
className="item__share">Share memory</div>
    <div onClick={openUserBacket}
className="item">Корзина <img src = {bucketFile} width={"20px"}
height={"20px"}></img></div>
  </div>
  <div className="header">Вместимость
хранилища</div>
  <ProgressBar max = {userMemory}
percentageNew={sizeofUserFiles} />
  </div>
</div>
</div>
</div>
);
}

export default UserStore;

import './userBacket.scss';
import $api from '../.../api';
import { useEffect } from 'react';
import SERVADDRESS from '../.../servAdress';
import { useState } from 'react';
import pdficon from '../.../Images/pdf.png';
import excelicon from '../.../Images/excel.png';
import wordicon from '../.../Images/word.png';
import bucketFile from '../.../Images/bucket_file.png';
import audioIcon from '../.../Images/audioIcon.png';
import videoicon from '../.../Images/video.png';
import zipImage from '../.../Images/allFilesIcon.png';
import folderImage from '../.../Images/folderImage.png';

function UserBacket({close, currentFolderId}) {
  const [userData, setUserData] = useState({});

  const recoverFile = async (file) => {
    let folder = file;
    if (file.hasOwnProperty('isDeleted')) {
      await $api.post(SERVADDRESS +
"/files/recoverFolderFromBacket", {folder})
      .then((res) => {
        setUserData({...userData, bucket: res.data.bucket});
        console.log(res);
      });
    }
  };
}

```

```

    })
    .catch((err) => {
      console.log(err);
    })
  }
  else {
    await $api.post(SERVADDRESS + "/files/recoverUserFile",
{file}))
    .then((res) => {
      setUserData({...userData, bucket: res.data.bucket});
      console.log(res);
    })
    .catch((err) => {
      console.log(err);
    })
  }
}
const deleteFile = async (file) => {
  if (file.hasOwnProperty('isDeleted')) {
    await $api.post(SERVADDRESS + '/files/deleteFolder',
{folderId: file.folderId, parentFolderId: currentFolderId})
    .then((res) => {
      setUserData({...userData, bucket: res.data.bucket});
      console.log(res.data);
    })
    .catch((error) => {
      console.log(error);
    })
  }
  else {
    await $api.post(SERVADDRESS +
"/files/deleteFileFromBucket", {file}))
    .then((res) => {
      setUserData({...userData, bucket: res.data.bucket});
      console.log(res);
    })
    .catch((err) => {
      console.log(err);
    })
  }
}
useEffect(() => {
  $api.get(SERVADDRESS + '/auth/getUserData')
  .then((res) => {
    setUserData(res.data.userData);
    console.log(res.data);
  })
  .catch((error) => {
    console.log(error);
  })
}, [])
return (
  <div className="bucket">
    <div className="bucket__main">
      <div onClick={close} className='close__button'>x</div>
      {

```

```

        userData?.bucket?.length > 0 ?
        null
        : <h3>Ваша корзина пуста</h3>
    }
    {
        userData?.bucket?.map((el, index) => {
            return (
                <div key={index} className='bucket__file'>
                    { (el.folderName !== undefined) ?
                        (
                            <img src = {folderImage}
width={"25px"} height={"25px"} />
                        )
                        :
                        (el.type.indexOf('image') > -1)
?
                        (
                            <img src = {el.path}
width={"25px"} height={"25px"} />
                        )
                        : (el.type.indexOf('video') > -
1) ?
                        (
                            <img src = {videoicon}
width={"25px"} height={"25px"} />
                        )
                        :
                        (el.type.indexOf('application/vnd.openxmlformats-
officedocument.wordprocessingml.document') > -1)
||
                        (el.type.indexOf('application/msword') > -1) ?
                        (
                            <img src = {wordicon}
width={"25px"} height={"25px"} />
                        )
                        :
                        (el.type.indexOf('application/pdf') > -1) ?
                        (
                            <img src = {pdficon}
width={"25px"} height={"25px"} />
                        )
                        :
                        (el.type.indexOf('application/vnd.ms-excel') > -1) ?
                        (
                            <img src = {excelicon}
width={"25px"} height={"25px"} />
                        )
                        : (el.type.indexOf('audio')
> -1) ?
                        (
                            <img src = {audioIcon}
width={"25px"} height={"25px"} />
                        )
                        : (el.type.indexOf('zip') >
-1) ?
                        (

```

```

width={"25px"} height={"25px"} />
                                <img src = {zipImage}
                                )
                                : null
                                }
                                <div
className='file__name'>{el.fileName || el.folderName}</div>
                                <div className="bucket__settings">
                                <div onClick={() =>
recoverFile(el)}>Восстановить</div>
                                <div onClick={() =>
deleteFile(el)}>Удалить</div>
                                </div>
                                </div>
                                )
                                })
                                }
                                </div>
                                </div>
                                );
                                }

export default UserBucket;

import { useNavigate } from 'react-router-dom';
import './storeForShowing.scss';
import zipImage from '../Images/allFilesIcon.png';
import pdficon from '../Images/pdf.png';
import excelicon from '../Images/excel.png';
import wordicon from '../Images/word.png';
import audioIcon from '../Images/audioIcon.png';
import videoicon from '../Images/video.png';
import foldericon from '../Images/folderImage.png';
import { useState } from 'react';
import { useEffect } from 'react';

function StoreForShowing({arrayOfFiles, arrayOfFolders}) {
  const history = useNavigate();

  const [searchingName, setSearchingName] = useState("");
  const [sortedArray, setSortedArray] = useState(arrayOfFiles);
  const [currentFolderId, setCurrentFolderId] = useState(0);
  const [sortedArrayOfFolders, setSortedArrayOfFolders] =
    useState(arrayOfFolders);
  const [historyOfFolders, setHistoryOfFolders] = useState([0]);

  const searchFiles = () => {
    if (searchingName === "") {
      console.log(arrayOfFiles);
      setSortedArray(arrayOfFiles.filter((el) => el.folderId
=== currentFolderId));
    }
    else {
      const newSortedArray = sortedArray.filter(e =>
e.fileName.toLowerCase().includes(searchingName.toLowerCase()));
      setSortedArray(newSortedArray);
    }
  }
}

```

```

    }
  }
  const goToFolder = (folderId) => {
    setCurrentFolderId(folderId);
    setHistoryOfFolders([...historyOfFolders, folderId]);
  }
  const backToFolder = () => {
    if (historyOfFolders.length == 1) {
      return;
    }
    let newHistoryOfFolders = historyOfFolders;
    let lastFolder = newHistoryOfFolders[newHistoryOfFolders.length - 2];
    newHistoryOfFolders.pop();
    setHistoryOfFolders(newHistoryOfFolders);
    setCurrentFolderId(lastFolder);
  }
  useEffect(() => {
    console.log(historyOfFolders);
  }, [historyOfFolders])
  useEffect(() => {
    setCurrentFolderId(0);
  }, [])
  useEffect(() => {
    let newFiles = arrayOfFiles.filter((el) => (el.folderId === currentFolderId && el.status == "public"));
    let newFolders = arrayOfFolders.filter((el) => (el.parentFolderId === currentFolderId && el.status == "public"));
    setSortedArrayOfFolders(newFolders);
    setSortedArray(newFiles);
  }, [currentFolderId])
  return (
    <div className="StoreForShowing">
      <div className="settings">
        <input onChange={setSearchingName} type="text" placeholder="Поиск файлов"/>
        <button onClick={searchFiles}>Найти</button>
        <button onClick={backToFolder}>Назад</button>
      </div>
      <div className="store">
        {sortedArray.map((e, id) => {
          if ((e.type.indexOf('image') > -1) && e.status !== "private") {
            return (
              <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                  <img width={"150px"} height={"150px"} src = {e.path}/>
                </a>
                <span>{e.fileName}</span>
              </div>
            )
          }
          else if ((e.type.indexOf('video') > -1) && e.status !== "private") {

```

```

        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {videoicon}/>
                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    }
    else if
    ((e.type.indexOf('application/vnd.openxmlformats-
officedocument.wordprocessingml.document') > -1)
    || (e.type.indexOf('application/msword') > -1)
    && e.status !== "private") {
        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {wordicon}/>
                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    else if ((e.type.indexOf('application/pdf') > -
1) && e.status !== "private") {
        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {pdficon}/>
                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    else if ((e.type.indexOf('application/vnd.ms-
excel') > -1) && e.status !== "private") {
        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {excelicon}/>
                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    else if ((e.type.indexOf('audio') > -1) &&
e.status !== "private") {
        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {audioIcon}/>

```

```

                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    else if ((e.type.indexOf('zip') > -1) && e.status
    !== "private") {
        return (
            <div key={id} className="file">
                <a target='_blank' href = {e.path}>
                    <img width={"150px"}
height={"150px"} src = {zipImage}/>
                </a>
                <span>{e.fileName}</span>
            </div>
        )
    }
    }
    }
    {sortedArrayOfFolders?.map((e) => {
        if (e.isDeleted == false) {
            return (
                <div onClick={() => goToFolder(e.folderId)}
key={e.folderId} className="file">
                    <img width={"150px"} height={"150px"}
src = {foldericon}/>
                    <span>{e.folderName}</span>
                </div>
            )
        }
        else return null;
    })}
    </div>
</div>
);
}

```

```
export default StoreForShowing;
```

```

import { useEffect, useRef } from 'react';
import { useNavigate } from 'react-router-dom';
import './profile.scss';
import $api from '../..api';
import { useState } from 'react';
import Modal from './changePasswordModal/modal';
import SERVADDRESS from "../servAdress";
import { useSelector } from 'react-redux';

```

```

function Profile() {
    const history = useNavigate();
    const [user, setUser] = useState({});
    const [showModal, setShowModal] = useState(false);
    const [secretCode, setSecretCode] = useState("");
    const theme = useSelector(store => store.theme);
    const inputRef = useRef();

    const goBack = () => {

```

```

        history('/Store');
    }
    const logout = () => {
        localStorage.removeItem("token");
        history('/');
    }
    const changePassword = () => {
        showModal ? setShowModal(false) : setShowModal(true);
    }
    const changeAvatar = async (e) => {
        let formData = new FormData();
        formData.append('uploadFile', e[0]);

        await $api.post(SERVADDRESS + '/upload', formData)
        .then((res) => {
            console.log(res);
            setUser({...user, avatar: res.data.avatar})
        })
        .catch((error) => {
            console.log(error);
        })
    }
    const changeSecretCode = async () => {
        if (secretCode !== "") {
            await $api.post(SERVADDRESS +
'/profile/changeUserSecretCode', {newSecretAccessCode: secretCode})
            .then((res) => {
                inputRef.current.value = "";
                console.log(res);
            })
            .catch((error) => {
                console.log(error);
            })
        }
        else return;
    }
    useEffect(() => {
        $api.get(SERVADDRESS + '/auth/getUserData')
        .then((res) => {
            console.log(res.data.userData);
            setUser(res.data.userData);
        })
        .catch((error) => {
            console.log(error);
        })
    }, [])
    return (
        <div className="main">
            {showModal ? <Modal show = {changePassword} /> : null}
            <div className="Profile">
                <div className="main__content">
                    <div onClick={goBack} className="close__button">
                        X
                    </div>
                    <img width={"150px"} height={"150px"} src =
{user.avatar}></img>

```

```

        <div className='upload__file__div'>
            <span>Загрузить фото</span>
            <input accept="image/*" onChange={(e) =>
changeAvatar(e.target.files)} type = "file" />
        </div>
        <div className="user__info">
            <div
className="user__login">{user.login}</div>
            <div
className="user__password">*****</div>
            <div className="change__secret__code">
                <input ref = {inputRef} placeholder='Новый
код доступа' onChange={(e) => setSecretCode(e.target.value)} type =
"text" />
                <button
onClick={changeSecretCode}>Send</button>
            </div>
            <div className="link__to__show__store">
                <p>{"http://localhost:3000/SecretAccessForm
/"+user.id}</p>
            </div>
            <div className="settings">
                <button onClick={changePassword}>Сменить
пароль</button>
                <button onClick={logout}>Выйти</button>
                <button onClick={() => history("/Store/" +
user.id)}>Перейти в хранилище</button>
            </div>
        </div>
    </div>
</div>
);
}

export default Profile;

import { useState } from 'react';
import './enter.scss';
import $api from '../../../api';
import { useNavigate } from 'react-router-dom';
import logo from '../../../Images/logo.png';
import ErrorBox from '../errorBox/errorBox';
import SERVADRESS from "../../../servAdress";

function Enter() {
    const [userLogin, setUserLogin] = useState();
    const [userPassword, setUserPassword] = useState();
    const history = useNavigate();
    const [isError, setIsError] = useState();
    const [errorText, setErrorText] = useState("");

    const login = () => {
        history('/Login')
    }
    const showModal = () => {

```

```

        isError ? setIsError(false) : setIsError(true);
    }
    const enter = async () => {
        let userData = {
            userLogin: userLogin,
            userPassword: userPassword,
        }
        let rule = /^[\\w]{1}[\\w-\\.]*@[\\w-]+\\. [a-z]{2,4}$/i;
        if ((userPassword.length >= 6) && (rule.test(userLogin))) {
            await $api.post(SERVADDRESS + '/auth/login', userData)
                .then((res) => {
                    localStorage.setItem("token", res.data.token);
                    history("/Profile");
                    console.log(res.data);
                })
                .catch((error) => {
                    setErrorText("Данный пользователь не найден. Проверьте
логин и пароль");
                    setIsError(true);
                    console.log(error);
                })
        }
        else {
            setErrorText("Проверьте правильность email и длину пароля
(не менее 6 символов)");
            setIsError(true);
        }
    }
    return (
        <div className="Enter">
            <div className="content">
                <div className='text'>Вход</div>
                <input      onChange={ (e)      =>
{setUserLogin(e.target.value)}} placeholder='Ваш  логин'  type  =
"text" />
                <input      onChange={ (e)      =>
{setUserPassword(e.target.value)}} placeholder='Ваш  пароль' type  =
"password" />
                <p onClick={login}>Регистрация</p>
                {isError ? <ErrorBox showModal = {showModal} text =
{errorText} /> : null }
                <button onClick={enter}>Войти</button>
            </div>
        </div>
    );
}

export default Enter;

import './login.scss';
import $api from '../.../api';
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import logo from '../.../Images/logo.png';
import ErrorBox from '../errorBox/errorBox';
import SERVADDRESS from "../.../servAdress";

```

```

function Login() {
  const [userEmail, setUserEmail] = useState();
  const [userPassword, setUserPassword] = useState();
  const [isError, setIsError] = useState(false);
  const history = useNavigate();

  const goToEnter = () => {
    history('/Enter')
  }
  const registration = () => {
    let rule = /^[\\w]{1}[\\w-\\.]*@[\\w-]+\\. [a-z]{2,4}$/i;
    const user = {
      userLogin: userEmail,
      userPassword: userPassword
    }
    if ((userPassword.length >= 6) && (rule.test(userEmail))) {
      $api.post(SERVADDRESS + '/auth/registration', user)
        .then((res) => {
          localStorage.setItem("token", res.data.token);
          history("/Profile");
          console.log(res.data);
        })
        .catch((error) => {
          console.log(error);
        })
    }
    else {
      setIsError(true);
      return;
    }
  }
  return (
    <div className="Login">
      <div className="login__form">
        { isError ? <ErrorBox showModal = {( ) => setIsError(false)}
text={"Проверьте длину пароля (не менее 6) и правильность введенного
вами email"} /> : null }
        <div className="content">
          <div>Регистрация</div>
          <input onChange={ (e) => setUserEmail(e.target.value)}
type = "text" placeholder='Введите логин' />
          <input      onChange={ (e)      =>
setUserPassword(e.target.value)}      type      =      "password"
placeholder='Введите пароль' />
          <p onClick={goToEnter}>Войти</p>
          <button onClick={registration}>Регистрация</button>
        </div>
      </div>
    </div>
  );
}

export default Login;

import { useState } from 'react';

```

```

import $api from '../../../api';
import './createFolderModal.scss';
import SERVADDRESS from "../../../servAdress";

function CreatedFolderModal({hide, choosenFolderId,
updateUserFolders}) {
  const [folderName, setFolderName] = useState("");

  const createFolder = async () => {
    await $api.post(SERVADDRESS + '/createFolder', {folderName:
folderName, parentFolderId: choosenFolderId})
    .then((res) => {
      updateUserFolders(res.data.folders);
      hide();
      console.log(res.data);
    })
    .catch((error) => {
      console.log(error);
    })
  }

  return (
    <div className="createFolderModal">
      <div className="content">
        <div onClick={hide} className="close__button">x</div>
        <input onChange={(e) => setFolderName(e.target.value)}
type = "text" placeholder='Имя папки' />
        <button onClick ={createFolder}>Подтвердить</button>
      </div>
    </div>
  );
}
export default CreatedFolderModal;

```

Приложение Б
(рекомендуемое)

Программный код серверной части приложения

```

const User = require("../models/User")
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const jwt_decode = require('jwt-decode')

const generateAccessToken = (id) => {
  const payload = {
    id: id,
  }
  return jwt.sign(payload, "SECRET_KEY_RANDOM", {expiresIn:
"24h"});
}
class authController {
  async registration(req, res) {
    try {
      const userLogin = req.body.userLogin;
      const userPassword = req.body.userPassword;
      const balance = 5000;
      const candidate = await User.find({userLogin})

      if (candidate.length !== 0) {
        res.status(400).json({message: "This email is
already used"})
      }
      else {
        const hashPassword =
bcrypt.hashSync(userPassword, 7);
        const user = new User({
          userLogin,
          userPassword: hashPassword,
          balance: balance,
          avatar: "http://localhost:5000/" +
"avatar.png",
          secretAccessCode: userPassword,
          memory: 8192,
          bucket: []
        });
        const token = generateAccessToken(user._id);
        user.save();
        res.json({message: "Registration succeeded!",
token: token});
      }
    }
    catch (e) {
      console.log(e);
      res.status(400).json({message: "Registration
error!"});
    }
  }
}

```

```

    async login(req, res) {
      try {
        const {userLogin, userPassword} = req.body;
        const user = await User.findOne({userLogin});

        if(!user) {
          return res.status(400).json({message: `User
${userLogin} not found`});
        }

        const validPassword =
bcrypt.compareSync(userPassword, user.userPassword);
        if (!validPassword) {
          return res.status(400).json({message: `Uncorrect
password!`});
        }
        else {
          const token = generateAccessToken(user._id);
          return res.json({message: `Login succeeded`,
token: token});
        }
      }
      catch (e) {
        console.log(e);
        res.status(400).json({message: "Login error!"});
      }
    }

    async users(req, res) {
      try {
        res.json('server works!')
      }
      catch (e) {
        console.log(e);
      }
    }

    async getUserData(req, res) {
      const token = req.headers.authorization;
      const userId = jwt_decode(token);
      let isEmpty = await User.findOne({_id: userId.id});
      if (isEmpty) {
        return res.status(200).json({message: "Sucessfull",
userData: {
          //balance: isEmpty.balance,
          login: isEmpty.userLogin,
          id: isEmpty._id,
          avatar: isEmpty.avatar,
          files: isEmpty.files,
          secretAccessCode: isEmpty.secretAccessCode,
          folders: isEmpty.folders,
          //

```

```

        memory: isEmpty.memory,
        bucket: isEmpty.bucket
    }
    }
    else {
        return res.status(400).json("Geting data error! No
such user!")
    }
}

async changeUserBalance(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        await User.updateOne({ _id: userId.id }, { $set: {
balance: req.body.balance } })
        res.json({message: "Balance updated", newBalance:
req.body.balance});
    } catch (e) {
        console.log(e);
        res.status(400).json({message: "Error!"})
    }
}
}

module.exports = new authController()
const cors = require('cors');
const express = require('express');
const mongoose = require('mongoose');
const authRouter = require('./authRoutes');
const profileRouter = require('./Profile/profileRoutes');
const filesRouter = require('./Files/filesRoutes');
const adminRouter = require('./AdminPanel/adminPanelRoutes');
const fileUpload = require('express-fileupload');
const User = require("./models/User");
const jwt_decode = require('jwt-decode');
const fs = require('fs');

const PORT = 5000;
const BD_URL =
`mongodb+srv://danila:355473288@cluster0.ag5ij.mongodb.net/?r
etryWrites=true&w=majority`;
const app = express();
app.use(cors({origin: '*'}));
mongoose.set('strictQuery', false);

/*var corsOptions = {
    origin: '*',
    optionsSuccessStatus: 200 // some legacy browsers (IE11,
various SmartTVs) choke on 204
}*/

```

```

app.use(fileUpload())
app.use(express.json());
app.use('/auth', authRouter);
app.use('/profile', profileRouter);
app.use('/files', filesRouter);
app.use('/admin', adminRouter);
app.use(express.static('./userAvatars'));

app.post("/test", cors(), async function(req, res) {
    res.status(200).json({message: "Hello world!"});
});

app.post("/upload", async function (req, res) {
    if (req.files && Object.keys(req.files).length !== 0) {

        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        let isEmpty = await User.findOne({_id: userId.id});

        const uploadedFile = req.files.uploadFile;
        const uploadPath = __dirname + "/userAvatars/" +
isEmpty.userLogin + "_" + uploadedFile.name;
        await User.updateOne({_id: isEmpty._id }, { $set: {
avatar: "http://localhost:5000/" + isEmpty.userLogin + "_" +
uploadedFile.name } });

        uploadedFile.mv(uploadPath, function (err) {
            if (err) {
                console.log(err);
                res.send("Failed !!");
            }
            else res.json({message: "Successfully Uploaded !!",
avatar: "http://localhost:5000/" + isEmpty.userLogin + "_" +
uploadedFile.name});
        });
    }
    else res.send("No file uploaded !!");
});

app.post("/uploadFiles", async function (req, res) {
    if (req.files && Object.keys(req.files).length !== 0) {

        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        let isEmpty = await User.findOne({_id: userId.id});

        const folderId = Number(req.body.folderId);
        const uploadedFile = req.files.uploadFile;
        let uploadPath = __dirname + "/userAvatars/userFiles/"
+ isEmpty.userLogin + "_" + uploadedFile.name;

        //Получаем дату загрузки файла

```

```

        let utc = new Date().toISOString().slice(0,10).replace(/-/g, '/');

        const obj = {
            path: "http://localhost:5000/userFiles/" +
isEmpty.userLogin + "_" + uploadedFile.name,
            fileName: uploadedFile.name,
            size: (uploadedFile.size/1048576).toFixed(2),
            type: uploadedFile.mimetype,
            folderId: folderId,
            status: "public",
            date: utc
        }

        if (fs.existsSync(uploadPath)) {
            uploadPath = __dirname + "/userAvatars/userFiles/"
+ isEmpty.userLogin + "_" + "1" + uploadedFile.name;
            obj.fileName = "1"+uploadedFile.name;
            obj.path = "http://localhost:5000/userFiles/" +
isEmpty.userLogin + "_" + "1" + uploadedFile.name;
            //res.status(400).json({message: "Данный файл уже
существует в вашем хранилище"});
        }

        await User.updateOne({ _id: isEmpty._id }, { $set:
{ files: [...isEmpty.files, obj]}
        });

        uploadedFile.mv(uploadPath, function (err) {
            if (err) {
                console.log(err);
                res.send("Failed !!");
            }
            else res.json({message: "Succesfully uploaded!",
files: [...isEmpty.files, obj]});
        });

    }
    else res.send("No file uploaded !!");
});
app.post("/createFolder", async function (req, res) {
    const token = req.headers.authorization;
    const userId = jwt_decode(token);
    let currentUser = await User.findOne({_id: userId.id});

    try {
        await User.updateOne({ _id: currentUser._id }, { $set:
{ folders: [...currentUser.folders, {
            folderName: req.body.folderName,
            folderId: currentUser.folders.length + 1,
            parentFolderId: req.body.parentFolderId,

```

```

        status: "public",
        isDeleted: false
    ]}}});
    currentUser.folders = [...currentUser.folders, {
        folderName: req.body.folderName,
        folderId: currentUser.folders.length + 1,
        parentFolderId: req.body.parentFolderId,
        status: "public",
        isDeleted: false
    }]
    res.status(200).json({
        message: "Folder created sucessfull",
        folders: currentUser.folders.filter(el =>
el.parentFolderId === req.body.parentFolderId)
    })
    } catch (err) {
        console.log(err);
    }
});
app.post("/getFilesFromFolder", async function(req, res) {
    const token = req.headers.authorization;
    const userId = jwt_decode(token);
    let currentUser = await User.findOne({_id: userId.id});

    let newFiles = currentUser.files.filter(el => el.folderId
=== req.body.folderId);
    let newFolders = currentUser.folders.filter(el =>
el.parentFolderId === req.body.folderId);

    res.status(200).json({message: "Getting file sucessfull",
folders: newFolders, files: newFiles});
});
app.post("/changeFolderStatus", async function(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        const newfolders = user.folders;
        const newFolder = req.body.folder;

        let indexOfChangedFolder = newfolders.findIndex(el =>
el.folderId === newFolder.folderId);
        newfolders[indexOfChangedFolder] = newFolder;

        await User.updateOne({ _id: userId.id }, { $set: {
folders: newfolders } });

        res.json({message: "Sucessfull!", folders: newfolders})
    }
    catch (e) {

```

```

        console.log(e);
        res.status(400).json({message: "Error!"});
    }
})
app.post("/shareUserMemory", async function(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        const userToShare = await User.findOne({_id:
req.body.userId});

        await User.updateOne({ _id: user._id }, { $inc: {
memory: -req.body.memory } });
        await User.updateOne({ _id: userToShare._id }, { $inc:
{ memory: req.body.memory } });

        res.status(200).json({message: "Вы успешно передали
память другому пользователю!"});
    }
    catch (error) {
        res.status(200).json({message: "Error!"});
        console.log(error);
    }
})
async function startApp() {
    try {
        await mongoose.connect(BD_URL);
        app.listen(PORT, () => console.log('Server started at
PORT' + " " + PORT));
    } catch (error) {
        console.log(error);
    }
}
startApp();

const User = require("../models/User")
const jwt_decode = require('jwt-decode')
const fs = require('fs');

class profileController {
    async changeFileName(req, res) {
        try {
            const token = req.headers.authorization;
            const userId = jwt_decode(token);
            const user = await User.findOne({_id: userId.id});

            const newfiles = user.files;
            const fileName = req.body.fileName;

```

```

        let indexOfChangedFile = newfiles.findIndex(el =>
el.fileName == fileName);
        newfiles[indexOfChangedFile] = {
            path: newfiles[indexOfChangedFile].path,
            fileName: req.body.newName,
            size: newfiles[indexOfChangedFile].size,
            type: newfiles[indexOfChangedFile].type,
            folderId: newfiles[indexOfChangedFile].folderId,
            status: newfiles[indexOfChangedFile].status,
            date: newfiles[indexOfChangedFile].date
        };

        await User.updateOne({ _id: userId.id }, { $set: {
files: newfiles } });

        res.json({message: "Sucessfull!", files: newfiles})
    }
    catch (e) {
        console.log(e);
        res.status(400).json({message: "Error!"});
    }
}
async deleteFile(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});
        const path = `./userAvatars/" +
req.body.deletedFilePath.replace('http://localhost:5000/',
'');

        let newFilesArray = user.files.filter(element =>
element.path !== req.body.deletedFilePath);

        let newBucketArray = user.bucket;
        let deletedUserFile = user.files.filter(el =>
el.path == req.body.deletedFilePath);
        newBucketArray = [...newBucketArray,
deletedUserFile[0]];

        if (newBucketArray.length > 10) {
            let firstElement = newBucketArray.shift();
            const deletedPath = `./userAvatars/" +
firstElement.path.replace('http://localhost:5000/', '');
            fs.unlinkSync(deletedPath);
        }

        await User.updateOne({ _id: userId.id }, { $set: {
files: newFilesArray, bucket: newBucketArray } });

```

```

        //fs.rmdir(path, {recursive: true}, (err) =>
console.log(err));

        res.status(200).json({message: "File was deleted
succesfull", files: newFilesArray, bucket: newBucketArray})
    }
    catch (e) {
        console.log(e);
        res.status(400).json({message: "Error!"});
    }
}
async deleteFolder(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        let arr = [req.body.folderId];
        user.folders.map(el => {
            if (arr.includes(el.parentFolderId)) {
                arr = [...arr, el.folderId];
            }
        })
        let newFoldersArray = user.folders.filter(el =>
!arr.includes(el.folderId));
        let newFilesArray = user.files.filter(el =>
!arr.includes(el.folderId));
        let newUserBucket = user.bucket.filter(el =>
!el.folderId == req.body.folderId);

        user.files.map((el) => {
            if (arr.includes(el.folderId)) {
                const path = `./userAvatars/" +
el.path.replace('http://localhost:5000/', '');
                fs.unlinkSync(path);
            }
        })

        await User.updateOne({ _id: userId.id }, { $set: {
            folders: newFoldersArray,
            files: newFilesArray,
            bucket: newUserBucket }
        });
        res.status(200).json({
            message: "Folder was deleted succesfull",
            folders: newFoldersArray.filter(el =>
el.parentFolderId === req.body.parentFolderId),
            files: newFilesArray,
            bucket: newUserBucket
        })
    }
}

```

```

        catch (e) {
            console.log(e);
            res.status(400).json({message: "Error!"});
        }
    }
    async changeFolderName(req, res) {
        try {
            const token = req.headers.authorization;
            const userId = jwt_decode(token);
            const user = await User.findOne({_id: userId.id});

            const userFolders = user.folders.filter(e =>
e.folderId !== req.body.folderIndex)
            const newFolder = user.folders.filter(e =>
e.folderId == req.body.folderIndex);
            newFolder[0].folderName = req.body.folderName;
            const newFoldersArray = [...userFolders,
newFolder[0]];
            await User.updateOne({ _id: userId.id }, { $set: {
folders: newFoldersArray } });

            res.json({message: "Sucessfull!", folders:
newFoldersArray});
        }
        catch (e) {
            console.log(e);
            res.status(400).json({message: "Could not change
folder name!"})
        }
    }
    async changeFileStatus(req, res) {
        try {
            const token = req.headers.authorization;
            const userId = jwt_decode(token);
            const user = await User.findOne({_id: userId.id});

            const newfiles = user.files;
            const newFile = req.body.file;
            let indexOfChangedFile = newfiles.findIndex(el =>
el.path == newFile.path);
            newfiles[indexOfChangedFile] = newFile;

            await User.updateOne({ _id: userId.id }, { $set: {
files: newfiles } });

            res.json({message: "Sucessfull!", files: newfiles})
        }
        catch (e) {
            console.log(e);
            res.status(400).json({message: "Error!"});
        }
    }
}

```

```

    }
    async recoverUserFile(req, res) {
      try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        let newUserFiles = user.files;
        newUserFiles = [...newUserFiles, req.body.file];
        let newUserBucket = user.bucket.filter((el) =>
el.path !== req.body.file.path);

        await User.updateOne({ _id: userId.id }, { $set: {
files: newUserFiles, bucket: newUserBucket } });

        res.status(200).json({message: "Sucessfully
recovered!", files: newUserFiles, bucket: newUserBucket});
      }
      catch (error) {
        res.status(200).json({message: "Error!"})
        console.log(error);
      }
    }
    async deleteFileFromBucket(req, res) {
      try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        let newBucket = user.bucket.filter((el) => el.path
!== req.body.file.path);

        const path = "./userAvatars/" +
req.body.file.path.replace('http://localhost:5000/', '');
        fs.unlinkSync(path);

        await User.updateOne({ _id: userId.id }, { $set: {
bucket: newBucket } });
        res.status(200).json({message: "Sucessfully
deleted!", bucket: newBucket});
      }
      catch (error) {
        res.status(200).json({message: "Error!"})
        console.log(error);
      }
    }
    async moveFolderToBucket(req, res) {
      try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

```

```

        let newBucketArray = user.bucket;
        let deletedFolder = req.body.folder;
        deletedFolder.isDeleted = true;
        newBucketArray = [...newBucketArray, deletedFolder];

        let newFoldersArray = user.folders.filter(el =>
el.folderId !== deletedFolder.folderId);
        newFoldersArray = [...newFoldersArray,
deletedFolder];

        await User.updateOne({ _id: userId.id }, { $set:
        {
            folders: newFoldersArray,
            bucket: newBucketArray
        }
        });

        res.status(200).json({
            message: "Folder now in your bucket",
            folders: newFoldersArray.filter(el =>
el.parentFolderId === req.body.parentFolderId),
            bucket: newBucketArray
        })
    }
    catch (error) {
        res.status(400).json({
            message: "Error!",
        })
    }
}

async recoverFolderFromBucket(req, res) {
    try {
        const token = req.headers.authorization;
        const userId = jwt_decode(token);
        const user = await User.findOne({_id: userId.id});

        let folderForRecovering = req.body.folder;
        folderForRecovering.isDeleted = false;

        let newFoldersArray = user.folders.filter(el =>
el.folderId !== folderForRecovering.folderId);
        newFoldersArray = [...newFoldersArray,
folderForRecovering];

        let newBucketArray = user.bucket.filter(el =>
el.folderId !== folderForRecovering.folderId);

        await User.updateOne({ _id: userId.id }, { $set: {
            folders: newFoldersArray,

```

```

        bucket: newBucketArray
      }
    });
    res.status(200).json({message: "folderSuccessfully
recovered!", bucket: newBucketArray});
  }
  catch (error) {
    res.status(400).json({message: "Error!"});
    console.log(error);
  }
}

}

module.exports = new profileController()
const bcrypt = require('bcryptjs')
const User = require("../models/User")
const jwt_decode = require('jwt-decode')

class profileController {
  async changeUserPassword(req, res) {
    try {
      const userPassword = req.body.newPassword;
      const hashPassword = bcrypt.hashSync(userPassword,
7);

      const token = req.headers.authorization;
      const userId = jwt_decode(token);

      await User.updateOne({ _id: userId.id }, { $set: {
userPassword: hashPassword } });
      res.status(200).json({message: "Password
changed!"})
    }
    catch (e) {
      console.log(e);
      res.status(400).json({message: "Error!"});
    }
  }
  async changeUserSecretCode(req, res) {
    try {
      const newSecretAccessCode =
req.body.newSecretAccessCode;
      const token = req.headers.authorization;
      const userId = jwt_decode(token);

      await User.updateOne({ _id: userId.id }, { $set: {
secretAccessCode: newSecretAccessCode } });
      res.status(200).json({message: "Your secret code
changed!"})
    }
    catch (e) {

```

```

        console.log(e);
        res.status(400).json({message: "Error!"});
    }
}
async getUserDataById(req, res) {
    try {
        const gettingDataId = req.body.id;

        let foundedUser = await User.findOne({_id:
gettingDataId});
        if (req.body.secretAccessCode ===
foundedUser.secretAccessCode) {
            return res.status(200).json({message:
"Succesfull", userData: {
                id: foundedUser._id,
                avatar: foundedUser.avatar,
                files: foundedUser.files,
                folders: foundedUser.folders,
                //
                memory: foundedUser.memory
            }})
        }
        else res.status(400).json({message: "Access dined!
Try again."})
    }
    catch (e) {
        console.log(e);
        res.status(400).json({message: "Error!"});
    }
}
}
module.exports = new profileController()

```

Приложение В
(рекомендуемое)
Результаты тестирования приложения

Таблица В.1 – Результаты тестирования интерфейса

Критерий оценки	Описание	Ожидаемый результат	Фактический результат	Итог	Время отклика
3	Первый вход пользователя на сайт	На экране появилась форма регистрации с возможностью переключения на форму входа	На экране появилась форма регистрации с возможностью переключения на форму входа	Результат тестирования представлен на рисунках 1–2 в приложении Г.	70мс
3	Появление модального окна при ошибке входа или регистрации	При совершении ошибки во время входа или регистрации появляется модальное окно с ошибкой	При совершении ошибки во время входа или регистрации появляется модальное окно с ошибкой	Результат тестирования представлен на рисунке 3 в приложении Г.	75мс
3	Появление формы личного кабинета (профиля) пользователя после успешного входа и регистрации	После успешной регистрации или входа должна появиться форма личного кабинета пользователя	После успешной регистрации или входа появилась форма личного кабинета пользователя	Результат тестирования представлен на рисунке 4 в приложении Г.	99мс
2	При заходе с мобильного устройства форма личного кабинета должна отображаться корректно	Корректное отображение формы личного пользователя на мобильном устройстве	Корректное отображение формы личного пользователя на мобильном устройстве	Результат тестирования представлен на рисунке 5 в приложении Г.	-
3	Переход в личное хранилище из личного кабинета	Корректное отображение файлов, папок и навигационного меню внутри хранилища	Корректное отображение файлов, папок и навигационного меню внутри хранилища	Результат тестирования представлен на рисунке 6 в приложении Г.	140мс

Продолжение таблицы В.1

3	Наведение на файл или папку внутри хранилища	Появление настроек вокруг файла или папки	Появление настроек вокруг файла или папки	Результат тестирования представлен на рисунке 7 в приложении Г.	20мс
3	Нажатие на имя файла или папки	Появление модального окна для смены имени файла или папки	Появление модального окна для смены имени файла или папки	Результат тестирования представлен на рисунке 8 в приложении Г.	20мс
3	Нажатие на кнопку создания папки	Появление модального окна создания папки	Появление модального окна создания папки	Результат тестирования представлен на рисунке 9 в приложении Г.	21мс
3	Нажатие на кнопку настроек файла или папки	Появление модального окна с настройками файла	Появление модального окна с настройками файла	Результат тестирования представлен на рисунке 10 в приложении Г.	79мс
3	Нажатие на кнопку перехода в корзину	Появление модального окна корзины и корректное отображение всех файлов и папок хранящихся в ней, присутствие кнопок удаления и восстановления файлов возле каждого файла	Появление модального окна корзины и корректное отображение всех файлов и папок хранящихся в ней, присутствие кнопок удаления и восстановления файлов возле каждого файла	Результат тестирования представлен на рисунке 11 в приложении Г.	120мс
3	Нажатие на кнопку share memory	Появление модального окна и корректное отображение элементов формы share memory	Появление модального окна и корректное отображение элементов формы share memory	Результат тестирования представлен на рисунке 12 в приложении Г.	100мс

Продолжение таблицы В.1

3	Переход в статистику файлов	Появление формы с графиками и меню выбора необходимого графика, корректное отображение всех графиков	Появление формы с графиками и меню выбора необходимого графика, корректное отображение всех графиков	Результат тестирования представлен на рисунке 13–14 в приложении Г.	190мс
1	Корректное позиционирование элементов формы регистрации и входа на мобильном устройстве	Элементы на странице имеют корректное позиционирование	Элементы на странице имеют корректное позиционирование	Результат тестирования представлен на рисунке 15–16 в приложении Г.	-
1	Корректное позиционирование элементов профиля на мобильном устройстве	Элементы на странице имеют корректное позиционирование	Элементы на странице имеют корректное позиционирование	Результат тестирования представлен на рисунке 17 в приложении Г.	-
1	Корректное позиционирование элементов хранилища на мобильном устройстве	Элементы на странице имеют корректное позиционирование	Элементы на странице имеют корректное позиционирование	Результат тестирования представлен на рисунке 18–19 в приложении Г.	-
1	Корректное позиционирование элементов модальных окон в хранилище	Элементы на странице имеют корректное позиционирование	Элементы на странице имеют корректное позиционирование	Результат тестирования представлен на рисунке 20 в приложении Г.	-

Окончание таблицы В.1

2	Текстовая часть профиля пользователя отображена корректно, текст виден отчетливо	Текстовое содержимое компонентов имеет читабельный вид	Текстовое содержимое компонентов имеет читабельный вид	Результат тестирования представлен на рисунке 21 в приложении Г.	-
2	Текстовая часть элементов хранилища отображена корректно, текст виден отчетливо	Текстовое содержимое компонентов имеет читабельный вид	Текстовое содержимое компонентов имеет читабельный вид	Результат тестирования представлен на рисунке 22 в приложении Г.	-

Проведенное тестирование показало, что интерфейс пользователя и каждый его элемент функционирует без ошибок и выполняет набор необходимых функций.

Приложение Г
(рекомендуемое)

Скриншоты тестирования пользовательского интерфейса

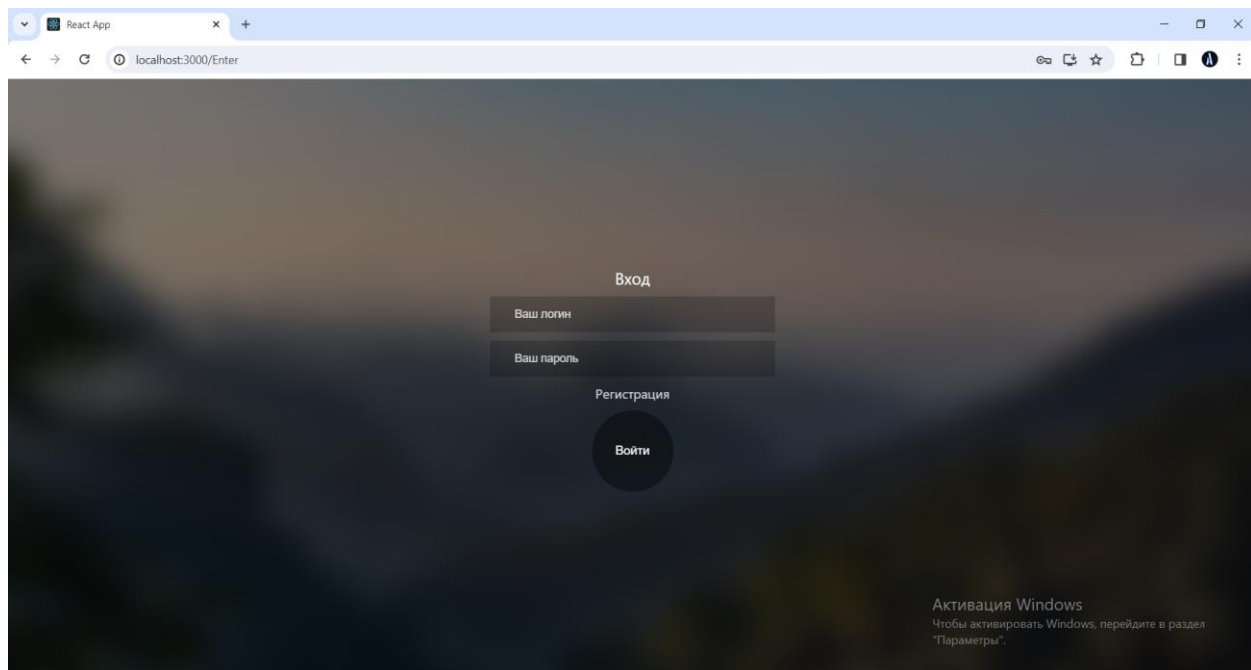


Рисунок Г.1 – Форма входа

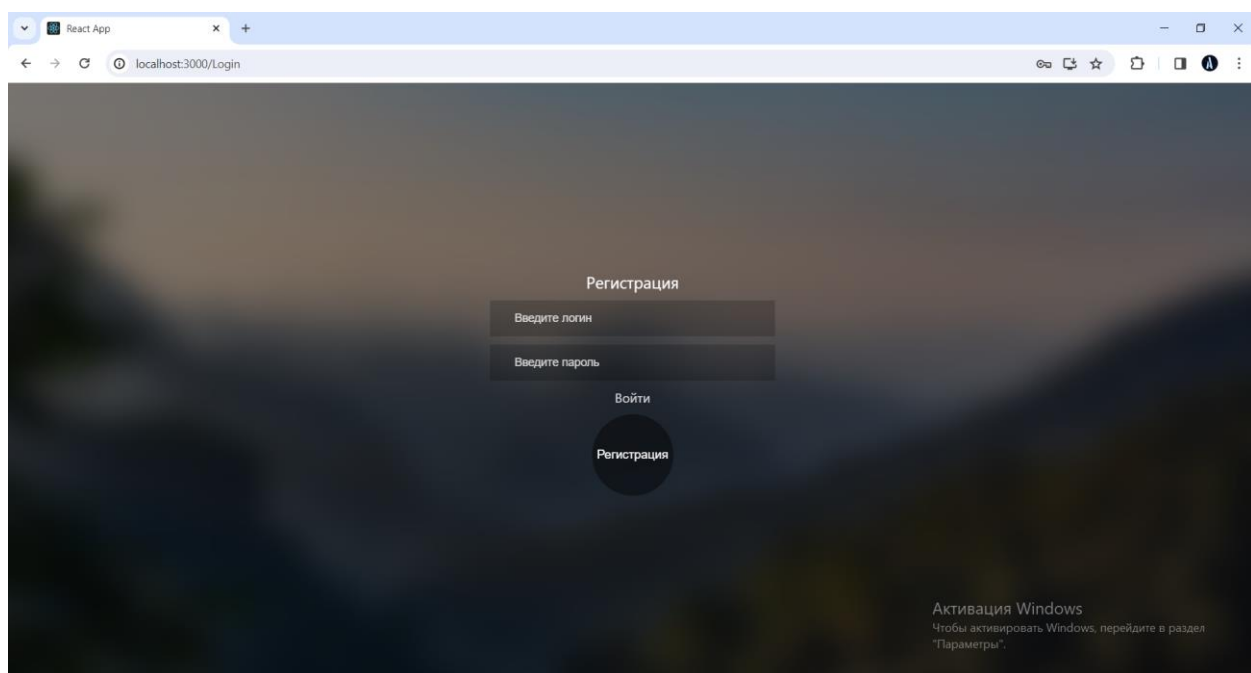


Рисунок Г.2 – Форма регистрации

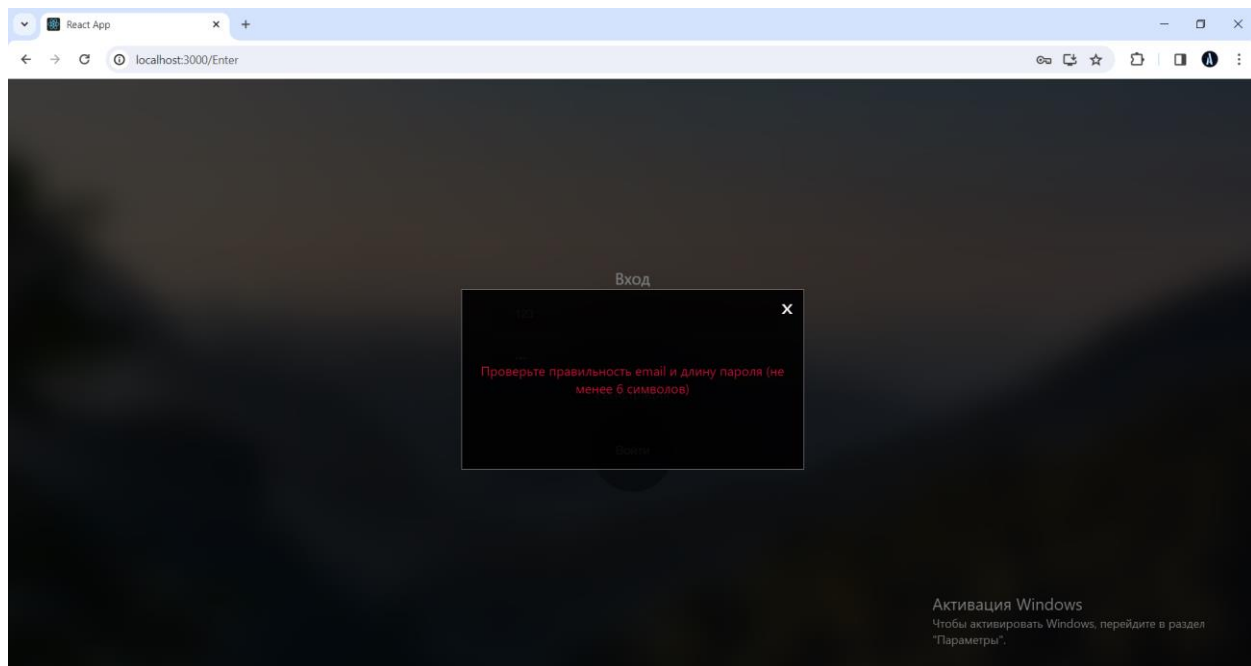


Рисунок Г.3 – Ошибка входа или регистрации пользователя

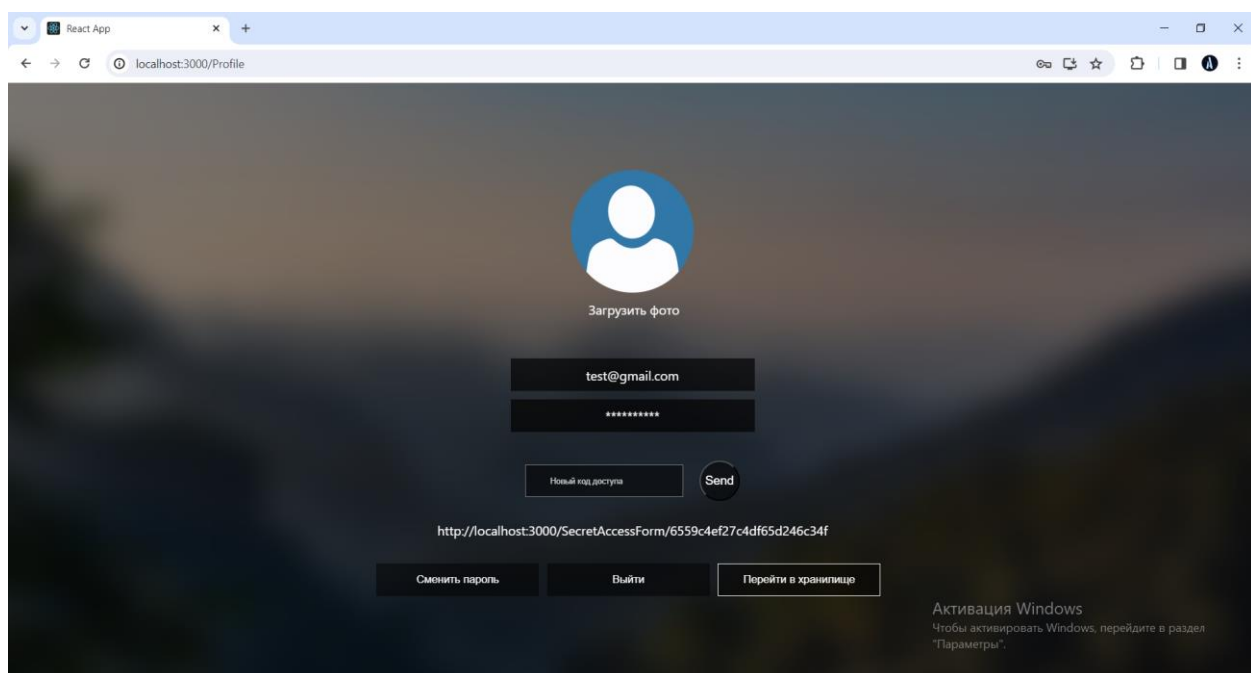


Рисунок Г.4 – Форма личного кабинета пользователя

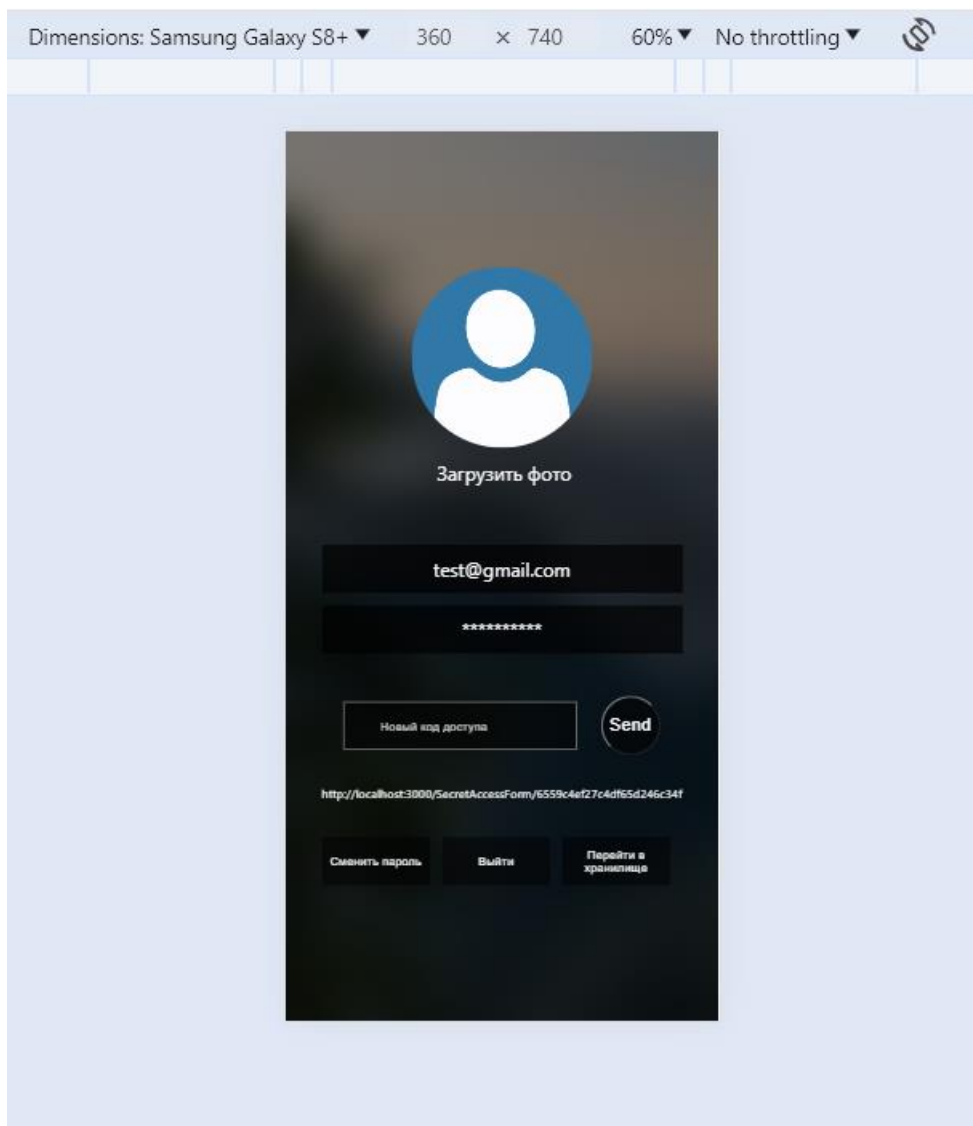


Рисунок Г.5 – Адаптация под мобильное устройство формы личного кабинета

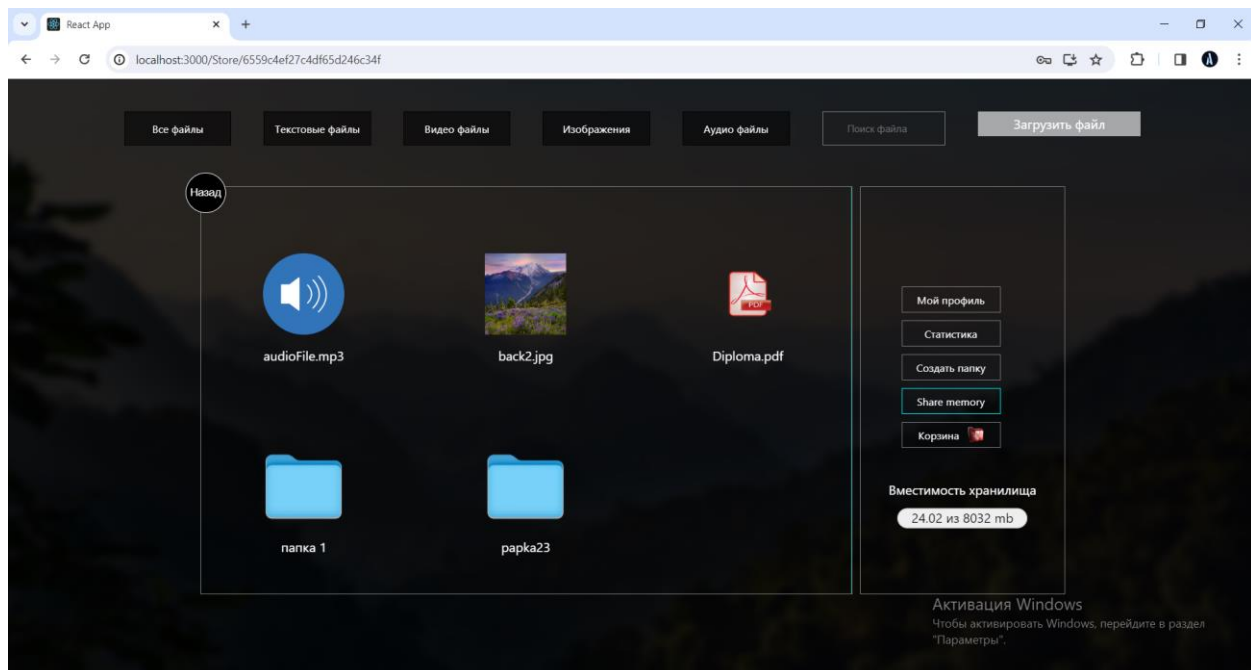


Рисунок Г.6 – Хранилище пользователя

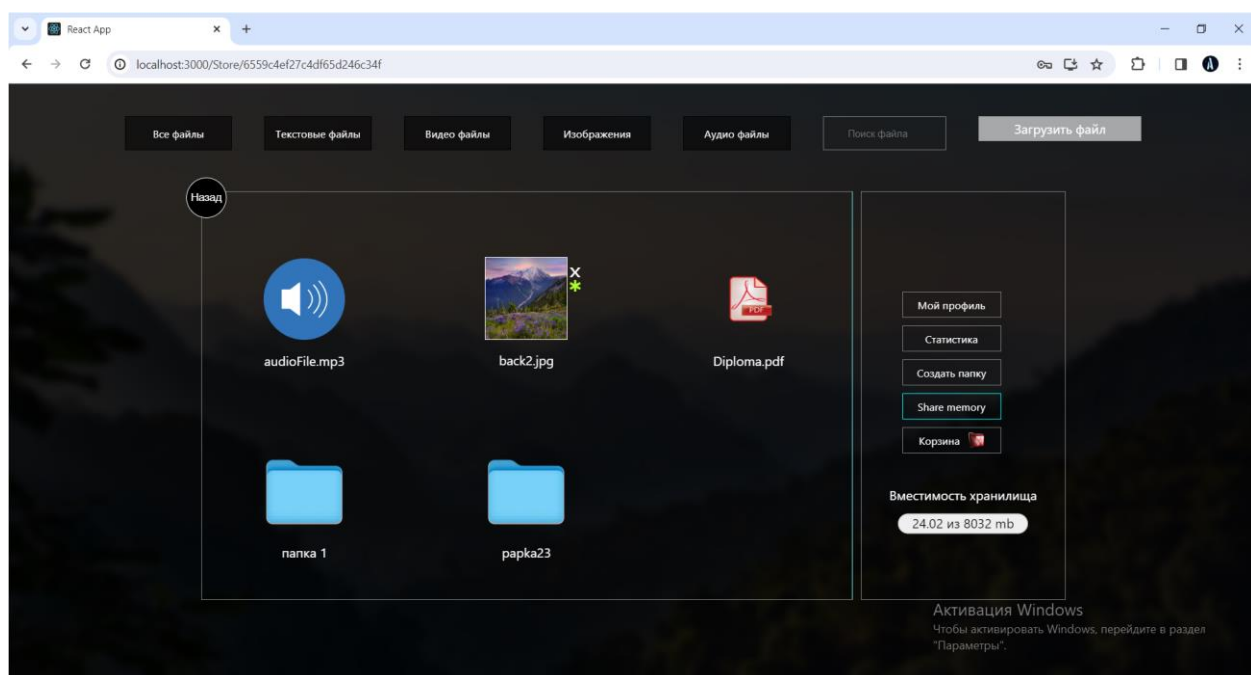


Рисунок Г.7 – Отображение настроек файла при наведении

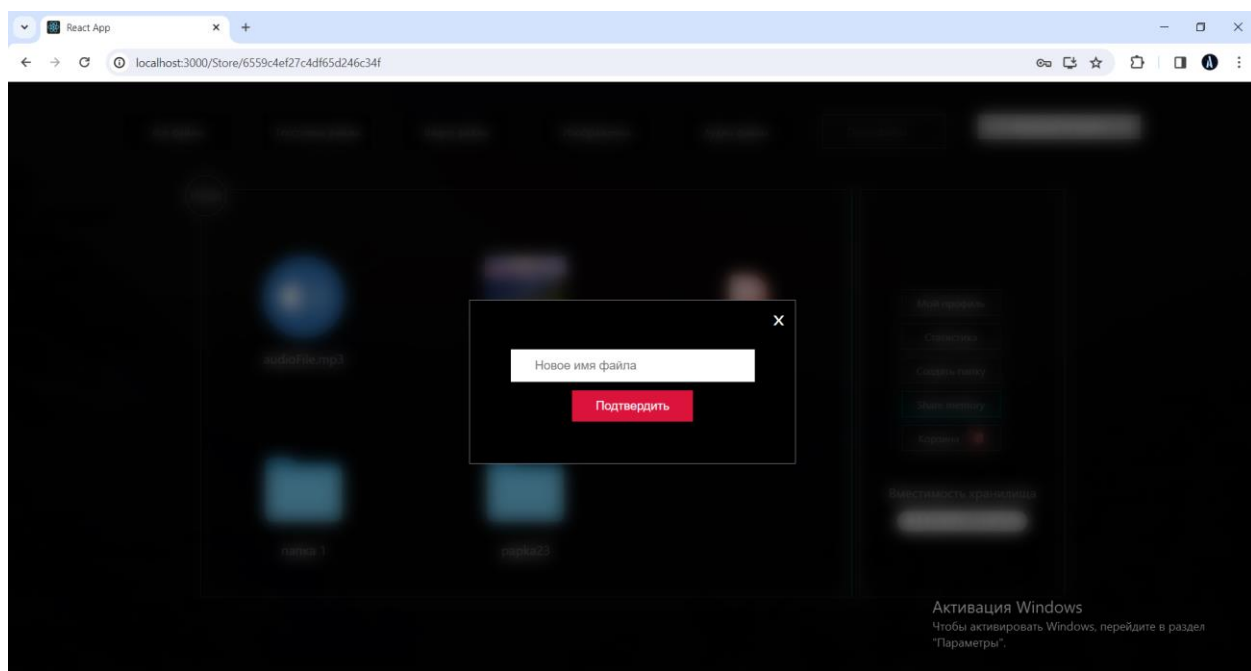


Рисунок Г.8 – Модальное окно изменения названия файла

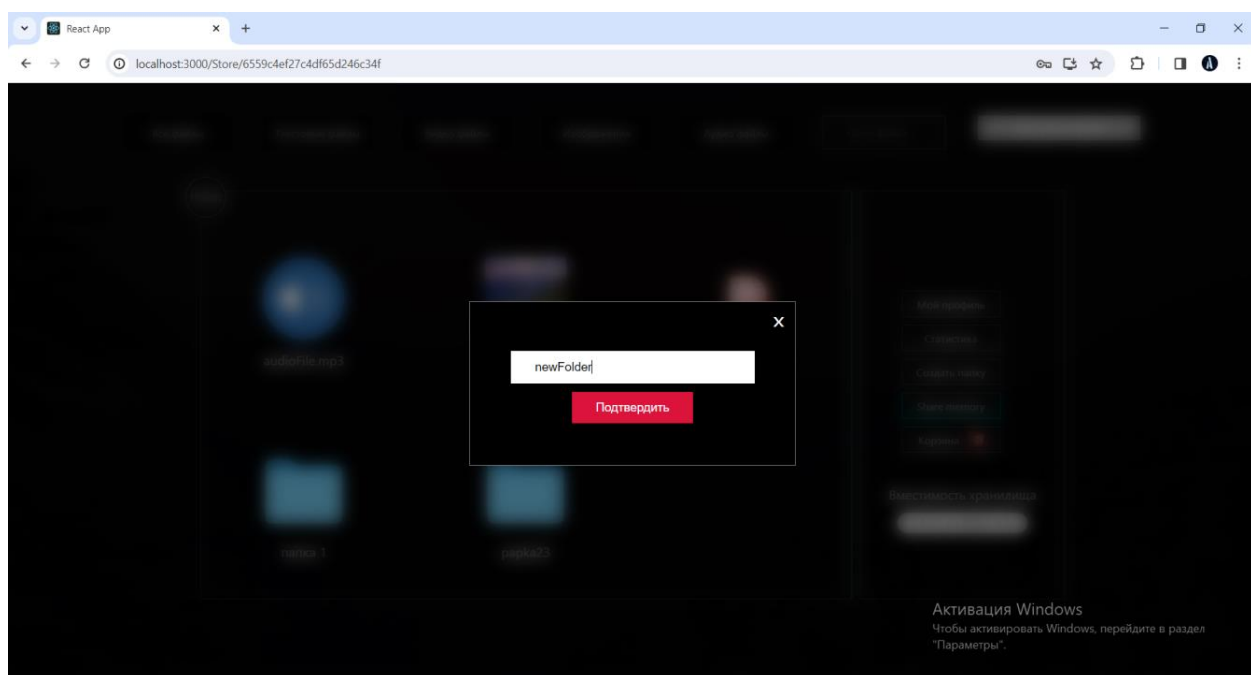


Рисунок Г.9 – Форма создания папки

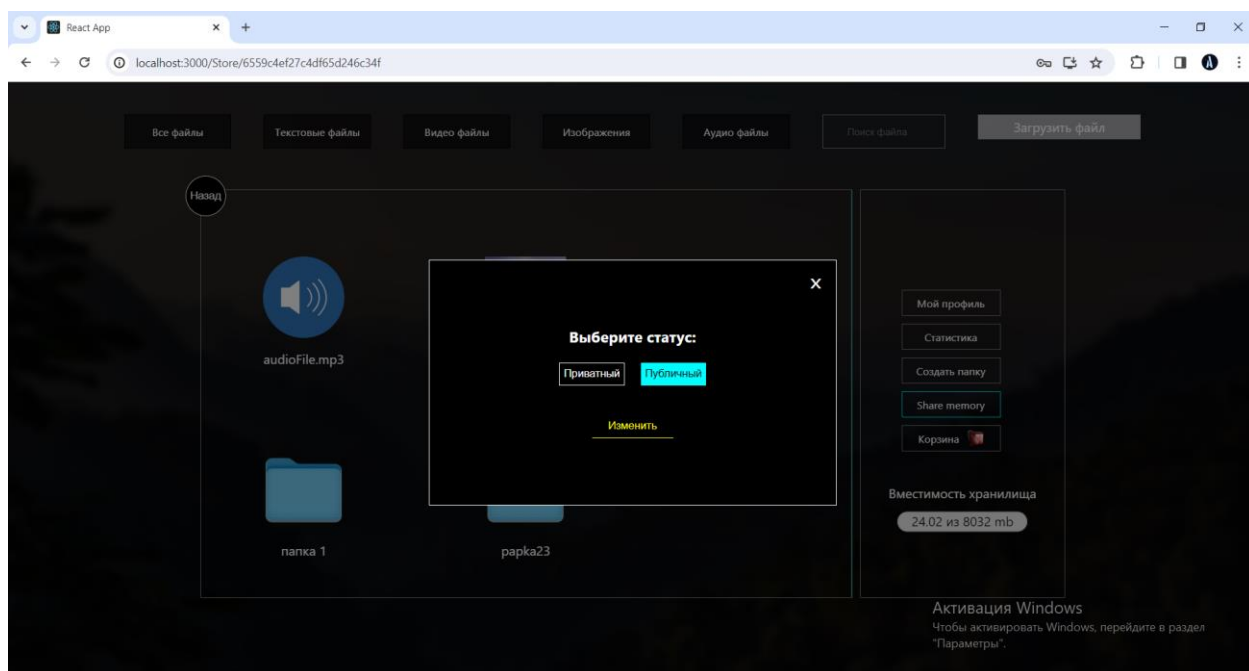


Рисунок Г.10 – Настройки файла

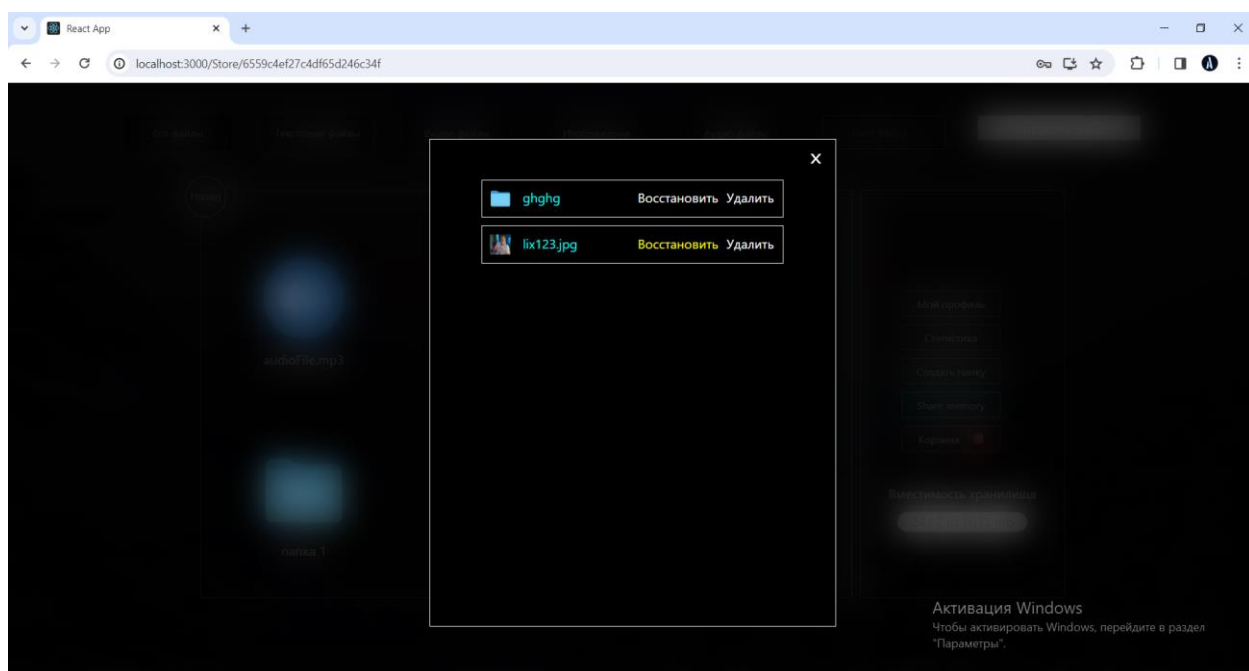


Рисунок Г.11 – Корзина

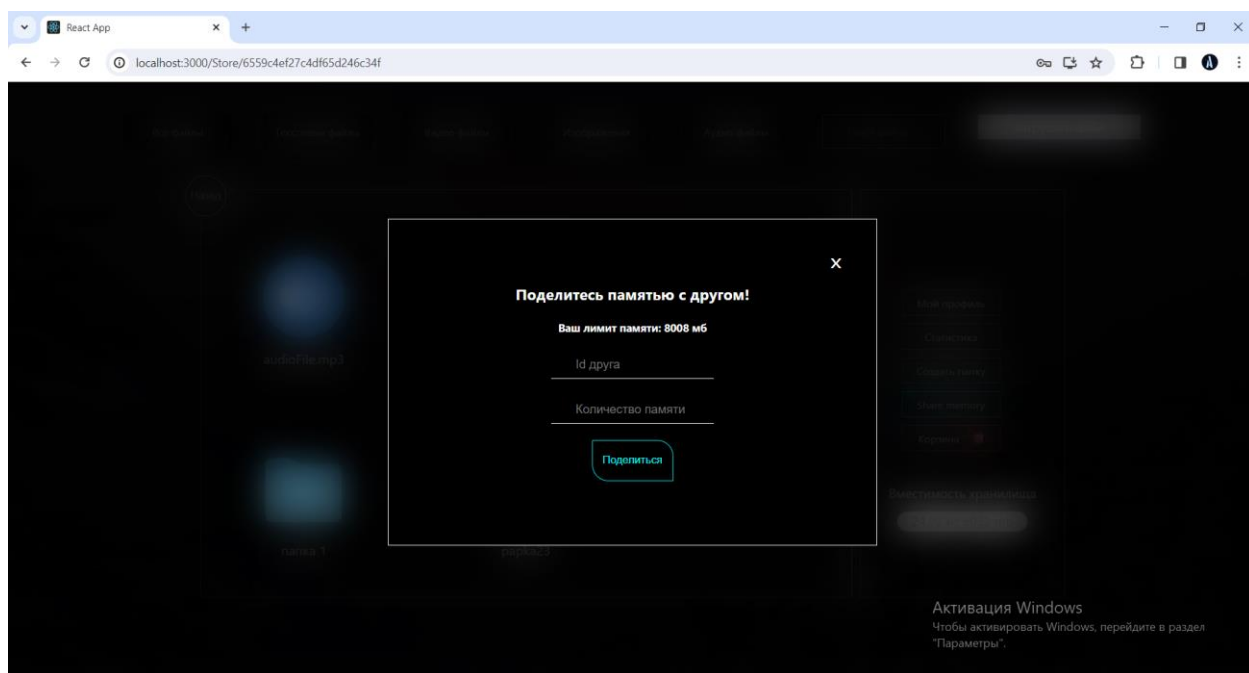


Рисунок Г.12 – Share memory

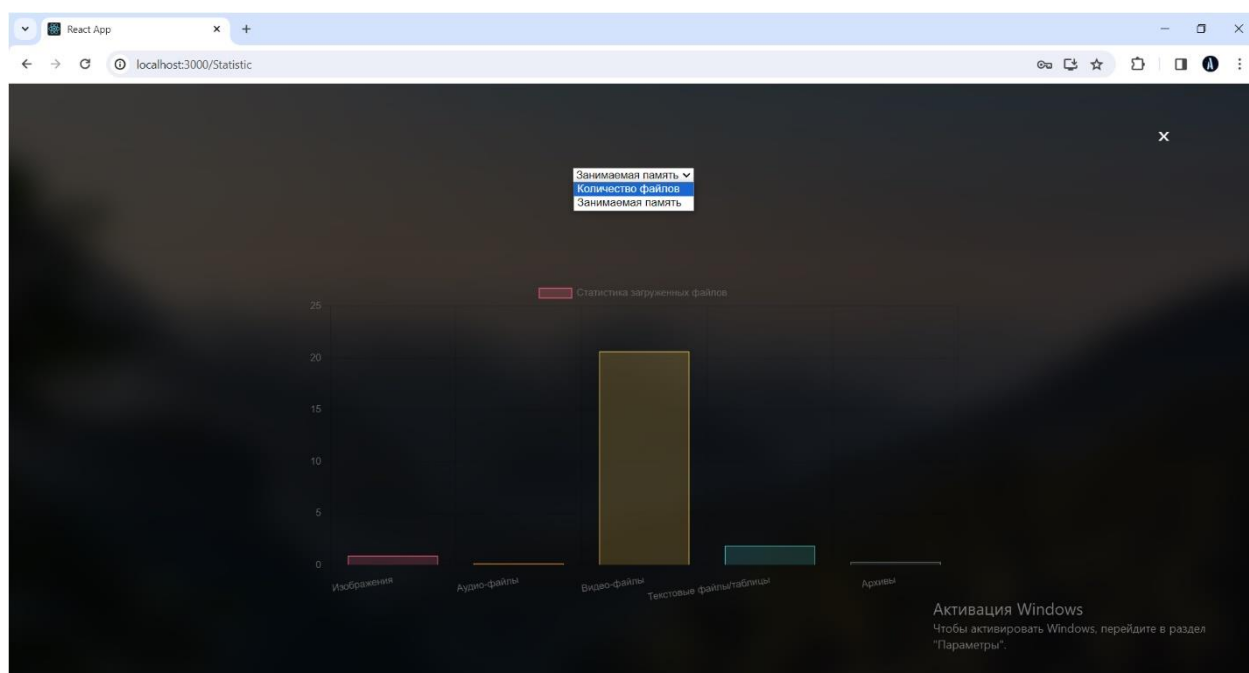


Рисунок Г.13 – Статистика файлов по памяти

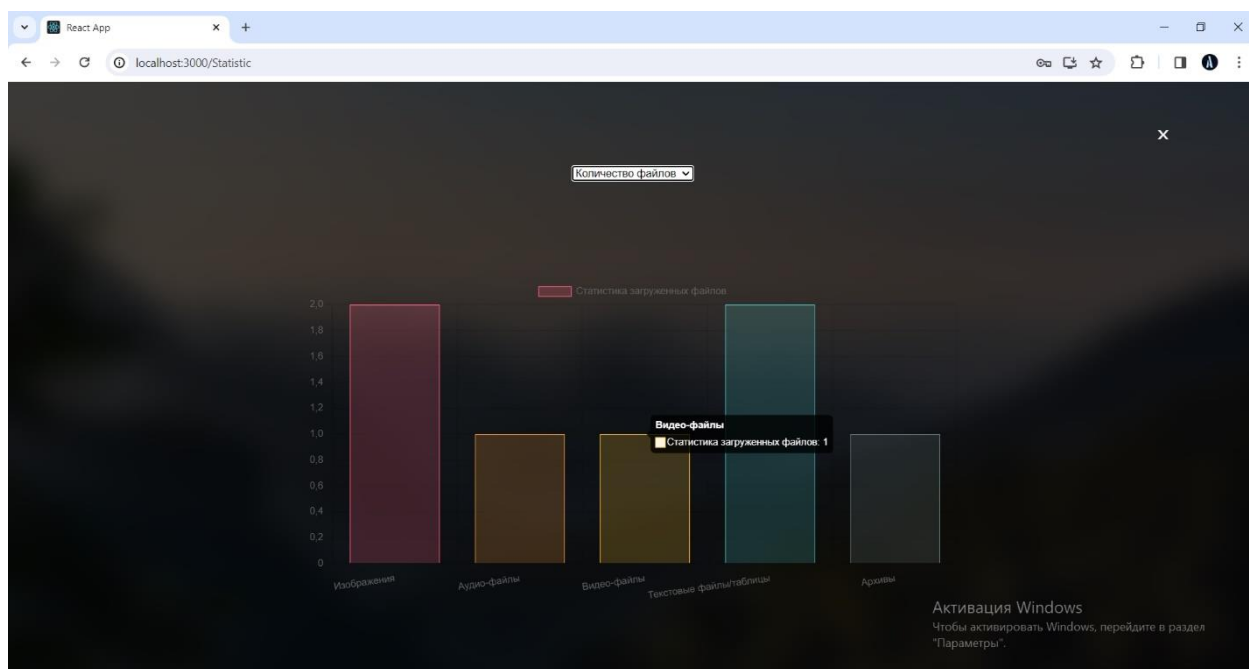


Рисунок Г.14 – Статистика файлов по количеству

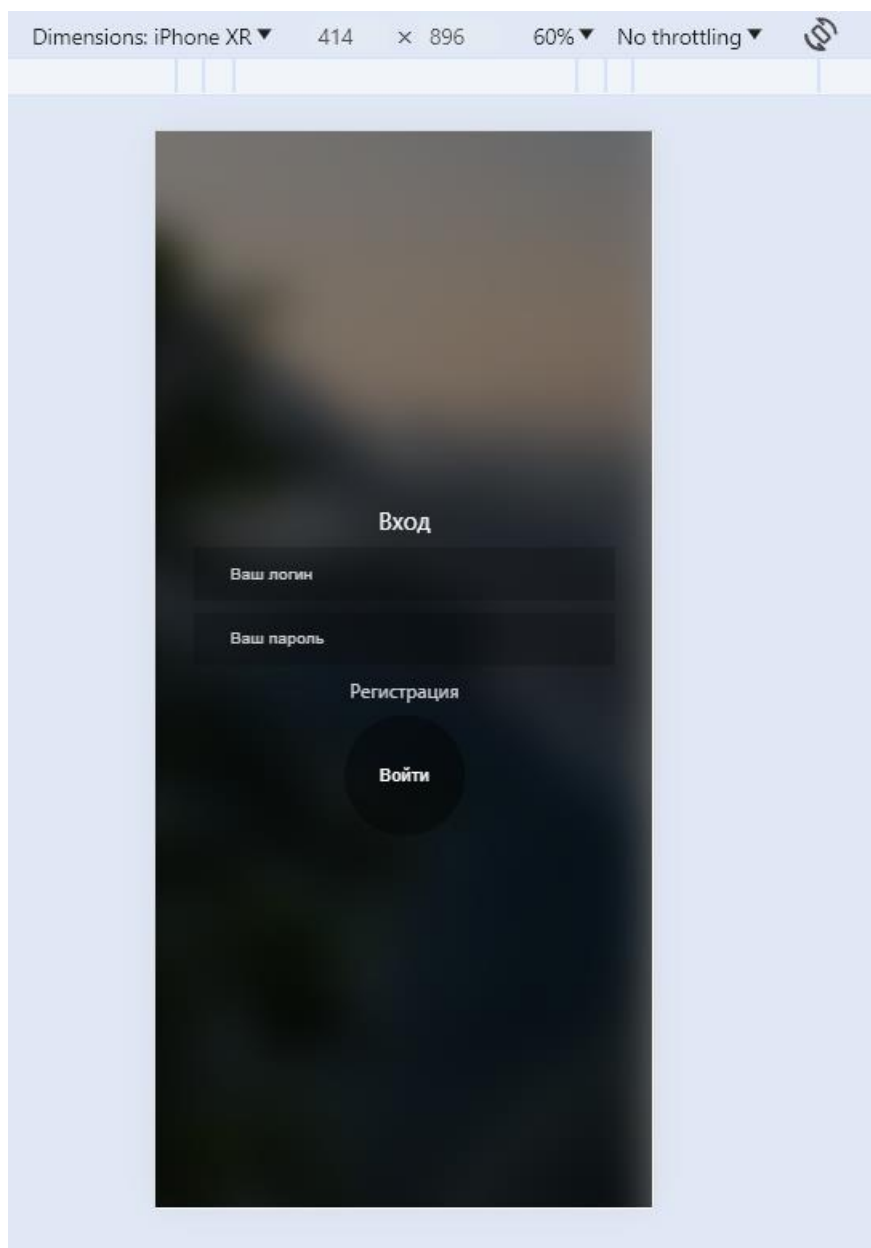


Рисунок Г.16 – Мобильная адаптация входа

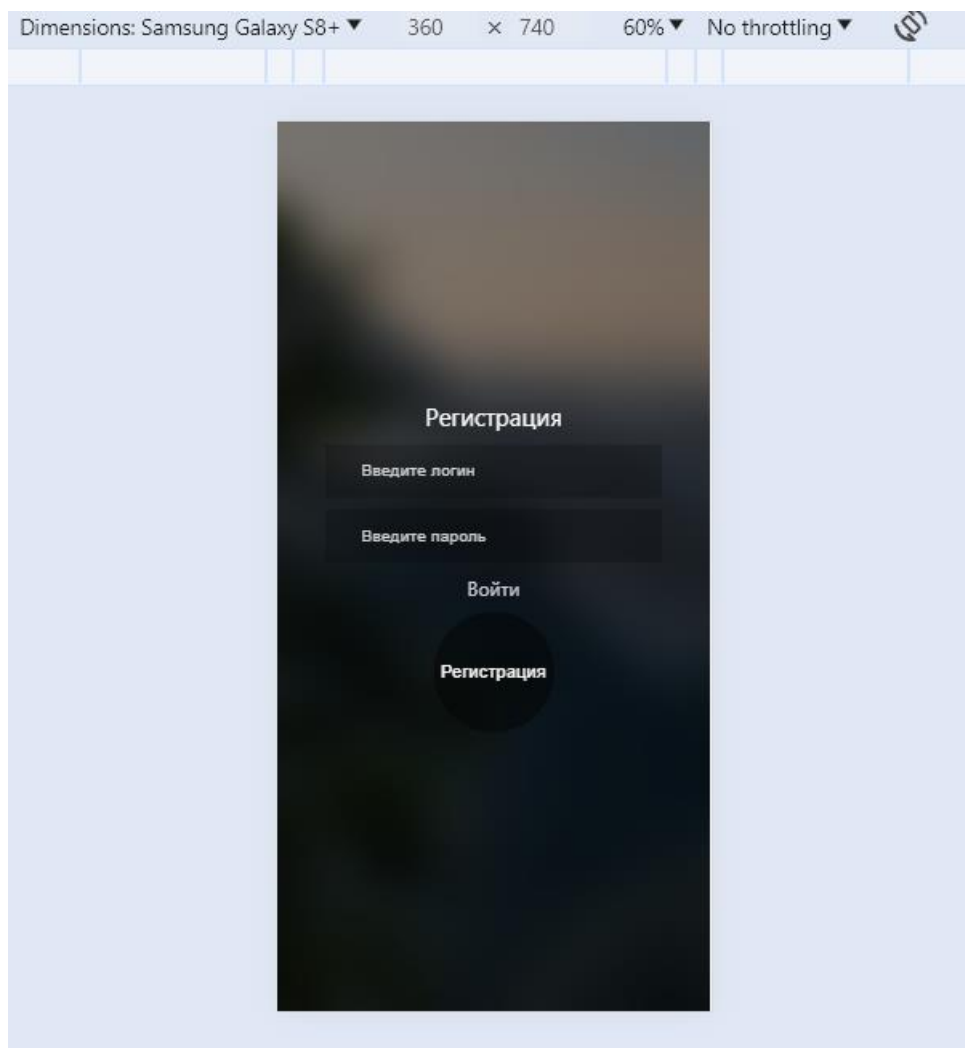


Рисунок Г.17 – Мобильная адаптация регистрации

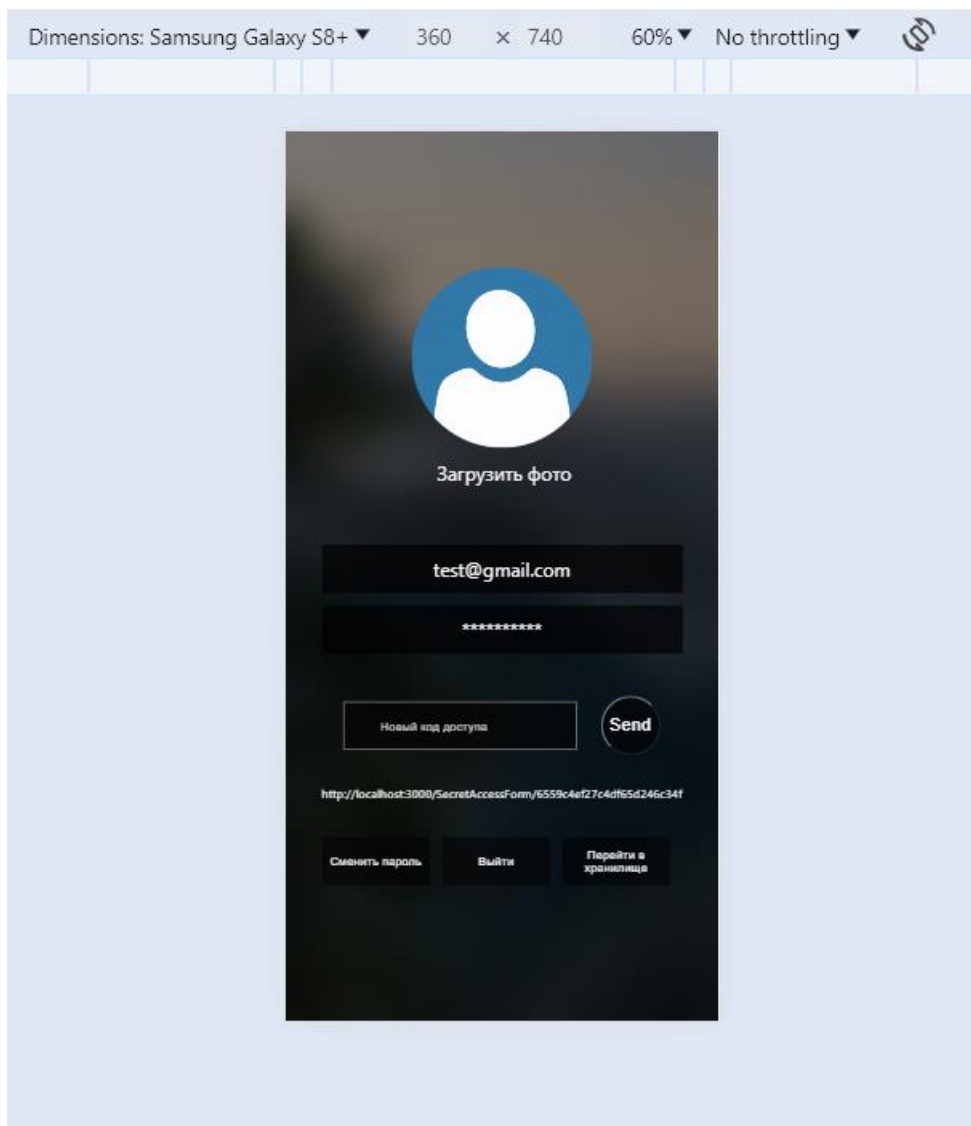


Рисунок Г.18 – Мобильная адаптация профиля

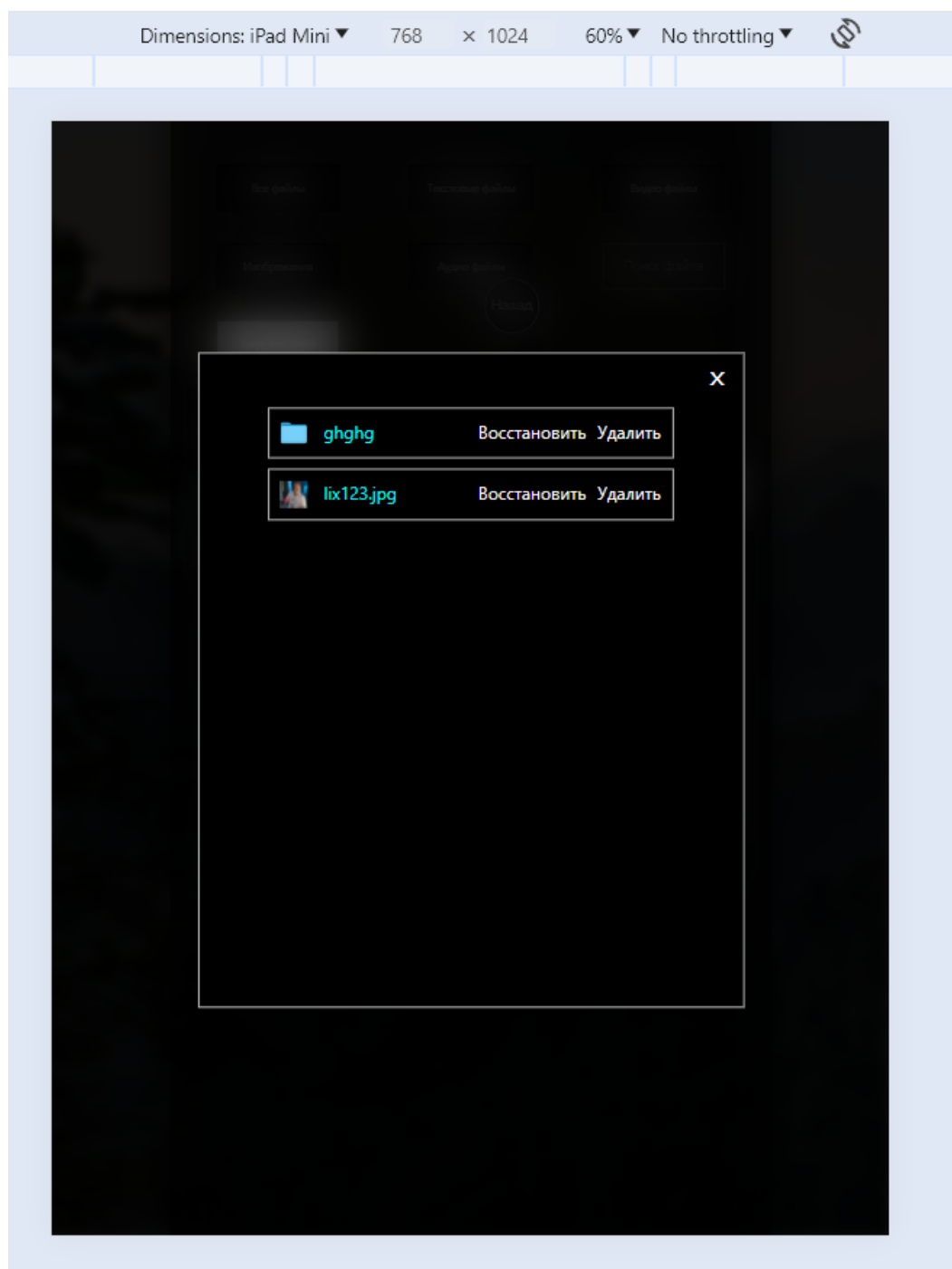


Рисунок Г.19 – Мобильная адаптация модального окна хранилища

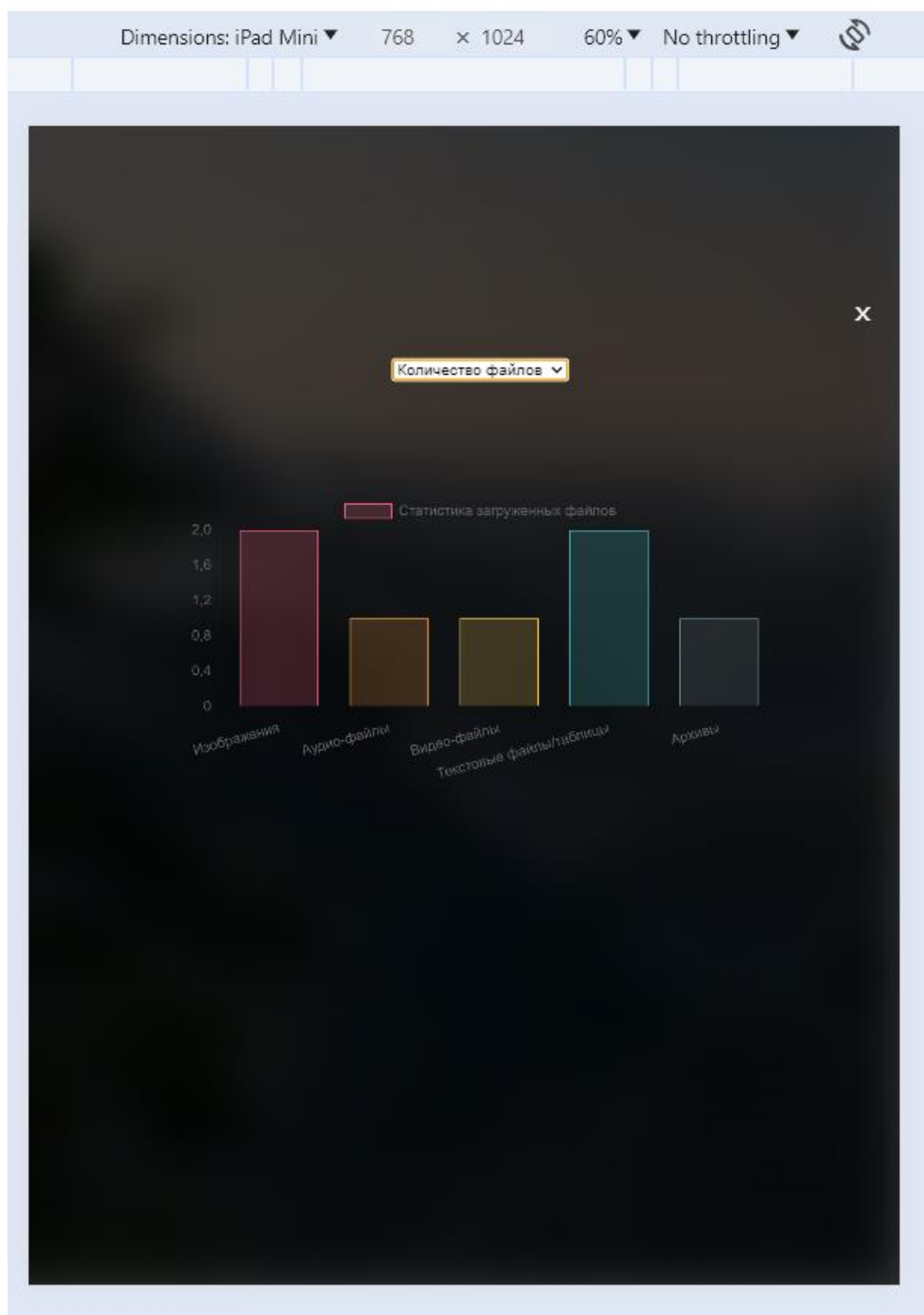


Рисунок Г.20 – Мобильная адаптация графиков

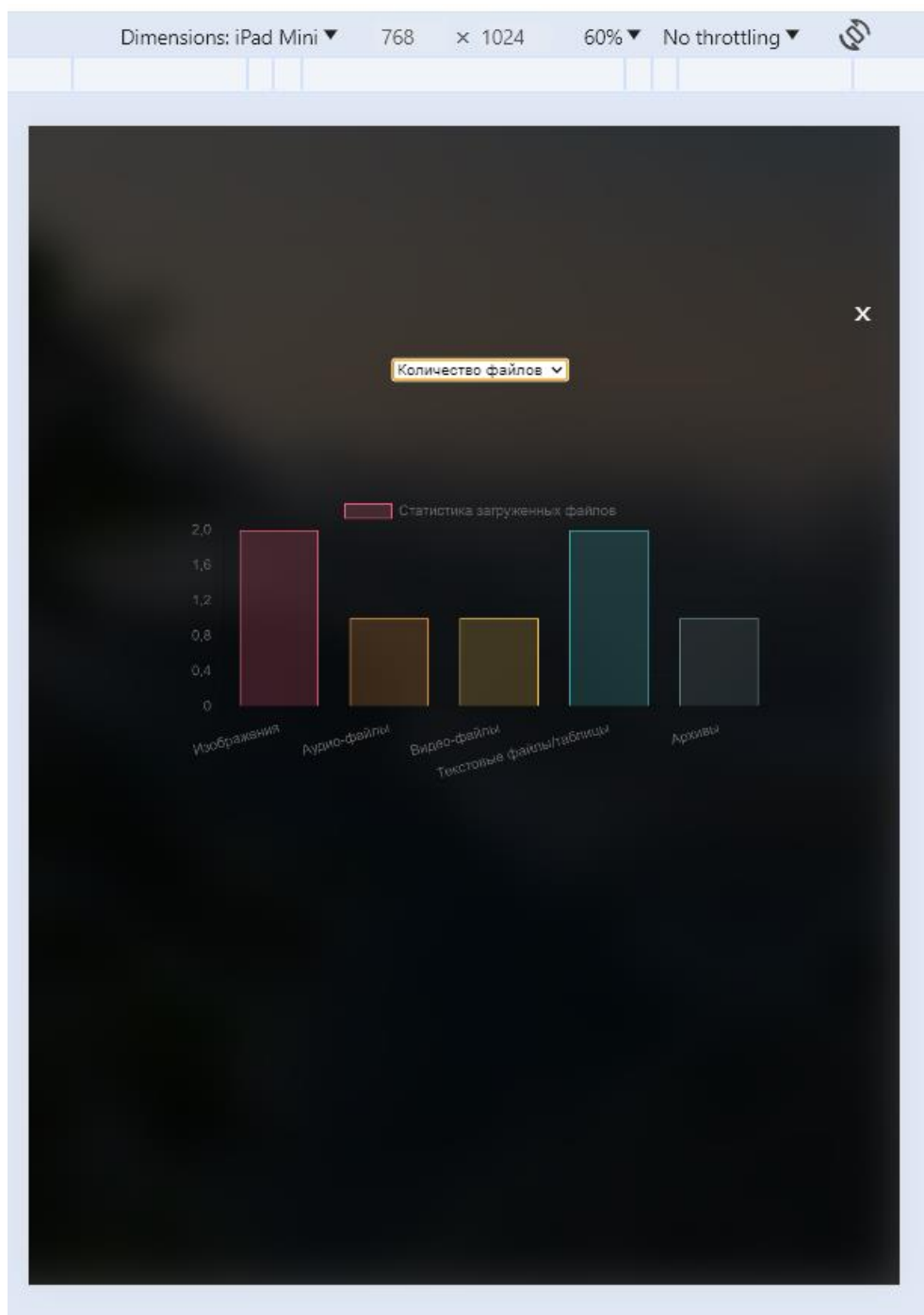


Рисунок Г.21 – Мобильная адаптация графиков