

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ОБЛАЧНЫЙ СЕРВИС ХРАНЕНИЯ

ФАЙЛОВ

Студент Тюгаев 18.06.24 Тюгаев Камиль Алиевич
(подпись, дата) (ФИО полностью)

Руководитель Токарев 18.06.2024 Токарев А. Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков 18.06.2024 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 27.05.2024 г. №11)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от
20.06.2024 г. № 6)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«19» 04 2024г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Тюгаев Камиль Алиевич гр. 20BVB1 факультета вычислительной техники.
2. Тема работы «Облачный сервис хранения файлов»
Тема утверждена приказом ПГУ № 405/0 от "19" 04 2024 г.
3. Руководитель бакалаврской работы к.т.н., доцент Токарев А. Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать приложение «Облачный сервис хранения файлов», выполняющее следующие функции:
 - мультипользовательский доступ к приложению;
 - возможность загрузки, удаления и переименования папок и файлов;
 - возможность поиска файлов и папок по их названию.Данное приложение должно быть спроектировано на основе модели клиент-серверной архитектуры с использованием паттернов *MVC* и *REST*. Клиентская часть должна обладать визуальным графическим интерфейсом, а серверная должна обрабатывать запросы и работать с базой данных.
5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области: изучить аналоги существующих облачных сервисов хранения файлов, изучить способы проектирования и разработки клиент-серверного приложения с применением *REST API* методологии. Необходимо провести проектирование и разработку приложения с использованием фреймворка *Spring* и языка программирования *Java*.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2024-30.04.2024	
2.	Анализ требований к проектированию и разработки клиент – серверных приложений	10	31.04.2024-02.05.2024	
3.	Выбор технологий, применяемых при проектировании и разработке клиент – серверных приложений	50	03.05.2024-07.05.2024	
4.	Описание структур, алгоритмов проектирования и реализации облачного хранилища файлов на базе клиент – серверного приложения	40	08.05.2024-11.05.2024	
5.	Разработка приложения «Облачный сервис хранения файлов»	130	12.05.2024-27.05.2024	
6.	Тестирование разработанного ПО	20	27.05.2024-29.05.2024	
7.	Оформление пояснительной записки	24	30.05.2024-31.05.2024	

Дата выдачи задания "26" апреля 2024 г.

Руководитель бакалаврской работы Токарев А.Н.
(подпись, дата)

Задание к исполнению принял студент Тюгаев К.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 19.06.2024 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 112 листов, 29 рисунков, 10 источников, 2 приложения
ХРАНЕНИЕ ФАЙЛОВ, BOOTSTRAP, MVC, REACT, HIBERNATE,
POSTGESQL, JAVA, SPRING

Объект разработки – облачный сервис хранения файлов.

Целью дипломной работы является разработка приложения для
возможности сохранения файлов в удаленном облачном хранилище.

В процессе работы над программным обеспечением был проведен анализ
требований к разрабатываемому продукту, анализ существующих
современных средств проектирования ПО, анализ аналогичных систем,
изучены методологии написания сервисов для хранения файлов.

В результате проведенной работы создано и протестировано клиент-
серверное приложение, позволяющее пользователю осуществлять
регистрацию и аутентификацию, загружать папки и файлы, переименовывать
их, удалять, а также осуществлять поиск по их названиям.

В пояснительной записке представлены: анализ предметной области,
проектирование и реализация приложения, технология разработки и отладки
программы, раздел БЖД.

Графическая часть содержит модели БД и UML-диаграммы приложения.

					ПГУ1.090301.12.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата	Облачный сервис хранения файлов Пояснительная записка	Лит.	Лист	Листов
Разраб.	Тюгаев К. А.			18.06.24				
Провер.	Токарев А. Н.			18.06.24			2	112
Н. Контр.	Бычков А. С.			18.06.24		ФВТ, каф. ВТ гр. 20BBB1		
Утверд.	Митрохин			18.06.24				

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Облачное хранилище данных	7
1.1 Предпроектный анализ.....	7
1.1.1 Анализ предметной области	7
1.1.2 Анализ аналогичных решений облачного хранилища файлов.	9
1.1.3 Анализ стека технологий	11
1.1.4 Выбор СУБД.....	17
1.1.5 Выбор фреймворка для реализации клиентской части приложения... 18	
1.2 Проектирование и разработка.....	20
1.2.1 Реализация серверной части.	21
1.2.2 Разработка БД.....	28
1.2.3 Разработка клиентской части.....	31
1.3 Анализ результатов.....	33
2 Охрана труда.....	38
2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ	39
Заключение	41
Приложение А – UML - диаграммы.....	43
Приложение Б – Листинг программы	48
Приложение В – Презентация.....	96

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

СУБД – система управления базами данных

API – программный интерфейс приложения

CRUD – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete)

MVC – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

MVP – минимально жизнеспособный продукт

ORM – способ доступа к базе данных с помощью объектно – ориентированного языка

REST – архитектурный стиль взаимодействия компонентов распределенного приложения в сети

				ПГУ1.090301.12.001 ПЗ	Лист
					4
Изм.	Лист	№ докум	Подпись Дата		

Введение

На сегодняшний день современные технологии всё больше и больше внедряются в повседневную жизнь людей. Цифровизация с каждым днём охватывает всё больше различных сфер деятельности человека, что приводит к появлению большого количества данных, которые нужно хранить, обрабатывать, группировать и сортировать. Можно заметить, что работа с информацией является задачей не только больших компаний и корпораций, но уже и повседневной персональной деятельностью каждого человека: документы, музыка, фотографии – со всем этим человек сегодня взаимодействует чаще всего в цифровом варианте.

Способы хранения персональной информации прошли большой путь: от дискотек и цифровых дисков с крайне ограниченным объемом памяти до современных SSD накопителей с огромной скоростью чтения и записи и при этом крайне большим объёмом, порой более одного терабайта. Однако, помимо этого всё большую популярность набирают удалённые способы хранения информации – облачные хранилища.

Развитие данных сервисов позволяет людям экономить память на собственном компьютере, используя больше мест под различное программное обеспечение, а не простые файлы с данными, такими как музыка или word документы, а также иметь доступ к своим файлам из любой точки мира с любого устройства, имеющего выход в интернет, редактировать их в любой удобный момент, а также крайне легко и просто делиться ими с другими пользователями. Благодаря всем этим преимуществам можно с уверенностью сказать, что популярность данных сервисов в будущем будет расти.

Цель данной работы – разработка отечественной системы облачного хранилища файлов в рамках импортозамещения.

				ПГУ1.090301.12.001 ПЗ	Лист
					5
Изм.	Лист	№ докум	Подпись Дата		

Архитектурно данное приложение должно быть реализовано по классическом принципу клиент – серверное приложение, где клиентская часть представлена в виде web – интерфейса в браузере (фронтенд), а серверная часть непрерывно “слушает” запросы с клиентской и производит необходимые манипуляции с записью в базы данных (бэкенд).

Архитектурно приложение будет реализовано по принципу “клиент – сервер”: клиентом выступает web – интерфейс в браузере пользователя, а сервер будет реагировать на присылаемые от пользователя запросы и отвечать за бизнес – логику и корректное хранение данных.

Для реализации выбранного проекта были проведены определенные этапы разработки, по итогам которой демонстрировались результаты работы, а именно:

- изучена предметная область;
- изучены аналоги данного приложения;
- выбран стек технологии для реализации приложения;
- реализовано приложение в соответствии с техническим заданием;
- проведена отладка и тестирование приложения и соответствующих инструментов, на которых она разрабатывалось;
- описан раздел безопасности жизнедеятельности.

				ПГУ1.090301.12.001 ПЗ	Лист
					6
Изм.	Лист	№ докум	Подпись Дата		

1 Облачное хранилище данных

1.1 Предпроектный анализ

1.1.1 Анализ предметной области

Для того, что разработать качественное ПО, которое будет не только реализовывать необходимый функционал, но также и быть удобным и понятным для конечного пользователя, перед началом работы необходимо провести тщательный анализ всех аспектов его разработки – какие существуют языки для реализации ПО подобного типа, как с ним будет взаимодействовать конечный пользователь, какие существуют архитектурные решения для хранения файлов в базе и т. д. Данная глава будет рассматривать необходимые теоретические моменты для реализации облачной системы хранилища файлов.

Облачное хранилище данных – это структура распределенных в сети онлайн – серверов, представляющих собой службу для хранения, группировки и изменения данных путём передачи их по Интернету или другой сети в систему хранения, обслуживаемую третьей стороной. Данная третья сторона, которую в будущем в дальнейшем называть “Поставщик”, обеспечивает безопасное хранение и обслуживание серверов, а также окружающей их инфраструктуры и сети, по которой они обмениваются данными, за счёт чего гарантируется бесперебойный доступ к данным и их безопасность. Поставщик предоставляет практически безграничный объём места для хранения данных, что даёт гибкость, масштабируемость и надежность доступа к ним в любое время и в любом месте.

В данной модели хранения данных “Облаком” называется, как раз та группа серверов, упомянутая выше, которая и обеспечивает хранение данных. Суть названия состоит в том, что, во-первых, данные сервера физически находятся на удаленном от пользователя расстоянии, и пользователь имеет к ним доступ только по сети интернет, а во-вторых, для конечного пользователя

				ПГУ1.090301.12.001 ПЗ	Лист
					7
Изм.	Лист	№ докум	Подпись Дата		

они выглядят как один единый масштабный виртуальный сервер, откуда и возникают ассоциации с большим “облаком”. Физически же сервера могут находиться на отдалении также и друг от друга.

Огромная популярность облачных серверов на сегодняшний день вызвана рядом преимуществ, которые они предоставляют; упомянуть данные преимущества необходимо также в контексте того, что по сути они и отображают тот функционал, который необходимо реализовать любому современному облачному хранилищу:

- доступ к файлам, хранящимся в облачном хранилище с любого
- устройства, имеющего выход в сеть Интернет;
- совместная работа и редактирование файлов одновременно
- несколькими пользователями;
- плата за обслуживание взимается с пользователя только за
- фактически используемое пользователем место, что позволяет не платить за дополнительные излишние мощности:
- отсутствие у клиента издержек по организации способов для
- хранения данных и обеспечения их надежности;
- использование облачного шлюза.

Облачный шлюз – технология по предоставления пользователю удобного доступа к “облаку” путем создания интерфейса, имитирующего работу с “облаком” как работу с локальным жёстким диском. При достаточной скорости интернета пользователь может не отличить удалённую работу и работу на локальном компьютере.

1.1.2 Анализ аналогичных решений облачного хранилища файлов.

Т.к. принцип хранения файлов не локально, а в “облаке” стал сегодня достаточно распространён, то существует уже несколько решений, как импортных, так и отечественных, реализующих данную технологию. Рассмотрим несколько из них.

Самым популярным на сегодняшний день облачным хранилищем файлов является Google Drive. Согласно различным источникам, на сегодняшний день у данного сервиса около 2 миллиардов активных пользователей. На рисунке 1.1 продемонстрирован клиентский интерфейс данного web – сервиса.

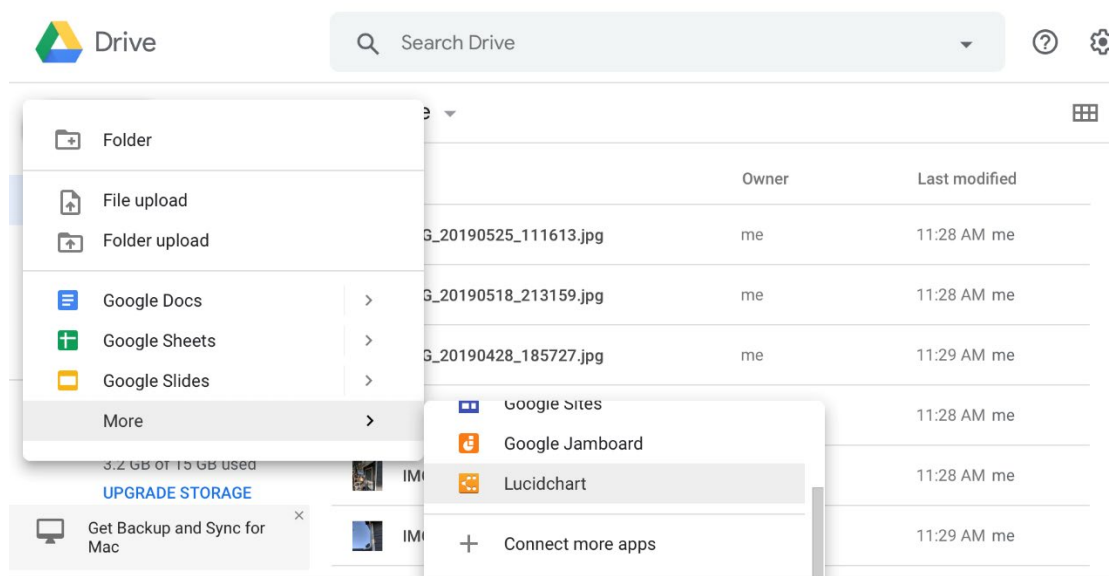


Рисунок 1.1 – Главная страница Google Drive

К основным достоинствам этого приложения можно отнести простую интеграцию с другими решениями от Google, такими как Google Docs, Gmail, Google Photos и т. д., большое количество относительно конкурентов бесплатного предоставляемого места для хранения файлов, а также продвинутые решения по безопасности, такие как двухфакторная аутентификация.

К недостаткам можно отнести то, что изначальное доступное пространство разделено между Google Drive, Gmail и Google Photos, из – за

чего оно может довольно быстро исчерпаться для пользователей, хранящих большое количество фото и эмейлов, а также ограничение на размер загружаемых файлов, что может не соответствовать потребностями некоторых пользователей.

Вторым по популярности и количеству пользователей решением на рынке для удаленного хранения файлов является сервис Microsoft OneDrive. На рисунке 1.2 представлена главная страница данного приложения.

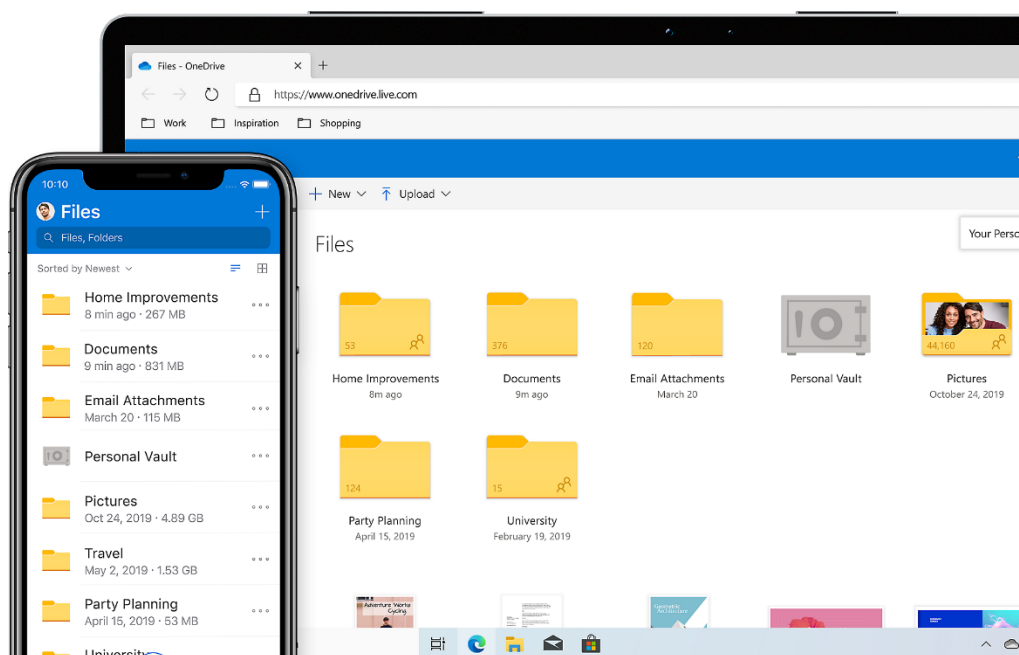


Рисунок 1.2 – Главная страница Microsoft OneDrive на ПК и мобильных устройствах

К его преимуществам относятся возможность онлайн – работы над одним документом несколькими пользователями сразу, возможность настраивать уровень доступа для различных пользователей к “разделяемым” документам, а также “версионность файлов” – OneDrive хранит историю версий файлов, позволяя пользователям возвращаться к предыдущим версиям при необходимости.

К недостаткам можно отнести очень ограниченное по умолчанию свободное место – всего 5 гб, что в три раза меньше места, которое предоставляет Google Drive, практически полное отсутствие интеграции с

любым ПО, кроме ПО Microsoft, что вынуждает пользователей привязываться только к устройствам с операционной системой Windows, а также сложность в использовании сервиса новыми пользователями из – за его крайне обширного и при этом не всегда смежного функционала.

1.1.3 Анализ стека технологий

Так как целью данной работы является создание облачного хранилища файлов, то есть хранилища, к которому необходимо будет обращаться удаленно, то стандартом на сегодняшний день для построения подобного рода приложений является клиент – серверная архитектура. Данная архитектура состоит из трёх основных частей:

- клиентская часть: компонент архитектуры, отвечающий за представление данных для пользователя и отправку запросов на удалённый веб – сервер, чаще всего клиент представлен веб – браузером или мобильным приложением;
- серверная часть: компонент архитектуры, ответственностью которого является обработка запроса, выполнение необходимой бизнес – логики приложения и предоставление ответа на запрос пользователя, может обрабатывать большое количество запросов одновременно;
- база данных: компонент, отвечающие за хранение различного типа данных (системных, пользовательских и т. д.) в приложении в корректном виде, управляется СУБД через команды к базе данных, с помощью которых можно создавать БД, настраивать её схему, изменять данные с помощью языка SQL и т. д.

Мы рассмотрим отдельно выбор необходимой технологии реализации приложения для каждого из описанных компонентов выбранной ранее архитектуры. Начать анализ выбора языка для разработки web – приложения стоит с рассмотрения современных языков программирования, подходящих для реализации серверной части. На сегодняшний день данный раздел разработки предоставляет достаточно большой выбор, однако каждый язык

				ПГУ1.090301.12.001 ПЗ	Лист
					11
Изм.	Лист	№ докум	Подпись Дата		

имеет как свои определённые достоинства, так и недостатки. Критериями выбора языка будут являться его “экосистема” – чем больше будет решений вокруг языка, таких как библиотеки и фреймворки, тем проще будет выбрать необходимое для наших нужд решение и тем проще будет сама разработка, современное состояние языка, т. е. язык должен активно развиваться в текущий момент времени и иметь “вес” в сообществе разработчиков, чтобы можно было использовать различные подходы к разработке (например, удобным образом совмещать функциональное и ООП программирование), а также быстро находить решения возникающих проблем, что довольно легко сделать, если у языка есть активное “коммьюнити”, а также язык должен иметь автоматический сборщик мусора, поскольку у нашего приложения нет крайне высокой потребности в производительности, и не хотелось бы сосредотачивать свои усилия помимо разработки самого приложения также на аспекте управления его памятью. На основе данных критериев и будет осуществлять свой выбор.

Облачное хранилище файлов может быть реализовано на таких современных языках программирования, как Java, Python, C#, Go. Рассмотрим их преимущества и недостатки.

Java – это объектно-ориентированный язык программирования, созданный компанией Sun Microsystems (ныне принадлежащей Oracle) в 1995 году. Основная цель Java — обеспечить платформенезависимость, что достигается за счет использования виртуальной машины Java (JVM), позволяющей запускать программы на любом устройстве с установленной JVM. Java характеризуется строгой типизацией, автоматическим управлением памятью посредством сборки мусора (garbage collection), а также поддержкой многопоточности, что делает его подходящим для разработки масштабируемых и производительных приложений. Благодаря широкому набору библиотек и развитой экосистеме, Java активно используется в

				ПГУ1.090301.12.001 ПЗ	Лист
					12
Изм.	Лист	№ докум	Подпись Дата		

корпоративных приложениях, веб-разработке, мобильных приложениях (особенно для Android) и больших системах данных

Python – это высокоуровневый язык программирования, созданный Гвидо ван Россумом и впервые выпущенный в 1991 году. Он известен своей простотой и читаемостью, что делает его доступным для начинающих программистов и эффективным для разработки сложных проектов. Python поддерживает несколько парадигм программирования, включая объектно-ориентированное, процедурное и функциональное. Особенности Python включают динамическую типизацию, автоматическое управление памятью и обширную стандартную библиотеку, которая ускоряет разработку. Благодаря своей универсальности и мощным библиотекам, Python широко используется в веб-разработке, анализе данных, машинном обучении, научных вычислениях и автоматизации задач.

C# – это современный объектно-ориентированный язык программирования, разработанный Microsoft и выпущенный в 2000 году в составе платформы .NET. C# предназначен для создания разнообразных приложений, от настольных и веб-приложений до облачных и мобильных решений. Он объединяет в себе мощные возможности и удобство использования, поддерживая такие особенности, как строгая типизация, управление памятью через сборку мусора, и развитая система событий и делегатов. C# также поддерживает асинхронное программирование и параллельные вычисления, что делает его подходящим для разработки высокопроизводительных приложений. Благодаря интеграции с .NET, C# предоставляет обширные библиотеки и фреймворки, упрощающие разработку и обеспечивающие высокую производительность, и безопасность кода.

Go – это язык программирования, разработанный Google в 2009 году, отличающийся простотой и эффективностью в написании кода. Одним из главных преимуществ Go является его высокая производительность и быстрая компиляция, что делает его идеальным для создания высоконагруженных

				ПГУ1.090301.12.001 ПЗ	Лист
					13
Изм.	Лист	№ докум	Подпись Дата		

серверных приложений. Также язык обладает встроенной поддержкой параллелизма, что облегчает разработку многопоточных программ. Однако Go имеет и недостатки, например, ограниченную библиотеку стандартных функций по сравнению с другими языками и отсутствие обобщённого программирования (generics) до недавнего времени. В целом, Go славится своей простотой, надёжностью и эффективностью в разработке сетевых и многопоточных приложений.

Учитывая описанные возможности каждого из языков, для разработки приложения было решено выбрать язык Java. Он полностью удовлетворяет всем критериям, что позволит успешно написать и протестировать на нём серверную часть приложения.

Также немаловажным достоинством этого языка является наличие в его экосистеме фреймворка Spring, развивающегося уже более 20 лет и предоставляющего огромное количество инструментов для быстрого написания веб – приложений, таких как встроенный веб – сервер, контейнер для настройки и хранения объектов в приложении, возможность обращаться к базе данных через методы самого языка без использования SQL и т. д. Рассмотрим подробнее все преимущества данного фреймворка.

Spring – это комплексный и гибкий фреймворк для разработки серверной части веб – приложений на языке Java. Он предоставляет широкий ряд функциональных возможностей, которые облегчают задачу написания надёжного, расширяемого и тестируемого кода. Главной концепцией Spring являются принципы внедрения зависимостей и аспектно – ориентированного программирования, которые помогают разработчику уменьшить связность модулей в приложении, и написать простой для тестирования код. Рассмотрим ключевые части данного фреймворка.

Spring Core – главная часть фреймворка Spring. Состоит из таких важнейших частей компонентов как “ядро” – важнейший модуль фреймворка, включающий в себя принцип внедрения зависимостей и инверсии управления,

				ПГУ1.090301.12.001 ПЗ	Лист
					14
Изм.	Лист	№ докум	Подпись Дата		

благодаря которым зависимости в классы внедряются самим фреймворком, а не прописываются разработчиком, “бины” – обычные Java объекты, но под управлением фреймворка Spring, а также контекст – способ доступа к объектам во фреймворке. Всё вышеописанное позволяет писать гибкий код, легко “внедряемые” в объекты зависимости.

Spring AOP – компонент Spring, который предоставляет реализацию аспектно – ориентированного программирования, что позволяет разработчикам определять “перехватчики” для методов и “ключевые точки”, через которые можно отделить “сквозные задачи” (такие как логгирование, настройку транзакций для взаимодействия с БД и т. д.) от объектов, на которые они влияют. Чаще всего данные действия можно легко добавить с помощью аннотаций Spring.

Spring Data Access – часть Spring, упрощающая доступ к базе данных и предоставляющая инструменты для управления транзакциями. Она состоит из JDBC (инкапсулирует в себе код по управлению доступом к базе данных, настройке соединений и т. д.), ORM (инструмент для отображения Java – объектов на объекты таблиц в базе данных), JMS (модуль включает в себя функцию для создания и использования сообщений).

Spring Web – модуль, спроектированный для создания надёжных web – приложений. Основными его составляющими являются: web (часть, которая содержит в себе основные функции интеграции, нацеленные на web, такие как загрузка составных файлов и инициализация контейнера “бинов” с помощью прослушивателя сервлетов), web – servlet (содержит в себе реализацию паттерна MVC в рамках Spring).

Spring Security - мощная и легко настраиваемая платформа аутентификации и контроля доступа. Она предоставляет комплексные услуги безопасности для корпоративных программных приложений на базе Spring фреймворка.

				ПГУ1.090301.12.001 ПЗ	Лист
					15
Изм.	Лист	№ докум	Подпись Дата		

Это важнейшие части фреймворка, но, разумеется, за более чем двадцатилетнее развитие появились также и иные, не менее важные модули, но они в данной работе уже не будут так подробно рассматриваться.

Для того, чтобы убедиться в надёжности приложения и найти возможные баги в его бизнес – логике приложение должно быть “покрыто” тестами. В рамках данной работы наиболее оптимальным видом тестирования представляется интеграционное тестирование.

Интеграционное тестирование – это уровень тестирования ПО, на котором отдельные модули приложения объединяются и тестируются как группа для выявления проблем, связанных с их взаимодействием. Этот тип тестирования направлен на выявление таких проблем, как неправильная передача данных между модулями, несоответствие интерфейсов и общие проблемы интеграции, которые могут быть незаметны при модульном тестировании.

Для приложений Java Spring интеграционное тестирование включает в себя тестирование приложения в целом или его более крупных частей, чтобы убедиться, что различные уровни и компоненты (например, контроллеры, службы, репозитории) работают вместе должным образом. На рисунке 1.3 представлена схема, в которой видно, какое место занимает интеграционное тестирование в общей иерархии тестирования приложений.



Рисунок 1.3 – Иерархия тестирования web – приложений

1.1.4 Выбор СУБД

Для проектирования приложения также необходимо выбрать систему управления базами данных. Для решения поставленной задачи необходимо выбрать современную, гибкую, стабильную и надежную СУБД. Рассмотрим ряд наиболее популярных из них.

PostgreSQL – это мощная объектно - реляционная система управления базами данных (СУБД) с открытым исходным кодом. Она была разработана в Калифорнийском университете в Беркли и впервые выпущена в 1989 году. PostgreSQL поддерживает стандарт SQL и предоставляет расширенные возможности для работы с данными, включая поддержку транзакций с ACID - свойствами (атомарность, согласованность, изолированность, долговечность), сложные запросы, внешние ключи, триггеры, представления, и хранимые процедуры. СУБД также поддерживает расширяемость, что позволяет пользователям добавлять новые функции, такие как типы данных, операторы и функции, без изменения исходного кода.

Oracle – это широко используемая коммерческая система управления базами данных (СУБД), разработанная компанией Oracle Corporation. Выпущенная впервые в 1979 году, Oracle Database зарекомендовала себя как надежное и высокопроизводительное решение для управления данными. Она поддерживает как реляционную, так и объектно-реляционную модели данных, обеспечивая богатый набор функций для работы с данными, включая транзакции с ACID - свойствами, сложные запросы, индексацию, хранимые процедуры и триггеры. Oracle Database широко применяется в крупных предприятиях и критически важных приложениях благодаря своим возможностям масштабируемости и высокой производительности.

MySQL – это популярная СУБД с открытым исходным кодом, первоначально разработанная компанией MySQL AB и теперь поддерживаемая Oracle Corporation. Выпущенная в 1995 году, MySQL стала основой многих веб-приложений и используется в таких платформах, как

				ПГУ1.090301.12.001 ПЗ	Лист
					17
Изм.	Лист	№ докум	Подпись Дата		

WordPress, Joomla и Drupal. Она поддерживает стандарт SQL и предлагает такие функции, как транзакции, индексация, и репликация. MySQL известна своей производительностью, простотой использования и эффективной обработкой больших объемов данных.

В качестве основной СУБД была выбрана Postgres, данный выбор был сделана по нескольким причинам.

Во-первых, PostgreSQL надежна и производительна, что очень важно для работы с большими объемами данных и поддержания стабильности системы. Она гарантирует, что все операции с данными будут выполнены правильно и полностью, даже при высокой нагрузке и одновременном доступе множества пользователей.

Во-вторых, PostgreSQL легко масштабируется, что позволяет системе расти вместе с увеличением количества данных и пользователей. Она поддерживает сложные запросы и расширяется под любые потребности проекта благодаря возможностям добавления новых типов данных и функций. Также у PostgreSQL есть активное сообщество, которое регулярно выпускает обновления, обеспечивающие высокую безопасность и актуальность системы. Все эти качества делают PostgreSQL отличным выбором для облачного хранилища файлов, требующего надежности, гибкости и высокой производительности.

1.1.5 Выбор фреймворка для реализации клиентской части приложения.

Клиентская часть нашего приложения будет представлена web – клиентом, этого будет достаточно, для того чтобы иметь доступ к облачному хранилищу файлов с любого устройства, имеющего доступ к сети Интернет, а без доступа к данной сети пользование им в целом не представляется возможным, поэтому web – клиент должен полностью удовлетворить все пользовательские задачи. Под web – клиентом понимается некий web – обозреватель, или, проще говоря, браузер, такой, например, как Google Chrome или Safari, который есть сегодня на любом мобильном устройстве.

				ПГУ1.090301.12.001 ПЗ	Лист
					18
Изм.	Лист	№ докум	Подпись Дата		

Чаще всего, использование браузера полностью предопределяет тот стек технологий, который будет использоваться для разработки клиентской части. Конечно, существуют различные “экзотические” решения, такие, например, как Blazor (фреймворк, разработанный компанией Microsoft для написания интерактивных сценариев на страницах сети Интернет), но чаще всего они только усложняют разработку проекта, так как требуют для написания качественных решений глубокого погружения, при этом не предоставляя никаких явных преимуществ. Основывая на этом, клиентская часть приложения будет написан на стеке HTML, CSS и Javascript. Кратко рассмотрим данные технологии.

HTML – гипертекстовый язык разметки, являющийся стандартным решением для создания и структурирования контента на web – страницах. По сути формирует собой “каркас” сайта, предоставляя для этого такие инструменты, как теги для разметки “шапки”, “подвала”, абзаца и других важнейших элементов. Помимо тегов модифицировать элементы можно с помощью принадлежащих им атрибутов. Браузер обрабатывает данные теги и затем отображает пользователю готовую web – страницу с мультимедиа контентом.

CSS – язык таблицы стилей, описывающий визуальный дизайн элементов сайта. Он позволяет разработчикам управлять внешним видом HTML – элементов, включая цвета, шрифты, интервалы и анимацию. Важнейшим предназначением этого языка является отделение контента (HTML) от дизайна (CSS), что даёт гибкость и удобство сопровождения дизайна страниц сайта.

Javascript – язык программирования высокого уровня, который обеспечивает динамичное и интерактивное поведение на веб – страницах. В отличие от HTML и CSS, которые используются для структурирования и стилизации, JavaScript позволяет разработчикам реализовывать сложные функции, такие как проверка формы, обновление контента без перезагрузки

				ПГУ1.090301.12.001 ПЗ	Лист
					19
Изм.	Лист	№ докум	Подпись Дата		

страницы (AJAX), интерактивные карты и анимация. Это неотъемлемая часть современного Интернета, часто используемая наряду с HTML и CSS и поддерживаемая всеми основными браузерами.

После того, как был выбран язык для написания клиентской части, необходимо также выбрать фреймворк. Он значительно ускорит написание кода, поскольку в нём уже решены много типовые проблемы, встающие перед разработчиком при написании кода. Для языка Javascript на сегодняшний день наиболее популярным решением является React, поскольку он позволяет писать гибкие и кастомизируемые приложения. Данный фреймворк отлично подойдет для написания сервиса хранения файлов.

1.2 Проектирование и разработка

Концептуально приложение будет состоять из двух важнейших частей – клиентская сторона и серверная часть. Реализовано оно будет путем стандартного на сегодняшний день клиент – серверного взаимодействия: клиент отвечает за предоставление пользователю графического интерфейса (путём отрисовки элементов взаимодействия) и, в соответствии с действиями пользователя, отправкой тех или иных запросов на серверную часть, а серверная часть отвечает за обработку данных запросов в соответствии с бизнес – логикой приложения (например, сохранением файлов на жёсткий диск) и предоставлением ответов на полученные запросы по http – протоколу. В приложении будет использоваться архитектурный подход REST, для передачи самого “тела” запросов будет использоваться формат JSON.

Основной функционал состоит из двух важнейших частей: работа с пользователями и работа с файлами и папками. Под работой с пользователями подразумевается функционал по их регистрации, авторизация и “разлогиниванию”. Под работой с файлами и папками подразумевается их загрузка, возможность создания новой пустой папки, а также удаление и переименование.

				ПГУ1.090301.12.001 ПЗ	Лист
					20
Изм.	Лист	№ докум	Подпись Дата		

1.2.1 Реализация серверной части.

Каркасом серверной части приложения выступает компонент фреймворка Spring – SpringBoot. Его особенностью является то, что он значительно упрощает создание веб – приложений, сразу со старта создавая огромное количество необходимых для функционирования приложения “бинов”, таких, например, как бин для взаимодействия с базой данных, и, соответственно, нам не нужно вручную создавать и конфигурировать данные бины. За подключение в проект различных библиотек и зависимостей будет отвечать фреймворк Maven.

Для быстрого и удобного создания приложения, с включёнными со старта зависимостями воспользуемся сервисом Spring Initializr. Это онлайн – платформа, в которой задаются ключевые настройки web – приложения на Spring, такие как выбор языка приложения (помимо Java доступны Kotlin и Groovy), версия Spring Boot, ключевые зависимости, которые необходимы будут со старта в проекте, например, драйвер PostgreSQL, а сервис затем формирует zip – архив, который представляет собой настроенное приложение и на основе которого можно начинать писать свои собственные классы в приложении. На рисунке 1.4 представлен web – интерфейс данного приложения.

				ПГУ1.090301.12.001 ПЗ	Лист
					21
Изм.	Лист	№ докум	Подпись Дата		

Рисунок 1.4 – Сервис spring initializr для быстрого создания и инициализации Spring – приложений

После того, как каркас приложения был сформирован, следующим шагом является создание пакетов в приложении, каждый из которых представляют собой отдельный слой приложения или же группирует определенный набор связанных классов. Ключевые пакеты в нашем приложении: controllers, dto, exceptions, models, repositories, services. Рассмотрим подробнее каждый из данных пакетов и какую роль в приложении играют сгруппированные в них классы.

Пакет controllers включает в себя контроллеры – классы, обрабатывающие http – запросы, посылаемые клиентской частью приложения на серверную для обработки данных. Представлен четырьмя сходными по смыслу классами: AuthenticationController – принимает запросы, связанные с регистрацией и аутентификацией пользователя в приложении; FileController – обрабатывает запросы, связанные с работой с файлами, такие как их перечисление, загрузка, удаление и т.д.; FolderController – класс, отвечающий за работу с папками в облачном хранилище, по смыслу схож с предыдущим классом; SearchController – отвечает за поиск файлов в приложении. Обособленно от них стоит класс GlobalExceptionHandler – он реализует логику обработки исключений в контроллерах, что позволяет централизованно проводить данную операцию, а не в рамках каждого контроллера в

отдельности. На рисунке 1.5 представлена UML – диаграмма описанных классов.

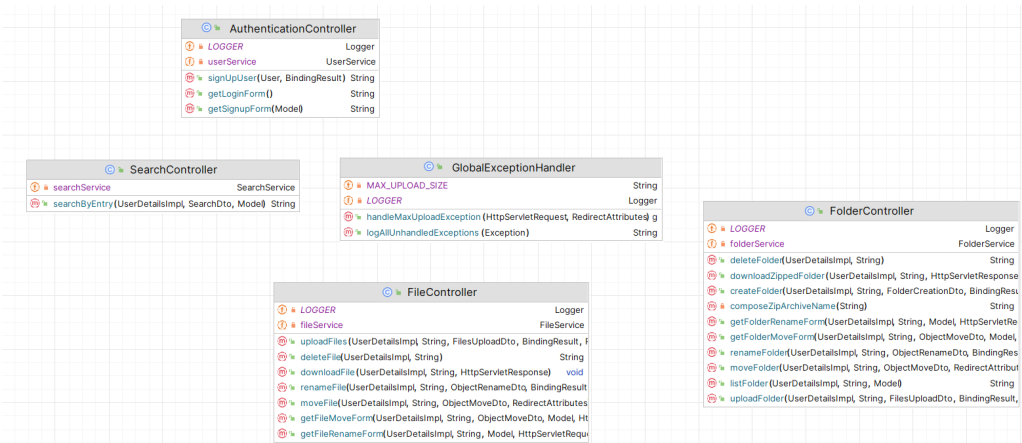


Рисунок 1.5 – UML диаграмма классов – контроллеров

Следующий представленный в серверной части приложения пакет – dto. В данном пакете собраны классы группы DTO – это паттерны под названием “Data Transfer Object”, суть которого состоит в том, чтобы отделить состояние объектов приложения на слое представления от состояния на слое доменной модели. Например, нам может понадобиться у объекта поле id, чтобы мы могли сохранить его в базу данных и затем производить поиск по данному полю, однако данная информация излишняя на слое представления, поэтому мы создаем промежуточный класс – DTO, в котором представлены ключевые поля класса, но отсутствует поле id. На рисунке 1.6 представлена UML – диаграмма классов из пакета dto.

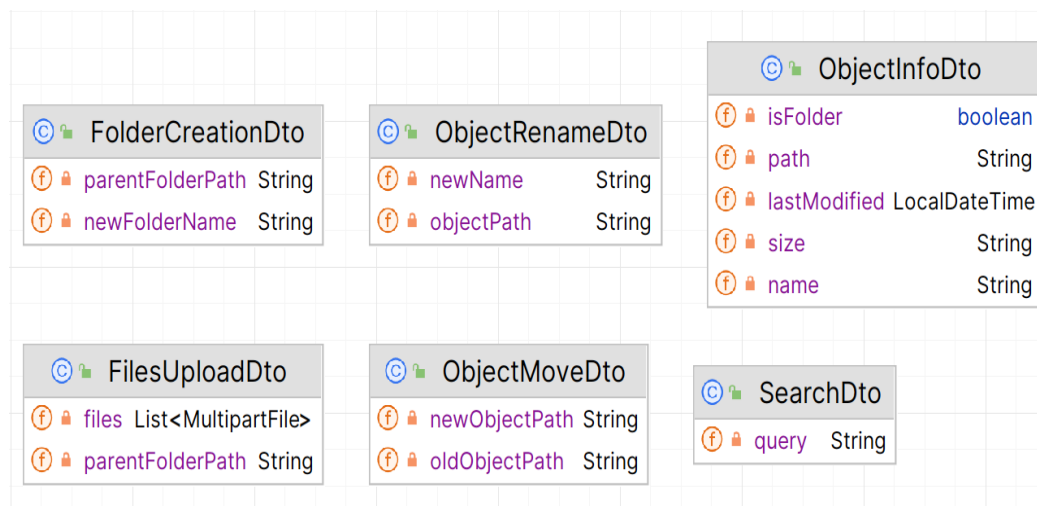


Рисунок 1.6 – UML диаграмма DTO классов

Следующий пакет в приложении – пакет exceptions. Данный пакет хранит в себе классы, отображающие возникшую в приложении исключительную ситуацию – то есть некую ошибку, которую при корректной работе программы быть не должно. В целом, в языке Java есть ключевой класс, отображающий исключения, возникшие во время работы программы – RuntimeException, и можно было бы пользоваться исключительно этим решением, но крайне желательно, чтобы исключение не просто отображало исключительную ситуацию, но из его названия (названия класса исключения) также можно было бы выяснить контекст появления данного исключения, то есть по какой причине оно было выброшено. Для этого разработчики создают свои классы – исключения, наследуясь от класса RuntimeException, и такая же логика реализации исключений представлена и в данном приложении. На рисунке 1.7 на UML – диаграмме отображены классы – исключения, представленные в пакете exceptions.

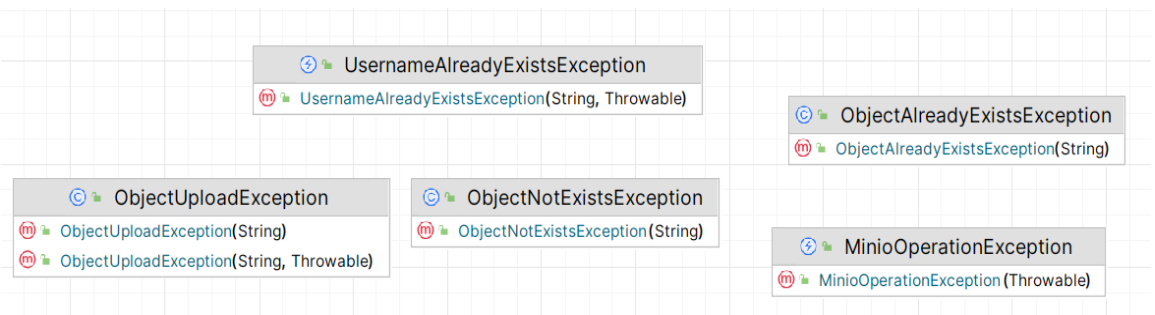


Рисунок 1.7 – UML диаграмма классов исключений

Пакет models хранит в себе классы, представляющие собой доменную область приложения. Данные классы называются моделями, поскольку имеют в себе только поля, без методов, а также отображают две главные сущности приложения – файлы и пользователи. На рисунке 1.8 показана UML – диаграмма данных классов.

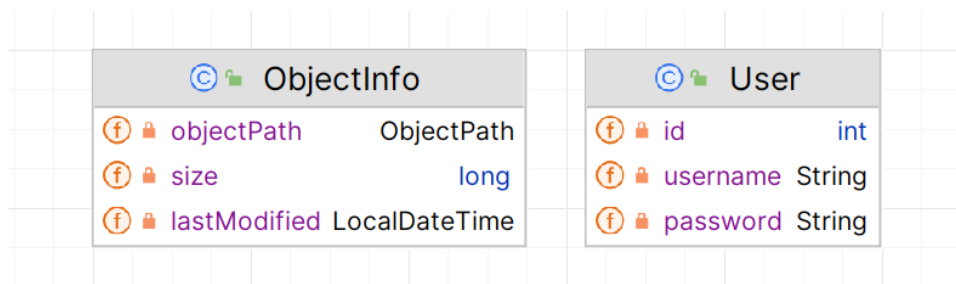


Рисунок 1.8 – UML диаграмма классов моделей

Пакет repositories – важнейший пакет, в нём представлены интерфейсы и их реализация для доступа к используемым в приложении хранилищам (конкретнее, СУБД для хранения информации о пользователях и Minio для хранения файлов). Данные классы формируют конкретный запрос к API хранилищ, записывают в них необходимую информацию, соответствующую бизнес – логике приложения и возвращают её на слой “выше”. Из особенностей реализации данного слоя в приложении можно выделить то, что для доступа к СУБД достаточно унаследоваться от класса JpaRepository, который по умолчанию предоставляет java – методы, которые затем будут конвертированы в sql – запросы, то есть разработчику нужно прописывать только конкретную, “кастомную” логику, относящуюся к его приложению, в

то время как для доступа к хранилищу Minio всю логику необходимо полностью прописывать с нуля, однако в целом на скорости реализации приложения эти особенности не сказывались. На рисунке 1.9 представлены UML – диаграмма классов, входящих в пакет repositories.

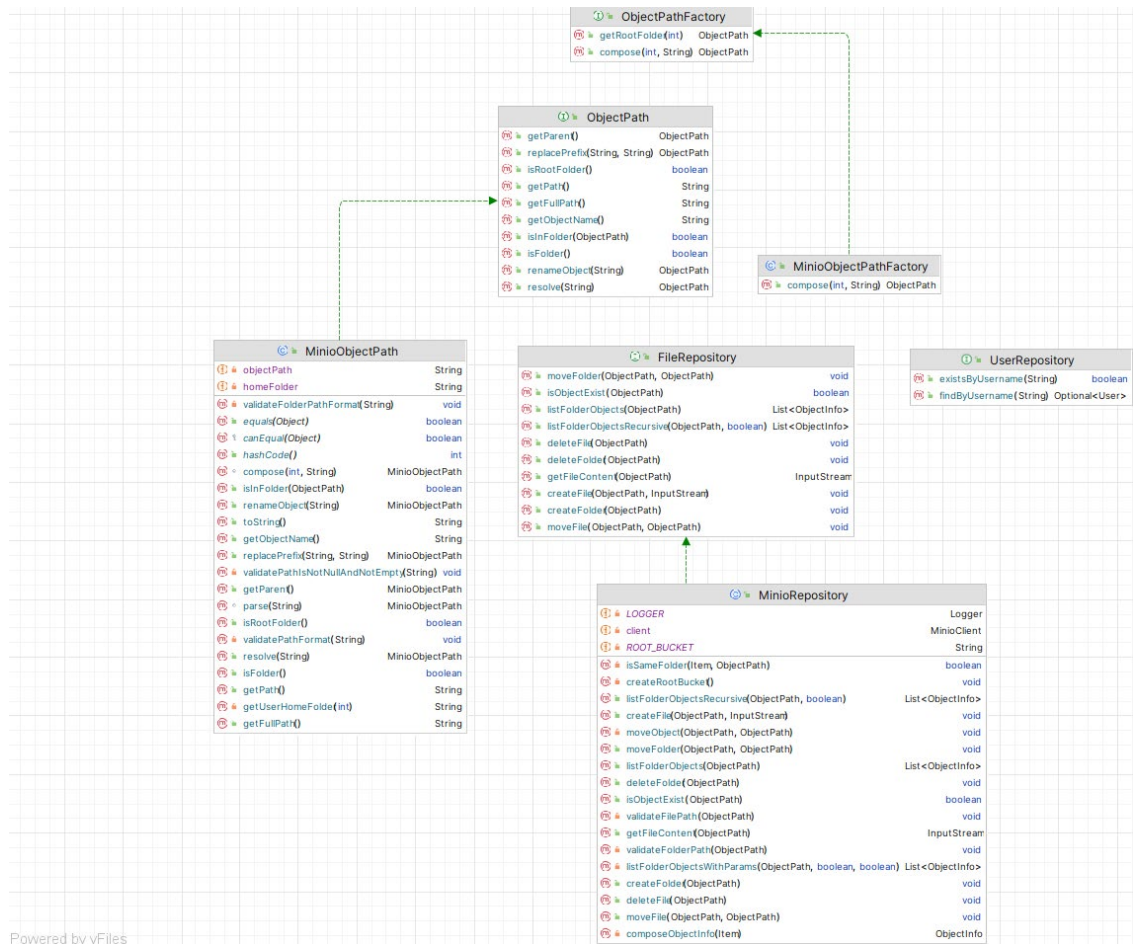


Рисунок 1.9 – UML диаграмма классов – “репозитории”

Пакет services включает в себя классы, представляющие собой бизнес – логику приложения. После того, как на классы группы “контроллер” приходит конкретный запрос, данный запрос “посылается” на уровень бизнес – логики, где чаще всего выполняются различные бизнес – “проверки”, происходит обращение к внешним источникам хранения информации, идет обработка полученного ответа и представление данных в удобном для слоя представления виде. Надо заметить, что описанные операции опциональны и выполняются в зависимости от того, какой конкретно метод будет выполняться. В случае разрабатываемого приложения классы слоя “сервиса”

реализуют логику по работе с файлами, папками, пользователями и поиском файлов и папок. На рисунке 1.10 показана UML – диаграмма данных классов.

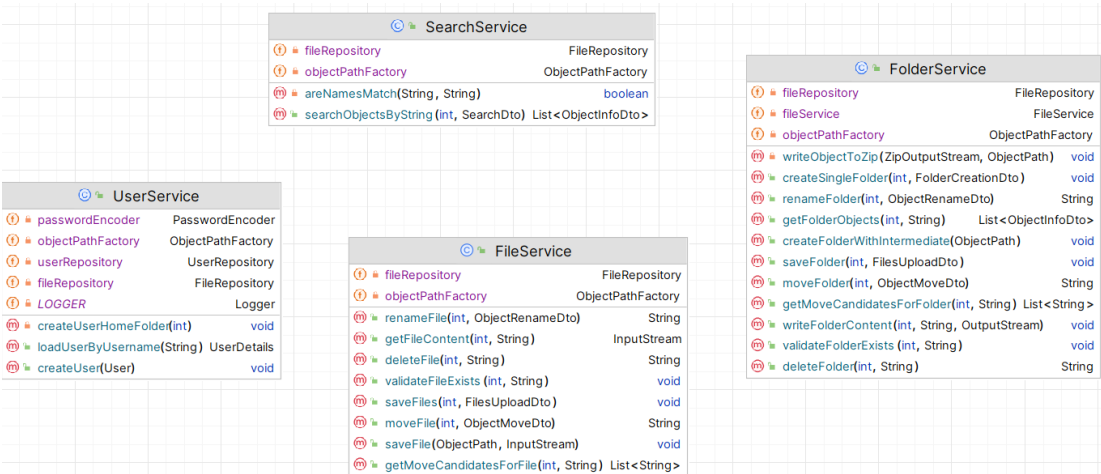


Рисунок 1.10 – UML диаграмма классов – “сервисов”

Пакет utils – объединяет “утилитарные” классы – вспомогательные классы, в которых представлены только статические поля и методы. Данные методы выполняют схожие по смыслу задачи (например, математические операции над числовым типом данных и т. д.). В данном приложении основные операции, выполняемые классами: “перевод” размера файла в читаемый формат, “маппинг” классов – моделей в DTO – классы, переконвертирование путей к файлу и т.д. На рисунке 1.11 показана UML – диаграмма данных классов.

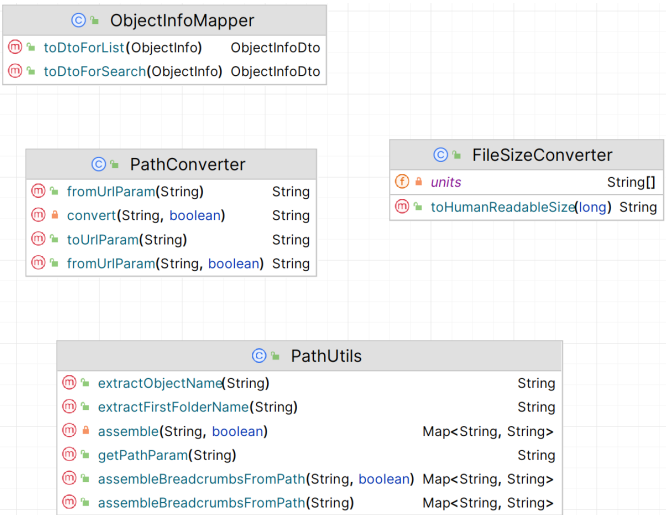


Рисунок 1.11 – UML диаграмма “утилитарных” классов

Последний пакет, представленный в приложении – пакет validators. В нём представлены классы, проверяющие валидность названия файлов, создаваемых пользователем. На рисунке 1.12 представлена UML диаграмма классов в данном пакете.

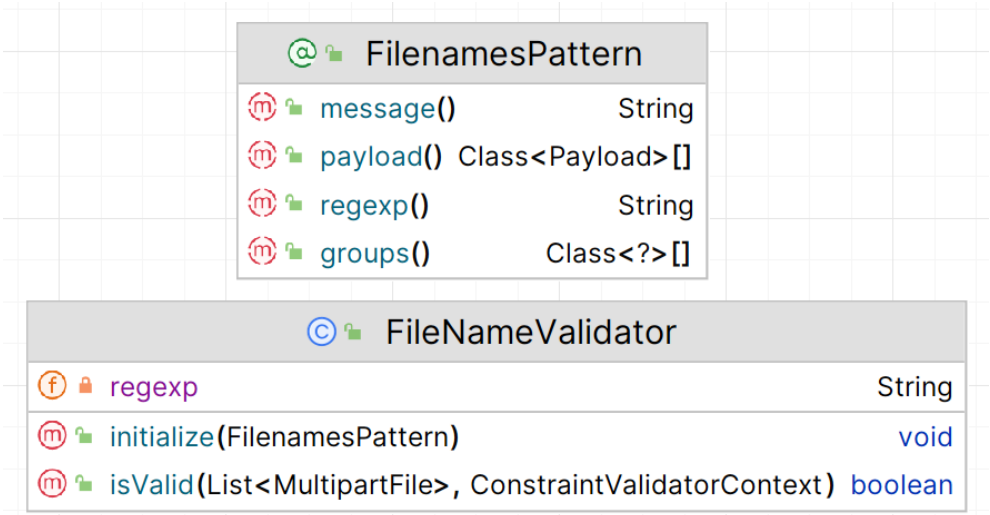


Рисунок 1.12 – UML диаграмма классов пакета validators

1.2.2 Разработка БД

В приложении будет использоваться два типа хранилища для сохранения данных – PostgreSQL и Minio. Рассмотрим подробнее каждый из них.

Предварительную настройку СУБД PostgreSQL, а именно создание самой базы данных и защиту базы данных паролем можно реализовать через графический клиент для ОС Windows pgAdmin 4. На рисунке 1.13 изображён данный графический клиент.

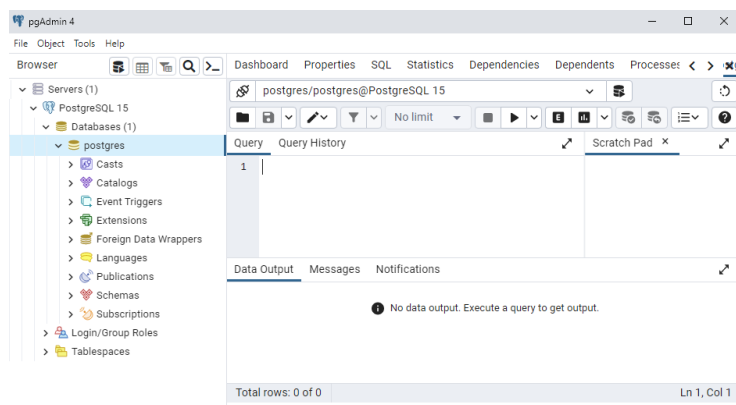


Рисунок 1.13 – Графический клиент pgAdmin4

После того, как база данных настроена, необходимо подключиться к ней через наше Java приложение. Для этого пропишем настройки подключения в файле `application.yml`. На рисунке 1.14 показаны прописанные для соединения настройки. Нам не нужно будет вручную создавать бин соединения с базой данных, SpringBoot сделает это за нас.

```
spring:
  datasource:
    username: postgres
    password: postgres
    url: jdbc:postgresql://localhost:5432/cloud-file-storage
```

Рисунок 1.14 – Настройки для подключения к базе данных

После этого мы можем свободно обращаться к базе данных и сохранять в неё объекты нашего приложения. Наиболее удобно это можно сделать через библиотеку SpringDataJpa. Это интерфейс, входящий в экосистему Spring, который предоставляет API для взаимодействия с СУБД. Использовать его можно создав свой класс, унаследованный от данного интерфейса, а в generic типе интерфейса указать тип, используемый в соответствующей таблице как `id`, и саму сущность, которая будет сохраняться в базу данных.

База данных будет хранить в себе всего одну таблицу под названием `users`. В ней будет храниться информация о пользователе (таблица 1.1).

Таблица 1.1 – Атрибуты сущности “Пользователи”

Наименование атрибута	Тип	Обязательный
Идентификатор пользователя	Целый	Да
Юзернейм пользователя	Строка	Да
Пароль пользователя	Строка	Да

Хранение файлов в приложении будет реализовано через Amazon S3(Simple Storage Service). Это сервис облачного хранилища данных, предоставляемый компаний Amazon Web Services (AWS). Он предназначен для хранения и извлечения любых объемов данных в любое время из любой точки мира. S3 используется для хранения данных, таких как медиафайлы, документы, резервные копии, логи, и других типов данных.

Для того, чтобы воспользоваться данной технологией, необходимо будет загрузить и локально развернуть Minio. Minio предоставляет высокопроизводительное, масштабируемое и безопасное хранилище данных с интерфейсом, совместимым с Amazon S3. К его главным преимуществам относятся высокая производительность, безопасность и простота развертывания и управления. На рисунке 1.15 показано подключение к данному хранилищу в конструкторе класса MinioRepository. После этого в нём можно создать bucket и сохранять в данный bucket пользовательские файлы.

```
public MinioRepository(Environment env) {
    String endpoint = env.getProperty( key: "minio.endpoint", defaultValue: "localhost");
    String accessKey = env.getRequiredProperty( key: "minio.username");
    String secretKey = env.getRequiredProperty( key: "minio.password");

    client = MinioClient.builder()
        .endpoint(endpoint)
        .credentials(accessKey, secretKey)
        .build();
    LOGGER.info("Connected to MinIO server successfully. Endpoint: {}", endpoint);

    createRootBucket();
}
```

Рисунок 1.15 – Подключение к хранилищу Minio

Все файлы будут лежать в одном bucket, а полный путь до файла будет состоять из совокупности родительской папки – имени пользователя, сохранившего файл, и самого названия файла. Пользователь будет иметь доступ только к своей папке.

1.2.3 Разработка клиентской части

Клиентскую часть приложения реализуем на фреймворке React. Генерация исходного кода клиентской части и установка необходимых библиотек производится с помощью `nrpx`. `Nrpx` – инструмент, который используется для выполнения пакетов `node.js` и позволяет легко выполнять необходимые операции для предварительной настройки приложения через консольный терминал. Также импортируем в приложение библиотеку `Bootstrap` для использования готовых стилей. В данный момент клиентская часть приложения уже может быть запущена для проверки работоспособности, однако пока она никак не связана с серверной частью. Запустим код и откроем приложение по адресу `http://localhost:3000`. Результат данных действия показан на рисунке 1.16.

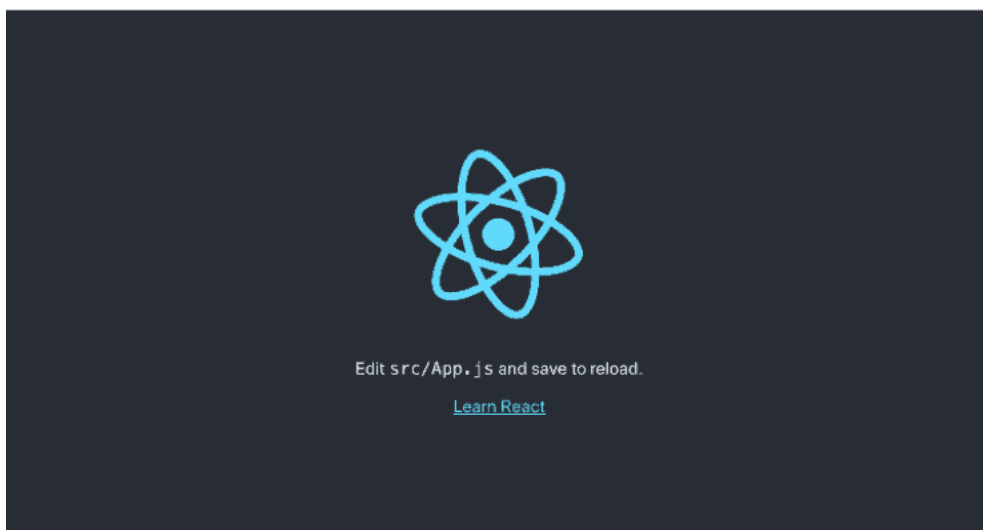


Рисунок 1.16 – Результат запуска клиентской части приложения

Для того, чтобы клиентская часть могла посылать запросы на клиентскую, необходимо явно прописать в `package.json` свойство `"proxy"`: `http://localhost:8080`.

				ПГУ1.090301.12.001 ПЗ	Лист
					31
Изм.	Лист	№ докум	Подпись Дата		

Приложение на react состоит из компонентов, представляющих из себя сочетание html и javascript кода, который затем будет обработан фреймворком и встроен в DOM – дерево сайта. Все компоненты наследуются от главного Component с помощью ключевого слова языка extends. На рисунке 1.17 приведена часть кода компонента login.

Всего в приложении 10 основных компонентов: login, signup, breadcrumbs, navbar, error, main, move, rename, search, app. Компоненты посылают запрос на серверную часть с помощью метода await fetch(), а также отображают клиентский интерфейс с помощью метода render().

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min';

class Login extends Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: '',
      error: false,
    };

    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(event) {
    const { name, value } = event.target;
    this.setState({ [name]: value });
  }
}
```

Рисунок 1.17 – Код компонента Login

После того, как все компоненты будут созданы, с помощью инструмента React Router пропишем в главном компоненте app роутинг, закрепляющий компоненты за конкретным url адресом. Пример кода роутинга представлен на рисунке 1.18.

```
<Route path="/" exact={true} component={Main}/>
```

Рисунок 1.18 – Роутинг клиентской части приложения

1.3 Анализ результатов

Результатом работы выполнения вышеописанных действий является клиент – серверное приложение, облачное хранилище файлов. Данное приложение предоставляет следующий набор возможностей для пользователя:

- регистрация нового пользователя;
- авторизация существующего пользователя;
- загрузка файлов и папок;
- поиск файлов и папок по названию;
- отображение списка файлов в текущей директории;
- переименование и удаление файлов и папок.

На рисунке 1.18 представлена UML – диаграмма вариантов использования приложения пользователем.

Проверим работоспособность приложения путем ручного тестирования.

При запуске приложения, поскольку у нас ещё нет пользовательского аккаунта, нам открывается окно регистрации и авторизации, представленное на рисунке 1.19.

Рисунок 1.19 – Окно регистрации и авторизации в приложении

Зарегистрируем нового пользователя под тестовыми данными admin/admin. После того, как регистрация успешно завершена, авторизуемся в

				ПГУ1.090301.12.001 ПЗ	Лист
					33
Изм.	Лист	№ докум	Подпись Дата		

приложении. Авторизация также прошла успешно, и мы попадаем на главную страницу приложения, что отображено на рисунке 1.20.

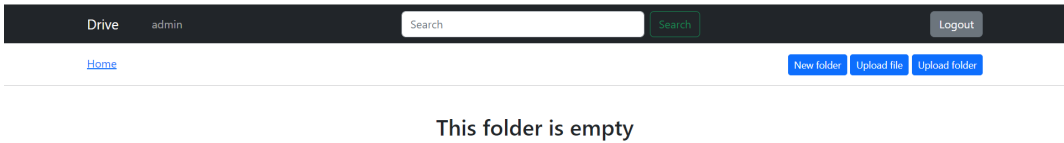


Рисунок 1.20 – Главная страница приложения

Создадим новую папку – для этого нажмем на кнопку New folder и назовем папку folder – 1. На рисунке 1.21 отображена успешно созданная папка.

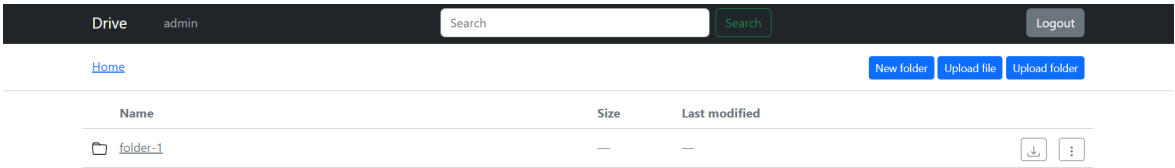


Рисунок 1.21 – Новая папка folder – 1 создана и отображается на главной странице приложения

Проверим возможность загрузки данной папки к себе на компьютер. Для этого нажмем на стрелку загрузки, которая располагается около названия компании. Видим, что папка успешно загрузилась на ПК. Это отображено на рисунке 1.22.

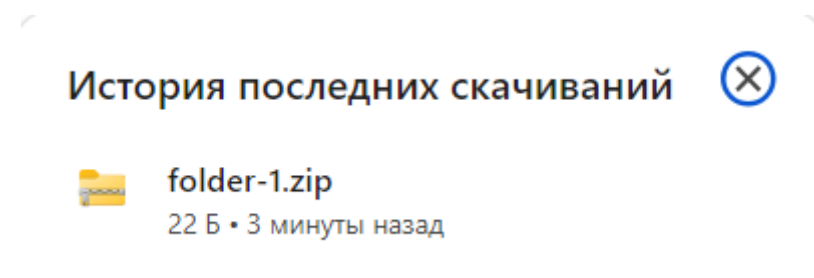


Рисунок 1.22 – Папка успешно загружена на ПК через браузер

Попробуем сохранить в созданную нами папку файл. Для этого нажмем на неё и попадем внутрь папки. После этого нажмем кнопку Upload file и загрузим с компьютера заранее заготовленный тестовый файл. Файл успешно загружен в облачное хранилище, это показано на рисунке 1.23.

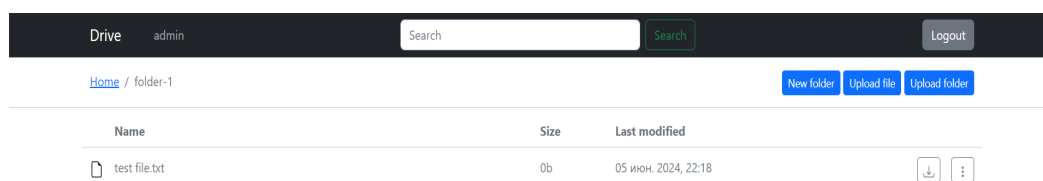


Рисунок 1.23 – Файл успешно загружен в созданную ранее папку

Проверим возможность переименования файла. Для этого нужно нажать на три вертикально расположенные точки около названия файла и выбрать кнопку Rename. Новое название укажем как test file – 2. Переименование произошло успешно, результат можем видеть на рисунке 1.24.

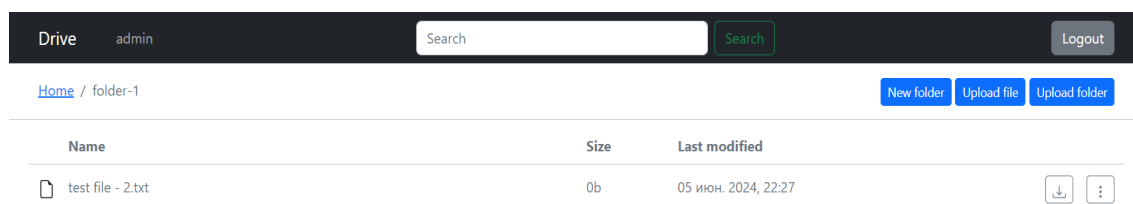


Рисунок 1.24 – Успешные результат переименования файла

Проверим теперь возможность перемещения файла между папками. Для этого внутри папки folder – 1 создадим папку folder – 2. Далее нажмем снова на три точки, расположенные около название файла, и выберем опцию move.

В результате у нас откроется окно “Target location”, в котором можно выбрать любую папку, созданную в приложении, кроме той, в которой хранится сам файл. Выберем её и нажмем кнопку move. Перемещение файла произошло успешно, можем наблюдать это на рисунке 1.25.

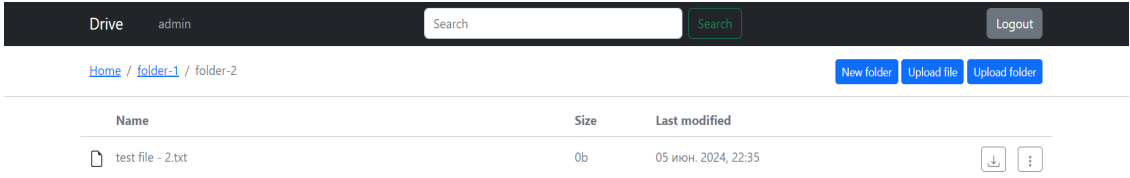


Рисунок 1.25 – Результат перемещения файла в другую папку

Проверим далее возможность поиска файла. Для этого вверху в поисковой строке введем название папки folder – 1. На рисунке 1.26 показано, что поиск папки успешно осуществлён.

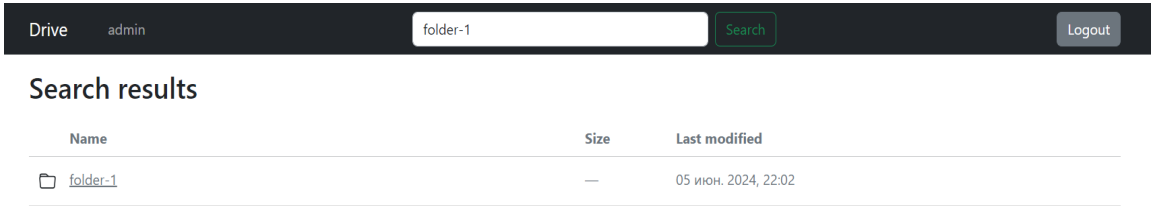


Рисунок 1.26 – Результат поиска в приложении ранее созданной папки

Аналогично успешно найден и файл, это отражено на рисунке 1.27.

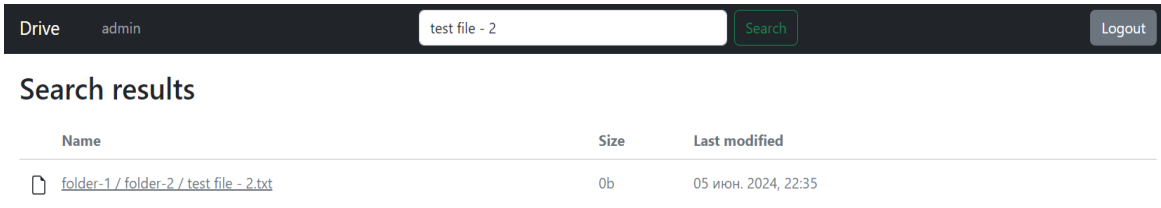


Рисунок 1.27 – Созданный ранее файл успешно найден в приложении

Теперь проверим возможность удаления файлов. Для этого нажмем на три точки у названия созданного файла и далее нажмем на кнопку delete. На рисунке 1.28 показано, что файл успешно удалён.

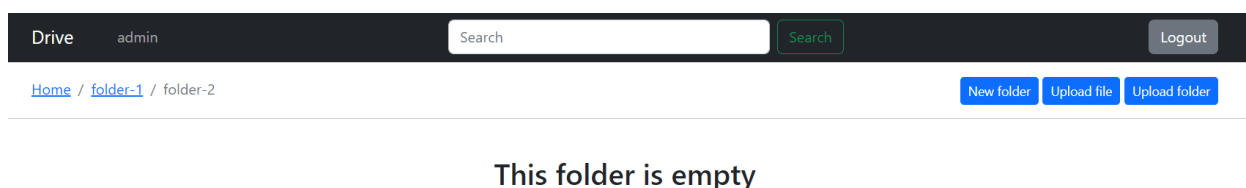


Рисунок 1.28 – Созданный ранее файл успешно удалён из папки folder-2

Осуществим последнюю проверку в приложении – возможность выхода из приложения. Нажмём на кнопку Logout в правом верхнем углу. Результатом будет то, что нас “перекидывает” на страницу авторизации. Это отражено на рисунке 1.29.

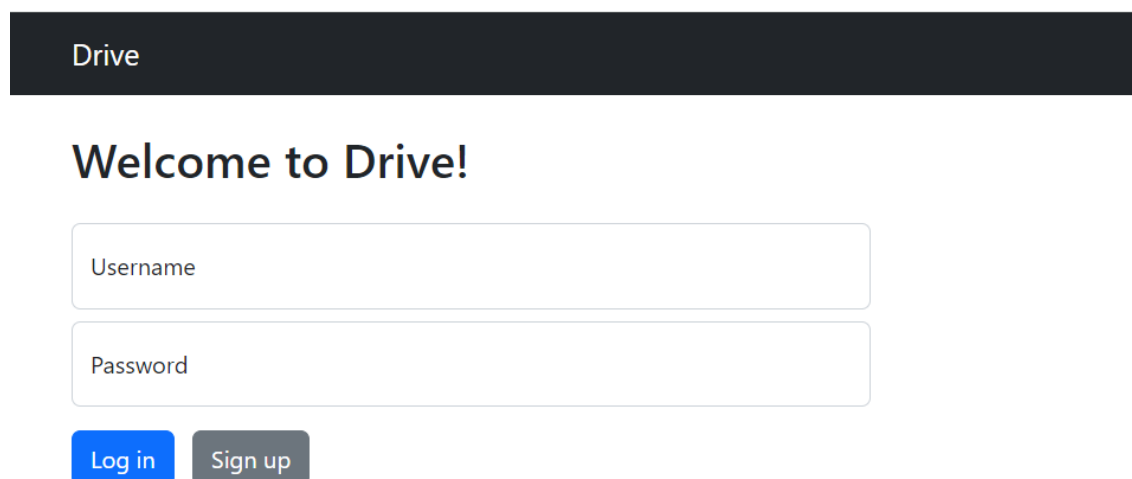


Рисунок 1.29 – Выход из приложения успешно осуществлён

Таким образом, ручное тестирование показало, что приложение успешно функционирует и выполняет весь ранее заявленный функционал. Багов в ходе эксплуатации приложения обнаружено не было

				ПГУ1.090301.12.001 ПЗ	Лист
					37
Изм.	Лист	№ докум	Подпись Дата		

2 Охрана труда

В современном обществе, очевидно, происходит всё большая цифровизация всех сфер в жизни, что влечёт за собой то, что уже не только разработчики проводят практически всё своё рабочее время за ЭВМ, но также и у огромного ряда иных профессий рабочая деятельность становится неразрывно связана с ПК. Это, в свою очередь, поднимает вопрос того, как должна регулироваться охрана труда в этой сфере человеческой деятельности, то есть какие должны устанавливаться правила работы для сохранения физического, психического и иных видов здоровья людей.

С одной стороны, работы за ПК не несёт явной прямой физической опасности для человека, как, например, работа на производстве за специфическим оборудованием, способным нанести физический вред человеку, с другой стороны, такая деятельность практически всегда связана с сидячим образом жизни, вредом для зрения и рядом прочих негативно влияющих на человека факторов, что, разумеется, ведет к негативным последствиям в будущем. Таким образом, важно явным образом сформулировать и описать правила по работе за ЭВМ, которые смогут как можно более значительно снизить описанный ранее вред.

Рассмотрим данные факторы:

– необходимо, что рабочее место разработчика было площадью не менее 5 квадратных метров, для того чтобы в нём можно было разместить рабочее оборудование и не чувствовался дискомфорт от нахождения в нём, также желательно обеспечение как естественным, так и искусственным освещением для комфортной работы в вечернее время суток, и обеспечение помещением вентиляции для возможности его проветривания свежим воздухом;

– при работе за ПК из – за постоянной фокусировки человек редко моргает, что вызывает пересыхание слизистой глаза, поэтому рекомендуется работа в специальных защитных очках, которые за счёт специального

покрытия блокируют излучение сине – фиолетового спектра и блики от монитора;

– во время работы в сидячем режиме крайне рекомендуется подобрать удобный стул для поддержания правильной осанки и сохранения работы спины, периодически делать разминку, а также подобрать правильное положение монитора для того, чтобы не перегружать шейный отдел позвоночника;

– желательно не перегружать нервную систему крайне большой по временным интервалам работой за компьютером, и работать не более 8 – 10 часов в день, после завершения работы рекомендуются пешие прогулки на свежем воздухе.

Соблюдая вышеописанные правила можно значительно снизить негативные эффекты от долгой работы за ЭВМ и сохранить своё здоровье.

2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Для того, чтобы ещё раз подчеркнуть значимость и важность соблюдения вышеописанных правил, опишем более подробно, к каким серьёзным последствиям со здоровьем может привести их несоблюдение.

Возможные негативные последствия:

– проблемы с глазами. Симптомы так называемого “компьютерного зрения” включают в себя сухость глаз, жжение, зуд, усталость, головную боль и в дальнейшем снижение общего уровня зрения обоих глаз человека;

– проблемы с опорно–двигательным аппаратом. У пользователя ПК могут появляться боли в шее и спине, связанные с долгой работой в сидячем режиме в неудобной позе, возможно онемение в запястьях и пальцах из – за долгой работы с клавиатурой и нажатием на мышь, воспаление сухожилий, связанное с повторяющимися движениями кистей и предплечей;

				ПГУ1.090301.12.001 ПЗ	Лист
					39
Изм.	Лист	№ докум	Подпись Дата		

– проблемы с психическим здоровьем. Из –за ощущения постоянного давления и необходимости многозадачности при работе с большим количеством приложений и “мессенджеров” во время выполнения профессиональной деятельности у людей может значительно повышаться уровень стресса и тревожности;

– проблемы с сердечно – сосудистой системой. Низкая физическая активность при долгой работе за ПК увеличивает риск развития сердечно – сосудистых заболеваний, таких как гипертония, ожирения, диабета и других значительных заболеваний.

Таким образом, вышеописанные примеры демонстрируют нам, что охрана труда в любой деятельности, связанной с длительной работой за ПК, крайне важна, так как несоблюдение её условий влечёт крайние серьезные риски для здоровья, а соблюдение значительно данные риски снижает, хотя полностью их убрать на текущий момент, к сожалению, невозможно.

				ПГУ1.090301.12.001 ПЗ	Лист
					40
Изм.	Лист	№ докум	Подпись Дата		

Заключение

В результате написания дипломной работы был произведён анализ выбранной темы, включающий в себя изучение аналогов разрабатываемого продукта, выбор инструментов для разработки и их применение на практике, что позволило достичь финального результата – разработанного и протестированного сервиса для облачного хранилища файлов.

Данное ПО позволяет путём удаленного сохранения файлов обеспечивать надежное хранение информации, работу с данными с большого количества устройств, а также возможность поиска и переименования сохранённых файлов.

В процессе разработки были закреплены навыки работы с языком Java, было изучено многообразие экосистемы фреймворка Spring, изучен и закреплён на практике путём написания приложения архитектурный шаблон MVC, также изучена СУБД PostgreSQL путем работы с ней через её графический клиент pgAdmin4.

Дальнейшее улучшение приложения может включать возможность редактирования файлов в режиме многозадачности онлайн одновременно несколькими пользователями, а также возможность введения платной подписки для закупки большего количества устройств для хранения данных и расширения тем самым доступного места для хранения файлов пользователей.

В ходе написаны выпускной квалификационной работы выбранное приложение было успешно разработано, протестировано, багов в ходе его эксплуатации обнаружено не было. Данный опыт выполнения работы однозначно пригодится в будущем при работе над большими комплексными коммерческими проектами.

				ПГУ1.090301.12.001 ПЗ	Лист
					41
Изм.	Лист	№ докум	Подпись Дата		

Список использованных источников

1. Блох Д. Эффективная Java; Диалектика – Вильямс – М., 2019. – 464 с.
2. Кларенс Х., Харроп Р. Spring 5 для профессионалов; Диалектика – Вильямс – М., 2020. – 1120 с.
3. Берт Б., Сьерра К. Изучаем Java; Эксмо – М., 2022. – 720 с.
4. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста; Питер – М., 2022. – 464 с.
5. Хеклер М. Spring Boot по – быстрому; Эксмо – М., 2022. – 352 с.
6. Фримен Э., Робсон Э. Head First. Паттерны проектирования. 2 – е издание; Питер – М., 2022. – 640 с.
7. Эккель Б. Философия Java. 4 – е полное изд.; Эксмо – М., 2020. – 1242 с.
8. Уоллс К. Spring в действии. Шестое издание; ДМК Пресс – М., 2022. – 544 с.
9. Бауэр К., Кинг Г., Грэгори Г. Java Persistence API и Hibernate; ДМК Пресс – М., 2018. – 632 с.
10. Документация к фреймворку Spring. [Электронный ресурс] / Режим доступа: <https://docs.spring.io/spring-framework/reference/> свободный

Приложение А

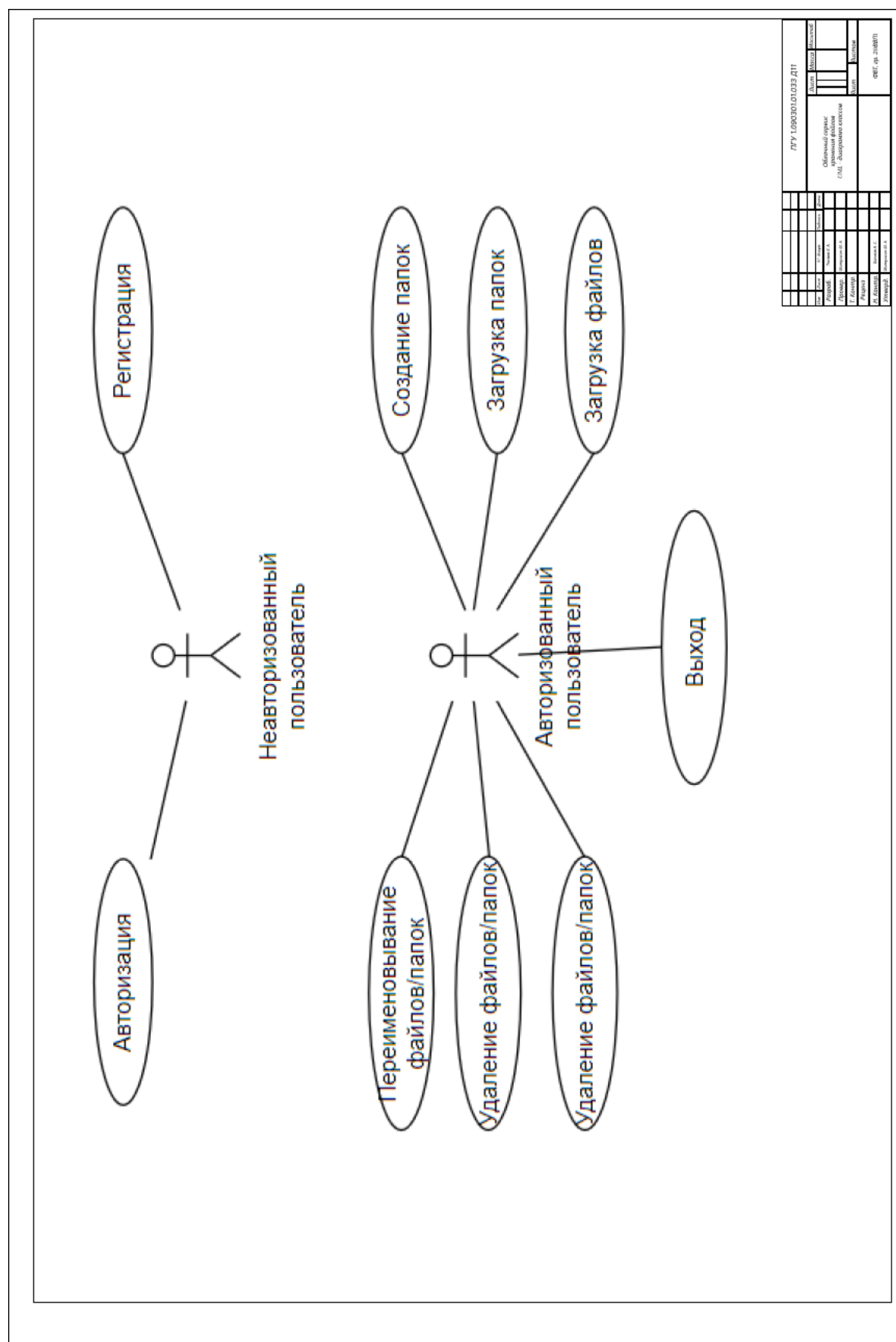
(обязательное)

Облачный сервис хранения файлов

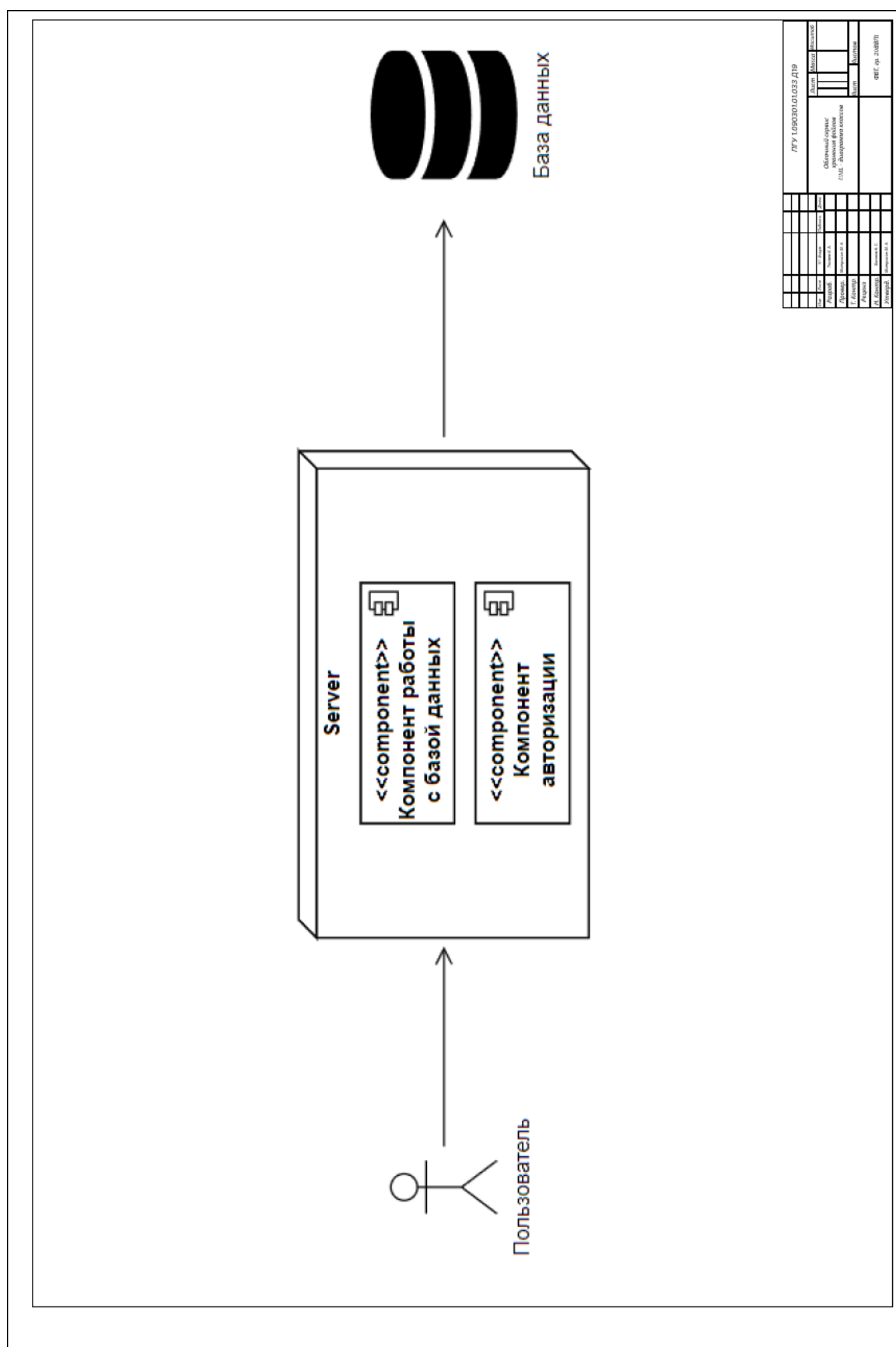
UML-диаграммы

				ПГУ1.090301.12.001 ПЗ	Лист
					43
Изм.	Лист	№ докум	Подпись Дата		

Приложение А.1 – Диаграмма вариантов использования



Приложение А.2 – Диаграмма развертывания



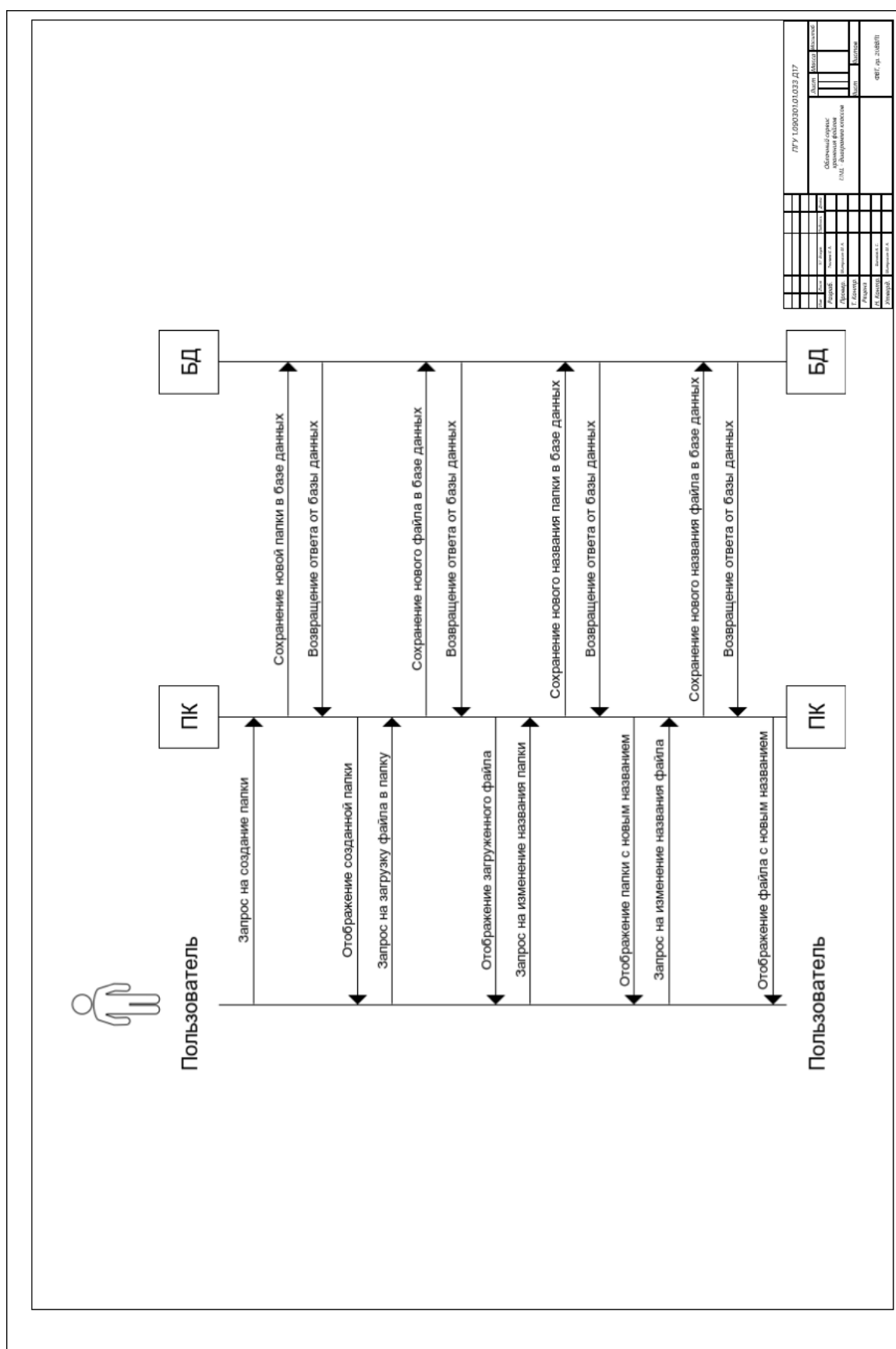
Изм.	Лист	№ докум	Подпись Дата

ПГУ1.090301.12.001 ПЗ

Лист

45

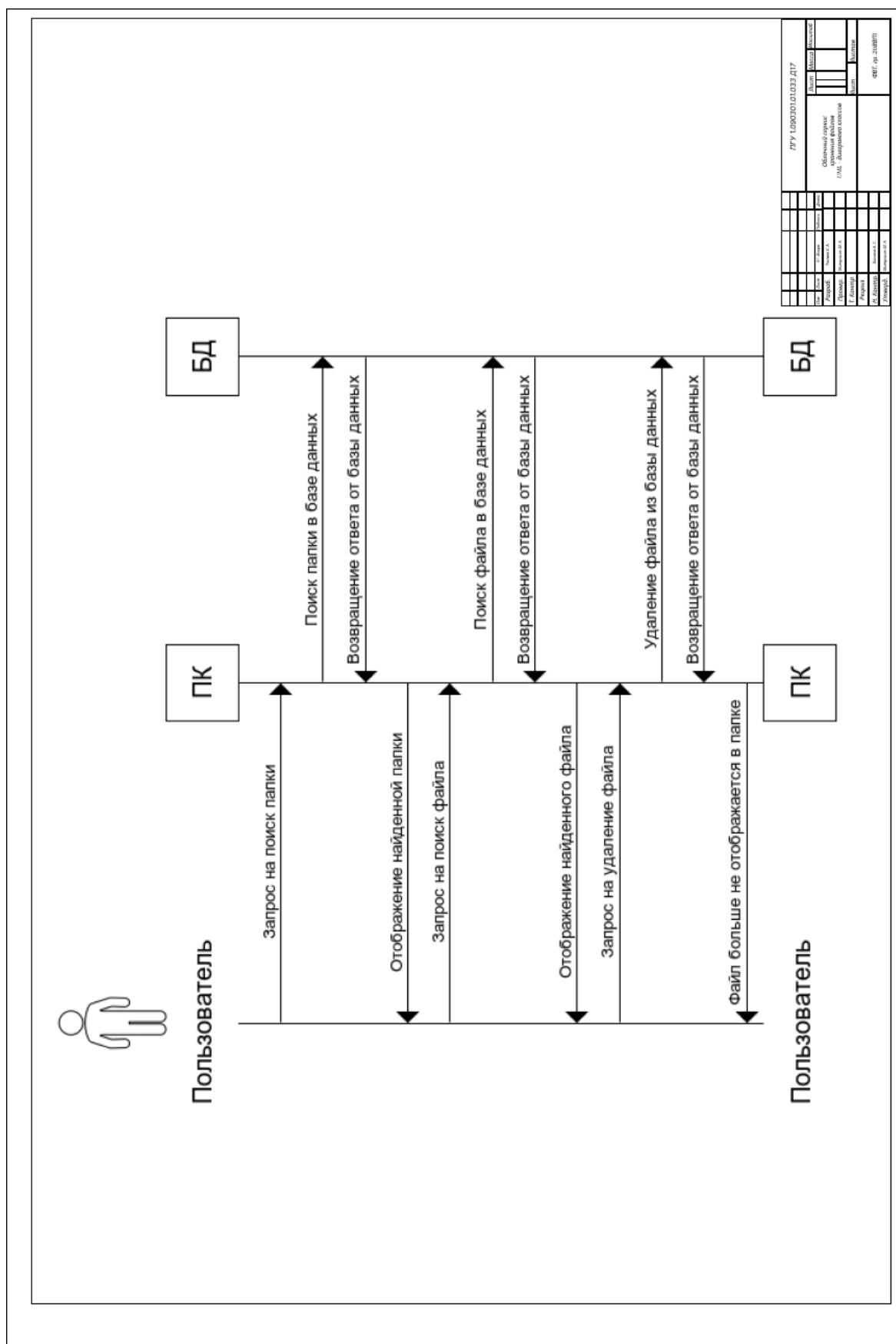
Приложение А.3 – Диаграмма вариантов использования



Изм.	Лист	№ докум	Подпись Дата

ПГУ1.090301.12.001 ПЗ

Приложение А.4 – Диаграмма вариантов использования



Приложение Б

(обязательное)

Облачный сервис хранения файлов

Листинг программы

Файл CloudStorageApplication.java

```
@SpringBootApplication
public class CloudStorageApplication {
    public static void main(String[] args) {
        SpringApplication.run(CloudStorageApplication.class,
args);
    }
}
```

Файл SecurityConfig.java

```
@Configuration
public class SecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager
authManager(UserDetailsService detailsService) {
        DaoAuthenticationProvider daoProvider = new
DaoAuthenticationProvider();
        daoProvider.setUserDetailsService(detailsService);
        daoProvider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(daoProvider);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity
httpSecurity) throws Exception {
        return httpSecurity
            .csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/signup",
"/login").permitAll()
                .anyRequest().authenticated())
    }
}
```

				ПГУ1.090301.12.001 ПЗ	Лист
					49
Изм.	Лист	№ докум	Подпись Дата		

```

        .formLogin(login -> login
                    .loginPage("/login")

        .loginProcessingUrl("/login/process")
                    .defaultSuccessUrl("/", true)
                    .failureUrl("/login?error"))
        .logout(logout -> logout
                 .logoutUrl("/logout")
                 .logoutSuccessUrl("/login"))
        .build();
    }
}

```

Файл GlobalExceptionHandler.java

```

@ControllerAdvice
public class GlobalExceptionHandler {
    private static final Logger LOGGER =
LoggerFactory.getLogger(GlobalExceptionHandler.class);

    @Value("${spring.servlet.multipart.max-file-size:10MB}")
    private String MAX_UPLOAD_SIZE;

    @ExceptionHandler(MaxUploadSizeExceededException.class)
    public String
handleMaxUploadException(HttpServletRequest request,

RedirectAttributes redirectAttributes) {
        LOGGER.warn("Max upload size exceeded: {}",
MAX_UPLOAD_SIZE);
        redirectAttributes.addFlashAttribute("failureAlert",
        "Uploaded object size must not exceed " +
MAX_UPLOAD_SIZE);
        String path = request.getParameter("path");
        return "redirect:/" + getPathParam(path);
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					50
Изм.	Лист	№ докум	Подпись Дата		

```

        @ExceptionHandler(Throwable.class)
        public String logAllUnhandledExceptions(Exception
exception) {
            LOGGER.error("An error occurred", exception);
            return "redirect:/error";
        }
    }
}

```

Файл AuthenticationController.java

```

@Controller
@RequiredArgsConstructor
public class AuthenticationController {
    private static final Logger LOGGER =
LoggerFactory.getLogger(AuthenticationController.class);
    private final UserService userService;

    @GetMapping("/login")
    public String getLoginForm() {
        return "auth/login";
    }

    @GetMapping("/signup")
    public String getSignupForm(Model model) {
        model.addAttribute("user", new User());
        return "auth/signup";
    }

    @PostMapping("/signup")
    public String signUpUser(@ModelAttribute("user") @Valid
User user,
                                BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            return "auth/signup";
        }
        try {
            userService.createUser(user);

```

				ПГУ1.090301.12.001 ПЗ	Лист
					51
Изм.	Лист	№ докум	Подпись Дата		

```

        LOGGER.debug("Sign up successful for user \"{}\" (id={})", user.getUsername(), user.getId());
        return "redirect:/login";
    } catch (UsernameAlreadyExistsException exc) {
        LOGGER.debug("Sign up failed for user \"{}\"", user.getUsername(), exc);
        bindingResult.rejectValue("username", "user.alreadyExists", "User with this username already exists");
        return "auth/signup";
    }
}
}
}

```

Файл FileController.java

```

@Controller
@RequestMapping("/file")
@RequiredArgsConstructor
public class FileController {
    private static final Logger LOGGER =
        LoggerFactory.getLogger(FileController.class);
    private final FileService fileService;

    @GetMapping("/download")
    public void downloadFile(
        @AuthenticationPrincipal UserDetailsImpl
        userDetails,
        @RequestParam String path,
        HttpServletResponse response) {
        try (InputStream fileContentStream =
            fileService.getFileContent(userDetails.getUserId(), path)) {
            response.setHeader("Content-Disposition",
                String.format("attachment;
filename=\"%s\"", extractObjectName(path)));
            FileCopyUtils.copy(fileContentStream,
                response.getOutputStream());
        }
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					52
Изм.	Лист	№ докум	Подпись Дата		

```

        } catch (ObjectNotExistsException | IOException exc)
        {
            LOGGER.warn("Failed to download file \"{}\";",
path, exc);
        }
    }

    @PostMapping("/upload")
    public String uploadFiles(
        @AuthenticationPrincipal UserDetailsImpl
userDetails,
        @RequestParam String path,
        @ModelAttribute("filesUploadDto") @Valid
FilesUploadDto filesUploadDto,
        BindingResult bindingResult,
        RedirectAttributes redirectAttributes) {
        if (bindingResult.hasErrors()) {

redirectAttributes.addFlashAttribute("failureAlert",
bindingResult.getFieldError().getDefaultMessage());
            return "redirect:/" + getPathParam(path);
        }
        try {
            fileService.saveFiles(userDetails.getUserId(),
filesUploadDto);
        } catch (ObjectAlreadyExistsException exc) {
            LOGGER.debug("Failed to upload files", exc);

redirectAttributes.addFlashAttribute("failureAlert", "File with
this name already exists");
        } catch (ObjectUploadException exc) {
            LOGGER.warn("Failed to upload files", exc);

redirectAttributes.addFlashAttribute("failureAlert", "An error
occurred during uploading");
        }
    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					53
Изм.	Лист	№ докум	Подпись Дата		

```

        }
        return "redirect:/" + getPathParam(path);
    }

    @GetMapping("/rename")
    public String getFileRenameForm(
        @AuthenticationPrincipal UserDetailsImpl
        userDetails,
        @RequestParam String path,
        Model model,
        HttpServletRequest request) {
        try {

            fileService.validateFileExists(userDetails.getUserId(), path);
        } catch (ObjectNotExistsException exc) {
            LOGGER.warn("Failed to get file renaming form for
            path \"{}\\\":", path, exc);
            return "redirect:/" ;
        }

        if (!model.containsAttribute("objectRenameDto")) {
            ObjectRenameDto renameDto = new
            ObjectRenameDto();
            renameDto.setNewName(extractObjectName(path));
            model.addAttribute("objectRenameDto",
            renameDto);
        }
        model.addAttribute("requestURI",
        request.getRequestURI());
        model.addAttribute("breadcrumbs",
        assembleBreadcrumbsFromPath(path));
        model.addAttribute("searchDto", new SearchDto());
        return "rename";
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					54
Изм.	Лист	№ докум	Подпись Дата		

```

        @PatchMapping("/rename")
        public String renameFile(
            @AuthenticationPrincipal UserDetailsImpl
            userDetails,
            @RequestParam(required = false) String path,
            @ModelAttribute("objectRenameDto") @Valid
            ObjectRenameDto objectRenameDto,
            BindingResult bindingResult,
            RedirectAttributes redirectAttributes) {
            if (bindingResult.hasErrors()) {

                redirectAttributes.addFlashAttribute("org.springframework.validation.BindingResult.objectRenameDto", bindingResult);

                redirectAttributes.addFlashAttribute("objectRenameDto",
                objectRenameDto);

                return "redirect:/file/rename" +
                getPathParam(path);
            }

            try {
                String parentFolderPath =
                fileService.renameFile(userDetails.getUserId(), objectRenameDto);
                return "redirect:/" +
                getPathParam(parentFolderPath);
            } catch (ObjectAlreadyExistsException exc) {
                LOGGER.debug("Failed to rename file", exc);

                redirectAttributes.addFlashAttribute("failureAlert", "File with
                this name already exists");

                redirectAttributes.addFlashAttribute("objectRenameDto",
                objectRenameDto);

                return "redirect:/file/rename" +
                getPathParam(path);
            }
        }
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					55
Изм.	Лист	№ докум	Подпись Дата		

```

        }
    }

    @GetMapping("/move")
    public String getFileMoveForm(
        @AuthenticationPrincipal UserDetailsImpl
        userDetails,

        @RequestParam String path,
        @ModelAttribute("objectMoveDto") ObjectMoveDto
        objectMoveDto,

        Model model,
        HttpServletRequest request) {
        try {
            model.addAttribute("moveCandidates",
                fileService.getMoveCandidatesForFile(userDetails.getUserId(),
                path));
        } catch (ObjectNotExistsException exc) {
            LOGGER.warn("Failed to get file moving form for
                path \"{}\\\":", path, exc);
            return "redirect:/";
        }
        model.addAttribute("requestURI",
            request.getRequestURI());
        model.addAttribute("breadcrumbs",
            assembleBreadcrumbsFromPath(path));
        model.addAttribute("searchDto", new SearchDto());
        return "move";
    }

    @PutMapping("/move")
    public String moveFile(
        @AuthenticationPrincipal UserDetailsImpl
        userDetails,

        @RequestParam(required = false) String path,

```

```

        @ModelAttribute("objectMoveDto")    ObjectMoveDto
objectMoveDto,

        RedirectAttributes redirectAttributes) {
    try {
        String                oldParentPath                =
fileService.moveFile(userDetails.getUserId(), objectMoveDto);

        redirectAttributes.addFlashAttribute("successAlert",    "File    was
moved successfully");

        return                "redirect:/"                +
getPathParam(oldParentPath);
    } catch (ObjectAlreadyExistsException exc) {
        LOGGER.debug("Failed to move file", exc);

        redirectAttributes.addFlashAttribute("failureAlert",
                "File with this name already exists in
target location");
    }
    return "redirect:/file/move" + getPathParam(path);
}

    @DeleteMapping
    public String deleteFile(
        @AuthenticationPrincipal                UserDetailsImpl
userDetails,

        @RequestParam String path) {
        String                parentFolder                =
fileService.deleteFile(userDetails.getUserId(), path);
        return "redirect:/" + getPathParam(parentFolder);
    }
}

Файл FolderController.java

@Controller
@RequiredArgsConstructor
public class FolderController {

```

				ПГУ1.090301.12.001 ПЗ	Лист
					57
Изм.	Лист	№ докум	Подпись Дата		

```

        private static final Logger LOGGER =
LoggerFactory.getLogger(FileController.class);

        private final FolderService folderService;

        @GetMapping
        public String listFolder(

                @AuthenticationPrincipal UserDetailsImpl
userDetails,

                @RequestParam(required = false) String path,

                Model model) {

            try {

                model.addAttribute("objects",
folderService.getFolderObjects(userDetails.getUserId(), path));

                model.addAttribute("breadcrumbs",
assembleBreadcrumbsFromPath(path));

                model.addAttribute("folderCreationDto", new
FolderCreationDto());

                model.addAttribute("folderUploadDto", new
FilesUploadDto());

                model.addAttribute("filesUploadDto", new
FilesUploadDto());

                model.addAttribute("searchDto", new SearchDto());
            } catch (ObjectNotExistsException exc) {

                LOGGER.debug("Failed to list folder \"{}\"", path ==
null || path.isEmpty() ? "/" : path, exc);

                model.addAttribute("wrongPath", path);
            }

            return "main";
        }

        @PostMapping

```

				ПГУ1.090301.12.001 ПЗ	Лист
					58
Изм.	Лист	№ докум	Подпись Дата		

```

        public String createFolder(
            @AuthenticationPrincipal UserDetailsImpl
userDetails,
            @RequestParam(required = false) String path,
            @ModelAttribute("folderCreationDto") @Valid
FolderCreationDto folderCreationDto,
            BindingResult bindingResult,
            RedirectAttributes redirectAttributes) {
        if (bindingResult.hasErrors()) {
            redirectAttributes.addFlashAttribute("failureAlert",
bindingResult.getFieldError().getDefaultMessage());
        } else {
            try {

folderService.createSingleFolder(userDetails.getUserId(),
folderCreationDto);

            } catch (ObjectAlreadyExistsException exc) {
                LOGGER.debug("Failed to create folder", exc);

redirectAttributes.addFlashAttribute("failureAlert", "This
folder already exists");
            }
        }
        return "redirect:/" + getPathParam(path);
    }

    @PostMapping("/upload")
    public String uploadFolder(
        @AuthenticationPrincipal UserDetailsImpl
userDetails,
        @RequestParam String path,
        @ModelAttribute("folderUploadDto") @Valid
FilesUploadDto filesUploadDto,
        BindingResult bindingResult,
        RedirectAttributes redirectAttributes) {

```

				ПГУ1.090301.12.001 ПЗ	Лист
					59
Изм.	Лист	№ докум	Подпись Дата		

```

        if (bindingResult.hasErrors()) {
            redirectAttributes.addFlashAttribute("failureAlert",
bindingResult.getFieldError().getDefaultMessage());
            return "redirect:/" + getPathParam(path);
        }

        try {
            folderService.saveFolder(userDetails.getUserId(),
filesUploadDto);
        } catch (ObjectAlreadyExistsException exc) {
            LOGGER.debug("Failed to upload folder", exc);
            redirectAttributes.addFlashAttribute("failureAlert",
"Folder with this name already exists");
        } catch (ObjectUploadException exc) {
            LOGGER.warn("Failed to upload folder", exc);
            redirectAttributes.addFlashAttribute("failureAlert",
"An error occurred during uploading");
        }
        return "redirect:/" + getPathParam(path);
    }

    @GetMapping("/download")
    public void downloadZippedFolder(
        @AuthenticationPrincipal UserDetailsImpl
userDetails,
        @RequestParam String path,
        HttpServletResponse response) {
        response.setHeader("Content-Disposition",
            String.format("attachment; filename=\"%s\"",
composeZipArchiveName(path)));
        try {
            folderService.writeFolderContent(userDetails.getUserId(), path,
response.getOutputStream());
        }
    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					60
Изм.	Лист	№ докум	Подпись Дата		

```

        } catch (IOException exc) {
            LOGGER.warn("Failed to download folder", exc);
        }
    }

    private String composeZipArchiveName(String path) {
        String zipArchiveName;
        try {
            zipArchiveName = extractObjectName(path);
        } catch (IllegalArgumentException exc) {
            zipArchiveName = "all-files";
        }
        return zipArchiveName + ".zip";
    }

    @GetMapping("/rename")
    public String getFolderRenameForm(
        @AuthenticationPrincipal UserDetailsImpl
        userDetails,
        @RequestParam String path,
        Model model,
        HttpServletRequest request) {
        try {

            folderService.validateFolderExists(userDetails.getUserId(),
            path); // protection from manual url editing

            } catch (ObjectNotExistsException exc) {

                LOGGER.warn("Failed to get folder rename form for
            path \"{}\"", path, exc);

                return "redirect:/";

            }

            if (!model.containsAttribute("objectRenameDto")) {

```

				ПГУ1.090301.12.001 ПЗ	Лист
					61
Изм.	Лист	№ докум	Подпись Дата		

```

        ObjectRenameDto renameDto = new ObjectRenameDto();
        renameDto.setNewName(extractObjectName(path));
        model.addAttribute("objectRenameDto", renameDto);
    }

    model.addAttribute("requestURI",
request.getRequestURI());

    model.addAttribute("breadcrumbs",
assembleBreadcrumbsFromPath(path));

    model.addAttribute("searchDto", new SearchDto());
    return "rename";
}

@PatchMapping("/rename")
public String renameFolder(
        @AuthenticationPrincipal UserDetailsImpl
userDetails,
        @RequestParam(required = false) String path,
        @ModelAttribute("objectRenameDto") @Valid
ObjectRenameDto objectRenameDto,
        BindingResult bindingResult,
        RedirectAttributes redirectAttributes) {
    if (bindingResult.hasErrors()) {

redirectAttributes.addFlashAttribute("org.springframework.valida
tion.BindingResult.objectRenameDto", bindingResult);

redirectAttributes.addFlashAttribute("objectRenameDto",
objectRenameDto);

        return "redirect:/rename" + getPathParam(path);
    }

    try {

        String parentFolderPath =
folderService.renameFolder(userDetails.getUserId(),
objectRenameDto);

```

				ПГУ1.090301.12.001 ПЗ	Лист
					62
Изм.	Лист	№ докум	Подпись Дата		

```

        return "redirect:/" +
getPathParam(parentFolderPath);

    } catch (ObjectAlreadyExistsException exc) {

        LOGGER.debug("Failed to rename folder", exc);

        redirectAttributes.addFlashAttribute("failureAlert",
"Folder with this name already exists");

redirectAttributes.addFlashAttribute("objectRenameDto",
objectRenameDto);

        return "redirect:/rename" + getPathParam(path);

    }

}

@GetMapping("/move")
public String getFolderMoveForm(

    @AuthenticationPrincipal UserDetailsImpl
userDetails,

    @RequestParam String path,

    @ModelAttribute("objectMoveDto") ObjectMoveDto
objectMoveDto,

    Model model,

    HttpServletRequest request) {

    try {

        model.addAttribute("moveCandidates",
folderService.getMoveCandidatesForFolder(userDetails.getUserId()
, path));

    } catch (ObjectNotExistsException exc) {

        LOGGER.warn("Failed to get folder moving form for
path \"{}}\"", path, exc);

        return "redirect:/" ;

    }

    model.addAttribute("requestURI",
request.getRequestURI());

```

				ПГУ1.090301.12.001 ПЗ	Лист
					63
Изм.	Лист	№ докум	Подпись Дата		

```

        model.addAttribute("breadcrumbs",
assembleBreadcrumbsFromPath(path));

        model.addAttribute("searchDto", new SearchDto());

        return "move";

    }

    @PutMapping("/move")
    public String moveFolder(

        @AuthenticationPrincipal UserDetailsImpl
userDetails,

        @RequestParam(required = false) String path,

        @ModelAttribute("objectMoveDto") ObjectMoveDto
objectMoveDto,

        RedirectAttributes redirectAttributes) {

        try {

            String oldParentPath =
folderService.moveFolder(userDetails.getUserId(),
objectMoveDto);

            redirectAttributes.addFlashAttribute("successAlert",
"Folder was moved successfully");

            return "redirect:/" + getPathParam(oldParentPath);

        } catch (ObjectAlreadyExistsException exc) {

            LOGGER.warn("Failed to move folder", exc);

            redirectAttributes.addFlashAttribute("failureAlert",
"Folder with this name already exists in
target location");

        }

        return "redirect:/move" + getPathParam(path);

    }

    @DeleteMapping
    public String deleteFolder(

        @AuthenticationPrincipal UserDetailsImpl
userDetails,

```

				ПГУ1.090301.12.001 ПЗ	Лист
					64
Изм.	Лист	№ докум	Подпись Дата		

```

        @RequestParam String path) {

        String parentFolder =
folderService.deleteFolder(userDetails.getUserId(), path);

        return "redirect:/" + getPathParam(parentFolder);

    }

}

```

Файл SearchController.java

```

@Controller
@RequiredArgsConstructor
@RequestMapping("/search")
public class SearchController {

    private final SearchService searchService;

    @GetMapping
    public String searchByEntry(

        @AuthenticationPrincipal UserDetailsImpl
userDetails,

        @ModelAttribute("searchDto") SearchDto searchDto,

        Model model) {

        model.addAttribute("objects",
searchService.searchObjectsByString(userDetails.getUserId(),
searchDto));

        return "search";

    }

}

```

Файл FilesUploadDto.java

```

@Data
public class FilesUploadDto {

    private String parentFolderPath;

    @FilenamePattern(regexp = "^[\\w !.*+\\[\\]\\'()\\-
]+/)*[\\w !.*+\\[\\]\\'()\\-]+$",

        message = "File names must not contain
unsupported characters or non-English letters")

```

				ПГУ1.090301.12.001 ПЗ	Лист
					65
Изм.	Лист	№ докум	Подпись Дата		

```

        private List<MultipartFile> files;
    }

```

Файл FolderCreationDto.java

```

@Data

public class FolderCreationDto {

    private String parentFolderPath;

    @Pattern(regexp = "^[\\w !.*+'\"\\[\\] ()\\-]+$", message =
"Folder name must not contain unsupported characters or non-
English letters")

    private String newFolderName;

}

```

Файл ObjectInfoDto.java

```

@Data

@AllArgsConstructor

public class ObjectInfoDto {

    private String name;

    private boolean isFolder;

    private String path;

    private String size;

    private LocalDateTime lastModified;

}

```

Файл ObjectMoveDto.java

```

@Data

public class ObjectMoveDto {

    private String oldObjectPath;

    private String newObjectPath;

}

```

Файл ObjectRenameDto.java

```

@Data

public class ObjectRenameDto implements Serializable {

    private String objectPath;

    @Pattern(regexp = "^[\\w !.*+'\"\\[\\] ()\\-]+$", message =
"Name must not contain unsupported characters or non-English
letters")

}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					66
Изм.	Лист	№ докум	Подпись Дата		

```

        private String newName;
    }

```

Файл SearchDto.java

```

@Data
public class SearchDto {
    private String query;
}

```

Файл MinioOperationException.java

```

@Repository
public interface ProductDao extends CrudRepository<Product,
Integer> {

    public List<Product> findAll(Pageable pageable);

    public List<Product>
findByProductNameContainingIgnoreCaseOrProductDescriptionContain
ingIgnoreCase(

        String key1, String key2, Pageable pageable

    );
}

```

Файл ObjectAlreadyExistsException.java

```

public class ObjectAlreadyExistsException extends
RuntimeException {

    public ObjectAlreadyExistsException(String message) {

        super(message);

    }
}

```

Файл ObjectNotExistsException.java

```

public class ObjectNotExistsException extends RuntimeException {

    public ObjectNotExistsException(String message) {

        super(message);

    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					67
Изм.	Лист	№ докум	Подпись Дата		

Файл ObjectUploadException.java

```
public class ObjectUploadException extends RuntimeException {  
    public ObjectUploadException(String message) {  
        super(message);  
    }  
  
    public ObjectUploadException(String message, Throwable  
cause) {  
        super(message, cause);  
    }  
}
```

Файл UsernameAlreadyExistsException.java

```
public class UsernameAlreadyExistsException extends  
RuntimeException {  
    public UsernameAlreadyExistsException(String message,  
Throwable cause) {  
        super(message, cause);  
    }  
}
```

Файл UserDetailsImpl.java

```
public class UserDetailsImpl implements UserDetails {  
    private final String password;  
    private final String username;  
    private final int userId;  
  
    public UserDetailsImpl(User user) {  
        this.username = user.getUsername();  
        this.password = user.getPassword();  
        this.userId = user.getId();  
    }  
  
    @Override
```

				ПГУ1.090301.12.001 ПЗ	Лист
					68
Изм.	Лист	№ докум	Подпись Дата		

```

    public Collection<? extends GrantedAuthority>
getAuthorities() {

        return new ArrayList<>();

    }

    @Override
    public String getPassword() {

        return password;

    }

    @Override
    public String getUsername() {

        return username;

    }

    public int getUserId() {

        return userId;

    }

    @Override
    public boolean isAccountNonExpired() {

        return true;

    }

    @Override
    public boolean isAccountNonLocked() {

        return true;

    }

    @Override
    public boolean isCredentialsNonExpired() {

        return true;

    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					69
Изм.	Лист	№ докум	Подпись Дата		

```

    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}

```

Файл ObjectInfo.java

```

@Data
@AllArgsConstructor
public class ObjectInfo {
    private ObjectPath objectPath;
    private long size;
    private LocalDateTime lastModified;
}

```

Файл User.java

```

@Data
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "username", unique = true)
    @Size(min = 2, max = 50, message = "Username length must be
in between 2 and 50 symbols")
    private String username;
    @Column(name = "password")
    @Size(min = 5, max = 100, message = "Password length must be
in between 5 and 100 symbols")
    private String password;
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					70
Изм.	Лист	№ докум	Подпись Дата		

```

        public User(String username, String password) {
            this.username = username;
            this.password = password;
        }
    }

    Файл MinioObjectPath.java
    @EqualsAndHashCode

    @RequiredArgsConstructor(access = AccessLevel.PRIVATE)
    public class MinioObjectPath implements ObjectPath {
        private final String homeFolder;
        private final String objectPath;

        static MinioObjectPath parse(String fullPath) {
            validatePathIsNotNullAndNotEmpty(fullPath);

            Pattern pattern = Pattern.compile("(?<home>user-\\d+-files)(?<path>/.*)$");

            Matcher matcher = pattern.matcher(fullPath);

            if (!matcher.matches()) {
                throw new IllegalArgumentException("Wrong path
format: " + fullPath);
            }

            String homeFolder = matcher.group("home");
            String objectPath = matcher.group("path");
            validatePathFormat(objectPath);
            return new MinioObjectPath(homeFolder, objectPath);
        }

        static MinioObjectPath compose(int userId, String
objectPath) {
            validatePathIsNotNullAndNotEmpty(objectPath);
            validatePathFormat(objectPath);

```

```

        String homeFolder = getUserHomeFolder(userId);
        return new MinioObjectPath(homeFolder, objectPath);
    }

    private static void validatePathIsNotNullAndNotEmpty(String
path) {
        if (path == null) {
            throw new IllegalArgumentException("Path cannot be
null");
        }
        if (path.isEmpty()) {
            throw new IllegalArgumentException("Path cannot be
empty");
        }
    }

    private static void validatePathFormat(String objectPath) {
        if
(!objectPath.matches("(^/([^/]+)/)*$|^/?([^/]+)/*[^/]+$")) {
            throw new IllegalArgumentException("Invalid path
format: " + objectPath);
        }
    }

    private static String getUserHomeFolder(int userId) {
        return String.format("user-%d-files", userId);
    }

    @Override
    public boolean isRootFolder() {
        return objectPath.equals("/");
    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					72
Изм.	Лист	№ докум	Подпись Дата		

```

@Override
public String getObjectNames() {
    if (objectPath.equals("/")) {
        return "";
    }
    String path = objectPath;
    if (isFolder()) {
        path = path.substring(0, path.length() - 1);
    }
    return path.substring(path.lastIndexOf("/") + 1);
}

@Override
public String getPath() {
    return objectPath;
}

@Override
public String getFullPath() {
    return homeFolder + objectPath;
}

@Override
public MinioObjectPath getParent() {
    if (objectPath.equals("/")) {
        throw new UnsupportedOperationException("Root folder
has no path without object name");
    }
    String path = objectPath;
    if (isFolder()) {
        path = path.substring(0, path.length() - 1);
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					73
Изм.	Лист	№ докум	Подпись Дата		

```

        path = path.substring(0, path.lastIndexOf("/") + 1);
        return new MinioObjectPath(homeFolder, path);
    }

    @Override
    public boolean isFolder() {
        return objectPath.endsWith("/");
    }

    @Override
    public boolean isInFolder(ObjectPath folderPath) {
        if (!folderPath.isFolder()) {
            throw new IllegalArgumentException(String.format(
                "isInFolder() for objectPath %s failed,
because passed object is not a folder: %s",
                this, folderPath));
        }

        return
this.getFullPath().startsWith(folderPath.getFullPath());
    }

    @Override
    public MinioObjectPath replacePrefix(String prefixToReplace,
String replacement) {
        validateFolderPathFormat(prefixToReplace);
        validateFolderPathFormat(replacement);
        if (!objectPath.startsWith(prefixToReplace)) {
            return this;
        }

        String newObjectPath =
objectPath.replaceFirst(prefixToReplace, replacement);
        return new MinioObjectPath(homeFolder, newObjectPath);
    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					74
Изм.	Лист	№ докум	Подпись Дата		

```

@Override

public MinioObjectPath resolve(String objectName) {
    if (!isFolder()) {
        throw new UnsupportedOperationException("Cannot
append for path that is not a folder: " + this);
    }
    if (objectName == null) {
        throw new IllegalArgumentException("Cannot append
null value for path: " + this);
    }
    String newObjectPath = objectPath + objectName;
    return new MinioObjectPath(homeFolder, newObjectPath);
}

private void validateFolderPathFormat(String folderPath) {
    if (!folderPath.matches("^(/[\\w !.*'()\\-]+/)*$")) {
        throw new IllegalArgumentException("Invalid folder
path format: " + folderPath);
    }
}

@Override

public MinioObjectPath renameObject(String newName) {
    if (objectPath.equals("/")) {
        throw new UnsupportedOperationException("Cannot
rename root folder");
    }
    String pathWithoutName = objectPath;
    if (isFolder()) {
        pathWithoutName = pathWithoutName.substring(0,
pathWithoutName.length() - 1);
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					75
Изм.	Лист	№ докум	Подпись Дата		

```

        pathWithoutName = pathWithoutName.substring(0,
pathWithoutName.lastIndexOf("/") + 1);

        String newObjectPath = pathWithoutName + newName +
(isFolder() ? "/" : "");

        return new MinioObjectPath(homeFolder, newObjectPath);
    }

    @Override
    public String toString() {
        return getFullPath();
    }
}

```

Файл MinioObjectPathFactory.java

```

@Component

public class MinioObjectPathFactory implements ObjectPathFactory
{
    @Override
    public ObjectPath compose(int userId, String objectPath) {
        return MinioObjectPath.compose(userId, objectPath);
    }
}

```

Файл MinioRepository.java

```

@Service

public class MinioRepository implements FileRepository {
    private static final Logger LOGGER =
LoggerFactory.getLogger(MinioRepository.class);

    private static final String ROOT_BUCKET = "user-files";
    private final MinioClient client;

    public MinioRepository(Environment env) {
        String endpoint = env.getProperty("minio.endpoint",
"localhost");
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					76
Изм.	Лист	№ докум	Подпись Дата		

```

        String accessKey =
env.getRequiredProperty("minio.username");

        String secretKey =
env.getRequiredProperty("minio.password");

        client = MinioClient.builder()
                .endpoint(endpoint)
                .credentials(accessKey, secretKey)
                .build();

        LOGGER.info("Connected to MinIO server successfully.
Endpoint: {}", endpoint);

        createRootBucket();
    }

    private void createRootBucket() {
        try {
            boolean exists =
client.bucketExists(BucketExistsArgs.builder()
                .bucket(ROOT_BUCKET)
                .build());

            if (!exists) {
                client.makeBucket(MakeBucketArgs.builder()
                        .bucket(ROOT_BUCKET)
                        .build());

                LOGGER.info("Root bucket created");
            }

            LOGGER.debug("Root bucket not created: already
exists");
        } catch (Exception exc) {
            throw new MinioOperationException(exc);
        }
    }
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					77
Изм.	Лист	№ докум	Подпись Дата		

```

@Override

public boolean isObjectExist(ObjectPath objectPath) {
    try {
        client.statObject(
            StatObjectArgs.builder()
                .bucket(ROOT_BUCKET)
                .object(objectPath.getFullPath())
                .build());

        return true;
    } catch (ErrorResponseException exc) {
        return false;
    } catch (Exception exc) {
        throw new MinioOperationException(exc);
    }
}

@Override

public List<ObjectInfo> listFolderObjects(ObjectPath
folderPath) {
    return listFolderObjectsWithParams(folderPath, false,
false);
}

@Override

public List<ObjectInfo>
listFolderObjectsRecursive(ObjectPath folderPath, boolean
includeSelf) {
    return listFolderObjectsWithParams(folderPath, true,
includeSelf);
}

private List<ObjectInfo> listFolderObjectsWithParams(

```

```

        ObjectPath folderPath, boolean recursive, boolean
includeSelf) {
    validateFolderPath(folderPath);
    try {
        Iterable<Result<Item>> results = client.listObjects(
            ListObjectsArgs.builder()
                .bucket(ROOT_BUCKET)
                .prefix(folderPath.getFullPath())
                .recursive(recursive)
                .build());
        List<ObjectInfo> objects = new ArrayList<>();
        for (Result<Item> result : results) {
            Item item = result.get();
            if (isSameFolder(item, folderPath) &&
!includeSelf) {
                continue;
            }
            objects.add(composeObjectInfo(item));
        }
        return objects;
    } catch (Exception exc) {
        throw new MinioOperationException(exc);
    }
}

```

```

    private boolean isSameFolder(Item item, ObjectPath
folderPath) {
        return
item.objectName().equals(folderPath.getFullPath());
    }

```

```

    private ObjectInfo composeObjectInfo(Item item) {
        LocalDateTime lastModified = item.isDir()

```

				ПГУ1.090301.12.001 ПЗ	Лист
					79
Изм.	Лист	№ докум	Подпись Дата		

```

        ? null

        :
item.lastModified().withZoneSameInstant(ZoneOffset.of("+03:00"))
.toLocalDateTime();

return new ObjectInfo(

    MinioObjectPath.parse(item.objectName()),

    item.size(),

    lastModified

);

}

@Override

public void createFile(ObjectPath filePath, InputStream
fileInputStream) {

    validateFilePath(filePath);

    try {

        client.putObject(

            PutObjectArgs.builder()

                .bucket(ROOT_BUCKET)

                .object(filePath.getFullPath())

                .stream(fileInputStream, -1,

10485760)

                .build());

    } catch (Exception exc) {

        throw new MinioOperationException(exc);

    }

}

private void validateFolderPath(ObjectPath folderPath) {

    if (!folderPath.isFolder()) {

        throw new IllegalArgumentException("Passed path is
not a folder: " + folderPath.getFullPath());

    }

}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					80
Изм.	Лист	№ докум	Подпись Дата		

```

    }

    private void validateFilePath(ObjectPath filePath) {
        if (filePath.isFolder()) {
            throw new IllegalArgumentException("Passed path is
not a file: " + filePath.getFullPath());
        }
    }

    @Override
    public void createFolder(ObjectPath folderPath) {
        validateFolderPath(folderPath);
        try {
            client.putObject(
                PutObjectArgs.builder()
                    .bucket(ROOT_BUCKET)
                    .object(folderPath.getFullPath())
                    .stream(new ByteArrayInputStream(new
byte[]{}), 0, -1)
                    .build());
        } catch (Exception exc) {
            throw new MinioOperationException(exc);
        }
    }

    private void moveObject(ObjectPath oldPath, ObjectPath
newPath) {
        if (newPath.equals(oldPath)) {
            return;
        }

        try {
            client.copyObject(

```

				ПГУ1.090301.12.001 ПЗ	Лист
					81
Изм.	Лист	№ докум	Подпись Дата		

```

        CopyObjectArgs.builder()
            .bucket(ROOT_BUCKET)
            .object(newPath.getFullPath())
            .source(CopySource.builder()
                .bucket(ROOT_BUCKET)
                .object(oldPath.getFullPath())
                .build())
            .build();

        client.removeObject(
            RemoveObjectArgs.builder()
                .bucket(ROOT_BUCKET)
                .object(oldPath.getFullPath())
                .build());
    } catch (Exception exc) {
        throw new MinioOperationException(exc);
    }
}

@Override
public void moveFile(ObjectPath oldFilePath, ObjectPath
newFilePath) {
    validateFilePath(oldFilePath);
    validateFilePath(newFilePath);
    moveObject(oldFilePath, newFilePath);
}

@Override
public void moveFolder(ObjectPath oldFolderPath, ObjectPath
newFolderPath) {
    validateFolderPath(oldFolderPath);
    validateFolderPath(newFolderPath);
}

```

```

        if (newFolderPath.equals(oldFolderPath)) {
            return;
        }

        listFolderObjectsWithParams(oldFolderPath, true,
true).forEach(
            objectInfo -> {
                ObjectPath oldSubfolderObjectPath =
objectInfo.getObjectPath();
                ObjectPath newSubfolderObjectPath =
oldSubfolderObjectPath.replacePrefix(
                    oldFolderPath.getPath(),
newFolderPath.getPath());
                moveObject(oldSubfolderObjectPath,
newSubfolderObjectPath);
            });
    }

@Override
public void deleteFile(ObjectPath filePath) {
    validateFilePath(filePath);
    try {
        client.removeObject(
            RemoveObjectArgs.builder()
                .bucket(ROOT_BUCKET)
                .object(filePath.getFullPath())
                .build());
    } catch (Exception exc) {
        throw new MinioOperationException(exc);
    }
}
}

```

```

@Override

public void deleteFolder(ObjectPath folderPath) {
    validateFolderPath(folderPath);

    List<DeleteObject> objectsToDelete =
listFolderObjectsRecursive(folderPath, true).stream()
        .map(objInfo -> new
DeleteObject(objInfo.getObjectPath().getFullPath()))
        .toList();

    try {
        Iterable<Result<DeleteError>> results =
client.removeObjects(
            RemoveObjectsArgs.builder()
                .bucket(ROOT_BUCKET)
                .objects(objectsToDelete)
                .build());

        for (Result<DeleteError> result : results) {
            DeleteError deleteError = result.get();
            LOGGER.debug("Error deleting object {}: {}",
deleteError.objectName(), deleteError.message());
        }
    } catch (Exception exc) {
        throw new MinioOperationException(exc);
    }
}

// returned stream must be closed in order to release
network resources

@Override

public InputStream getFileContent(ObjectPath filePath) {
    validateFilePath(filePath);

    try {
        return client.getObject(
            GetObjectArgs.builder()

```

				ПГУ1.090301.12.001 ПЗ	Лист
					84
Изм.	Лист	№ докум	Подпись Дата		

```

        .bucket(ROOT_BUCKET)
        .object(filePath.getFullPath())
        .build()

    );
} catch (Exception exc) {
    throw new MinioOperationException(exc);
}
}
}

```

Файл FileRepository.java

```

public interface FileRepository {

    boolean isObjectExist(ObjectPath objectPath);

    List<ObjectInfo> listFolderObjects(ObjectPath folderPath);

    List<ObjectInfo> listFolderObjectsRecursive(ObjectPath
folderPath, boolean includeSelf);

    void createFile(ObjectPath filePath, InputStream
fileInputStream);

    void createFolder(ObjectPath folderPath);

    void moveFile(ObjectPath oldFilePath, ObjectPath
newFilePath);

    void moveFolder(ObjectPath oldFolderPath, ObjectPath
newFolderPath);

    void deleteFile(ObjectPath filePath);

    void deleteFolder(ObjectPath folderPath);
}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					85
Изм.	Лист	№ докум	Подпись Дата		

```

        InputStream getFileContent(ObjectPath filePath);
    }

Файл ObjectPath.java

public interface ObjectPath {

    String getObject_name();

    String getPath();

    String getFullPath();

    ObjectPath getParent();

    boolean isFolder();

    boolean isRootFolder();

    boolean isInFolder(ObjectPath folderPath);

    ObjectPath replacePrefix(String prefixToReplace, String replacement);

    ObjectPath renameObject(String newName);

    ObjectPath resolve(String objectName);
}

```

Файл ObjectPathFactory.java

```

public interface ObjectPathFactory {

    ObjectPath compose(int userId, String objectPath);

    default ObjectPath getRootFolder(int userId) {
        return compose(userId, "/");
    }

}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					86
Изм.	Лист	№ докум	Подпись Дата		

Файл UserRepository.java

```
@Repository

public interface UserRepository extends JpaRepository<User,
Integer> {

    Optional<User> findByUsername(String name);

    boolean existsByUsername(String username);

}
```

Файл FileService.java

```
@Service

@RequiredArgsConstructor

public class FileService {

    private final FileRepository fileRepository;

    private final ObjectPathFactory objectPathFactory;

    public void validateFileExists(int userId, String path) {

        ObjectPath filePath = objectPathFactory.compose(userId,
fromUrlParam(path, true));

        if (!fileRepository.isObjectExist(filePath)) {

            throw new
ObjectNotExistsException(String.format("File %s not exists",
filePath));

        }

    }

    public void saveFiles(int userId, FilesUploadDto
filesUploadDto) {

        List<MultipartFile> files = filesUploadDto.GetFiles();

        if (files == null || files.isEmpty()) {

            throw new ObjectUploadException("No files were
provided");

        }

    }

}
```

				ПГУ1.090301.12.001 ПЗ	Лист
					87
Изм.	Лист	№ докум	Подпись Дата		

```

        ObjectPath parentFolderPath =
objectPathFactory.compose(userId,
fromUrlParam(filesUploadDto.getParentFolderPath()));

        for (MultipartFile file : files) {

            try {

                ObjectPath newFilePath =
parentFolderPath.resolve(file.getOriginalFilename());

                saveFile(newFilePath, file.getInputStream());

            } catch (IOException exc) {

                throw new ObjectUploadException("An error
occurred during uploading file to " + parentFolderPath, exc);

            }

        }

    }

    public void saveFile(ObjectPath filePath, InputStream
fileInputStream) {

        if (filePath.isFolder()) {

            throw new IllegalArgumentException("Ehe object being
saved is not a file: " + filePath);

        }

        if (fileRepository.isObjectExist(filePath)) {

            throw new ObjectAlreadyExistsException(

                String.format("File with name %s already
exists", filePath.getFullPath()));

        }

        fileRepository.createFile(filePath, fileInputStream);

    }

    public InputStream getFileContent(int userId, String path) {

        ObjectPath filePath = objectPathFactory.compose(userId,
fromUrlParam(path, true));

        if (!fileRepository.isObjectExist(filePath)) {

```

```

        throw new
ObjectNotExistsException(String.format("File %s not exists",
filePath));

    }

    return fileRepository.getFileContent(filePath);

}

    public String renameFile(int userId, ObjectRenameDto
objectRenameDto) {

        ObjectPath oldFilePath =
objectPathFactory.compose(userId,

            fromUrlParam(objectRenameDto.getObjectPath(),
true));

        ObjectPath newFilePath =
oldFilePath.renameObject(objectRenameDto.getNewName());

        if (fileRepository.isObjectExist(newFilePath)) {

            throw new ObjectAlreadyExistsException(

                String.format("File with name %s already
exists", newFilePath));

        }

        fileRepository.moveFile(oldFilePath, newFilePath);

        return toUrlParam(oldFilePath.getParent().getPath());

    }

    public List<String> getMoveCandidatesForFile(int userId,
String path) {

        ObjectPath filePathToMove =
objectPathFactory.compose(userId, fromUrlParam(path, true));

        if (!fileRepository.isObjectExist(filePathToMove)) {

            throw new ObjectNotExistsException(

                String.format("File with name %s does not
exist", filePathToMove));

        }

        ObjectPath parentFolderPath =
filePathToMove.getParent();

```

				ПГУ1.090301.12.001 ПЗ	Лист
					89
Изм.	Лист	№ докум	Подпись Дата		

```

        ObjectPath rootFolderPath =
objectPathFactory.getRootFolder(userId);

        return
fileRepository.listFolderObjectsRecursive(rootFolderPath,
true).stream()

                .map(ObjectInfo::getObjectPath)

                .filter(objectPath -> objectPath.isFolder() &&
!objectPath.equals(parentFolderPath))

                .map(folderPath ->
toUrlParam(folderPath.getPath()))

                .toList();
    }

    public String moveFile(int userId, ObjectMoveDto
objectMoveDto) {

        ObjectPath oldFilePath =
objectPathFactory.compose(userId,

                fromUrlParam(objectMoveDto.getOldObjectPath(),
true));

        ObjectPath newFolderPath =
objectPathFactory.compose(userId,

                fromUrlParam(objectMoveDto.getNewObjectPath()));

        ObjectPath newFilePath =
newFolderPath.resolve(oldFilePath.getObjectName());

        if (fileRepository.isObjectExist(newFilePath)) {

            throw new ObjectAlreadyExistsException(

                    String.format("Cannot move file, because
target file with name %s already exists", newFilePath));

        }

        fileRepository.moveFile(oldFilePath, newFilePath);

        return toUrlParam(oldFilePath.getParent().getPath());

    }

    public String deleteFile(int userId, String path) {

        ObjectPath filePathToDelete =
objectPathFactory.compose(userId, fromUrlParam(path, true));

```

				ПГУ1.090301.12.001 ПЗ	Лист
					90
Изм.	Лист	№ докум	Подпись Дата		

```

        fileRepository.deleteFile(filePathToDelete);

        String parentFolderPath =
filePathToDelete.getParent().getPath();

        return toUrlParam(parentFolderPath);

    }

}

```

Файл FolderService.java

```

@Service
@RequiredArgsConstructor
public class FolderService {

    private final FileRepository fileRepository;

    private final FileService fileService;

    private final ObjectPathFactory objectPathFactory;

    public void validateFolderExists(int userId, String path) {

        ObjectPath folderPath =
objectPathFactory.compose(userId, fromUrlParam(path));

        if (!fileRepository.isObjectExist(folderPath)) {

            throw new ObjectNotExistsException(

                String.format("Folder with name %s does not
exist", folderPath));

        }

    }

    public List<ObjectInfoDto> getFolderObjects(int userId,
String path) {

        ObjectPath folderPath =
objectPathFactory.compose(userId, fromUrlParam(path));

        if (!fileRepository.isObjectExist(folderPath)) {

            throw new ObjectNotExistsException(

                String.format("Folder with name %s does not
exist", folderPath));

        }

    }

}

```

				ПГУ1.090301.12.001 ПЗ	Лист
					91
Изм.	Лист	№ докум	Подпись Дата		

```

        return
fileRepository.listFolderObjects(folderPath).stream()
                .map(ObjectInfoMapper::toDtoForList)

.sorted(Comparator.comparing(ObjectInfoDto::isFolder).reversed()
.thenComparing(ObjectInfoDto::getName))

                .toList();

    }

    public void saveFolder(int userId, FilesUploadDto
filesUploadDto) {

        List<MultipartFile> files = filesUploadDto.GetFiles();

        if (files.size() == 0) {
            return;
        }

        ObjectPath parentFolderPath = objectPathFactory.compose(
                userId,
fromUrlParam(filesUploadDto.getParentFolderPath()));

        String uploadedFolderName = extractFirstFolderName(
Objects.requireNonNull(files.get(0).getOriginalFilename()));

        if
(fileRepository.isObjectExist(parentFolderPath.resolve(uploadedF
olderName + "/"))) {
            throw new ObjectAlreadyExistsException(
                String.format("Cannot upload folder %s,
because folder with this name already exists",
uploadedFolderName));
        }

        for (MultipartFile file : files) {

```

				ПГУ1.090301.12.001 ПЗ	Лист
					92
Изм.	Лист	№ докум	Подпись Дата		

```

        ObjectPath filePath =
parentFolderPath.resolve(file.getOriginalFilename());

        if (filePath.getObjectName().startsWith(".")) { //
exclude hidden files

            continue;

        }

        try {

            fileService.saveFile(filePath,
file.getInputStream());

createFolderWithIntermediate(filePath.getParent());

        } catch (IOException exc) {

            throw new ObjectUploadException("An error
occurred during uploading folder to " + parentFolderPath, exc);

        }

    }

}

public void createSingleFolder(int userId, FolderCreationDto
folderCreationDto) {

    String parentFolder =
fromUrlParam(folderCreationDto.getParentFolderPath());

    ObjectPath newFolderPath =
objectPathFactory.compose(userId, parentFolder)

        .resolve(folderCreationDto.getNewFolderName() +
"/");

    if (fileRepository.isObjectExist(newFolderPath)) {

        throw new ObjectAlreadyExistsException(

            String.format("Folder with name %s already
exists", newFolderPath));

    }

    fileRepository.createFolder(newFolderPath);

}

public void createFolderWithIntermediate(ObjectPath path) {

```

				ПГУ1.090301.12.001 ПЗ	Лист
					93
Изм.	Лист	№ докум	Подпись Дата		

```

        if (!path.isFolder()) {
            throw new IllegalArgumentException("Saved object is
not a folder: " + path);
        }
        while (!path.isRootFolder()) {
            if (fileRepository.isObjectExist(path)) {
                break;
            }
            fileRepository.createFolder(path);
            path = path.getParent();
        }
    }

    public String moveFolder(int userId, ObjectMoveDto
objectMoveDto) {
        ObjectPath oldFolderPath =
objectPathFactory.compose(userId,
            fromUrlParam(objectMoveDto.getOldObjectPath()));
        ObjectPath newParentFolderPath =
objectPathFactory.compose(userId,
            fromUrlParam(objectMoveDto.getNewObjectPath()));
        ObjectPath newFolderPath =
newParentFolderPath.resolve(oldFolderPath.getObjectName() +
"/");
        if (fileRepository.isObjectExist(newFolderPath)) {
            throw new ObjectAlreadyExistsException(
                String.format("Cannot move folder, because
target folder with name %s already exists", newFolderPath));
        }
        fileRepository.moveFolder(oldFolderPath, newFolderPath);
        return toUrlParam(oldFolderPath.getParent().getPath());
    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					94
Изм.	Лист	№ докум	Подпись Дата		

```

    public String renameFolder(int userId, ObjectRenameDto
objectRenameDto) {

        ObjectPath oldFolderPath =
objectPathFactory.compose(userId,
fromUrlParam(objectRenameDto.getObjectPath()));

        ObjectPath newFolderPath =
oldFolderPath.renameObject(objectRenameDto.getNewName());

        if (fileRepository.isObjectExist(newFolderPath)) {

            throw new ObjectAlreadyExistsException(

                String.format("Cannot rename folder, because
target folder with name %s already exists", newFolderPath));

        }

        fileRepository.moveFolder(oldFolderPath, newFolderPath);

        return toUrlParam(oldFolderPath.getParent().getPath());

    }

```

				ПГУ1.090301.12.001 ПЗ	Лист
					95
Изм.	Лист	№ докум	Подпись Дата		

Приложение В

(обязательное)

Облачный сервис хранения файлов

Презентация

				ПГУ1.090301.12.001 ПЗ	Лист
					96
Изм.	Лист	№ докум	Подпись Дата		

Тема работы: Облачный сервис хранения файлов

Автор работы: студент группы 20BV1 Тюгаев К.А.
Научный руководитель: к.т.н., доцент каф. ВТ ПГУ Токарев А. Н.

Пенза 2024

Актуальность работы

Способы хранения персональной информации прошли большой путь: от дискотек и цифровых дисков с крайне ограниченным объемом памяти до современных SDD накопителей с огромной скоростью чтения и записи и при этом крайне большим объемом, порой более одного терабайта. Однако, помимо этого всё большую популярность набирают удалённые способы хранения информации – облачные хранилища.

Развитие данных сервисов позволяет людям экономить память на собственном компьютере, используя больше мест под различное программное обеспечение, а не простые файлы с данными, такими как музыка или word документы, а также иметь доступ к своим файлам из любой точки мира с любого устройства, имеющего выход в интернет, редактировать их в любой удобный момент, а также крайне легко и просто делиться ими с другими пользователями.

Вышеописанное демонстрирует нам актуальность выбранного в рамках выпускной квалификационной работы проекта.

Изм.	Лист	№ докум	Подпись Дата

Предмет и объект работы

Предмет

- Разработка облачного сервиса хранения файлов

Объект

- Разработка клиент – серверного web – приложения

				ПГУ1.090301.12.001 ПЗ	Лист
					99
Изм.	Лист	№ докум	Подпись Дата		

Цель работы

Цель работы – разработать облачный сервис хранения файлов.

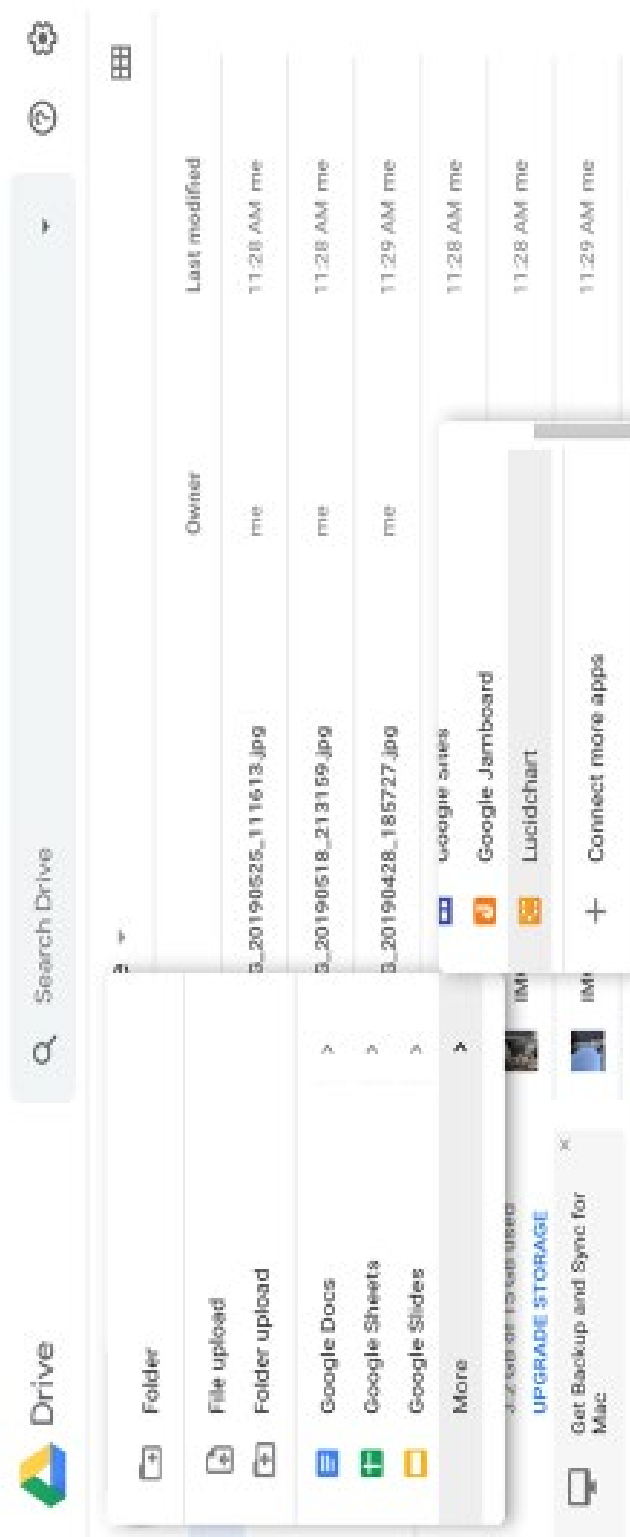
Данное приложение позволит пользователю регистрироваться и авторизоваться в приложении, создавать папки, загружать файлы и папки, сохранять на свой ПК, переименовывать их, производить поиск по файлам и папкам, а также удалять их

Постановка задачи

Данное приложение должно быть спроектировано на основе модели клиент – серверной архитектуры с использованием паттернов *MVC* и *REST*.

Клиентская часть должна обладать визуальным графическим интерфейсом, а серверная должна обрабатывать запросы пользователя и работать с базой данных

Аналоги Google Drive



Изм.	Лист	№ докум	Подпись Дата

ПГУ1.090301.12.001 ПЗ

Лист

102

Стек технологий

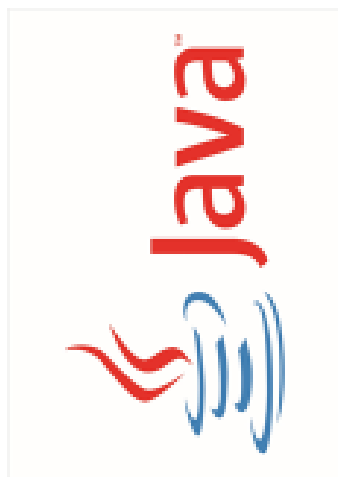
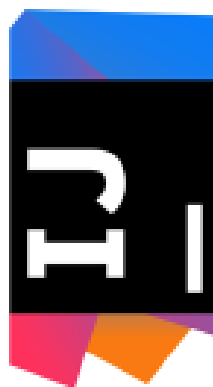
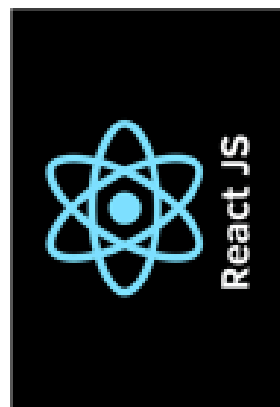
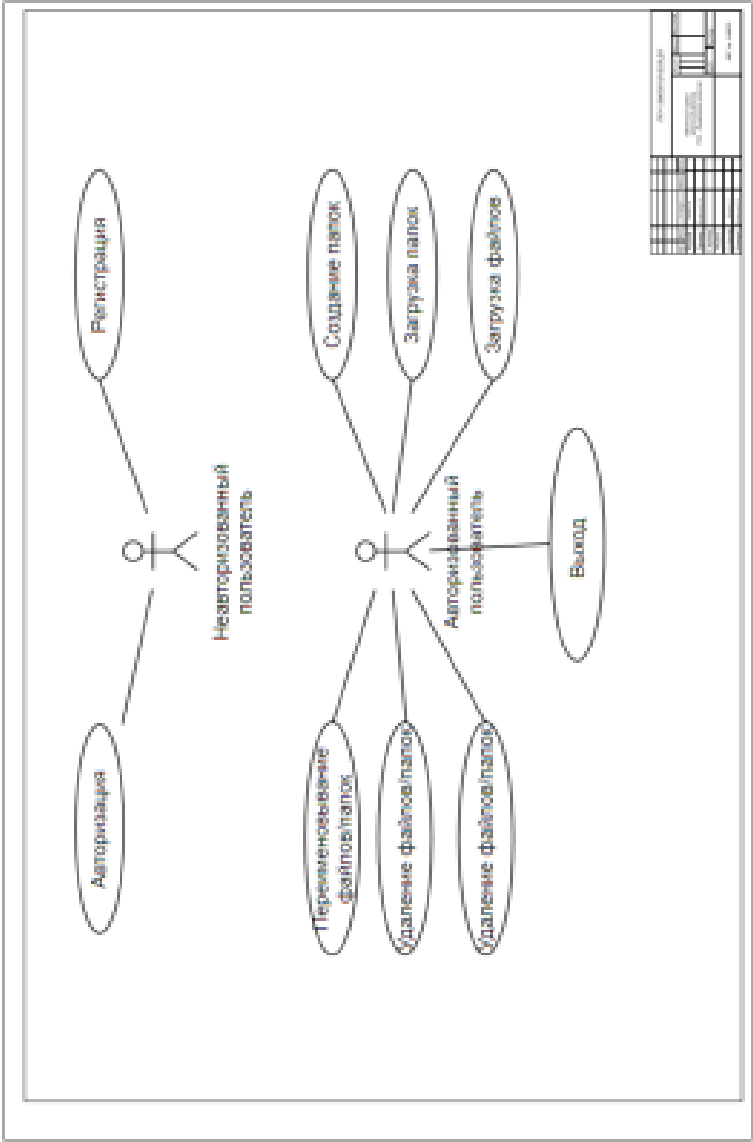
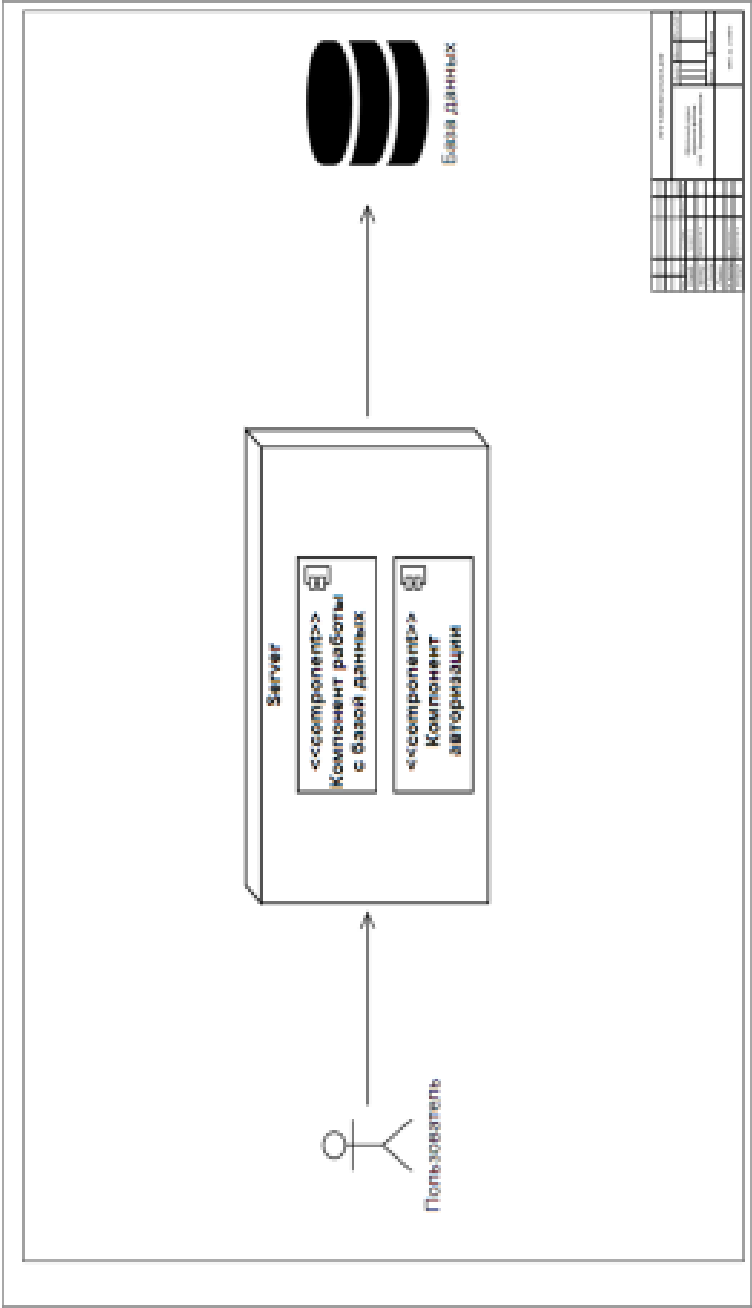


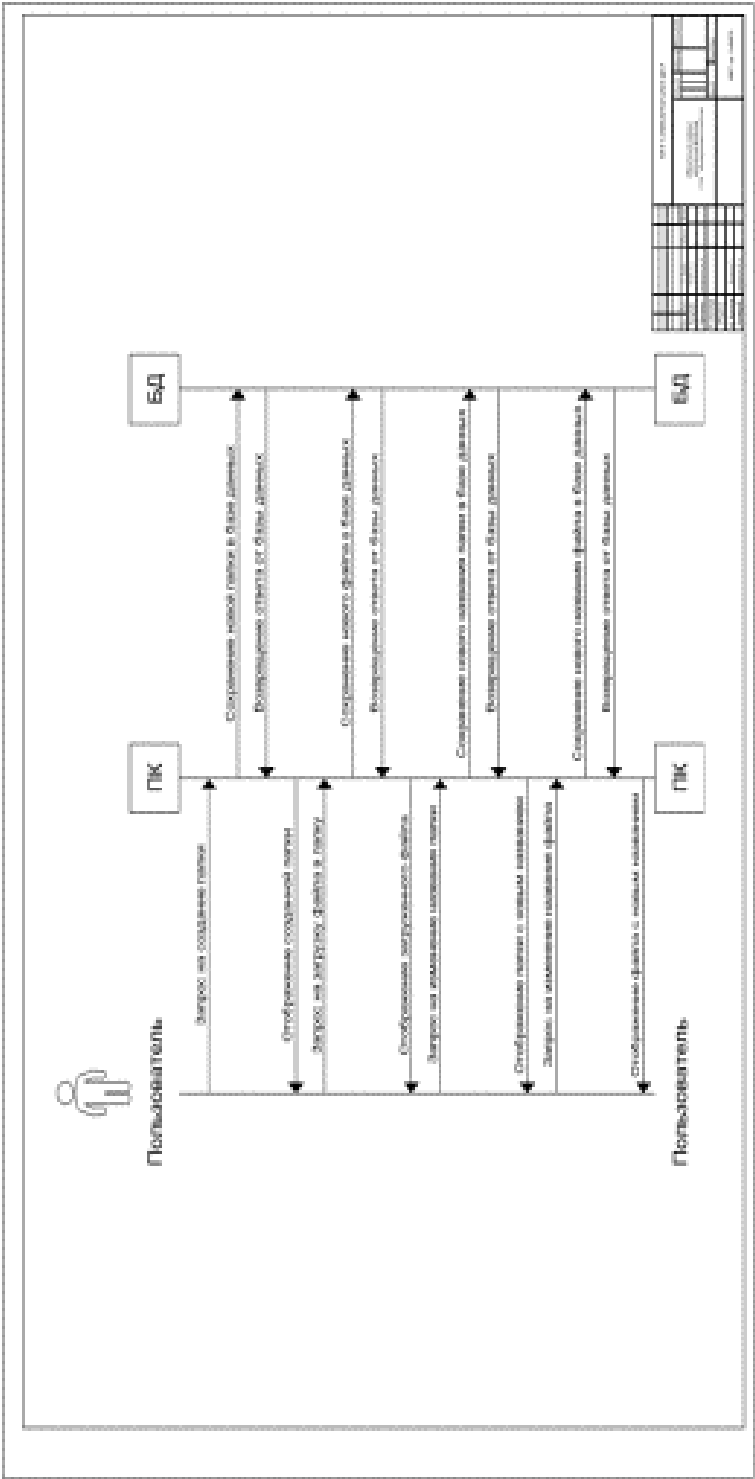
Диаграмма вариантов использования



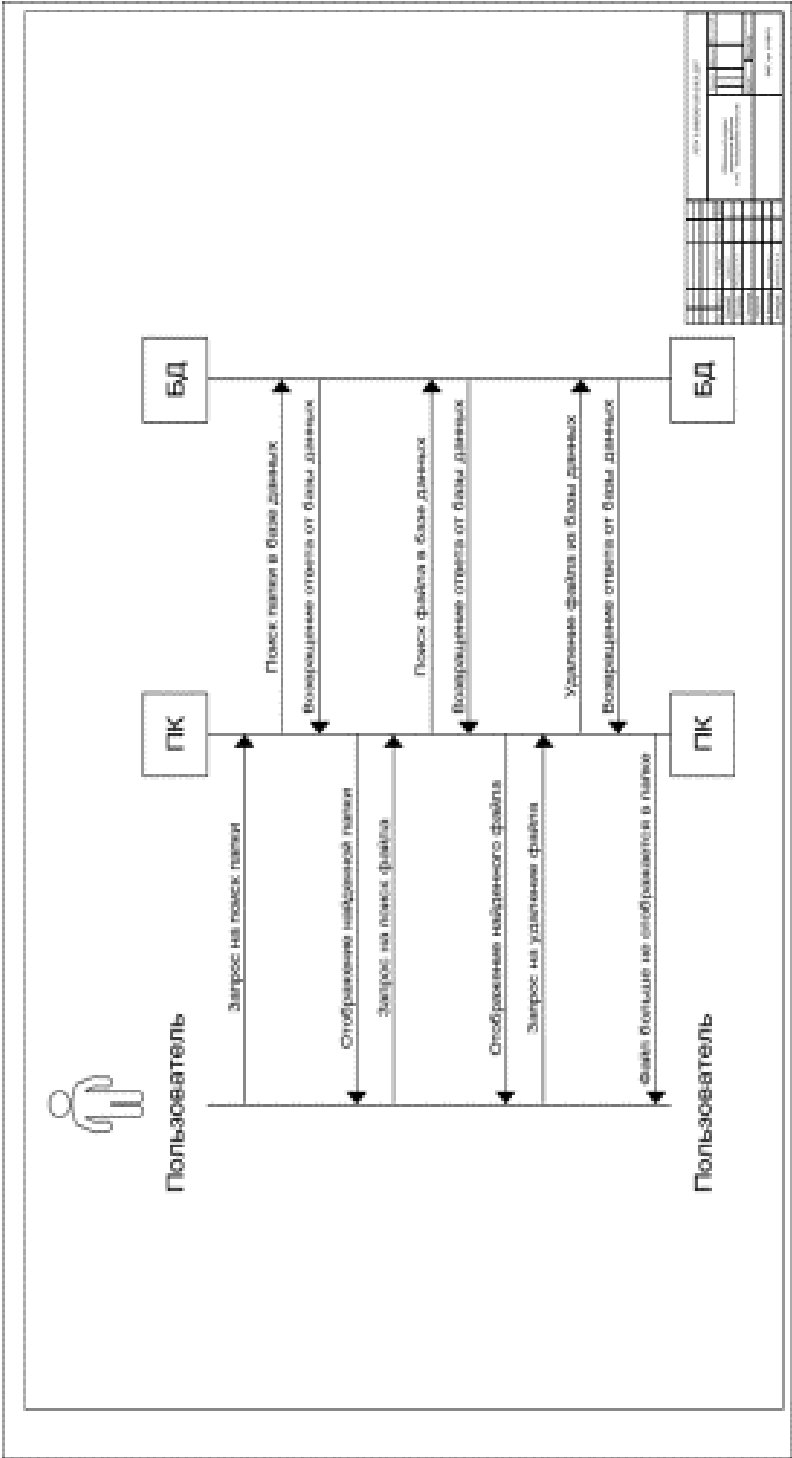
Диаграммы развертывания



Диаграммы вариантов использования



Диаграммы вариантов использования



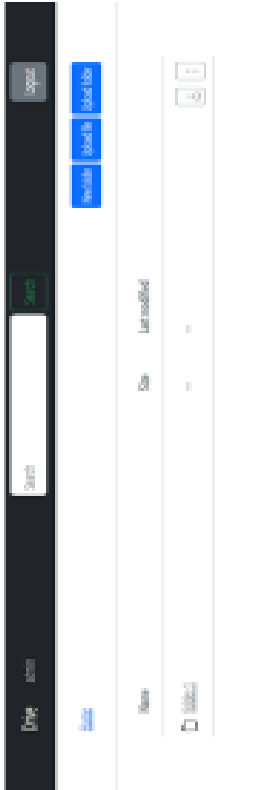
Демонстрация работы



Окно регистрации и
авторизации

Главная страница приложения

Демонстрация работы



Новая папка, созданная в приложении



Файл, загруженный в созданную папку

Демонстрация работы

Print

action

Logout

and file - J

Search

Search results

Home

/

Index

/

Index

2

New title

Update title

Update title

Update title

This folder is empty

Созданный ранее файл успешно найден в приложении

Созданный файл успешно удален из папки

				ПГУ1.090301.12.001 ПЗ	Лист
					110
Изм.	Лист	№ докум	Подпись Дата		

Заключение

- В результате написания дипломной работы был произведён анализ выбранной темы, включающий в себя изучение аналогов разрабатываемого продукта, выбор инструментов для разработки и их применение на практике, что позволило достичь финального результата – разработанного и протестированного сервиса для облачного хранения файлов.
- В процессе разработки были закреплены навыки работы с языком Java, было изучено многообразие экосистемы фреймворка Spring, изучен и закреплён на практике путём написания приложения архитектурный шаблон MVC, также изучена СУБД PostgreSQL путем работы с ней через её графический клиент pgAdmin4.
- Дальнейшее улучшение приложения может включать возможность редактирования файлов в режиме многозадачности онлайн одновременно несколькими пользователями, а также возможность введения платной подписки для закупки большего количества устройств для хранения данных и расширения тем самым доступного места для хранения файлов пользователей.

Изм.	Лист	№ докум	Подпись Дата

Спасибо за внимание!

				ПГУ1.090301.12.001 ПЗ	Лист
					112
Изм.	Лист	№ докум	Подпись Дата		