

Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский технический университет связи и информатики»

Разрешаю
допустить к защите
Зав. кафедрой

_____ 20 ____ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

НА ТЕМУ

Разработка файловой системы хранения данных

Студент: Богданов Д.О.

Руководитель: Маклачкова В.В.

Москва 2023 г.

Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский технический университет связи и информатики»

Кафедра

Сетевых информационных технологий и сервисов

(название полностью)

«Утверждаю»

Зав. кафедрой

« »

2023 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студенту Богданову Дмитрию Олеговичу гр. БСУ1901

Направление (специальность) Инфокоммуникационные технологии и системы связи
11.03.02

Форма выполнения выпускной квалификационной работы Бакалаврская работа

(Дипломный проект, дипломная работа, магистерская диссертация, бакалаврская работа)

Тема выпускной квалификационной работы Разработка файловой системы хранения данных

Утверждена приказом ректора № 97-С от 26.01 20 23 г.

1. Исходные данные
Документация Docker
Документация Nginx
Документация PostgreSQL
Документация Node.js

2. Содержание расчетно-пояснительной записки
(перечень подлежащих разработке вопросов)

Введение
1. Описание предметной области
2. Выбор средств разработки
3. Проектирование приложения
4. Реализация приложения
Заключение

Объем работы в % и сроки
выполнения по разделам

5 %	15.03.23
25 %	20.03.23
20 %	25.04.23
20 %	04.05.23
25 %	15.05.23
5 %	29.05.23

3. Вопросы конструктивных разработок

4. Разработка вопросов по экологии и безопасности жизнедеятельности

5. Техничко-экономическое обоснование (подлежащее расчету)

6. Перечень графического материала (с точным указанием обязательных чертежей)

1. Требования к приложению
2. Средства и технологии разработки
3. Логическая архитектура приложения
4. Системная архитектура приложения
5. Концептуальная модель БД
6. Инфологическая модель БД
7. Даталогическая модель БД
8. Структура веб-интерфейса
9. Блок-схема алгоритма авторизации
10. Таблица тестирования приложения

7. Консультанты по ВКР (с указанием относящихся к ним разделов проекта):

(подпись)

(ФИО)

(подпись)

(ФИО)

8. Срок сдачи студентом законченной ВКР: 31.05.2023

Дата выдачи задания: 21.02.2023

Руководитель


(подпись)

Маклачкова Виктория Валентиновна
(ФИО)

почасовая

(штатная или почасовая)

нагрузка

Задание принял к исполнению


(подпись студента)

Аннотация

Вид работы.

Выпускная квалификационная работа бакалавра по направлению 11.03.02 «Инфокоммуникационные технологии и системы связи». Профиль – «Инфокоммуникационные технологии в сервисах и услугах связи».

Тема работы.

«Разработка файловой системы хранения данных».

Область применения.

Отрасль инфокоммуникаций.

Цель работы.

Разработка приложения для хранения данных на сервере.

Краткое содержание.

ВКР состоит из введения, четырех разделов, заключения, списка использованных источников и трех приложений.

Первый раздел посвящен описанию предметной области, формулируются требования к разрабатываемому приложению.

Во второй разделе дается описание средств разработки приложения.

В третьем разделе проектируется приложение: разрабатывается архитектура, проектируется база данных, разрабатываются макеты, описываются средства обеспечения безопасности приложения

Четвёртый раздел посвящен реализации и тестированию приложения. В работе разработано два документа: руководство по установке приложения и руководство пользователя.

Работа содержит 74 страницы текста, 27 рисунков, 6 таблиц, 12 источников.

Содержание

Введение.....	7
1. Описание предметной области	9
1.1. Система хранения данных.....	9
1.2. Отличие файловой системы хранения данных от систем файлообмена..	11
1.3. Анализ существующих схожих решений	13
1.3.1. Битрикс 24.....	14
1.3.2. Docsvision.....	14
1.3.3. OwnCloud	15
1.3.4. Pydio.....	16
1.3.5. Сравнение решений	16
1.4. Обоснования необходимости разработки приложения.....	18
1.5. Определение основных пользователей	19
1.6. Описание функциональных требований к приложению	20
1.7. Ограничения приложения	21
1.8. Программно-аппаратные требования к системе.....	22
1.9. Выводы по первому разделу	23
2. Выбор средств разработки	24
2.1. Выбор языка программирования.....	24
2.2. Выбор системы управления базами данных	27
2.3. Выбор обратного прокси-сервиса	30
2.4. Выбор технологий для упрощения разработки и развёртывания	33
2.5. Выводы по второму разделу	35
3. Проектирование приложения.....	36
3.1. Выбор архитектуры.....	36
3.2. Проектирование базы данных.....	40
3.3. Определение ролей пользователей.....	46

3.4. Разработка макетов	47
3.5. Обеспечение безопасности приложения.....	48
3.6. Выводы по третьему разделу	50
4. Реализация приложения	51
4.1. Структура веб-интерфейса	51
4.2. Описание функциональности приложения	52
4.3. Процесс разработки	54
4.4. Алгоритм авторизации	57
4.5. Создание основных окон приложения.....	59
4.6. Тестирование приложения	62
4.7. Руководство по установке приложения.....	64
4.8. Руководство пользователя.....	66
4.9. Выводы по четвертому разделу	67
Заключение	68
Список использованных источников	69
Приложение А. Настройка Docker	70
Приложение Б. Бок-схема алгоритма авторизации.....	73
Приложение В. Результаты тестирования приложения.....	74

Введение

Приложения для хранения файлов на сервере в настоящее время являются актуальными и востребованными, так как они позволяют пользователям хранить свои данные на сервере и иметь к ним доступ из любой точки мира с помощью интернета.

Более того, сейчас все больше компаний и организаций переходят на удаленную работу, что делает приложения для хранения файлов на сервере еще более актуальными.

Кроме того, важным аспектом является безопасность хранения и передачи данных, что достигается с помощью различных механизмов защиты информации, таких как шифрование данных, защита паролем, контроль доступа, аудит и мониторинг действий пользователей. Это обеспечивает конфиденциальность и надежность хранения данных пользователей и компаний.

Таким образом, разработка приложения для хранения файлов на сервере является актуальной и важной задачей в современном мире, и предоставляет множество возможностей для оптимизации бизнес-процессов и улучшения взаимодействия между пользователями.

Целью работы является разработка приложения для хранения файлов на сервере, которая будет предоставлять пользователям возможность подключаться через веб-интерфейс, загружать и скачивать файлы. Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Изучить существующие программные решения для хранения файлов на сервере.
- 2) Выбрать наиболее подходящий под задачи приложения инструментарий и технологии для разработки.
- 3) Разработать архитектуру приложения.
- 4) Разработать веб-интерфейс.

- 5) Реализовать функционал приложения, включая загрузку и скачивание файлов, создание и удаление папок, управление правами доступа пользователей.
- 6) Протестировать приложение на соответствие заданным требованиям.

Объект исследования – система хранения данных

Предмет исследования – приложение для организации и управления файловой системой хранения данных.

ВКР состоит из введения, четырех разделов, заключения, списка использованных источников и трех приложений.

В первом разделе анализируются системы хранения данных. Определяются существующие готовые решения на рынке, и проводится анализ их преимуществ и недостатков. Также в этом разделе определяются основные требования к приложению, которые необходимо учесть при ее разработке.

Во втором разделе выбираются средства разработки для приложения. Проводится анализ различных языков программирования, баз данных, инструментов контроля версий, и других необходимых технологий. Обосновываются выбранные технологии и их преимущества, а также описываются требования проекта.

В третьем разделе проектируется приложение. Выбирается подходящая архитектура, разрабатывается структура базы данных, создаются макеты основных окон приложения, а также описываются средства обеспечения безопасности. Это поможет создать четкую и эффективную структуру приложения.

В четвертом разделе разрабатывается приложение. Разрабатываются функциональности приложения, проводится тестирование, создается руководство по установке и руководство пользователя.

1. Описание предметной области

1.1. Система хранения данных

Система хранения данных (СХД) — это специализированная инфраструктура или программное обеспечение, предназначенное для хранения и управления данными. Они обеспечивают высокую производительность, надежность и масштабируемость для хранения больших объемов данных [1].

Существуют различные типы СХД, включая:

Блочные СХД — это системы хранения данных, которые работают на уровне блоков данных. Они предоставляют доступ к данным в виде блоков фиксированного размера и часто используются для хранения и обработки структурированных данных.

Преимущества:

- Высокая производительность и низкая задержка доступа к данным благодаря прямому доступу к блокам данных.
- Гибкость в работе с различными типами данных и приложениями.
- Возможность конфигурирования и масштабирования системы для обеспечения высокой доступности и отказоустойчивости.

Недостатки:

- Требуется более высокий уровень абстракции и программирования для работы с блоками данных.
- Требуется более сложная настройка и управление системой для обеспечения надежности и безопасности данных.
- Могут возникать проблемы с фрагментацией данных и эффективным использованием пространства.

Файловые СХД — это системы, которые организуют данные в виде файловой структуры. Они предоставляют доступ к данным в виде файлов и поддерживают операции чтения и записи. Файловые СХД обычно используются

для хранения и обработки различных типов файлов, таких как документы, изображения, видео и т.д.

Преимущества:

- Простота в использовании и понимании, так как данные организованы в виде файлов и каталогов.
- Хорошая поддержка множества приложений, особенно тех, которые работают с большими файлами.
- Хорошая поддержка сетевого доступа к данным.

Недостатки:

- Низкая гибкость в работе с различными типами данных, так как файловые системы предназначены преимущественно для хранения файлов.
- Ограниченные возможности в масштабировании и распределении данных.
- Проблемы с конкурентным доступом к файлам, особенно в случае множественного доступа.

Объектные СХД — это системы, которые организуют данные в виде объектов, каждый из которых содержит как данные, так и метаданные о данных. Они широко используются для хранения и управления неструктурированными данными, такими как медиафайлы, веб-контент, архивы и другие.

Преимущества:

- Высокая гибкость и расширяемость в работе с различными типами данных, так как данные хранятся в виде объектов с метаданными.
- Поддержка более сложных структур данных и связей между объектами.
- Хорошая поддержка встроенных механизмов безопасности и контроля доступа.

Недостатки:

- Более сложная разработка и программирование для работы с объектами и их метаданными.

- Ограниченная поддержка существующих приложений и инструментов, особенно тех, которые ожидают использование файловой или блочной системы хранения данных.
- Ограниченные возможности в масштабировании и распределении данных.

Выбор между блочными, файловыми и объектными СХД зависит от требований конкретного приложения и его особенностей. Если приложение работает с большими файлами или требует прямого доступа к блокам данных, блочные СХД могут быть предпочтительны. Файловые СХД лучше подходят для приложений, работающих с файлами и требующих простоты использования. Объектные СХД хороши для приложений, которые требуют гибкости в работе с различными типами данных и более сложными структурами данных.

В рамках ВКР будет реализована файловая система хранения данных, так как она позволяет легко организовать файлы на сервере и быстро получать к ним доступ. Кроме того, файловая система не требует специальных знаний для управления и обеспечивает высокую производительность при чтении и записи файлов. Также файловая система имеет высокую надежность и устойчивость к сбоям, что важно при хранении данных на сервере.

1.2. Отличие файловой системы хранения данных от систем файлообмена

Системы файлообмена — это программные решения или платформы, которые предназначены для обмена файлами между пользователями или устройствами. Они предоставляют функциональность загрузки, скачивания, хранения и совместного использования файлов.

Системы файлообмена и системы хранения данных представляют разные подходы к обработке и управлению информацией. Вот сравнение между ними:

Системы файлообмена:

- Ориентированы на обмен и передачу файлов между пользователями или устройствами.
- Позволяют загружать, скачивать и обмениваться файлами.
- Часто основаны на облачных хранилищах или сетевых серверах.
- Предоставляют функции синхронизации файлов между устройствами и доступа к файлам через веб-интерфейс или клиентские приложения.
- Обычно предлагают удобный интерфейс для управления файлами, шифрования данных и контроля доступа.

Файловые системы хранения данных:

- Ориентированы на организацию и хранение структурированных данных в базах данных или хранилищах данных.
- Позволяют создавать, изменять и извлекать данные, используя специальные языки запросов или программные интерфейсы.
- Предлагают различные модели данных, такие как реляционная, документная, ключ-значение и другие, для хранения и обработки данных.
- Поддерживают функции индексирования, поиска, агрегации и анализа данных.

В отличие от систем файлообмена, файловые системы хранения данных предоставляют более продвинутые возможности для работы с структурированными данными, обеспечивая эффективное хранение, поиск и анализ информации. Они обычно применяются в бизнес-приложениях, где требуется организация и управление большим объемом данных. Системы файлообмена, с другой стороны, удобны для обмена файлами между пользователями и совместной работы над файлами в режиме реального времени.

Приложение, разрабатываемое в рамках ВКР, предоставляет возможность пользователям загружать файлы на сервер и хранить их в централизованном хранилище. Пользователи могут обращаться к своим файлам, скачивать их, редактировать и делиться с другими пользователями при необходимости.

Хотя данное приложение также обеспечивает функциональность обмена файлами, оно фокусируется преимущественно на хранении и управлении файлами, а не на реализации функций совместной работы и коллаборации, которые часто встречаются в полноценных системах файлообмена.

Таким образом, можно сказать, что разрабатываемое приложение ближе к файловой системе хранения данных, нежели к полноценной системе файлообмена.

1.3. Анализ существующих схожих решений

Некоторые из популярных российских решений, которые имеют схожий функционал разрабатываемым приложением:

- "Битрикс24" - платформа для управления бизнес-процессами, включающая веб-интерфейс и возможность хранения файлов на сервере. Она предоставляет средства для организации работы команды, управления проектами, обмена файлами и других бизнес-процессов.
- "Docsvision" - система электронного документооборота и управления документами, которая также предоставляет возможность хранить и управлять файлами на сервере. Она позволяет организовать хранение и поиск документов, контролировать доступ к ним, а также вести журнал истории изменений.
- "OwnCloud" - программное решение для создания собственного облачного хранилища данных. Оно предоставляет функции по загрузке, хранению и совместному использованию файлов, а также синхронизацию данных между различными устройствами.
- "Pydio" - открытое программное обеспечение для создания собственной файловой системы с веб-интерфейсом. Оно позволяет пользователям загружать, хранить и организовывать свои файлы на сервере, а также делиться ими с другими пользователями.

Упомянутые решения предлагают возможности хранения файлов на сервере, веб-интерфейс для управления файлами и совместной работы с ними. Однако, каждое из них имеет свои особенности и функциональные возможности, поэтому важно провести более детальное сравнение и анализ, чтобы определить, какое решение лучше всего подходит для конкретных потребностей и требований разрабатываемого приложения.

1.3.1. Битрикс 24

Битрикс24 представляет собой облачную платформу для управления бизнес-процессами. Она объединяет в себе широкий спектр функциональных возможностей, включая инструменты для управления проектами, коммуникации внутри команды, планирования задач, организации документооборота и многое другое [2].

Функциональность: Битрикс24 позволяет создавать рабочие группы, делиться файлами, общаться через чаты и видеозвонки, управлять задачами и проектами, а также вести учет клиентов и продаж.

Особенности: Платформа имеет расширяемую архитектуру, позволяющую добавлять дополнительные модули и интегрировать сторонние сервисы. Она также предоставляет API для разработки собственных приложений и интеграции с другими системами.

Преимущества: Удобный веб-интерфейс, широкий набор функций, гибкость настройки и возможность интеграции с другими сервисами.

1.3.2. Docsvision

Docsvision представляет собой систему управления документами и электронного документооборота. Она позволяет организовать хранение, поиск, редактирование и совместную работу с документами в компании [3].

Функциональность: В Docsvision можно создавать электронные документы, устанавливать права доступа к ним, назначать версии, контролировать жизненный цикл документов, автоматизировать рабочие процессы и т.д.

Особенности: Платформа предлагает различные модули и инструменты для управления документами, учета и контроля доступа. Она может быть настроена под нужды конкретной компании.

Преимущества: Высокий уровень безопасности данных, автоматизация рабочих процессов, возможность интеграции с другими системами, удобный поиск и фильтрация документов.

1.3.3. OwnCloud

OwnCloud является программным решением для создания собственного облачного хранилища данных. Оно позволяет пользователям загружать, хранить и совместно использовать файлы на сервере [4].

Функциональность: OwnCloud предлагает возможности для загрузки и скачивания файлов, организации папок, установки прав доступа, совместной работы над документами, синхронизации файлов между устройствами и т.д.

Особенности: Платформа имеет гибкую архитектуру и может быть установлена на собственном сервере или облачной инфраструктуре. Она поддерживает расширения и интеграцию с другими сервисами.

Преимущества: Полный контроль над данными, возможность создания собственного облачного хранилища, совместная работа над файлами, синхронизация с различными устройствами.

1.3.4. Pydio

Pydio (ранее известный как AjaXplorer) — это открытое программное обеспечение для создания собственной файловой системы с веб-интерфейсом. Оно позволяет загружать, хранить и организовывать файлы на сервере [5].

Функциональность: Pydio предлагает функции загрузки и скачивания файлов, создания и редактирования папок, установки прав доступа, совместной работы над документами, синхронизации данных и т.д.

Особенности: Платформа имеет гибкую архитектуру, расширяемую через плагины, и может быть установлена на сервер или в облачной среде. Она также поддерживает интеграцию с другими сервисами и API для разработки собственных приложений.

Преимущества: Открытое программное обеспечение, гибкость настройки, возможность создания собственного хранилища данных, совместная работа над файлами.

1.3.5. Сравнение решений

Давайте сравним функциональность рассматриваемых приложений:

Битрикс24:

- Управление проектами и задачами.
- Организация коммуникации и коллаборации внутри команды.
- Учет клиентов и продаж.
- Организация документооборота.
- Планирование встреч и совещаний.
- Аналитика и отчетность.
- Интеграция с другими сервисами и платформами.

Docsvision:

- Управление документами и электронный документооборот.

- Создание, редактирование и хранение электронных документов.
- Установка прав доступа к документам.
- Автоматизация рабочих процессов.
- Контроль жизненного цикла документов.
- Учет и контроль доступа к документам.
- Интеграция с другими системами и платформами.

OwnCloud:

- Загрузка и скачивание файлов.
- Организация хранения файлов и папок.
- Установка прав доступа к файлам.
- Совместная работа над файлами и документами.
- Синхронизация файлов между устройствами.
- Шифрование данных и защита информации.
- Интеграция с другими сервисами и приложениями.

Pydio:

- Загрузка и скачивание файлов.
- Создание и редактирование папок.
- Установка прав доступа к файлам.
- Совместная работа над файлами и документами.
- Синхронизация данных и файлов.
- Расширяемая архитектура через плагины.
- Интеграция с другими сервисами и платформами.

Каждое из этих приложений имеет свой уникальный набор функций, ориентированный на управление процессами, хранение и совместную работу с файлами и документами. Выбор наиболее подходящего решения будет зависеть от конкретных требований проекта, бюджета, предпочтений пользователя и интеграционных возможностей с другими системами и сервисами.

Более подробное сравнение представлено в таблице 1.3.1.

Таблица 1.3.1 – Сравнение готовых решений

Функциональные возможности	Битрикс24	Docsvision	OwnCloud	Pydio
Управление проектами и задачами.	+			
Учет клиентов и продаж.	+			
Организация документооборота.	+	+		
Планирование встреч и совещаний.	+			
Аналитика и отчетность.	+			
Интеграция с другими сервисами и платформами.	+	+	+	+
Установка прав доступа к документам.	+	+	+	+
Автоматизация рабочих процессов.		+		
Контроль жизненного цикла документов.		+		
Загрузка и скачивание файлов.	+	+	+	+
Установка прав доступа к файлам.	+		+	+
Совместная работа над файлами и документами.		+	+	+
Синхронизация файлов между устройствами.			+	+
Шифрование данных и защита информации.			+	
Создание и редактирование папок.			+	
Расширяемая архитектура через плагины.				+

1.4. Обоснования необходимости разработки приложения

Разработка приложения обусловлена несколькими основными обоснованиями:

Упрощение и автоматизация процесса хранения и управления файлами: приложение позволит пользователям загружать, хранить и организовывать свои файлы на сервере. Это позволит упростить процесс хранения данных, обеспечивая централизованное место для доступа к файлам.

Безопасность и контроль доступа: приложение обеспечит систему авторизации и управления доступом, где каждый пользователь будет иметь свой собственный аккаунт и доступ только к своим файлам. Также будет реализована система ролей, позволяющая определенным пользователям, таким как администраторы, просматривать историю действий других пользователей.

Низкие затраты на поддержку: клиенту предоставляется приложение с открытым исходным кодом, что позволит им самостоятельно поддерживать приложение.

Пользовательский интерфейс и удобство использования: разрабатываемое приложение будет обладать интуитивно понятным пользовательским интерфейсом, что обеспечит удобство использования для конечных пользователей. Они смогут легко загружать, просматривать и управлять своими файлами, не требуя специальных навыков или обучения.

Гибкость и расширяемость: при разработке приложения внимание его гибкости и расширяемости, чтобы в дальнейшем можно было добавлять новые функции и интегрировать его с другими системами или сервисами в соответствии с потребностями пользователей.

Будет создан инструмент для безопасного хранения и управления файлами.

1.5. Определение основных пользователей

Разрабатываемое приложение предназначено для организации хранения и управления файлами на сервере с использованием веб-интерфейса. Она подходит для любой организации, которая хочет предоставить своим

пользователям возможность загружать и хранить свои файлы на сервере, но при этом иметь контроль над доступом к этим файлам.

Приложение может использоваться различными пользователями, в зависимости от их потребностей и задач:

Бизнес-пользователи: компании и организации, которые нуждаются в безопасном хранении файлов. В частности, организации могут использовать это приложение для хранения документов и файлов внутри компании.

Индивидуальные пользователи: это приложение также может быть полезно для индивидуальных пользователей, которые хотят хранить файлы на сервере и иметь к ним доступ через интернет.

Администраторы: администраторы, которые отвечают за управление и поддержку системы, могут использовать приложение для управления пользователями и файлами, включая назначение ролей пользователям, мониторинг действий пользователей и поддержку безопасности.

Таким образом, разрабатываемое приложение может использоваться различными пользователями для различных целей и задач.

1.6. Описание функциональных требований к приложению

На основании сравнения существующих решений и учета потребностей пользователей, были определены следующие функциональные требования к разрабатываемому приложению:

Регистрация и авторизация:

- Возможность регистрации новых пользователей.
- Авторизация пользователей при входе в систему.

Управление пользователями:

- Создание и редактирование пользовательских аккаунтов.
- Назначение ролей и прав доступа для каждого пользователя.
- Управление паролями и восстановление доступа.

Загрузка и хранение файлов:

- Возможность загрузки файлов на сервер.
- Организация файлов по категориям или папкам.
- Поддержка различных типов файлов (документы, изображения, аудио и др.).
- Просмотр, скачивание и удаление файлов.

История действий:

- Ведение и хранение истории действий пользователей.
- Возможность просмотра и анализа истории действий для администраторов.

Поиск и фильтрация:

- Поиск файлов по названию, типу, дате загрузки и другим атрибутам.
- Фильтрация файлов по категориям, размеру и другим параметрам.

Безопасность:

- Защита файлов и данных пользователей от несанкционированного доступа.
- Шифрование передачи данных по сети.

Пользовательский интерфейс:

- Интуитивно понятный и привлекательный интерфейс.
- Легкая навигация по приложению.
- Адаптивный дизайн для поддержки различных устройств и экранов.

Эти функциональные требования позволят разрабатываемому приложению обеспечить удобное и безопасное хранение, управление файлами, улучшить коммуникацию и сотрудничество внутри организации и повысить эффективность работы с данными.

1.7. Ограничения приложения

В разрабатываемом приложении есть некоторые ограничения и функциональные ограничения, которые следует учесть. Вот некоторые из них:

1. Отсутствие поддержки синхронизации между устройствами: приложение не предоставляет механизм автоматической синхронизации данных между устройством пользователя и приложением.
2. Ограниченная работа в офлайн-режиме: приложение требует подключения к Интернету для полноценной работы.
3. Ограниченные возможности редактирования файлов: приложение предоставляет базовые возможности управления файлов, такие как создание, загрузка и удаление. Однако, продвинутые функции, такие как совместное редактирование или история изменений, могут быть ограничены или отсутствовать.
4. Отсутствие интеграции с другими сервисами: приложение не имеет интеграции с другими популярными сервисами, такими как облачные хранилища или социальные сети. Пользователь не сможет автоматически импортировать или экспортировать данные из/в другие сервисы.

Эти ограничения следует учитывать при использовании разрабатываемого приложения. Важно понимать его возможности и пределы, чтобы пользователи могли правильно использовать приложение в соответствии со своими потребностями.

1.8. Программно-аппаратные требования к приложению

Минимальные требования к серверу:

- Операционная система: Windows Server, Linux, Unix или Mac OS X.
- Процессор: минимум двухъядерный процессор, предпочтительно четырехъядерный или более.
- Оперативная память: минимум 2 ГБ, рекомендуется 8 ГБ или более.
- Жесткий диск: минимум 50 ГБ свободного пространства на жестком диске.
- Сетевое подключение: минимум 100 Мбит/с Ethernet, рекомендуется 1 Гбит/с.
- Дополнительные программы: для работы необходим docker.

Минимальные требования к устройству пользователя:

- Браузер с поддержкой HTML5, CSS3 и JavaScript. Рекомендуется использовать последние версии популярных браузеров, таких как Google Chrome, Mozilla Firefox, Safari или Microsoft Edge.
- Стабильное подключение к Интернету для обмена данными с сервером приложения.
- Клавиатура и мышь или другое устройство ввода для взаимодействия с интерфейсом приложения.

1.9. Выводы по первому разделу

1. Существуют различные типы систем хранения данных, включая блочные, файловые и объектные СХД. Было принято решение выбрать файловую СХД, так как она лучше соответствует требованиям приложения, позволяя эффективно хранить и управлять файлами на сервере.
2. Проведенное сравнение системы хранения данных с системой файлообмена показало, что система хранения данных предоставляет более гибкие возможности по хранению и управлению файлами, а также обеспечивает лучшую производительность и безопасность данных.
3. Анализ существующих схожих решений, таких как Битрикс24, Docsvision, OwnCloud, Pydio выявил проблемы существующих решений, такие как ограниченные функциональные возможности, сложность использования и неудовлетворительная производительность, что обосновывает необходимость разработки приложения.
4. На основе анализа были определены потенциальные пользователи разрабатываемого приложения, функциональные требования к приложению и программно-аппаратные требования к устройствам пользователей.

2. Выбор средств разработки

2.1. Выбор языка программирования

Выбор языка программирования является одним из ключевых решений при разработке программного обеспечения. Существует множество языков программирования, каждый со своими особенностями, преимуществами и областями применения. По данным GitHub был составлен рейтинг самых популярных языков среди разработчиков на 2023 год представленные на рисунке 2.1.1 [6].

# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	Python	17.199% (-0.923%)	
2	Java	11.258% (+0.151%)	^
3	C++	10.060% (+0.321%)	^
4	Go	9.859% (+1.129%)	^
5	JavaScript	9.373% (-1.766%)	v
6	TypeScript	8.270% (-0.078%)	
7	PHP	5.174% (-0.438%)	
8	Ruby	4.679% (-0.149%)	
9	C	4.338% (+0.077%)	
10	C#	3.395% (+0.298%)	

Рисунок 2.1.1 – Самые популярные языки программирования среди разработчиков по версии GitHub на 2023 год

Несмотря на то, что Python, Java, C++ и Go также являются самыми популярными языками программирования с широкими возможностями, они обладают другими особенностями и применяются в разных сферах. Python часто используется в научных и аналитических вычислениях, а также в разработке скриптов. Java широко применяется в разработке масштабных корпоративных приложений. C++ используется в системном программировании и разработке высокопроизводительных приложений.

В данном случае, JavaScript и TypeScript лучше соответствуют потребностям разработки веб-приложения и обладают богатой экосистемой инструментов и библиотек, что позволяет нам создать эффективное и гибкое веб-приложение.

TypeScript — это язык программирования, который является надмножеством JavaScript. С другой стороны, TypeScript компилируется в JavaScript, что позволяет запускать его на любой совместимой с JavaScript платформе или среде выполнения [7].

С точки зрения JavaScript, TypeScript предоставляет дополнительные возможности и функции, которых нет в чистом JavaScript. TypeScript включает в себя все функции JavaScript, позволяя разработчикам писать код на JavaScript, а также добавляет дополнительные функции и синтаксические возможности, которые делают разработку более эффективной и безопасной.

Некоторые из основных функций TypeScript, которые недоступны в JavaScript, включают:

1. Статическая типизация: TypeScript позволяет указывать типы переменных, параметров функций и возвращаемых значений, что обеспечивает раннюю обнаружение ошибок и повышает надежность кода.
2. Классы и модули: TypeScript поддерживает классы и модули, что делает код более организованным и позволяет использовать принципы объектно-ориентированного программирования.
3. Проверка типов на этапе компиляции: TypeScript выполняет проверку типов на этапе компиляции, что позволяет обнаруживать и исправлять ошибки до запуска программы. Это помогает улучшить качество и надежность кода.
4. Интеграция с инструментами разработки: TypeScript имеет широкую поддержку в популярных инструментах разработки, таких как Visual Studio Code, WebStorm и других, что облегчает разработку и отладку приложений.

5. Поддержка JavaScript-экосистемы: TypeScript совместим с существующей JavaScript-экосистемой, что позволяет использовать существующие библиотеки, фреймворки и инструменты разработки JavaScript.

Выбор TypeScript в качестве языка программирования для разработки приложения обусловлен его преимуществами по сравнению с чистым JavaScript. Статическая типизация, поддержка классов и модулей, проверка типов на этапе компиляции и другие возможности TypeScript помогают создавать более надежный и поддерживаемый код, повышают производительность разработчиков и улучшают опыт разработки приложений.

Для создания приложения, помимо языка программирования, необходимо выбрать библиотеки для создания графического интерфейса и серверной части приложения.

Для создания графического интерфейса была выбрана библиотека React. React позволяет разработчикам создавать модульные компоненты, которые затем могут быть повторно использованы в других частях приложения. React также использует виртуальный DOM для оптимизации производительности приложения.

Для создания серверной части приложения была выбрана Node.js — это среда выполнения JavaScript, которая позволяет запускать JavaScript на стороне сервера. Node.js предоставляет возможность разработчикам создавать высокопроизводительные приложения на основе JavaScript, используя множество встроенных модулей и библиотек.

В разрабатываемом приложении были выбраны TypeScript, React и Node.js, потому что они позволяют создавать высокопроизводительные, масштабируемые и структурированные веб-приложения. TypeScript позволяет нам написать более безопасный код и облегчает его поддержку в будущем. React позволяет нам создавать модульные компоненты, которые могут быть повторно использованы в разных частях приложения, что упрощает разработку и облегчает поддержку. Node.js позволяет нам создавать высокопроизводительные

приложения, которые могут обрабатывать множество запросов и пользователей одновременно.

Кроме того, использование этих технологий обеспечивает большое сообщество разработчиков, множество готовых библиотек и инструментов для разработки, а также легко интегрируются между собой.

2.2. Выбор системы управления базами данных

Система управления базами данных (СУБД) — это программное обеспечение, которое предоставляет удобный интерфейс для создания, управления и обработки баз данных. Она обеспечивает структурированное хранение данных, а также механизмы для выполнения запросов, обновления и извлечения информации.

Существует несколько типов СУБД, включая:

Реляционные СУБД (РСУБД): они основаны на модели реляционной базы данных и используют таблицы с отношениями между ними. Примеры таких СУБД: Oracle, MySQL, PostgreSQL, Microsoft SQL Server. РСУБД широко используются для хранения структурированных данных во многих приложениях.

Иерархические СУБД: они используют иерархическую модель данных, где данные организованы в виде древовидной структуры. Примеры иерархических СУБД: IBM's Information Management System (IMS), Integrated Data Store (IDS).

Сетевые СУБД: они основаны на сетевой модели данных, где связи между данными устанавливаются через сеть ссылок. Примеры сетевых СУБД: Integrated Data Store (IDS), Integrated Database Management System (IDMS).

Объектно-ориентированные СУБД (ООСУБД): они предоставляют возможность хранить и обрабатывать сложные объекты и структуры данных,

включая наследование и полиморфизм. Примеры ООСУБД: MongoDB, Couchbase, ObjectStore.

NoSQL СУБД: они предназначены для работы с большими объемами неструктурированных данных и обладают гибкой схемой. Примеры NoSQL СУБД: MongoDB, Cassandra, Redis.

Каждый тип СУБД имеет свои особенности и преимущества, и выбор определенного типа зависит от требований и характера проекта. Например, реляционные СУБД широко применяются в бизнес-приложениях, где требуется сильная структура данных и поддержка транзакций, в то время как NoSQL СУБД часто используются в приложениях Big Data и IoT, где необходимо обрабатывать и хранить большие объемы неструктурированных данных.

Реляционные СУБД были выбраны из-за их широкой распространенности и надежности. Они предлагают четкую структуру хранения данных в таблицах, что делает их удобными для анализа, поиска и манипулирования данными. Также, реляционные СУБД хорошо поддерживаются и имеют богатый выбор инструментов для работы с данными, что позволяет создавать сложные и масштабируемые приложения.

Среди реляционных СУБД был выбран PostgreSQL — это мощная и расширяемая реляционная база данных с открытым исходным кодом, которая используется для хранения, управления и обработки структурированных данных. PostgreSQL поддерживает многие функции реляционных баз данных, такие как ограничения целостности, транзакции, поддержка ключевых слов, подзапросы и многое другое. PostgreSQL поддерживает многопоточность и многопроцессорность, что обеспечивает высокую производительность и масштабируемость при обработке больших объемов данных [8].

Основные преимущества PostgreSQL:

- **Надежность и целостность данных:** PostgreSQL обеспечивает высокий уровень надежности данных, благодаря использованию механизма транзакций и поддержке ограничений целостности данных.

- **Расширяемость:** PostgreSQL позволяет расширять свои возможности через множество расширений, модулей и плагинов, которые можно использовать для создания собственных приложений.
- **Масштабируемость:** PostgreSQL может масштабироваться как вертикально, так и горизонтально. Это позволяет обрабатывать большие объемы данных с высокой производительностью и надежностью.
- **Открытый исходный код:** PostgreSQL имеет открытый исходный код, что позволяет разработчикам настраивать и модифицировать систему по своему усмотрению.

Сравнение с другими СУБД:

- **Oracle** - более мощная и функциональная СУБД, но требует больших затрат на лицензирование и обслуживание. PostgreSQL является хорошей альтернативой Oracle, предоставляя схожие возможности и функциональность, но при более низких затратах.
- **MySQL** - более простая и легковесная СУБД, но имеет ограниченную функциональность и производительность. PostgreSQL обеспечивает более широкий набор функций и возможностей, а также высокую производительность при работе с большими объемами данных.
- **SQLite** - легковесная СУБД, которая хорошо подходит для мобильных приложений и небольших проектов, но не подходит для работы с большими объемами данных или сложными запросами.
- **Microsoft Access** - является простым СУБД для настольных приложений, разработанным Microsoft. Он предоставляет инструменты для создания баз данных, форм и отчетов с использованием графического интерфейса. Access обладает меньшими возможностями и ограниченной масштабируемостью по сравнению с PostgreSQL. Он часто используется для небольших проектов и индивидуальных пользователей, которым требуется быстрое создание баз данных без необходимости в сложной архитектуре и функциональности.

На рисунке 2.3.1 представлена таблица сравнения СУБД.

Название	Тип	ОС	Лицензия	Исходный код	Доступность	Особенности
Oracle Database	Объектно-реляционная	Linux, Microsoft Windows, Oracle Solaris, IBM AIX, HP-UX	Коммерческая	Закрытый	Высокая стоимость продукта для юридических лиц. Бесплатная версия позволяет развертывать небольшие по объему базы данных	Поддерживает подключение баз данных через облачные сервисы, что позволяет размещать базу данных на нескольких серверах и хранить огромный объем данных. Возможность использования подключаемых БД обеспечивает мультиарендность.
MySQL	Реляционная	Linux, Microsoft Windows, Oracle Solaris, macOS, FreeBSD	GNU GPL и коммерческая	Открытый	Платные версии для коммерческих организаций. В бесплатной версии сделан упор на надежность и скорость работы, но не на полноту функционала	Является довольно гибкой СУБД за счет возможности использовать множество типов таблиц.
Microsoft Access	Реляционная	Microsoft Windows	Проприетарное программное обеспечение EULA	Открытый	Низкая стоимость: Microsoft Access является частью пакета Microsoft Office, что делает его более доступным для многих пользователей.	Имеет богатые функциональные возможностями и удобна в использовании. Особенно удобной является визуальная среда разработки, значительно ускоряющая процесс создания БД. Не многопоточный.
PostgreSQL	Объектно-реляционная	Linux, Microsoft Windows, Oracle Solaris, IBM AIX, macOS, HP-UX, QNX	Свободное и открытое программное обеспечение, разрешительная лицензия	Открытый	Бесплатная и для личного, и для коммерческого использования	PostgreSQL — расширяемая система, ее работа базируется на каталогах (подход catalog-driven). Она хранит информацию не только о таблицах и столбцах, но и о типах данных, типах индексов, функциональных языках и т.д.
SQLite	Файловая	Кроссплатформенная	Свободное и открытое программное обеспечение	Открытый	Бесплатная	Поддерживает только 5 типов данных: Blob, Integer, Null, Text, Real. Не имеет каких-либо конкретных функций управления пользователями и, следовательно, не подходит для многопользовательского доступа.

Рисунок 2.3.1 – Таблица сравнения СУБД

2.3. Выбор обратного прокси-сервиса

Обратный прокси-сервис (reverse proxy) — это сервер, который принимает входящие запросы от клиентов и перенаправляет их на один или несколько серверов, обрабатывающих запросы и возвращающих ответы. Он является посредником между клиентом и сервером, принимая на себя задачи маршрутизации и обработки трафика.

В разрабатываемом приложении, обратный прокси-сервис может использоваться для следующих целей:

Балансировка нагрузки: Обратный прокси-сервис может распределять входящий трафик между несколькими серверами, обрабатывающими запросы. Это позволяет равномерно распределить нагрузку между серверами и обеспечить более стабильную и отзывчивую работу приложения.

Кэширование: Обратный прокси-сервис может кэшировать ответы от серверов, чтобы ускорить обработку последующих запросов от клиентов. Кэширование может быть особенно полезным для статических ресурсов, таких как изображения, CSS-файлы и JavaScript-файлы, которые редко меняются.

SSL-терминация: Обратный прокси-сервис может выполнять SSL-терминацию, что означает, что он расшифровывает зашифрованный трафик от клиента, а затем перенаправляет его на серверы в незашифрованном виде. Это может снизить нагрузку на серверы, освободив их от необходимости обрабатывать шифрование/расшифрование SSL.

Маршрутизация трафика: Обратный прокси-сервис может маршрутизировать запросы на различные серверы в зависимости от заданных правил или условий. Это может быть полезно, если у вас есть несколько серверов с разными функциональными возможностями или разделёнными по географическим зонам.

Выбор использования обратного прокси-сервиса в приложении обусловлен необходимостью обеспечения масштабируемости, безопасности, отказоустойчивости и удобства управления трафиком. Он позволит эффективно управлять входящим трафиком, обеспечивая балансировку нагрузки и защиту серверов от непосредственного доступа со стороны клиентов. Кроме того, обратный прокси-сервис позволяет нам легко внедрять различные функциональные возможности, такие как кэширование и SSL-терминация, улучшая производительность и безопасность разрабатываемого приложения.

Компания HostAdvice произвела исследование веб-серверов с наибольшим количеством пользователей по всему миру, рейтинг сравнения представлен на рисунке 2.4.1. Самыми популярными являются Nginx и Apache [9].

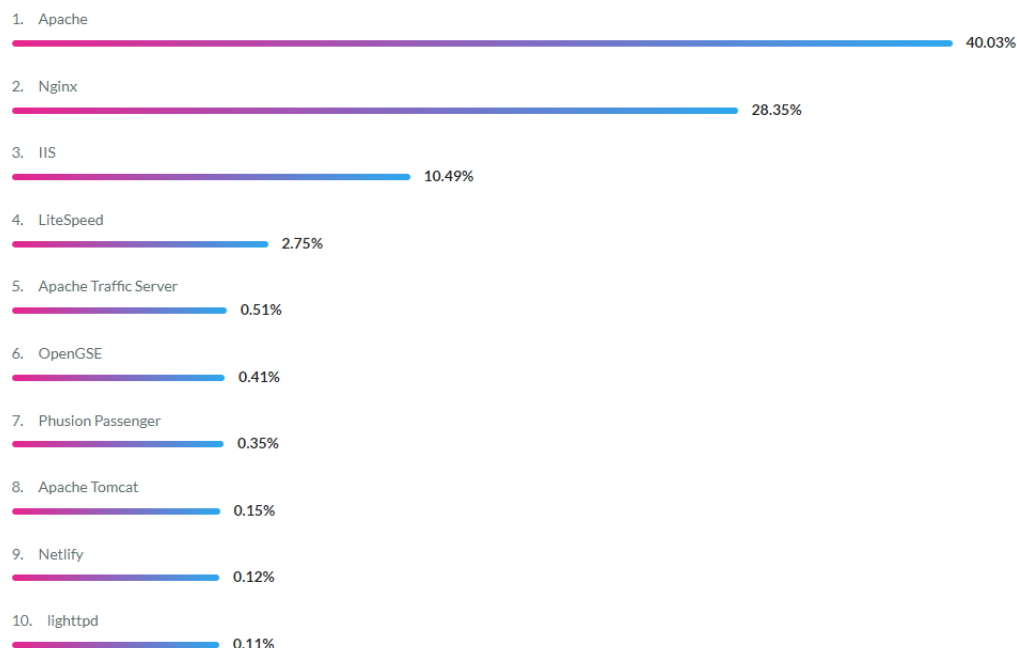


Рисунок 2.4.1 - Рейтинг веб-серверов на 2023 по мнению HostAdvice

Выбор между Nginx и Apache зависит от конкретных потребностей и требований проекта. Оба сервера являются популярными и широко используемыми веб-серверами с отличной функциональностью, но имеют некоторые различия.

Вот несколько причин, по которым был выбран Nginx в качестве обратного прокси-сервиса для разрабатываемого приложения:

Производительность: Nginx известен своей высокой производительностью и низким потреблением ресурсов. Он обрабатывает множество одновременных подключений и распределяет нагрузку эффективно, что особенно полезно при обработке большого количества запросов.

Эффективное управление ресурсами: Nginx работает по принципу асинхронной и событийно-управляемой модели, что позволяет эффективно использовать системные ресурсы. Он легко масштабируется и способен обрабатывать большое количество запросов при небольшом потреблении памяти.

Легкость конфигурации: Конфигурация Nginx основана на простом и понятном синтаксисе. Это делает его более простым в использовании и

настройке по сравнению с Apache, особенно при работе с большими конфигурационными файлами.

Проксирование и балансировка нагрузки: Nginx предоставляет гибкие возможности проксирования и балансировки нагрузки, позволяя эффективно распределять трафик между несколькими серверами и обеспечивать отказоустойчивость системы.

Расширяемость: Nginx поддерживает модульную архитектуру, что позволяет легко расширять его функциональность с помощью сторонних модулей. Это дает большую гибкость в настройке и расширении возможностей сервера.

Хотя Apache также является мощным и популярным веб-сервером с широкими возможностями, был выбран Nginx в соответствии с требованиями к производительности, эффективности использования ресурсов и простоте настройки.

В разрабатываемом приложении Nginx используется в качестве обратного прокси-сервера, который принимает запросы от клиентов и перенаправляет их на сервер Node.js, где запускается разрабатываемое приложение. Таким образом, Nginx играет важную роль в обеспечении безопасности и производительности приложения [10].

2.4. Выбор технологий для упрощения разработки и развёртывания

Для управления версиями исходного кода разрабатываемого приложения был выбран Git. Он является одним из наиболее популярных и надежных инструментов для контроля версий, который позволяет удобно работать с различными ветками разработки, сливать изменения и откатывать неправильные изменения. Кроме того, Git также предоставляет возможность работы в команде, что важно для совместной разработки проекта. Git также имеет обширную

документацию и большое сообщество пользователей, что облегчает его использование и решение возможных проблем [11].

Для упрощения разработки и развертывания приложений был выбран Docker. Он позволяет разработчикам и DevOps-инженерам упаковывать приложения в контейнеры, которые могут быть запущены в любой среде, без необходимости установки зависимостей и настройки окружения.

Docker — это платформа для разработки, доставки и запуска приложений с использованием контейнеризации. Контейнеры — это изолированные среды, которые содержат все необходимое для запуска приложения, включая код, зависимости, библиотеки и настройки окружения. Они позволяют разработчикам создавать приложения, которые могут быть запущены в любой среде без изменения кода [12].

Контейнеры Docker являются легковесными и мобильными, и могут быть развернуты в любой среде, которая поддерживает Docker. Контейнеры не включают в себя операционную систему, они используют ядро хост-системы, что позволяет их запускать на различных платформах.

Контейнеры Docker могут храниться в Docker-реестре, который представляет собой хранилище контейнеров Docker. Этот реестр позволяет делиться контейнерами с другими пользователями и разработчиками. Контейнеры также могут быть сохранены в образы Docker, которые содержат все необходимое для запуска контейнера.

В разрабатываемом приложении Docker используется для упрощения развертывания и масштабирования приложения. Мы используем Docker для создания контейнеров, которые содержат приложение, базу данных PostgreSQL и сервер nginx. Контейнеры могут быть развернуты на любой машине, которая поддерживает Docker, и могут быть масштабированы в зависимости от нагрузки на приложение. Это позволяет быстро и эффективно развертывать, и масштабировать приложение в любой среде.

2.5. Выводы по второму разделу

Во втором разделе были выбраны ключевые технологии и инструменты для разработки и развертывания приложения. Были выбраны следующие компоненты:

1. В качестве основного языка разработки был выбран TypeScript. Это обусловлено его преимуществами, такими как статическая типизация и поддержкой классов и модулей.
2. Для разработки пользовательского интерфейса был выбран фреймворк React, так как предоставляет эффективный и гибкий подход к созданию интерактивных пользовательских интерфейсов. А в качестве серверной среды для разработки приложения был выбран Node.js, так как позволяет разрабатывать масштабируемые и быстрые серверные приложения.
3. После сравнения различных систем управления базами данных был выбран PostgreSQL, предоставляющий надежное хранение и управление данными.
4. Для упрощения разработки и развертывания приложения был выбран Docker. Он позволяет создавать легковесные и изолированные контейнеры, в которых можно упаковать все зависимости и настройки приложения, обеспечивая его переносимость и масштабируемость. В качестве системы контроля версий был выбран Git.
5. В качестве обратного прокси-сервиса был выбран Nginx. Он предоставляет высокую производительность и эффективность при обработке запросов, балансировке нагрузки, кешировании, обеспечивая безопасность и удобное управление маршрутизацией запросов.

Выбор этих инструментов и технологий обусловлен их преимуществами, совместимостью, расширяемостью и поддержкой в разработке приложений. Они позволяют разработчикам создавать высококачественное, масштабируемое и надежное приложение, обеспечивают удобство разработки, тестирования, развертывания и поддержки.

3. Проектирование приложения

3.1. Выбор архитектуры

Архитектура приложения определяет организацию компонентов и способ взаимодействия между ними для достижения определенных функциональных и нефункциональных требований. Она описывает структуру приложения, распределение его компонентов и связей между ними.

Существуют различные типы архитектур приложений, включая:

- **Клиент-серверная архитектура:** это распределенная архитектура, где клиентские устройства (клиенты) обращаются к серверам для получения данных или выполнения определенных задач. Серверы отвечают на запросы клиентов, обеспечивают хранение данных и выполнение бизнес-логики. Эта архитектура позволяет разделить ответственность между клиентской и серверной частями, обеспечивает масштабируемость и удобство обновлений.
- **Монолитная архитектура:** в этом типе архитектуры вся функциональность приложения интегрирована в единый модуль (монолит), который запускается на одном сервере или устройстве. Монолит содержит все компоненты приложения, включая пользовательский интерфейс, бизнес-логику и доступ к данным. Это простой подход, но может ограничивать масштабируемость и гибкость разработки и развертывания.
- **Микросервисная архитектура:** в этой архитектуре приложение разбивается на отдельные независимые сервисы, каждый из которых выполняет определенную функцию и взаимодействует с другими сервисами через сетевые вызовы. Микросервисы могут быть развернуты и масштабированы независимо, что облегчает разработку и поддержку приложения. Однако это требует дополнительных усилий для управления и координации межсервисными вызовами.

В сравнении с другими архитектурами, клиент-серверная архитектура обладает следующими преимуществами:

- Лучшая масштабируемость и производительность: клиент-серверная архитектура позволяет распределить нагрузку между клиентами и серверами, что обеспечивает более высокую производительность и возможность масштабирования.
- Легкая разработка и обновление: разделение функциональности между клиентской и серверной частями упрощает разработку и поддержку приложения. Кроме того, изменения в одной части (например, на сервере) могут быть внесены без влияния на другую часть (например, клиентский интерфейс).
- Лучшая безопасность: клиент-серверная архитектура позволяет реализовать различные уровни безопасности, включая аутентификацию, авторизацию и шифрование данных, что повышает общую безопасность приложения.
- Лучшая гибкость и расширяемость: разделение приложения на клиентскую и серверную части делает его более гибким и расширяемым. Можно добавлять новые функциональные возможности, модифицировать существующие и интегрировать сторонние сервисы, не затрагивая весь код приложения.

Однако, каждая архитектура имеет свои особенности и выбор зависит от конкретных требований и целей проекта.

В разрабатываемом приложении клиент-серверная архитектура используется для реализации распределенной системы, где клиентское приложение взаимодействует с сервером через сетевые запросы. Клиентская часть приложения написана с использованием React и TypeScript, а серверная часть использует Node.js и PostgreSQL.

Логически архитектура клиент-серверной системы выглядит следующим образом: на серверной стороне находится база данных PostgreSQL и сервер Node.js, который обрабатывает запросы от клиентского приложения и взаимодействует с базой данных для получения и хранения информации. На

клиентской стороне находится веб-браузер, в котором работает React-приложение, которое посылает запросы к серверу и отображает полученные данные в пользовательском интерфейсе. Логическая архитектура приложения представлена на рисунке 3.1.1.

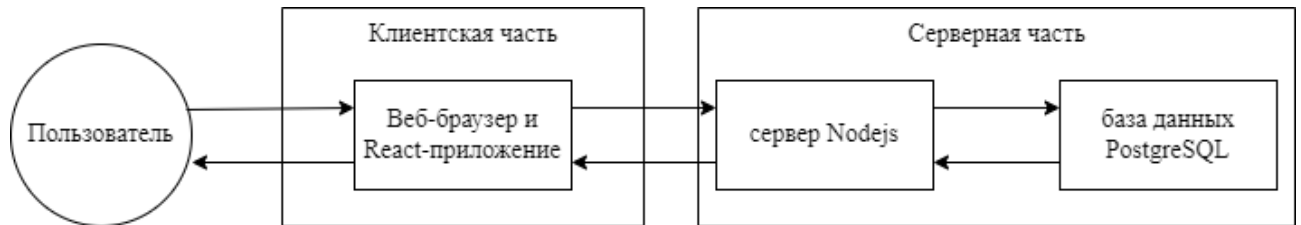


Рисунок 3.1.1 – Логическая архитектура приложения

Клиент-серверная архитектура была выбрана для разрабатываемого приложения, потому что она обеспечивает высокую масштабируемость и расширяемость системы, позволяет легко добавлять новые функциональные возможности, а также упрощает тестирование и отладку приложения.

Системная архитектура разрабатываемого приложения представлена на рисунке 3.1.2 и выглядит следующим образом:

1. Виртуальный или физический сервер, на котором работает операционная система.
2. На сервере установлен и запущен Docker.
3. В Docker контейнерах запущены следующие сервисы:
 - React-приложение - посылает запросы к серверу и отображает полученные данные в пользовательском интерфейсе
 - Сервер Node.js – сервер, который обрабатывает запросы от пользователей и взаимодействует с базой данных.
 - PostgreSQL - система управления реляционными базами данных, где хранится вся необходимая информация для работы приложения.
 - Nginx - веб-сервер и обратный прокси, который используется для балансировки нагрузки и обеспечения безопасности.
4. Пользователи могут обращаться к приложению через интернет-браузер, который отображает интерфейс на React-приложение через Nginx.

5. React-приложение отправляет запросы Сервер Node.js через Nginx и получает данные, которые отображает пользователю через Nginx.
6. Сервер Node.js обрабатывает запросы, обращается к базе данных PostgreSQL для получения или изменения необходимых данных и отправляет ответ React-приложению через Nginx.

Таким образом, все приложение работает в контейнерах Docker на одном сервере, что обеспечивает высокую производительность и масштабируемость, а также упрощает управление и развертывание приложения.

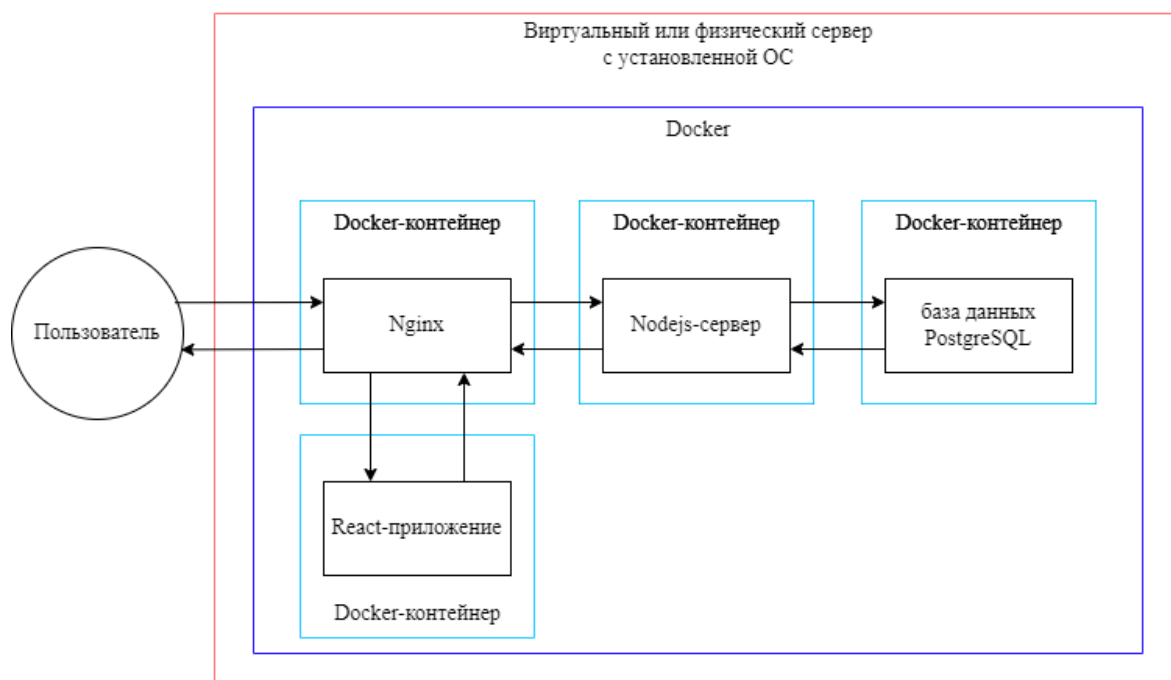


Рисунок 3.1.2 – Системная архитектура приложения

Docker-контейнеры могут располагаться на разных устройствах и взаимодействовать между собой. Для этого используется Docker Swarm, который позволяет управлять группой Docker-хостов в качестве единой виртуальной машины и координировать работу контейнеров на разных узлах. Docker Swarm автоматически распределяет контейнеры по узлам, чтобы обеспечить балансировку нагрузки и высокую доступность. Кроме того, можно использовать Docker Compose, который позволяет описывать и запускать множество контейнеров на разных устройствах в одной сети, что обеспечивает их взаимодействие друг с другом.

3.2. Проектирование базы данных

Концептуальная ER-модель представляет собой высокоуровневое описание базы данных, не зависящее от конкретных технологий и инструментов. Она описывает основные сущности, их атрибуты и связи между ними.

В разрабатываемом приложении концептуальная ER-модель будет содержать следующие сущности:

- Пользователь: представляет собой пользователя системы, который может загружать, просматривать и скачивать файлы, а также назначать роли другим пользователям.
- Файл: представляет собой загруженный пользователем файл, который может быть просмотрен и скачан другими пользователями с соответствующими правами доступа.
- Роль: представляет собой роль пользователя, которая определяет его права доступа к файлам. Роль может быть назначена нескольким пользователям.
- История действий: представляет собой запись о действиях пользователей в системе, таких как загрузка, просмотр и скачивание файлов.

Связи между сущностями можно описать следующим образом:

- Пользователь может загружать несколько файлов, а каждый файл может быть загружен одним пользователем.
- У пользователя может быть только одна роль, а роль может быть назначена нескольким пользователям.
- Для каждого действия в истории действий должен быть определен пользователь, который его совершил, и файл, над которым это действие было выполнено.

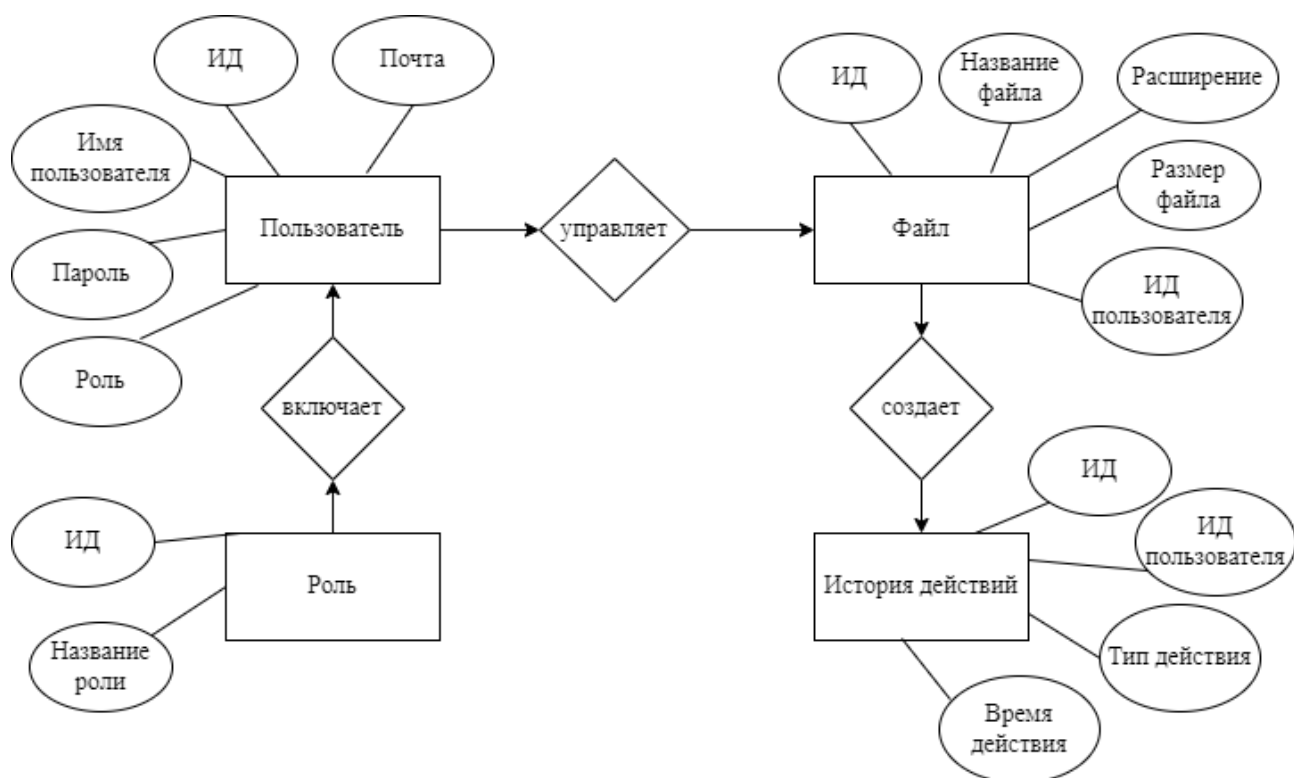


Рисунок 3.2.1 - Концептуальная ER-модель

На основе концептуальной модели выделим сущности и составим инфологическую модель.

Инфологическая модель – это более конкретная модель, чем концептуальная. Она описывает структуру данных и отношения между ними. В этой модели представлены таблицы базы данных и связи между ними.

Для разрабатываемого приложения, инфологическая модель может быть следующей:

Таблица "users" содержит информацию о пользователях системы:

- id (целое число) – уникальный идентификатор пользователя.
- username (строка) – имя пользователя.
- password (строка) – пароль пользователя.
- email (строка) – адрес электронной почты пользователя.
- role_id (целое число) – уникальный идентификатор роли.

Таблица "files" содержит информацию о файлах:

- id (целое число) – уникальный идентификатор файла.
- filename (строка) – название файла.

- extension (строка) – расширение файла.
- size (целое число) – размер файла.
- user_id (целое число) – идентификатор пользователя, которому принадлежит файл.

Таблица "roles" содержит информацию о ролях пользователей:

- id (целое число) – уникальный идентификатор роли.
- name (строка) – название роли.

Таблица "actions_history" содержит информацию об истории действий пользователей:

- id (целое число) – уникальный идентификатор записи об истории.
- user_id (целое число) – идентификатор пользователя, который совершил действие.
- file_id (целое число) – идентификатор файла, над которым совершено действие.
- action_type (строка) – тип совершенного действия.
- timestamp (дата и время) – время совершения действия.

Связи между таблицами:

- Таблица "users" связана с таблицей "files" через поле user_id.
- Таблица "users" связана с таблицей "roles" через поле role_id.
- Таблица "actions_history" связана с таблицей "files" через поле user_id.

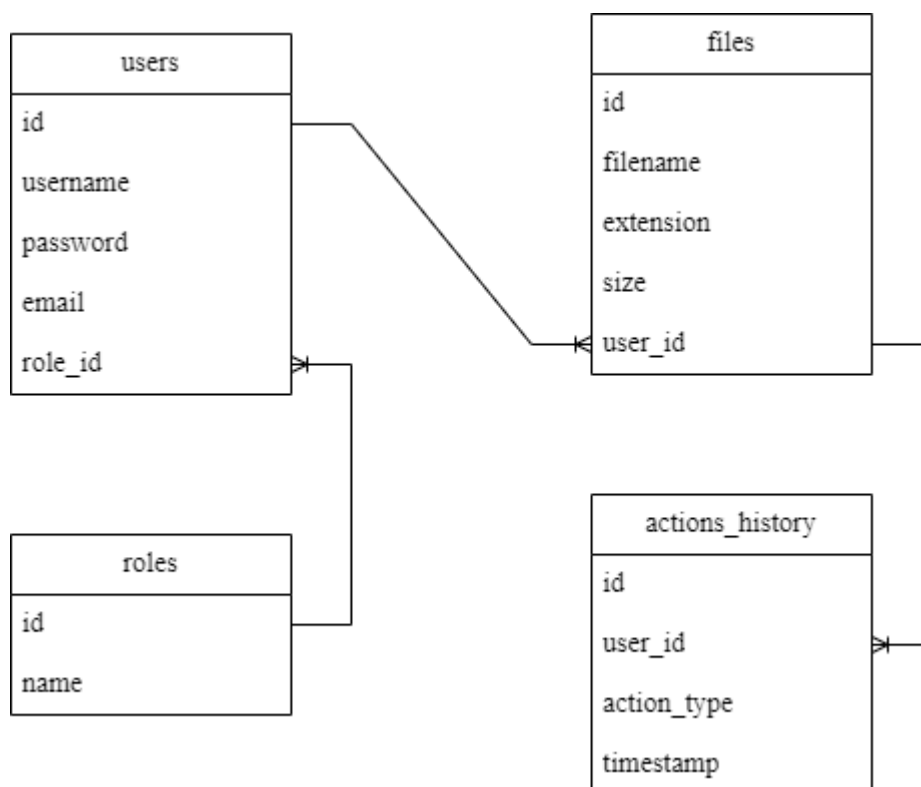


Рисунок 3.2.2 – Инфологическая модель

На основе инфологической модели разработаем даталогическую модель.

Даталогическая модель — это модель базы данных, которая описывает структуру данных и ограничения целостности, применяемые к данным в базе данных. Она более конкретна, чем инфологическая модель, и включает в себя все детали, необходимые для реализации базы данных в конкретной системе управления базами данных (СУБД).

В разрабатываемом приложении даталогическая модель будет содержать следующие таблицы представленные в таблицах 3.2.1-3.2.4:

Таблица 3.2.1 – Таблица «Users»

Название поля	Ключ	Тип данных	Описание
id	PK	SERIAL PRIMARY KEY	Уникальный идентификатор пользователя
username		VARCHAR (255) NOT NULL	Имя пользователя
password		VARCHAR (255) NOT NULL	Хэш пароля пользователя
email		VARCHAR (255) NOT NULL	Электронная почта пользователя
role_id	FK	INTEGER NOT NULL	ID роли, которую имеет пользователь

Таблица 3.2.2 – Таблица «Roles»

Название поля	Ключ	Тип данных	Описание
id	PK	SERIAL PRIMARY KEY	Уникальный идентификатор роли
name		VARCHAR (255) NOT NULL	Название роли

Таблица 3.2.3 – Таблица «Files»

Название поля	Ключ	Тип данных	Описание
id	PK	SERIAL PRIMARY KEY	Уникальный идентификатор файла
name		VARCHAR (255) NOT NULL	Название файла
size		INTEGER NOT NULL	Размер файла в байтах
extension		VARCHAR (255) NOT NULL	Расширение файла
path		VARCHAR (255) NOT NULL	Путь к файлу на сервере
created_at		TIMESTAMP NOT NULL	Дата и время создания файла
user_id	FK	INTEGER NOT NULL	ID пользователя, создавшего файл
updated_at		TIMESTAMP NOT NULL	Дата и время последнего обновления файла

Таблица 3.2.4 – Таблица «Actions_History»

Название поля	Ключ	Тип данных	Описание
id	PK	SERIAL PRIMARY KEY	Уникальный идентификатор действия
user_id		INTEGER NOT NULL	ID пользователя, который выполнил действие
file_id	FK	INTEGER NOT NULL	ID файла, с которым связано действие
action_type		VARCHAR (255) NOT NULL	Тип действия (создание/редактирование/удаление)
action_time		TIMESTAMP NOT NULL	Дата и время выполнения действия

На рисунке 3.2.3 представлена разработанная даталогическая модель

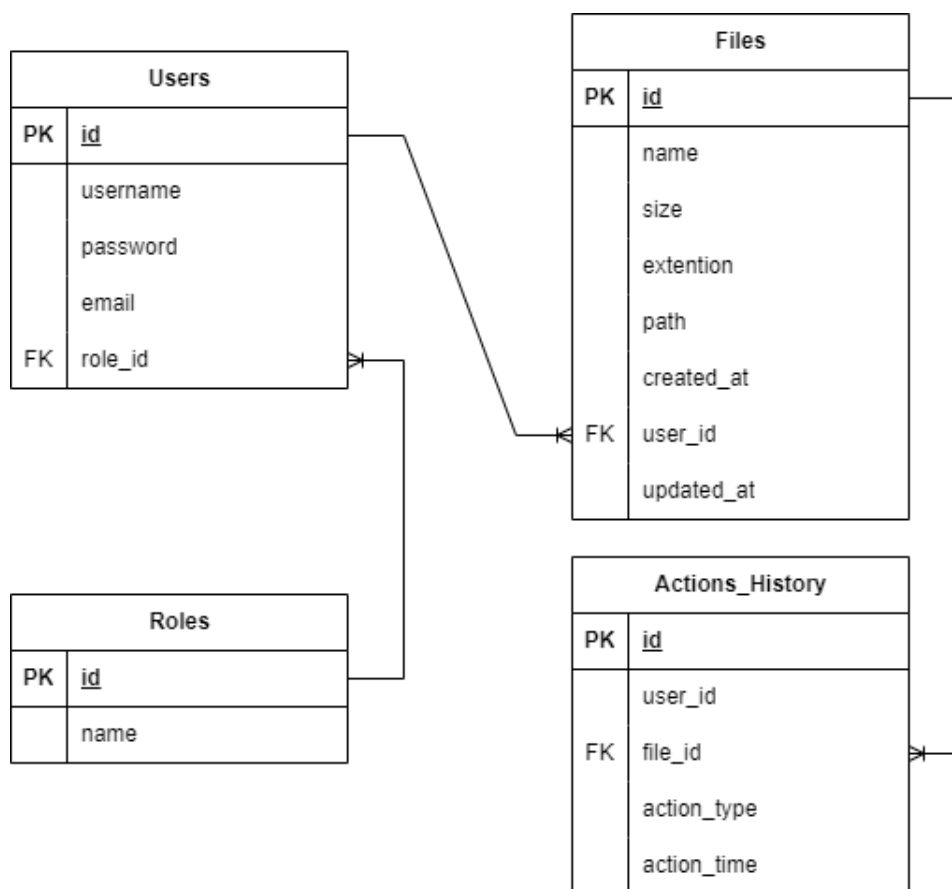


Рисунок 3.2.3 – Даталогическая модель

3.3. Определение ролей пользователей

В разрабатываемом приложении будет существовать три основные роли пользователей: пользователь (User), администратор (Admin) и суперпользователь (Superuser).

1. Пользователь (User):

- Пользователь может загружать файлы в систему.
- Пользователь может скачивать файлы с системы на свое устройство.
- Пользователь может удалять свои собственные файлы из системы.
- Пользователь может просматривать историю своих действий и операций с файлами.

2. Администратор (Admin):

- Администратор имеет все возможности пользователя, включая загрузку, скачивание и удаление файлов, а также просмотр своей истории.
- Администратор может просматривать историю всех пользователей в системе, включая их действия и операции с файлами.

3. Суперпользователь (Superuser):

- Суперпользователь имеет все возможности администратора, включая загрузку, скачивание, удаление файлов и просмотр истории всех пользователей.
- Суперпользователь может изменять роли других пользователей в системе. Это означает, что он может повышать или понижать пользователей до роли администратора или пользователя.

Каждая роль имеет определенные права и функциональности, которые определяют, какие действия пользователь может выполнять в приложении. Такая система ролей позволяет эффективно управлять доступом и контролировать действия пользователей в системе.

3.4. Разработка макетов

Для приложения были разработаны два основных макета: макет для однотипных окон с изменяемым содержимым блоков и макет окна формы с полями для заполнения и кнопками.

Макет для однотипных окон представлен на рисунке 3.4.1.

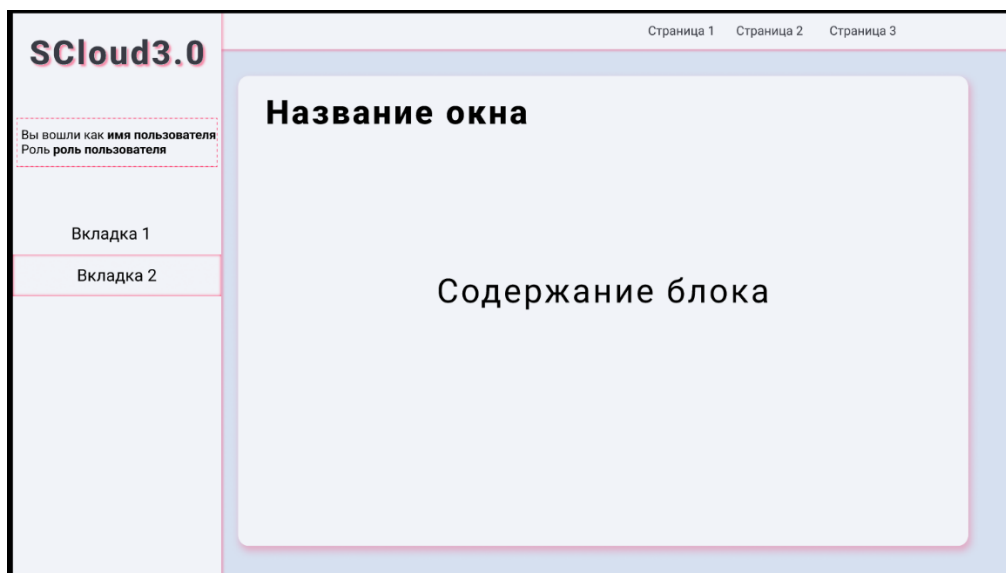


Рисунок 3.4.1 – Макет для однотипных окон

Данный макет предназначен для экранов, где основная структура и компоненты окна остаются постоянными, а меняется только содержимое блоков в зависимости от открытой страницы. Он включает общие элементы интерфейса, такие как шапка, навигационное меню, боковая панель и основной контент.

Для каждого типа окна определены специфические блоки, которые могут изменяться в зависимости от страницы. Например, для страницы списка файлов может быть блок с таблицей файлов, для страницы профиля пользователя - блок с личными данными и настройками.

Макет окна формы представлен на рисунке 3.4.2.

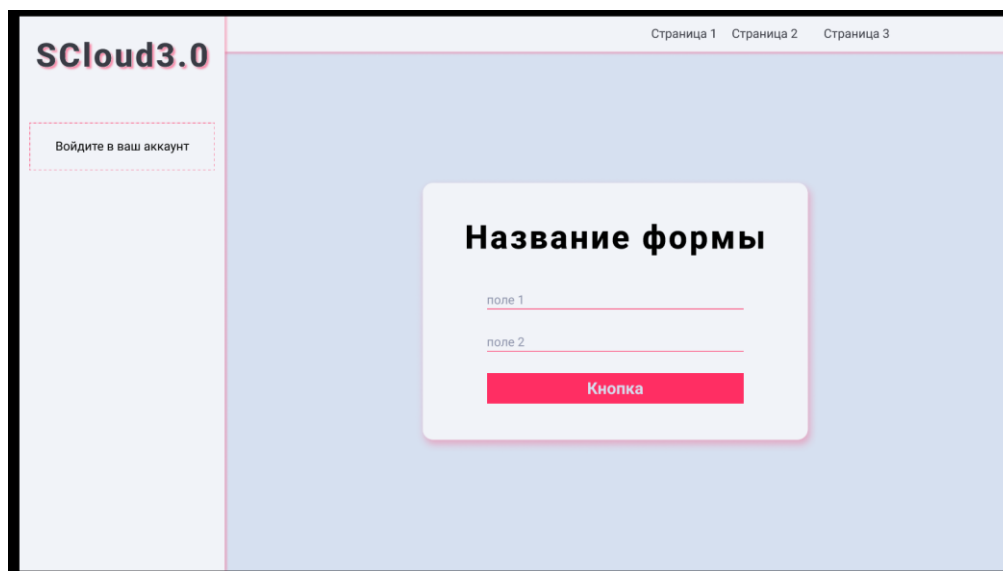


Рисунок 3.4.2 – Макет окна формы

Этот макет предназначен для окна формы, например формы авторизации, где пользователи вводят свои учетные данные для доступа к приложению. Он включает поля для заполнения, такие как поле ввода логина и поле ввода пароля. Также в макете присутствуют кнопки для выполнения действий, например, кнопка "Войти" для отправки учетных данных и выполнения процесса авторизации.

Оба макета разработаны с учетом принципов удобства использования, эстетики и согласованности дизайна. Их цель - обеспечить понятность, интуитивность и эффективность работы пользователя с приложением. Они служат основой для разработки реального пользовательского интерфейса приложения.

3.5. Обеспечение безопасности приложения

Обеспечение безопасности приложения является важной задачей, чтобы защитить данные и функциональность от несанкционированного доступа, атак и утечек информации. Для этого можно использовать различные средства и технологии. Будут использованы следующие технологии:

1. **Аутентификация и авторизация:** реализация механизмов аутентификации и авторизации позволяет проверить подлинность пользователей и определить их права доступа к различным ресурсам и функциональности приложения. Один из распространенных способов аутентификации — это использование токенов.
2. **Использование токенов:** токены являются безопасными и уникальными идентификаторами, которые выдаются после успешной аутентификации пользователей. Токены могут быть сгенерированы на стороне сервера и переданы клиенту, который включает их в каждый последующий запрос. Это позволяет серверу проверить подлинность запроса и авторизовать пользователя без необходимости передачи пароля или учетных данных при каждом запросе.
3. **Хэширование паролей:** для хранения паролей пользователей в базе данных рекомендуется использовать хэширование, чтобы защитить пароли от несанкционированного доступа. Хэширование паролей преобразует пароль в непрочитаемую форму, что делает его сложным для восстановления в исходном виде.
4. **Защита от инъекций:** для предотвращения атак инъекций, таких как SQL-инъекции или XSS-атаки, рекомендуется использовать подготовленные запросы и проверку пользовательского ввода перед его использованием в запросах или выводе на веб-страницу.
5. **Защита данных в покое и в передаче:** для защиты данных от несанкционированного доступа или прослушивания, рекомендуется использовать шифрование данных при их хранении и передаче. Протоколы шифрования, такие как SSL/TLS, обеспечивают защищенное соединение между клиентом и сервером.
6. **Ограничение доступа:** для обеспечения безопасности приложения необходимо ограничить доступ к системным ресурсам и функциональности

только для авторизованных пользователей или ролей. Это позволяет предотвратить несанкционированный доступ или изменение данных.

Важно отметить, что обеспечение безопасности является непрерывным процессом и требует постоянного обновления и мониторинга с учетом новых угроз и методов атак.

3.6. Выводы по третьему разделу

1. Для разработки приложения будет использоваться клиент-серверная модель для разделения функциональности между клиентской и серверной сторонами. Это позволяет более эффективно управлять данными, обеспечивать безопасность и масштабируемость приложения.
2. Разработанная схема базы данных позволит оптимально хранить и управлять данными с помощью СУБД PostgreSQL.
3. Разработаны макеты для основных окон приложения, включая макеты типичных окон с изменяющимися содержимыми блоков и макет окна авторизации, обеспечивают пользователям удобную работу с приложением.
4. Для обеспечения безопасности работы приложения будут использоваться аутентификация и авторизация пользователей, использование токенов для проверки подлинности запросов, хэширование паролей, защита от инъекций и шифрование данных. Эти меры помогают защитить приложение от несанкционированного доступа и атак.

Эти решения обеспечивают надежность, масштабируемость и безопасность приложения, что является важным для успешной реализации поставленных требований и удовлетворения потребностей пользователей.

4. Реализация приложения

4.1. Структура веб-интерфейса

Для приложения был разработан веб-интерфейс, основанный на библиотеке React. Интерфейс включает в себя несколько страниц:

- Страница авторизации - пользователь может войти в систему, введя свою почту и пароль.
- Главная страница - после успешной авторизации пользователь перенаправляется на главную страницу. На этой странице может просмотреть историю
- Страница профиля - на этой странице пользователь может просмотреть и отредактировать свой профиль. Он может изменить свое имя, адрес электронной почты и пароль.
- Страница информации - на этой странице пользователь может посмотреть информацию о приложении и ее разработчиках.

Более подробная структура веб-интерфейса с разбиением на роли представлена на рисунке 4.1.1.

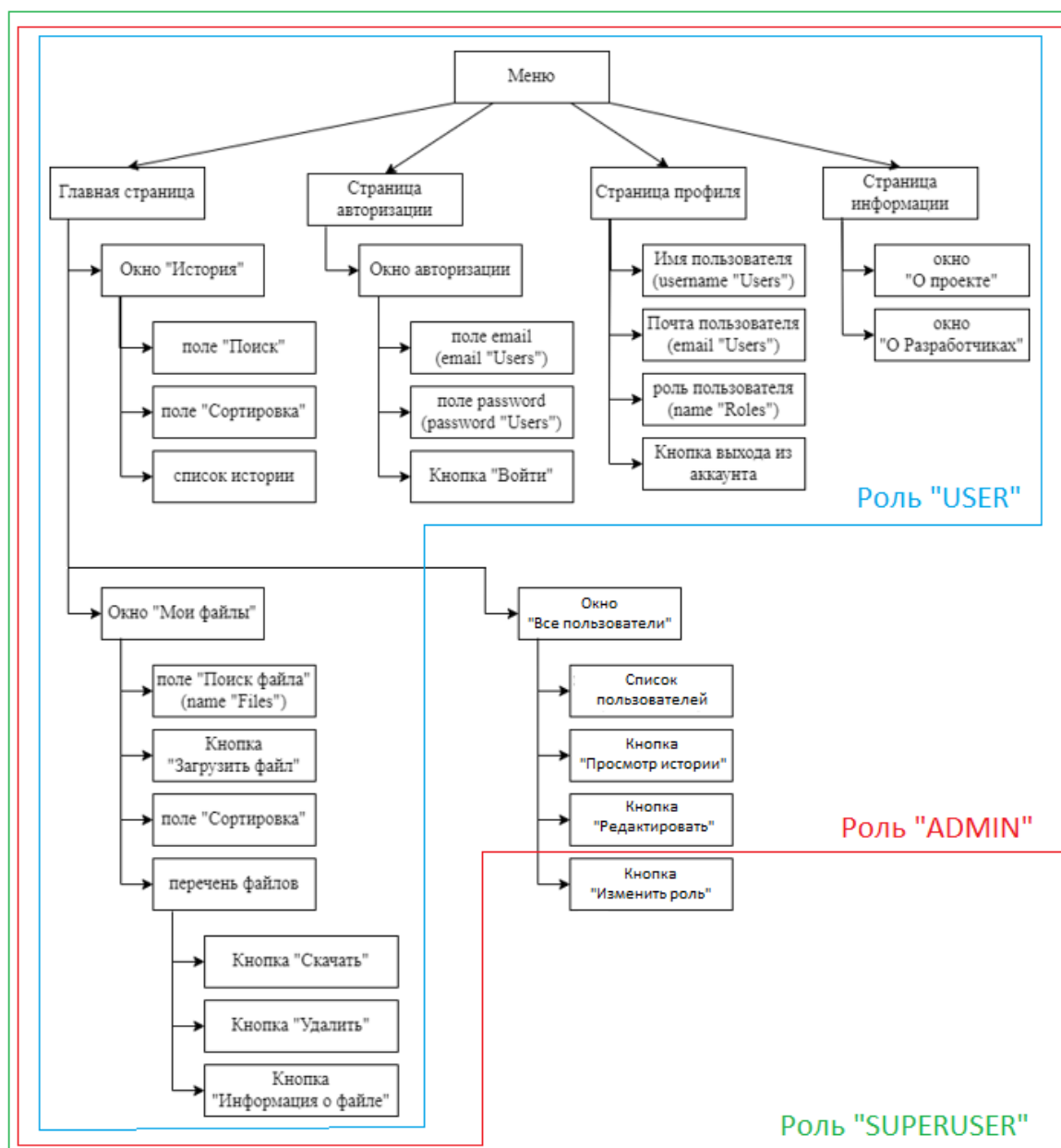


Рисунок 4.1.1 – Структура веб-интерфейса

4.2. Описание функциональности приложения

Функциональность разрабатываемого приложения включает следующие аспекты:

1. Авторизация:

- Пользователь может создать аккаунт или использовать существующий для входа в приложение.

- При авторизации пользователь вводит свой логин и пароль.
- После успешной авторизации пользователь получает доступ к функционалу приложения.

2. Управление файлами:

- Пользователь может загружать файлы на сервер и хранить их в своем личном пространстве.
- Приложение предоставляет возможность создавать папки и структурировать файлы внутри них.
- Пользователь может просматривать, редактировать и удалять файлы.
- Для удобства навигации и поиска предоставляются функции сортировки и фильтрации файлов.

3. Поиск и фильтрация:

- Приложение предоставляет возможность поиска файлов по различным критериям, таким как название, расширение или дата создания.
- Пользователь может применять фильтры для отображения только определенных типов файлов или определенной группы файлов.

4. Защита данных:

- Приложение обеспечивает безопасное хранение и передачу файлов и пользовательских данных с использованием шифрования и других механизмов защиты.
- Пользовательские данные хранятся в базе данных с использованием соответствующих механизмов безопасности, таких как хеширование паролей.

5. Административный функционал:

- Существует роль администратора, которая имеет расширенные права доступа и возможности управления пользователями, настройками приложения и другими административными функциями.

6. Требования к пользователям:

- Пользователь должен иметь аккаунт в системе для использования функционала приложения.

- Для доступа к определенным функциям приложения могут быть установлены дополнительные роли или права.

7. Дублирование данных:

- В приложении предусмотрено хранение пользовательских файлов на сервере.
- Каждый пользователь имеет свое пространство для хранения файлов, что обеспечивает разделение данных между пользователями.
- Файлы могут быть дублированы на сервере в случае, если пользователь загружает один и тот же файл несколько раз.

В таблице 4.2.1 представлены роли пользователей (Пользователь, Администратор, Суперпользователь) и возможности, которыми обладает каждая роль. Как видно из таблицы, каждая роль имеет определенные привилегии в приложении, которые определяют их функциональность и доступ к определенным операциям и данным.

Таблица 4.2.1 – Возможности ролей пользователей

Возможности	Пользователь	Администратор	Суперпользователь
Загрузка файлов	+	+	+
Скачивание файлов	+	+	+
Удаление файлов	+	+	+
Просмотр своей истории	+	+	+
Просмотр истории всех пользователей	-	+	+
Изменение ролей пользователей	-	-	+

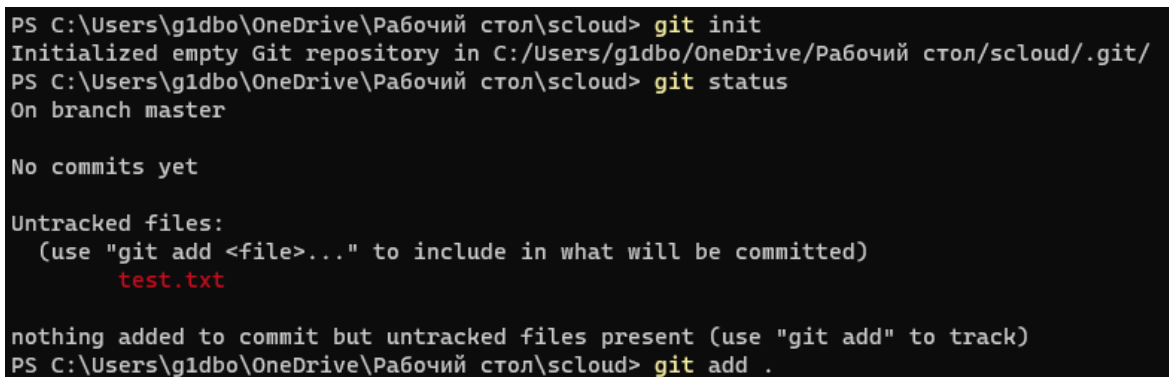
4.3. Процесс разработки

Процесс создания приложения включает несколько этапов, таких как настройка системы контроля версий, настройка Docker, настройка Node.js сервера и разработка веб-интерфейса на React. Рассмотрим каждый этап поэтапно:

1. Настройка системы контроля версий (Git):

- Устанавливается Git на компьютере, если его еще нет.
- Создается новый репозиторий Git для проекта.
- Инициализируется репозиторий с помощью команды **git init**.
- Создается .gitignore файл, чтобы исключить ненужные файлы и папки из отслеживания Git.
- Исходный код проекта добавляется и коммитится в репозиторий с помощью команд **git add** и **git commit**.

Пример настройки представлен на рисунке 4.3.1.



```
PS C:\Users\gldbo\OneDrive\Рабочий стол\scloud> git init
Initialized empty Git repository in C:/Users/gldbo/OneDrive/Рабочий стол/scloud/.git/
PS C:\Users\gldbo\OneDrive\Рабочий стол\scloud> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
       test.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\gldbo\OneDrive\Рабочий стол\scloud> git add .
```

Рисунок 4.3.1 - Настройка системы контроля версий

2. Настройка Docker:

- Устанавливается Docker на компьютере или сервере.
- Создаются файлы настройки Docker для каждого компонента приложения (например, Dockerfile для Node.js сервера и Docker-compose для связи контейнеров).
- В Dockerfile указывается базовый образ, зависимости, порты и другие настройки для каждого компонента.
- В Docker-compose файле определяются сервисы, их зависимости, порты и другие параметры.
- Собираются Docker-образы с помощью команды **docker build** и запускаются контейнеры с помощью команды **docker-compose up**, пример запуска представлен на рисунке 4.3.2.

Пример файлов настройки Docker можно посмотреть в приложении А.

```
(chat-env) C:\Users\g1dbo\OneDrive\Рабочий стол\chatBot>docker-compose up
[+] Running 2/2
✓ Network chatbot_default Created
0.1s
✓ Container chatbot-bot-1 Created
0.4s
Attaching to chatbot-bot-1
Gracefully stopping... (press Ctrl+C again to force)
Aborting on container exit...
[+] Running 0/1
[+] Running 0/1atbot-bot-1 Stopping
2.2s
[+] Running 1/1atbot-bot-1 Killed
0.6s
✓ Container chatbot-bot-1 Stopped
2.7s
```

Рисунок 4.3.2 – Пример запуска docker-контейнеров

3. Настройка Node.js сервера:

- Устанавливается Node.js и npm на компьютере.
- Создается директория для серверной части приложения.
- Инициализируется проект с помощью команды **npm init**.
- Устанавливаются необходимые пакеты и зависимости для сервера с помощью команды **npm install**, пример инициализации проекта представлен на рисунке 4.3.3.
- Создаются файлы серверной логики, пишется код для обработки запросов и взаимодействия с базой данных.

```
PS C:\Users\g1dbo\OneDrive\Рабочий стол\scloud> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (scloud)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

Рисунок 4.3.3 – Пример инициализации проекта

4. Разработка веб-интерфейса на React:

- Создается директория для клиентской-части приложения.
- Инициализируется проект с помощью команды **npx create-react-app** или аналогичной, пример можно посмотреть на рисунке 4.3.4.
- Устанавливаются необходимые пакеты и зависимости для React с помощью команды **npm install**.
- Разрабатываются компоненты и стили для веб-интерфейса, используя React, TSX, CSS и другие технологии.
- Пишется код для обработки событий, связи с сервером и отображения данных в приложении.

```
PS C:\Users\gldbo\OneDrive\Рабочий стол\scloud> npx create-react-app client
Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will
upgrade asap.

Creating a new React app in C:\Users\gldbo\OneDrive\Рабочий стол\scloud\client.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1434 packages in 2m

234 packages are looking for funding
  run 'npm fund' for details

Installing template dependencies using npm...
|
```

Рисунок 4.3.4 – Пример инициализации клиентской части

Каждый этап требует внимательности и умения работать с соответствующими инструментами. После завершения всех шагов получается полностью функциональное приложение, готовое для развёртывания и использования.

4.4. Алгоритм авторизации

Алгоритм авторизации представляет собой последовательность шагов, которые позволяют проверить подлинность пользователя и предоставить ему

доступ к защищенным ресурсам или функциональности приложения. Вот общий алгоритм авторизации:

1. Пользователь вводит свои учетные данные (например, имя пользователя и пароль) на странице авторизации приложения.
2. Приложение получает введенные учетные данные и передает их на серверную сторону. Код для передачи данных на серверную сторону представлен на рисунке 4.4.1.

```
import axios from 'axios';

async function login(username: string, password: string): Promise<string> {
  try {
    const response = await axios.post('/api/login', { username, password });
    const token = response.data.token;
    // Возвращаем полученный токен для дальнейшего использования
    return token;
  } catch (error) {
    throw new Error('Ошибка авторизации: ' + error.message);
  }
}

// Пример использования функции login
login('username', 'password')
  .then((token) => {
    console.log('Авторизация успешна. Токен:', token);
  })
  .catch((error) => {
    console.error('Ошибка авторизации:', error);
  });
```

Рисунок 4.4.1 – Код авторизации

3. На сервере происходит проверка введенных данных. Это включает следующие шаги:
- Проверка существования пользователя в базе данных.
 - Сравнение введенного пароля с хэшированным паролем, хранящимся в базе данных.
 - Проверка других условий, таких как активность аккаунта или роли пользователя.

4. Если проверка учетных данных прошла успешно, сервер генерирует уникальный токен доступа для пользователя. Токен содержит информацию о пользователе и его правах.
5. Сгенерированный токен доступа возвращается на клиентскую сторону приложения.
6. Клиент сохраняет токен доступа, например, в локальном хранилище (например, в `localStorage` или `sessionStorage`) или в куках браузера.
7. Далее, при каждом запросе на сервер, клиент включает токен доступа в заголовок запроса или включать его в теле запроса, чтобы сервер мог проверить подлинность пользователя.
8. Сервер проверяет валидность токена доступа при каждом запросе и предоставляет доступ к защищенным ресурсам или функциональности, если токен является действительным и имеет соответствующие права.
9. В случае истечения срока действия токена доступа или его недействительности (например, из-за смены пароля или выхода пользователя), пользователь должен пройти процедуру авторизации заново.

Блок-схема алгоритма авторизации приведена в приложении Б.

4.5. Создание основных окон приложения

При первом запуске открывается окно авторизации, представленное на рисунке 4.5.1, в котором нужно ввести электронную почту и пароль, после чего нажать кнопку «Войти».

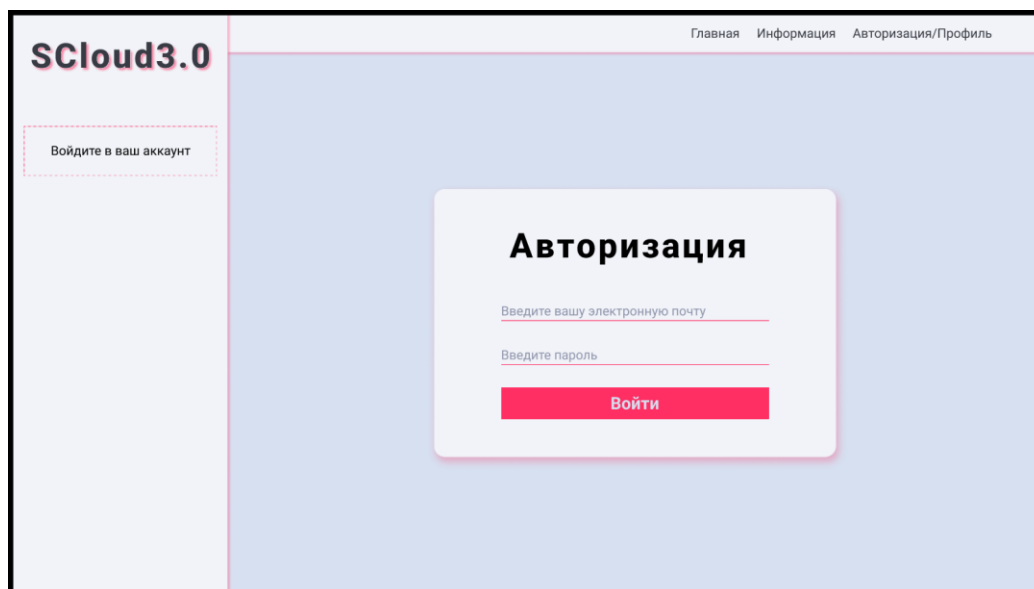


Рисунок 4.5.1 – Страница авторизации

После прохождения авторизации будет открыто окно профиля, представленное на рисунке 4.5.2, в нем можно просмотреть информацию о пользователе или выйти из аккаунта.

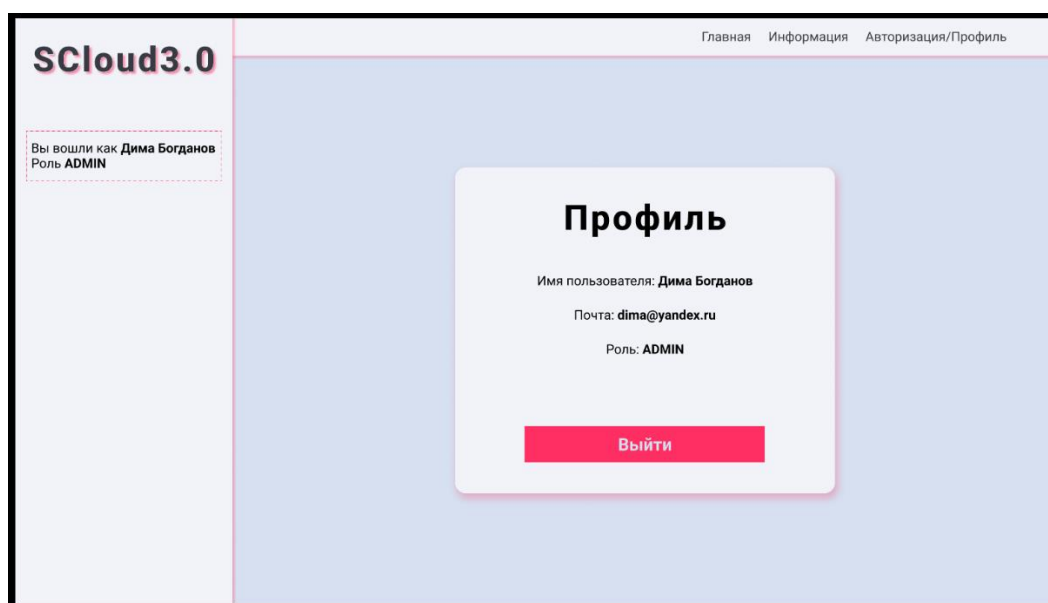


Рисунок 4.5.2 – Страница профиля

Через главное меню можно перейти на главную станицу или на страницу информации, представленные на рисунках 4.5.3-4.5.4. На странице информации можно посмотреть основную информацию о проекте и разработчиках.

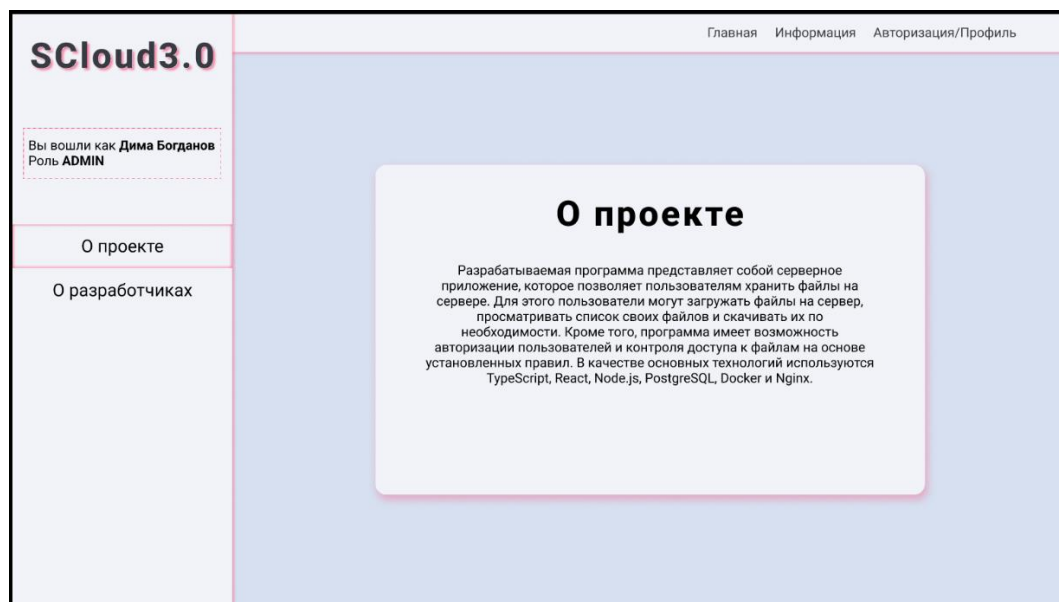


Рисунок 4.5.3 – Страница «Информация», окно «О проекте»

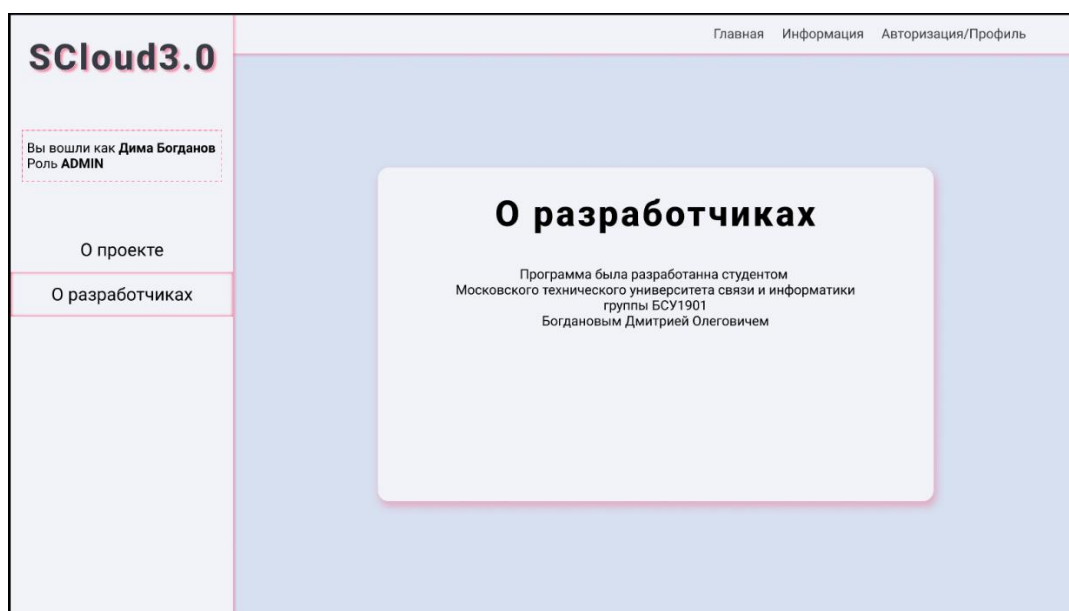


Рисунок 4.5.4 – Страница «Информация», окно «О разработчиках»

Перейдя на главную страницу, можно открыть два окна: «Мои файлы» и «История», представленные на рисунках 4.5.5-4.5.6. В окне «Мои файлы» отображен перечень всех загруженных файлов. Можно загрузить новые файлы, отсортировать их, открывать папки и произвести поиск по файлам. С помощью контекстного меню можно скачивать, удалять или смотреть информацию о выбранном файле.

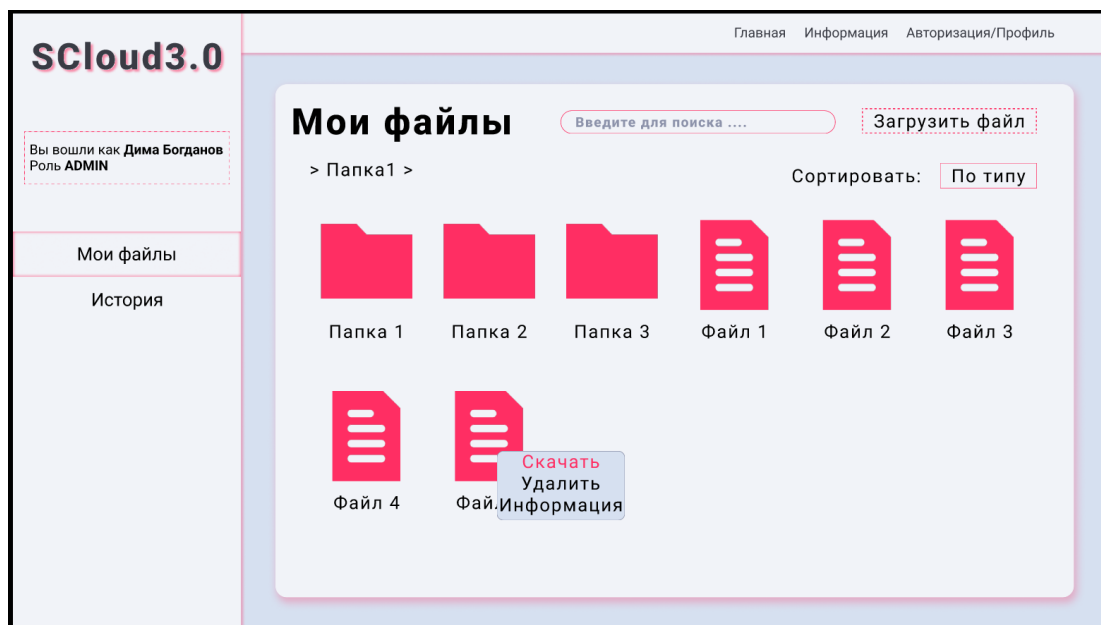


Рисунок 4.5.5 – Страница «Главная», окно «Мои файлы»

В окне «История» можно просмотреть историю действий с файлами на аккаунте. Информация представлена в виде таблицы, которую можно отсортировать или произвести поиск по ней

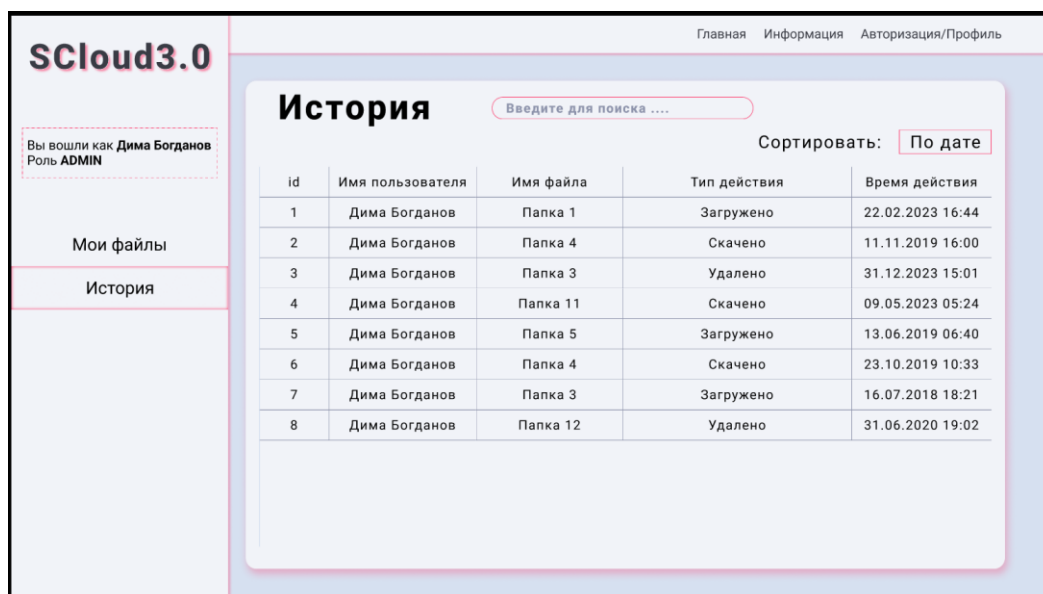


Рисунок 4.5.6 – Страница «Главная», окно «История»

4.6. Тестирование приложения

Сервер, на котором будет установлено приложение, имеет следующие технические характеристики:

- Операционная система: Ubuntu 18.0;
- Процессор: 1 ядро;
- Оперативная память: 1 ГБ;
- Жесткий диск: 30 ГБ;
- Сетевое подключение: 100 Мбит/с Ethernet;
- Дополнительные программы: установлен docker;
- Безопасность: включено защищенное соединение HTTPS.

Для тестирования, веб-интерфейс был запущен на разных устройствах с различными техническими характеристиками. Данные, полученные в результате тестирования веб-интерфейса представлены в таблице в приложении В

В результате проведенного тестирования веб-интерфейса на разных устройствах можно сделать следующие выводы:

1. Отзывчивость и адаптивность интерфейса: веб-интерфейс успешно адаптируется к разным размерам экранов и устройствам, обеспечивая правильное отображение контента и корректное расположение элементов. Это позволяет пользователям комфортно использовать приложение независимо от устройства, на котором они работают.
2. Совместимость с браузерами: веб-интерфейс хорошо совместим с различными веб-браузерами, обеспечивая одинаковую функциональность и отображение на разных платформах. Пользователи могут использовать приложение с любым предпочитаемым им браузером без ограничений.
3. Операционные системы: веб-интерфейс успешно работает на разных операционных системах, включая Windows, Linux и Android. Это обеспечивает гибкость и доступность для пользователей на разных платформах.
4. Разрешения экрана: веб-интерфейс адаптируется к различным разрешениям экрана, обеспечивая правильное отображение контента и сохраняя читабельность элементов интерфейса на устройствах с разной плотностью пикселей.

5. Ввод и взаимодействие: функциональность ввода и взаимодействия с элементами интерфейса работает надлежащим образом на разных устройствах. Пользователи могут удобно взаимодействовать с кнопками, формами, ссылками и другими интерактивными элементами приложения.
6. Скорость загрузки: веб-интерфейс демонстрирует хорошую производительность на разных устройствах, обеспечивая быструю загрузку страниц и отзывчивость интерфейса. Это способствует плавному и эффективному использованию приложения.
7. Навигация и пользовательский опыт: навигация по интерфейсу интуитивно понятна и удобна на разных устройствах. Пользовательский опыт соответствует ожиданиям пользователей, что способствует легкому и приятному взаимодействию с приложением.

Тестирование на разных устройствах позволило убедиться в правильной работе и качестве веб-интерфейса, а также обеспечить удовлетворительный пользовательский опыт для всех пользователей, независимо от их выбора устройства и платформы.

4.7. Руководство по установке приложения

Для установки приложения, нужно сделать следующее:

Шаг 1: Предварительные требования

- Убедиться, что на компьютере установлены Docker и Docker Compose. Если они не установлены, следовать инструкциям на официальном сайте Docker (<https://www.docker.com/>) для операционной системы и установите их.

Шаг 2: Загрузка и настройка проекта

1. Скачать проект с репозитория Git или получите его из другого источника.
2. Разархивировать скачанный проект на компьютере.

Шаг 3: Конфигурация приложения

1. Открыть файл настроек `.env` в корневой папке проекта.

2. Настроить значения переменных окружения в соответствии с предпочтениями. Внимательно прочитать комментарии в файле `.env` для понимания каждой переменной и ее значения.
3. Сохранить файл `.env`.

Шаг 4: Запуск приложения

1. Открыть командную строку или терминал и перейти в корневую папку проекта.
2. Выполнить команду **`docker-compose up -d`**. Docker Compose автоматически загрузит и настроит контейнеры для приложения, согласно определенным настройкам в файле `docker-compose.yml`. Пример настройки Docker представлен в приложении А.
3. Подождать, пока Docker загрузит и настроит все контейнеры. Вывод командной строки должен показать успешное создание и запуск контейнеров.
4. После успешного запуска контейнеров, приложение будет доступно по адресу `http://localhost` в веб-браузере.

Шаг 5: Использование приложения

1. Открыть веб-браузер и перейдите по адресу `http://localhost`.
2. Если выводится страница авторизации, ввести свои учетные данные для входа в систему.
3. После успешной авторизации будет перенаправление на главную страницу приложения, где можно взаимодействовать с функциональностью приложения.

Теперь приложение должно быть установлено и готово к использованию. Можно начать работу с приложением, загружать файлы, взаимодействовать с функциями и наслаждаться его возможностями.

Важно помнить, что эта инструкция предполагает использование Docker и Docker Compose для установки и запуска приложения. Если возникли проблемы или вопросы, обратиться к документации Docker или обратиться к разработчикам приложения для получения дополнительной поддержки.

4.8. Руководство пользователя

Ниже представлено руководство пользователя, которое поможет ознакомиться с функциональностью и основными возможностями приложения.

1. Регистрация и вход в систему:

- Чтобы начать использовать приложение, необходимо создать учетную запись. На странице авторизации нажмите кнопку "Зарегистрироваться" и заполнить необходимые поля.
- После успешной регистрации нужно ввести свои учетные данные (электронную почту и пароль) на странице авторизации, чтобы войти в систему.

2. Главная страница:

- После успешного входа происходит перенаправление на главную страницу приложения. Здесь можно найти основные функции и навигацию по приложению.
- Возможности главной страницы включают просмотр списка файлов, создание новых папок, загрузку файлов на сервер, редактирование и удаление файлов.

3. Навигация и использование функций:

- Для перемещения между разделами приложения нужно использовать боковую панель навигации. Здесь можно найти ссылки на различные разделы, такие как Мои файлы, История и др.
- Для загрузки нового файла на сервер нужно выбрать соответствующую опцию загрузки и выбрать файлы с компьютера.
- Для создания новой папки нужно выбрать опцию "Создать папку" и ввести имя новой папки.
- Чтобы открыть файл или папку, нужно нажать на них.

4. Управление файлами и папками:

- Можно редактировать или удалить файлы и папки, используя соответствующие функции на странице просмотра файла или папки.

- Чтобы удалить файл или папку, нужно выбрать опцию "Удалить" и подтвердить действие.

5. Безопасность и конфиденциальность:

- Приложение обеспечивает безопасность информации с помощью аутентификации и авторизации.
- Для обеспечения безопасности учетной записи рекомендуется использовать надежный пароль и не передавать данные для входа третьим лицам.

6. Выйти из системы:

- Чтобы выйти из системы, нужно перейти на страницу профиля и нажать кнопку "Выйти".

Руководство пользователя поможет легко ориентироваться в приложении и использовать его функциональность для удобного и эффективного управления файлами.

4.9. Выводы по четвертому разделу

1. Созданная структура веб-интерфейса, обеспечивает удобную навигацию и доступ к функциональным возможностям приложения.
2. Реализованы основные окна приложения на основе макетов, отображающие интерфейс пользователя, предоставляют интуитивно понятные элементы управления и обеспечивают комфортное взаимодействие пользователей с приложением.
3. Проведённое тестирование продемонстрировало соответствие разработанного приложения требованиям.
4. Для работы с приложением были созданы два документа: руководство по установке приложения и руководство пользователя.

Заключение

В ходе работы было выполнено проектирование и разработка приложения для организации хранения данных на сервере.

В первом разделе были проанализированы типы систем хранения данных, проведено сравнение существующих решений и определены основные требования к разрабатываемому приложению. Это позволило лучше понять потребности пользователей и сформулировать функциональные требования.

Во втором разделе были выбраны ключевые технологии и инструменты для разработки и развертывания приложения.

В третьем разделе была выбрана архитектура приложения, разработана база данных, созданы макеты основных окон приложения. Это позволило создать четкую структуру приложения и обеспечить безопасность пользовательских данных.

В четвертом разделе было реализовано приложение в соответствии с требованиями, проведено тестирование, разработаны руководство по установке и руководство пользователя. Проведенное тестирование показало работоспособность приложения и готовность его к использованию.

Результатом работы является разработанное приложение, которое соответствует требованиям, обладает необходимой функциональностью, обеспечивает безопасность и простоту использования. Разработанные руководства помогут пользователям легко установить и начать использовать приложение. В целом, выполнение поставленных задач и принятых решений позволяет сделать вывод о полной готовности разработанного приложения для внедрения в рабочую среду и предоставления пользователям удобного и безопасного инструмента для хранения данных.

Список использованных источников

1. Статья «Все о СХД – и даже больше». [Электронный ресурс]. URL: <https://www.retail.ru/articles/vse-o-skhd-i-dazhe-bolshe/> (дата обращения: 20.05.2023)
2. Официальный сайт Битрикс24. [Электронный ресурс]. URL: <https://www.bitrix24.ru/features/> (дата обращения: 20.05.2023)
3. Официальный сайт Docsvision. [Электронный ресурс]. URL: <https://docsvision.com/ecm-bpm/> (дата обращения: 20.05.2023)
4. Официальный сайт OwnCloud. [Электронный ресурс]. URL: <https://owncloud.com/product> (дата обращения: 20.05.2023)
5. Официальный сайт Pydio. [Электронный ресурс]. URL: https://www.mivocloud.com/ru/blog/Pydio_overview_and_benefits (дата обращения: 20.05.2023)
6. Рейтинг востребованных языков программирования GitHub. [Электронный ресурс]. URL: https://madnight.github.io/githut/#/pull_requests/2023/3 (дата обращения: 22.05.2023)
7. Документация TypeScript. [Электронный ресурс]. URL: <https://www.typescriptlang.org/docs/> (дата обращения: 22.05.2023)
8. Документация PostgreSQL. [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 22.05.2023)
9. Рейтинг востребованных веб-серверов HostAdvice. [Электронный ресурс]. URL: <https://ru.hostadvice.com/marketshare/server/> (дата обращения: 22.05.2023)
10. Документация Nginx. [Электронный ресурс]. URL: <https://nginx.org/ru/docs/> (дата обращения: 22.05.2023)
11. Документация Git. [Электронный ресурс]. URL: <https://git-scm.com/book/ru/v2> (дата обращения: 22.05.2023)
12. Документация Docker. [Электронный ресурс]. URL: <https://docs.docker.com/> (дата обращения: 22.05.2023)

Приложение А

Настройка Docker

Docker представляет собой открытую платформу, которая позволяет упаковывать, развертывать и запускать приложения в изолированных контейнерах. Это обеспечивает удобство разработки, доставки и масштабирования приложений.

Настройка Docker включает несколько шагов, которые позволят создать контейнеры для каждого компонента приложения. Вот основные этапы настройки:

1. **Установка Docker:** сначала необходимо установить Docker в систему. Для этого можно посетить официальный сайт Docker и следовать инструкциям для установки, соответствующим операционной системе.
2. **Создание Dockerfile:** Dockerfile — это текстовый файл, содержащий инструкции для создания образа Docker. В нем указываются зависимости, настройки и команды, необходимые для настройки и запуска приложения в контейнере. Необходимо создать Dockerfile для каждого компонента приложения, таких как React, Node.js сервер, Nginx и PostgreSQL. В Dockerfile определить базовый образ, установить зависимости, скопировать необходимые файлы и настройки, а также определить команду для запуска приложения. Пример Dockerfile для Node.js сервера представлен на рисунке А.1.

```
1  # Используем базовый образ Node.js
2  FROM node:14-alpine
3
4  # Устанавливаем рабочую директорию внутри контейнера
5  WORKDIR /app
6
7  # Копируем package.json и package-lock.json в контейнер
8  COPY package*.json ./
9
10 # Устанавливаем зависимости
11 RUN npm install
12
13 # Копируем все файлы сервера в контейнер
14 COPY . .
15
16 # Открываем порт, на котором будет работать сервер
17 EXPOSE 3000
18
19 # Запускаем сервер при старте контейнера
20 CMD ["npm", "start"]
```

Рисунок А.1 – Пример кода Dockerfile

3. **Сборка образа Docker:** следующий шаг - сборка образа Docker на основе созданного Dockerfile. Для этого выполните команду «**docker build**» с указанием пути к Dockerfile и желаемому имени образа. Docker выполнит инструкции из Dockerfile и создаст образ, включающий все необходимые компоненты вашего приложения.
4. **Создание контейнера:** после успешной сборки образа нужно создать контейнеры на его основе. Контейнеры представляют собой запущенные экземпляры образов Docker. Нужно использовать команду «**docker run**» для создания контейнера, указав имя образа, необходимые параметры и порты для взаимодействия с контейнером.
5. **Сетевая настройка:** если в приложении требуется взаимодействие между контейнерами, необходимо настроить сетевое взаимодействие между ними. Docker предоставляет различные возможности для создания сетей и связывания контейнеров. Связи прописываются в файле Docker-compose.yml. Пример такого файла представлен на рисунке А.2.

```

1  version: '3'
2  services:
3    nodejs-server:
4      build:
5        context: ./path/to/nodejs-server
6      ports:
7        - 3000:3000
8      environment:
9        - NODE_ENV=production
10     depends_on:
11       - db
12
13     nginx:
14       image: nginx:latest
15       ports:
16         - 80:80
17       volumes:
18         - ./path/to/nginx.conf:/etc/nginx/nginx.conf:ro
19       depends_on:
20         - nodejs-server
21
22     db:
23       image: postgres:latest
24       environment:
25         - POSTGRES_USER=your_username
26         - POSTGRES_PASSWORD=your_password
27         - POSTGRES_DB=your_database
28       volumes:
29         - ./path/to/database:/var/lib/postgresql/data
30

```

Рисунок А.2 – Пример кода Docker-compose.yml

6. Управление контейнерами: Docker предоставляет команды для управления контейнерами, такие как запуск, остановка, перезапуск и удаление. Можно использовать эти команды для управления контейнерами в процессе разработки и эксплуатации приложения.

Настройка Docker позволяет легко развернуть приложение в изолированных контейнерах, обеспечивая удобство разработки. Это также облегчает развертывание и масштабирование приложения на различных средах.

Приложение Б

Блок-схема алгоритма авторизации

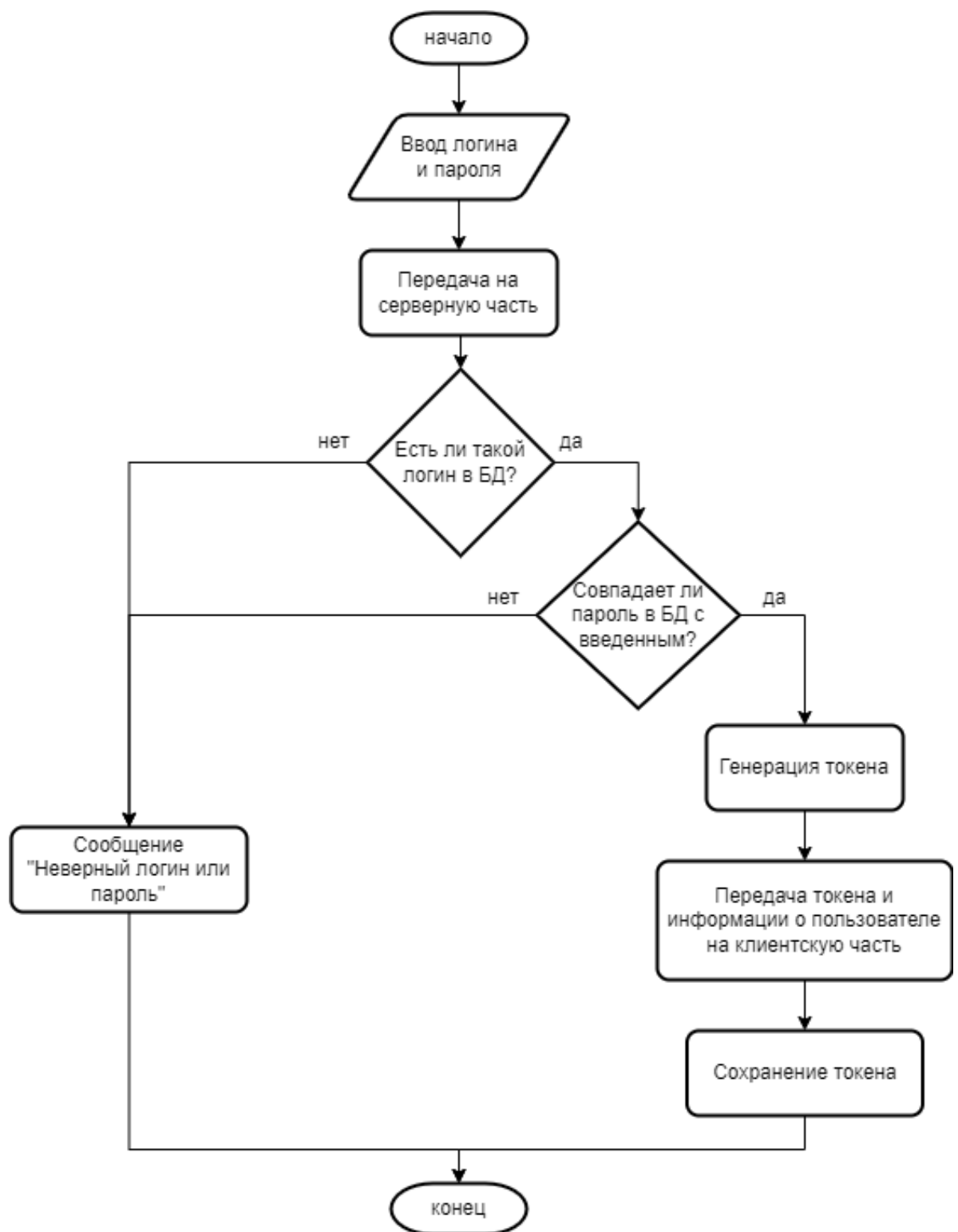


Рисунок Б.1 – Блок-схема алгоритма авторизации

Приложение В

Результаты тестирования

Устройства	Отзывчивость	Совместимость с браузерами	Операционные системы	Разрешениями экрана	Ввод и взаимодействие	Скорость загрузки страниц	Навигация
Компьютер 1 Процессор: intel pentium g4560, 2 ядра Оперативная память: 8ГБ Видеокарта: NVIDIA GTX 1050 2ГБ Жесткий диск: 1ТБ Операционная система: Windows 11 Браузер: Microsoft Edge Сетевое подключение: 100 Мбит/с Разрешение экрана: 2560x1440	+	+	+	+	+	+	+
Компьютер 2 Процессор: AMD Ryzen 5, 6 ядер Оперативная память: 16ГБ Видеокарта: NVIDIA GTX 3060Ti 8ГБ Жесткий диск: 1ТБ Операционная система: Windows 10 Браузер: Google Chrome 113.0.5672.92 Сетевое подключение: 1Гбит/с Разрешение экрана: 3840x2160	+	+	+	+	+	+	+
Ноутбук 1 Процессор: AMD Ryzen 5, 6 ядер Оперативная память: 8ГБ Видеокарта: AMD Radeon Жесткий диск: 512ГБ Операционная система: Manjaro Linux 22.0.1 Браузер: Yandex browser 23.3.4.603 Сетевое подключение: 30 Мбит/с Разрешение экрана: 1920x1080	+	+	+	+	+	+	+
Смартфон 1 Процессор: Qualcomm Snapdragon 712, 8 ядер Оперативная память: 6ГБ Видеокарта: Adreno 616 Жесткий диск: 128ГБ Операционная система: Android 9.0 Браузер: Yandex browser mobile Сетевое подключение: 80 Мбит/с Разрешение экрана: 2340x1080	+	+	+	+	+	+	+

Рисунок В.1 – Результаты тестирования