Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ОБЛАЧНОЕ ХРАНИЛИЩЕ С ПОДДЕРЖКОЙ ВЕРСИОНИРОВАНИЯ ФАЙЛОВ**

БГУИР КП 6-05-0612-01-020 ПЗ

Студент Касперец Е.А.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199096055)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199096056)

[1.1 Обзор аналогов 6](#_Toc199096057)

[1.2 Постановка задачи 8](#_Toc199096058)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9](#_Toc199096059)

[2.1 Структура программы 9](#_Toc199096060)

[2.2 Проектирование интерфейса программного средства 10](#_Toc199096061)

[2.3 Проектирование функционала программного средства 14](#_Toc199096062)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 19](#_Toc199096063)

[3.1 Управление файлами 19](#_Toc199096064)

[3.2 Работа с папками и структурой данных 21](#_Toc199096065)

[3.3 Аутентификация и безопасность 22](#_Toc199096066)

[3.4 Работа с базой данных 24](#_Toc199096067)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 26](#_Toc199096068)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 27](#_Toc199096069)

[5.1 Интерфейс программного средства 27](#_Toc199096070)

[5.2 Главное окно пользователя 27](#_Toc199096071)

[5.3 Управление папками 28](#_Toc199096072)

[ЗАКЛЮЧЕНИЕ 31](#_Toc199096073)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32](#_Toc199096074)

[ПРИЛОЖЕНИЕ А. Исходный код программы 33](#_Toc199096075)

# ВВЕДЕНИЕ

Облачные технологии стали неотъемлемой частью современной цифровой экосистемы, предоставляя пользователям удобные инструменты для хранения, управления и совместного использования данных. Облачные хранилища обеспечивают высокий уровень доступности информации, позволяя получить к ней доступ с любого устройства, подключенного к интернету. В отличие от традиционных способов хранения данных, облачные решения предлагают отказоустойчивость, распределенную инфраструктуру и механизмы защиты от потери файлов. Благодаря этим преимуществам такие сервисы, как Google Drive, Dropbox и OneDrive, получили широкое распространение среди пользователей и бизнеса.

Одной из ключевых функций современных облачных хранилищ является поддержка версионирования файлов. Этот механизм позволяет автоматически сохранять несколько версий одного документа, что дает возможность отслеживать изменения, откатываться к предыдущим версиям и восстанавливать случайно удаленные данные. Версионирование особенно полезно в условиях командной работы, когда необходимо зафиксировать историю изменений и избежать конфликтов при редактировании. Однако не все существующие сервисы предоставляют гибкие настройки управления версиями, что делает актуальной задачу создания собственного облачного хранилища с поддержкой версионирования.

Разработка индивидуального решения позволяет реализовать гибкое хранение данных, расширенные механизмы безопасности и индивидуальные настройки доступа. В отличие от стандартных облачных сервисов, собственная система включает в себя автоматическое удаление устаревших версий, возможность восстановления данных по временным меткам и интеграцию с различными внешними платформами. Это особенно важно для компаний и частных пользователей, которым требуется повышенный контроль над данными и защита от несанкционированных изменений.

Целью данного курсового проекта является разработка распределенного облачного хранилища с поддержкой версионирования файлов. В рамках работы будут изучены основные принципы хранения данных, методы организации структуры базы данных, механизмы работы с версиями файлов, а также способы защиты информации и аутентификации пользователей. Реализация проекта позволит углубить знания в области веб-разработки, серверных технологий, работы с базами данных и взаимодействия клиент-сервер. Итогом станет функциональное облачное хранилище, которое обеспечит удобную загрузку, хранение, управление файлами с возможностью скачивания их предыдущих версий и распределенностью хранения данных.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На сегодняшний день существует множество облачных хранилищ, предоставляющих пользователям возможность безопасного хранения и управления файлами. Каждое из них обладает собственными особенностями, функциональными возможностями и ограничениями. В рамках данного обзора рассмотрим наиболее популярные сервисы, включая Google Drive, Dropbox и OneDrive.

В первую очередь стоит рассмотреть облачное хранилище Google Drive, так как оно является одним из самых популярных решений. Внешний вид веб-интерфейса Google Drive представлен на рисунке 1.1.

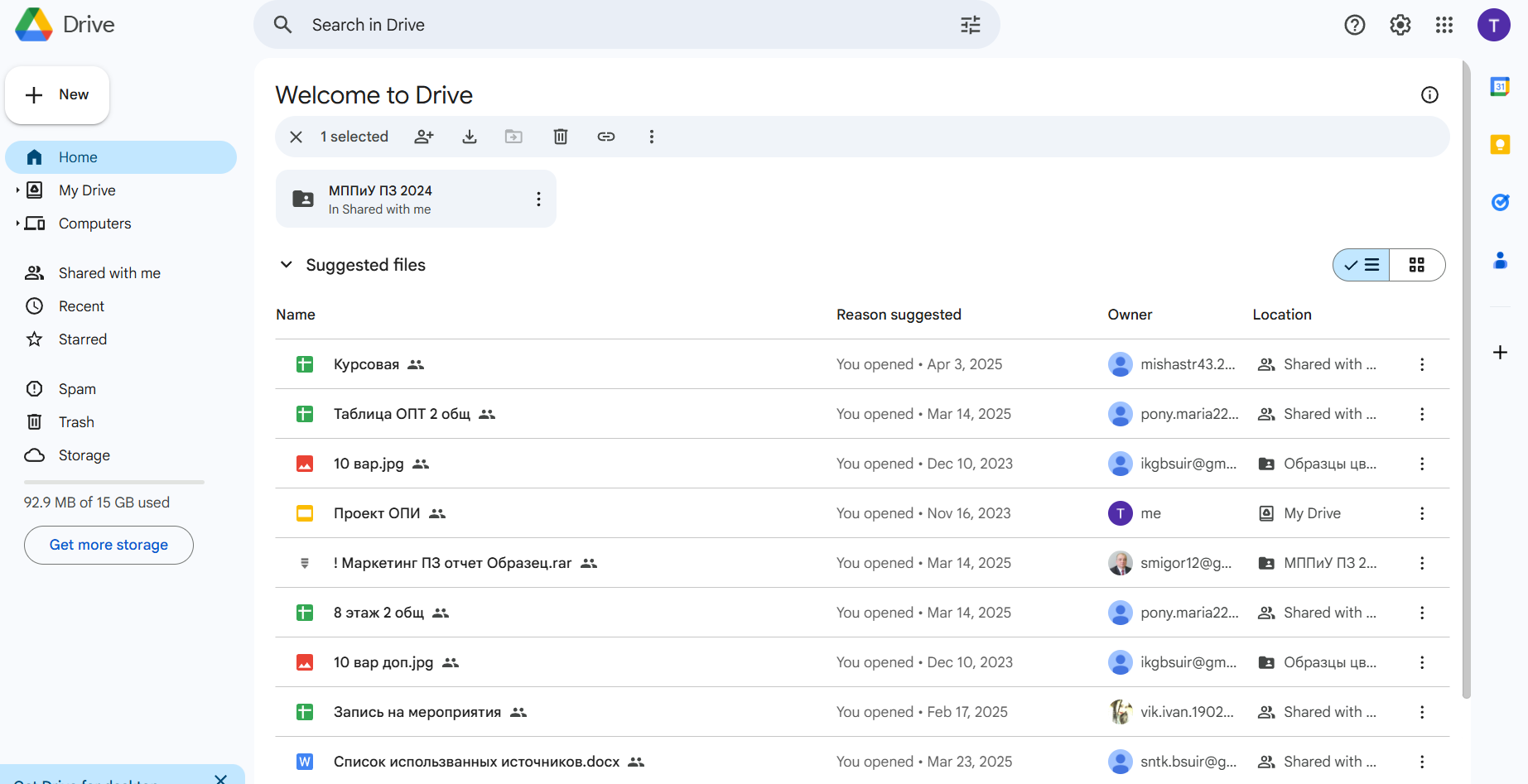


Рисунок 1.1 – Облачное хранилище Google Drive

Данный сервис предлагает пользователям 15 ГБ бесплатного пространства, синхронизацию данных между устройствами, а также интеграцию с другими сервисами Google, включая Google Docs, Sheets и Slides. Одной из ключевых функций Google Drive является поддержка версионирования файлов, что позволяет восстановить ранее сохраненные версии документов. Однако в бесплатном варианте хранение версий ограничено, и старые версии могут автоматически удаляться через определенный период.

Следующим облачным хранилищем, заслуживающим внимания, является Dropbox. Внешний вид данного приложения представлен на рисунке 1.2.

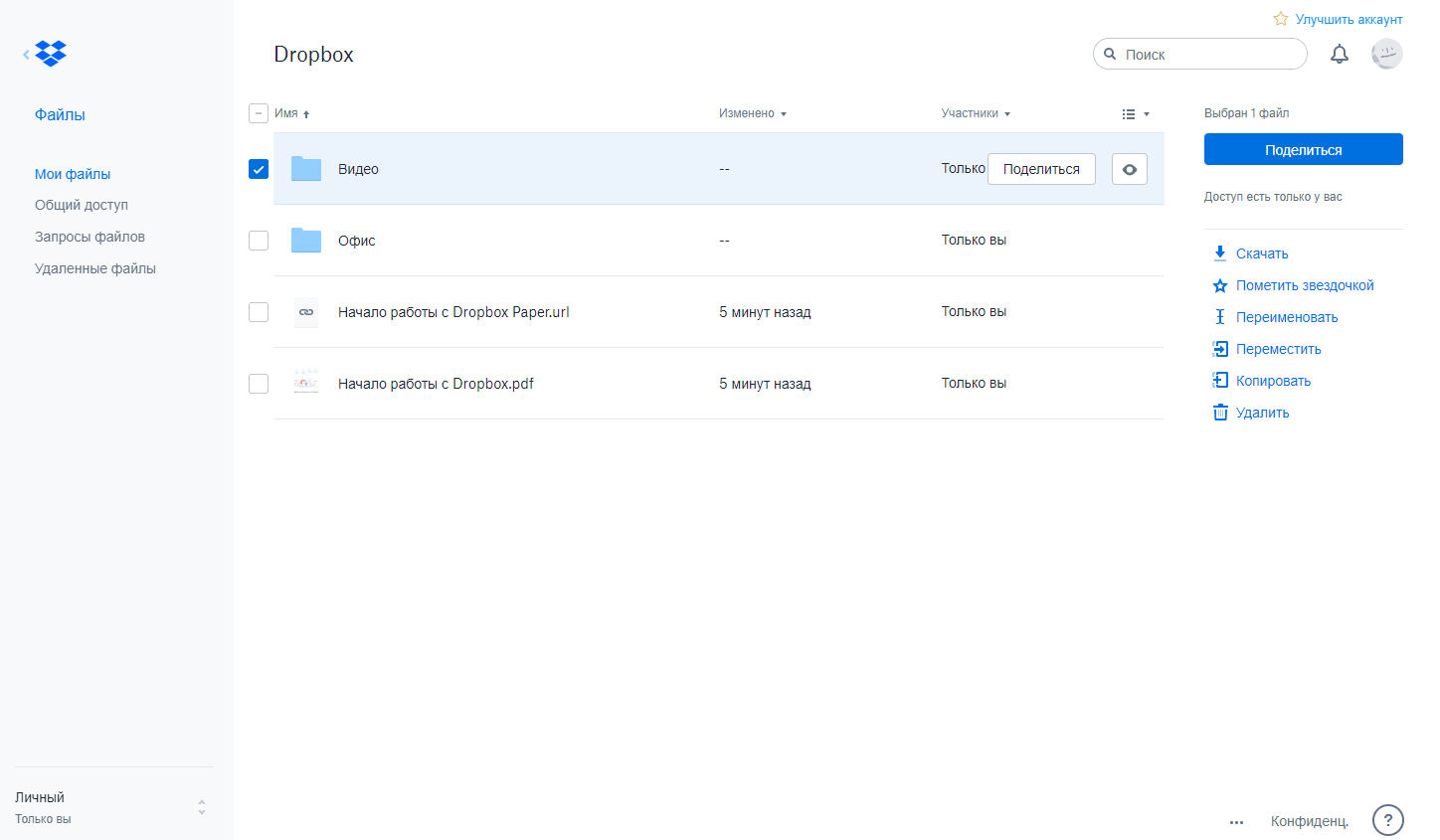


Рисунок 1.2 – Облачное хранилище Dropbox

Dropbox ориентирован на простоту использования и надежную синхронизацию файлов. Он предоставляет 2 ГБ бесплатного пространства, но позволяет расширять объем при помощи различных бонусных программ. Среди особенностей можно выделить систему интеллектуальной синхронизации, возможность восстановления файлов и поддержку истории изменений, что делает сервис удобным для командной работы. Однако функциональность версии для бесплатных пользователей ограничена, а полноценное использование требует подписки.

Еще одним крупным игроком на рынке облачных хранилищ является OneDrive от компании Microsoft. Его веб-интерфейс показан на рисунке 1.3.

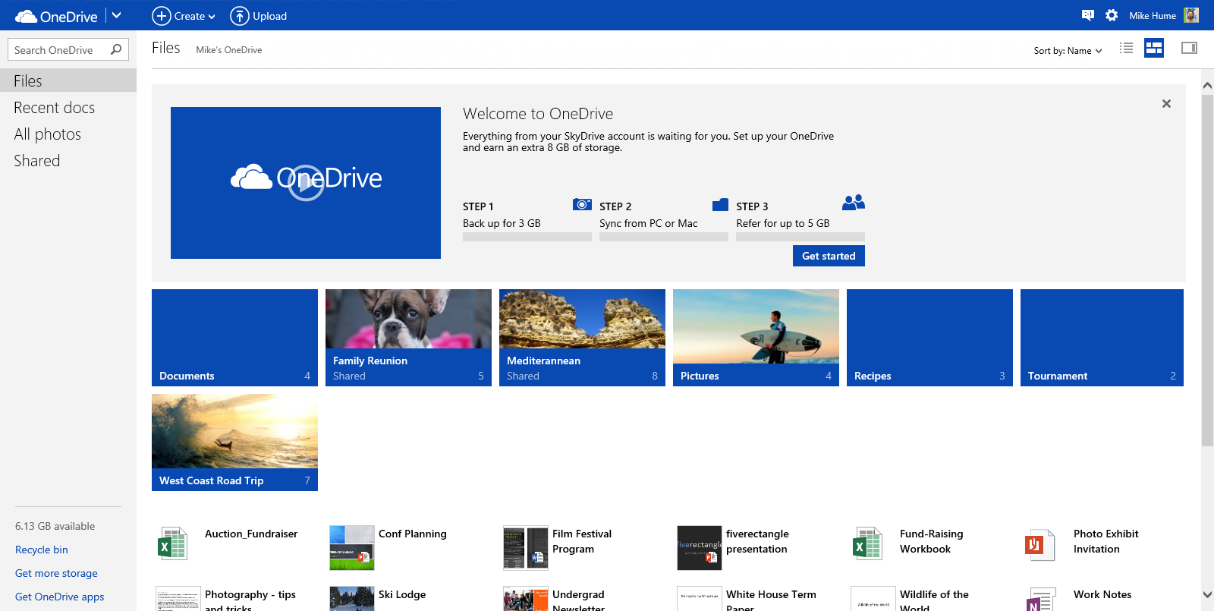


Рисунок 1.3 – Облачное хранилище OneDrive

OneDrive предоставляет 5 ГБ свободного места, а при подписке на Microsoft 365 объем увеличивается до 1 ТБ. Главным преимуществом является глубокая интеграция с продуктами Microsoft, включая Windows и Office 365. Версионирование файлов в OneDrive поддерживается, но ограничено по времени хранения, а восстановление удаленных версий доступно в течение 30 дней.

Таким образом, существующие облачные хранилища обладают схожими базовыми возможностями, но отличаются объемом бесплатного пространства, глубиной интеграции с другими сервисами и политикой хранения версий файлов. На основе анализа аналогов можно выделить ключевые требования к разрабатываемому облачному хранилищу «Net Cloud», включая поддержку гибкого версионирования файлов, возможность настройки хранения версий, удобный пользовательский интерфейс и интеграцию с различными сервисами для работы с файлами.

## Постановка задачи

В рамках данной курсовой работы необходимо разработать облачное хранилище с поддержкой версионирования файлов. Данное программное средство должно обеспечивать надежное хранение данных, удобный доступ к файлам и управление версиями.

Система должна включать следующие основные функции:

* Загрузка файлов на сервер;
* Скачивание файлов с сервера;
* Организация файлов по папкам;
* Удаление файлов и папок;
* Переименование папок;
* Автоматическое сохранение новых версий файлов с одинаковыми названиями;
* Просмотр истории изменений для каждого файла;
* Возможность скачать предыдущую версию файла;
* Каскадное удаление старых версий, когда удаляется основная;
* Регистрация и авторизация пользователей;
* Шифрование данных при передаче и хранении;
* Поиск файлов по имени и содержимому;
* Поддержка различных типов файлов (документы, изображения, видео и др.);
* Интерфейс для управления файлами и папками через веб-приложение.

Для реализации проекта будет использоваться язык программирования Python и веб-фреймворк FastAPI. В качестве базы данных для хранения информации о файлах и версиях будет использоваться PostgreSQL. Интерфейс взаимодействия с системой будет реализован в виде веб-приложения. Для поддержки распределённости хранилища и балансировки нагрузки будет использоваться модульная архитектура, предполагающая размещение файлов на разных логических узлах. Несмотря на то, что проект разрабатывается на одном физическом компьютере, система реализует принцип распределённого хранения, направляя файлы и их версии на различные локальные директории, имитирующие отдельные узлы хранения

Разрабатываемое программное средство должно обеспечивать надежность хранения данных, удобство использования и гибкость в управлении версиями файлов, что позволит пользователям эффективно работать с документами и защищать их от случайных изменений или удаления.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

При разработке приложения будет использовано 14 модулей:

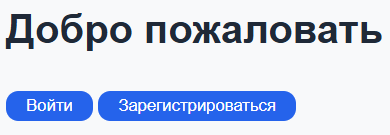
* main – модуль, для запуска и инициализации приложения;
* config – модуль, обеспечивающий работу с конфигурацией приложения;
* database – модуль, обеспечивающий подключение и взаимодействие с базой данных;
* models.file – модуль, хранящий описание модели файла;
* models.folder – модуль, хранящий описание модели папки;
* models.user – модуль, хранящий описание модели пользователя;
* models.file\_version – модуль, хранящий описание модели версий файлов;
* routers.auth – модуль, обеспечивающий маршруты для аутентификации и регистрации;
* routers.files – модуль, обеспечивающий маршруты для работы с файлами;
* routers.folders – модуль, обеспечивающий маршруты для работы с папками;
* routers.file\_version – модуль, обеспечивающий маршруты для работы с версиями файлов;
* schemas.auth – модуль, обеспечивающий схемы данных для аутентификации;
* schemas.file – модуль, обеспечивающий схемы данных для файлов;
* schemas.folder – модуль, обеспечивающий схемы данных для папок;
* schemas.user – модуль, обеспечивающий схемы данных для пользователей;
* schemas.file\_version – модуль, обеспечивающий схемы данных для версий файлов;
* services.auth\_service – модуль, обеспечивающий бизнес-логику аутентификации;
* services.file\_version\_service – модуль, обеспечивающий логику работы с версиями файлов;
* services.folder\_service – модуль, обеспечивающий логику работы с папками;
* services.storage\_node – модуль, обеспечивающий работу с физическим хранилищем файлов;
* design.static.css.style.css – модуль, обеспечивающий стилизацию интерфейса;
* templates.file.html – модуль, отображающий страницу управления файлами;
* templates.home.html – модуль, отображающий главную страницу;
* templates.login.html – модуль, отображающий страницу входа;
* templates.register.html – модуль, отображающий страницу регистрации;
* .env – модуль конфигурации для переменных окружения, таких как данные для подключения к базе данных и секретный ключ для сигнатуры jwt-токена.

## 2.2 Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят стандартный внешний вид файловых хранилищ и логика работы с файлами и папками.

2.2.1 Регистрация и вход

Окна регистрации и входа содержат поля для ввода электронной почты и пароля. Пользователь может перейти между формами входа и регистрации с помощью соответствующих ссылок. Внешний вид макетов окон регистрации и входа представлен на рисунке 2.1.



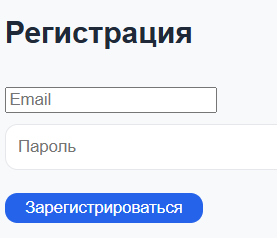
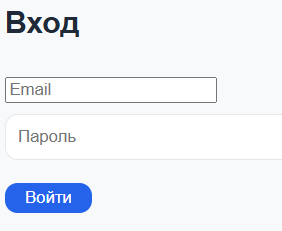


Рисунок 2.1 – Вход и регистрация

2.2.2 Главное окно

Главное окно веб-приложения представляет собой веб-страницу с основным функционалом распределенного облачного хранилища. В верхней части страницы расположена приветственная надпись и панель авторизации. Если пользователь вошёл в систему, отображается его электронная почта и кнопка выхода.

Ниже находится навигация по текущей папке и её родительской директории, а также функциональные формы и списки:

* Создание новой папки и переименование;
* Загрузка файлов в текущую папку;
* Просмотр списка папок и файлов с возможностью навигации, скачивания и удаления.

Внешний вид главного окна представлен на рисунке 2.2.

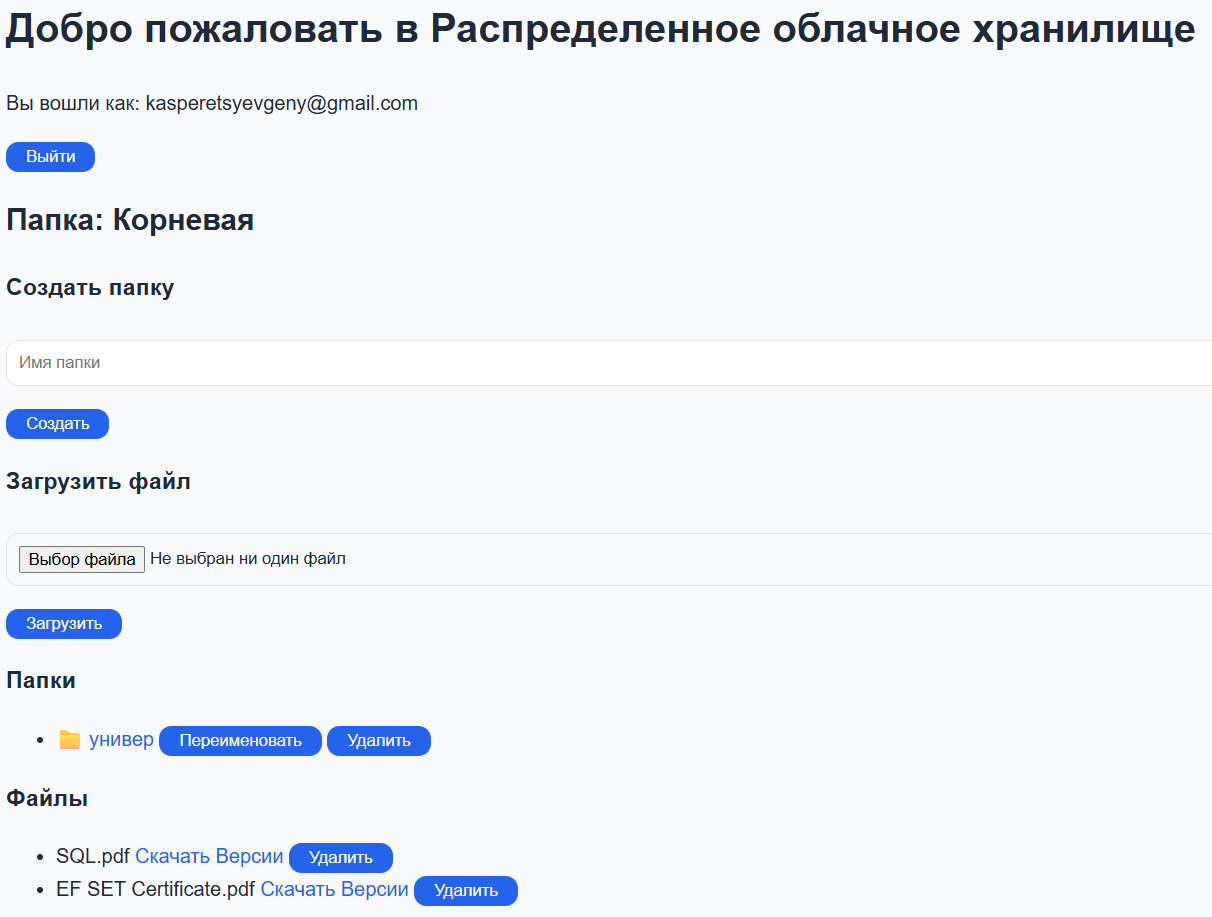


Рисунок 2.2 – Главное окно приложения

2.2.3 Элементы управления приложения

Основные элементы управления веб-интерфейсом:

* «Создать папку» – открывает форму для ввода имени новой папки и создаёт её в текущем каталоге;
* «Загрузить файл» – открывает диалог выбора файла и загружает его в текущую папку;
* «Название папки» (для папок) – переход к содержимому выбранной папки;
* «Скачать» (для файлов) – загрузка файла на устройство пользователя;
* «Удалить» – удаление выбранного файла или папки с подтверждением;
* «Версии» (для файлов) – открывает список всех доступных версий файла с возможностью просмотра или скачивания;
* «..» (назад) – возвращение к родительской папке.

Кнопка подтверждения удаления – появляется во всплывающем окне при попытке удалить элемент:

* «Подтвердить»
* «Отмена»

Внешний вид элементов управления приложением представлен на рисунке 2.3.

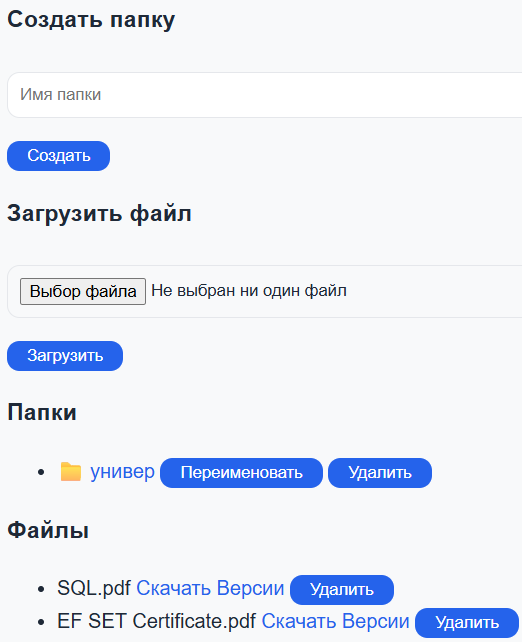
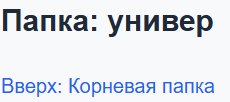
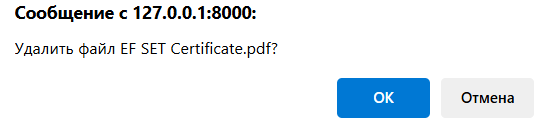
 

Рисунок 2.3 – Элементы управления приложением

2.2.4 Навигация по папкам

Пользователь может перемещаться между папками с помощью кликов по ссылкам:

* При переходе к родительской папке ссылка отображается как «Вверх: имя папки».
* В корневой папке переход вверх ведёт к корню.

Внешний вид навигации по папкам представлен на рисунке 2.4.

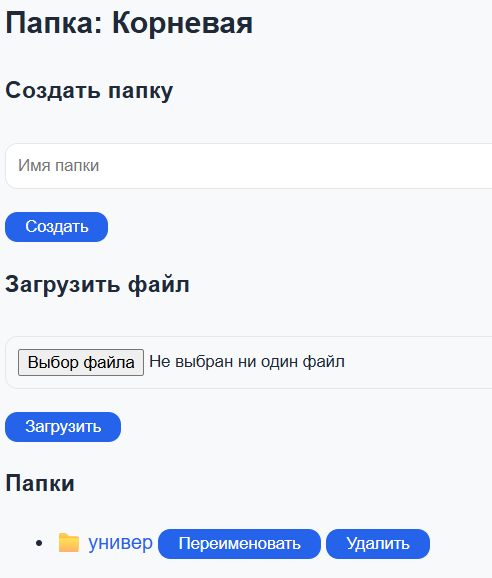
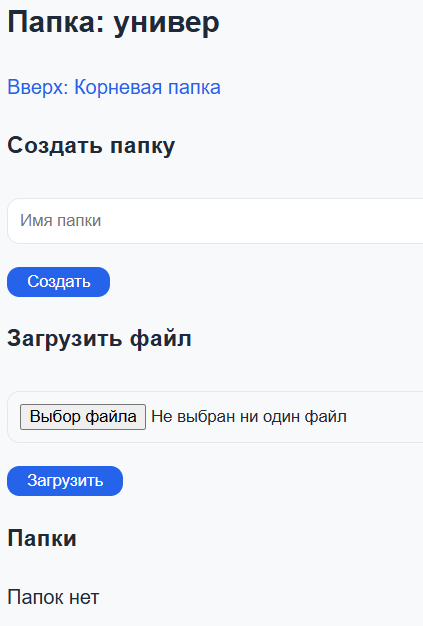
 

Рисунок 2.4 – Навигация по папкам

2.2.5 Управление версиями файлов

Программное средство будет иметь следующие возможности для работы с версиями файлов:

* Просмотр текущей версии файла;
* История версий файла:
* номера версии,
* даты загрузки каждой версии.
* Скачивание конкретной версии;
* Навигация назад.

Внешний вид макета версий файлов представлен на рисунке 2.5.

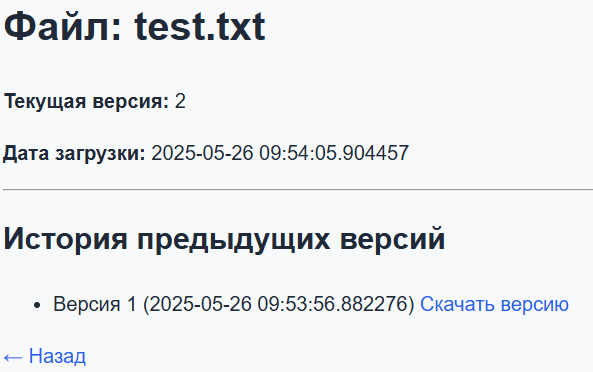


Рисунок 2.5 – Версии файлов

2.2.6 Обработка сессии и безопасности

Для предотвращения неожиданного выхода сессии в браузере реализован скрипт, предлагающий обновить access-токен за 1 минуту до его истечения (через 14 минут после входа). При отказе или ошибках происходит перенаправление на страницу входа. Внешний вид макета обновления access-токена представлен на рисунке 2.6.

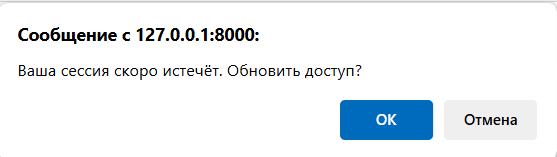


Рисунок 2.6 – Обновление access-токена

## 2.3 Проектирование функционала программного средства

Для обеспечения удобного и эффективного взаимодействия пользователя с распределённым облачным хранилищем приложение должно реализовывать следующий минимальный набор функций:

* Авторизация и регистрация пользователей;
* Навигация по папкам;
* Создание и удаление папок;
* Загрузка, скачивание, удаление файлов;
* Просмотр версий файлов;
* Обновление сессии (токена доступа);
* Распределенность хранения данных

2.3.1 Авторизация и регистрация пользователей

Пользователь может зарегистрироваться, указав электронную почту и пароль, после чего войти в систему. При входе выполняется проверка корректности введённых данных. В случае успешной аутентификации сервер возвращает JWT-токены для управления сессией. В случае истечения срока действия токена пользователю предлагается обновить его без повторного ввода данных. Блок-схема метода, который осуществляет регистрацию пользователей представлена на рисунке 2.7.

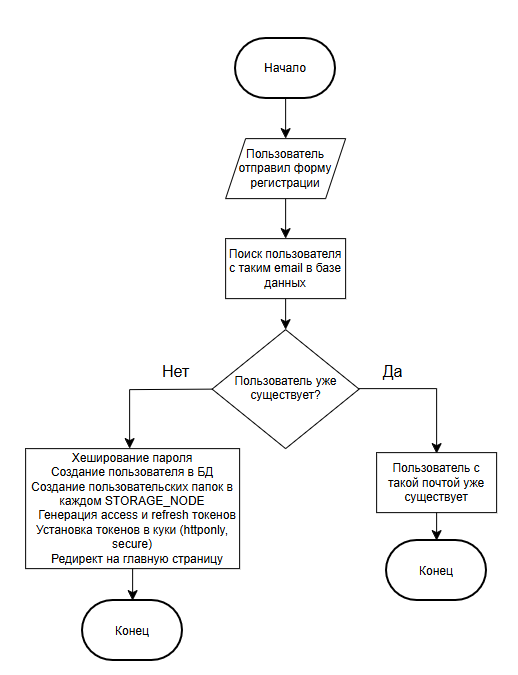


Рисунок 2.7 – Блок-схема регистрации

2.3.2 Навигация по папкам

Пользователь при входе попадает в корневую папку облачного хранилища. Для перехода между папками используются ссылки в интерфейсе.

При переходе в выбранную папку система отображает список её подпапок и файлов. Переход «вверх» ведёт к родительской папке, либо, если текущая папка – корневая, ссылка на корень. Блок-схема метода навигации по папкам представлена на рисунке 2.8.

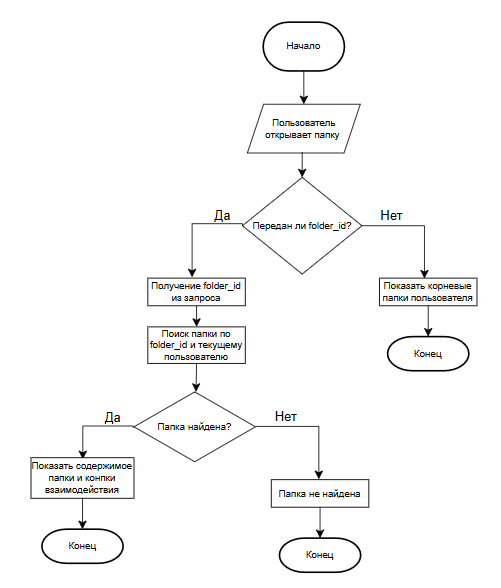


Рисунок 2.8 – Блок-схема метода навигации по папкам

2.3.3 Создание и удаление папок

Создание новой папки осуществляется через форму с указанием имени и отправкой запроса на сервер. При удалении папки с помощью кнопки «Удалить» появляется окно подтверждения. Удаление выполняется каскадно, то есть все находящиеся внутри файлы и папки будут удалены. Блок-схема создания папки представлена на рисунке 2.9.

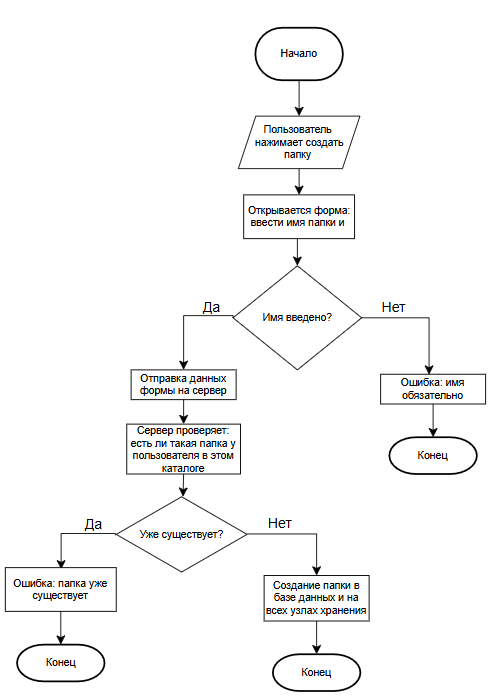


Рисунок 2.9 – Блок-схема создания папки

2.3.4 Загрузка, скачивание и удаление файлов

Для загрузки файлов предусмотрена форма с выбором файла и указанием папки назначения. При успешной загрузке файл отображается в списке текущей папки. Для скачивания файла пользователь нажимает на ссылку «Скачать». Для удаления – кнопку «Удалить» с подтверждением. Блок-схема метода, который осуществляет загрузку файлов представлена на рисунке 2.10.

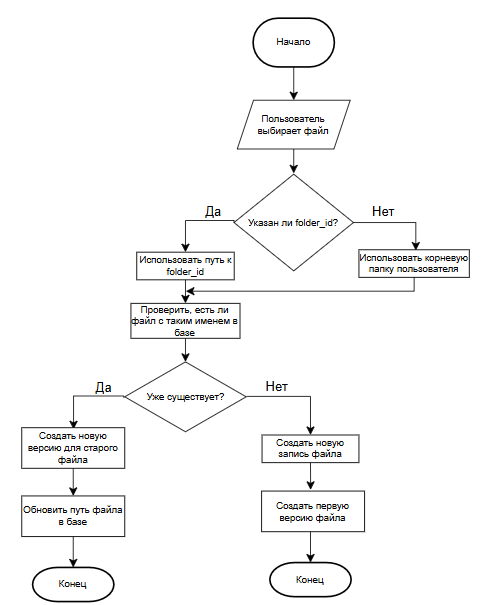


Рисунок 2.10 – Блок-схема загрузки файлов

2.3.5 Просмотр версий файлов

Для каждого файла доступна функция просмотра версий. Переход по ссылке «Версии» открывает страницу с историей изменений файла. Пользователь может просмотреть и скачать любую из доступных версий. Блок-схема метода, который осуществляет просмотр версий файлов представлена на рисунке 2.11.

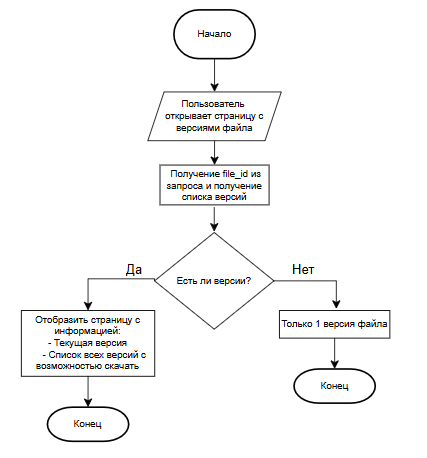


Рисунок 2.10 – Блок-схема метода просмотра версий файлов

2.3.6 Обновление сессии

Для предотвращения принудительного выхода по истечению срока действия токена реализован скрипт, предлагающий обновить токен за минуту до окончания сессии. Пользователь может подтвердить или отказаться от обновления. В случае отказа или ошибки происходит перенаправление на страницу входа. Блок-схема метода, который осуществляет обновление сессии представлена на рисунке 2.12.

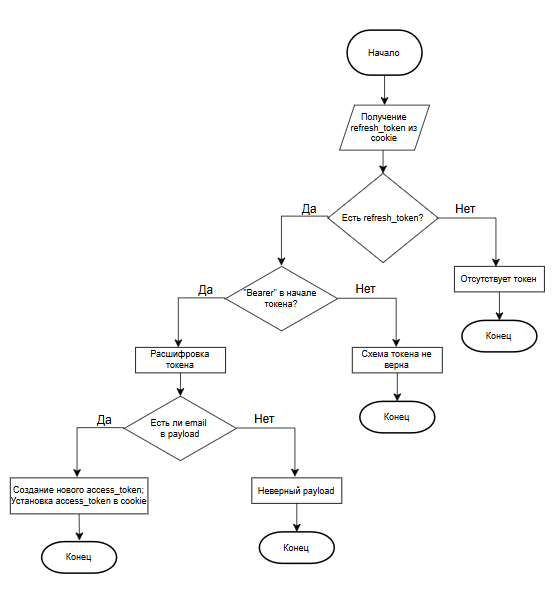


Рисунок 2.12 – Блок-схема обновления access-токена

2.3.7 Распределенность хранения данных

Для обеспечения масштабируемости и надёжности облачного хранилища в программном средстве предусмотрена возможность распределённого хранения данных. Архитектура приложения допускает использование нескольких физических узлов, на которых размещаются файлы пользователей. Модуль storage\_node абстрагирует взаимодействие с файловой системой и позволяет:

* направлять файлы на определённые узлы;
* получать файлы с нужного узла при запросе на скачивание;

Данное решение облегчает расширение системы — при увеличении числа пользователей возможно подключение дополнительных узлов без изменения основной логики API. Блок-схема распределения файлов представлена на рисунке 2.13.

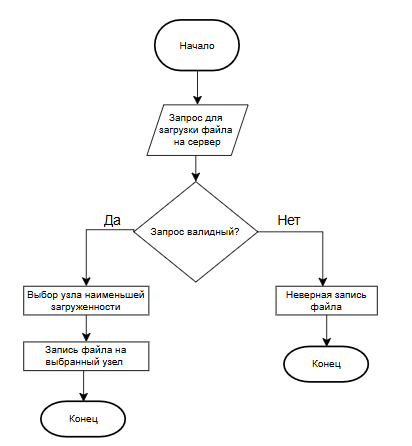


Рисунок 2.13 – Блок-схема распределения файлов

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Управление файлами

Основным функционалом системы является возможность загрузки, хранения, скачивания, удаления и управления версиями файлов. Все операции осуществляются через REST API, обеспечивающее масштабируемость и интеграцию с различными клиентами.

3.1.1 Загрузка файлов

Пользователь может загружать файлы на сервер через метод POST. Система принимает файл, сохраняет его в хранилище и создает метаданные в базе данных.

**@router.post("/upload")**

**async def upload\_file(**

**request: Request,**

**file: UploadFile = File(...),**

**folder\_id: Optional[str] = Form(default=None),**

**db: Session = Depends(get\_db),**

**user = Depends(get\_current\_user)**

**):**

**if folder\_id in (None, "", "null", "None"):**

**folder\_id\_int = None**

**else:**

**try:**

**folder\_id\_int = int(folder\_id)**

**except ValueError:**

**raise HTTPException(status\_code=400, detail="folder\_id must be an integer")**

**file\_path = save\_file(file, file.filename, user.id, db, folder\_id\_int)**

**if not file\_path:**

**raise HTTPException(status\_code=500, detail="Ошибка при сохранении файла")**

**if folder\_id\_int:**

**return RedirectResponse(url=f"/?folder\_id={folder\_id\_int}&msg=Файл загружен", status\_code=303)**

**return RedirectResponse(url="/?msg=Файл загружен", status\_code=303)**

Сначала система обрабатывает folder\_id, проверяя его корректность. Далее вызывается функция save\_file(), которая сохраняет файл в файловой системе и записывает данные в базу. После успешного завершения происходит перенаправление пользователя на главную или текущую папку.

3.1.2 Скачивание файлов

Пользователь может скачивать файлы по их идентификаторам. API возвращает содержимое файла, учитывая права доступа и безопасность.

**@router.get("/download/{file\_id}")**

**def download\_file(**

**file\_id: int,**

**db: Session = Depends(get\_db),**

**user = Depends(get\_current\_user)**

**):**

**file = get\_file\_by\_id(file\_id, db)**

**if not file:**

**raise HTTPException(status\_code=404, detail="Файл не найден")**

**return FileResponse(path=file.path, filename=file.name, media\_type="application/octet-stream")**

Метод получает объект файла по ID из базы данных. При его наличии возвращается файл через FileResponse, с указанием имени и типа. В случае отсутствия файла – клиенту отправляется ошибка 404.

3.1.3 Удаление файлов

Удаление файлов происходит по запросу пользователя через POST-запрос

**@router.post("/delete/{file\_id}")**

**def delete\_file(**

**file\_id: int,**

**db: Session = Depends(get\_db),**

**user = Depends(get\_current\_user)**

**):**

**delete\_file\_by\_id(file\_id, db)**

**return RedirectResponse(url="/?msg=Файл удален", status\_code=303)**

В первую очередь вызывается функция delete\_file\_by\_id(), которая удаляет запись из базы и файл из хранилища. После этого пользователь перенаправляется на главную страницу с сообщением об успешном удалении.

3.1.4 Версионирование файлов

Для каждого файла поддерживается история версий. При загрузке новой версии создается новая запись, что позволяет пользователю скачать предыдущие состояния файла.

**@router.post("/{file\_id}", response\_model=FileVersionOut)**

**def add\_version(**

**file\_id: int,**

**version\_data: FileVersionCreate,**

**db: Session = Depends(get\_db),**

**user=Depends(get\_current\_user)**

**):**

**return create\_file\_version(file\_id, version\_data, db)**

Система принимает данные новой версии и сохраняет их в таблице версий, связывая с конкретным файлом. Это обеспечивает возможность скачивания или анализа предыдущих состояний.

## 3.2 Работа с папками и структурой данных

Для удобства хранения и навигации по файлам реализована поддержка каталогов.

3.2.1 Создание, удаление и переименование папок

Пользователь может создавать и удалять папки, а также изменять их названия через соответствующие API вызовы. В качестве примера – функция создания папки.

@router.post("/", response\_model=FolderRead, status\_code=status.HTTP\_201\_CREATED)

def create\_new\_folder(

    name: str = Form(...),

    parent\_id: Optional[str] = Form(None),

    db: Session = Depends(get\_db),

    current\_user=Depends(get\_current\_user)

):

    if parent\_id == "" or parent\_id is None:

        parent\_id\_int = None

    else:

        try:

            parent\_id\_int = int(parent\_id)

        except ValueError:

            raise HTTPException(status\_code=400, detail="parent\_id must be an integer")

    folder\_data = FolderCreate(name=name, parent\_id=parent\_id\_int)

    create\_folder(db, folder\_data, owner\_id=current\_user.id)

    if parent\_id\_int in (None, 0):

        redirect\_url = "/"

    else:

        redirect\_url = f"/?folder\_id={parent\_id\_int}"

    return RedirectResponse(url=redirect\_url, status\_code=status.HTTP\_303\_SEE\_OTHER)

Метод проверяет значение parent\_id, после чего формирует объект FolderCreate. Папка сохраняется в базе данных с привязкой к владельцу. В завершение выполняется редирект на нужную папку.

3.2.2 Навигация по структуре папок

Клиенты могут запрашивать список содержимого папки, получать список файлов и папок, что облегчает навигацию по файловому хранилищу. Далее представлена функция вывода всех папок.

@router.get("/", response\_model=List[FolderRead])

def read\_folders(db: Session = Depends(get\_db), current\_user=Depends(get\_current\_user)):

    folders = get\_folders(db, owner\_id=current\_user.id)

    return folder

Функция get\_folders() возвращает список всех папок, принадлежащих текущему пользователю. Это позволяет построить иерархическое дерево каталога на фронтенде.

## 3.3 Аутентификация и безопасность

Для защиты данных и обеспечения конфиденциальности реализованы механизмы аутентификации и авторизации.

**3.3.1** Регистрация пользователя

Пользователи проходят процесс регистрации с валидацией данных и последующей авторизацией через JWT токены. Дальше представлена функция регистрации.

@router.post("/register", response\_class=HTMLResponse)

def register\_user(

    request: Request,

    email: str = Form(...),

    password: str = Form(...),

    db: Session = Depends(get\_db)

):

    existing\_user = db.query(User).filter(User.email == email).first()

    if existing\_user:

        return templates.TemplateResponse("register.html", {

            "request": request,

            "error": "Пользователь уже существует"

        })

    user = auth\_service.create\_user(db, email, password)

    access\_token = auth\_service.create\_access\_token({"sub": user.email})

    refresh\_token = auth\_service.create\_refresh\_token({"sub": user.email})

    response = RedirectResponse(url="/", status\_code=302)

    response.set\_cookie("access\_token", f"Bearer {access\_token}", httponly=True)

    response.set\_cookie("refresh\_token", f"Bearer {refresh\_token}", httponly=True, secure = True, samesite="Strict")

    return response

Метод проверяет наличие пользователя в базе. Если он не найден – создаётся новая запись, генерируются JWT токены, и они сохраняются в куки. Пользователь сразу попадает на главную страницу.

**3.3.2** Авторизация

Вход осуществляется по email и паролю. Дальше представлена функция входа.

@router.post("/login", response\_class=HTMLResponse)

def login(

    request: Request,

    email: str = Form(...),

    password: str = Form(...),

    db: Session = Depends(get\_db)

):

    user = auth\_service.authenticate\_user(db, email, password)

    if not user:

        return templates.TemplateResponse("login.html", {

            "request": request,

            "error": "Неверный логин или пароль"

        })

    access\_token = auth\_service.create\_access\_token({"sub": user.email})

    refresh\_token = auth\_service.create\_refresh\_token({"sub": user.email})

    response = RedirectResponse("/", status\_code=302)

    response.set\_cookie("access\_token", f"Bearer {access\_token}", httponly=True)

    response.set\_cookie("refresh\_token", f"Bearer {refresh\_token}", httponly=True, secure = True, samesite="Strict" )

    return response

Метод идентифицирует пользователя с помощью auth\_service, генерирует JWT токены, сохраняет их в HttpOnly куки и перенаправляет пользователя на главную страницу при успешной авторизации.

## 3.4 Работа с базой данных

Вся информация о пользователях, файлах, папках, версиях и сессиях хранится в реляционной базе данных (PostgreSQL).

**3.4.1** Структура базы данных

База данных состоит из следующих ключевых таблиц:

* users – содержит информацию о пользователях (электронная почта, хэш пароля, ID);
* files – основная таблица для хранения данных о файлах: путь, имя, владелец, дата загрузки;
* folders – реализует иерархическую структуру папок с возможностью указания родительской папки;
* file\_versions – таблица для хранения информации о версиях файлов с привязкой к основному файлу.

В качестве примера представлена модель user:

**class User(Base):**

**\_\_tablename\_\_ = "users"**

**id = Column(Integer, primary\_key=True, index=True)**

**email = Column(String, unique=True, nullable=False, index=True)**

**password\_hash = Column(String, nullable=False)**

**files = relationship(**

**"File",**

**back\_populates="owner",**

**cascade="all, delete-orphan",**

**passive\_deletes=True**

**)**

**folders = relationship(**

**"Folder",**

**back\_populates="owner",**

**cascade="all, delete-orphan",**

**passive\_deletes=True**

**)**

Структура базы данных логично отражает ключевые сущности системы, обеспечивая чёткую организацию данных и поддерживая механизмы связей между пользователями, файлами, папками и их версиями. Благодаря использованию SQLAlchemy ORM становится возможным управление данными на уровне Python-кода без необходимости ручного написания SQL-запросов.

**3.4.2** Взаимодействие с базой данных в коде

Для взаимодействия с PostgreSQL в проекте используется связка SQLAlchemy, Pydantic и FastAPI. Основной доступ к базе данных реализован через внедрение зависимости Depends(get\_db), что гарантирует корректное создание и закрытие сессии SQLAlchemy для каждого запроса. Реализация функции get\_db:

**def get\_db():**

**db = SessionLocal()**

**try:**

**yield db**

**finally:**

**db.close()**

Также в проекте применяется Alembic – инструмент для управления миграциями схемы базы данных. С его помощью отслеживаются изменения в моделях SQLAlchemy и автоматически формируются миграции, которые можно применять пошагово, обеспечивая согласованность и контроль версий структуры базы данных.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В процессе тестирования распределённого облачного файлового хранилища были выявлены и устранены различные ошибки и недочёты, касающиеся как клиентской, так и серверной части программного средства.

Одной из основных проблем, обнаруженных на раннем этапе тестирования, была возможность создания нескольких папок с одинаковым именем в пределах одной родительской директории. Это приводило к путанице при отображении содержимого папок и при обращении к ним на файловой системе. Для решения данной проблемы была реализована проверка уникальности имени папки в пределах родительского каталога и одного пользователя. При попытке создать папку с уже существующим именем в том же каталоге пользователю возвращается сообщение об ошибке, и создание блокируется на уровне серверной логики:

**existing\_folder = db.query(Folder).filter\_by(**

**name=folder.name,**

**parent\_id=parent\_id,**

**owner\_id=owner\_id**

**).first()**

**if existing\_folder:**

**raise HTTPException(status\_code=400, detail="Папка с таким именем уже существует в этом каталоге.")**

В процессе тестирования распределённого файлового хранилища была проведена проверка основных функций и поведения системы при различных пользовательских сценариях:

* Создание, удаление и навигация по папкам: проверена корректность операций, включая отображение родительских и вложенных каталогов.
* Сохранение на все узлы хранилища и доступны ли по ссылке скачивания.
* Отображение списка всех версий файла, реализована возможность загрузки предыдущих версий.
* Всплывающие уведомления и подтверждения: протестировано отображение сообщений об успешных действиях, а также подтверждение удаления.
* Автоматическое обновление JWT-токена: через 14 минут пользователю предлагается обновить токен, предотвращая принудительный выход.
* Интерфейс авторизации и регистрации: проверена работа форм входа и регистрации, валидация полей и возврат сообщений об ошибках (например, при уже зарегистрированном email).

Большинство проблем возникало на этапе проектирования и были успешно устранены в процессе модульного и ручного тестирования. Финальная версия программного средства обеспечивает устойчивую работу, обработку исключений и корректную логику взаимодействия пользователя с файловой системой.

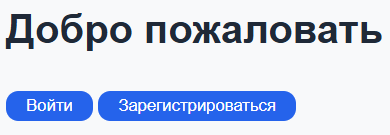
# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 Окно регистрации и авторизации

При первом запуске веб-приложения пользователю отображается приветственное сообщение и кнопки регистрации и входа. При нажатии на кнопку регистрации пользователь заполняет форму регистрации, формы регистрации и авторизации включают в себя:

* адрес электронной почты;
* пароль.



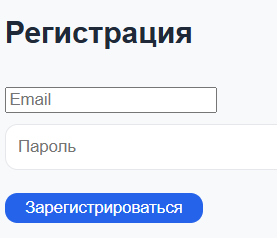
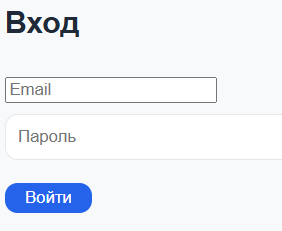


Рисунок 5.1 – Форма регистрации и авторизации

## 5.2 Главное окно пользователя

После авторизации пользователю доступен основной интерфейс работы

с файлами и папками.

5.2.1 Структура интерфейса

Интерфейс состоит из следующих элементов:

* Заголовок с приветствием и отображением текущего пользователя;
* Кнопка выхода из аккаунта;
* Название текущей папки;
* Кнопка перехода в родительскую директорию;
* Форма создания новой папки;
* Форма загрузки файла;
* Списки папок и файлов в текущей директории.

Папки и файлы отображаются в виде списков. Для каждой папки доступны действия:

* Переход внутрь;
* Переименование;
* Удаление.

Для каждого файла доступны действия:

* Скачивание;
* Просмотр версий;
* Удаление.

Далее будет представлена главная страница хранилища:

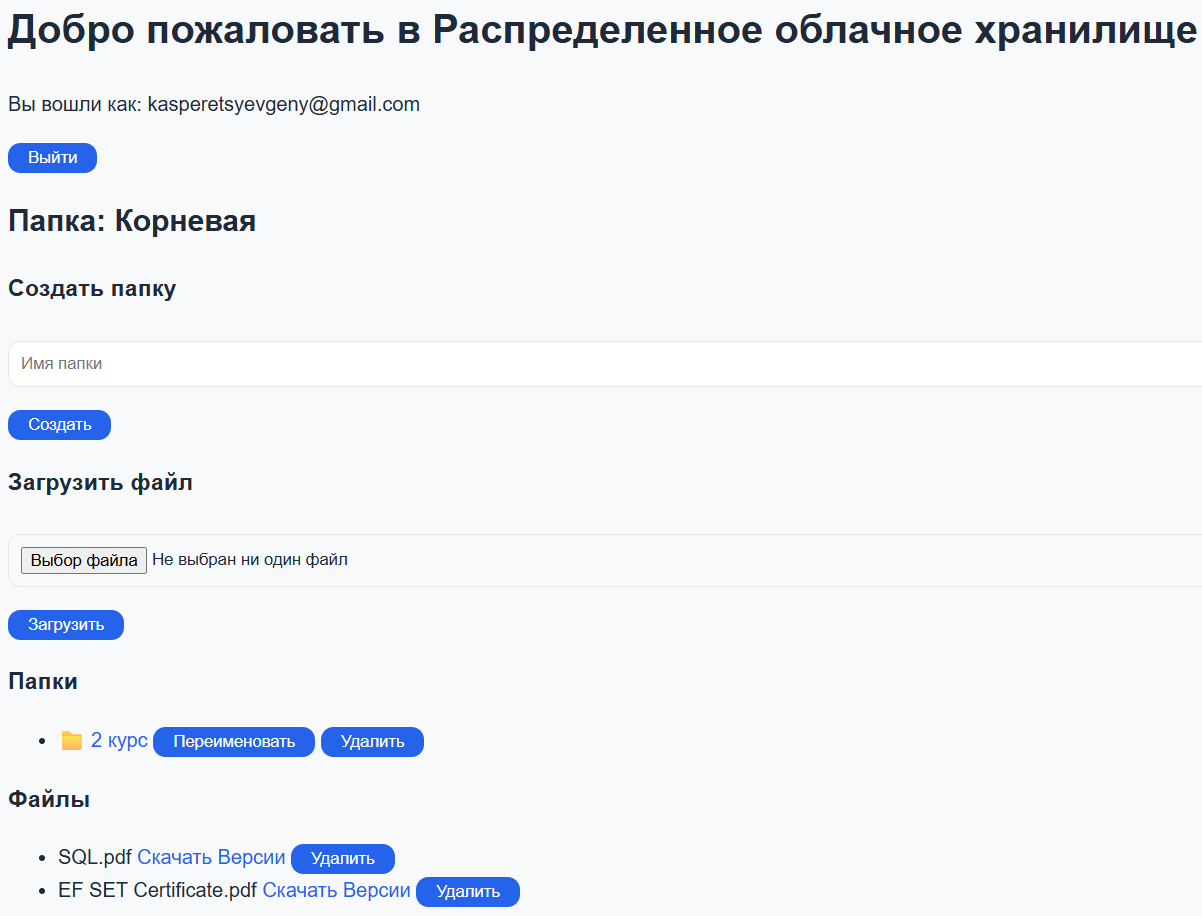


Рисунок 5.2 – Главное окно облачного хранилища

## 5.3 Управление папками

5.3.1 Переименование

Для переименования папки отображается скрытая форма, которая активируется по нажатию кнопки «Переименовать». Далее будет представлена форма переименования:

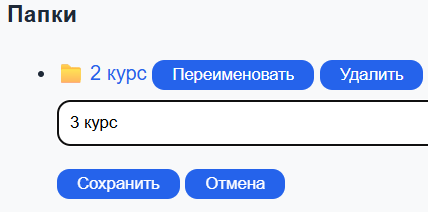


Рисунок 5.3 – Переименование папок

5.3.2 Удаление

Удаление папки выполняется через стандартную HTML-форму при нажатии на удалить, после появляется окошко подтверждения:

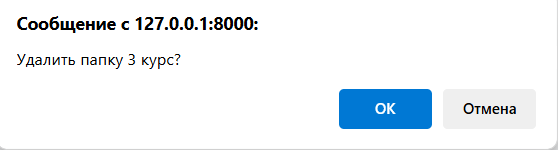


Рисунок 5.4 – Сообщение об удалении папки

5.4 Управление файлами

5.4.1 Загрузка

Файлы можно загрузить в хранилище, нажав на «Выбор файла». После выбора файла пользователь нажимает загрузить и файл отобразится в хранилище.

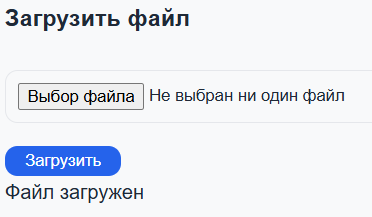


Рисунок 5.5 – Загрузка файла

5.4.2 Скачивание

Файлы можно скачать, нажав на ссылку «Скачать» рядом с названием файла. При нажатии на ссылку скачивается файл.

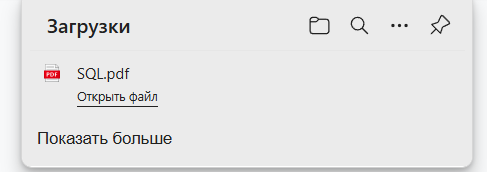


Рисунок 5.6 – Скачивание файла

5.4.2 Удаление

Удаление файлов осуществляется аналогично папкам — через HTML-форму с подтверждением.

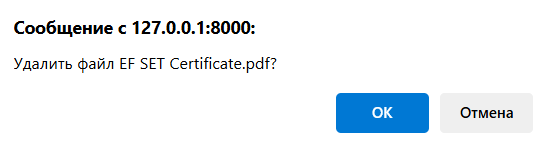


Рисунок 5.7 – Сообщение об удалении файла

5.4.3 Просмотр версий

Кнопка «Версии» открывает страницу с информацией о всех версиях выбранного файла. Отображается список с номерами версий и датами создания.

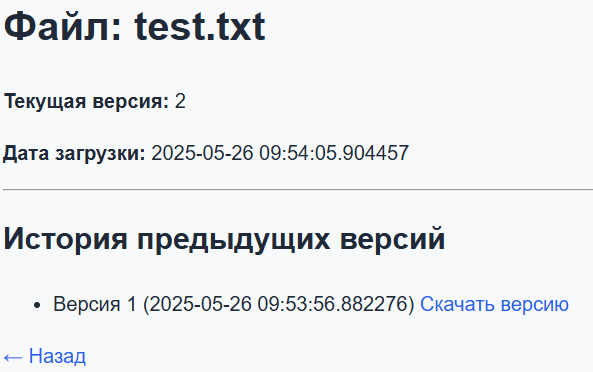


Рисунок 5.8 – Версии файла

5.5 Уведомление об истечении токена и выход из приложения

Для повышения безопасности и удобства реализована функция напоминания пользователю об окончании действия сессии. Примерно за 1 минуту до окончания действия access-токена появляется всплывающее окно с предложением обновить его.

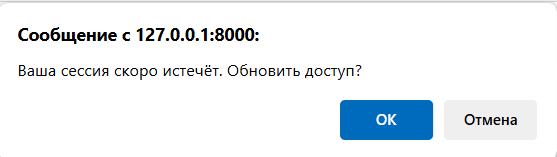


Рисунок 5.9 – Уведомление об истечении сессии

Если пользователь захочет завершить сессию, то он может нажать на кнопку «Выйти»



Рисунок 5.10 – Выход из аккаунта

# ЗАКЛЮЧЕНИЕ

В современном цифровом мире потребность в надёжном, безопасном и доступном способе хранения и управления файлами стремительно возрастает. Одним из эффективных решений данной задачи являются облачные технологии, позволяющие пользователям работать с данными из любой точки мира. В рамках курсового проекта было разработано распределённое облачное файловое хранилище, реализующее ключевые функции современного облачного сервиса.

В процессе разработки были успешно достигнуты все поставленные цели и выполнены основные задачи. Пользователю доступен удобный веб-интерфейс с возможностью регистрации и авторизации, навигации по структуре папок и файлов, а также загрузки, скачивания и удаления объектов. Особое внимание было уделено поддержке версионирования файлов, что позволяет хранить историю изменений и загружать предыдущие версии.

Хранилище реализует разделение доступа по пользователям, защиту сессий с помощью JWT-аутентификации, автоматическое уведомление о завершении действия токена, а также базовую защиту данных. В системе предусмотрена создание, удаление и переименование папок.

Кроме того, были реализованы дополнительные функции, повышающие удобство и надёжность: всплывающие окна подтверждения при удалении, формы для загрузки и создания объектов, система уведомлений и взаимодействие с сервером через REST API.

Разработанное хранилище обладает масштабируемостью и может быть использовано в качестве основы для построения более сложных систем хранения данных с расширенной функциональностью. В дальнейшем возможно добавление расширенного поиска, систем шифрования, репликации данных.

Таким образом, проект успешно решает задачу построения функционального и безопасного облачного хранилища, демонстрируя практическое применение современных веб-технологий и принципов распределённых систем.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Grinberg M. Flask Web Development: Developing Web Applications with Python. – 2nd ed. – O’Reilly Media, 2018. – 320 p.

[2] Sebastián Ramírez. FastAPI Documentation [Электронный ресурс]. – Режим доступа: https://fastapi.tiangolo.com/

[3] Vasiliev A.A. PostgreSQL для начинающих. – СПб.: БХВ-Петербург, 2019. – 352 с.

[4] Tanenbaum A.S., Van Steen M. Распределённые системы: принципы и парадигмы. – М.: Издательский дом "Вильямс", 2020. – 832 с.

[5] Beazley D. Python. Подробный справочник. – СПб.: Питер, 2021. – 880 с.

[6] Соловьев С.В. Проектирование REST API. Практика создания веб-сервисов. – М.: ДМК Пресс, 2021. – 256 с.

# ПРИЛОЖЕНИЕ А. Исходный код программы

main.py

**from fastapi import Depends, FastAPI, Request**

**from fastapi.responses import HTMLResponse, RedirectResponse**

**from fastapi.staticfiles import StaticFiles**

**from fastapi.templating import Jinja2Templates**

**from sqlalchemy.orm import Session**

**from app.config import settings**

**from app.database import get\_db**

**from app.models.folder import Folder**

**from app.routers import auth, folders, users, files, file\_version**

**import app.models**

**from app.services.auth\_service import get\_current\_user**

**from app.models.file import File**

**from starlette.middleware.sessions import SessionMiddleware**

**app = FastAPI()**

**app.add\_middleware(SessionMiddleware, secret\_key=settings.SECRET\_KEY)**

**# Подключение статики и шаблонов**

**app.mount("/static", StaticFiles(directory="design/static"), name="static")**

**templates = Jinja2Templates(directory="design/templates")**

**# Подключение маршрутов**

**app.include\_router(auth.router, prefix="/auth", tags=["Auth"])**

**app.include\_router(users.router, prefix="", tags=["Users"])**

**app.include\_router(files.router, prefix="/file", tags = ["Files"])**

**app.include\_router(folders.router)**

**app.include\_router(file\_version.router, prefix="/versions", tags=["File Versions"])**

**@app.get("/", response\_class=HTMLResponse)**

**def read\_root(**

**request: Request,**

**folder\_id: int = None,**

**db: Session = Depends(get\_db),**

**user=Depends(get\_current\_user)**

**):**

**current\_folder = None**

**subfolders = []**

**files = []**

**if not user:**

**return templates.TemplateResponse("home.html", {"request": request, "user": None})**

**if folder\_id:**

**current\_folder = db.query(Folder).filter(Folder.id == folder\_id, Folder.owner\_id == user.id).first()**

**if current\_folder:**

**subfolders = current\_folder.subfolders**

**files = current\_folder.files**

**else:**

**subfolders = db.query(Folder).filter(Folder.owner\_id == user.id, Folder.parent\_id == None).all()**

**files = db.query(File).filter(File.owner\_id == user.id, File.folder\_id == None).all()**

**return templates.TemplateResponse("home.html", {**

**"request": request,**

**"user": user,**

**"current\_folder": current\_folder,**

**"folders": subfolders,**

**"files": files**

**})**

Routers/auth.py

from fastapi import APIRouter, Depends, Request, Form, HTTPException

from fastapi.responses import HTMLResponse, JSONResponse, RedirectResponse

from fastapi.templating import Jinja2Templates

from sqlalchemy.orm import Session

from app.database import get\_db

from app.models.user import User

from app.services import auth\_service

from jose import JWTError, jwt

from app.config import settings

router = APIRouter()

templates = Jinja2Templates(directory="design/templates")

@router.get("/register", response\_class=HTMLResponse)

def register\_form(request: Request):

    return templates.TemplateResponse("register.html", {"request": request})

@router.get("/login", response\_class=HTMLResponse)

def login\_form(request: Request):

    return templates.TemplateResponse("login.html", {"request": request})

@router.post("/register", response\_class=HTMLResponse)

def register\_user(

    request: Request,

    email: str = Form(...),

    password: str = Form(...),

    db: Session = Depends(get\_db)

):

    existing\_user = db.query(User).filter(User.email == email).first()

    if existing\_user:

        return templates.TemplateResponse("register.html", {

            "request": request,

            "error": "Пользователь уже существует"

        })

    user = auth\_service.create\_user(db, email, password)

    access\_token = auth\_service.create\_access\_token({"sub": user.email})

    refresh\_token = auth\_service.create\_refresh\_token({"sub": user.email})

    response = RedirectResponse(url="/", status\_code=302)

    response.set\_cookie("access\_token", f"Bearer {access\_token}", httponly=True)

    response.set\_cookie("refresh\_token", f"Bearer {refresh\_token}", httponly=True, secure = True, samesite="Strict")

    return response

@router.post("/login", response\_class=HTMLResponse)

def login(

    request: Request,

    email: str = Form(...),

    password: str = Form(...),

    db: Session = Depends(get\_db)

):

    user = auth\_service.authenticate\_user(db, email, password)

    if not user:

        return templates.TemplateResponse("login.html", {

            "request": request,

            "error": "Неверный логин или пароль"

        })

    access\_token = auth\_service.create\_access\_token({"sub": user.email})

    refresh\_token = auth\_service.create\_refresh\_token({"sub": user.email})

    response = RedirectResponse("/", status\_code=302)

    response.set\_cookie("access\_token", f"Bearer {access\_token}", httponly=True)

    response.set\_cookie("refresh\_token", f"Bearer {refresh\_token}", httponly=True, secure = True, samesite="Strict" )

    return response

@router.post("/logout")

async def logout():

    response = RedirectResponse(url="/", status\_code=302)

    response.delete\_cookie("access\_token")

    response.delete\_cookie("refresh\_token")

    return response

@router.post("/refresh", response\_class=JSONResponse)

async def refresh\_token(request: Request):

    token = request.cookies.get("refresh\_token")

    if not token:

        raise HTTPException(status\_code=401, detail="Missing refresh token")

    scheme, \_, param = token.partition(" ")

    if scheme.lower() != "bearer":

        raise HTTPException(status\_code=401, detail="Invalid token scheme")

    try:

        payload = jwt.decode(param, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM])

        email = payload.get("sub")

        if not email:

            raise HTTPException(status\_code=401, detail="Invalid token payload")

    except JWTError:

        raise HTTPException(status\_code=401, detail="Invalid refresh token")

    new\_access\_token = auth\_service.create\_access\_token({"sub": email})

    response = JSONResponse({"message": "Token refreshed"})

    response.set\_cookie(

        "access\_token", f"Bearer {new\_access\_token}",

        httponly=True,

        secure=True,

        samesite="Strict"

    )

    return response

Routers/file\_version.py

# app/routers/file\_version\_router.py

import os

from fastapi import APIRouter, Depends, HTTPException, Request

from fastapi.responses import FileResponse, HTMLResponse

from fastapi.templating import Jinja2Templates

from sqlalchemy.orm import Session

from typing import List

from app.database import get\_db

from app.models.file import File

from app.services.auth\_service import get\_current\_user

from app.services.file\_service import get\_file\_by\_id

from app.services.file\_version\_service import (

    create\_file\_version, get\_versions\_by\_file\_id, get\_file\_version

)

from app.schemas.file\_version import FileVersionCreate, FileVersionOut

router = APIRouter()

@router.post("/{file\_id}", response\_model=FileVersionOut)

def add\_version(

    file\_id: int,

    version\_data: FileVersionCreate,

    db: Session = Depends(get\_db),

    user=Depends(get\_current\_user)

):

    return create\_file\_version(file\_id, version\_data, db)

@router.get("/{file\_id}", response\_model=List[FileVersionOut])

def list\_versions(

    file\_id: int,

    db: Session = Depends(get\_db),

    user=Depends(get\_current\_user)

):

    return get\_versions\_by\_file\_id(file\_id, db)

@router.get("/single/{version\_id}", response\_model=FileVersionOut)

def get\_version(

    version\_id: int,

    db: Session = Depends(get\_db),

    user=Depends(get\_current\_user)

):

    return get\_file\_version(version\_id, db)

templates = Jinja2Templates(directory="design/templates")

@router.get("/file/download\_version/{version\_id}")

def download\_version(version\_id: int, db: Session = Depends(get\_db), user=Depends(get\_current\_user)):

    version = get\_file\_version(version\_id, db)

    file = db.query(File).filter(File.id == version.file\_id).first()

    if not file:

        raise HTTPException(status\_code=404, detail="Оригинальный файл не найден")

    filename, ext = os.path.splitext(file.name)

    versioned\_filename = f"{filename}\_v{version.version\_number}{ext}"

    return FileResponse(path=version.path, filename=versioned\_filename)

@router.get("/file/{file\_id}", response\_class=HTMLResponse)

def file\_detail(file\_id: int, request: Request, db: Session = Depends(get\_db), user=Depends(get\_current\_user)):

    file = get\_file\_by\_id(file\_id, db)

    versions = get\_versions\_by\_file\_id(file\_id, db)

    current\_version = versions[0] if versions else None

    return templates.TemplateResponse("file.html", {

        "request": request,

        "file": {

            "id": file.id,

            "name": file.name,

            "current\_version": current\_version,

            "versions": versions

        }

    })

Routers/files.py

from typing import Optional

from fastapi import APIRouter, Depends, UploadFile, File, HTTPException, Request, Form

from fastapi.responses import FileResponse, RedirectResponse

from sqlalchemy.orm import Session

from app.database import get\_db

from app.services.auth\_service import get\_current\_user

from app.services.file\_service import (

    delete\_file\_by\_id,

    get\_file\_by\_id,

    save\_file

)

router = APIRouter()

@router.post("/upload")

async def upload\_file(

    request: Request,

    file: UploadFile = File(...),

    folder\_id: Optional[str] = Form(default=None),

    db: Session = Depends(get\_db),

    user = Depends(get\_current\_user)

):

    if folder\_id in (None, "", "null", "None"):

        folder\_id\_int = None

    else:

        try:

            folder\_id\_int = int(folder\_id)

        except ValueError:

            raise HTTPException(status\_code=400, detail="folder\_id must be an integer")

    file\_path = save\_file(file, file.filename, user.id, db, folder\_id\_int)

    if not file\_path:

        raise HTTPException(status\_code=500, detail="Ошибка при сохранении файла")

    if folder\_id\_int:

        return RedirectResponse(url=f"/?folder\_id={folder\_id\_int}&msg=Файл загружен", status\_code=303)

    return RedirectResponse(url="/?msg=Файл загружен", status\_code=303)

@router.get("/download/{file\_id}")

def download\_file(

    file\_id: int,

    db: Session = Depends(get\_db),

    user = Depends(get\_current\_user)

):

    file = get\_file\_by\_id(file\_id, db)

    if not file:

        raise HTTPException(status\_code=404, detail="Файл не найден")

    return FileResponse(path=file.path, filename=file.name, media\_type="application/octet-stream")

@router.post("/delete/{file\_id}")

def delete\_file(

    file\_id: int,

    db: Session = Depends(get\_db),

    user = Depends(get\_current\_user)

):

    delete\_file\_by\_id(file\_id, db)

    return RedirectResponse(url="/?msg=Файл удален", status\_code=303)

Routers/folders.py

from fastapi import APIRouter, Depends, Form, HTTPException, status

from fastapi.responses import RedirectResponse

from sqlalchemy.orm import Session

from typing import List, Optional

from app.database import get\_db

from app.schemas.folder import FolderCreate, FolderRead, FolderUpdate

from app.services.auth\_service import get\_current\_user

from app.services.folder\_service import (

    get\_folder,

    get\_folders,

    create\_folder,

    update\_folder,

    delete\_folder,

)

router = APIRouter(

    prefix="/folders",

    tags=["Folders"],

)

@router.get("/", response\_model=List[FolderRead])

def read\_folders(db: Session = Depends(get\_db), current\_user=Depends(get\_current\_user)):

    folders = get\_folders(db, owner\_id=current\_user.id)

    return folders

@router.get("/{folder\_id}", response\_model=FolderRead)

def read\_folder(folder\_id: int, db: Session = Depends(get\_db), current\_user=Depends(get\_current\_user)):

    folder = get\_folder(db, folder\_id)

    if not folder or folder.owner\_id != current\_user.id:

        raise HTTPException(status\_code=404, detail="Folder not found")

    return folder

@router.post("/", response\_model=FolderRead, status\_code=status.HTTP\_201\_CREATED)

def create\_new\_folder(

    name: str = Form(...),

    parent\_id: Optional[str] = Form(None),

    db: Session = Depends(get\_db),

    current\_user=Depends(get\_current\_user)

):

    if parent\_id == "" or parent\_id is None:

        parent\_id\_int = None

    else:

        try:

            parent\_id\_int = int(parent\_id)

        except ValueError:

            raise HTTPException(status\_code=400, detail="parent\_id must be an integer")

    folder\_data = FolderCreate(name=name, parent\_id=parent\_id\_int)

    create\_folder(db, folder\_data, owner\_id=current\_user.id)

    if parent\_id\_int in (None, 0):

        redirect\_url = "/"

    else:

        redirect\_url = f"/?folder\_id={parent\_id\_int}"

    return RedirectResponse(url=redirect\_url, status\_code=status.HTTP\_303\_SEE\_OTHER)

@router.put("/{folder\_id}", response\_model=FolderRead)

def update\_existing\_folder(folder\_id: int, folder\_update: FolderUpdate, db: Session = Depends(get\_db), current\_user=Depends(get\_current\_user)):

    folder = get\_folder(db, folder\_id)

    if not folder or folder.owner\_id != current\_user.id:

        raise HTTPException(status\_code=404, detail="Folder not found")

    updated\_folder = update\_folder(db, folder\_id, folder\_update)

    return updated\_folder

@router.post("/delete/{folder\_id}", response\_class=RedirectResponse)

def delete\_folder\_and\_redirect(

    folder\_id: int,

    db: Session = Depends(get\_db),

    current\_user=Depends(get\_current\_user)

):

    folder = get\_folder(db, folder\_id)

    if not folder or folder.owner\_id != current\_user.id:

        raise HTTPException(status\_code=404, detail="Folder not found")

    parent\_id = folder.parent\_id

    delete\_folder(db, folder\_id, current\_user.id)

    if parent\_id:

        redirect\_url = f"/?folders/{parent\_id}"

    else:

        redirect\_url = "/"

    return RedirectResponse(url=redirect\_url, status\_code=303)

services/auth\_service.py

import os

from fastapi import Depends, Request

from passlib.context import CryptContext

from jose import JWTError, jwt

from datetime import datetime, timedelta

from app.database import get\_db

from app.models.folder import Folder

from app.models.user import User

from sqlalchemy.orm import Session

from app.config import STORAGE\_NODES, UPLOAD\_DIR, settings

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash\_password(password: str) -> str:

    return pwd\_context.hash(password)

def create\_token(data: dict, expires\_delta: timedelta = timedelta(hours=1)):

    to\_encode = data.copy()

    to\_encode["exp"] = datetime.utcnow() + expires\_delta

    return jwt.encode(to\_encode, settings.SECRET\_KEY, algorithm=settings.ALGORITHM)

def create\_access\_token(data:dict):

    return create\_token(data, timedelta(minutes=15))

def create\_refresh\_token(data:dict):

    return create\_token(data, timedelta(days=7))

def create\_user(db: Session, email: str, password: str):

    hashed\_pw = hash\_password(password)

    user = User(email=email, password\_hash=hashed\_pw)

    db.add(user)

    db.commit()

    db.refresh(user)

    # Создаём физические директории для пользователя на диске в каждом узле

    user\_folder\_name = f"user\_{user.id}"

    for node in STORAGE\_NODES:

        node\_user\_path = os.path.join(UPLOAD\_DIR, node, user\_folder\_name)

        os.makedirs(node\_user\_path, exist\_ok=True)

    return user

def verify\_password(plain\_password: str, hashed\_password: str) -> bool:

    return pwd\_context.verify(plain\_password, hashed\_password)

def authenticate\_user(db: Session, email: str, password: str):

    user = db.query(User).filter(User.email == email).first()

    if not user:

        return None

    if not verify\_password(password, user.password\_hash):

        return None

    return user

async def get\_current\_user(request: Request, db: Session = Depends(get\_db)):

    token = request.cookies.get("access\_token")

    if not token:

        return None

    try:

        scheme, \_, param = token.partition(" ")

        if scheme.lower() != "bearer":

            return None

        payload = jwt.decode(param, settings.SECRET\_KEY, algorithms=[settings.ALGORITHM])

        email: str = payload.get("sub")

        if email is None:

            return None

    except JWTError:

        return None

    user = db.query(User).filter(User.email == email).first()

    return user

services/file\_service.py

import os

from typing import List, Optional

import uuid

from fastapi import HTTPException, UploadFile

from app.schemas.file\_version import FileVersionCreate

from app.services.file\_version\_service import create\_file\_version, get\_versions\_by\_file\_id

from app.services.folder\_service import get\_folder\_disk\_path

from app.services.storage\_node import choose\_storage\_node

from app.config import UPLOAD\_DIR

from sqlalchemy.orm import Session

from app.models import File

def save\_file(

    file: UploadFile,

    filename: str,

    user\_id: int,

    db: Session,

    folder\_id: Optional[int] = None

) -> str:

    node = choose\_storage\_node()

    if folder\_id:

        user\_dir = get\_folder\_disk\_path(db, user\_id, folder\_id, node)

    else:

        user\_dir = os.path.join(UPLOAD\_DIR, node, f"user\_{user\_id}")

    os.makedirs(user\_dir, exist\_ok=True)

    existing\_file = db.query(File).filter\_by(

        owner\_id=user\_id,

        folder\_id=folder\_id,

        name=filename

    ).first()

    unique\_filename = f"{uuid.uuid4()}\_{filename}"

    new\_file\_path = os.path.join(user\_dir, unique\_filename)

    with open(new\_file\_path, "wb") as buffer:

        content = file.file.read()

        buffer.write(content)

    if existing\_file:

        if os.path.exists(existing\_file.path):

            existing\_versions = get\_versions\_by\_file\_id(existing\_file.id, db)

            new\_version\_number = (existing\_versions[0].version\_number + 1) if existing\_versions else 1

            create\_file\_version(

                existing\_file.id,

                FileVersionCreate(

                    version\_number=new\_version\_number,

                    path=existing\_file.path

                ),

                db

            )

        existing\_file.path = new\_file\_path

        db.commit()

        db.refresh(existing\_file)

        return new\_file\_path

    db\_file = File(

        name=filename,

        path=new\_file\_path,

        owner\_id=user\_id,

        folder\_id=folder\_id

    )

    db.add(db\_file)

    db.commit()

    db.refresh(db\_file)

    create\_file\_version(

        db\_file.id,

        FileVersionCreate(version\_number=1, path=new\_file\_path),

        db

    )

    return new\_file\_path

def get\_file\_by\_id(file\_id: int, db: Session) -> File:

    file = db.query(File).filter(File.id == file\_id).first()

    if not file:

        raise HTTPException(status\_code=404, detail="Файл не найден в базе данных")

    if not os.path.exists(file.path):

        raise HTTPException(status\_code=410, detail="Файл отсутствует на сервере (возможно удалён)")

    return file

def delete\_file\_by\_id(file\_id: int, db: Session) -> None:

    file = db.query(File).filter(File.id == file\_id).first()

    if not file:

        raise HTTPException(status\_code=404, detail="Файл не найден")

    versions = get\_versions\_by\_file\_id(file\_id, db)

    for version in versions:

        if os.path.exists(version.path):

            try:

                os.remove(version.path)

            except Exception as e:

                print(f"Ошибка при удалении версии файла {version.path}: {e}")

        else:

            print(f"Версия файла {version.path} не найдена на диске")

    if os.path.exists(file.path):

        try:

            os.remove(file.path)

        except Exception as e:

            raise HTTPException(status\_code=500, detail=f"Ошибка при удалении файла с диска: {e}")

    else:

        print(f"Файл {file.path} не найден на диске, но запись удаляется из БД")

    # Удаляем запись из БД

    db.delete(file)

    db.commit()

def get\_files\_by\_user(user\_id: int, db: Session) -> List[File]:

    all\_files = db.query(File).filter(File.owner\_id == user\_id).all()

    existing\_files = [f for f in all\_files if os.path.exists(f.path)]

    return existing\_files

def get\_file\_by\_id(file\_id: int, db: Session) -> File:

    file = db.query(File).filter(File.id == file\_id).first()

    if not file:

        raise HTTPException(status\_code=404, detail="Файл не найден")

    return file

services/file\_version\_service.py

from sqlalchemy.orm import Session

from app.models.file\_version import FileVersion

from app.models.file import File

from app.schemas.file\_version import FileVersionCreate

from fastapi import HTTPException

def create\_file\_version(file\_id: int, version\_data: FileVersionCreate, db: Session) -> FileVersion:

    file = db.query(File).filter(File.id == file\_id).first()

    if not file:

        raise HTTPException(status\_code=404, detail="Такого файла не существует")

    new\_version = FileVersion(

        file\_id=file\_id,

        version\_number=version\_data.version\_number,

        path=version\_data.path

    )

    db.add(new\_version)

    db.commit()

    db.refresh(new\_version)

    return new\_version

def get\_versions\_by\_file\_id(file\_id: int, db: Session):

    return db.query(FileVersion).filter(FileVersion.file\_id == file\_id).order\_by(FileVersion.version\_number.desc()).all()

def get\_file\_version(version\_id: int, db: Session) -> FileVersion:

    version = db.query(FileVersion).filter(FileVersion.id == version\_id).first()

    if not version:

        raise HTTPException(status\_code=404, detail="Версия не найдена")

    return version

services/folder\_service.py

import os

import shutil

from typing import List, Optional

from fastapi import HTTPException

from sqlalchemy.orm import Session

from app.config import STORAGE\_NODES, UPLOAD\_DIR

from app.models import Folder

from app.models.file import File

from app.schemas.folder import FolderCreate, FolderUpdate

def get\_folder(db: Session, folder\_id: int) -> Optional[Folder]:

    return db.query(Folder).filter(Folder.id == folder\_id).first()

def get\_folders(db: Session, owner\_id: int) -> List[Folder]:

    return db.query(Folder).filter(Folder.owner\_id == owner\_id).all()

def get\_folder\_disk\_path(db: Session, owner\_id: int, folder\_id: int, node: str) -> str:

    path\_parts = []

    current\_folder\_id = folder\_id

    while current\_folder\_id:

        folder = db.query(Folder).filter(Folder.id == current\_folder\_id).first()

        if not folder:

            break

        path\_parts.append(f"folder\_{folder.id}")

        current\_folder\_id = folder.parent\_id

    path\_parts.reverse()

    base\_path = os.path.join(UPLOAD\_DIR, node, f"user\_{owner\_id}")

    full\_path = os.path.join(base\_path, \*path\_parts)

    return full\_path

def create\_folder(db: Session, folder: FolderCreate, owner\_id: int) -> Folder:

    parent\_id = folder.parent\_id if folder.parent\_id and folder.parent\_id != 0 else None

    # Проверка: нет ли уже папки с таким именем в том же родителе у того же пользователя

    existing\_folder = db.query(Folder).filter\_by(

        name=folder.name,

        parent\_id=parent\_id,

        owner\_id=owner\_id

    ).first()

    if existing\_folder:

        raise HTTPException(status\_code=400, detail="Папка с таким именем уже существует в этом каталоге.")

    db\_folder = Folder(

        name=folder.name,

        parent\_id=parent\_id,

        owner\_id=owner\_id

    )

    db.add(db\_folder)

    db.commit()

    db.refresh(db\_folder)

    for node in STORAGE\_NODES:

        folder\_path = get\_folder\_disk\_path(db, owner\_id, db\_folder.id, node)

        os.makedirs(folder\_path, exist\_ok=True)

    return db\_folder

def update\_folder(db: Session, folder\_id: int, folder\_update: FolderUpdate) -> Optional[Folder]:

    db\_folder = get\_folder(db, folder\_id)

    if not db\_folder:

        return None

    if folder\_update.name is not None:

        db\_folder.name = folder\_update.name

    if folder\_update.parent\_id is not None:

        db\_folder.parent\_id = folder\_update.parent\_id

    db.commit()

    db.refresh(db\_folder)

    return db\_folder

def delete\_folder(db: Session, folder\_id: int, owner\_id: int) -> bool:

    folder = db.query(Folder).filter(Folder.id == folder\_id, Folder.owner\_id == owner\_id).first()

    if not folder:

        return False

    def delete\_folder\_recursive(fold: Folder):

        files = db.query(File).filter(File.folder\_id == fold.id).all()

        for file in files:

            if os.path.exists(file.path):

                os.remove(file.path)

            db.delete(file)

        subfolders = db.query(Folder).filter(Folder.parent\_id == fold.id).all()

        for subf in subfolders:

            delete\_folder\_recursive(subf)

        for node in STORAGE\_NODES:

            try:

                full\_path = get\_folder\_disk\_path(db, owner\_id, fold.id, node)

                if os.path.exists(full\_path):

                    shutil.rmtree(full\_path)

            except Exception as e:

                print(f"Ошибка при удалении папки на узле '{node}': {e}")

        db.delete(fold)

    delete\_folder\_recursive(folder)

    db.commit()

    return True

services/storage\_node \_service.py

import os

from app.config import UPLOAD\_DIR, STORAGE\_NODES

def get\_node\_load(node\_name: str) -> int:

    node\_path = os.path.join(UPLOAD\_DIR, node\_name)

    total\_size = 0

    for root, dirs, files in os.walk(node\_path):

        for f in files:

            try:

                fp = os.path.join(root, f)

                total\_size += os.path.getsize(fp)

            except OSError:

                pass

    return total\_size

def choose\_storage\_node() -> str:

    loads = {node: get\_node\_load(node) for node in STORAGE\_NODES}

    chosen\_node = min(loads, key=loads.get)

    return chosen\_node

templates/home.html

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8" />

    <title>Файловое хранилище</title>

    <link rel="stylesheet" href="/static/css/style.css" />

</head>

<body>

    <h1>Добро пожаловать в Распределенное облачное хранилище</h1>

    {% if user %}

        <p>Вы вошли как: {{ user.email }}</p>

        <form method="post" action="/auth/logout">

            <button type="submit">Выйти</button>

        </form>

        <h2>Папка: {{ current\_folder.name if current\_folder else "Корневая" }}</h2>

        <!-- Навигация вверх -->

        {% if current\_folder and current\_folder.parent %}

            <p><a href="/?folder\_id={{ current\_folder.parent.id }}">Вверх: {{ current\_folder.parent.name }}</a></p>

        {% elif current\_folder %}

            <p><a href="/">Вверх: Корневая папка</a></p>

        {% endif %}

        <!-- Создать новую папку -->

        <h3>Создать папку</h3>

        <form method="post" action="/folders">

            <input type="hidden" name="parent\_id" value="{{ current\_folder.id if current\_folder else '' }}" />

            <input type="text" name="name" placeholder="Имя папки" required />

            <button type="submit">Создать</button>

        </form>

        <!-- Загрузка файла -->

        <h3>Загрузить файл</h3>

        <form action="/file/upload" method="post" enctype="multipart/form-data">

            <input type="hidden" name="folder\_id" value="{{ current\_folder.id if current\_folder else '' }}" />

            <input type="file" name="file" required />

            <button type="submit">Загрузить</button>

        </form>

        {% if request.query\_params.msg %}

            <div class="flash-message">{{ request.query\_params.msg }}</div>

        {% endif %}

        <!-- Список папок -->

        <h3>Папки</h3>

        {% if folders %}

            <ul>

                {% for folder in folders %}

                    <li>

                        📁 <a href="/?folder\_id={{ folder.id }}">{{ folder.name }}</a>

                        <button onclick="showRenameForm('{{ folder.id }}', '{{ folder.name }}')" class="rename-btn">Переименовать</button>

                        <form method="post" action="/folders/delete/{{ folder.id }}" style="display:inline;" onsubmit="return confirm('Удалить папку {{ folder.name }}?');">

                            <button type="submit" class="delete-btn">Удалить</button>

                        </form>

                        <!-- Форма для переименования (изначально скрыта) -->

                        <div id="rename-form-{{ folder.id }}" style="display: none;">

                            <form onsubmit="renameFolder(event, '{{ folder.id }}')">

                                <input type="text" id="new-name-{{ folder.id }}" value="{{ folder.name }}" required />

                                <button type="submit">Сохранить</button>

                                <button type="button" onclick="hideRenameForm('{{ folder.id }}')">Отмена</button>

                            </form>

                        </div>

                    </li>

                {% endfor %}

            </ul>

        {% else %}

            <p>Папок нет</p>

        {% endif %}

        <script>

            // Показываем форму переименования

            function showRenameForm(folderId, currentName) {

                // Скрываем все другие формы переименования

                document.querySelectorAll('[id^="rename-form-"]').forEach(form => {

                    form.style.display = 'none';

                });

                const form = document.getElementById(`rename-form-${folderId}`);

                form.style.display = 'block';

                document.getElementById(`new-name-${folderId}`).value = currentName;

            }

            // Скрываем форму переименования

            function hideRenameForm(folderId) {

                document.getElementById(`rename-form-${folderId}`).style.display = 'none';

            }

            // Отправляем запрос на переименование

            async function renameFolder(event, folderId) {

                event.preventDefault();

                const newName = document.getElementById(`new-name-${folderId}`).value.trim();

                if (!newName) {

                    alert('Введите новое имя папки');

                    return;

                }

                try {

                    const response = await fetch(`/folders/${folderId}`, {

                        method: 'PUT',

                        headers: {

                            'Content-Type': 'application/json',

                        },

                        body: JSON.stringify({ name: newName })

                    });

                    if (response.ok) {

                        // Обновляем страницу, чтобы показать новое имя

                        window.location.reload();

                    } else {

                        const error = await response.json();

                        alert(error.detail || 'Ошибка при переименовании папки');

                    }

                } catch (err) {

                    console.error('Ошибка:', err);

                    alert('Ошибка соединения с сервером');

                }

            }

        </script>

        <!-- Список файлов -->

        <h3>Файлы</h3>

        {% if files %}

            <ul>

                {% for file in files %}

                    <li>

                            {{ file.name }}

                            <a href="/file/download/{{ file.id }}">Скачать</a>

                            <a href="/versions/file/{{ file.id }}">Версии</a>

                            <form method="post" action="/file/delete/{{ file.id }}" style="display:inline;" onsubmit="return confirm('Удалить файл {{ file.name }}?');">

                            <button type="submit" class="delete-btn">Удалить</button>

                        </form>

                    </li>

                {% endfor %}

            </ul>

        {% else %}

            <p>Файлов нет</p>

        {% endif %}

    {% else %}

        <a href="/auth/login"><button>Войти</button></a>

        <a href="/auth/register"><button>Зарегистрироваться</button></a>

    {% endif %}

    <script>

        function askToRefreshToken() {

            const shouldRefresh = confirm("Ваша сессия скоро истечёт. Обновить доступ?");

            if (shouldRefresh) {

                fetch("/auth/refresh", {

                    method: "POST",

                    credentials: "include"

                }).then(response => {

                    if (response.ok) {

                        console.log("Access токен обновлён");

                    } else {

                        alert("Не удалось обновить токен. Пожалуйста, войдите заново.");

                        window.location.href = "/auth/login";

                    }

                }).catch(error => {

                    console.error("Ошибка при обновлении токена:", error);

                    alert("Ошибка соединения. Пожалуйста, войдите заново.");

                    window.location.href = "/auth/login";

                });

            }

            else {

                console.log("Пользователь отказался от обновления токена.");

            }

        }

        setTimeout(askToRefreshToken, 14 \* 60 \* 1000);

    </script>

</body>

</html>