Государственное бюджетное образовательное учреждение города

Москвы

Школа №1502 «Энергия»

**Частное облачное хранилище**

Назолин Иван Андреевич

10Л класс

Руководитель:

Сесин Анатолий Андреевич,

учитель информатики

Москва 2025

Оглавление

[ВВЕДЕНИЕ 3](#_Toc184674634)

[1. Теоретическая часть. 4](#_Toc184674635)

[1.1 Основы разработки веб-приложений с использованием фреймворков на Python. 4](#_Toc184674636)

[1.2 Причина выбора Flask для реализации проекта. 4](#_Toc184674637)

[1.3 Работа с файлами и вложенными папками. 5](#_Toc184674638)

[1.4 Работа с архивами и потоковой передачей данных. 6](#_Toc184674639)

[1.5 Хеширование данных 7](#_Toc184674640)

[1.6 Система аутентификации и управления сессиями в Flask. 10](#_Toc184674641)

[1.7 Виды протоколов передачи данных http и https. 10](#_Toc184674642)

[1.8 Потоковая передача данных. 11](#_Toc184674643)

[1.9 Архитектура сетей и IPv4. 12](#_Toc184674644)

[1.10 Использование сервисов Apache, ngrok, localtunel. 14](#_Toc184674645)

[1.11 Работа зашифрованных туннелей. 15](#_Toc184674646)

# ВВЕДЕНИЕ

В современном мире облачные хранилища стали неотъемлемой частью нашей жизни. В данной работе рассмотрено создание облачного хранилища с объединёнными сервером и местом происхождения файлов. Сервисы облачного хранилища включает в себя множество разных задач, например, сжатие, замена, шифрование, передача данных. В современном мире, где цифровизация проникает во все сферы жизни, облачные технологии становятся всё более востребованными. Они предоставляют пользователям гибкие и удобные решения для хранения, обработки и обмена данными. Создание частного облачного хранилища является актуальным и перспективным направлением, которое может предложить ряд преимуществ как для компаний, так и для частных лиц. Цель работы: создание облачного хранилища и получение к нему доступа из любого места. Данный проект написан с использованием языка программирование Python и фреймворка Flask.

Для разметки и стилизации сайта были использованы HTML и CSS.

Задачи проекта:

1. Создание серверной части для сайта на Flask.
2. Реализация системы аутентификации.

2.1 Использование хеширования паролей с помощью hashlib и hmac.

2.2 Создание формы аутентификации

2.3 Использования механизма временных сессий.

3. Реализация функций работы с файлами.

3.1 Поиск файлов и папок в выбранной директории.

3.2 Сжатие директорий в архив в случае их пересылки.

# Теоретическая часть.

## 1.1 Основы разработки веб-приложений с использованием фреймворков на Python.

Для разработки веб-приложений на Python существует множество Фреймворков таких как:

1. Django большой фреймворк для создания веб сайтов и приложений имеющий большое количество встроенных функций таких как готовые админ панели и ORM системы.
2. Flask легкий фреймворк, предоставляющий все базовые инструменты и функции. Из плюсов можно отметить высокую производительность и возможность использования сторонних библиотек.
3. FastAPI новый фреймворк нацеленный на создание REST API приложений на Python.
4. Tornado фреймворк с асинхронной обработкой запросов, созданный для приложений требующих большое количество соединений в реальном времени.
5. CherryPy фреймворк, позволяющий создавать HTTP сервера с поддержкой сессий и многозадачности.

## 1.2 Причина выбора Flask для реализации проекта.

Преимуществом Flask относительно других являются несколько пунктов.

1.Легкость в освоении, он не имеет жесткие стандарты и предоставляет большую свободу в создании проекта.

2. Поддержка потоковой передачи данных. Это очень важно для данного проекта так как в нем будет передаваться большие данные и потоковая передача данных ускорит этот процесс.

3. Большое количество пользователей и исчерпывающая документация.

Многие из этих пунктов встречаются и в других библиотеках и фреймворках, но Flask был создан для небольших веб-приложений и сайтов и он легок в изучении.

## 1.3 Работа с файлами и вложенными папками.

Так как проект связан передачей фалов требуется выгружать мена фалов и папок на сайт. Здесь существует два пути:

1. Получить пути ко всем элементам в папке включая все вложенные элементы.
2. Выгружать имена файлов и папок только по требованию, то есть когда пользователь нажимает на какую-то папку только тогда определяется какие файлы и папки лежат в ней.

В обоих случаях временная сложность будет O(n), но в случае если множество элементов находятся во вложенных папках то в первом случае изначальное время будет очень велико, а во втором случае когда выгружается только по требованию, то не все пути будут выгружены так как какие-то будут не задействованы.

Верхняя Папка/

├── Папка1/

│ ├── Файл1.txt

│ └── Файл2.txt

└── Папка2/

├── Файл1.txt

└── Файл2.txt

При использовании первого метода временная сложность будет O(6), а при втором методе если пользователь выберет Папка2/ то пути к файлам в Папка1 не будут погружаться и временная сложность будет O(4).

Еже нужно учитывать временные файлы, их создаю программы для временного хранения данных. В операционной системы они не видны пользователю, но при получении данных через модуль Os в Python они будут видны.

Верхняя Папка/

├── Файл1.txt

├── ~Файл1.txt

├── .Файл1.txt

└── #Файл1.txt

~Файл1.txt и #Файл1.txt будут примером того как выглядят временных файлы. В операционной системе Windows они начинаются с “~”, а в UNIX подобных системах с “#” или “.”

## 1.4 Работа с архивами и потоковой передачей данных.

Так как в проекте предусмотрена функция передачи папки. Для того, чтобы передать целую папку нужно ее представить в виде архива.

Данные которые хорошо сжимаются:

1. Текстовые данные.
2. Файлы с большим количеством повторяющихся паттернов.
3. Некоторые виды изображений такие как PNG или GIF.
4. Звуковые файлы.

Данные которые плохо сжимаются:

1. Данные которые уже сжаты это могут быть видео в формате MP4 фото в формате JPEG архивы ZIP и RAR
2. Шумные изображения и видео.

Сжатие данных есть двух видов с потерями и без потерь. В проекте используется только сжатие без потерь. Для экономии времени на передачу информации для директорий имеющих вес более 500 Мб сжатие не производится, они просто упаковываются в архив без сжатия. Для сравнения сжатия с потерями и без потерь приведена таблица 1.

|  |  |  |
| --- | --- | --- |
| **Критерий** | Сжатие без потерь | Сжатие с потерями |
| Цель | Сохранить все данные без изменений | Уменьшить размер за счет потери данных |
| Применение | Текстовые файлы, код, архивы | Изображения, аудио, видео |
| Пример алгоритмов | Huffman, LZ77, ZIP, GZIP | |  | | --- | |  |  |  | | --- | | JPEG, MP3, H.264 | |
| Качество данных | Без изменений, полное восстановление | Потери в данных, качество может ухудшиться |
| Размер после сжатия | Обычно не так сильно сжато как с потерями | Сжатие в 10-100 раз эффективнее |
| Время восстановления | Полностью восстанавливается за время декодирования | Восстановление невозможно, качество может быть потеряно |

## 1.5 Хеширование данных

Хеширование — это процесс шифрования данных путем преобразования входных данных (строки, файла или пароля) в уникальный выходной результат фиксированной длины, который называется хешем. Существует несколько алгоритмов хеширования такие, как SHA-256, MD5, SHA-1. Для примера возьмем строку “Hello, world” c использование SHA-256 эта строка будет выглядеть:

“ a591a6d40bf420404a011733cfb7b190d62c65bf0bcda3be27c542b8d0c3e107”

У хеширования есть 3 основных принципа:

1. Принцип однонаправленности. (Это означает что на основе хеша нельзя узнать изначальный пароль)
2. Преобразование данных в фиксированную длину. (Несмотря на то какого размера был изначальный объект, конечный размер будет всегда одинаковым. При использовании функции SHA-256 результат будет всегда 256 бит или 32 байта)
3. Уникальность. (Хеш-функция должна минимизировать вероятность совпадения различных данных. 256-битный хеш имеет степени значений.)

Алгоритм хеширования это процесс применения алгебраических и алгебрологических функций, который преобразует данные произвольного размера в строку определённой длины. Алгоритм использует логические операции такие как XOR и AND, побитовые сдвиги, чтобы создать почти уникальный ключ.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | Размер хеша | Безопасность | Скорость | Применение | Известные уязвимости |
| MD5 | 128 бит | Слабая | Быстрый | Проверка целостности, цифровые подписи (устарел) | Уязвимость к коллизиям |
| SHA-1 | 160 бит | Умеренная | Средний | Цифровые подписи, SSL/TLS (устарел) | Уязвимость к коллизиям |
| SHA-256 | 256 бит | Высокая | Средний | Блокчейн, криптография, SSL/TLS | Нет известных уязвимостей |
| SHA-3 | 224-512 бит | Высокая | Медленный | Криптография, цифровые подписи | Нет известных уязвимостей |
| BLAKE2 | 256/512 бит | Очень высокая | Очень быстрый | Хеширование паролей, проверки целостности | Нет известных уязвимостей |
| RIPEMD-160 | 160 бит | Умеренная | Средний | Проверка целостности, криптография | Нет известных уязвимостей |

Таблица сравнения алгоритмов хеширования:

## 1.6 Система аутентификации и управления сессиями в Flask.

Сессии во Flask нужны чтобы хранить данные о взаимодействии пользователя с веб приложением. Веб- приложения основаны на принципе “stateless” , то есть без сохранения состояния каждый следующий запрос не связан с предыдущим. С помощью сессий в проекте хранится аутентифицирован ли пользователь. Когда пользователь переходит на страницу аутентификации (/login), приложение отображает форму, где пользователь может ввести имя пользователя и пароль. Когда пользователь отправляет форму с данными (метод POST) отправит данные на сервер, и система проверяет введенные данные, сравнивает имя пользователя с сохраненным значением, а пароль — с хэшированным значением в конфигурации. Если введенные данные верны, система сохраняет имя пользователя в объекте session, что сигнализирует о том, что пользователь успешно вошел в систему. После этого его перенаправляют на главную страницу или в раздел, к которому требуется доступ.

## 1.7 Виды протоколов передачи данных http и https.

В мире есть два основных протокола для передачи данных это http и https.

1.HTTP это протокол передачи гипертекста, который используется для передачи дынных от клиента к серверу. Он позволяет передавать веб страницы. У этого протокола есть несколько основных статусных кодов таких как 200- успешно 404 не найдено и так дале. Минусами этого протокола являются отсутствие шифрования и данные могут быть перехвачены третьими лицами.

2. HTTPS это защищенная версия протокола HTTP, где данные передаются через зашифрованное ssl соединение. Из-за этого через этот протокол можно передавать такие данные как пароли, данные банковских карт и так далее. Для шифрования сервер предоставляет свой ssl сертификат. Это гарантирует, что данные не будут расшифрованы если их перехватят.

Сравнение протоколов:

|  |  |  |
| --- | --- | --- |
| Характеристика | HTTP | HTTPS |
| Безопасность | Не шифрует данные. Уязвим для перехвата и изменения. | Шифрует данные с использованием SSL/TLS, что защищает от перехвата и изменения. |
| Протокол | Передача данных без шифрования. | Передача данных с шифрованием через SSL/TLS. |
| Порт | 80 | 443 |
| Скорость | Быстрее из-за отсутствия шифрования. | Может быть немного медленнее из-за процесса шифрования. |
| Использование | Используется для большинства публичных веб-сайтов без чувствительной информации. | Используется для сайтов, где требуется защита данных, таких как онлайн-банкинг, интернет-магазины, формы аутентификации. |
| SSL/TLS сертификат | Не требуется. | |  | | --- | |  |  |  | | --- | | Требуется для установления защищенного соединения. | |
| Подтверждение подлинности | Не подтверждает подлинность сервера. | Подтверждает подлинность сервера через SSL-сертификат, обеспечивая доверие к сайту. |
| Влияние на SEO | Меньше приоритет у поисковых систем. | Google и другие поисковики отдают предпочтение HTTPS-страницам, давая им больший рейтинг в поиске. |
| Проблемы с перехватом | Высокие риски перехвата данных. | Защищает данные от перехвата и манипуляций. |

Исходя из данных из таблицы можно сказать, что протокол HTTPS более предпочтителен для этого проекта, так как одной из задач является сохранение конфиденциальности данных.

## 1.8 Потоковая передача данных.

Потоковая передача данных это часто использующаяся функция нужная для того, чтобы передавать данные по сети, не загружая их в память компьютера полностью этот метод полезен при работе с большими файлами или архивами.

Для передачи файла нужно загрузить его в оперативную память компьютера, но может быть так, что файлы больше чем объем памяти выделенный операционной системой программе. Для решения этой проблемы можно запросить больше памяти через malloc, но ОЗУ не бесконечно, поэтому можно использовать потоковую передачу данных. Сервер или компьютер считывает небольшую часть файла и генерирует поток к клиенту, далее клиент принимает исходный файл по частям. Этот метод дает большие возможности чтобы передавать большие файлы. Большие файлы или группы файлов удобно передавать архивами. Потоковая передача данных может ускорить передачу этого архива пользователю. В то время, когда создается архив, его уже можно передавать с помощью потоковой передачи, будет передаваться уже заархивированная часть файла или группы файлов.

## 1.9 Архитектура сетей и IPv4.

Схема интернет сетей от провайдера до домашней локальной сети:

Интернет-провайдер (ISP)

├── Магистральный канал

│ └── Передача данных к абоненту (оптоволокно, кабель)

└── Абонентское оборудование

├── Модем (DSL, DOCSIS, GPON)

└── Роутер

├── NAT (преобразование адресов)

├── Локальная сеть (LAN)

│ ├── Проводное соединение

│ │ ├── Коммутатор (Switch)

│ │ │ ├── ПК

│ │ │ └── Игровая консоль

│ │ └── Прямое соединение

│ └── Беспроводное соединение (Wi-Fi)

│ ├── Точка доступа (Access Point)

│ └── Устройства

│ ├── Смартфоны

│ ├── Планшеты

│ ├── Умные устройства (IoT)

│ └── Ноутбуки

└── Межсетевой экран (Firewall)

Для получения доступа в интернете от одного устройства к другому нужно знать его IPv4 (Internet Protocol v4) адрес и открытый порт. Но этого не достаточно так как сейчас большинство провайдеров ‘прячут свои роутеры’ за NAT. Поэтому нужно иметь белый IP адрес. После получения белого IP адреса можно открыть порт на роутере в меню настроек. Далее для получения доступа к домашней локальной сети есть 2 варианта:

1. Подключение напрямую по IP адресу и открытому порту.
2. Создание VPN для домашней сети.

Сравнение первого и второго варианта в таблице:

|  |  |  |
| --- | --- | --- |
| Критерий | Белый IP + Открытый порт | VPN-соединение |
| Доступность | Прямой доступ к устройствам из интернета | Требуется настройка VPN-клиента на каждом устройстве |
| Безопасность | Уязвимость к прямым атакам (сканирование портов, DDoS) | Высокая безопасность благодаря шифрованию |
| Простота настройки | Легко настроить с помощью NAT и проброса портов | Настройка VPN требует знаний и программного обеспечения |
| Скорость соединения | Скорость зависит только от пропускной способности провайдера | Возможны задержки из-за шифрования данных |
| Стабильность | Высокая, нет зависимости от сторонних серверов | Зависит от стабильности VPN-сервера |
| Анонимность | IP-адрес устройства виден в интернете | IP-адрес устройства скрыт, используется IP VPN-сервера |

Разберем оба варианта: в первом случае после получения статического IP адреса например 255.255.255.255 и открытии порта к примеру 80. Мы сможем подлечиться к серверу запущенному на пк в локальной сети по адресу 255.255.255.255:80. Во втором случае нам нужно настроить свой vpn сервер, здесь есть еще два пути:

1. Объединять два и более устройств в локальную сеть через третье. Схема работы: арендуется сервер VPN, он будет центральным узлом, к которому будут подключаться клиенты. Он может быть либо на домашнем сервере, но у него должен быть статический белый ip адрес, либо в облаке например AWS или Azure. Устройства будут подключаться к одному серверу и каждому подлеченному устройству будет назначен виртуальный ip адрес, по этим адресам устройства будут объединены в одну локальную сеть.
2. Если дома есть статический ip адрес, то можно поднять свой VPN сервер и через него подключаться к домашней локальной сети.

Для создания своего VPN сервера можно воспользоваться outline vpn. Для примера я сделаю это на компьютере с операционной системой Ubuntu. Этапы создания:

1. С сайта outline нужно скачать outline manager и запустить его.
2. Нужно выбрать разместить сервер где угодно.
3. После чего нужно ввести команду в терминал и получить ключ
4. Ключ нужно будет скопировать в outline client.

После получения доступа к домашней локальной сети можно обращаться к устройствам по внутренним ip адресам из локальной сети.

## 1.10 Использование сервисов Apache, ngrok, localtunel.

Все эти сервисы используются для создания туннелей для удаленного доступа к локально запущенным приложениям.

1. Apache – это один из самых популярных сервисов для обслуживания запросов из интернета или локальной сети и перенаправления их. Работает по принципу клиент-серверной архитектуры. Клиенты (обычно браузеры или другие устройства) отправляют запросы к серверу Apache, который обрабатывает их и возвращает соответствующий ответ.
2. Ngrok – это инструмент для создания защищённых туннелей к серверу, находящемуся в локальной сети, через публичный интернет. Для его работы нужно установить ngrok на сервере. После изменения настроек и указания порта будет создан зашифрованный туннель к одному из серверов ngrok. Сервер выдаст доменный адрес, по которому можно будет подключаться к своему приложению. Когда клиент отправит запрос на доменный адрес, ngrok передаст его по туннелю с TSL шифрованием на наш домашний сервер.
3. LocalTunel – это аналог ngrok работающий с ним по схожему принципу. Отличием является возможность настроек, в LocalTunel нельзя зарезервировать доменный адрес.

## 1.11 Работа зашифрованных туннелей.

TLS-туннель — это зашифрованное соединение, которое создаётся для передачи данных между двумя устройствами (например, компьютером и сервером) через интернет.

Создание туннеля происходит в несколько этапов:

1. Клиент отправляет на сервер запрос что-то в виде Handshake как у wi-fi. В нем говориться я (IP), вот список известных мне видов шифрования.
2. Сервер выбирает один из видов шифрования и отправляет подписанный сертификат шифрования.
3. Обмен ключами. Клиент и сервер обмениваются кодом и генерируется симметричный ключ, это делается для безопасности, чтобы данные шифрования были в безопасности.
4. Создание туннеля. Теперь клиент и сервер обмениваются данными с помощью протокола ‘tcp’.