|  |  |
| --- | --- |
| *Gerb-BMSTU_01* | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ  *Информатика и системы управления*

КАФЕДРА *Программное обеспечение ЭВМ и информационные технологии*

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

Сетевое приложение для передачи и обработки пользовательского видеоряда

Студент ИУ7-75Б  К. К. Руднев

(Группа) (Подпись, дата) (И.О.Фамилия)

Студент ИУ7-75Б  С. А. Мирзоян

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы  Н.О. Рогозин

(Подпись, дата) (И.О.Фамилия)

Москва, 2020 г.

**Содержание**

[**Введение** 4](#_Toc59667037)

[**1.** **Аналитический раздел** 5](#_Toc59667038)

[**1.1** **Описание предметной области** 5](#_Toc59667039)

[**1.2** **Метод проектирования приложения** 6](#_Toc59667040)

[**1.3** **Метод проектирования протокола** 7](#_Toc59667041)

[**1.3.1** **Транспортные протоколы** 7](#_Toc59667042)

[**1.3.2** **Прикладные протоколы** 8](#_Toc59667043)

[**1.3.2.1** **Протокол FTP** 9](#_Toc59667044)

[**1.3.2.2** **Протокол HTTP** 9](#_Toc59667045)

[**1.4** **Методы шифрования** 10](#_Toc59667046)

[**1.5** **Децентрализованная система серверов** 11](#_Toc59667047)

[**1.6** **Вывод** 12](#_Toc59667048)

[**2.** **Конструкторский раздел** 13](#_Toc59667049)

[**2.1** **Формализация задачи** 13](#_Toc59667051)

[**2.1.1** **IDEF0 диаграмма** 13](#_Toc59667052)

[**2.1.2** **Диаграмма способов использования** 18](#_Toc59667053)

[**2.2** **Разработанные и используемые структуры данных** 18](#_Toc59667054)

[**2.2.1** **Структуры данных протокола при получении данных пользователя** 18](#_Toc59667055)

[**2.2.2** **Структуры данных протокола при получении результатов обработки** 19](#_Toc59667056)

[**2.3** **Алгоритм взаимодействия пользователя и протокола** 19](#_Toc59667057)

[**2.4** **Вывод** 20](#_Toc59667058)

[**3.** **Технологический раздел** 21](#_Toc59667059)

[**3.1** **Обоснование выбора языка, среды и платформы программирования** 21](#_Toc59667061)

[**3.2** **Выбор сервера** 22](#_Toc59667062)

[**3.2.1** **Сервер для получения файла с клиента** 22](#_Toc59667063)

[**3.2.2** **Сервер для обработки полученных данных** 22](#_Toc59667064)

[**3.3** **Описание структуры программы** 23](#_Toc59667065)

[**3.3.1** **Работа на клиенте** 23](#_Toc59667066)

[**3.3.2** **Работа на сервере 1** 25](#_Toc59667067)

[**3.3.3** **Работа на сервере 2** 27](#_Toc59667068)

[**3.4** **Реализация общения нескольких серверов** 30](#_Toc59667069)

[**3.5** **Результаты работы приложения** 32](#_Toc59667070)

[**3.5.1** **Клиент** 32](#_Toc59667071)

[**3.5.2** **Сервер 1** 33](#_Toc59667072)

[**3.5.3** **Сервер 2** 34](#_Toc59667073)

[**3.5.4** **Интерфейс приложения** 34](#_Toc59667074)

[**3.6** **Выводы** 35](#_Toc59667075)

[**Заключение** 36](#_Toc59667076)

[**Список используемой литературы** 37](#_Toc59667077)

# **Введение**

Человечество всегда пыталось развиваться и двигать вперед существующие технологии. Так на рубеже 40-х – 50-х годов появилась первая ЭВМ. С тех пор они совершенствовались. Для них создавались более сложные прикладные программы. Было лишь одно ограничение. Все запускалось локально. С тех пор человечество начало думать, как связать хотя бы два устройство так, чтобы данные можно было передавать удаленно…

Спустя годы, на рубеже 60-х – 70-х годов [2], удалось совершить первый удаленный сеанс связи с помощью сети ARPANET. Протоколы, лежащие в ее основе, в будущем станут основой сети Internet, какой мы знаем ее сегодня.

И вот уже сегодня люди активно используют сеть Internet. Загружают полезную информацию, создают огромные хранилища данных или просто посылают друг другу смешные картинки. Сегодня, в эпоху потребления контента, когда визуализировать и передавать данные так же важно, как и хотя бы иметь их, необходимы инструменты для этого.

Необходимо уметь создавать контент и делиться им. И если создание контента – не наша цель, то организовать способ его передачи между устройствами – цель этой работы.

В работе будет предложен протокол, который позволить обмениваться видеофайлами между двумя устройствами для дальнейшей обработки и возврата какого-то результата с соблюдением конфиденциальности данных пользователя.

# **Аналитический раздел**

В аналитическом разделе будет дано описание выбранной предметной области, будут предъявлены требования к функционалу, а также будут рассмотрены возможные существующие методы для выполнения одного или нескольких требований.

## **Описание предметной области**

Разрабатываемое приложение базируется на работе с сетью. Это значит, что приложение доступно не только локально, но и удаленно при условии, что конечный пользователь знает расположение клиентской части в сети Интернет. Для этого необходимо разработать несколько программных компонентов. Среди них клиентская часть, она же пользовательская, непосредственно с которой и взаимодействует конечный потребитель, а также серверная, та часть, которая реализует основную работу приложения. И самая важная часть – сам протокол, по которому реализуется передача файла.

В совокупности приложение должно иметь возможность обрабатывать файлы любого размера, где единственное ограничение – размер хранилища сервера, передавать обработанные результаты обратно пользователю.

В эпоху развития понятий информационной безопасности исходные пользовательские данные должны быть скрыты от внешнего мира. Эти данные нельзя никак прочитать без специального ключа, кодовой фразы и т.п. То есть данные должны передаваться от клиента к серверу в зашифрованном виде.

И самое важное требование – сохранность данных. Приложение должно обработать ровно те данные, что предоставил пользователь без каких-либо изменений в процессе доставки, например, потери пакетов или добавления лишних сегментов.

После обработки данные должны быть возвращены пользователю. Они могут представлять из себя сырые данные, с которыми будет взаимодействовать пользователь или в своем проекте, или для передачи в сторонние сервисы, или данные могут быть отображены для ознакомления с результатами. Это может быть веб-страница, если речь о браузерном приложении, или любое другое решение, позволяющее ознакомиться с результатом работы приложения.

## **Метод проектирования приложения**

Для проектирования подобного приложения с взаимодействием с сетью существует только два подхода – предоставление пользователю приложения, которое целиком находится в Сети, и предоставление пользователю клиента, который может быть как локальным для пользователя, так и удаленным. Последний подход позволяет снизить требования к ресурсам пользователя. Такой подход основывается на клиент-серверной архитектуре. Ее суть кратко отражена на рисунке 1.1.



Рис. 1.1. Схема клиент-серверной архитектуры

Принцип такого взаимодействия в том, что есть две стороны: отправляющая – клиент и принимающая – сервер. Клиент будет предоставлять данные на одном конце соединения, сервер будет их обрабатывать на другом.

## **Метод проектирования протокола**

Разрабатываемый протокол опирается на два уровня модели OSI – транспортный – для передачи данных от клиента к серверу – и прикладной – для предоставления доступа к переданным данным с помощью запросов или другого механизма.

### **Транспортные протоколы**

Основных транспортных протоколов, которые позволят осуществить передачу данных от клиента к серверу, два: TCP и UDP [1].

Разница между ними в том, что первый оперирует сегментами данных, второй – дейтаграммами. Однако главное отличие, что и повлияет на выбор того или оного протокола, в том, что TCP доставляет данные гарантированно, а UDP не гарантировано.

Если происходит ошибка и данные не доходят до получателя, протокол посылает их заново. Это важно потому, что требование передачи файлов любых размеров обязует иметь возможность передачи и невероятных больших, например, размером в 1GB+, файлов. В чистом виде без разбиения на пакеты невозможно передать такие большие объемы данных. Однако при разбиении файла на пакеты и их пересылке есть риск потерять некоторые пакеты, что приведет к невозможности получить целый исходный файл. Протокол TCP исключает этот риск.

TCP исключает потери данных, дублирование и перемешивание пакетов, задержки. UDP все это допускает, и соединение для работы ему не требуется. Процессы, которым данные передаются по UDP, должны обходиться полученным, даже и с потерями. TCP контролирует загруженность соединения, UDP не контролирует ничего, кроме целостности полученных датаграмм. Однако, положительной стороной UDP является его скорость, благодаря которой работают приложения, которые рассчитаны на широкую пропускную способность и быстрый обмен, такие как, например, сетевые и браузерные игры.

Краткое сравнение протоколов представлено в таблице 1.1.

*Таблица 1.1. Сравнение протоколов TCP и UDP*

|  |  |  |
| --- | --- | --- |
|  | TCP | UDP |
| Размер заголовка, байт | 20-60 | 8 |
| Форма передачи данных | Сегмент | Датаграмма |
| Надежность | Да | Нет |
| Упорядоченность | Да | Нет |
| Контроль перегрузок | TCP Congestion Avoidance Algorithm | Нет |
| Тяжеловесность | Дополнительные 3 пакеты для установки соединения | Дополнительные пакеты не требуются |
| Применение | Там, где нужна надежность и упорядоченность | Там, где высокая нагрузка на сервер и потеря некоторых пакетов некритична |

### **Прикладные протоколы**

Для обеспечения взаимодействия с файлом можно выделить следующие прикладные протоколы: FTP (File Transfer Protocol), HTTP (HyperText Transfer Protocol) [1].

#### **Протокол FTP**

Один из старейших протоколов, разработанный в 1971 году задолго до HTTP и TCP/IP. Он служит для передачи больших бинарных файлов, гарантирует передачу (либо выдачу ошибки) за счёт применения квотируемого протокола.

Однако старость данного протокола, и предпочтение ему протокола HTTP, приводит к очевидному заключению о том, что «вымирание» FTP является лишь вопросом времени, и стоит рассмотреть более современные протоколы.

#### **Протокол HTTP**

HTTP был предусмотрен для передачи, изначально — в виде гипертекстовых документов в формате HTML, однако со временем он стал использоваться также для передачи небольших файлов с явным указанием типа файла (MIME-type).

Для реализации приложения можно было бы взять готовый протокол FTP, который решил бы все проблемы, однако необходимо решить задачу разработки протокола.

Разрабатываемый протокол будет представлять собой некую надстройку над протоколом HTTP, что позволит передавать данные любого размера.

В основе лежит простейший метод деления файла, что представляет собой подготовку файла перед отправкой, а именно разбиение на n-частей, которые могут быть переданы по протоколу HTTP. Варьирование размеров разбиения позволит оптимизировать нагрузку на сеть и повысить скорость передачи файла. Принцип работы отображен на рисунке 1.2.

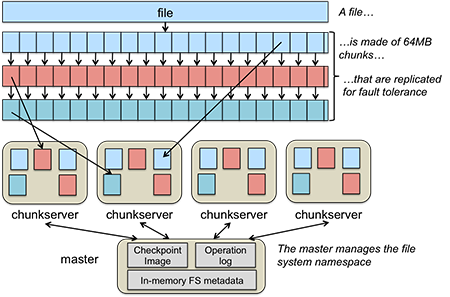


Рис. 1.2. Метод деления файла

В условиях работы с HTTP этот метод имеет немалое значение, потому что этот протокол работает с текстовыми данными. Для того, чтобы отправить произвольный бинарный файл, его необходимо закодировать в base64 строку. А это чревато увеличением размера исходного файла в 1.33 раза. Это следует из следующих соображений: каждый символ использует для представления 6 бит, потому что log2(64) = 6. Поэтому 4 символа используют для представления 4 \* 6 = 24 битов = 3 байта. Таким образом, необходимо 4\*(n/3) символов для представления n байтов, и это должно быть округлено до кратного 4.

## **Методы шифрования**

Для шифрования данных существует две группы алгоритмов: симметричные и асимметричные алгоритмы. Из этих групп подлежат рассмотрению следующие алгоритмы: Enigma, DES, AES, RSA. Алгоритмы из первой группы не могут использоваться из-за слишком большого времени обработки данных, что может вызвать достаточную задержку на больших объемах данных. А из второй группы не может использоваться алгоритм RSA. Дело в том, что он позволяет шифровать данные, размер которых не больше ключа. Это привносит требование ограничения размеров входного файла или бесконечного размера ключа. Оба варианта недопустимы.

Также еще одним требованием стала поддержка кроссплатформенного переноса алгоритма без непосредственной разработки с нуля. Поскольку шифрование – важная, но не основная задача, она будет реализовываться с использованием готовых библиотек.

Отличным выбором станет алгоритм AES, который сочетает в себе высокую скорость работы и поддержку данных любого размера.

## **Децентрализованная система серверов**

Для разделения функционала и распределения нагрузки используется система из нескольких серверов, которые работают независимо друг от друга и взаимодействуют друг с другом, только когда приходит запрос на обработку. В рамках проекта рассматривается система, в которой все сервера системы находятся на одной машине и разделяют единое хранилище. Однако данная система вполне может быть реализована и при удаленном расположении всех серверов без существенных изменений реализации самих серверов.

Сперва данные приходят на первый сервер, который занимается приемом и расшифровкой данных. Временно, на время работы приложения, данные сохраняются в хранилище. После завершения первой фазы на второй сервер поступает запрос, предписывающий обработать данные. Этот сервер занимается обработкой расшифрованных данных и может выполнять любые операции, связанные с этим. Для разрабатываемого приложения это обработка видеоряда с помощью нейронных сетей. По завершении второй фазы второй сервер возвращает ответ на первый сервер, который возвращает результат уже пользователю.

## **Вывод**

В рамках раздела было дано описание предметной области. Было дано определение используемой архитектуры приложения, рассмотрены протоколы, которые могут лечь в основу разрабатываемого протокола, а также методы, которые могут помочь в разработке. Определена система из нескольких серверов-обработчиков, которые работают над своей задачей.

Таким образом, приложение разрабатывается с клиентом и несколькими серверами, с шифрованием пользовательских данных по алгоритму AES и передача данных по протоколу, который базируется на HTTP, с доставкой данных по протоколу TCP.

# **Конструкторский раздел**

В конструкторском разделе будет проведен разбор используемых алгоритмов, будет проведена формализация задачи – представлены все необходимые диаграммы для понимания сути проекта.



## **Формализация задачи**

В текущем подразделе будут представлены все необходимые диаграммы проекта для формализации поставленной задачи.

### **IDEF0 диаграмма**

На рисунках 2.1 – 2.5 представлена IDEF0 диаграмма.

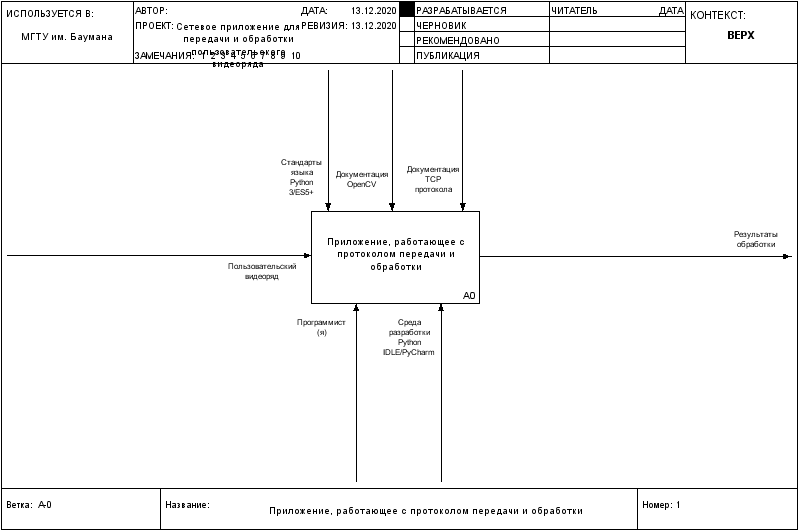


Рис. 2.1. Приложение, работающее с протоколом передачи данных

Приложению требуется клиент и два сервера для своей работы. Поэтому вначале необходимо их поднять, чтобы иметь возможность обращаться к приложению.

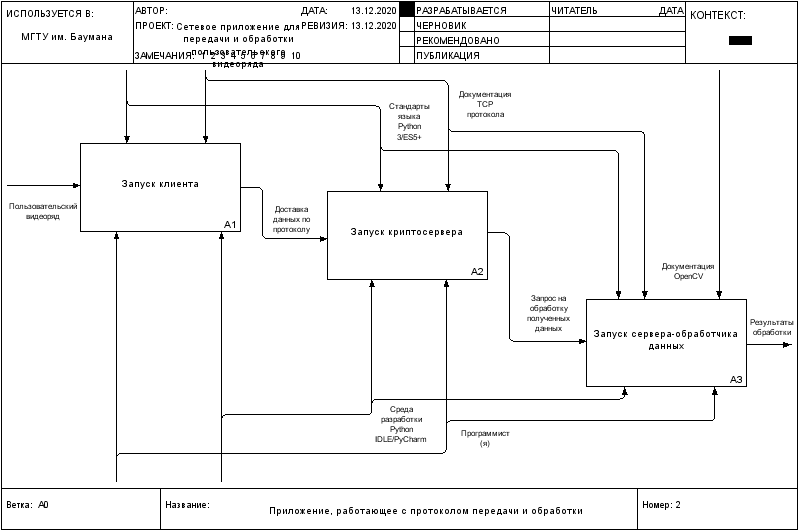


Рис. 2.2. Декомпозиция первого уровня

Работа представляет собой следующую последовательность: получение данных от пользователя на клиенте, подготовка и передача полученных данных на первый сервер, преобразование и временное сохранение полученных данных, обработка сохраненных данных вторым сервером, возврат результатов на первый сервер, возврат результатов на клиент.

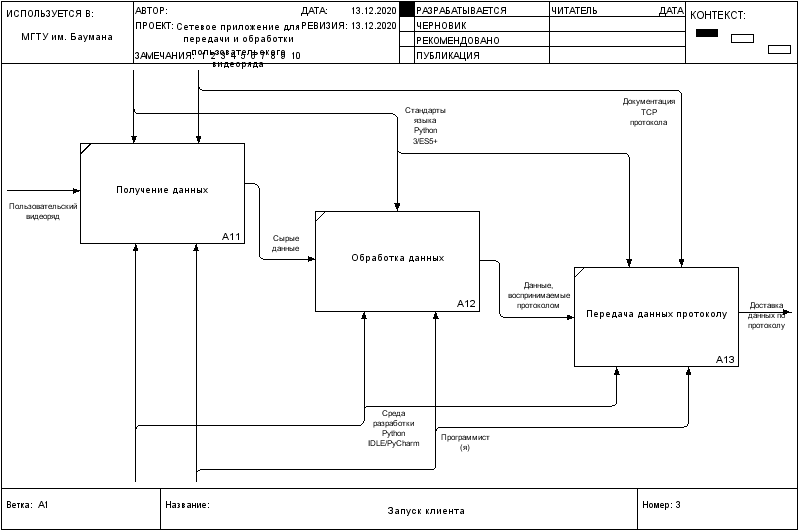


Рис. 2.3. Декомпозиция работы клиента

Протокол работает так, что сперва он должен принять данные от клиента, как-то их сохранить, а затем передать расшифрованные (преобразованные) данные прикладной части, реализация которой может варьироваться в зависимости от требований к обработке.

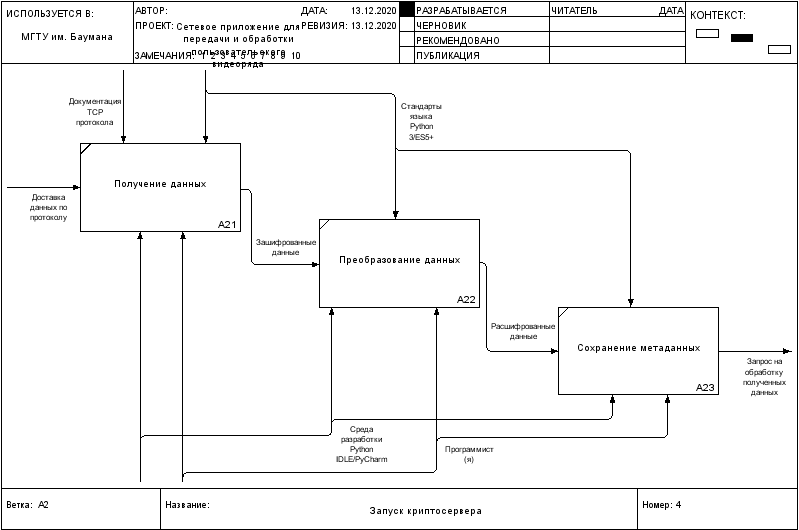


Рис. 2.4. Декомпозиция работы первого сервера

Протокол требует наличия двух и более серверов для реализации задумки. Первый сервер должен заниматься аутентификацией, если это поддерживается сервисом, и шифровкой/расшифровкой данных.

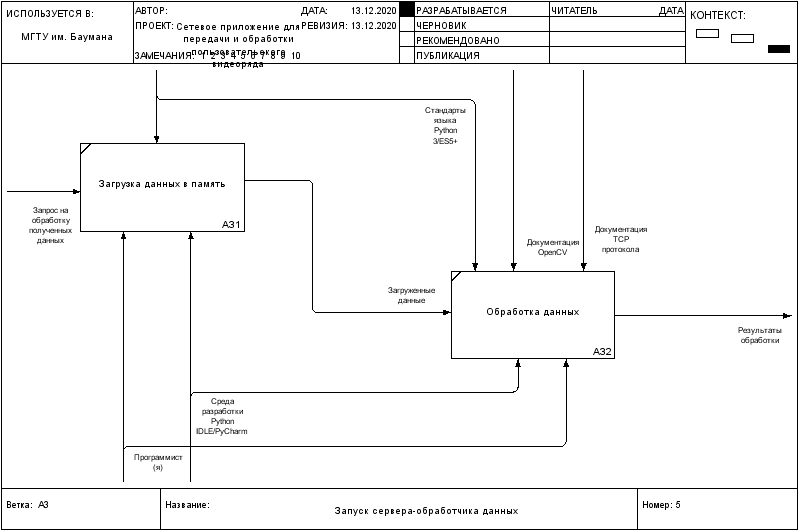


Рис. 2.5. Декомпозиция работы второго сервера

Когда данные успешно получены и расшифрованы, второму и последующим, если есть, серверам передаются или данные, если сервера располагаются удаленно, или местоположение загруженного файла, если сервера располагаются на одной машине и разделяют пользовательское адресное пространство.

### **Диаграмма способов использования**

На рисунке 2.6 представлена диаграмма способов использования, объясняющая возможные сценарии использования приложения.

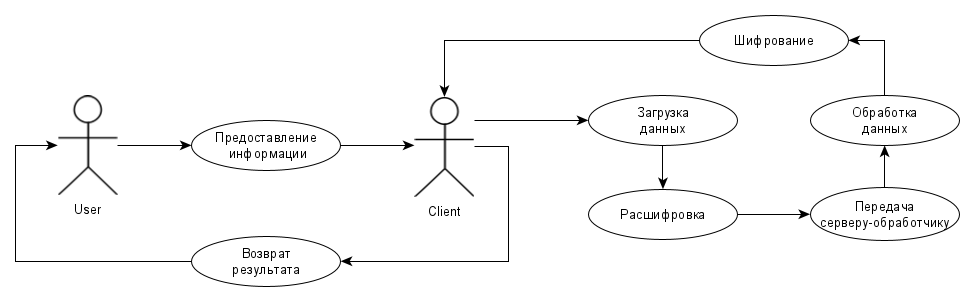


Рис. 2.6. Диаграмма способов использования

## **Разработанные и используемые структуры данных**

В данном подразделе будут описаны разработанные для решения задачи структуры и типы данных.

### **Структуры данных протокола при получении данных пользователя**

Для передачи файла от клиента к серверу используется подход разбиения файла на некоторое количество кусков. Каждый такой кусок инкапсулируется в отдельный пакет. Устройство такого пакета представлено в таблице 2.1.

*Таблица 2.1. Пакет передачи куска файла*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| chunk | chunk\_num | total | file\_name | file\_size |
| … | 1 | N | … | … |

Поле chunk используется для хранения самих данных в зашифрованном виде.

Поле chunk\_num используется для определения номера куска файла.

Поле total используется для определения общего количества разбиений.

Поле file\_name используется для хранения имени оригинального файла.

Поле file\_size используется для хранения размера оригинального файла.

### **Структуры данных протокола при получении результатов обработки**

Для отправки результата обратно клиенту используется еще одна структура данных. Она определяется результатами обработки. Её устройство представлено в таблице 2.2.

*Таблица 2.2. Структура данных для возврата результата*

|  |
| --- |
| process\_name |
| … |

Поле process\_name обладает переменным названием. По названию определяется операция, которая производилась при обработке. Используется для хранения результата обработки для конкретной операции.

## **Алгоритм взаимодействия пользователя и протокола**

На рисунке 2.7 представлена схема разработанного алгоритма для взаимодействия с протоколом.

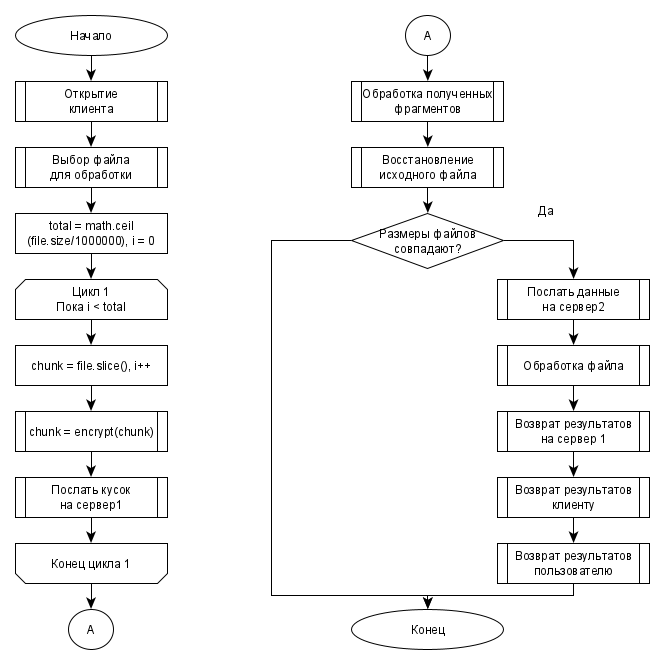


Рис. 2.7 Схема алгоритма

## **Вывод**

В данном разделе была проведена формализация задачи, было приведено множество диаграмм, на основании которых можно начинать проектировать архитектуру программного обеспечения и в рамках нее писать функционирующий код. На основании требований предметной области было дано описание структур данных. Сейчас совершается переход от теоретического изложения задачи к практическому ее выполнению.

# **Технологический раздел**

В технологическом разделе будет дано пояснение по выбору конкретного способа реализации, выбору того или иного средства/инструментария, будет проведен обзор и выбор конкретного сервера для каждой задачи.



## **Обоснование выбора языка, среды и платформы программирования**

Для создания серверной части проекта был выбран язык программирования Python версии 3 [8]. Это язык программирования, поддерживающий объектно-ориентированную парадигму (ООП), обладающий поддержкой фреймворков для написания графического пользовательского интерфейса, большим количеством стандартных библиотек для выполнения часто используемых задач. В языке отсутствует статическая типизация данных, что позволяет не отвлекаться на выбор типа для той или иной переменной, содержащей информацию. Также для него предоставляется ряд библиотек, позволяющих разворачивать собственные сервера в пару строк кода, например Flask [7], Django [5], CherryPy [3]. Помимо того, данный язык программирования обладает встроенным сборщиком мусора, что позволяет не отвлекаться на выделение и очистку памяти.

Язык Python является интерпретируемым, потому приложение, написанное на нем, запустится везде, где есть этот интерпретатор и необходимые библиотеки. Поэтому для разработки можно выбрать любую доступную среду. Я остановил свой выбор на IDE PyCharm 2019.3 от JetBrains [10]. Эта среда позволяет отслеживать зависимости файлов, одним кликом переходить к реализациям функций/методов/классов, а также обладает встроенной подсветкой синтаксиса и некоторыми другими вещами, упрощающими разработку. Также PyCharm работает «из коробки», для ее работы не нужно что-то устанавливать кроме интерпретатора Python и необходимых библиотек. Однако если что-то понадобится, эта среда обладает поддержкой более 1000 совместимых плагинов. И для проектов с открытым исходным кодом является бесплатной.

Для создания клиентской части проекта был выбран язык программирования JavaScript стандарта EcmaScript 5+ [6]. Это язык программирования, разработанный для управления содержимым веб-страницы. Язык является интерпретируемым и выполняется в браузере. Был выбран, потому что при разработке браузерного веб-приложения нет других вариантов, кроме использования WebAssembly [9] нет.

Соответственно, для работы с кодом JavaScript была выбрана среда IDE WebStorm 2020.1 от JetBrains [11]. Эта среда дополняет экосистему JetBrains для разработки, обладает всеми возможностями, что и PyCharm, только направлена на работу с JavaScript, HTML, CSS. Данная среда разработки распространяется лишь на коммерческой основе, однако для студентов есть бесплатная лицензия, что было использовано для проекта.

## **Выбор сервера**

В данном подразделе будут описаны используемые серверы для построения децентрализованной системы.

### **Сервер для получения файла с клиента**

Клиентская часть запускается из веб-браузера. Соответственно, первичное взаимодействие с протоколом будет осуществляться именно оттуда. Поскольку протокол разрабатывается поверх протокола HTTP, запросы будут выполняться с помощью него, но с некоторой надстройкой.

Работа с HTTP означает необходимость взаимодействия с CORS [4]. Для того, чтобы не писать вручную обработку лишнего OPTIONS метода, а также специфические заголовки в ответе, было решено использовать Flask фреймворк с CherryPy сервером. Это позволит не отвлекаться на CORS, поскольку в этой библиотеке есть свое решение этой проблемы.

### **Сервер для обработки полученных данных**

Сервер обработки по своей природе чем-то напоминает HTTP сервер, лишь за тем исключением, что ему не нужно обрабатывать методы HTTP и возвращать специфические заголовки. По этой причине этот сервер пишется с нуля с использованием системных сокетов, подключение к которым производится по TCP протоколу.

## **Описание структуры программы**

В данном подразделе будет проведет детальный обзор структуры приложения, включающий в себя обзор разработанных классов и их назначение.

### **Работа на клиенте**

На листинге 3.1 представлен код для разделения файла на несколько пакетов данных для передачи.

Листинг. 3.1 Обработка на клиенте

const toBase64 = file => new Promise((resolve, reject) => {  
 const reader = new FileReader();  
 reader.readAsDataURL(file);  
 reader.onload = () => resolve(reader.result);  
 reader.onerror = error => reject(error);  
});  
  
async function Main() {  
 const file = document.querySelector('#video\_file').files[0];  
 const base64video = await toBase64(file);  
  
 const chunkSize = 1000000;  
 const numChunks = Math.ceil(base64video.length / chunkSize);  
 console.log(base64video.length);  
 localStorage.setItem('current\_file', file.name);

for (let i = 0; i < numChunks; i++)  
 {  
 let flag = true;  
 let start = i\*chunkSize;  
 let end = Math.min(start + chunkSize, base64video.length);  
 let chunk = encrypt(base64video.slice(start, end), key);  
  
 let httpRequest = new XMLHttpRequest();  
 httpRequest.onreadystatechange = function()  
 {  
 if(httpRequest.readyState === 4 && httpRequest.status !== 200) {  
 alert('error');  
 flag = false;  
 }  
 }  
 httpRequest.open('POST', 'http://localhost:1010/upload2', false);  
 httpRequest.setRequestHeader('Content-Type', 'application/json;charset=utf-8');  
 httpRequest.send(JSON.stringify({  
 "chunk": chunk,  
 "chunk\_num": i+1,  
 "total": numChunks,  
 "file\_name": file.name,  
 "file\_size": file.size  
 }));  
 if (!flag) {  
 break;  
 }  
 }  
}  
const file = document.querySelector('#video\_file');  
file.addEventListener("change", Main, false);

Функция toBase64 необходима для кодирования бинарного содержимого файла в строковое представление, которое можно передать по HTTP.

Функция Main получает загруженный файл, который определяется через изменение состояния кнопки (загрузка файла на стороне клиента не относится к работе протокола и может быть реализовано любым удобным способом), а затем запускает основной процесс разбиения и пересылки.

Реализация защиты данных путем шифрования производится с помощью функции encrypt. Encrypt шифрует данные по алгоритму AES с ключом заданным в протоколе.

### **Работа на сервере 1**

На листинге 3.2 представлен код для расшифровки и сборки файла из несколько пакетов данных после передачи с клиента.

Листинг. 3.2 Обработка на сервере 1

@blueprint.route('/upload2', methods=['POST'])  
@flask\_cors.cross\_origin()

def upload2():  
 data = request.json  
  
 save\_path = os.path.join(config.data\_dir, secure\_filename(data["file\_name"]))  
 save\_path\_metadata = os.path.join(config.data\_dir, "metadata" + secure\_filename(data["file\_name"]))  
 current\_chunk = int(data["chunk\_num"])

if os.path.exists(save\_path) and current\_chunk == 1:  
 return make\_response(('File already exists', 400))  
  
 try:  
 with open(save\_path\_metadata, 'a') as f:  
 chunk\_to\_write = decrypt(data["chunk"].encode("ASCII"), key)  
 if current\_chunk == 1:  
 f.write(chunk\_to\_write[22:])  
 else:  
 f.write(chunk\_to\_write)  
 except OSError:  
 log.exception('Could not write to file')  
 return make\_response(("Not sure why,"  
 " but we couldn't write the file to disk", 500))  
  
 total\_chunks = int(data['total'])  
  
 if current\_chunk == total\_chunks:  
 with open(save\_path\_metadata, 'r') as f2:  
 data\_to\_convert = f2.read()  
 with open(save\_path, 'wb') as fh:  
 fh.write(base64.b64decode(data\_to\_convert))  
  
 if os.path.getsize(save\_path) != int(data['file\_size']):  
 log.error(f"File {data['file\_name']} was completed, "  
 f"but has a size mismatch."  
 f"Was {os.path.getsize(save\_path)} but we"  
 f" expected {data['file\_size']} ")  
 return make\_response(('Size mismatch', 500))

else:  
 os.remove(save\_path\_metadata)  
 log.info(f'File {data["file\_name"]} has been uploaded successfully')  
 r = requests.get('http://MeSSivO2:1010/'+data["file\_name"])  
 return make\_response((r.text, 200))  
 else:  
 log.debug(f'Chunk {current\_chunk + 1} of {total\_chunks} '  
 f'for file {data["file\_name"]} complete')  
  
 return make\_response(("OK", 200))

Функция upload2 обрабатывает входящий пакет от клиента. Она анализирует содержимое следующим образом: сначала проверяется номер текущего фрагмента – если это первый фрагмент и файл уже сохранен, значит файл уже существует и где-то произошла ошибка, иначе выполнение функции продолжается –, затем производится попытка доступа к диску – если успешно, производится расшифровка фрагмента и записываются метаданные, причем если фрагмент первый, необходимо пропустить 22 байта, потому что файл кодируется в BASE64 и 22 байта являются служебной информацией для стандарта. После того, как все фрагменты считаны и расшифрованы, производится сохранение бинарного файла с оригинальным названием. И в случае, если полученный и переданный размеры совпадают, протокол посылает команду для перехода ко второй фазе.

### **Работа на сервере 2**

На листинге 3.3 представлен код самописного сервера-обработчика.

Листинг. 3.3 Обработка на сервере 2

class WebServer(object):  
 def \_\_init\_\_(self, port=9999):  
 self.host = socket.gethostname().split('.')[0]  
 self.port = port  
  
 def start(self):  
 self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 try:  
 print("Starting server on {host}:{port}".format(host=self.host, port=self.port))  
 self.socket.bind((self.host, self.port))  
 print("Server started on port {port}.".format(port=self.port))  
 except Exception as e:  
 print("Error: Could not bind to port {port}".format(port=self.port))  
 self.shutdown()  
 sys.exit(1)  
  
 self.\_listen()  
  
 def shutdown(self):  
 try:  
 print("Shutting down server")  
 self.socket.shutdown(socket.SHUT\_RDWR)  
  
 except Exception as e:  
 pass  
 def generate\_headers(self, response\_code):  
 header = ''  
 if response\_code == 200:  
 header += 'HTTP/1.1 200 OK\n'  
 elif response\_code == 404:  
 header += 'HTTP/1.1 404 Not Found\n'  
 time\_now = time.strftime("%a, %d %b %Y %H:%M:%S", time.localtime())  
 header += 'Date: {now}\n'.format(now=time\_now)  
 header += 'Server: Simple-Python-Server\n'  
 header += 'Connection: close\n\n'  
 return header  
  
 def \_listen(self):  
 self.socket.listen(5)  
 while True:  
 (client, address) = self.socket.accept()  
 client.settimeout(60)  
 print("Received connection from {addr}\n".format(addr=address))  
 threading.Thread(target=self.\_handle\_client, args=(client, address)).start()  
  
 def \_handle\_client(self, client, address):  
 PACKET\_SIZE = 200000  
 while True:  
 data = client.recv(PACKET\_SIZE).decode()  
  
 if not data:  
 break  
  
 request\_method = data.split(' ')[0]  
  
 if request\_method == "GET":  
 file\_requested = data.split(' ')[1]  
 file\_requested = file\_requested.split('?')[0][1:]  
 print(file\_requested)  
 response\_data = handle\_file(file\_requested)  
 response\_header = self.generate\_headers(200)  
 response = response\_header.encode()  
 response += response\_data  
  
 client.send(response)  
 client.close()  
 break  
 else:  
 print("Unknown HTTP request method: {method}".format(method=request\_method))  
  
def shutdownserver(sig, unused):  
 server.shutdown()  
 sys.exit(1)  
  
signal.signal(signal.SIGINT, shutdownserver)  
server = WebServer(1010)  
server.start()  
print("Press Ctrl+C to shut down server.")

Сервер по умолчанию запускается на 9999 порте или указанном при создании. Сервер использует технологию мультипоточности для обеспечения нескольких одновременных подключений. После того, как серверу передано указание об обработке файла, происходит создание формы ответа и подготовка результата с помощью функции handle\_file().

### **Клиент**

На рисунках 3.1 – 3.2 представлен результат загрузки видеофайла на клиенте и его преобразования перед отправкой.

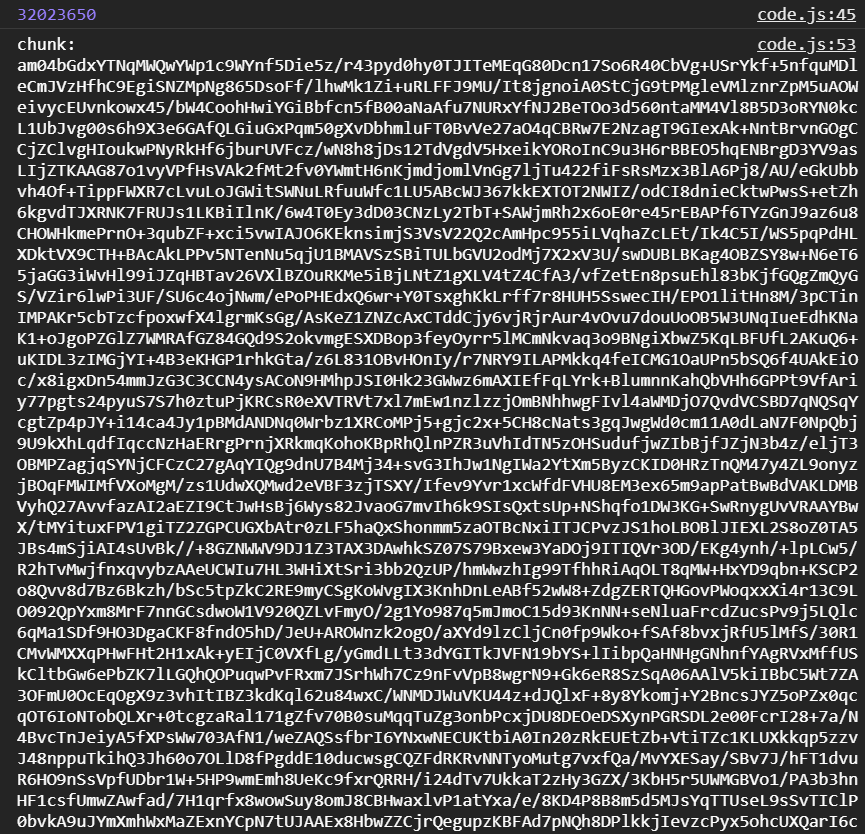


Рис. 3.1 Пакет данных. Часть 1

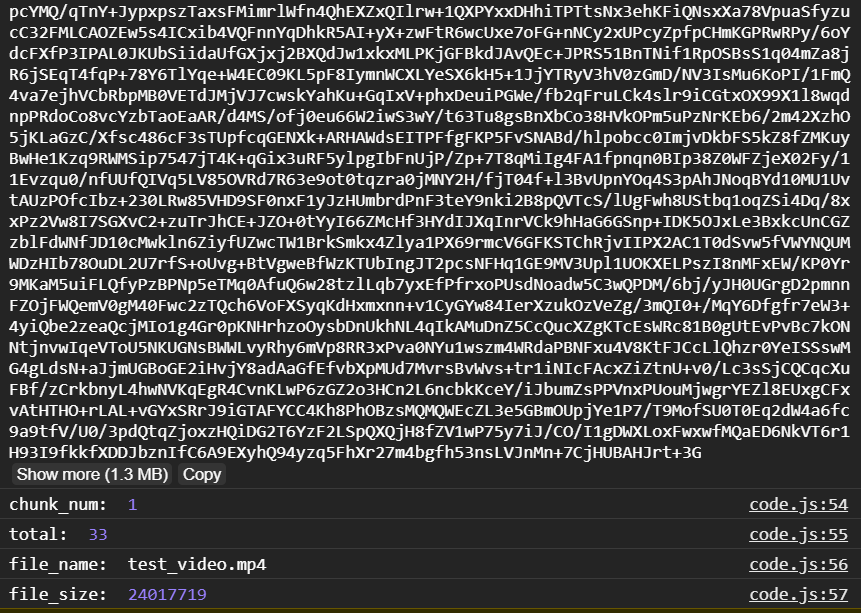


Рис. 3.2 Пакет данных. Часть 2

Видно содержание первого пакета, содержащего первый фрагмент видеофайла. Оригинальный размер файла составляет 24017719 байт, в закодированном виде – 24017719 \* 4/3 = 32023650 байт. Закодированный файл разбивается на фрагменты по 1000000 байт. В таком случае получаем 32.023650 = 33 фрагмента.

## **Реализация общения нескольких серверов**

По логике взаимодействия все серверы системы независимы друг от друга. Единственное, чем они связаны, - каждый последующий сервер получает данные от предыдущего узла в цепи работы протокола. Таким образом, для связи серверов используется служебный запрос, который сигнализирует о том, что серверу необходимо начать свою работу. По завершении ее сервер возвращает результат и продолжает слушать входящие подключения до тех пор, пока что-то снова не заставит его выполнить свою роль.

По своей природе оба сервера, используемых в реализации, являются HTTP серверами. Именно поэтому протокол является разрабатываемым поверх HTTP. Для упрощения работы, отладки и тестирования сервера связываются с помощью библиотеки requests [9] на стороне Python и стандартного объекта XMLHttpRequest [6] на стороне JavaScript. Пример использования обеих функций продемонстрирован на листингах 3.4 – 3.5 соотвественно.

Листинг. 3.4 Взаимодействие серверов. JavaScript сторона

let httpRequest = new XMLHttpRequest();  
httpRequest.open('POST', 'http://localhost:1010/upload2', false);  
httpRequest.setRequestHeader('Content-Type', 'application/json;charset=utf-8');  
httpRequest.send(JSON.stringify({  
 "chunk": chunk,  
 "chunk\_num": i+1,  
 "total": numChunks,  
 "file\_name": file.name,  
 "file\_size": file.size  
}));

Листинг. 3.5 Взаимодействие серверов. Python сторона

r = requests.get('http://MeSSivO2:1010/'+data["file\_name"])

# На отсылку обработчику

client.send(response)

# На отсылку от обработчика

## **Результаты работы приложения**

В данном подразделе будут приведены результаты работы всего приложения в целом, а также отдельных его компонент.

### **Сервер 1**

На первом сервере данные расшифровываются и собираются в целый файл. Результат такой сборки представлен на рисунке 3.3.

Заметим, что размеры файлов – оригинального и полученного – совпадают (увидеть размер оригинального файла можно на рисунке 3.1), у файла появилось превью изображения, автоматически определилось приложение по умолчанию. Это значит, что пересылка произведена успешно, операционная система понимает полученный файл.

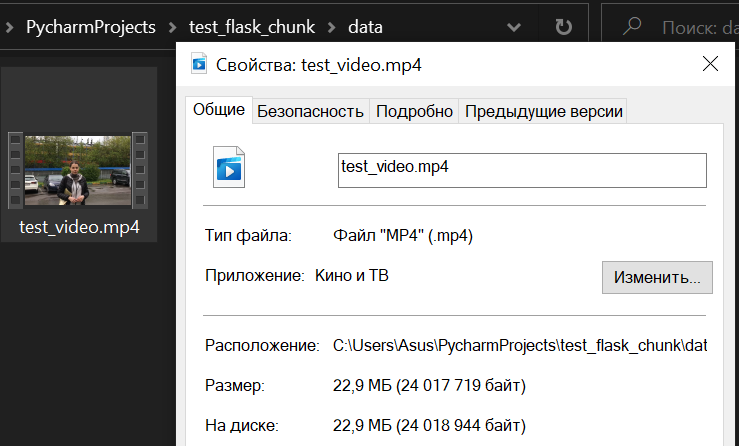


Рис. 3.3 Полученный расшифрованный файл

### **Сервер 2**

На втором сервере данные обрабатываются в соответствии с прикладной задачей. На рисунке 3.4 изображен готовый результат, который вернется на первый сервер, а впоследствии и клиенту.

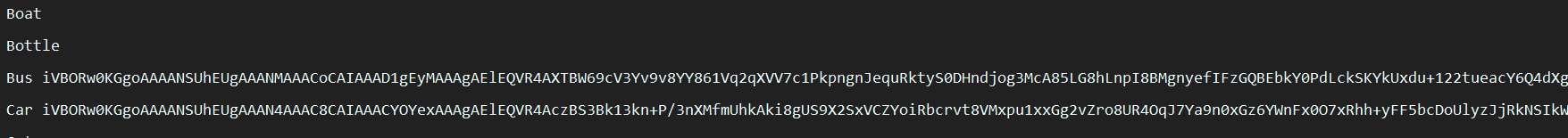


Рис. 3.4 Полученный расшифрованный файл

### **Интерфейс приложения**

На рисунке 3.5 изображен интерфейс приложения, использующего разработанный протокол. Поскольку основной акцент ставился на разработке и использовании протокола, а также демонстрации его работоспособности, внешний вид приложения далек от идеала и представляет лишь наиболее необходимые элементы.



Рис. 3.5 Пример работы приложения

## **Выводы**

В данном разделе было дано обоснование для выбора всего инструментария разработки. Была описана структура приложения. Описаны некоторые целые классы, команды, методы для решения определенной задачи в рамках приложения. Был приведен пример работы приложения и протокола.

# **Заключение**

В ходе работы было реализовано приложение, обрабатывающее пользовательский видеоряд. В своей работе это приложение использует разработанный протокол.  
Данный протокол позволяет передавать видеофайлы и принимать результат.  
При решении задач, возникали разного рода трудности. От выбора языка программирования, до решения проблем передачи данных. Особенную сложность вызвало взаимодействие вида «Клиент-Сервер-Сервер», передача данных в котором была реализована с помощью протокола TCP. После множества неудачных попыток было принято решение отказаться от данной технологии в пользу протокола HTTP, который в последствии и дал нам возможность реализовать поставленную задачу.

Касаемо созданной программной системы, в качестве ее перспектив можно предложить выделение серверам отдельных машин, работу в режиме потоковой передачи видеоряда.  
Такого рода программные системы уже имеют существуют, как например дорожные камеры, фиксирующие номера автомобилей, нарушающих правила дорожного движения, или системы видеонаблюдения в Китае с внедренной системой распознавания лиц. Однако защита шифрованием передаваемых данных раскрывает потенциал данной системы в области безопасности и защиты информации, что, безусловно, важно в условиях жизни в XXI веке, когда любая информация может нести в себе большой риск для жизни человека, если попадет в руки злоумышленников.

# **Список используемой литературы**

1. Рогозин Н. О., Курс лекций по компьютерным сетям – М., 2020
2. Э. Таненбаум, Д. Уэзеролл. Компютерные сети. Пятое издание
3. Документация CherryPy [Электронный ресурс]. URL: https://docs.cherrypy.org/en/latest/ (дата обращения 21.12.2020)
4. Документация CORS [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/CORS/ (дата обращения 21.12.2020)
5. Документация Django [Электронный ресурс]. URL: https://docs.djangoproject.com/en/3.1/ (дата обращения 21.12.2020)
6. Документация EcmaScript [Электронный ресурс]. URL: http://www.ecma-international.org/ecma-262/6.0/ (дата обращения 21.12.2020)
7. Документация Flask [Электронный ресурс]. URL: https://flask.palletsprojects.com/en/1.1.x/ (дата обращения 21.12.2020)
8. Документация Python [Электронный ресурс]. URL: https://docs.python.org/3.6/ (дата обращения 21.12.2020)
9. Документация requests [Электронный ресурс]. URL: https://requests.readthedocs.io/en/master/ (дата обращения 21.12.2020)
10. Документация WebAssembly [Электронный ресурс]. URL: https://webassembly.org/ (дата обращения 21.12.2020)
11. PyCharm [Электронный ресурс]. URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения 21.12.2020)
12. WebStorm [Электронный ресурс]. URL: https://www.jetbrains.com/ru-ru/webstorm/ (дата обращения 21.12.2020)