Титульный лист

Задание 1

Задание 2

Задание 3

Задание 4

Задание Экономическое

Содержание

Введение

С каждым годом, количество цифровых данных неумолимо растёт. Фотографии, видео, документы пользователи предпочитают хранить в электронном формате в виде файлов. В 2020 году во всём мире было создано и скопировано 59 зетабайта или 59 триллионов гигабайтов данных, из которых большую часть, а именно 44 зетабайт, составляют неструктурированные данные (тексты, изображения, видео, аудио и пр.), то есть уже сейчас 75% всей цифровой информации – это неструктурированные данные.

Неструктурированные данные – это данные, которые либо не имеет заранее определенной структуры, либо не организованы в установленном порядке. Неструктурированные данные представлены, как правило, в виде текстовых документов, электронных писем и презентаций. Примерами не текстовых неструктурированных данных являются видео, изображения и аудиофайлы.

На сегодняшний день существует механизм хранения данных, который используется с момента появления компьютеров, − файловая система. Однако существует ряд ограничений, связанных с работой с неструктурированными данными. Во-первых, используемая для хранения данных система должна хорошо масштабироваться. Масштабировать файловую систему - сложная задача: требуется не только управлять ненужными метаданными и иерархией, которые навязывают файловые системы, но также необходимо учитывать такие аспекты обслуживания, как управление резервными копиями. Во-вторых, недостаточно просто собрать и хранить неструктурированные данные, необходимо также применить некоторый уровень организации, чтобы эффективно с ними работать. Такие методы, как анализ текста, автоматическая категоризация и автоматическая пометка, имеют решающее значение для получения бизнес-смысла для неструктурированных данных. Этого сложно добиться с файловыми системами, потому что они имеют фиксированные слои (или расположение). Для решения этих проблем появилась концепция объектного хранилища.

Важным аспектом любых систем хранения является обработка метаданных. Объектное хранилище обеспечивает большую гибкость, поскольку метаданные объекта изначально не определены. Метаданные не ограничиваются тем, что система хранения считает важным. Имеется возможность добавить любой тип или количество метаданных. Объектное хранилище позволяет назначить тип приложения, с которым связан объект; важность для приложения; уровень защиты данных, который требуется конкретному объекту.

Объектное хранилище обладает следующими свойствами:

- Масштабируемость. Объектное хранилище может содержать практически любое количество данных без необходимости в разбиении набора данных на разделы;
- Эффективность. Отсутствие иерархии означает отсутствие узких мест, возникающих вследствие использования сложных систем каталогов;
- Доступность. Объектные системы хранения имеют механизмы для сохранения целостности данных, обеспечивают репликацию данных, последовательные обновления и отсутствие простоев.

На IT рынке уже существует немало облачных объектных хранилищ от крупных IT компаний. Например, Google cloud storage, Amazon s3, IBM object storage, отечественные решения от Mail.ru и Yandex. Их объединяет одна важная деталь: размещение данных в облаке собственных серверах компании. Вследствие этого все они предоставляют услуги по использованию места на их серверах за определённую плату. Плата взимается за каждый запрос, а так же за само хранение данных. Еще одним важным недостатком является то, что клиент не знает, где и как хранятся его данные. Кроме того, компания, предоставляющая услугу, имеет доступ к конфиденциальной информации пользователей. Не смотря на обещания крупных IT компаний в безопасности на 99,9999999%, по данным InfoWatch только за прошлый год в сеть утекло более 14 миллиардов конфиденциальных записей. Поэтому многие потенциальные клиенты, боясь за очередную утечку, хотят найти решение, которое позволит им сохранить свои данные в целости и безопасности. Так же на некоторых предприятиях требования

информационной безопасности или внутреннего регламента обязывает сотрудников или само предприятие использовать отечественные разработки и решения, которые могу функционировать без доступа в сеть интернет.

В современном мире каждый человек где-то хранит персональные данные: в облаке, на флеш-накопителе или жестком диске компьютера. Сейчас, в эпоху развития компьютерных технологий, практически в любом виде деятельности необходимо передавать данные другим людям или машинам и хранить эти данные. Системы хранения и передачи информации должны быть удобны как для пользователя, так и для обслуживающего персонала или машины.

Возможны два варианта решения этой задачи:

- Самый безопасный: передать данные лично на USB-флеш-накопителе;
- Самый простой: разместить файлы в облачном хранилище, предлагаемом IT гигантами.

Однако существует альтернативное решение - это быстроразвёртываемое объектное хранилище, которое запускается на любом компьютере и обеспечивает удобство облачных систем хранения и безопасность физических накопителей. Для работы в хранилищем необходимо просто запустить исполняемый файл и указать путь до места, где необходимо развернуть объектное хранилище. Пользовательское взаимодействие с объектным хранилищем выполняется через специальный интерфейс. Интерфейс же в свою очередь взаимодействует с объектным хранилищем посредством API.

Таким образом, быстроразвёртывыаемое объектное хранилище является перспективным механизмом организации хранения неструктурированных данных. Предлагаемый механизм обеспечивает удобство работы с неструктурированными данными, а также безопасность и надежность их хранения.

1 Описание проблемы

Данные существуют во множестве различных форм и размеров, но большинство из них могут быть представлены в виде структурированных и неструктурированных данных (таблица 1.1).

Таблица 1.1 – Сравнение структурированных и неструктурированных данных

	Структурированные данные	Неструктурированные данные
Организация	Структурированные данные представляют собой высокоорганизованную, фактическую (важную) и точную информацию	Неструктурированные данные не имеют заранее определённой структуры и представлены во всём многообразии форм
Характеристика данных	Количественная	Качественная
Место размещения	Хранилища данных Реляционные базы данных	Пул данных Нереляционные базы данных
Формат и модель	Несколько предопределённых форматов и моделей данных	Формат и модель данных заранее не определены

Структурированные данные – это хорошо организованные и точно отформатированные данные. Эти данные существуют в формате реляционных баз данных, то есть, информация хранится в соответствии с моделью данных предоставляемой СУБД. Таким образом, структурированные данные аккуратно упорядочиваются и записываются, поэтому их можно легко найти и обработать. Пока данные вписываются в структуру базы данных, мы можем легко искать конкретную информацию и выделять отношения между ее частями. Такие данные

можно использовать только по прямому назначению. Кроме того, для структурированных данных обычно не требуется много места для хранения.

В аналитических целях можно использовать хранилища данных . Data Warehouse – это центральные хранилища данных, используемые компаниями для анализа данных и составления отчетов.

Неструктурированные данные – данные, которые не соответствуют заранее определённой модели данных и не имеют легко идентифицируемой структуры. Вследствие этого их нелегко использовать в компьютерных программах. Неструктурированные данные не организованны заранее определённым образом или не имеют заранее определённой модели данных, что делает их неудобными для размещения в реляционной базе данных. Сегодня крупнейшие и не только предприятия осознают ценность и важность данных на всех этапах деятельности и жизненного цикла. В различных областях – в маркетинге, в медицине и здравоохранении, в медиа-сервисах и т.д. – неструктурированные данные крайне ценны тем, что находятся новые способы их анализа, изменения и форматирования. К примеру:

- Медицинские учреждения применяют различные методы анализа архивных данных для прогнозирования хода болезни
- Автоконцерны исследуют особые данные с датчиков автомобилей, чтобы предугадать вероятные неисправности
- Телекомпании повторно используют передачи, фильмы, сериалы для показа. Чтобы их повторно не обрабатывать под требуемый формат их необходимо хранить в надёжном месте

Но ценность такие данные имеют только при условии экономного хранения, удобного доступа и безопасности от «утечек».

Уже существует механизм хранения, который люди используют с появления современных компьютеров − файловая система. Но когда речь идёт о неструктурированных данных, важно понимать, что система, используемая для хранения данных, должна очень хорошо масштабироваться. Однако масштабировать файловую систему это сложная задача. Вам требуется не только

управлять ненужными метаданными и иерархией, которые навязывают файловые системы, но также необходимо учитывать такие аспекты обслуживания, как управление резервными копиями.

Недостаточно просто собрать и хранить неструктурированные данные. Также необходимо применить некоторый уровень организации, чтобы разобраться в них. Такие методы, как анализ текста, автоматическая категоризация и автоматическая пометка, имеют решающее значение для получения бизнес-смысла для всех неструктурированных данных, которые вы собираете. Этого сложно добиться с файловыми системами, потому что они имеют фиксированные слои или расположение.

Для решения этой проблемы появились объектные хранилища.

Объектное хранилище представляет собой стратегию управления и манипуляции хранилищем данных как отдельными единицами, объектами. Эти объекты хранятся в одном хранилище и не интегрируются в файлы, находящиеся в других папках. Вместо этого хранилище объектов объединяет фрагменты данных, из которых состоит файл, добавляет в него все соответствующие метаданные и прикрепляет пользовательский идентификатор.

Объектное хранилище добавляет в файл полные метаданные, устраняя многоуровневую структуру файлового хранилища, и помещает все в плоское адресное пространство — пул хранилища. Именно метаданные являются ключом к успеху объектного хранилища: они обеспечивают глубокий анализ использования и функций данных в пуле.

Очень важно, чтобы данные были доступны через HTTP(S), чтобы обеспечить лёгкий доступ к файлу. Затем его можно подвергнуть анализу или другим методам. Объектное хранилище справляется с этим хорошо. Почти все объектные хранилища имеют REST API, которое позволяет крайне просто получить доступ к файлам через HTTP(S).

REST API полезно не только для получения доступа к объектам, оно позволяет авторизоваться и получать токен сессии, получать свойства файла,

управлять разрешениями и т.д., в общем всё, что необходимо будет делать вручную в файловой системе.

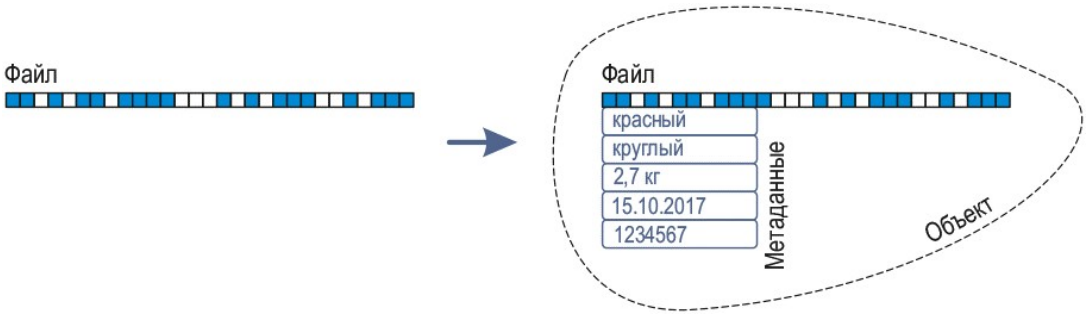

Рисунок 1.1 – Объект

Важным аспектом любых систем хранения является обработка метаданных. Объектное хранилище обеспечивает большую гибкость, поскольку метаданные объекта изначально не определены. Метаданные не ограничиваются тем, что система хранения считает важным. Имеется возможность добавить любой тип или количество метаданных. Объектное хранилище позволяет назначить тип приложения, с которым связан объект; важность для приложения; уровень защиты данных, который требуется конкретному объекту. И так далее, возможности безграничны.

2 Обзор аналогов

Несмотря на то, что реализаций объектных хранилищ существует огромное множество, почти все они являются облачным решением и хранят персональные данные пользователей на собственных серверах. Быстроразвёртываемое объектное хранилище обладает уникальной особенностью – запуск и работа на локальной машине или устройстве, что позволяет вам настроить его в локальной сети без доступа к интернету и ограничить возможный несанкционированный доступ извне. Так же важным преимуществом является цена, а точнее ценообразование. Если при использовании облачных решений вы платите за обслуживание, пользование и хранение данных, то в случае с локальным быстроразвёртываемым хранилищем вы платите исключительно за «железо» или выбранному Вами провайдеру по утверждённому тарифу. Для наглядности я приведу таблицу сравнения цены пользования облачным решением Amazon s3 и MinIO за год и стоимость системы с накопителями объёмом 8ТБ. Во время преддипломной практики было экспериментально определено количество GET и PUT, COPY, POST запросов к объектному хранилищу на проекте «ВДиалоге».

Таблица 1 – Сравнение с аналогами

	Amazon s3	MinIO	Быстроразвёртываемое объектное хранилище
Объём накопителей	8ТБ	8ТБ	8ТБ
Количество GET запросов	10000	Не ограничено	Не ограничено
Количество PUT, COPY, POST запросов	100	Не ограничено	Не ограничено
Обслуживание	Включено в	Ручное (1	Ручное (1 человек)

	стоимость	человек)	
Размещение	В облаке	В облаке или локально	В облаке и/или локально
Цена	2329.8 USD/год	960 USD/год за облачное решение и 500 USD за комплект оборудования	500 USD за комплект оборудования Или облачное размещение по цене тарифа провайдера, у которого будет размещено

Из таблицы видно, что Быстроразвёртываемое объектное хранилище выигрывает в цене в долгосрочной перспективе и возможностях конфигурации системы: всё зависит от вашего бюджета и требований. А за счёт простоты настройки и использования с обслуживанием справится обычный пользователь.

Для некоторых пользователей (клиентов) страна разработки является важным фактором при выборе программного продукта. Так как существуют ограничения (постановление Правительства Российской Федерации от 16 ноября 2015 г. №1236) на использование иностранного программного обеспечения в определённых структурах и предприятиях. Так же для обеспечения дополнительного уровня безопасности от возможного вмешательства или утечки требуется использовать программное обеспечение, которое не требует и может функционировать без доступа в интернет.

3 Программная документация

3.1 Техническое задание на программное обеспечение

3.1.1 Общие сведения

3.1.1.1 Наименование программы

Быстроразвёртываемое объектное хранилище

3.1.1.2 Наименование разработчика и заказчика программы

Разработчик: Ащеулов М. Р.
Заказчик: ООО «НПО «Криста»

3.1.1.3 Основание для проведения работ

Документ, на основании которого разрабатывается программное обеспечение: приказ № 550-04, утвержденный ректором РГАТУ им. П.А. Соловьева от 30.11.2020.

3.1.1.4 Область применения программы

Программа предназначена для персонального и корпоративного использования с целью организации, хранения, а так же получения и использования неструктурированных данных.

3.1.2 Назначение разработки

Быстроразвёртывыаемое объектное хранилище является перспективным механизмом организации хранения неструктурированных данных. Предлагаемый механизм обеспечивает удобство работы с неструктурированными данными, а также безопасность и надежность их хранения.

3.1.3 Требования к программному продукту или программному обеспечению

3.1.3.1 Требования к функциональным характеристикам

Программный продукт должен обеспечивать:

- Получение объекта из объектного хранилища;
- Загрузку одного объекта в объектное хранилище;
- Загрузку нескольких объектов последовательно;
- Обновление объекта в объектном хранилище;
- Удаление объекта из объектного хранилища;
- Аутентификацию существующего пользователя;
- Установление всех метаданных для заданного объекта;
- Добавление параметра к метаданным заданного объекта;
- Получение всех метаданных объекта;
- Получение одного параметра метаданных объекта по ключу;
- Регистрацию (добавление) нового пользователя.

3.1.3.2 Требования к надёжности

Надежность должна обеспечиваться за счет использования сертифицированных технических средств системного и базового ПО, комплектующих.

С целью повышения отказоустойчивости системы может быть использован источник бесперебойного питания с возможностью автономной работы системы и активного сетевого оборудования.

Для стабильной работы системы и обеспечения быстрого доступа к программному обеспечению необходимо устойчивое сетевое соединение либо в локальной сети, либо с глобальной сетью.

3.1.4 Требования к составу и параметрам технических средств

3.1.4.1 Сервер или персональный компьютер, который используется для запуска приложения

Требования к аппаратной части:

- Процессор с совокупным количеством ядер не менее 2
- Оперативная память: не менее 2 гб
- Подсистема хранения данных системы, ПО, конфигурационных файлов: не менее 100 гб
- Сетевой адаптер с максимальной пропускной скоростью не менее 10 Мбит/сек

Требования к каналам связи:

- Гарантированная пропускная способность не менее 10 Мбит/сек
- Доступ к сетевому адресу сервера из сети интернет для работы за пределами локальной сети

Требования к программному обеспечению:

- Операционная система: Ubuntu 20.04 и выше, Fedora 36 и выше, MacOS 11.0 и выше, Windows 10 1709 и выше.

3.1.4.2 Клиентская часть

Требования к аппаратной части:

- Процессор: 2 ядра, x86-64 с частотой 1.3 Ггц и выше
- Оперативная память: не менее 2 гб
- Свободное дисковое пространство: не менее 100 мб

Требования к каналам связи между клиентским устройством и сервером:

- Скорость передачи данных: не менее 512 Кбит/сек
- Требования к программному обеспечению:
- Операционная система: Ubuntu 20.04 и выше, Fedora 36 и выше, MacOS 11.0 и выше, Windows 10 1709 и выше.
- Браузер на основе движка Chromium (Google Chrome, Яндекс Браузер, Edge 89.0.774.50 и т.д.)

3.1.5 Специальные требования

Специальные требования к программному продукту не предъявляются.

3.1.6 Требования к программной документации

- Пояснительная записка к программному обеспечению
- Описание программы
- Программа и методика испытаний

3.2 Пояснительная записка к программному обеспечению

3.2.1 Анализ предметной области

Объектное хранилище, часто называемое объектно-ориентированным хранилищем, представляет собой архитектуру хранилища данных для обработки больших объемов неструктурированных данных. Это данные, которые не соответствуют или не могут быть легко организованы в традиционной реляционной базе данных со строками и столбцами. Сегодняшние данные Интернет-коммуникаций в большинстве своём неструктурированы. К ним относятся: электронная почта, видео, фотографии, веб-страницы, аудиофайлы, данные датчиков и другие типы мультимедийного и веб-контента (текстовые или нетекстовые). Этот контент непрерывно передается из социальных сетей, поисковых систем, мобильных и «умных» устройств.

По оценкам исследовательской компании IDC, к 2025 году неструктурированные данные, вероятно, будут составлять до 80% всех данных во всем мире.

Предприятиям сложно эффективно (и по доступной цене) хранить и управлять этим беспрецедентным объемом данных. Объектно-ориентированное хранилище стало предпочтительным методом архивирования и резервного копирования данных. Объектное хранилище предлагает уровень масштабируемости, недоступный для традиционных файловых или блочных хранилищ. С помощью объектно-ориентированного хранилища вы можете хранить и управлять томами данных порядка терабайт (ТБ), петабайт (ПБ) и даже больше.

Объекты - это дискретные единицы данных, которые хранятся в структурно плоской среде данных. Здесь нет папок, каталогов или сложных иерархий, как в файловой системе. Каждый объект представляет собой простой автономный репозиторий, который включает данные, метаданные (описательную

информацию, связанную с объектом) и уникальный идентификационный номер ID (вместо имени файла и пути к файлу). Эта информация позволяет приложению находить объект и обращаться к нему. Вы можете объединить устройства хранения объектов в более крупные пулы хранения и распределить эти пулы хранения по расположениям. Это обеспечивает неограниченное масштабирование, а также повышает отказоустойчивость данных и аварийное восстановление.

Объектное хранилище устраняет проблемы сложности и масштабируемости иерархической файловой системы с папками и каталогами. Объекты могут храниться локально, но чаще всего они находятся на облачных серверах и доступны из любой точки мира.

Доступ к объектам (данным) в системе хранения объектов осуществляется через интерфейсы прикладного программирования (API). Собственный API для хранилища объектов - это RESTful API на основе HTTP (также известный как веб-служба RESTful). Эти API запрашивают метаданные объекта, чтобы найти нужный объект (данные) через Интернет из любого места и на любом устройстве. API RESTful используют HTTP-команды, такие как «PUT» или «POST», для загрузки объекта, «GET» для извлечения объекта и «DELETE» для его удаления. (HTTP означает протокол передачи гипертекста и представляет собой набор правил для передачи текста, графических изображений, звука, видео и других мультимедийных файлов в Интернете).

Вы можете хранить любое количество статических файлов в инстансе хранилища объектов, который будет вызываться API. Появляются дополнительные стандарты RESTful API, которые выходят за рамки создания, извлечения, обновления и удаления объектов. Это позволяет приложениям управлять хранилищем объектов, его контейнерами, учетными записями, мультиарендностью, безопасностью, биллингом и т. Д.

Например, предположим, что вы хотите хранить все книги в очень большой библиотечной системе на одной платформе. Вам нужно будет хранить содержимое книг (данные), а также связанную с ними информацию, такую как

автор, дата публикации, издатель, тема, авторские права и другие детали (метаданные). Вы можете хранить все эти данные и метаданные в реляционной базе данных, организованной в папки в иерархии каталогов и подкаталогов.

Но с миллионами книг процесс поиска станет громоздким и трудоемким. Здесь хорошо работает объектная система хранения, поскольку данные статические или фиксированные. В этом примере содержание книги не изменится. Объекты (данные, метаданные и идентификатор) хранятся в виде «пакетов» в плоской структуре и легко обнаруживаются и извлекаются с помощью одного вызова API. Кроме того, поскольку количество книг продолжает расти, вы можете объединять устройства хранения в более крупные пулы хранения и распределять эти пулы хранения для неограниченного масштабирования.

Есть много причин рассмотреть решения на основе концепции объектного хранилища для хранения ваших данных, особенно в эпоху Интернета и цифровых коммуникаций, которые производят большие объемы мультимедийных данных с возрастающей скоростью.

Хранение / управление неструктурированными данными

В эпоху облачных вычислений объектные хранилища получают широкое распространение и используются для управления неструктурированными данными, которые, по оценкам аналитиков, в ближайшем будущем будут представлять подавляющее большинство всех данных во всем мире. Объем веб-контента - электронной почты, видео, социальных сетей, документов, данных датчиков, производимых устройствами Интернета вещей (IoT), и т. д. - огромен и продолжает расти. Неструктурированные данные обычно статичны (неизменны), но могут потребоваться в любое время и в любом месте (например, изображения и видеофайлы или резервные копии архивных данных).

Масштабируемость

Неограниченное масштабирование - это, пожалуй, самое значительное преимущество объектно-ориентированного хранения данных. Объекты или дискретные единицы данных (в любом количестве) хранятся в структурно плоской среде данных на устройстве хранения, таком как сервер. Вы просто

добавляете дополнительные устройства / серверы параллельно центральному узлу объектного хранилища для дополнительной обработки и поддержки более высокой пропускной способности, необходимой для больших файлов, таких как видео или изображения.

Простота

Хранилище объектов устраняет сложность, присущую иерархической файловой системе с папками и каталогами. Существует меньше факторов влияющих на производительность, улучшается эффективность при извлечении данных, поскольку нет папок, каталогов или сложных иерархий для навигации. Это повышает производительность, особенно при управлении очень большими объемами данных.

Аварийное восстановление / доступность

Вы можете настроить системы хранения объектов так, чтобы они реплицировали контент. Если диск в кластере выходит из строя, доступен дублирующий диск, гарантирующий, что система продолжит работу без прерывания или снижения производительности. Данные могут реплицироваться внутри узлов и кластеров, а также между распределенными центрами обработки данных для дополнительного резервного копирования за пределами площадки и даже в географических регионах.

Объектное хранилище - более эффективная альтернатива решениям для резервного копирования на магнитные ленты, для которых требуются ленты, которые необходимо физически загружать в ленточные накопители, снимать с них и перемещать за пределы предприятия для географической избыточности. Объектное хранилище может быть использовано для автоматического резервного копирования локальных баз данных в облако и / или для рентабельной репликации данных между распределенными центрами обработки данных.

Настраиваемые метаданные

Помните, что каждый объект - это автономный репозиторий, который включает связанные с ним метаданные или описательную информацию. Объекты используют эти метаданные для важных функций, таких как политики хранения,

удаления и маршрутизации, стратегии аварийного восстановления (защита данных) или проверка подлинности контента. Вы также можете настроить метаданные с дополнительным контекстом, который впоследствии может быть извлечен и использован для получения бизнес-аналитики и анализа, например, в отношении обслуживания клиентов или рыночных тенденций.

3.2.2 Формулировка проблемы

Формулировка проблемы приведена в разделе 1.

3.2.3 Обзор аналогов

Обзор аналогов приведён в разделе 2.

3.2.4 Назначение и область применения

Назначение разработки быстроразвёртываемого объектного хранилища приведено в подразделе 3.1.2.

Область применения быстроразвёртываемого объектного хранилища приведена в подразделе 3.1.1.4.

3.2.5 Технические характеристики

3.2.5.1 Описание алгоритма и функционирования программы

Быстроразвёртываемое объектное хранилище позволяет пользователю запускать новое объектное хранилище или уже инициализированное,

подключаться к центральному узлу, организовывая сеть быстроразвёртываемого объектного хранилища.

Общий алгоритм запуска объектного хранилища представлен на рисунке А.1. Он описывает поведение программы после запуска исполняемого файла через консоль или двойным кликом.

Существует 3 возможных варианта запуска программного продукта быстроразвёртываемое объектное хранилище:

- Подключение к центральному узлу
- Запуск уже существующего объектного хранилища
- Инициализация нового объектного хранилища

Подключение к центральному узлу – подключение к уже запущенной сети быстроразвёртываемого объектного хранилища. Позволяет расширить сеть, предоставив запущенный узел с его ресурсами. Алгоритм подключения к центральному узлу сети быстроразвёртываемого объектного хранилища представлен на рисунке А.2.

Запуск уже существующего объектного хранилища – выбор директории ранее проинициализированного объектного хранилища. При запуске пользователь должен подтвердить права доступа с помощью ввода данных учётной записи пользователя-администратора и пройти аутентификацию. Алгоритм запуска уже существующего объектного хранилища представлен на рисунке А.3.

Инициализация нового объектного хранилища – создание или выбор новой директории для инициализации быстроразвёртываемого объектного хранилища, создание учётной записи пользователя-администратора и индексация директории. Алгоритм инициализации нового объектного хранилища представлен на рисунке А.4.

После успешного запуска узла быстроразвёртываемого объектного хранилища пользователь (клиент) может осуществлять действия в соответствии с UML диаграммной на рисунке А.5.

3.2.5.2 Описание и обоснование выбора технических средств

Разработка быстроразвёртываемого объектного хранилища осуществлялась с использованием языка программирования Java, фреймворка Quarkus, Git, Maven.

Java является одним из самых распространенных и популярных языков программирования. Java — это язык программирования высокого уровня, первоначально разработанный Sun Microsystems и выпущенный в 1995 году. Java работает на различных платформах, таких как Windows, Mac OS и различные версии *NIX.

Язык программирования был выбран в следствии наличия соответствующей квалификации у разработчиков. Для управления зависимостями и конфигурациями использовался Apache Maven.

Apache Maven — это мощный инструмент управления проектами, основанный на объектной модели проекта. Он используется для сборки проектов, зависимостей и документации. Данный фреймворк использует большинство крупных корпоративных решений, написанных на Java.

В качестве Web-фреймворка был выбран Quarkus. Продукт, разрабатываемый разработчиками RedHat и IBM. Проект имеет открытый исходный код и имеет достаточно дружное сообщество, которое работает над постоянным улучшением фреймворка и исправлением ошибок.

В процессе разработки была использована система управления версиями Git. Git – бесплатная распределенная система управления версиями с открытым исходным кодом, предназначенная для быстрой и эффективной обработки любых проектов, от небольших до очень крупных.

Для тестирования использовался Postman. Postman – программный продукт, предоставляющий инструменты для тестирования API (Application Programming Interface). Несмотря на простоту использования, он обладает весьма внушительным функционалом и используется крупнейшими IT компаниями. Postman позволяет не только совершать одиночные HTTP(S) запросы, но и

создавать целый сценарии тестирования, автоматизируя процесс поиска неисправностей и ошибок.

3.3 Описание программы

3.3.1 Описание архитектуры

Архитектура быстроразвёртываемого объектного хранилища (Рисунок 3.1) состоит из двух основных уровней: уровень шлюза и уровень объектного хранилища.

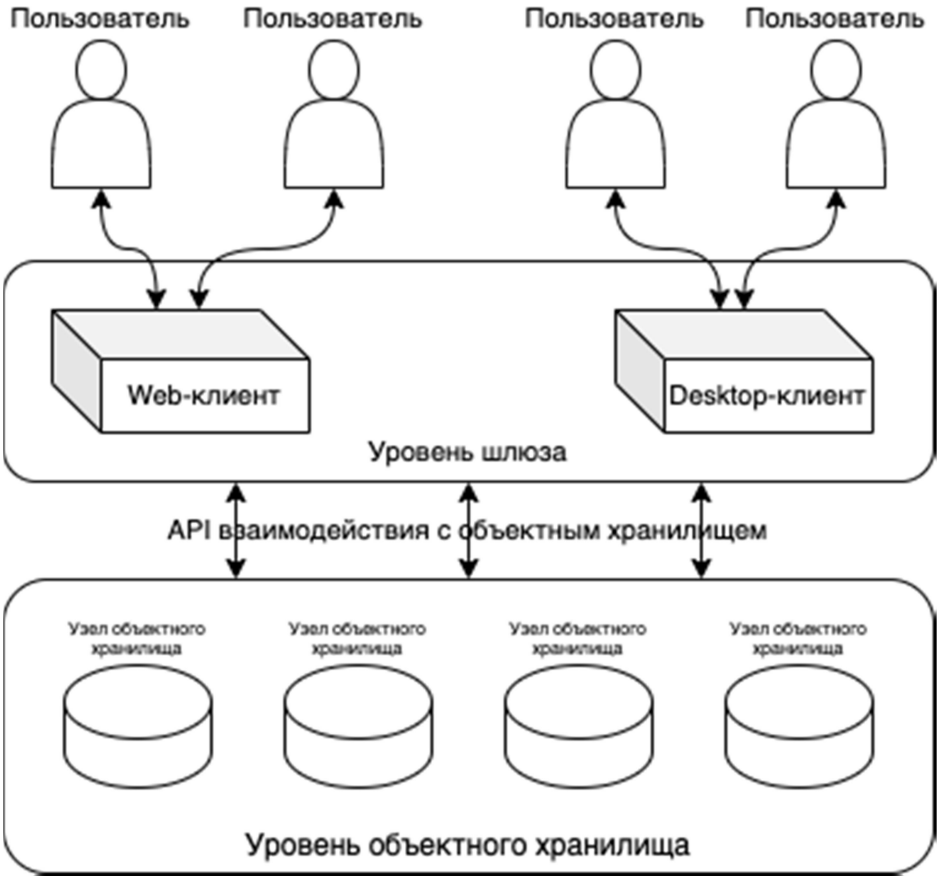

Рисунок 3.1 – Архитектура быстроразвёртываемого объектного хранилища

Пользователь взаимодействует с объектным хранилищем посредством прямых запросов к API или с помощью графического интерфейса, предоставляемым web-клиентом или desktop-клиентом, которые располагаются на уровне шлюза. Сам уровень шлюза включает в себя различные инструменты для взаимодействия с API быстроразвёртываемого объектного хранилища.

Примерами таких инструментов являются curl, postman, различные поддерживаемые браузеры, разработанные клиентские приложения для компьютера. Так как всё взаимодействие осуществляется через HTTP(S), клиент может использовать любой удобный ему инструмент для взаимодействия с быстроразвёртываемым объектным хранилищем.

Уровень объектного хранилища представляет собой своего рода сеть, состоящую из одного и более «узлов объектного хранилища». Так же на уровне объектного хранилища осуществляется маршрутизация запросов.

Узел объектного хранилища представляет запущенную и инициализированную копию программного обеспечения, после завершения настройки. Если запущенный узел является первым в процессе построения сети быстроразвёртываемого объектного хранилища, то он автоматически становится центральным узлом этой сети. При инициализации центрального узла необходимо указать данные для учётной записи администратора новой сети.

Центральный узел занимается маршрутизацией запросов и содержит информацию обо всех других подключённых к сети узлах. Этот узел содержит таблицы, в которых описаны все проиндексированные объекты и адрес, по которому они располагаются. При поступившем запросе центральный узел сначала верифицирует его, затем, при успешной проверке, перенаправляет запрос на узел в соответствии с правилами маршрутизации.

Если новый запущенный узел планируется подключить к уже существующей сети быстроразвёртываемого объектного хранилища, то необходимо указать адрес центрального узла и пройти аутентификацию с использованием учётной записи, которой разрешено добавлять новые узлы или учётной записи администратора этой сети. После успешной аутентификации новый узел будет включён в существующую сеть и передаст таблицу индексации объектов центральному узлу.

Уровень шлюза и уровень объектного хранилища взаимодействуют друг с другом с помощью уникального программного интерфейса, который реализует

основные функции для взаимодействия с быстроразвёртываемым объектным хранилищем.

3.3.2 Описание функций и модулей

Программный продукт обеспечивает следующие функции:

a) Получение объекта из объектного хранилища;
b) Загрузку одного объекта в объектное хранилище;
c) Загрузку нескольких объектов последовательно;
d) Обновление объекта в объектном хранилище;
e) Удаление объекта из объектного хранилища;
f) Аутентификацию существующего пользователя;
g) Установление всех метаданных для заданного объекта;
h) Добавление параметра к метаданным заданного объекта;
i) Получение всех метаданных объекта;
j) Получение одного параметра метаданных объекта по ключу;
k) Регистрацию (добавление) нового пользователя.

Объекты в быстроразвёртываемом объектном хранилище делятся на две категории по уровню доступа: публичные и приватные. Обращение к объектам с публичным уровнем доступа не требует учётной записи и разрешения. Обращение к объектам с приватным уровнем доступа требует наличие токена доступа, полученного в процессе аутентификации. Получить доступ к приватному объекту может только пользователь, которому разрешён доступ к этому объекту. Доступ предоставляет либо администратор сети быстроразвёртываемого объектного хранилища, либо пользователь с соответствующими правами.

a) Получение объекта из объектного хранилища.

Публичный объект: получить объект с публичным уровнем доступа может любой пользователь как прошедший аутентификацию, так и не прошедший.

Приватный объект: получить объект с приватным уровнем доступа может только пользователь, который прошел аутентификацию и получил токен доступа.

Описание запроса:

GET запрос осуществляется по адресу: host.address/api/get/{uid}, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта. Если пользователь, осуществляющий запрос прошёл аутентификацию, то при запросе в заголовке «X-TOKEN-X» передаётся токен доступа. При успешном запросе сервер вернёт статус 200, запрашиваемый объект и JSON, в котором содержится поле «response» со значением «объект передан». При не успешном запросе сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «доступ запрещён» или «запрашиваемого объекта не существует». В случае если пользователь не прошёл аутентификацию и запрашиваемого объекта не существует, то будет возвращено значение «доступ запрещён», в целях обеспечения сокрытия объектов с приватным уровнем доступа.

b) Загрузка одного или нескольких объектов в объектное хранилище

Загрузка одного или нескольких объектов в объектное хранилище может осуществляться только пользователем, прошедшим аутентификацию.

Описание запроса:

POST запрос осуществляется по адресу: host.address/api/post, где host.address – адрес центрального узла. При запросе в заголовке «X-TOKEN-X» передаётся токен доступа. Так же прикладывается в теле запроса файл или файлы с ключом «file» и с заголовками запроса

Content-Type: multipart/form-data

Content-Length: <размер файла в байтах>

При успешной загрузке сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «Объект загружен» и поле «uid» со значением «<uid объекта>». При не успешной загрузке сервер вернёт статус 200 и JSON , в котором содержится поле «response» со значением «Ошибка при загрузке объекта».

Если пользователь, осуществляющий загрузку, не прошёл аутентификацию и не предоставил токен при запросе или не имеет соответствующих прав, то сервер возвращает статус 403 и JSON, в котором содержится поле «response» со значением «Загрузка запрещена»

c) Обновление объекта в объектном хранилище

Обновление объекта в объектном хранилище может осуществляться только пользователем, прошедшим аутентификацию и имеющим соответствующие права. При обновлении объекта сохраняются все его метаданные, привязанные к идентификатору.

Описание запроса:

PUT запрос осуществляется по адресу: host.address/api/put/{uid}, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта. При запросе в заголовке «X-TOKEN-X» передаётся токен доступа. Так же прикладывается в теле запроса файл или файлы с ключом «file» и с заголовками запроса

Content-Type: multipart/form-data

Content-Length: <размер файла в байтах>

При успешном обновлении объекта сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «Объект обновлён». При не успешном обновлении объекта сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «Ошибка при обновлении» или «Объекта с таким идентификатором не существует».

Если пользователь, осуществляющий обновление объекта, не прошёл аутентификацию и не предоставил токен при запросе или не имеет соответствующих прав, то сервер возвращает статус 403 и JSON, в котором содержится поле «response» со значением «Обновление запрещено»

d) Удаление объекта из объектного хранилища

Удаление объекта в объектном хранилище может осуществляться только пользователем, прошедшим аутентификацию и имеющим соответствующие

права. При удалении объекта удаляются все его метаданные, привязанные к индентификатору.

Описание запроса:

DELETE запрос осуществляется по адресу: host.address/api/delete/{uid}, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта. При запросе в заголовке «X-TOKEN-X» передаётся токен доступа.

При успешном удалении объекта сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «Объект удалён». При не успешном удалении сервер вернёт статус 200 и JSON, в котором содержится поле «response» со значением «Ошибка удаления», «Удаляемого объекта не существует».

Если пользователь, осуществляющий удаление объекта, не прошёл аутентификацию и не предоставил токен при запросе или не имеет соответствующих прав, то сервер возвращает статус 403 и JSON, в котором содержится поле «response» со значением «Удаление запрещено»

е) Аутентификация существующего пользователя

Аутентификация пользователя проходит с использованием логина и пароля. Данные преобразуются в одну строку по формуле login+password, где login – логин; password – пароль. Затем преобразованная строка кодируются с помощью алгоритма BASE64 и отправляется на сервер вместе с заголовком «X-BASE64AUTH-X» по адресу host.address/api/auth. После получения сервером данных проверяется наличие такого пользователя в системе. При успешном выполнении предыдущей операции сервер генерирует токен доступа по формуле

$$SHA256(BASE64(login + password) + System.currentTimeMillis + UUID.randomUUID.toString)$$

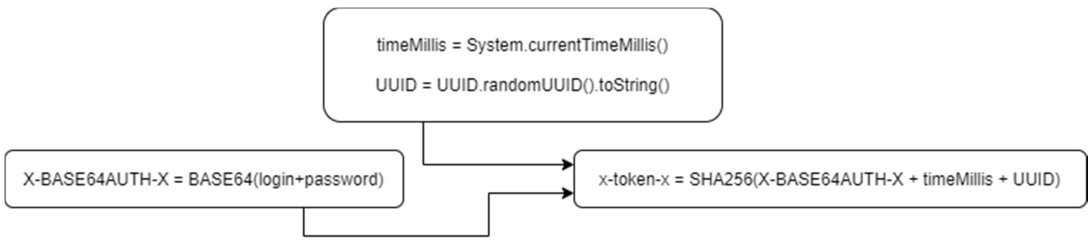

Рисунок 3.2 – Создание токена доступа

После успешной генерации токена доступа он заносится в таблицу сессий и возвращается пользователю в виде ответа от сервера со статусом 200 и JSON, в котором содержится поле «x-token-x» со значением сгенерированного токена и поле «response» со значением «Аутентификация пройдена».

При любой ошибке в процессе генерации токена доступа сервер возвращает ответ со статусом 200 и JSON, в котором содержится поле «response» со значением «Ошибка в процессе аутентификации».

f) Установление всех метаданных для заданного объекта

Установление всех метаданных для заданного объекта может осуществляться только пользователем, прошедшим аутентификацию и имеющим соответствующие права. Выполнение этого запроса уничтожает уже существующие и привязанные метаданные к объекту с уникальным идентификатором.

Описание запроса:

POST запрос осуществляется по адресу host.address/api/obj/{uid}/set-meta, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта. Метаданные передаются в теле запроса в формате JSON.

При успешном установлении всех метаданных сервер вернёт JSON, в котором содержится поле «response» со значением «Метаданные успешно установлены» и статус 200. При не успешном установлении всех метаданных сервер вернёт JSON, в котором содержится поле «response» со значением «Ошибка установления метаданных» или «Операция запрещена» и статус 200.

g) Добавление параметра к метаданным заданного объекта

Добавление параметра к метаданным заданного объекта может осуществляться только пользователем, имеющим соответствующие права и прошедшим аутентификацию. Выполнение этого запроса добавляет один параметр к метаданным.

Описание запроса:

POST запрос осуществляется по адресу host.address/api/obj/{uid}/add-meta, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта. Новый добавляемый параметр передаётся в теле запроса в формате JSON.

При успешном установлении всех метаданных сервер вернёт JSON, в котором содержится поле «response» со значением «Метаданные успешно добавлены» и статус 200. При не успешном установлении всех метаданных сервер вернёт JSON, в котором содержится поле «response» со значением «Ошибка установления метаданных» или «Операция запрещена» и статус 200.

h) Получение всех метаданных объекта

Публичный объект: получить все метаданные объекта с публичным уровнем доступа по уникальному идентификатору может любой пользователь как прошедший аутентификацию, так и не прошедший.

Приватный объект: получить все метаданные объекта с приватным уровнем доступа могут только пользователи, которые успешно прошли аутентификацию и предоставили токен доступа.

Описание запроса:

GET запрос осуществляется по адресу host.address/api/obj/{uid}/get-all-meta, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта.

При успешном запросе сервер возвращает все метаданные в теле ответа в формате JSON и статус 200. При не успешном запросе сервер возвращает статус 200 и JSON, содержащий поле «response» со значением «Метаданные объекта не определены», «Доступ запрещён» или «Объекта с заданным идентификатором не существует». Если пользователь не прошёл аутентификацию и пытается обратиться к приватному объекту, к несуществующему объекту или объекту, метаданные которого не определены, всегда возвращается «response» со значением «Доступ запрещён».

i) Получение одного параметра метаданных объекта по ключу

Публичный объект: получить метаданные по заданному ключу у объекта с публичным уровнем доступа по уникальному идентификатору может любой пользователь как прошедший аутентификацию, так и не прошедший.

Приватный объект: получить метаданные по заданному ключу у объекта с приватным уровнем доступа могут только пользователи, которые успешно прошли аутентификацию и предоставили токен доступа.

Описание запроса:

GET запрос осуществляется по адресу host.address/api/obj/{uid}/get-meta/{key}, где host.address – адрес центрального узла; {uid} – уникальный идентификатор объекта; {key} – ключ к запрашиваемым метаданным.

При успешном запросе сервер возвращает метаданные по заданному ключу в теле ответа в формате JSON и статус 200. При не успешном запросе сервер возвращает статус 200 и JSON, содержащий поле «response» со значением «Метаданные объекта не определены», «Доступ запрещён» или «Объекта с заданным идентификатором не существует». Если пользователь не прошёл аутентификацию и пытается обратиться к приватному объекту, к несуществующему объекту или объекту, метаданные которого не определены, всегда возвращается «response» со значением «Доступ запрещён».

j) Регистрация (добавление) нового пользователя

Регистрация (добавление) нового пользователя может осуществляться: анонимным пользователем; пользователем, прошедшим аутентификацию и имеющим соответствующие права; администратором сети быстроразвёртываемого объектного хранилища.

Описание запроса:

POST запрос осуществляется по адресу host.address/api/new-user. Для регистрации необходим новый логин и пароль. Данные преобразуются в одну строку по формуле login+password, где login – логин; password – пароль. Затем преобразованная строка кодируются с помощью алгоритма BASE64 и отправляется на сервер вместе с заголовком «X-BASE64NEWUSER-X». На сервере проверяется уникальность полученных данных. Если данные корректны,

то пользователь заносится в таблицу пользователей и сервер возвращает ответ в виде статуса 200 и JSON, который содержит поле «response» со значением «Успешное добавление нового пользователя». Если данные некорректны, то сервер возвращает ответ со статусом 200 и JSON, в котором содержится поле «response» со значением «Ошибка при добавлении нового пользователя».

3.4 Программа и методика испытаний

Для тестирования API будет использовано специализированное программное обеспечение Postman.

При тестировании проверяется соответствие программного продукта техническим требованиям. Испытания должны проводиться с соблюдением указанных в техническом задании требованиях к аппаратуре.

Программа должна корректно обрабатывать и отвечать на следующие запросы:

1. GET /api/get/{uid}

Получение объекта из объектного хранилища по уникальному идентификатору.

uid - уникальный идентификатор объекта.

Ожидаемый результат: Объект, соответствующий уникальному идентификатору.

2. POST /api/post

Загрузка одного или нескольких объектов в объектное хранилище.

В теле запроса прикладывается загружаемый(-е) файлы. Так как загрузку может осуществлять только пользователь, прошедший аутентификацию и имеющий соответствующие права, то так же прикладывается токен доступа в заголовке запроса.

Ожидаемый результат: Объект загружен и получен ответ об успешном завершении операции.

3. PUT /api/put/{uid}

Обновление объекта в объектном хранилище.

uid - уникальный идентификатор объекта.

В теле запроса прикладывается объект, который будет помещён на место объекта с идентификатором uid. Так как обновление объекта может осуществлять только пользователь, прошедший аутентификацию и имеющий соответствующие права, то так же прикладывается токен доступа в заголовке запроса.

Ожидаемый результат: Старый объект с соответствующим идентификатором удалён, новый загружен.

4. DELETE /api/delete/{uid}

Удаление объекта из объектного хранилища.

uid - уникальный идентификатор объекта.

Так как удаление объекта может осуществлять только пользователь, прошедший аутентификацию и имеющий соответствующие права, то так же прикладывается токен доступа в заголовке запроса.

Ожидаемый результат: Требуемый объект удалён.

5. GET /api/auth

Аутентификация существующего пользователя.

Передаются логин и пароль закодированные с помощью BASE64 с помощью заголовка запроса X-BASE64AUTH-X.

Ожидаемый результат: Токен доступа

6. POST /api/obj/{uid}/set-meta

Установление всех метаданных для заданного объекта.

uid - уникальный идентификатор объекта.

В теле запроса прикладывается JSON объект, который содержит все метаданные, которые необходимо установить объекту. Так как установка всех метаданных для заданного объекта может осуществляться только пользователем, прошедшим аутентификацию и имеющим соответствующие права, то так же прикладывается токен доступа в заголовке запроса.

Ожидаемый результат: Для объекта установлены метаданные.

7. POST /api/obj/{uid}/add-meta

Добавление параметра к метаданным заданного объекта.

В теле запроса прикладывается JSON объект, который содержит метаданные в формате ключ:значение, которые необходимо установить объекту. Добавить можно только один параметр. Так как добавление параметра к метаданным заданного объекта может осуществляться только пользователем,

имеющим соответствующие права и прошедшим аутентификацию, то так же прикладывается токен доступа в заголовке запроса.

Ожидаемый результат: Добавлен параметр в конец списка для соответствующего объекта.

8. GET /api/obj/{uid}/get-all-meta

Получение всех метаданных объекта.

uid - уникальный идентификатор объекта.

Метаданные заданного объекта должны возвращаться в теле запроса в формате JSON.

Ожидаемый результат: Все метаданные.

9. GET /api/obj/{uid}/get-meta/{key}

Получение одного параметра метаданных объекта по ключу.

uid - уникальный идентификатор объекта.

key – ключ.

Значение параметра метаданных заданного объекта возвращается в теле запроса в формате JSON.

Ожидаемый результат: Значение соответствующее определённому ключу.

10. POST /api/new-user

Регистрация (добавление) нового пользователя.

При совершении запроса на сервер передаются данные (логин и пароль) нового пользователя, которого необходимо зарегистрировать (добавить).

Ожидаемый результат: Пользователь зарегистрирован и получен ответ об успешном завершении операции.

В таблице 3.1 приведен скомпилированный набор тестов программного продукта.

Таблица 3.1 – Набор тестов программного продукта

Номер запроса	Описание запроса	Ожидаемый результат
1	Получение объекта по	Объект

	уникальному идентификатору	соответствующий уникальному идентификатору
2	Загрузка в объектное хранилище нового объекта	Объект загружен и получен ответ об успешном завершении операции
3	Обновление объекта в объектном хранилище	Старый объект с соответствующим идентификатором удалён, новый загружен
4	Удаление объекта в объектном хранилище	Требуемый объект удалён
5	Аутентификация и получение токена доступа	Токен доступа
6	Установление метаданных для объекта	Для объекта установлены метаданные
7	Добавление параметра к метаданным	Добавлен параметр в конец списка для соответствующего объекта
8	Получение всех метаданных	Все метаданные
9	Получение значения метаданных по ключу	Значение соответствующее определённому ключу

10	Регистрация нового пользователя	Пользователь зарегистрирован и получен ответ об успешном завершении операции

4 Эксплуатационная документация на программный продукт

4.1 Общие сведения о программе

Быстроразвёртываемое объектное хранилище является программным продуктом, который позволяет пользователю (клиенту) запустить локальное объектное хранилище на собственном компьютере или сервере. Взаимодействие с быстроразвёртываемым объектным хранилищем осуществляется посредством API.

Для взаимодействия с разрабатываемым программным обеспечением может быть разработан и использован графический интерфейс, который обращается к серверу и предоставляет пользователю возможность работать с программным продуктом через браузер или десктоп-клиент.

4.2 Руководство пользователя

Для начала работы с Быстроразвёртываемым объектным хранилищем необходимо запустить исполняемый файл через консоль или двойным кликом. Далее появляется диалоговое окно, которое предлагает запустить уже существующее объектное хранилище, инициализировать новое или подключиться к центральному узлу.

При инициализации нового система просит ввести данные учётной записи для администратора объектного хранилища. Введённые данные проверяются на корректность и соответствие установленным требованиям безопасности (длина пароля, наличие специальных символов, разные регистры символов, наличие цифр и т.д.). После успешной регистрации пользователя-администратора открывается диалог выбора корневой директории объектного хранилища или место, где будет оно инициализировано. При выборе файла вместо папки корневая директория будет местом расположения файла и файл автоматически

проиндексируется и добавится как объект. При выборе папки с файлами или другими папками, объектное хранилище инициализируется и индексирует всё как отдельные объекты. Т.е. и папки, и файлы станут отдельными объектами с идентификатором соответствующим имени. После успешного завершения этапа инициализации сервер запускается, и пользователь может начать работу с программным обеспечением.

При выборе пункта «Запустить уже существующее объектное хранилище» открывается диалог выбора расположения. Если пользователь выбирает директорию, в которой уже инициализировано объектное хранилище, то программа запрашивает у пользователя логин и пароль для учётной записи администратора этого объектного хранилища. После успешной верификации пользователя запускается сервер объектного хранилища, и пользователь может начать работу с программным обеспечением.

При выборе пункта «Подключиться к центральному узлу» открывается диалоговое окно, в которое требуется ввести адрес центрального узла. После ввода корректного адреса необходимо ввести данные учётной записи пользователя с правами на добавление новых узлов в сеть или данные учётной записи пользователя-администратора. После успешного подключения узла в сеть открывается диалог выбора расположения для инициализации объектного хранилища. Если пользователь выбирает директорию, в которой уже инициализировано быстроразвёртываемое объектное хранилище, то программа запрашивает у пользователя логин и пароль для учётной записи администратора этого объектного хранилища и затем запускается сервер.

Важным аспектом взаимодействия с программой является то, как организована работа с метаданными. Изначально метаданные объекта при инициализации не определены. По умолчанию доступ к получению, модификации, удалению и прочим операциям над объектом разрешены только пользователю-администратору после получения токена сессии (X-TOKEN-X). Чтобы разрешить публичный или общий доступ к объекту необходимо добавить параметр “access”:”all” к метаданным объекта. Ключ “access” обозначает то, кто

может иметь доступ к объекту. Если необходимо разрешить доступ конкретным пользователям, то необходимо в значении ключа ввести через пробел логины требуемых пользователей. Если необходимо разрешить доступ всем, то нужно указать значение “all”. Чтобы получить доступ к объекту с включённым значением “access”:”all” не требуется никаких дополнительных действий. Достаточно просто обратиться к объектному хранилищу через команду предоставив уникальный идентификатор объекта. Если объект имеет не публичный уровень доступа, то необходимо так же передать в заголовке запроса токен сессии, который может быть получен с помощью обращения к серверу объектного хранилища через /api/auth.

4.3 Системные требования

4.3.1 Сервер или персональный компьютер, который используется для запуска приложения

Требования к аппаратной части:

- Процессор с совокупным количеством ядер не менее 2
- Оперативная память: не менее 2 гб
- Подсистема хранения данных системы, ПО, конфигурационных файлов: не менее 100 гб
- Сетевой адаптер с максимальной пропускной скоростью не менее 10 Мбит/сек

Требования к каналам связи:

- Гарантированная пропускная способность не менее 10 Мбит/сек
- Доступ к сетевому адресу сервера из сети интернет для работы за пределами локальной сети

Требования к программному обеспечению:

- Операционная система: Ubuntu 20.04 и выше, Fedora 36 и выше, MacOS 11.0 и выше, Windows 10 1709 и выше.

4.3.2 Клиентская часть

Требования к аппаратной части:

- Процессор: 2 ядра, x86-64 с частотой 1.3 Ггц и выше
- Оперативная память: не менее 2 гб
- Свободное дисковое пространство: не менее 100 мб

Требования к каналам связи между клиентским устройством и сервером:

- Скорость передачи данных: не менее 512 Кбит/сек
- Требования к программному обеспечению:
- Операционная система: Ubuntu 20.04 и выше, Fedora 36 и выше, MacOS 11.0 и выше, Windows 10 1709 и выше.
- Браузер на основе движка Chromium (Google Chrome, Яндекс Браузер, Edge 89.0.774.50 и т.д.)

4.4 Установка программы

Для того чтобы установить быстроразвёртываемое объектное хранилище, необходимо загрузить и запустить исполняемый файл.

5 Акт испытаний программного продукта

Объектом испытаний является разработанное быстроразвёртываемое объектное хранилище. Испытания программного продукта проводили:

- Разработчик Ащеулов М. Р.
- Руководитель ВКР Задорина Н. А.

При тестировании проверялось соответствие программного продукта техническим требованиям.

Испытания проводились с соблюдением указанных в техническом задании требованиях к аппаратуре. Результаты испытаний зафиксированы в таблице 5.1. Испытания проводились по методике испытаний раздел 3-d.

Тестирование было проведено с помощью специализированного программного обеспечения для тестирования API – Postman.

Таблица 5.1 - Набор тестов и результат тестирования программного продукта

Номер запроса	Описание запроса	Ожидаемый результат	Полученный результат
1	Получение объекта по уникальному идентификатору	Объект соответствующий уникальному идентификатору	Объект соответствующий уникальному идентификатору
2	Загрузка в объектное хранилище нового объекта	Объект загружен и получен ответ об успешном завершении операции	Объект загружен и получен ответ об успешном завершении операции
3	Обновление объекта в объектном	Старый объект с соответствующим идентификатором	Старый объект с соответствующим идентификатором

	хранилище	удалён, новый загружен	удалён, новый загружен
4	Удаление объекта в объектном хранилище	Требуемый объект удалён	Требуемый объект удалён
5	Аутентификация и получение токена доступа	Токен доступа	Токен доступа
6	Установление метаданных для объекта	Для объекта установлены метаданные	Для объекта установлены метаданные
7	Добавление параметра к метаданным	Добавлен параметр в конец списка для соответствующего объекта	Добавлен параметр в конец списка для соответствующего объекта
8	Получение всех метаданных	Все метаданные	Все метаданные
9	Получение значения метаданных по ключу	Значение соответствующее определённому ключу	Значение соответствующее определённому ключу
10	Регистрация нового пользователя	Пользователь зарегистрирован и получен ответ об успешном завершении операции	Пользователь зарегистрирован и получен ответ об успешном завершении операции

По результатам тестирования по всем пунктам методики испытаний можно сделать вывод, что разработанный продукт работает корректно и удовлетворяет требованиям, описанным в техническом задании. Работу можно считать завершенной с положительным результатом.

Разработчик: _____ Ащеулов М. Р.

Руководитель ВКР: _____ Задорина Н. А.

6 Экономическое обоснование

Быстроразвёртываемое объектное хранилище является программным продуктом, который позволяет пользователю (клиенту) запустить локальное объектное хранилище на собственном компьютере или сервере. Взаимодействие с быстроразвёртываемым объектным хранилищем осуществляется посредством API.

Программный продукт обеспечивает:

- Получение объекта из объектного хранилища;
- Загрузку одного объекта в объектное хранилище;
- Загрузку нескольких объектов последовательно;
- Обновление объекта в объектном хранилище;
- Удаление объекта из объектного хранилища;
- Аутентификацию существующего пользователя;
- Установление всех метаданных для заданного объекта;
- Добавление параметра к метаданным заданного объекта;
- Получение всех метаданных объекта;
- Получение одного параметра метаданных объекта по ключу;
- Регистрацию (добавление) нового пользователя.

Быстроразвёртывыаемое объектное хранилище является перспективным механизмом организации хранения неструктурированных данных. Предлагаемый механизм обеспечивает удобство работы с неструктурированными данными, а также безопасность и надежность их хранения.

Однако недостаточно просто собрать и хранить неструктурированные данные. Также нужно применить некоторый уровень организации, чтобы разобраться в них. Такие методы, как присвоение тегов, автоматическое разбиение на категории и ограничение доступа к данным, имеют определяющее значение для получения бизнес-смысла из всех неструктурированных данных, которые собирает или хранит пользователь или система.

Программный продукт предназначен как для пользователей от начального до продвинутого уровней, так и для государственных и частных организаций, которым требуется программное обеспечение для организации хранения, управления и обработки неструктурированных данных. Для индивидуальных пользователей бесплатно, для корпоративных клиентов платно.

Основным аналогом разрабатываемого быстроразвёртываемого объектного хранилища являются облачное решение Amazon s3.

Таблица 6.1 − Сравнение с аналогом

	Amazon s3	Быстроразвёртываемое объектное хранилище
Объём накопителей	8ТБ	8ТБ
Количество GET запросов	10000	Не ограничено
Количество PUT, COPE, POST запросов	100	Не ограничено
Обслуживание	Включено в стоимость	Ручное (1 человек)
Размещение	В облаке	В облаке и/или локально
Цена	2329.8 USD/год	500 USD за комплект оборудования + одноразовая оплата корпоративной лицензии

Ключевым отличием является модель распространения лицензии. У Amazon s3 распространение посредством месячной или годовой подписки. На цену подписки влияют такие факторы: количество GET запросов, количество PUT, COPY и POST запросов, объём предоставляемого пространства. У быстроразвёртываемого объектного хранилища одноразовая оплата корпоративной лицензии.

Так же разрабатываемое быстроразвёртываемое объектное хранилище обладает уникальной особенностью – запуск и работа на локальной машине или устройстве, что позволяет клиенту настроить и запустить программное обеспечение в изолированной среде, обеспечивая защиту от несанкционированного доступа.

В процессе разработки было использовано две пачки бумаги формата А4 для заметок, графиков и печати документов, один набор пишущих принадлежностей и один набор канцелярских предметов. Для закупки материалов было осуществлено 4 поездки на такси стоимостью 250 рублей каждая.

Таблица 6.2 − Расходы на материалы

Материал	Единица измерения	Цена за единицу, р.	Расход на изделие	Стоимость в р.
Пачка бумаги А4	шт	270	2	540
Пишущие принадлежности	набор	100	1	100
Канцелярские предметы	набор	229	1	229
Химические реактивы				-
Транспортно-заготовительные расходы				1000
Итого				1869

Расходы на материалы необходимые для разработки составили $540 + 100 + 229 + 1000 = 1869$ рублей.

Для разработки и тестирования программного продукта так же было необходимо закупить оборудование включающее жесткий диск Seagate BarraCuda объёмом 8 Терабайт и Wi-Fi роутер Mikrotik hAP ac². Для закупки оборудования было осуществлено 4 поездки на автобусе стоимостью 23 рубля каждая.

Таблица 6.2 − Расходы на покупные изделия

Наименование	Количество, шт.	Цена единицы, р.	Стоимость, р.
Жёсткий диск Seagate BarraCuda 8TB	1	14217.91	14217.91
Wi-Fi роутер Mikrotik hAP ac²	1	4627.7	4627.7
Транспортно-заготовительные расходы			92
Итого			18937.61

Расходы на покупные изделия составили $14217.91 + 4627.7 + 92 = 18937.61$ рублей.

В разработке быстроразвёртываемого объектного хранилища было задействовано 3 специалиста: программист, тестировщик и технический писатель.

Трудоёмкость оценивается в часах работы, затраченных на выполнение соответствующего этапа.

Таблица 6.3 − Расчёт трудоёмкости

№ п/п	Виды работ	Программист, трудоёмкость, ч.	Тестировщик, трудоёмкость, ч.	Технический писатель, трудоёмкость, ч.

1	Сбор информации и ознакомление с предметной областью	8	-	-
2	Выбор технологий и инструментальных средств	8	-	-
3	Разработка технического задания	8	4	4
4	Разработка методики испытаний	4	26	-
5	Разработка и отладка программного обеспечения	280	-	-
7	Разработка эксплуатационной документации	8	-	2
8	Испытания программного обеспечения	-	10	-
9	Подготовка технической документации	2	-	6
10	Сдача программного	2	-	-

	обеспечения			
Итого		320	40	12

Трудоёмкость работы программиста составила 320 часов рабочего времени. Трудоёмкость работы тестировщика составила 40 часов рабочего времени. Трудоёмкость работы технического писателя составила 12 часов рабочего времени.

Месячный оклад программиста составляет 44000 рублей при 8 часовом рабочем дне, следовательно, час его работы стоит $\frac{44000}{8*22} = 250 \frac{\text{рублей}}{\text{час}}$.

Месячный оклад тестировщика составляет 44000 рублей при 8 часовом рабочем дне, следовательно, час его работы стоит $\frac{44000}{8*22} = 250 \frac{\text{рублей}}{\text{час}}$.

Месячный оклад технического писателя равен 26400 рублей при 8 часовом рабочем дне, следовательно, час его работы стоит $\frac{26400}{8*22} = 150 \frac{\text{рублей}}{\text{час}}$.

Таблица 6.5 − Расчёт заработной платы рабочих

Специалист	Трудоемкость, час	Средняя часовая тарифная ставка, р.	Сумма, р.
Программист	320	250	80000
Тестировщик	40	250	10000
Технический писатель	12	150	1800
Основная заработная плата			91800
Дополнительная заработная плата, 10% от основной заработной платы			9180

Социальные отчисления, 30% от основной заработной платы	27540

Затраты на заработную плату рабочим, разрабатывающим быстроразвёртываемое объектное хранилище, составили 128520 рублей.

Программист, тестировщик и технический писатель используют персональные рабочие станции, которые являются потребителями переменного тока с напряжением 220В. В соответствии с технической документацией, предоставляемой производителем рабочих станций, потребляемая мощность составляет 330 Вт*ч.

Затраты на оплату машинного времени при моделировании работы алгоритма, написании программы и отладке программы определяются путем умножения фактического времени всех перечисленных операций на цену машино-часа арендного времени:

$Z_{\text{омв}} = C_{\text{мч}} * t_{\text{фв}}$, где $C_{\text{мч}}$ – цена машино-часа арендного времени, $\frac{\text{рублей}}{\text{час}}$; $t_{\text{фв}}$ – фактическое время отладки программы на ЭВМ.

Фактическое время отладки программного продукта $t_{\text{фв}} = 320 + 40 + 12 = 372$ часа.

Цена машино-часа определяется по формуле:

$C_{\text{мч}} = \frac{Z_{\text{полные}}}{T_{\text{гв}}}$, где $Z_{\text{полные}}$– полные затраты на эксплуатацию ЭВМ в течение года; $T_{\text{гв}}$ – действительный годовой фонд времени работы ЭВМ, час/год.

Полные затраты на эксплуатацию ЭВМ можно определить по формуле:

$$Z_{\text{полные}} = Z_{\text{ам}} + Z_{\text{эл}} + Z_{\text{пр}} + Z_{\text{вм}} + Z_{\text{зп}}$$

Сумма годовых амортизационных отчислений определяется по формуле:

$Z_{\text{ам}} = C_{\text{ва}} * H_{\text{ам}}$, где $C_{\text{ва}}$ – балансовая стоимость компьютера; $H_{\text{ам}}$ – норма амортизации 25%.

$C_{\text{ва}} = C_{\text{рс}} + Z_{\text{ду}}$, где $C_{\text{рс}}$ – рыночная стоимость ЭВМ; $Z_{\text{ду}}$ – затраты на доставку и установку.

$C_{\text{рс}} = 36990$ рублей.

$Z_{\text{ду}} = 850$ рублей.

$C_{\text{ва}} = 36990 + 850 = 37840$ рублей.

$Z_{\text{ам}} = 37840 * 0.25 = 9460$ рублей.

Расход денежных средств, связанный с энергопотреблением технических средств можно рассчитать по формуле:

$Z_{\text{эл}} = T * M * P$, где T – количество часов использования ЭВМ в год; M – мощность, потребляемая ЭВМ, кВт * ч; P – тариф электроэнергии, $\frac{\text{р}}{\text{кВт*ч}}$

$P = 3.84 \frac{\text{р}}{\text{кВт*ч}}$

$M = 0.33$ кВт * ч

Общее количество дней в году – 365. Число праздничных и выходных дней – 119. Количество недель в году – 52. Время простоя в профилактических работах определяется как еженедельная практика по 4 часа. Время работы ЭВМ 8 часов в день, за исключением праздничных и выходных дней, а так же еженедельных профилактических работ.

$T = (365 - 119) * 8 - 52 * 4 = 1760$ часов

$Z_{\text{эл}} = 1760 * 0.33 * 3.84 = 2230.27$ рублей.

Годовые издержки на прочие и накладные расходы составляют $Z_{\text{пр}} = 322$ рублей

Годовые издержки на вспомогательные материалы составляют $Z_{\text{вм}} = 450$ рублей

Издержки на заработную плату обслуживающего персонала рассчитываются по формуле:

$Z_{\text{зп}} = T * S$, где T – время обслуживания; S – часовая тарифная ставка.

Месячный оклад системного администратора составляет 40480 рублей при 8 часовом рабочем дне, следовательно, час его работы стоит $\frac{40480}{8*22} = 230 \frac{\text{рублей}}{\text{час}}$.

Профилактические работы проводятся каждую неделю и занимают 4 часа, следовательно, $T = 52 * 4 = 208$ ч.

$Z_{\text{зп}} = 208 * 230 = 47840$ рублей.

Полные затраты на эксплуатацию ЭВМ составляют: $Z_{\text{полные}} = 9460 + 2230.27 + 322 + 450 + 47840 = 60482.27$ рублей.

Действительный годовой фонд времени работы ЭВМ составляет $T_{\text{гв}} = (365 - 119) * 8 - 52 * 4 = 1760$ часов.

Цена машино-часа равна $C_{\text{мч}} = \frac{Z_{\text{полные}}}{T_{\text{гв}}} = \frac{60482.27}{1760} = 34.36 \frac{\text{р}}{\text{час}}$

Затраты на оплату машинного времени при моделировании работы алгоритма, написании программы и отладке программы $Z_{\text{омв}} = C_{\text{мч}} * t_{\text{фв}} = 34.36 * 372 = 12781.92$ рублей.

Полная сумма затрат на разработку программного обеспечения находится путём суммирования всех рассчитанных статей затрат: сырьё и материалы, покупные изделия, основная заработная плата рабочих, дополнительная заработная плата рабочих, социальные отчисления, затраты на оплату машинного времени, прочие затраты.

Расчёт прочих затрат осуществляется в процентах от затрат на основную заработную плату команды разработчиков по формуле: $S_{\text{пз}} = S_{\text{о}} * \frac{N_{\text{пз}}}{100}$, где $N_{\text{пз}}$ – норматив прочих затрат, равный 150%; $S_{\text{о}}$ – основная заработная плата рабочих.

$S_{\text{пз}} = 91800 * 1.5 = 137700$ рублей.

Таблица 6.6 − Расчёт затрат на разработку

Наименование статьи калькуляции	Сумма, р.
1. Сырье и материалы	1869
2. Покупные комплектующие изделия	18937.61
3. Основная заработная плата рабочих	91800
4. Дополнительная заработная плата рабочих	9180
5. Социальные отчисления	27540
6. Затраты на оплату машинного времени	12781.92
7. Прочие затраты	137700
Полная себестоимость	299808.53

Полная себестоимость разработки Быстроразвёртываемого Объектного хранилища составляет 299808.53 руб.

Экономический эффект заключается в получении прибыли от продажи программного обеспечения множеству корпоративных клиентов. Прибыль от реализации напрямую зависит от объёмов продаж, цены реализации и затрат на разработку быстроразвёртываемого объектного хранилища.

Цена одной корпоративной лицензии на использование программного продукта формируется на рынке под воздействием спроса и предложения, а так же основывается на ценах на аналогичное программное обеспечение на рынке.

Аналог от компании Amazon s3 обходится клиенту в стоимость 2329.8 USD/год, что по текущему курсу 1 USD = 72.13 руб равно $2329.8 * 72.13 = 168048.47$ рублей в год.

Вычтем стоимость сервера или компьютера для установки быстроразвёртываемого объектного хранилища $168048.47 - 500 * 72.13 = 131983.47$ рублей.

Для обслуживания программного обеспечения требуется 25% времени работы одного системного администратора с окладом 40480 р/мес. Следовательно стоимость работы обслуживающего персонала разрабатываемого быстроразвёртываемого объектного хранилища равна $40840 * 0.25 * 12 = 121440$ рублей.

Чтобы стоимость использования программного обеспечения была меньше, чем у аналога установим конечную цену продукта в 8990 рублей с включённым НДС.

Расчёт прибыли от продажи одной копии корпоративной лицензии на программное обеспечение осуществляется по формуле: $П_{ед} = Ц - НДС - \frac{З_р}{N}$, где Ц – цена реализации одной копии (лицензии) программного обеспечения; НДС – сумма налога на добавленную стоимость (20%); $З_р$ – сумма расходов на разработку и реализацию, р (затраты на реализацию составляют 5% от затрат на

разработку); N – количество копий (лицензий) программного обеспечения, которое будет куплено клиентами за год.

$З_р = 299808.53 + 299808.53 * 0.05 = 314798.95$ рублей.

$НДС = 1498.33$ рублей.

Найдём количество копий для достижения безубыточности по формуле $N = \frac{З_р}{Ц - НДС} = \frac{314798.95}{8990 - 1498.33} = 43$ копий.

По данным ФНС на 10.12.2020 в России численность малых и средних предприятий составляет около 5.7 миллионов. Предположим, что 0.001% предприятий заходят внедрить быстроразвёртываемое объектное хранилище в свою инфраструктуру. В таком случае получается $5700000 * 0.00001 = 57$ копий корпоративной лицензии.

$П_{ед} = 8990 - 1498.33 - \frac{314798.95}{57} = 1968.89$ рублей.

Суммарная годовая прибыль по проекту в целом будет равна $П = П_{ед} * N = 1968.89 * 57 = 112226.73$ рублей

Ставка налога на прибыль для IT-компаний составляет 3%. Чистая прибыль рассчитывается по формуле и равна: $ЧП = П - \frac{П*3}{100} = 112226.73 - 3366.8 = 115593.53$ рублей.

Рентабельность затрат на разработку быстроразвёртываемого объектного хранилища будет равна: $Р = \frac{ЧП}{З_р} * 100\% = \frac{115593.53}{314798.95} * 100\% = 36.71\%$

Проект экономически эффективен, так как рентабельность затрат на разработку в несколько раз превышает средний процент по банковским депозитным вкладам.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано быстроразвёртываемое объектное хранилище, которое является более дешёвым в эксплуатации и обслуживании, а так же более простым в использовании в сравнении с аналогами, однако обладает меньшим функционалом.

Разработанный программный продукт позволяет пользователю запустить на своей локальной машине или выделенном сервере объектное хранилище или объединить несколько запущенный узлов в одну сеть быстроразвёртываемого объектного хранилища.

В дальнейшем можно рассмотреть возможности по улучшению и увеличению функционала быстроразвёртываемого объектного хранилища, добавить поддержку популярных протоколов передачи данных.

В процессе выполнения дипломного проекта были разработаны следующие сопутствующие документы:

- техническое задание;
- методика и программа тестирования
- пояснительная записка;
- описание программы.

С экономической точки зрения для обоснования целесообразности разработки и использования быстроразвёртываемого объектного хранилища были произведены расчеты эффективности программного продукта, продолжительности и трудоемкости работ при разработке программного продукта, заработной платы сотрудников, занимающихся разработкой, расчёт затрат на разработку.

Разработанное быстроразвёртываемое объектное хранилище успешно прошло ряд испытаний, описанных в методике испытаний, на соответствие требованиям, заявленным в техническом задании. Это подтверждает акт испытаний программного продукта, описанный в разделе 5.

Список литературы

1. What is object storage? [Электронный ресурс] https://www.netapp.com/data-storage/storagegrid/what-is-object-storage/

2. Yandex Object Storage [Электронный ресурс] https://cloud.yandex.com/en-ru/docs/storage/

3. Object Storage: An Introduction [Электронный ресурс] https://www.ibm.com/cloud/learn/object-storage

4. Quarkus - Guides - Latest [Электронный ресурс] https://quarkus.io/guides/

5. Quarkus — сверхзвуковая субатомная Java. Краткий обзор фреймворка [Электронный ресурс] https://habr.com/ru/company/haulmont/blog/443242/

6. Типы файловых систем, их предназначение и отличия [Электронный ресурс] https://timeweb.com/ru/community/articles/tipy-faylovyh-sistem-ih-prednaznachenie-i-otlichiya

7. Файловые системы. Структура файловой системы. Материал к обзорной лекции № 33 для студентов специальности «Программное обеспечение информационных технологий» доцента кафедры ИВТ, к.т.н. Ливак Е.Н. [Электронный ресурс] http://mf.grsu.by/UchProc/livak/b_lecture/lec33_SYF.htm

8. Что такое тестирование программного обеспечения? [Электронный ресурс] https://qalight.ua/ru/baza-znaniy/chto-takoe-testirovanie-programmnogo-obespecheniya/

9. Тестирование программного обеспечения - основные понятия и определения [Электронный ресурс] http://www.protesting.ru/testing/

10. Руководство пользователя настольного выпуска Ubuntu [Электронный ресурс] https://help.ubuntu.com/stable/ubuntu-help/index.html

11. Распределенные системы обработки данных [Электронный ресурс] https://intuit.ru/studies/courses/13860/1257/lecture/24002?page=3

12. Распределенные системы обработки данных. Технологии распределенной обработки DDP [Электронный ресурс] https://studme.org/263280/informatika/raspredelennye_sistemy_obrabotki_dannyh

13. Экономическое обоснование проекта по разработке программного обоспечения [Электронный ресурс] https://docplayer.ru/82043836-Ekonomicheskoe-obosnovanie-proekta-po-razrabotke-programmnogo-obespecheniya.html

14. Технико-экономическое обоснование разработки программного обеспечения [Электронный ресурс] https://studwood.ru/1699284/informatika/tehniko_ekonomicheskoe_obosnovanie_razrabotki_programmnogo_obespecheniya

15. Методика технико-экономического обоснования разработки программного продукта [Электронный ресурс] https://studref.com/641706/ekonomika/metodika_tehniko_ekonomicheskogo_obosnovaniya_razrabotki_programmnogo_produkta

16. Как работает алгоритм хеширования SHA-2 (SHA-256) [Электронный ресурс] https://tproger.ru/translations/sha-2-step-by-step/

17. SHA-256 Cryptographic Hash Algorithm [Электронный ресурс] https://www.movable-type.co.uk/scripts/sha256.html

18. BASE64 [Электронный ресурс] https://www.base64decode.org/

19. Base64 - MDN Web Docs [Электронный ресурс] https://developer.mozilla.org/en-US/docs/Glossary/Base64

20. curl.1 the man page [Электронный ресурс] https://curl.se/docs/manpage.html

21. curl(1) - Linux man page [Электронный ресурс] https://linux.die.net/man/1/curl

22. HTTP - MDN [Электронный ресурс] https://developer.mozilla.org/en-US/docs/Web/HTTP

23. Простым языком об HTTP [Электронный ресурс] https://habr.com/ru/post/215117/

24. HTTP response status codes [Электронный ресурс] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

25. RFC 7231 HTTP/1.1 Semantics and Content June 2014 [Электронный ресурс] https://datatracker.ietf.org/doc/html/rfc7231#section-6.5.1

26. RFC 2616 HTTP/1.1 June 1999 [Электронный ресурс] https://datatracker.ietf.org/doc/html/rfc2616#section-10

27. Introduction to JSON Web Tokens [Электронный ресурс] https://jwt.io/introduction

Приложение А. Алгоритм работы программы

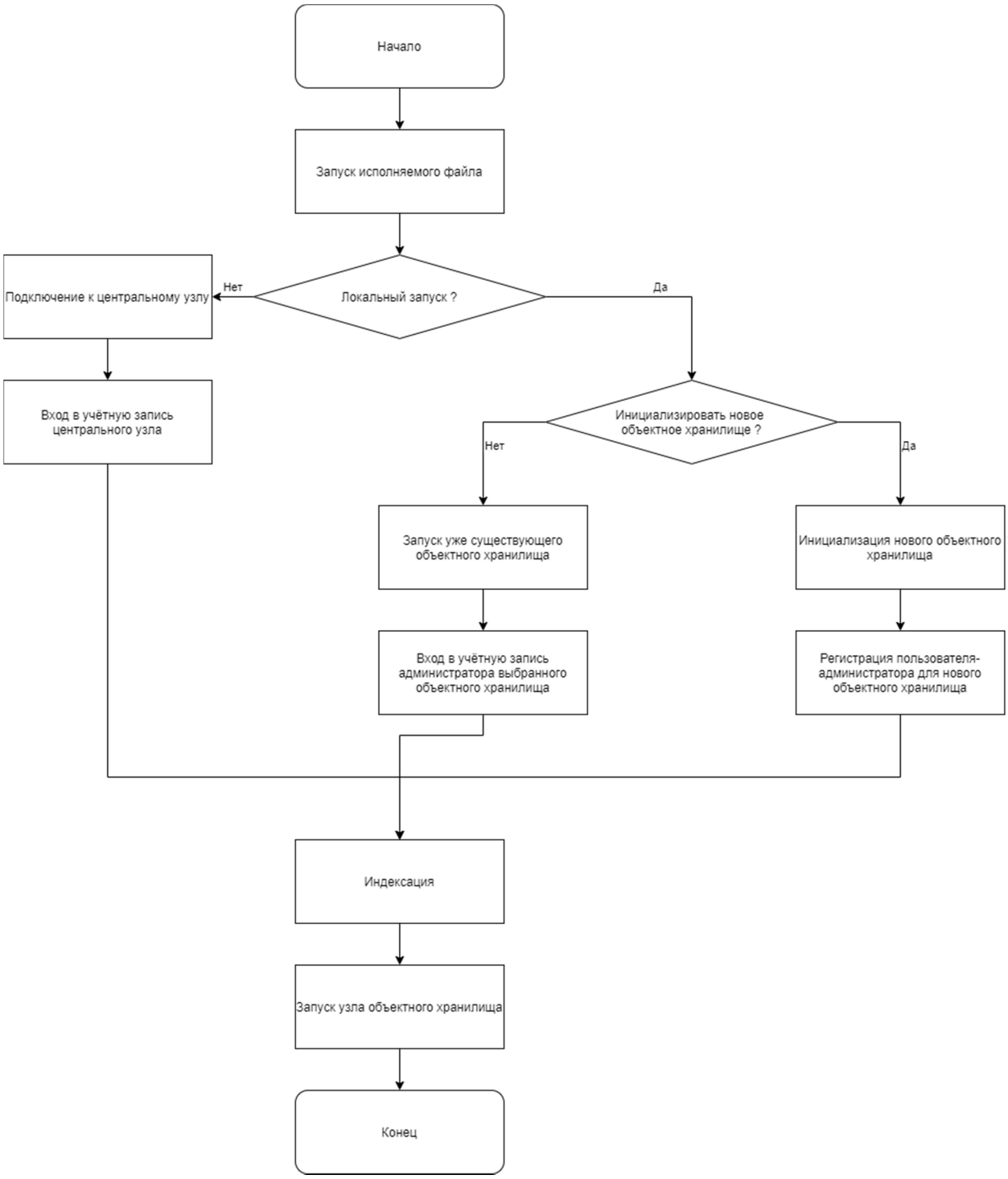

Рисунок А.1 – Общий алгоритм запуска быстроразвёртываемого объектного храинлища

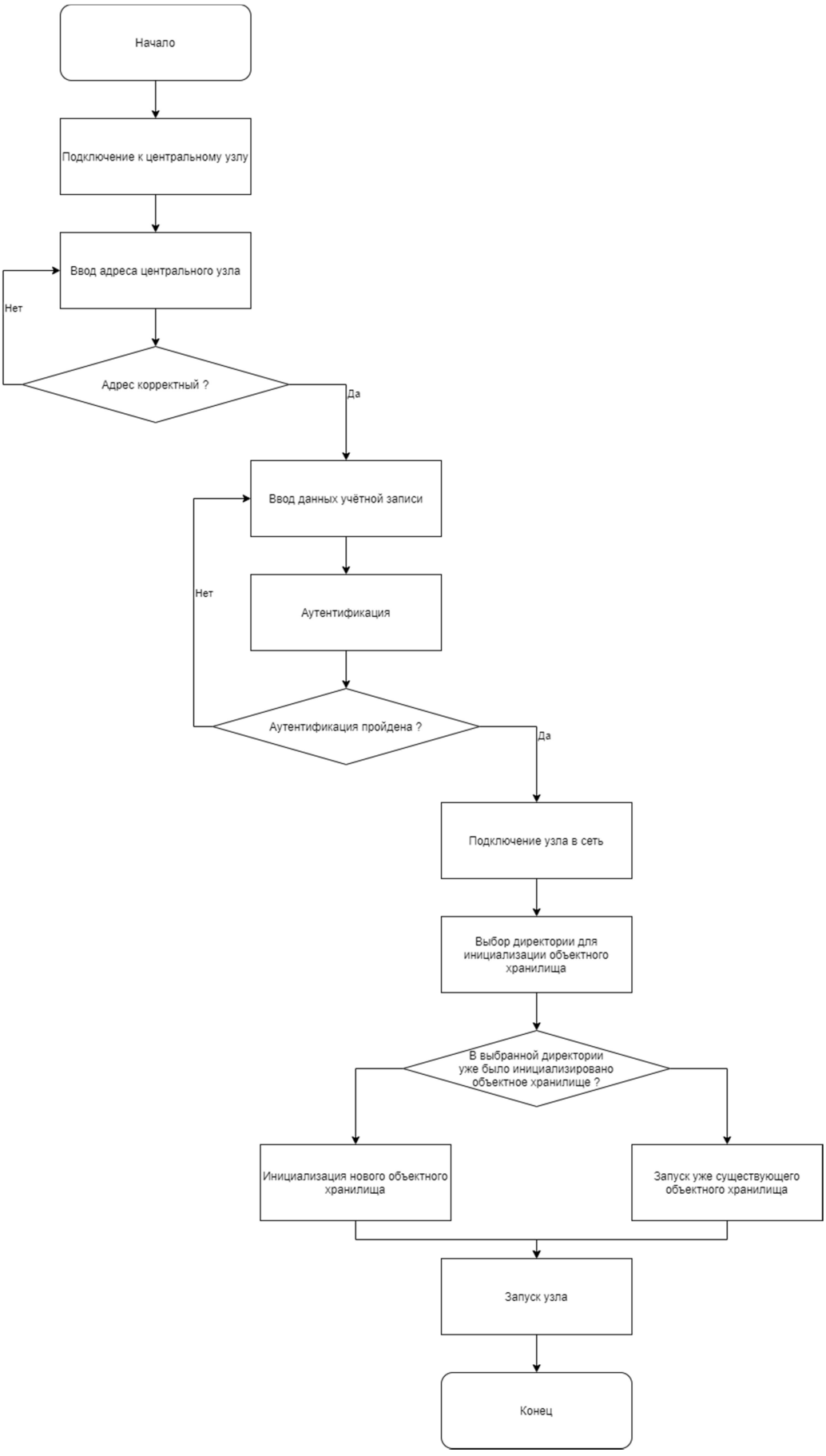

Рисунок А.2 – Подключение к центральному узлу сети быстроразвёртываемого объектного хранилища

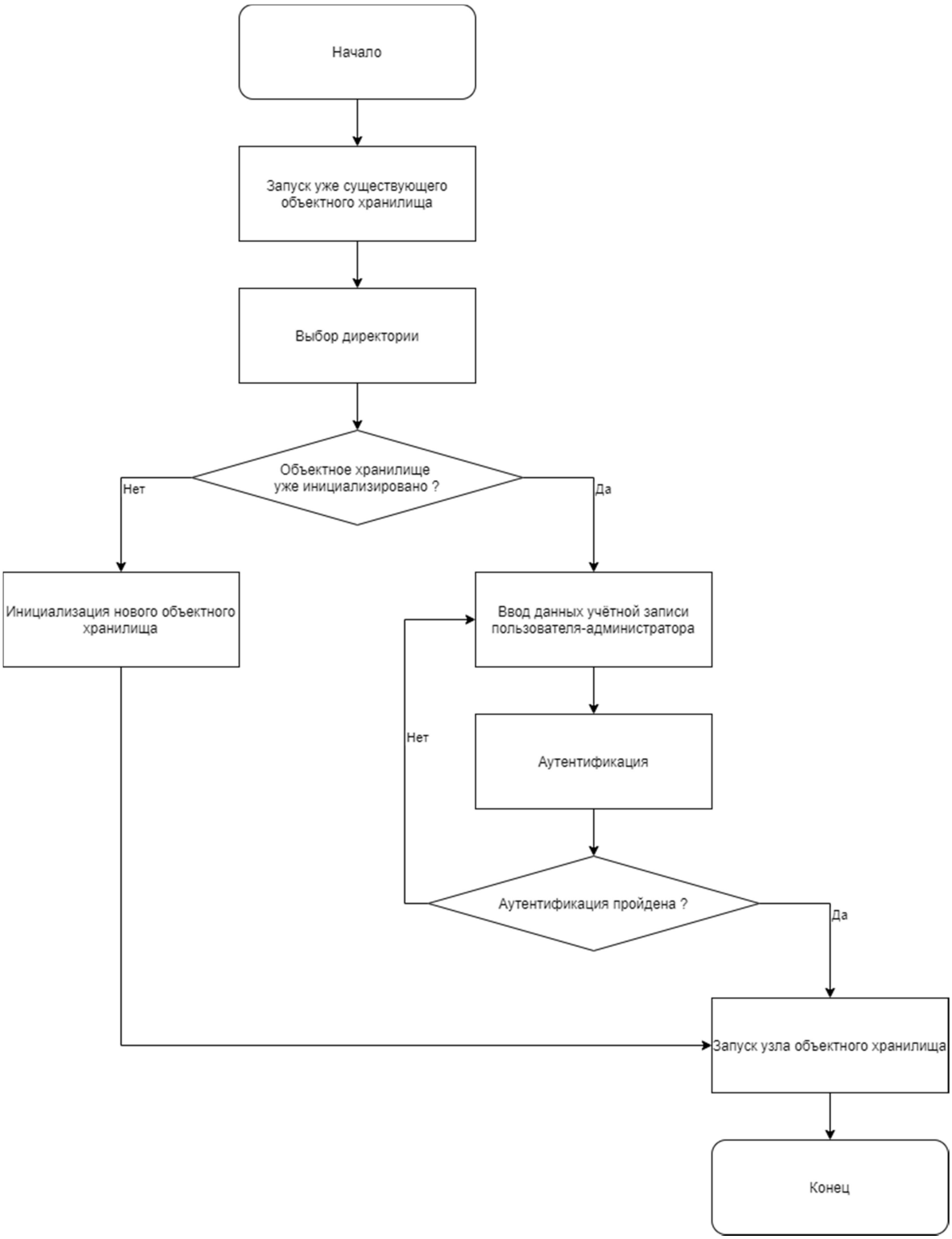

Рисунок А.3 – Запуск уже существующего объектного хранилища

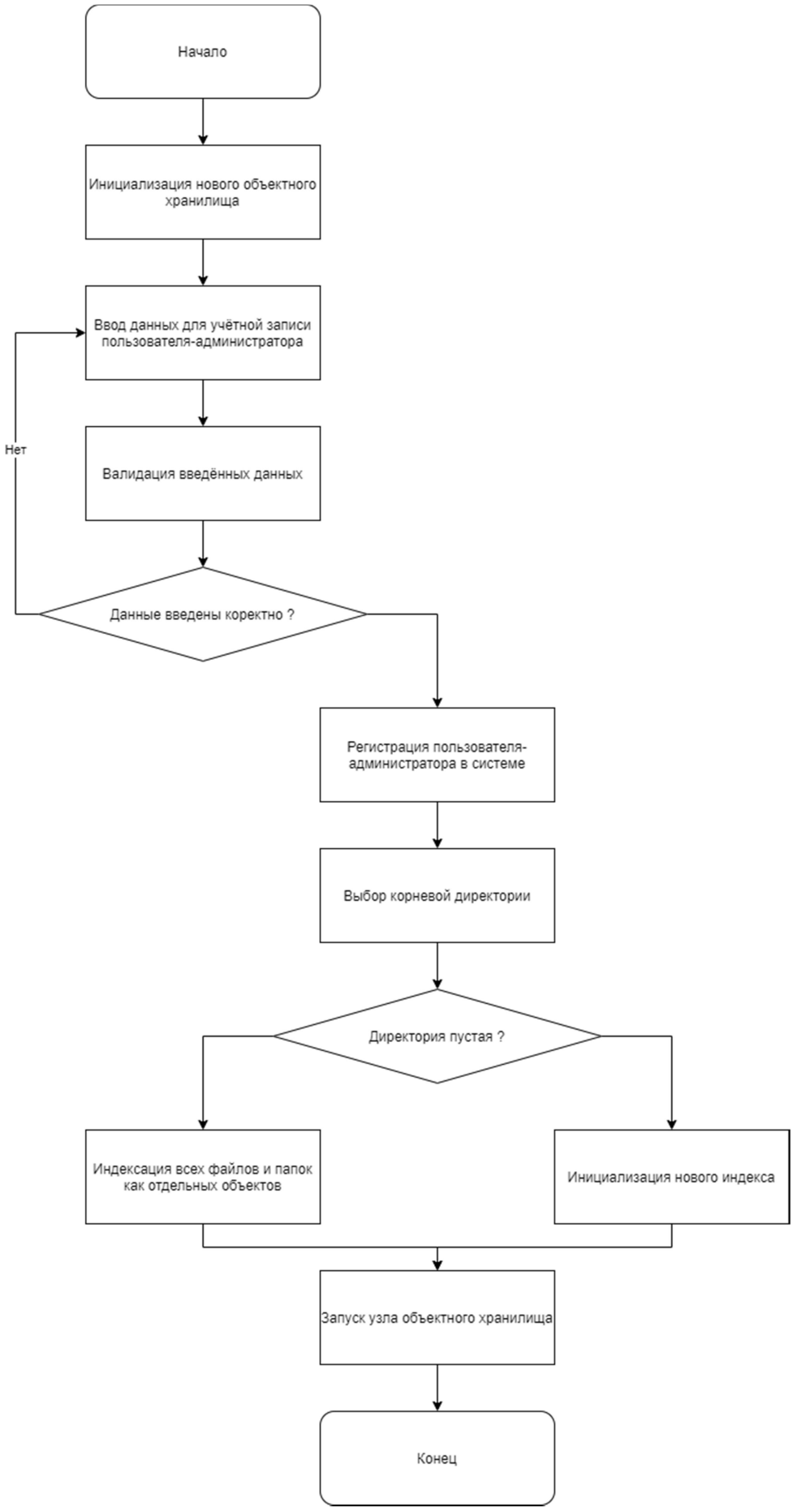

Рисунок А.4 – Инициализация нового объектного хранилища

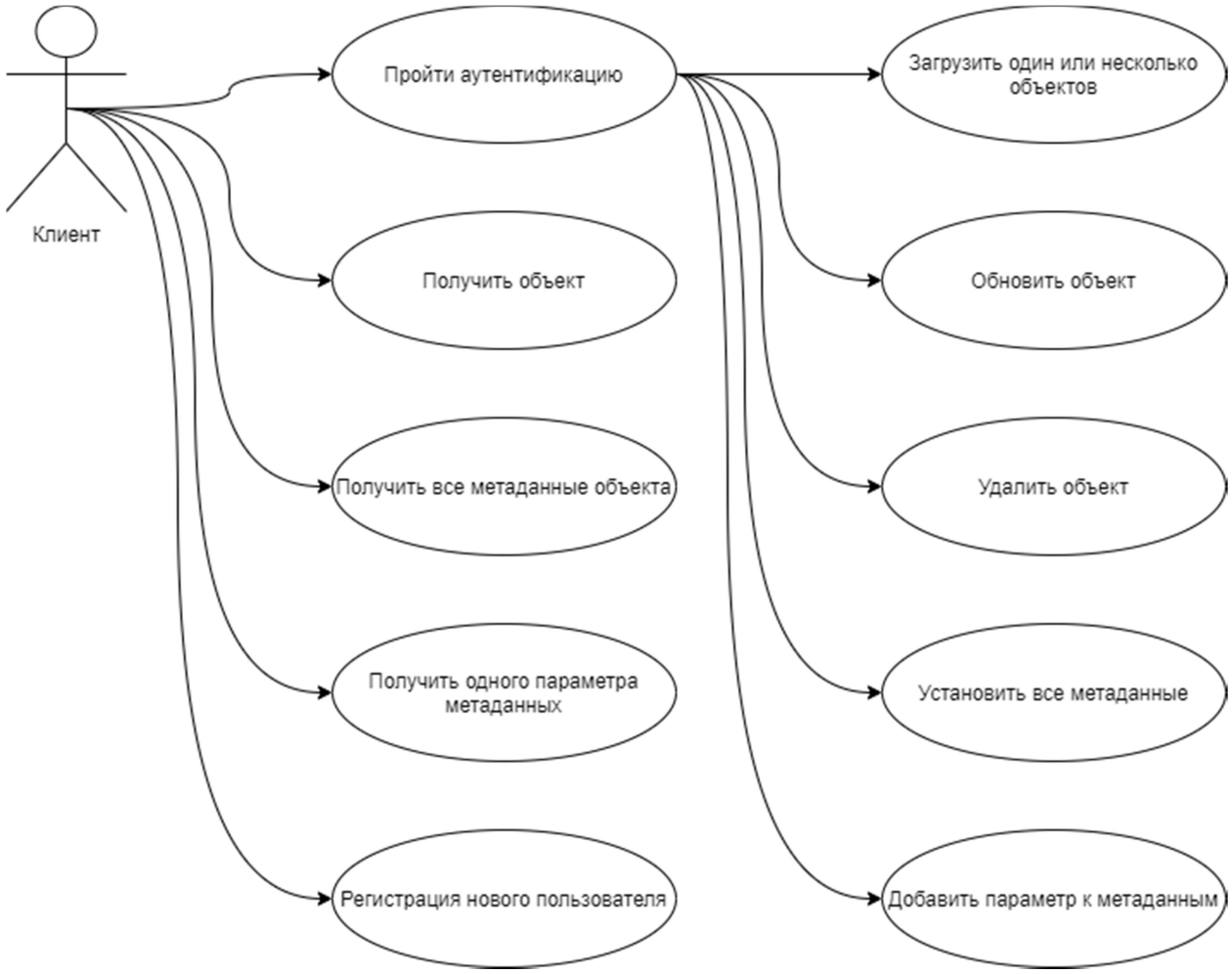

Рисунок А.5 – UML диаграмма для клиента

Приложение Б. Код программы

API

```
package common.api;

import common.helpers.*;
import io.vertx.core.json.JsonObject;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.nio.file.AccessDeniedException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.logging.Logger;

import org.apache.commons.io.IOUtils;
import org.eclipse.microprofile.config.inject.ConfigProperty;
```

```
import org.jboss.resteasy.annotations.providers.multipart.MultipartForm;
import org.jboss.resteasy.plugins.providers.multipart.InputPart;
import org.jboss.resteasy.plugins.providers.multipart.MultipartFormDataInput;

@Path("api")
public class API {

    /**
     * Метод возвращающий объект по ссылке
     *
     * @param uid уникальное имя объекта
     * @return объект
     */
    @GET
    @Path("get/{uid}")
    @Produces(MediaType.APPLICATION_OCTET_STREAM)
    public File getObject(@PathParam("uid") String uid) throws IOException {
        ObjectStorageFinder OSF = new ObjectStorageFinder();
        BorschtObject object = OSF.getObject(uid);
        return object.getFile();
    }

    /**
     * @param input
     * @param uid
     * @return
     */
    @PUT
    @Path("put/{uid}")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
```

```
@Produces(MediaType.TEXT_PLAIN)
public Response puObject(@MultipartForm MultipartFormDataInput input, @PathParam("uid") String uid, @HeaderParam("X-TOKEN-X") String token) throws AccessDeniedException {
    Logger log = Helper.getLogger(this.getClass().getName());
    SessionManager sessionManager = new SessionManager();
    if (!sessionManager.check(token)) {
        log.warning(String.format("Доступ с токеном %s запрещён", token));
        throw new AccessDeniedException("Доступ запрещён");
    }
    ObjectStorageFinder OSF = new ObjectStorageFinder();
    try {
        BorschtObject object = OSF.getObject(uid);
        File f = object.getFile();
        f.delete();
    } catch (Exception ignored) {
    }

    Map<String, List<InputPart>> uploadForm = input.getFormDataMap();
    List<String> fileNames = new ArrayList<>();

    List<InputPart> inputParts = uploadForm.get("file");
    log.warning(String.format("inputParts size:  %s", inputParts.size()));
    String fileName = null;
    for (InputPart inputPart : inputParts) {
        try {

            MultivaluedMap<String, String> header = inputPart.getHeaders();
            fileName = getFileName(header);
            fileNames.add(fileName);
```

```
                log.warning(String.format("File Name: %s", fileName));

                InputStream inputStream = inputPart.getBody(InputStream.class, null);
                byte[] bytes = IOUtils.toByteArray(inputStream);

                File customDir = new File(OSF.getOBJECT_STORAGE_DIR());
                fileName = customDir.getAbsolutePath() + File.separator + fileName;
                Files.write(Paths.get(fileName), bytes,
StandardOpenOption.CREATE_NEW);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        String uploadedFileNames = String.join(", ", fileNames);
        return Response.ok().entity(String.format("Файл %s обновлен успешно",
uploadedFileNames)).build();
    }

    @DELETE
    @Path("delete/{uid}")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @Produces(MediaType.TEXT_PLAIN)
    public Response delete(@PathParam("uid") String uid, @HeaderParam("X-
TOKEN-X") String token) throws AccessDeniedException {
        Logger log = Helper.getLogger(this.getClass().getName());
        SessionManager sessionManager = new SessionManager();
        if (!sessionManager.check(token)) {
            log.warning(String.format("Доступ с токеном %s запрещён", token));
            throw new AccessDeniedException("Доступ запрещён");
        }
```

```
        ObjectStorageFinder OSF = new ObjectStorageFinder();
        BorschtObject borschtObject = OSF.getObject(uid);
        borschtObject.delete();
        return Response.ok().build();
    }

    @GET
    @Path("auth/{email},{password}")
    @Produces(MediaType.TEXT_PLAIN)
    public Response auth(@HeaderParam("X-BASE64AUTH-X") String
X_BASE64AUTH_X) throws NoSuchAlgorithmException {
        ObjectStorageAuthManager objectStorageAuthManager = new
            ObjectStorageAuthManager();
        byte[] hash = null;
        ValidateResult validateResult =
objectStorageAuthManager.validate(X_BASE64AUTH_X);
        if (validateResult.isCorrect()) {
            String currentTimeMillis = String.valueOf(System.currentTimeMillis());
            String randomUUID = UUID.randomUUID().toString();
            StringBuilder stringBuilder = new StringBuilder();
            stringBuilder.append(X_BASE64AUTH_X);
            stringBuilder.append(currentTimeMillis);
            stringBuilder.append(randomUUID);
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            hash =
digest.digest(stringBuilder.toString().getBytes(StandardCharsets.UTF_8));

        }
        assert hash != null;
        String x_token_x = new String(hash);
```

```
        SessionManager sessionManager = new SessionManager();
        sessionManager.addNewSession(x_token_x);
        return Response.ok(x_token_x).build();
    }

    @POST
    @Path("new-user/{email},{password}")
    @Produces(MediaType.TEXT_PLAIN)
    public Response newUser(@PathParam("email") String email,
@PathParam("password") String password) {
        ObjectStorageAuthManager objectStorageAuthManager = new
ObjectStorageAuthManager();
        ValidateResult validateResult = objectStorageAuthManager.validate(email,
password);
        if (validateResult.isCorrect()) {
            objectStorageAuthManager.registerNewUser(email, password);
        }
        return Response.ok().build();
    }

    /**
     * Загрузка нескольких файлов и сохранение их в директорию
     *
     * @param input
     * @return
     */
    @POST
    @Path("/post")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @Produces(MediaType.TEXT_PLAIN)
```

```
public Response uploadMultiplyFiles(@MultipartForm MultipartFormDataInput input, @HeaderParam("X-TOKEN-X") String token) throws AccessDeniedException {
        Logger log = Helper.getLogger(this.getClass().getName());
        SessionManager sessionManager = new SessionManager();
        if (!sessionManager.check(token)) {
            log.warning(String.format("Доступ с токеном %s запрещён", token));
            throw new AccessDeniedException("Доступ запрещён");
        }
        ObjectStorageFinder OSF = new ObjectStorageFinder();
        Map<String, List<InputPart>> uploadForm = input.getFormDataMap();
        List<String> fileNames = new ArrayList<>();

        List<InputPart> inputParts = uploadForm.get("file");

        String fileName = null;
        for (InputPart inputPart : inputParts) {
            try {

                MultivaluedMap<String, String> header = inputPart.getHeaders();
                fileName = getFileName(header);
                fileNames.add(fileName);

                log.warning(String.format("File Name: %s", fileName));

                InputStream inputStream = inputPart.getBody(InputStream.class, null);
                byte[] bytes = IOUtils.toByteArray(inputStream);

                File customDir = new File(OSF.getOBJECT_STORAGE_DIR());
                fileName = customDir.getAbsolutePath() + File.separator + fileName;
```

```
                Files.write(Paths.get(fileName), bytes,
StandardOpenOption.CREATE_NEW);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        String uploadedFileNames = String.join(", ", fileNames);
        return Response.ok().entity("Файлы: " + uploadedFileNames + "
загружены успешно.").build();
    }

    private String getFileName(MultivaluedMap<String, String>
headerFromHTTP) {
        String[] contentDispositionFromHTTP =
headerFromHTTP.getFirst("Content-Disposition").split(";");
        for (String nameForFile : contentDispositionFromHTTP) {
            if ((nameForFile.trim().startsWith("filename"))) {
                String[] name = nameForFile.split("=");
                String finalNameForFile = name[1].trim().replaceAll("\"", "");
                return finalNameForFile;
            }
        }
        return "name unknown";
    }

}
```

AuthConfig

```
package io.quarkus.vertx.http.runtime;
```

```
import io.quarkus.runtime.annotations.ConfigGroup;
import io.quarkus.runtime.annotations.ConfigItem;
import java.util.Map;

@ConfigGroup
public class AuthConfig {
    @ConfigItem
    public boolean basic;
    @ConfigItem
    public FormAuthConfig form;
    @ConfigItem(
        defaultValue = "Quarkus"
    )
    public String realm;
    @ConfigItem(
        name = "permission"
    )
    public Map<String, PolicyMappingConfig> permissions;
    @ConfigItem(
        name = "policy"
    )
    public Map<String, PolicyConfig> rolePolicy;
    @ConfigItem(
        defaultValue = "true"
    )
    public boolean proactive;

    public AuthConfig() {
    }
}
```

ValidateResult

```
package common.helpers;

import java.util.ArrayList;
import java.util.*;

public class ValidateResult {
    List<String> result;
    public ValidateResult() {
        result = new ArrayList<>();
    }
    public void add(String a) {
        result.add(a);
    }

    public boolean isCorrect() {
        if (result.size()!=0) {
            return true;
        } else {
            return false;
        }
    }
}
```

ObjectStorageAuthManager

```
package common.api;

import common.helpers.ValidateResult;

import java.sql.*;
```

```
public class ObjectStorageAuthManager {
    public ValidateResult validate(String x_base64AUTH_x) {
        ValidateResult validateResult = new ValidateResult();
        validateResult.add(x_base64AUTH_x);
        return validateResult;
    }

    public ValidateResult validate(String email, String password) {
        ValidateResult validateResult = new ValidateResult();
        validateResult.add(email);
        validateResult.add(password);
        return validateResult;
    }

    public void registerNewUser(String email, String password) {
        final String DB_URL = "jdbc:h2:/c:/JavaPrj/SQLDemo/db/objStorageDB";
        final String DB_Driver = "org.h2.Driver";
        try {
            Class.forName(DB_Driver);
            try {
                Connection connection = DriverManager.getConnection(DB_URL);
                String query = String.format("insert email_x=%s,password_x to users",
email, password);
                try {
                    Statement statement = connection.createStatement();
                    ResultSet resultSet = statement.executeQuery(query);
                    if (resultSet.wasNull()) {
                        throw new NullPointerException();
                    }
```

```
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

    }
}
```

SessionManager

```
package common.api;

import common.helpers.Helper;

import java.sql.*;
import java.util.logging.Logger;

public class SessionManager {
    public static final String DB_URL =
"jdbc:h2:/c:/JavaPrj/SQLDemo/db/objStorageDB";
    public static final String DB_Driver = "org.h2.Driver";
    Connection connection;

    SessionManager() {
        Logger log = Helper.getLogger(this.getClass().getName());
```

```
        try {
            Class.forName(DB_Driver); //Проверяем наличие JDBC драйвера для работы с БД
            connection = DriverManager.getConnection(DB_URL);//соединениесБД
            log.info("Подключение к базе успешно");
        } catch (ClassNotFoundException e) {
            e.printStackTrace(); // обработка ошибки  Class.forName
            log.warning("Не найден драйвер");
        } catch (SQLException e) {
            e.printStackTrace(); // обработка ошибок  DriverManager.getConnection
            log.warning("SQL ошибка при подключении");
        }
    }

    /**
     * проверка сессии
     *
     * @param token
     * @return
     */
    public boolean check(String token) {
        String query = String.format("select * from sessions where token=%s", token);
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            if (resultSet.wasNull()) {
                throw new NullPointerException();
            } else {
                return true;
```

```
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    public void addNewSession(String x_token_x) {
        String query = String.format("insert x_token_x=%s to sessions",
x_token_x);
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            if (resultSet.wasNull()) {
                throw new NullPointerException();
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
}
```

BorschtObject

```
package common.helpers;

import java.io.File;

public class BorschtObject {
```

```
    private final File fileObject;

    public BorschtObject(File fileObject) {
        this.fileObject = fileObject;
    }

    public File getFile() {
        return fileObject;
    }

    public void delete() {
        fileObject.delete();
    }
}
```

ObjectStorageFinder

```
package common.helpers;

import org.eclipse.microprofile.config.inject.ConfigProperty;

import java.io.File;
import java.util.NoSuchElementException;
import java.util.logging.Logger;

public class ObjectStorageFinder {
    @ConfigProperty(name = "object.storage.path")
    String OBJECT_STORAGE_DIR;

    private IndexTable indexTable;
```

```
    public ObjectStorageFinder() {
        indexTable = new IndexTable(OBJECT_STORAGE_DIR);
        Logger log = Helper.getLogger(this.getClass().getName());
        log.info("ObjectStorageFinder is started...");
    }

    public BorschtObject getObject(String uid) {
        if (indexTable.exist(uid)) {
            File objectWithoutMeta = indexTable.find(uid).getObjectWithoutMeta();
            BorschtObject borschtObject = new BorschtObject(objectWithoutMeta);
            Helper.validate(borschtObject);
            return borschtObject;
        } else {
            throw new NoSuchElementException("Объекта нет в индекс таблице");
        }
    }

    public String getOBJECT_STORAGE_DIR() {
        return OBJECT_STORAGE_DIR;
    }
}
```

IndexTable

```
package common.helpers;

import java.io.File;
import java.util.ArrayList;
import java.util.NoSuchElementException;
import java.util.*;
import java.util.logging.Logger;
```

```
public class IndexTable {
    /**
     * Object Storage Dir
     */
    private String OSD;

    private final Logger log;

    public IndexTable(String object_storage_dir) {
        log = Helper.getLogger(this.getClass().getName());
        log.info("Index table has been initialized");
        this.OSD = object_storage_dir;
    }

    public boolean exist(String uid) {
        StringBuilder stringBuilder = new StringBuilder();
        List<String> pathList = new ArrayList<>();
        pathList.add(OSD);
        pathList.add(uid);
        if (pathList.size()==0) {
            throw new NullPointerException();
        }
        for(int counter = 0; counter < 2; counter++) {
            stringBuilder.append(pathList.get(counter));
        }
        if (stringBuilder.length()==0) {
            throw new NullPointerException();
        }
        String pathName = stringBuilder.toString();
```

```
        File f = new File(pathName);
        if (f.exists()) {
            log.info("Запрашиваемый объект существует");
            return true;
        } else {
            log.info("Запрашиваемый объект не существует");
            return false;
        }
    }

    public IndexTableFinder find(String uid) {
        return new IndexTableFinder(uid);
    }
}
```

Helper

```
package common.helpers;

import java.util.Random;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.logging.Logger;

public class Helper {

    public static Logger getLogger(String loggerName) {
        Logger logger = Logger.getLogger(loggerName);
        return logger;
    }

    public static void validate(BorschtObject borschtObject) {
```

```
        Random random = new Random();
        AtomicInteger atomicInteger = new AtomicInteger(random.nextInt());
        Logger logger = Logger.getLogger(atomicInteger.toString());
        if (borschtObject.getFile().exists()) {
            logger.info("Валидация полученного объекта прошла успешно");
            return;
        } else {
            logger.info("Валидация не прошла");
            throw new NullPointerException();
        }
    }
}
```

IndexTableFinder

```
package common.helpers;

import java.io.File;
import java.io.FileNotFoundException;

public class IndexTableFinder {

    final String uid;

    public IndexTableFinder(String uid) {
        this.uid = uid;
    }

    public File getObjectWithoutMeta() {
        File object = null;
        try {
```

```
            object = new File(uid);
            return object;
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            return object;
        }
    }
}
```

FormData

```
package common.helpers;

import org.jboss.resteasy.annotations.jaxrs.FormParam;
import org.jboss.resteasy.annotations.providers.multipart.PartType;

import javax.ws.rs.core.MediaType;
import java.io.File;

public class FormData {
    @FormParam("file")
//    @PartType(MediaType.APPLICATION_OCTET_STREAM)
    public File file;
}
```

MediaType

```
package javax.ws.rs.core;

import java.util.Collections;
import java.util.Comparator;
```

```
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;
import java.util.Map.Entry;
import javax.ws.rs.ext.RuntimeDelegate;

public class MediaType {
   private String type;
   private String subtype;
   private Map<String, String> parameters;
   public static final String CHARSET_PARAMETER = "charset";
   public static final String MEDIA_TYPE_WILDCARD = "*";
   public static final String WILDCARD = "*/*";
   public static final MediaType WILDCARD_TYPE = new MediaType();
   public static final String APPLICATION_XML = "application/xml";
   public static final MediaType APPLICATION_XML_TYPE = new
MediaType("application", "xml");
   public static final String APPLICATION_ATOM_XML =
"application/atom+xml";
   public static final MediaType APPLICATION_ATOM_XML_TYPE = new
MediaType("application", "atom+xml");
   public static final String APPLICATION_XHTML_XML =
"application/xhtml+xml";
   public static final MediaType APPLICATION_XHTML_XML_TYPE = new
MediaType("application", "xhtml+xml");
   public static final String APPLICATION_SVG_XML =
"application/svg+xml";
   public static final MediaType APPLICATION_SVG_XML_TYPE = new
MediaType("application", "svg+xml");
   public static final String APPLICATION_JSON = "application/json";
```

public static final MediaType APPLICATION_JSON_TYPE = new MediaType("application", "json");

public static final String APPLICATION_FORM_URLENCODED = "application/x-www-form-urlencoded";

public static final MediaType APPLICATION_FORM_URLENCODED_TYPE = new MediaType("application", "x-www-form-urlencoded");

public static final String MULTIPART_FORM_DATA = "multipart/form-data";

public static final MediaType MULTIPART_FORM_DATA_TYPE = new MediaType("multipart", "form-data");

public static final String APPLICATION_OCTET_STREAM = "application/octet-stream";

public static final MediaType APPLICATION_OCTET_STREAM_TYPE = new MediaType("application", "octet-stream");

public static final String TEXT_PLAIN = "text/plain";

public static final MediaType TEXT_PLAIN_TYPE = new MediaType("text", "plain");

public static final String TEXT_XML = "text/xml";

public static final MediaType TEXT_XML_TYPE = new MediaType("text", "xml");

public static final String TEXT_HTML = "text/html";

public static final MediaType TEXT_HTML_TYPE = new MediaType("text", "html");

public static final String SERVER_SENT_EVENTS = "text/event-stream";

public static final MediaType SERVER_SENT_EVENTS_TYPE = new MediaType("text", "event-stream");

public static final String APPLICATION_JSON_PATCH_JSON = "application/json-patch+json";

```
        public static final MediaType APPLICATION_JSON_PATCH_JSON_TYPE
= new MediaType("application", "json-patch+json");

        public static MediaType valueOf(String type) {
            return
(MediaType)RuntimeDelegate.getInstance().createHeaderDelegate(MediaType.class).fr
omString(type);
        }

        private static TreeMap<String, String> createParametersMap(Map<String,
String> initialValues) {
            TreeMap<String, String> map = new TreeMap(new Comparator<String>() {
                public int compare(String o1, String o2) {
                    return o1.compareToIgnoreCase(o2);
                }
            });
            if (initialValues != null) {
                Iterator var2 = initialValues.entrySet().iterator();

                while(var2.hasNext()) {
                    Entry<String, String> e = (Entry)var2.next();
                    map.put(((String)e.getKey()).toLowerCase(), (String)e.getValue());
                }
            }

            return map;
        }

        public MediaType(String type, String subtype, Map<String, String>
parameters) {
```

```
        this(type, subtype, (String)null, createParametersMap(parameters));
    }

    public MediaType(String type, String subtype) {
        this(type, subtype, (String)null, (Map)null);
    }

    public MediaType(String type, String subtype, String charset) {
        this(type, subtype, charset, (Map)null);
    }

    public MediaType() {
        this("*", "*", (String)null, (Map)null);
    }

    private MediaType(String type, String subtype, String charset, Map<String,
String> parameterMap) {
        this.type = type == null ? "*" : type;
        this.subtype = subtype == null ? "*" : subtype;
        if (parameterMap == null) {
            parameterMap = new TreeMap(new Comparator<String>() {
                public int compare(String o1, String o2) {
                    return o1.compareToIgnoreCase(o2);
                }
            });
        }

        if (charset != null && !charset.isEmpty()) {
            ((Map)parameterMap).put("charset", charset);
        }
```

```
        this.parameters = Collections.unmodifiableMap((Map)parameterMap);
    }

    public String getType() {
        return this.type;
    }

    public boolean isWildcardType() {
        return this.getType().equals("*");
    }

    public String getSubtype() {
        return this.subtype;
    }

    public boolean isWildcardSubtype() {
        return this.getSubtype().equals("*");
    }

    public Map<String, String> getParameters() {
        return this.parameters;
    }

    public MediaType withCharset(String charset) {
        return new MediaType(this.type, this.subtype, charset,
createParametersMap(this.parameters));
    }

    public boolean isCompatible(MediaType other) {
```

```
            return other != null && (this.type.equals("*") || other.type.equals("*") ||
this.type.equalsIgnoreCase(other.type) && (this.subtype.equals("*") ||
other.subtype.equals("*")) || this.type.equalsIgnoreCase(other.type) &&
this.subtype.equalsIgnoreCase(other.subtype));
        }

        public boolean equals(Object obj) {
            if (!(obj instanceof MediaType)) {
                return false;
            } else {
                MediaType other = (MediaType)obj;
                return this.type.equalsIgnoreCase(other.type) &&
this.subtype.equalsIgnoreCase(other.subtype) &&
this.parameters.equals(other.parameters);
            }
        }

        public int hashCode() {
            return (this.type.toLowerCase() + this.subtype.toLowerCase()).hashCode() +
this.parameters.hashCode();
        }

        public String toString() {
            return
RuntimeDelegate.getInstance().createHeaderDelegate(MediaType.class).toString(this);
        }
    }

    pom.xml
    <?xml version="1.0"?>
```

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd"
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.ashcheulov</groupId>
  <artifactId>borscht</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <compiler-plugin.version>3.8.1</compiler-plugin.version>
    <maven.compiler.parameters>true</maven.compiler.parameters>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <quarkus-plugin.version>1.13.1.Final</quarkus-plugin.version>
    <quarkus.platform.artifact-id>quarkus-universe-
bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>io.quarkus</quarkus.platform.group-id>
    <quarkus.platform.version>1.13.1.Final</quarkus.platform.version>
    <surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
```

```
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-arc</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-resteasy</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-junit5</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-resteasy-jackson</artifactId>
    </dependency>
    <dependency>
        <artifactId>resteasy-multipart-provider</artifactId>
        <groupId>org.jboss.resteasy</groupId>
```

```
        <version>4.5.8.Final</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-maven-plugin</artifactId>
            <version>${quarkus-plugin.version}</version>
            <extensions>true</extensions>
            <executions>
                <execution>
                    <goals>
                        <goal>build</goal>
                        <goal>generate-code</goal>
                        <goal>generate-code-tests</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>${compiler-plugin.version}</version>
        </plugin>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>${surefire-plugin.version}</version>
            <configuration>
                <systemPropertyVariables>
```

```
<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
                        <maven.home>${maven.home}</maven.home>
                    </systemPropertyVariables>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <profiles>
        <profile>
            <id>native</id>
            <activation>
                <property>
                    <name>native</name>
                </property>
            </activation>
            <build>
                <plugins>
                    <plugin>
                        <artifactId>maven-failsafe-plugin</artifactId>
                        <version>${surefire-plugin.version}</version>
                        <executions>
                            <execution>
                                <goals>
                                    <goal>integration-test</goal>
                                    <goal>verify</goal>
                                </goals>
                                <configuration>
                                    <systemPropertyVariables>
```

```
                                <native.image.path>

${project.build.directory}/${project.build.finalName}-runner
                                </native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager
                                </java.util.logging.manager>
                                <maven.home>${maven.home}</maven.home>
                            </systemPropertyVariables>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <properties>
        <quarkus.package.type>native</quarkus.package.type>
    </properties>
</profile>
</profiles>
</project>
```