МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
(НИЯУ МИФИ)

ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ И ТЕХНОЛОГИЙ
КАФЕДРА №40 «ФИЗИКА ЭЛЕМЕНТАРНЫХ ЧАСТИЦ»

УДК 539.12, 004.42

На правах рукописи

КОРШУНОВА ПОЛИНА АЛЕКСАНДРОВНА

РАЗРАБОТКА ПОДСИСТЕМЫ УПРАВЛЕНИЯ ДАННЫМИ КОМПЛЕКСА ПРОМЕЖУТОЧНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ SPD ONLINE FILTER

Направления подготовки:
09.04.04 «Программная инженерия»,
14.04.02 «Ядерные физика и технологии»
Диссертация на соискание степени магистра

Научный руководитель,
к.ф.-м.н. _____ Е. Ю. Солдатов

Научный консультант,
к.т.н. _____ Д. А. Олейник

Москва 2025

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

РАЗРАБОТКА ПОДСИСТЕМЫ УПРАВЛЕНИЯ ДАННЫМИ КОМПЛЕКСА ПРОМЕЖУТОЧНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ SPD ONLINE FILTER

Студент	_____ П. А. Коршунова
Научный руководитель, к.ф.-м.н.	_____ Е. Ю. Солдатов
Научный консультант, к.т.н.	_____ Д. А. Олейник
Рецензент, к.ф.-м.н.	_____ Н. Н. Войтишин
Рецензент, к.ф.-м.н.	_____ И. И. Денисенко
Секретарь ГЭК, к.ф.-м.н.	_____ Е. Ю. Солдатов
Зав. каф. №40, д.ф.-м.н., проф.	_____ М. Д. Скорохватов
Рук. учеб. прог., к.ф.-м.н.	_____ Е. Ю. Солдатов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Одной из неразрешенных проблем современной физики высоких энергий является «спиновый кризис», который заключается в том, что мы не знаем, как спины нуклонов распределены между их составляющими (кварками и глюонами).

Данный кризис был вызван экспериментом EMC [1], в котором пытались определить распределение спина внутри протона. Ожидалось, что весь спин протона несут валентные кварки, однако оказалось, что это не так.

Изучение спиновой структуры нуклона имеет большое значение, так как он отвечает за фундаментальные свойства природы. Но наши знания о его внутренней структуре все еще ограничены, особенно в отношении вклада глюонов. Именно поэтому строится новая установка SPD (Spin Physics Detector) для всестороннего изучения глюонного состава нуклона и других связанных со спином явлений [2]. При этом изучаемые процессы имеют сравнительно небольшую вероятность, в связи с чем возникает необходимость набора большого объема статистики, а их сложность приводит к невозможности на аппаратном уровне построить триггерную систему.

Таким образом, конструкционной особенностью детектора SPD является отсутствие «классической» триггерной системы, позволяющей реализовать отбор собираемых для дальнейшего исследования событий. Это приводит к необходимости собирать с подсистем весь набор произведенных сигналов, объединенных в блоки по времени. Такой поток данных может составлять до 20 ГБ/секунду, что, при планируемых режимах работы ускорительного комплекса и детектора, будет составлять до 200 ПБ/год. С целью сокращения объема данных для долговременного хранения и последующего анализа планируется провести их первичную обработку с использованием специализированной вычислительной системы – SPD Online Filter.

Целью работы является концептуальная доработка и реализация системы управления данными, которая является частью вычислительного комплекса

SPD Online Filter и предназначена для контроля над организацией, хранением и целостностью данных. При этом необходимо обеспечить требуемый уровень надежности.

В рамках данной работы сформулированы следующие **задачи**:

- Ознакомиться с целями и задачами эксперимента SPD, а также с особенностью экспериментальной установки;
- Изучить требования к комплексу SPD Online Filter и принципы его работы;
- Реализовать фоновый сервис для контроля состояния данных на хранилище;
- Доработать взаимодействие системы управления данными с другими компонентами комплекса SPD Online Filter – системой управления процессами обработки и системой управления нагрузкой;
- Провести этап интеграционного тестирования между системами (выявить неточности в реализованном функционале и доработать его).

1 ОБЗОР УСКОРИТЕЛЬНОГО КОМПЛЕКСА NICA

На базе объединенного института идерных исследований (ОИЯИ) строится новый ускорительный комплекс NICA (Nuclotron based Ion Collider fAcility), направленный на изучение свойств сильного взаимодействия [2].

Рисунок 1.1 — Схема ускорительного комплекса NICA [3].

В первой точке взаимодействия коллайдера будет размещен детектор MPD (MultiPurpose Detector), на котором будет проведено исследование плотной барионной материи. Во второй точке взаимодействия будет установлен детектор SPD, который будет использоваться для изучения спиновой структуры протона и дейтрона и других связанных со спином явлений с помощью поляризованных пучков протонов и дейтронов (рис. 1.1). Также в числе экспериментальных установок присутствует BM@N (Baryonic Matter at Nuclotron), как и MPD, предназначенная для изучения плотной барионной материи, но уже на выведенных пучках Нуклотрона.

1.1 NICA: ЦЕЛИ И ЗАДАЧИ

Мегасайнс проект NICA нацелен на воссоздание и исследование ядерной материи в экстремальных условиях, возникавших в природе на ранних стадиях эволюции Вселенной и в недрах нейтронных звезд.

Основными задачами ускорительного комплекса NICA являются:

1) **Поиск критической точки на фазовом переходе между адронной материей и кварк-глюонной плазмой**.

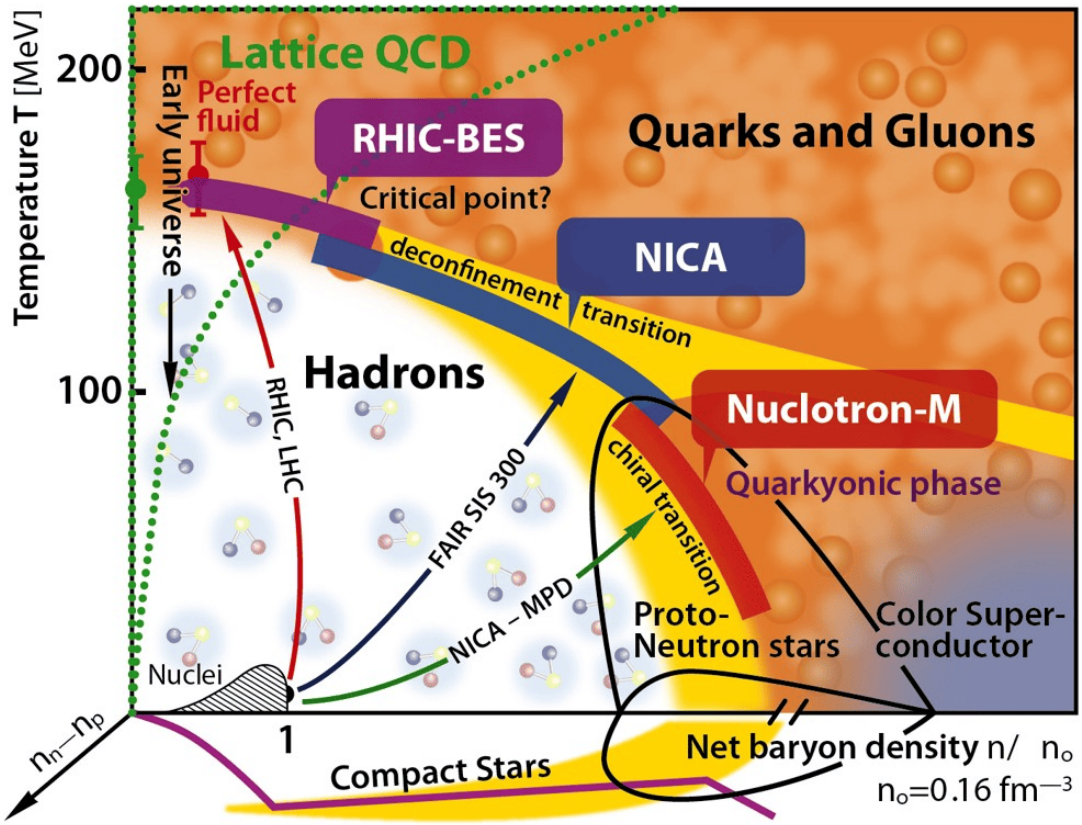


Рисунок 1.2 — Фазовая диаграмма адронного вещества.

Теория, описывающая сильные взаимодействия (квантовая хромодинамика – КХД), предсказывает для систем с высокой температурой и/или плотностью барионного заряда достижение деконфайнмента адронной материи и переход к состоянию с восстановленной киральной симметрией – кварк-глюонной плазме (рис. 1.2). Однако до сих пор остается открытым вопрос о типе кварк-адронного фазового перехода, достигаемого в соударениях тяжелых ионов.

Экспериментальные данные, полученные на коллайдере тяжелых ионов RHIC на LHC говорят скорее о фазовом переходе второго рода, – «крос-

совере», предсказываемом при высоких температурах и малом барионном химическом потенциале. Помимо плавного «кроссовера», недавние расчеты КХД на решетке указывают на возможность фазового перехода первого рода при низких температурах и высоком барионном химическом потенциале, а следовательно, и на существование «критической точки», где тип перехода меняется с первого рода на «кроссовер».

Ни фазовый переход первого рода, ни критическая точка не наблюдались экспериментально. Поэтому их поиск является одной из главных задач научной программы.

2) **Изучение спиновой структуры нуклона**.

Именно со спиновой структурой протона связана одна из наиболее важных нерешенных проблем физики элементарных частиц. Хорошо установленным фактом является то, что протон состоит из трех кварков. Казалось бы, что спин протона должен быть суммой спинов кварков. Но в 1988 году коллаборация EMC сообщила, что вклад спинов кварков, составляющих протон, составляет не более трети. Оставшийся вклад, по-видимому, приходится на спин глюонов и орбитальные моменты кварков и глюонов. Несмотря на интенсивную работу, проблема «спинового кризиса» до сих пор не решена. В связи с этим, исследование спиновой структуры нуклона является главной целью эксперимента SPD.

3) **Исследование поляризационных эффектов в столкновениях тяжелых ионов** [4].

Неисчезающая глобальная спиновая поляризация гиперонов, наблюдаемая коллаборациями STAR и HADES, поднимает новые теоретические вопросы. Такая поляризация может быть связана с характеристиками среды, такими как завихренность и гидродинамическая спиральность.

4) **Изучение модификации свойств адронов в ядерной материи**.

Дилептоны являются лучшими кандидатами для изучения модификаций спектральных функций адронов в среде при предполагаемом частичном восстановлении киральной симметрии в ядро-ядерных соударениях [5], поскольку в плотной адронной материи они взаимодействуют только электромагнитно и поэтому несут точную информацию о характеристиках среды в момент их рождения.

5) **Исследование рождения гиперонов и гиперядер** [6], [7].
Рождение странных частиц представляет большой интерес, поскольку их повышенный выход в ядро-ядерных столкновениях по сравнению с выходом в элементарных pp реакциях может служить указанием на образование кварк-глюонной плазмы. Изучение рождения гиперонов и гиперядер занимает одно из важных мест в физической программе эксперимента MPD на ускорительном комплексе NICA, а новые данные по выходам гиперонов при различных энергиях столкновения и размерах сталкивающихся систем позволят значительно расширить наши представления о динамике ядро-ядерных столкновений и свойствах плотной барионной материи.

1.2 ЭКСПЕРМЕНТ SPD

Глюоны, наряду с кварками, являются фундаментальными составляющими нуклона. Они играют ключевую роль в формировании массы нуклона и несут около половины его импульса. Спин нуклона также определяется его составляющими и складывается из спинов валентных и морских кварков, а также глюонов.

Несмотря на прогресс, достигнутый за последние десятилетия в понимании вклада кварков в спин нуклонов, который определен довольно точно в экспериментах по полуинклюзивному глубокому неупругому рассеянию, таких как EMC, HERMES и COMPASS, вклад глюонов все еще недостаточно хорошо изучен.

В связи с этим, основной целью SPD является изучение спиновой структуры протона и дейтрона и других связанных со спином явлений с помощью поляризованных пучков протонов и дейтронов со светимостью до 10^{32} см$^{-2} \cdot$ сек$^{-1}$ и при энергии столкновения до 27 ГэВ.

1.2.1 ИЗУЧАЕМЫЕ В ЭКСПЕРИМЕНТЕ ПРОЦЕССЫ

Спиновая структура нуклона в полуинклюзивных жестких процессах описывается так называемыми функциями распределения партонов, зависящими от поперечного импульса.

Распределение поляризованных глюонов в протоне и дейтроне будет исследоваться с помощью трех каналов (рис. 1.3): инклюзивного рождения чармониев (например, J/ψ), частиц с открытым очарованием (например, D-мезонов) и быстрых фотонов [8]. Изучение этих процессов дополняет обычные подходы к изучению партонной структуры нуклона при адронных столкновениях, такие как образование адронов с высоким поперечным импульсом и процесс Дрелла-Яна (процесс рождения лептонной пары при аннигиляции кварк-антикварковой пары в виртуальный фотон или Z-бозон).

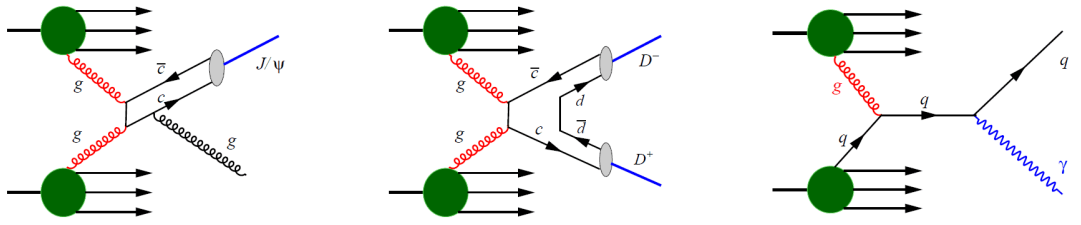


Рисунок 1.3 — Диаграммы, иллюстрирующие три канала для определения глюонной составляющей в протонах и дейтронах при поляризованных столкновениях в NICA SPD: рождение (a) чармония, (b) открытого очарования и (c) быстрых фотонов.

1.2.2 ЗАДАЧИ ЭКСПЕРИМЕНТА SPD

В задачи эксперимента входит:

1) На первой стадии:
   (а) Исследование спиновых эффектов в упругом pp и dd рассеянии и в образовании гиперонов;
   (б) Изучение многокварковых корреляций;
   (в) Изучение реакций с образованием дибарионов и странных гиперядер;
   (г) Образование чармония близко к пороговому значению.
2) На второй стадии эксперимента основное внимание уделяется исследованию распределения глюонов в реакциях рождения чармония, открытого очарования и быстрых фотонов.

1.2.3 УСТАНОВКА SPD

Сам детектор SPD (рис. 1.4) задуман как универсальный 4π-спектрометр с расширенными возможностями отслеживания и идентификации частиц, основанный на современных технологиях. Для эффективного обнаружения вышеупомянутых процессов установку SPD планируется оснастить мюонной системой (для эффективного разделения мюонов и адронов), электромагнитным калориметром (для обнаружения сигнальных и фоновых фотонов), времяпролетной системой (для идентификации частиц), straw-трекером и кремниевым вершинным детектором (для восстановления вторичных вершин при распадах D-мезонов и других короткоживущих частиц).

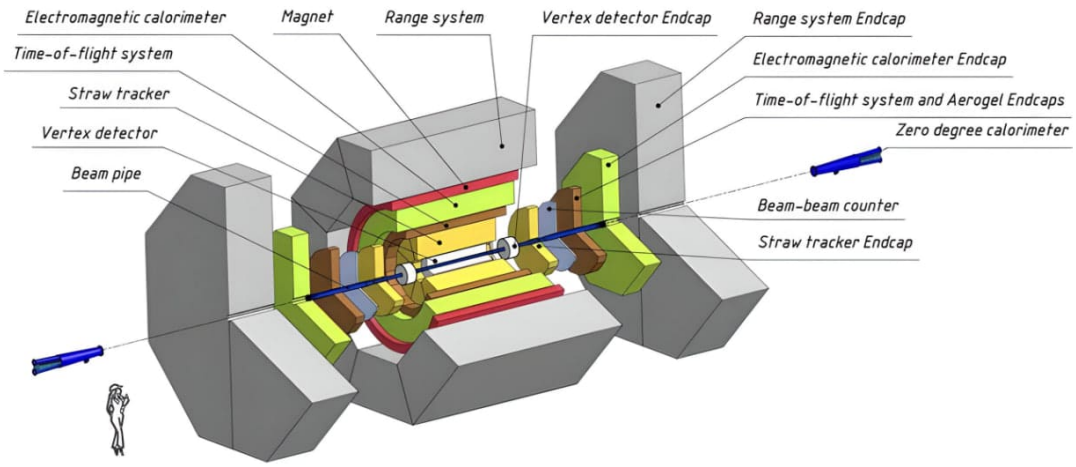


Рисунок 1.4 — Экспериментальная установка SPD [9].

1.2.4 СБОР ДАННЫХ С ДЕТЕКТОРА

С помощью детектора происходит регистрация «событий». Все они независимы и могут обрабатываться по отдельности. На первичном этапе событие представляет собой набор сигналов от чувствительных элементов детектора (сенсоров), передаваемых по каналам считывания сигналов. Количество таких сигналов считывания у эксперимента SPD ~ 500000.

Особенностью детектора SPD является отсутствие классического триггера, ввиду того, что на аппаратном уровне невозможен простой выбор физических событий, поскольку решение о записи того или иного события будет зависеть от измерения импульса и положения вершины. Это означает, что данные с

детектора будут считываться непрерывно и, как следствие, возникнет большой объем принимаемой информации.

Таким образом, система сбора данных является «безтриггерной» в том смысле, что в ней нет глобального аппаратного триггера. Вместо этого каждое событие помечается «временной меткой», которая используется при постобработке для восстановления событий и отбрасывания ложных срабатываний.

Важно отметить, что SPD – не первый детектор, который будет работать в «безтриггерной» моде. Так, в 2004 г. в эксперименте MINOS по изучению осцилляций нейтрино для детектора «Near Detector» была использована безтриггерная система сбора данных [10]. Также в экспериментах LHCb [11; 12] и ALICE [13] отказались от аппаратного триггера в пользу непрерывного считывания показаний детектора с последующим программным триггером в рамках модернизации ускорителя LHC для выполнения программы «Run 3».

Для реализации безтриггерной системы нам требуется специальный вычислительный комплекс **SPD Online Filter**, который осуществлял бы быструю частичную реконструкцию и отбор интересующих нас событий, что в сущности представляет собой программный триггер.

2 SPD ONLINE FILTER

SPD Online Filter – это высокопроизводительная вычислительная система для высокопропускной обработки данных.

Особенностью высокопропускной обработки данных является большой объем данных, как первичных, которые необходимо обработать, так и промежуточных, возникающих в процессе обработки. Основная цель – существенное сокращение объема данных для долговременного хранения, последующей обработки и анализа.

Основная цель онлайн-фильтра – восстановить события из непрерывного потока байтов и минимум в 20 раз сократить объем данных [14]. В качестве входных данных система получает файлы «сырых» данных, формат которых определяется электроникой и системой DAQ (Data Acquisition System). Результатом обработки данных является набор реконструированных событий, содержащий также исходную информацию.

2.1 ТРЕБОВАНИЯ К SPD ONLINE FILTER

Основной процесс первичной обработки данных включает в себя минимум три этапа:

- Декодирование данных, полученных от DAQ;
- Частичная реконструкция событий;
- Фильтрация событий в соответствии с заданными физическими критериями.

Для выполнения этих задач система SPD Online Filter должна включать в себя:

1) Гетерогенный высокопроизводительный вычислительный кластер;
2) Комплекс промежуточного программного обеспечения для многоступенчатой обработки данных;

3) Прикладное программное обеспечение.

Вычислительный кластер представляет собой аппаратную часть комплекса SPD Online Filter и состоит из высокоскоростного хранилища входных данных, буфера для промежуточных данных и данных, подготовленных к передаче в долгосрочное хранилище, совокупности многоядерных вычислительных узлов и набора управляющих серверов для размещения служб промежуточного программного обеспечения (рис. 2.1).

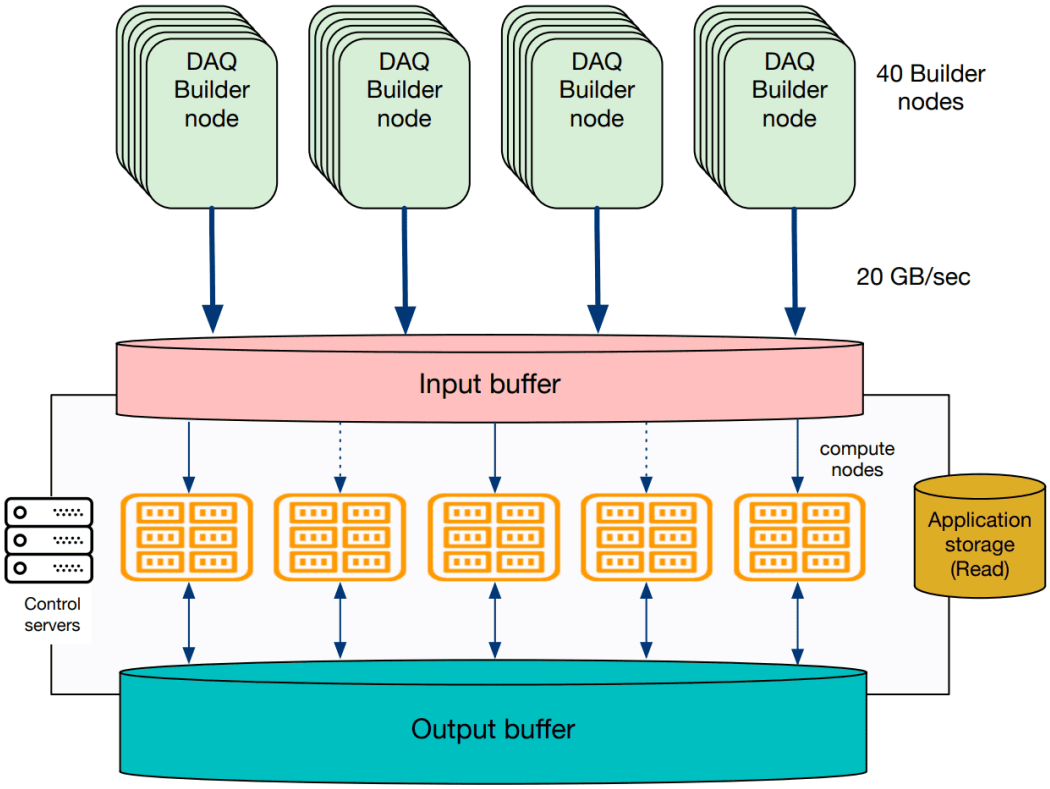


Рисунок 2.1 — Вычислительный кластер SPD Online Filter.

2.2 ОРГАНИЗАЦИЯ И ОБРАБОТКА ДАННЫХ С ДЕТЕКТОРА

Входные данные приходят из DAQ во входной буфер. Система сбора данных группирует данные с сенсоров следующим образом (рис. 2.2):

1) *Период набора* – ассоциирован с физической задачей. Состоит из набора ранов (run). Измеряется неделями. Имеет дату начала, дату окончания;
2) *Ран* – интервал, когда условия эксперимента неизменны. Измеряется ча-

сами и состоит из фреймов;

3) *Фрейм* – нумерованный блок данных с сенсоров. Продолжительность – секунды. По объему делится на файлы;
4) *Файл* – количество данных для обработки одной задачей. Каждый из файлов содержит данные за достаточно длительный период. В одном файле содержится информация о множестве событий.

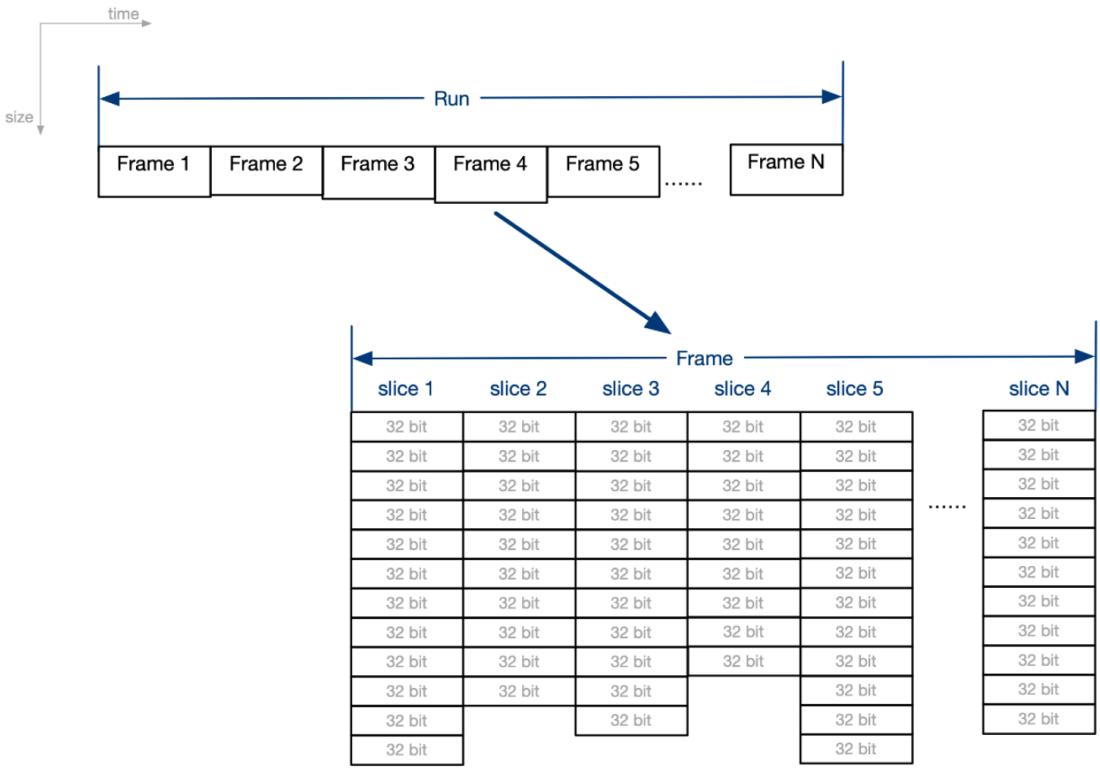


Рисунок 2.2 — Организация входных данных с DAQ [15].

Промежуточные данные – это данные, образующиеся в процессе обработки, но еще не готовые для использования в системах оффлайн обработки. В их организацию входит:

- привязка к фрейму;
- привязка к набору для последующего шага в процессе обработки.

Промежуточные данные живут только в рамках системы, не подразумевается их долговременное хранение, соответственно, они должны регулярно удаляться.

Выходные данные – данные для последующей обработки и использования вне функционала онлайн фильтра. К их организации относится:

- привязка к периоду набора и рану;
- логическая группировка в наборы.

Обработка входных данных

Каждый файл обрабатывается отдельно и независимо. Файл разбит на множество временных срезов (рис. 2.3). Для каждого такого среза:

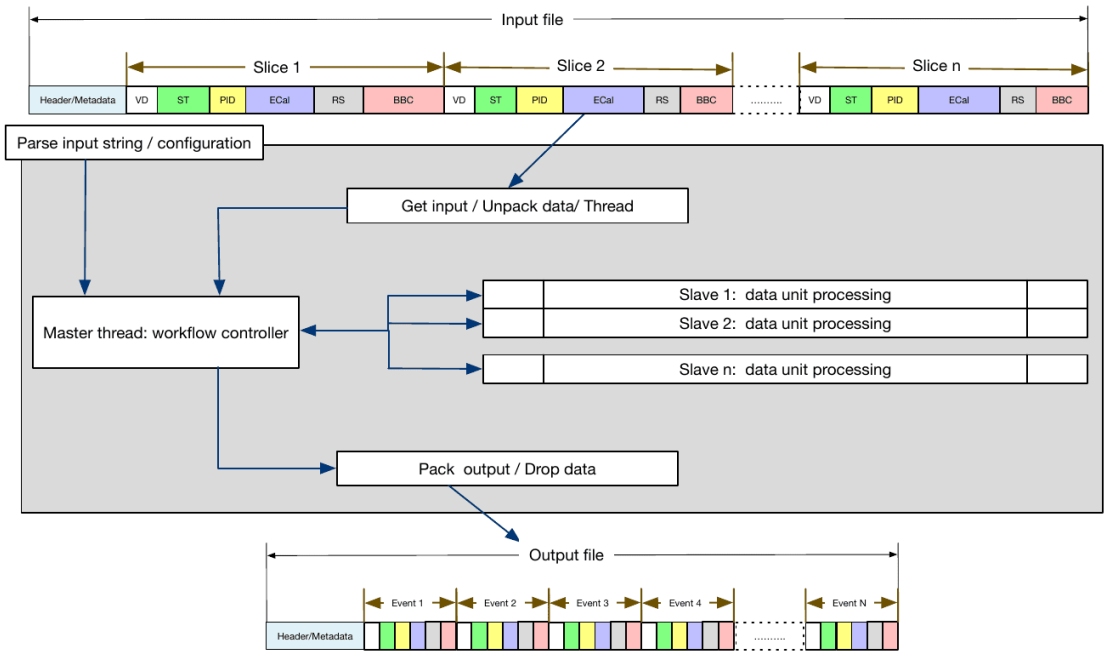


Рисунок 2.3 — Преобразование входного файла в выходной.

1) Реконструируем треки и связываем их с вершинами;
2) Определяем время пересечения пучков для каждой вершины;
3) Связываем срабатывания ECal и RS с каждой вершиной (по временной метке);
4) Присоединяем несвязанные треки в выбранном временном окне в соответствии со временем пересечения пучков;
5) Присоединяем необработанные данные с других поддетекторов в соответствии со временем пересечения пучков;
6) Блок информации, связанный с каждой вершиной, называем событием;
7) Сохраняем восстановленные события.

2.3 АРХИТЕКТУРА ПРОМЕЖУТОЧНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Каждый из этапов первичной обработки можно разделить на атомарные задачи, которые работают с однородными наборами данных, причем обработка

каждого отдельного файла может выполняться независимо, что подразумевает под собой многоступенчатую обработку.

Для автоматизации рабочего процесса необходимо промежуточное программное обеспечение (рис. 2.4), которое включает в себя следующие компоненты:

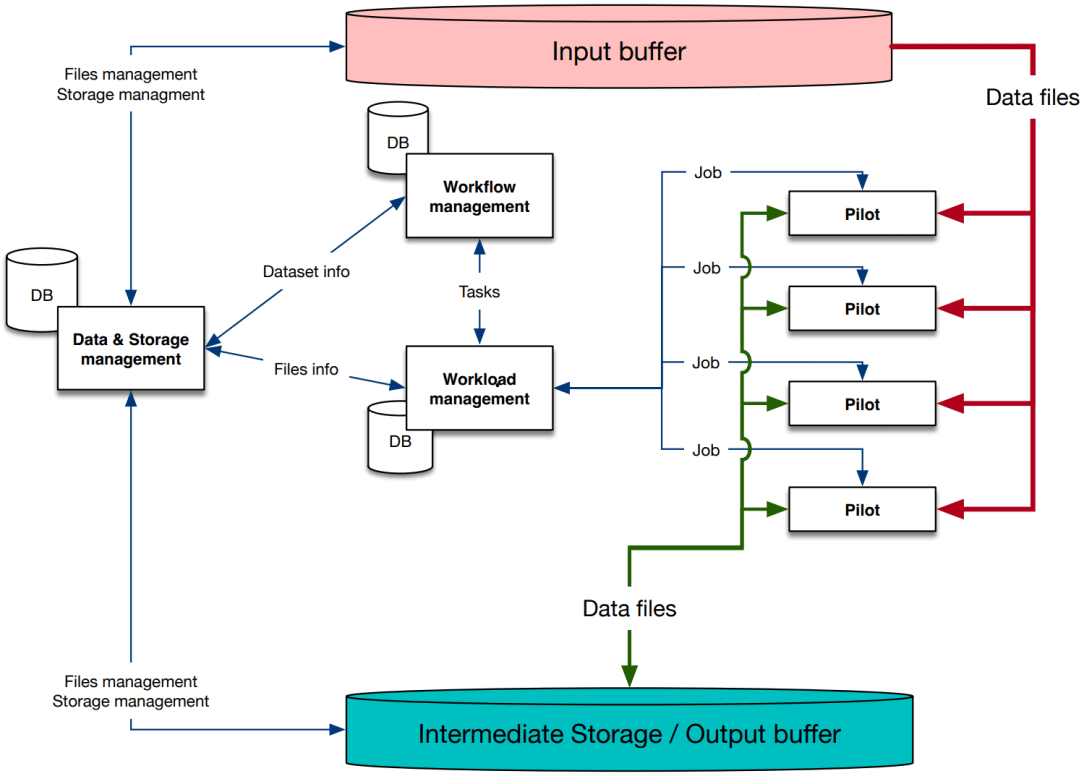


Рисунок 2.4 — Архитектура промежуточного ПО.

– *Система управления данными* (регистрация новых данных, каталогизация, контроль целостности и согласованности);
– *Система управления процессами обработки* (формирование и контроль исполнения этапов обработки данных);
– *Система управления нагрузкой* (реализация этапов обработки путем формирования и выполнения необходимого количества задач для обработки набора данных);
    * *Pilot* (агентское приложение, работающее на вычислительном узле и исполняющее задачи, поставляемые от системы управления нагрузкой).

Система управления данными. В процессе первичной обработки необходимо оперировать как файлами, так и их логическими объединениями (датасетами), причем датасеты могут быть входными (получены от системы сбора

данных), промежуточными (сгенерированы во время обработки) и выходными (данные для последующей обработки и анализа).

В итоге в системе возникает достаточно большой объем данных ввиду следующих процессов:

- DAQ передает данные на входное хранилище в виде файлов согласованного размера;
- Система управления процессами обработки создает и удаляет датасеты;
- Pilot-ы создают вторичные файлы на каждом шаге обработки;
- Система управления нагрузкой регистрирует вторичные файлы в датасетах.

Таким образом, необходима система, которая бы обеспечивала следующие функции:

- Возможность логической группировки файлов;
- Отслеживание жизненного цикла данных;
- Отделение метаданных от физического хранения файлов;
- Контроль согласованности информации в каталоге по отношению к хранилищу данных.

3 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

В рамках эксперимента ALICE была разработана новая online-offline вычислительная система, получившая название O^2 [16]. Она характеризуется непрерывным считыванием всех данных, при этом сокращение объема данных будет осуществляться посредством частичной реконструкции и калибровки «на лету» параллельно со сбором данных (рис. 3.1).

Таким образом, на диск записываются только реконструированные данные, а исходные данные удаляются. Такой механизм привел к очень тесной интеграции с DAQ, что не подходит для проекта SPD Online Filter. По схожему принципу работает и программный триггер в эксперименте LHCb [17].

Одной из самых известных программных платформ для управления данными на данный момент является Rucio [18], которая изначально разрабатывалась в соответствии с требованиями эксперимента ATLAS. Она была создана как комплексное решение для организации, управления и доступа к данным. Возможности Rucio заключаются в следующем (рис. 3.2):

- Хранение всех экспериментальных данных в распределенной среде;
- Поддержка новых сетевых протоколов и интерфейсов к хранилищам при помощ плагинов;
- Восстановление данных;
- Адаптивная репликация;
- Логическое объединение файлов в наборы;
- Авторизация и аутентификация пользователей.

Несмотря на то, что Rucio может обеспечить необходимый нам функционал, важно заметить, что возможности данной платформы этим не ограничиваются. Более половины функций использоваться не будет, а оставшаяся часть довольно специфична, что означает сложность ее встраивания. В связи с этим Rucio так же не подходит для наших целей.

Остальные решения не подлежат рассмотрению в связи с плохой документацией и отсутствием программного кода в открытом доступе.

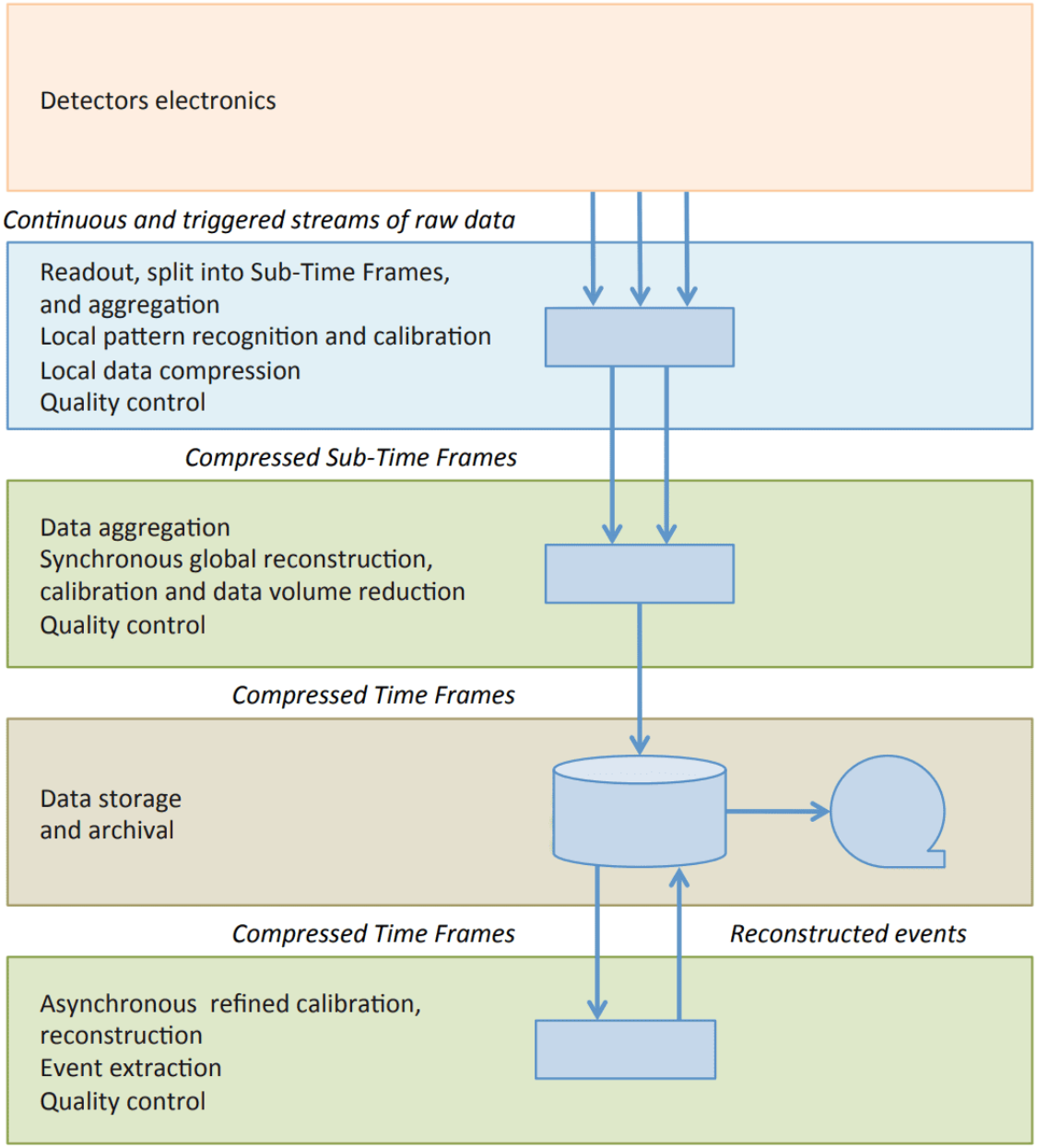


Рисунок 3.1 — Вычислительная система O^2.

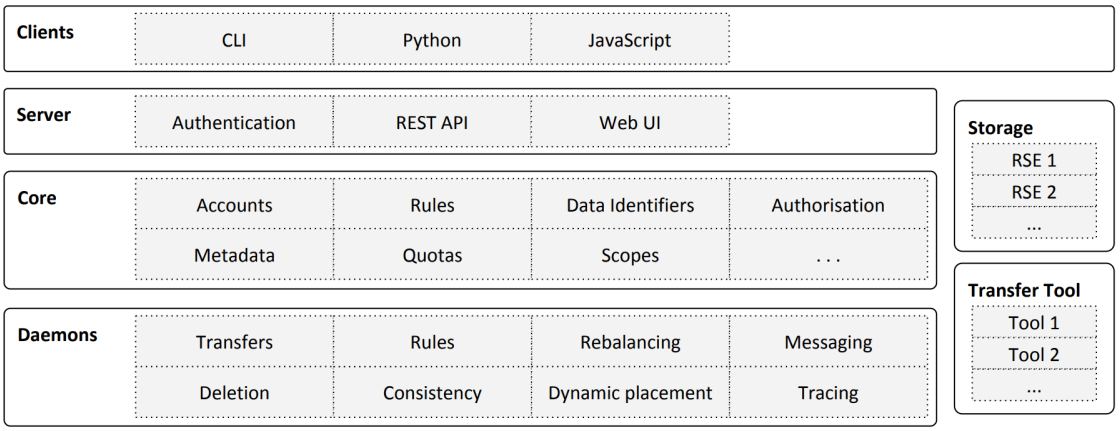


Рисунок 3.2 — Основные компоненты Rucio.

4 ОСНОВНЫЕ КОМПОНЕНТЫ СИСТЕМЫ УПРАВЛЕНИЯ ДАННЫМИ

Так как в процессе первичной обработки данных образуется большой объем вторичных данных, то нам необходима специальная система для управления этими данными. Такой системой является система управления данными.

4.1 ТРЕБОВАНИЯ К СИСТЕМЕ

Система должна обеспечивать регистрацию новых данных, каталогизацию и контроль согласованности каталога по отношению к хранилищу.

Для того чтобы система могла узнать о существовании файла на хранилище, необходим **интерфейс для регистрации** файлов, который включает в себя получение информации о местоположении файла (имя, физический путь, метаданные) и внесение его в каталог с привязкой к необходимому датасету.

Для осуществления каталогизации данных нам необходима **база данных**, в которой должна храниться информация о файлах и наборах. Помимо этого должен быть **интерфейс к каталогу** данных, через который можно размещать/удалять информацию о файлах и управлять датасетами.

Для корректного функционирования всей системы требуются **фоновые сервисы**, которые бы обеспечивали согласованность хранилища и каталога. Сюда входит контроль целостности и выгрузки файлов, удаление данных и мониторинг хранилища.

4.2 ПРИМЕНИМОСТЬ МИКОСЕРВИСНОЙ АРХИТЕКТУРЫ

Если мы хотим, чтобы система обеспечивала хорошую производительность при высокой нагрузке, необходимо следовать принципам микросервисной архитектуры при ее разработке. Согласно данному подходу, мы разбиваем большое приложение на независимые сервисы, каждый из которых выполняет отдельную функцию [19].

Помимо производительности, у микросервисного приложения есть еще ряд достоинств, которые выделяют его на фоне монолитной архитектуры: возможность использования большого диапазона технологий, гибкость разработки, масштабируемость и отказоустойчивость [20]. Использование такой архитектуры позволит легко развивать и обновлять все приложение, но для начала его нужно грамотно разбить на набор сервисов.

Декомпозиция большой системы на микросервисы может осуществляться по двум шаблонам: «разбиение по бизнес-возможностям» и «разбиение по поддоменам» [21]. Здесь в качестве стратегии разбиения используется последний шаблон, который основан на концепциях предметно-ориентированного проектирования (Domain-Driven Design, DDD).

Согласно выбранному шаблону, мы разбиваем всю предметную область (домен) на поддомены. У каждого поддомена своя модель данных, которая имеет четкие границы и действует только внутри ограниченного контекста. Основная задача при использовании DDD-подхода – подобрать поддомены и границы между ними так, чтобы они были максимально независимы друг от друга.

Помимо декомпозиции необходимо правильно определить механизмы, по которым сервисы будут взаимодействовать друг с другом. Это взаимодействие может быть как синхронным, когда один сервис ожидает ответа от другого, так и асинхронным, когда сервис отправляет сообщение, не ожидая немедленного ответа.

В тех случаях, где требуется немедленный ответ от сервиса, используется REST API (сервисы взаимодействуют по HTTP-протоколу) [22]. В остальных случаях общение между сервисами будет происходить асинхронно с помощью брокера сообщений.

4.3 АРХИТЕКТУРА

Таким образом, исходя из принципов микросервисной архитектуры, система управления данными была разбита на три микросервиса (рис. 4.1):

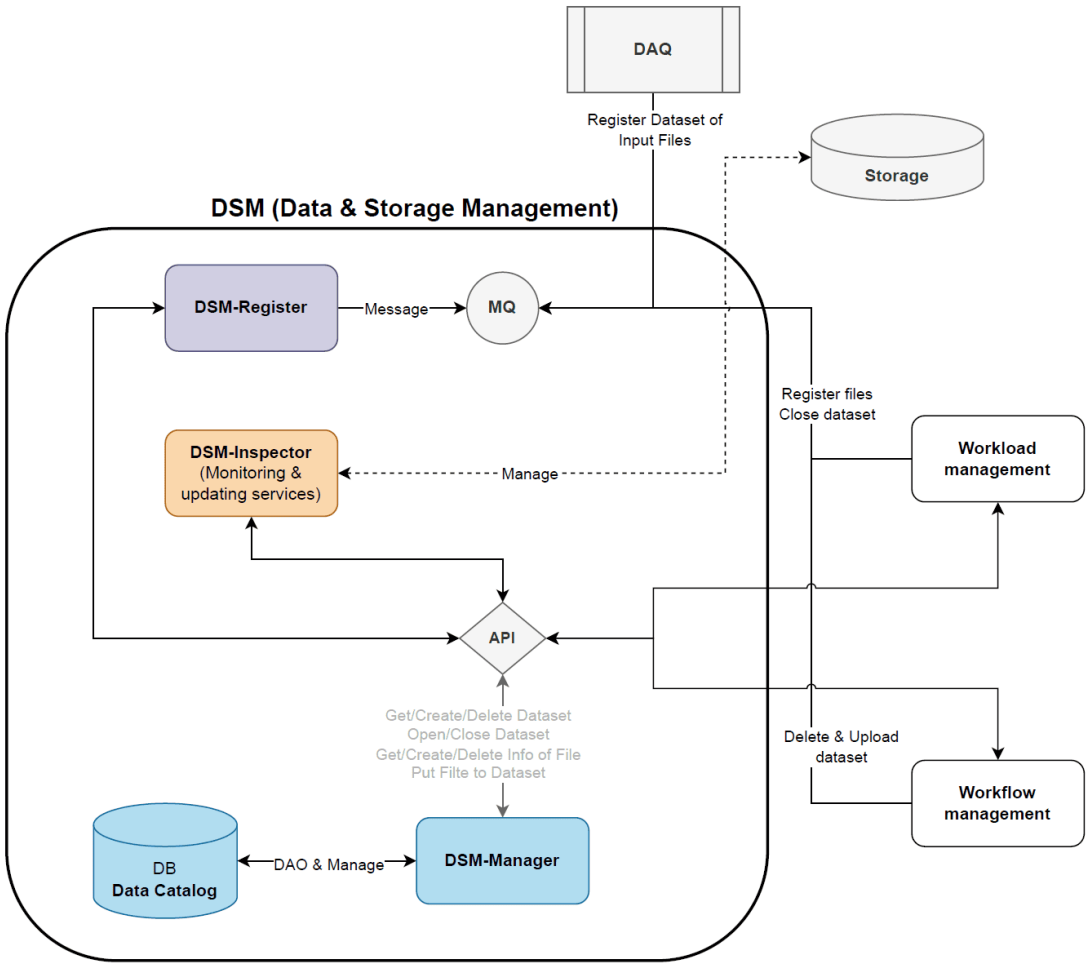


Рисунок 4.1 — Архитектура системы управления данными.

– **dsm-register**. Сервис, принимающий в асинхронном режиме (через очередь сообщений) заявки на добавление/удаление данных в системе. От DAQ он получает информацию о входных файлах, которые необходимо зарегистрировать, от системы управления нагрузкой – о промежуточных и выходных, а система управления процессами обработки отправляет заявки на удаление/выгрузку данных. При обработке заявок dsm-register вносит изменения в каталог данных через API сервиса dsm-manager;
– **dsm-manager**. Сервис, предоставляющий REST API к каталогу данных (размещение данных в каталоге, обращение к каталогу, изменение данных в каталоге). Он нужен как для внутреннего функционирования системы,

так и для внешнего взаимодействия (система управления процессами обработки получает из каталога инфомацию о датасетах, система управления нагрузкой – о содержимом датасета);

– **dsm-inspector**. Набор фоновых сервисов для мониторинга и контроля состояния данных в хранилище (проверка целостности файлов, контроль использования хранилища, удаление и выгрузка файлов на хранилищах).

4.4 ВЗАИМОДЕЙСТВИЕ С DAQ

DAQ – система сбора данных, которая размещает первичные файлы на входном буфере. В рамках взаимодействия DSM (Data & Storage Management) с DAQ существует только один механизм:

– Регистрация входных файлов. Через очередь сообщений DAQ отправляет информацию о новых файлах в хранилище:
  1) meta – метаданные к файлам (номер рана);
  2) files – информация о файлах:
     (а) имя файла;
     (б) путь до файла на хранилище;
     (в) размер файла;
     (г) контрольная сумма файла.

4.5 ВЗАИМОДЕЙСТВИЕ С WFMS

WFMS (WorkFlow Management System) отвечает за формирование заданий на обработку и контроль их выполнения. Процесс взаимодействия между системами можно описать с помощью следующих сценариев:

1) Информирование WFMS о создании входного датасета. После создания нового набора информация о нем отправляется в соответствующую очередь. В качестве передаваемых данных:
   (а) идентификатор входного датасета;
   (б) имя датасета.
2) Получение от WFMS заявки на удаление датасета. Через очередь получаем сообщение с id датасета, подлежащего удалению;
3) Получение заявки на выгрузку датасета. Через очередь получаем сообще-

ние с id датасета, который необходимо отправить на выгрузку;

4) Создание нового промежуточного/выходного датасета. Система управления процессами обработки создает новый датасет через HTTP POST-запрос:
   (а) Получаемые данные:
      i. имя датасета;
      ii. метаданные;
      iii. статус датасета.
   (б) Передаваемые данные:
      i. имя датасета;
      ii. метаданные;
      iii. статус датасета;
      iv. идентификатор датасета.
5) Закрытие датасета. WFMS отправляет нам PATCH запрос на изменение статуса датасета:
   (а) Получаемые данные:
      i. идентификатор датасета;
      ii. статус датасета.
   (б) Передаваемые данные:
      i. аналогично созданию нового датасета.

4.6 ВЗАИМОДЕЙСТВИЕ С WMS

WMS (Workload Management System) формирует и исполняет задачи для обработки данных. Взаимодействие системы управления данными с данной системой заключается в следующем:

1) Получение от WMS заявки на регистрацию файлов в датасете через очередь сообщений:
   (а) Получаемые данные:
      i. идентификатор датасета;
      ii. информация о файлах:
         A. идентификатор хранилища;
         B. путь до файла на хранилище (включая имя файла);
         C. размер файла;

D. контрольная сумма.

(б) Передаваемые данные:

i. статус регистрации файла;

ii. детали регистрации.

2) Предоставление информации о хранилище по его типу через GET-запрос:

(а) Получаемые данные:

i. тип хранилища.

(б) Передаваемые данные:

i. URL хранилища;

ii. путь;

iii. размер хранилища;

iv. размер занятого места;

v. тип хранилища;

vi. идентификатор хранилища.

3) Предоставление информации о содержимом датасета по его id через GET-запрос:

(а) Получаемые данные:

i. идентификатор датасета;

(б) Передаваемые данные:

i. информация о файлах в наборе:

A. имя файла;

B. путь до файла на храниище;

C. идентификатор хранилища;

D. размер файла;

E. контролная сумма;

F. статус файла;

G. идентификатор файла;

H. URL файла.

5 ТРЕБОВАНИЯ К СИСТЕМЕ УПРАВЛЕНИЯ ДАННЫМИ

5.1 ТРЕБОВАНИЕ К БАЗЕ ДАННЫХ

Исходя из таких критериев, как качество поддержки, функциональность, широкий круг пользователей, доступность и производительность, в качестве СУБД была выбрана PostgreSQL 12 [23].

5.1.1 КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ ДАННЫХ

Было необходимо, чтобы в базе данных хранилась информация о файлах, их логических объединениях (датасетах), доступных хранилищах, а также о так называемых «темных данных» (файлы, которые существуют на хранилище, но их нет в базе данных). Помимо этого, нам требуется отслеживать жизненный цикл данных, из чего следует хранение информации о статусах файлов и датасетов.

В связи с вышеперечисленными требованиями была предложена ER-диаграмма представленная на рисунке 5.1. Сюда вошел следующий набор таблиц:

- *DAT_FILE* – каталог файлов, обрабатываемых системой;
- *DAT_DARK_FILE* – каталог темных файлов, не зафиксированных в каталоге файлов;
- *DAT_DATASET* – каталог датасетов, созданных системой;
- *DAT_FILE_HISTORY* и *DAT_DATASET_HISTORY* – архивные таблицы по файлам и датасетам;
- *DAT_STORAGE* – информация о хранилищах;
- *DIC_FILE_STATUS* и *DIC_DATASET_STATUS* – справочники, хранящие возможные статусы по файлам и датасетам соответственно.

Каталог файлов является ключевой таблицей в базе данных. Состав ее

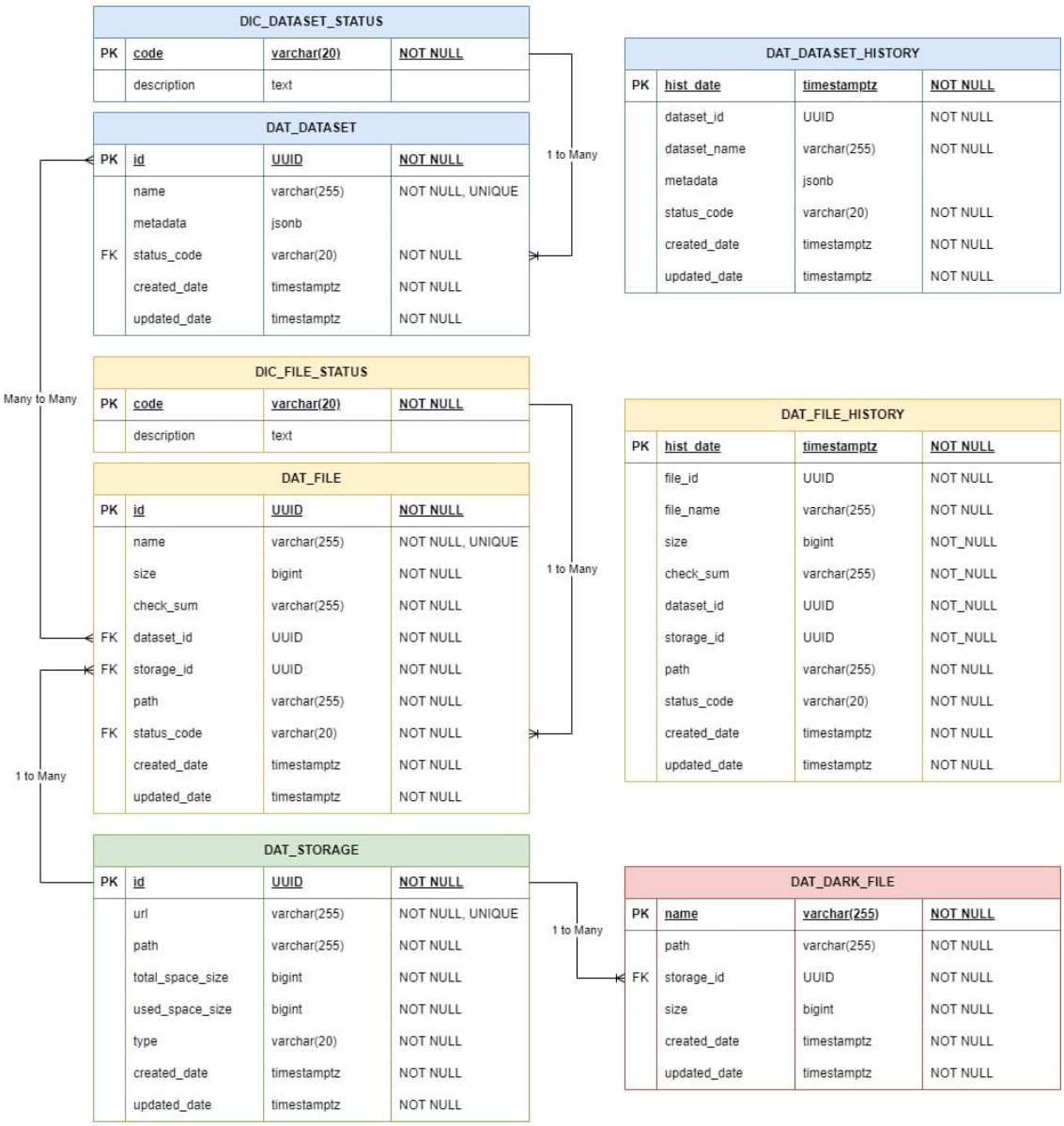


Рисунок 5.1 — Концептуальная схема базы данных.

атрибутов выбран таким образом, чтобы можно было обеспечить следующие механизмы:

1) Идентификация файла на хранилище (имя файла, путь на хранилище, идентификатор хранилища);
2) Контроль целостности файла (размер файла и его контрольная сумма);
3) Фиксация жизненного цикла файла (статус файла).

Причем файл может принадлежать одному или нескольким датасетам. Это осуществляется с помощью специальной таблицы, которая реализует отношение «многие ко многим» [24]. Она хранит информацию о том, как соотносятся идентификаторы датасетов и файлов.

Каталог датасетов служит для хранения логических объединений фай-

лов, не релевантных к физическому расположению. Имя каждого датасета уникально и определяется логикой группировки. Также здесь может быть размещена дополнительная мета-информация о датасете (например, в случае входного набора файлов, поступающих с DAQ, это может быть информация о временном срезе данных: номер фрейма, номер рана и т.п.). Для отслеживания жизненного цикла, как и в случае с файлами, в таблице имеется поле со статусом.

Каталог темных файлов необходим для отслеживания данных, которые по какой-либо причине не были зарегистрированы в системе. Такие файлы не объединяются в датасеты и не имеют жизненного цикла. Данная таблица служит для временного хранения информации о файле с возможностью его идентификации на хранилище.

Хранилища помимо адреса могут характеризоваться также дополнительной важной информацией:

1) Тип хранилища (входное, промежуточное, выходное);
2) Полный путь до хранилища (внешний и внутренний адрес);
3) Состояние хранилища (размер всего дискового пространства и уже занятого).

Отдельные **справочные таблицы** предназначены для строгого контроля возможных статусов файлов и датасетов (см. таблицы 5.1, 5.2).

Код статуса	**Описание**
CREATED	Файл добавлен в систему
DAMAGED	Файл поврежден
TO_DELETE	Файл с пометкой «на удалении»
UPLOADING	Файл выгружается
DELETED	Файл удален из системы

Таблица 5.1 — Справочник статусов файла (DIC_FILE_STATUS)

Так как изменения в таблице файлов и датасетов могут происходить достаточно часто, были введены дополнительные **history таблицы** для фиксации предыдущих состояний данных. Это полезно для последующего анализа корректности работы системы. Заполнение такого рода таблиц предполагается осуществлять автоматически при помощи триггера.

Код статуса	Описание
OPEN	Набор открыт
CLOSED	Набор закрыт
FROZEN	Набор временно «заморожен»
TO_UPLOAD	Набор с пометкой «на выгрузку»
UPLOADING	Набор выгружается
TO_DELETE	Набор с пометкой «на удалении»
DELETED	Набор удален из системы

Таблица 5.2 — Справочник статусов датасета (DIC_DATASET_STATUS)

5.1.2 ДОПОЛНИТЕЛЬНЫЕ МЕХАНИЗМЫ

Как уже упоминалось выше, заполнение history таблиц будет осуществляться с помощью **триггера**. Триггер – функция, запускающаяся автоматически при возникновении определенного события в таблице (вставка, изменение, удаление) [25]. Триггер может выполняться до, после или вместо события, его вызвавшего.

В нашем случае триггер должен срабатывать при любой операции на изменение данных в таблицах DAT_FILE и DAT_DATASET. После внесения изменений в одну из данных таблиц данное состояние сохраняется в соответствующей history таблице.

Помимо этого, нам необходимо предусмотреть **партиционирование** history таблиц для оптимизации операций на чтение и вставку, так как при увеличении размера таблиц скорость запросов будет неизбежно падать. Партиционирование представляет собой логическое разделение большой таблицы на небольшие части, называемые партициями [26]. Принадлежность каждой новой записи таблицы к той или иной партиции определяется на основе значения ключа партиционирования.

Здесь в качестве ключа партиционирования должен выступать столбец hist_date, а размер партиции стоит задавать равным месяцу. Таким образом, при указании ключа партиционирования операции на чтение и запись будут осуществляться в рамках одной партиции, что повысит их эффективность.

5.2 ТРЕБОВАНИЯ К СЕРВИСАМ

5.2.1 СЕРВИС DSM-MANAGER

В таблице 5.3 представлен перечень REST API к каталогу данных в системе, который должен предоставлять сервис dsm-manager. Часть из них нужна для внутреннего функционирования системы управления данными, остальная часть – для внешнего взаимодействия.

Функциональность	**URL**	**Контракт запроса**	**Контракт ответа**
	Внутреннее API		
Управление информацией о наборах в каталоге			
Создать набор	POST /dataset	Информация о наборе	ID набора
Получить набор	GET /dataset/<id>	ID набора	Информация о наборе
Изменить набор	PUT /dataset/<id>	ID набора и изменения	Обновленная информация о наборе
Удалить набор	DELETE /dataset/<id>	ID набора	-
Получить список наборов	GET /dataset	-	Информация о наборах
Управление информацией о файлах в каталоге			
Добавить файл	POST /file	Информация о файле и принадлежность к набору	ID файла
Получить файл	GET /file/<id>	ID файла	Информация о файле
Изменить файл	PUT /file/<id>	ID файла и изменения	Обновленная информация о файле
Удалить файл	DELETE /file/<id>	ID файла	-
Получить список файлов	GET /file	-	Информация о файлах
Управление информацией о темных файлах в каталоге			
Добавить файл	POST /dark_file	Информация о файле	Информация о файле
Получить файл	GET /dark_file/<name>	Имя файла	Информация о файле
Удалить файл	DELETE /dark_file/<name>	Имя файла	-
Получить список файлов	GET /dark_file	-	Информация о файлах
Управление информацией о хранилище			
Добавить хранилище	POST /storage	Информация о хранилище	ID хранилища
Получить хранилище	GET /storage/<id>	ID хранилища	Информация о хранилище
Изменить хранилище	PUT /storage/<id>	ID хранилища и изменения	Обновленная информация о хранилище
Удалить хранилище	DELETE /storage/<id>	ID хранилища	-
Получить список хранилищ	GET /storage	-	Информация о хранилищах
Получение информации для мониторинга			
Получение списка наборов, к которым принадлежит файл	GET /dataset/?file_id=<id>	ID файла	Информация о наборах
Список наборов в определенном статусе	GET /dataset/?status=<>	Статус набора	Информация о наборах
Список файлов в определенном статусе	GET /file/?status=<>	Статус файла	Информация о файлах
Поиск информации о файле по имени	GET /file/file_name/<name>	Имя файла	Информация о файле
	Внешнее API		
Взаимодействие с системой управления нагрузкой			
Содержание набора	GET /file/?dataset_id=<id>	ID набора	Информация о файлах
Получение информации о наборе по имени	GET /dataset/name/<name>	Имя набора	Информация о наборе
Получение информации о хранилище по типу	GET /storage/type/<type>	Тип хранилища	Информация о хранилище
Взаимодействие с системой управления процессами обработки			
Некоторые API управления наборами (создание выходного набора, получение статуса выходного набора)			
Измененить статус набора	PATCH /dataset/<id>	ID набора и статус	Информация о наборе

Таблица 5.3 — Описание REST API сервиса dsm-manager

5.2.2 СЕРВИС DSM-REGISTER

Сервис должен слушать очередь сообщений и обрабатывать заявки на добавление/удаление данных в системе. В таблице 5.4 представлены шлюзы приема сообщений. В качестве AMQP-брокера, который осуществляет маршрутизацию и подписку на нужные очереди, используется RabbitMQ [27].

Exchange	**Routing Key**	**Назначение**
dsm.register (direct)	file.input	Прием информации о поступивших файлах на входной буфер
	file.process	Прием информации о новых файлах, полученных в процессе обработки
	dataset.upload	Прием заявки на выгрузку файлов в наборе во внешнее хранилище
	dataset.delete	Прием заявки на удаление файлов в наборе на внутреннем хранилище

Таблица 5.4 — Перечень шлюзов приема сообщений сервиса dsm-register

Шлюз **dsm.register.file.input** принимает сообщения вида: путь к файлу на хранилище, имя файла, его размер и контрольная сумма. Далее выполняются следующие шаги:

1) Заводится входной набор со статусом OPEN;
2) Регистрируются файлы с привязкой к этому набору. Устанавливается первичный статус CREATED.

Шлюз **dsm.register.file.process** принимает сообщения вида: идентификатор набора, информация о файлах в наборе (путь к файлу на хранилище, включая имя файла, идентификатор хранилища, размер файла и его контрольная сумма). Далее происходит регистрация новых файлов (в статусе CREATED) с привязкой к указанному набору.

Шлюзы **dsm.register.dataset.upload** и **dsm.register.dataset.delete** принимают только идентификатор набора. Далее, выполняются следующие шаги:

1) Запрашивается текущий статус набора;
2) Если набор находится в статусе OPEN, то сообщение возвращается обратно в очередь для отложенного исполнения;
3) Иначе устанавливается статус TO_UPLOAD и TO_DELETE соответственно.

Также после регистрации первичных и вторичных файлов необходимо дополнительно отправлять сообщения в соответствующие очереди для информирования других подсистем. Шлюзы отправки сообщений представлены в таблице 5.5.

Exchange	**Routing Key**	**Назначение**
dsm.register (direct)	file.process.reply	Отправка информации о статусе регистрации файлов, полученных в процессе обработки
	dataset.input	Отправка информации о входных наборах

Таблица 5.5 — Перечень шлюзов отправки сообщений сервиса dsm-register

Шлюз **dsm.register.file.process.reply** отправляет сообщения вида: статус регистрации файла (SUCCESS или ERROR), детали регистрации (если SUCCESS – полное имя зарегистрированного файла, если ERROR – полное имя файла и тип возникшей ошибки).

Шлюз **dsm.register.dataset.input** отправляет сообщения с информацией о входных наборах (имя и id).

Рассмотрим теперь каждый механизм работы сервиса dsm-register по отдельности.

Получение информации о входных файлах

Регистрация входных данных происходит следующим образом (рис. 5.2):

1) Система сбора данных записывает файлы на входной буфер и отправляет в очередь dsm.register.file.input информацию о входных файлах;
2) Сервис dsm-register делает запрос в dsm-manager на создание нового датасета в статусе OPEN;
3) Далее каждый файл из сообщения регистрируется со статусом CREATED в созданном датасете через dsm-manager
   (а) Помимо этого сервис dsm-register делает запрос в dsm-manager на поиск файла по имени в табице с темными файлами;
   (б) Если файл обнаружен, то dsm-register удаляет его из таблицы с помощью соответствующего запроса;
4) После того, как все файлы зарегистрированы в системе, dsm-register от-

правляет запрос на изменение статуса датасета на CLOSED. Датасет закрывается только один раз и повторно не открывается, файлы в закрытый датасет записывать нельзя;

5) Информация о новом датасете направляется в виде сообщения в очередь dsm.register.dataset.input для информирования системы управления процессами обработки.

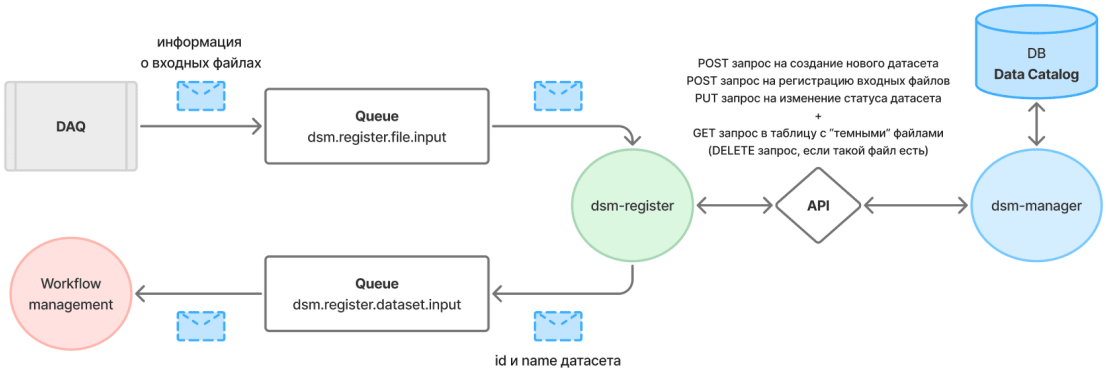


Рисунок 5.2 — Механизм регистрации входных файлов.

Получение информации о промежуточных файлах

Когда заканчивается очередной этап обработки, pilot-ы записывают выходные файлы на хранилище и информируют систему управления нагрузкой. Ей, в свою очередь, необходимо оповестить систему управления данными. Взаимодействие происходит следующим образом (рис. 5.3):

1) WMS отправляет в очередь dsm.register.file.process информацию о промежуточных файлах и идентификатор заранее созданного датасета, которому должны принадлежать данные файлы;
2) Сервис dsm-register получает сообщение и отправляет запрос в dsm-manager на получение информации о датасете;
3) Далее проверяется его статус, так как записывать файлы мы можем только в открытый датасет:
   (a) Если он имеет статус, отличный от OPEN, то dsm-register информирует систему управления нагрузкой о возникшей ошибке через очередь dsm.register.file.process.reply. На этом обработка сообщения заканчивается;

(б) В противном случае обработка продолжается;

4) Каждый файл в списке dsm-register регистрирует (с привязкой к данному датасету) в базе данных с помощью сервиса dsm-manager:
   (а) Если в процессе регистрации возникла какая-либо ошибка, оповещаем об этом WMS с указанием файла и типа ошибки;
   (б) Иначе в ту же очередь направляетя сообщение об успешной регистрации с указанием файла;
   (в) Помимо этого сервис, как и в случае с регистрацией входных файлов, проверяет наличие файла в таблице с темными файлами и удаляет его при успешном выполнении запроса.

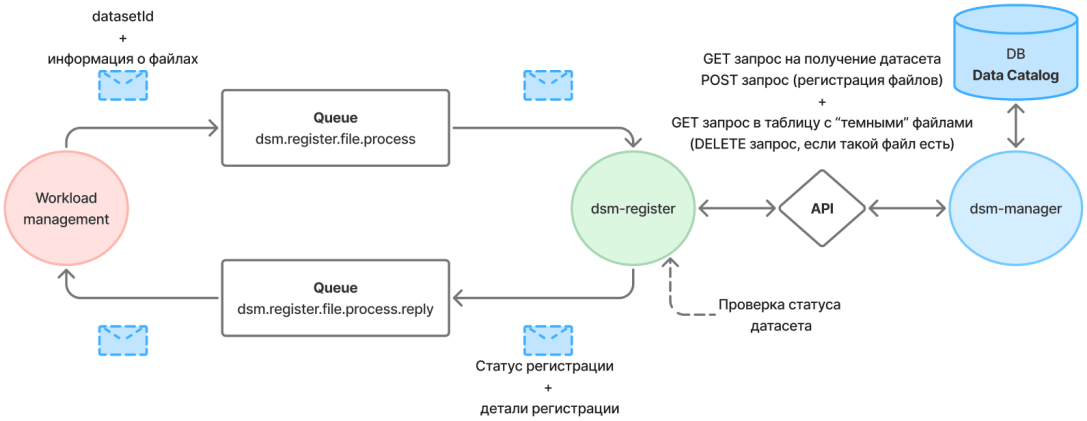


Рисунок 5.3 — Механизм регистрации промежуточных данных.

Важно упомянуть, какого типа ошибки могут возникать при попытке зарегистрировать файлы:

- Ошибка соединения (не удалось подключться к сервису dsm-manager или к БД);
- Ошибка формата (сообщение от WMS имеет неверный формат, например, отсутствуют некоторые поля);
- Логическая ошибка (попытка зарегистрировать файлы в закрытом датасете).

Необходимо отметить, что данный список не является законченным. Этап интеграционного тестирования поможет выявить возникновение ошибок другого типа.

Удаление данных

Наступит момент, когда системе управления процессами обработки потребуется удалить датасеты. Данное взаимодействие должно осуществляться следующим образом (рис. 5.4):

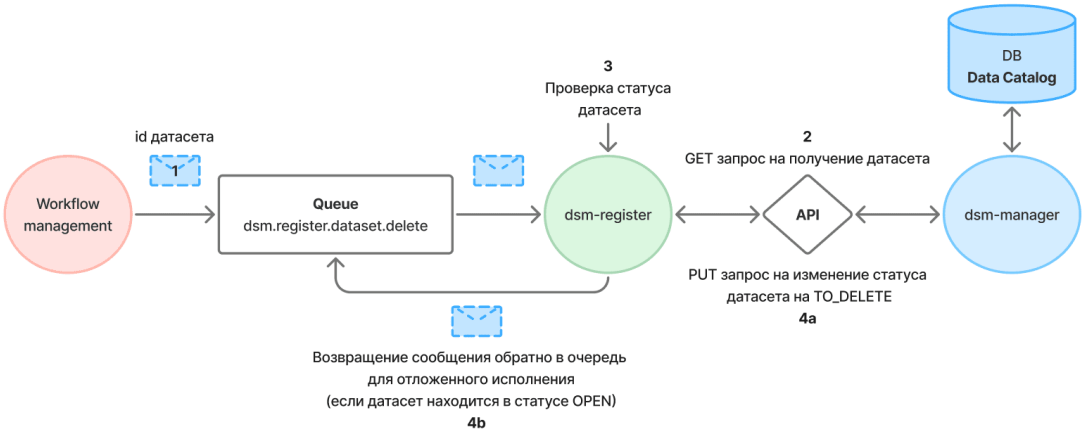


Рисунок 5.4 — Механизм приема заявок на удаление датасетов.

1) WFMS отправляет в очередь dsm.register.dataset.delete сообщение, которое содержит id датасета, который необходимо удалить;
2) Сервис dsm-register получает сообщение и отправляет запрос в dsm-manager на получение информации о наборе по его идентификатору;
3) Далее происходит проверка статуса датасета. Здесь возможны два варианта:
   (а) Если датасет находится в статусе OPEN, то мы не можем его удалить ввиду того, что в него записаны не все файлы, поэтому сообщение возвращается в конец очереди, из которой оно пришло;
   (б) В остальных случаях dsm-register делает запрос в dsm-manager на перевод датасета в статус TO_DELETE.

После этого непосредственным удалением файлов и датасетов занимается сервис dsm-inspector.

Выгрузка данных во внешнее хранилище

После завершения последнего этапа обработки мы получаем выходные файлы, которые необходимо выгрузить во внешнее хранилище для последую-

щей обработки и анализа. Принятие решения о выгрузке того или иного датасета лежит на системе управления процессами обработки.

Здесь алгоритм действий (рис. 5.5) полностью повторяет пункт с удалением датасетов, за тем исключением, что взаимодействие происходит через очередь dsm.register.dataset.upload и статус набора изменяется на TO_UPLOAD.

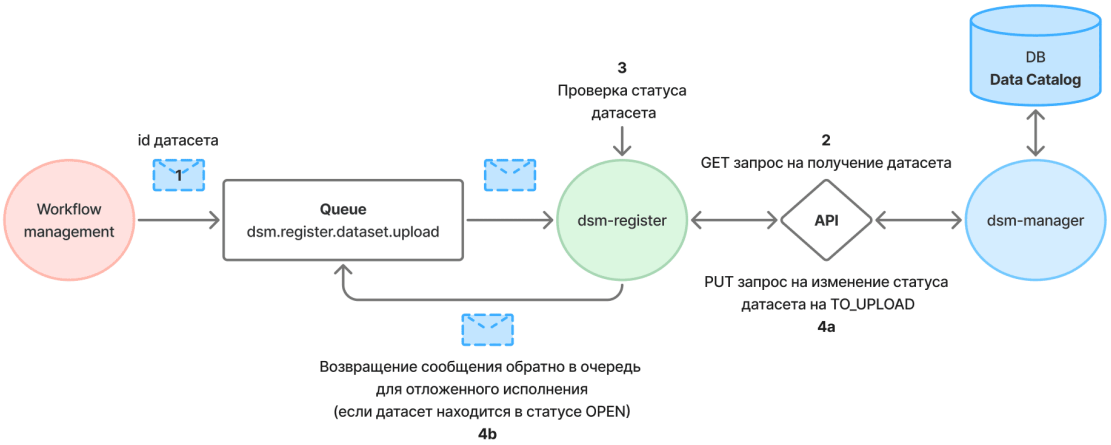


Рисунок 5.5 — Механизм приема заявок на выгрузку датасетов.

5.2.3 СЕРВИС DSM-INSPECTOR

Сервис, состоящий из набора фоновых задач для мониторинга состояния системы:

- Проверка целостности файлов;
- Удаление датасетов и файлов;
- Контроль использования хранилища;
- Контроль выгрузки данных.

Dsm-inspector в процессе работы тесно взаимодействует с сервисом dsm-manager и непосредственно с хранилищами.

Проверка целостности файлов

Сервис по проверке целостности файлов является сервисом мониторинга и осуществляет проверку каждые 5 часов. Важно отметить, что он работает только с теми файлами, которые находятся в закрытых датасетах. Алгоритм работы следующий (рис. 5.6):

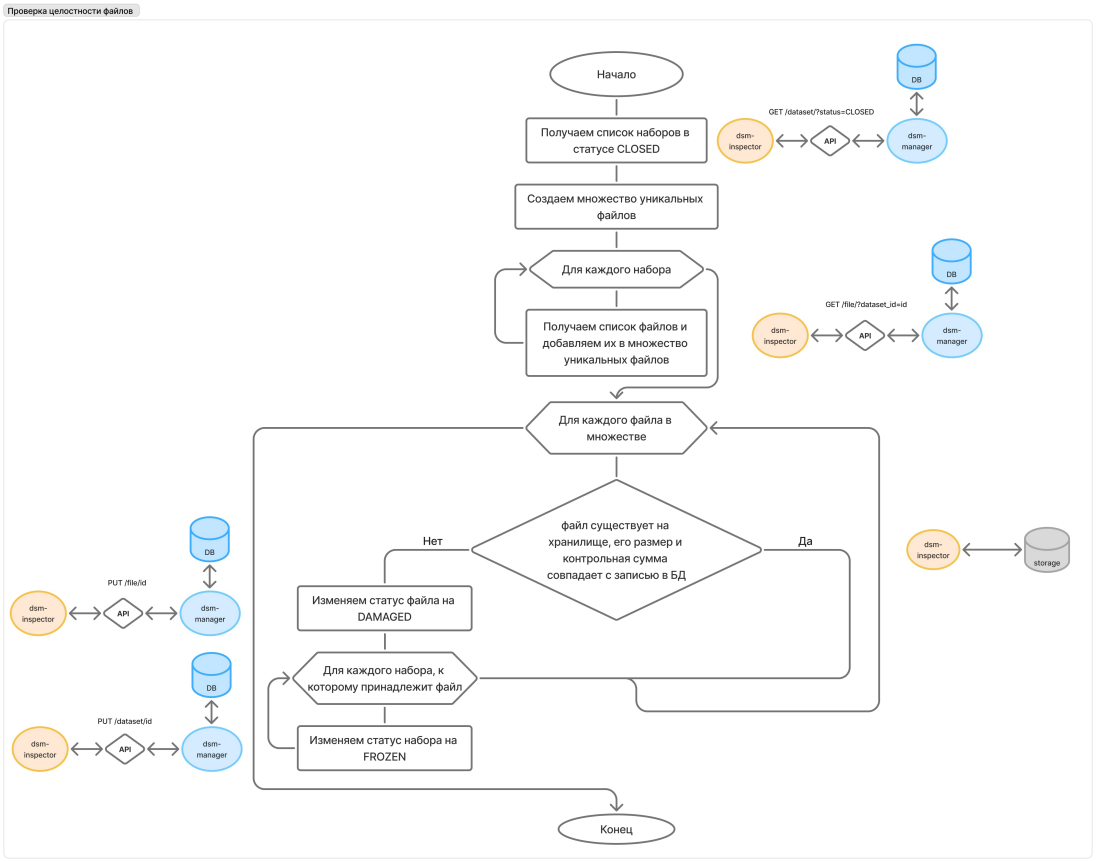


Рисунок 5.6 — Алгоритм проверки целостности файлов.

1) Через API, которое предоставляет dsm-manager, сервис получает список датасетов, которые находятся в статусе CLOSED;
2) По каждому датасету получаем его содержимое (информацию о файлах) по ID через dsm-manager;
3) Далее формируется множество уникальных файлов, так как один файл может принадлежать сразу нескольким наборам;
4) По каждому файлу:
   - проверяется наличие файла на хранилище;
   - проверяется размер файла;
   - проверяется контрольная сумма.
5) Если какая-то из вышеперечисленных проверок не пройдена, статус файла в БД изменяется на DAMAGED, а статус соотвествующего датасета на FROZEN.

Замороженные датасеты в последующем будут удаляться, но такое решение будет принимать WFMS. Сама процедура удаления описана ниже.

Удаление датасетов и файлов

Удаление датасетов и файлов является сервисом исполнения заявок на удаление наборов и выполняется при помощи трех процессов (рис. 5.7):

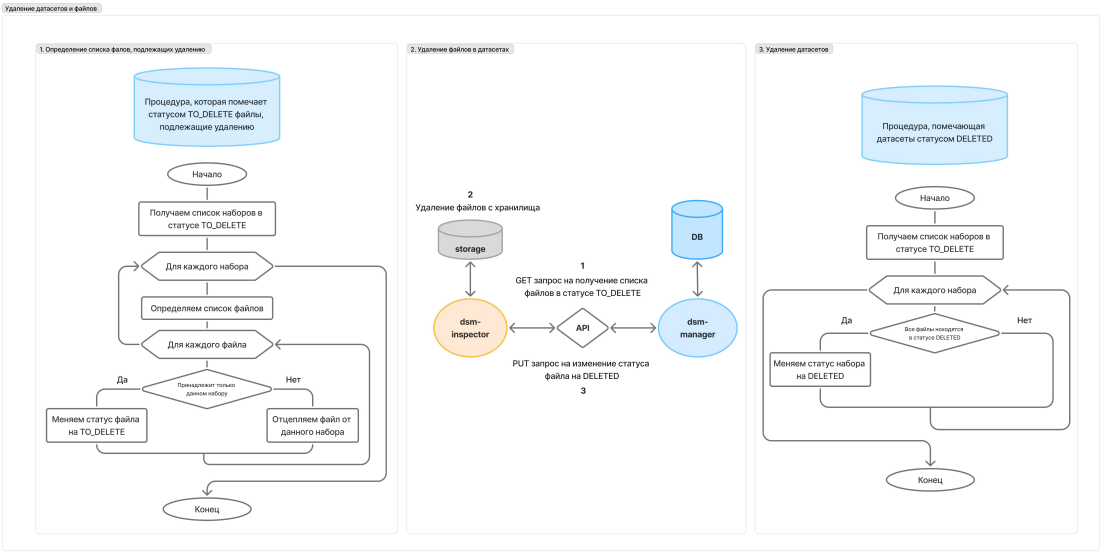


Рисунок 5.7 — Механизм удаления файлов.

1) Определение списка файлов, подлежащих удалению (хранимая процедура в БД):
   (а) Определяем список наборов в статусе TO_DELETE;
   (б) Для каждого набора определяем список файлов;
   (в) Для каждого файла проверяем, принадлежит ли он еще какому-нибудь набору:
      – Если нет, то устанавливаем ему статус TO_DELETE;
      – Если да, то отцепляем файл от данного датасета (удаляем соответствующую запись в таблице).
2) Удаление файлов в датасетах:
   (а) Сервис получает список файлов со статусом TO_DELETE;
   (б) Для каждого файла:
      i. Сервис dsm-inspector удаляет файл с соответствующего хранилища;
      ii. После удаления статус файла изменяется на DELETED посредством запроса в dsm-manager (если такого файла на хранилище нет, всё равно устанавливаем ему в каталоге статус DELETED).

3) Удаление датасетов (хранимая процедура в БД):
   (а) Получаем список датасетов со статусом TO_DELETE;
   (б) Для каждого датасета смотрим, все ли файлы находятся в статусе DELETED;
   (в) Если да, то помечаем датасет статусом DELETED.

Помимо исполнения заявок на удаление, сервис также занимается удалением «темных» данных.

«Темные» данные могут возникать в системе по разным причинам. Они и механизм обнаружения таких файлов будут описаны в части, посвященной контролю использования хранилища. Здесь же нас будет интересовать то, как мы принимаем решение об удалении того или иного файла.

Часть «темных» файлов удаляется из системы в момент регистрации входных/промежуточных данных. Вернее будет сказать, что файлы в этот момент перестают быть «темными» и информация об их присутствии на хранилище поступает в DSM. Подробнее этот момент разобран в разделе, посвящённом сервису dsm-register.

Оставшаяся часть файлов, информация о которых так и не поступила в систему в течение суток, удаляется сервисом dsm-inspector (рис. 5.8). Механизм работы следующий:

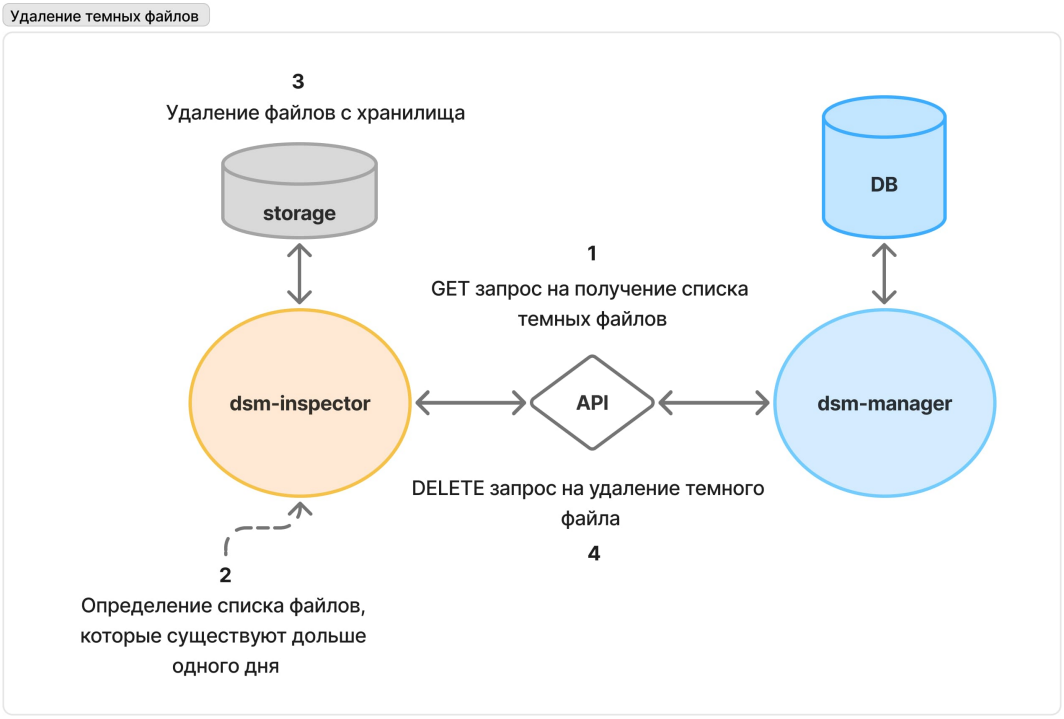


Рисунок 5.8 — Механизм удаления «темных» файлов.

1) Сервис получает список всех темных файлов в системе;
2) Формируется список файлов, которые существуют в системе дольше одного дня;
3) По каждому файлу:
   (а) Сервис удаляет файл с хранилища;
   (б) Далее через dsm-manager происходит удаление файла из БД по имени.

Контроль испозьзования хранилища

Контроль использования хранилища является сервисом мониторинга и имеет два сценария проверки:

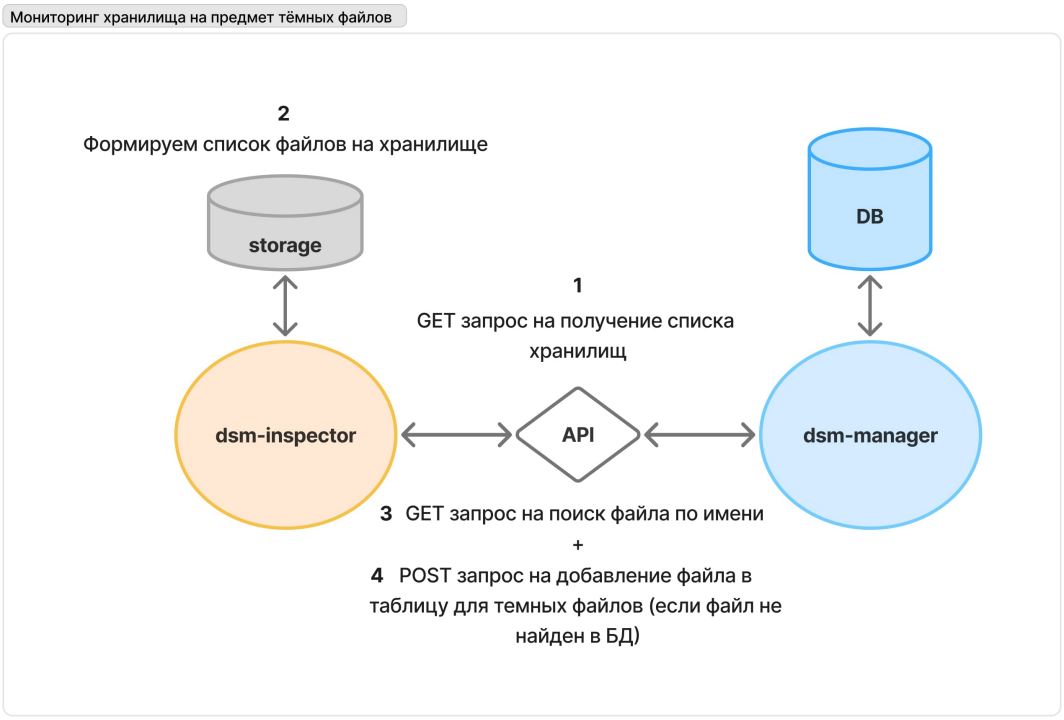


Рисунок 5.9 — Механизм мониторинга хранилища на предмет «темных» файлов.

– Мониторинг хранилища на предмет «темных» файлов (рис. 5.9):
   1) Сервис получает информацию о доступных хранилищах;
   2) По каждому хранилищу:
      (а) Формируется список файлов, которые существуют на хранилище;
      (б) Далее происходит поиск файла в базе данных по его имени;

(в) В случае отсутствия файла в каталоге, сервис добавляет его в таблицу для «темных» файлов через dsm-manager.

– Мониторинг заполненности хранилища (рис. 5.10):
  1) Сервис получает информацию по всем хранилищам;
  2) По каждому хранилищу считаем размер занятого пространства (суммарный размер всех файлов) и обновляем эту информацию в каталоге.

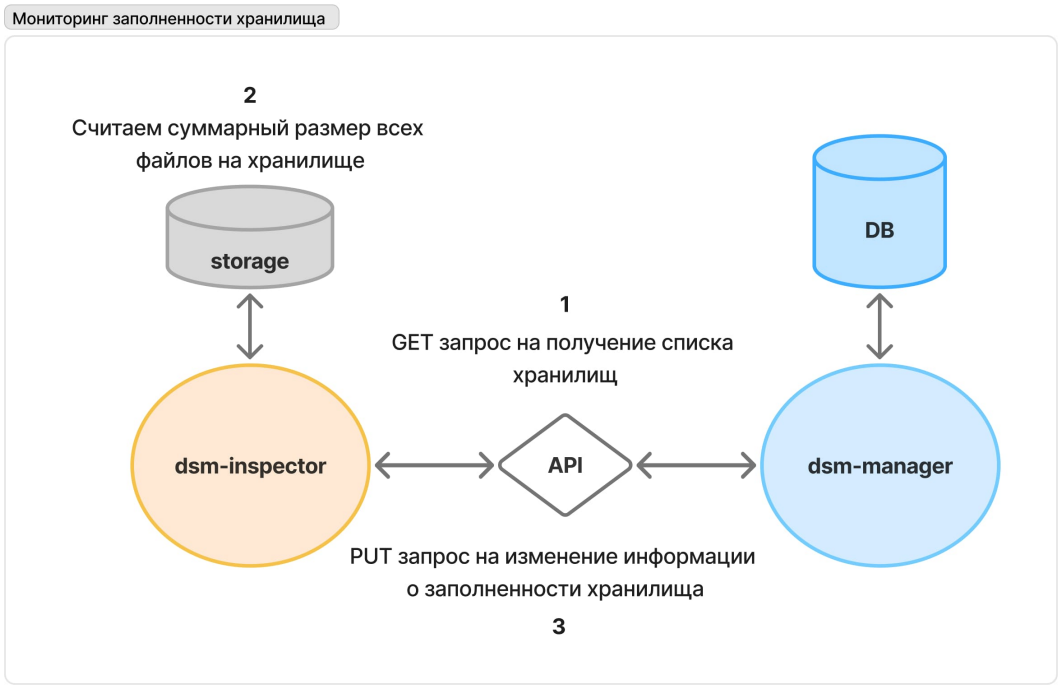


Рисунок 5.10 — Механизм мониторинга заполненности хранилища.

Стоит выделить основные причины, по которым в системе могут появляться «темные» данные:

1) В момент проверки файлы уже лежат на хранилище, но сообщение с информацией об этих файлах еще не обработано. В таком случае вскоре после получения сообщения файлы перестают быть «темными» и информация о них удаляется из соответствующей таблицы;
2) Ввиду сбоя в работе системы, информация о файлах так и не была отправлена в dsm-register и, соответственно, они не были зарегистрированы.

Контроль выгрузки данных

Контроль выгрузки данных является сервисом исполнения заявок на выгрузку файлов в наборе во внешнюю систему.

Внешнее хранилище, в которое будут выгружаться файлы, находится в зоне ответственности Rucio. Rucio – это программный комплекс с открытым исходным кодом, который обеспечивает функционал для организации, управления и доступа к данным.

Сами же файлы, предполагается, будут выгружаться посредством FTS3 (File Transfer Service) – это система передачи больших объемов данных, созданная для глобального распространения нескольких петабайтов данных с LHC [28]. Пользователь даёт FTS задание, указывая источник данных и назначение, а FTS ставит задание в очередь, планирует и выполняет передачу, повторяя ее, если нужно. Не менее важно, что система предоставляет комплексный интерфейс мониторинга, включая публикацию данных о передаче в формате JSON.

Следует отметить, что FTS лишь копирует файлы с внутреннего хранилища на внешнее. Поэтому, после успешной выгрузки, файлы с внутреннего хранилища необходимо будет удалить.

Планируется, что выгрузка будет происходить следующим образом:

1) Сервис отправляет запрос в dsm-manager на получение списка наборов в статусе TO_UPLOAD;
2) По каждому набору:
   (а) Получаем его содержимое (список файлов);
   (б) Создаем в Rucio новый датасет и узнаем, куда должны быть помещены файлы;
   (в) В FTS отправляем список файлов, которые необходимо выгрузить. Сюда входит полный путь до файла и новый путь до него во внешнем хранилище;
   (г) Каждому файлу устанавливается статус UPLOADING;
   (д) Набору так же устанавливается статус UPLOADING;
3) Получаем у FTS список выгруженных файлов;
4) Отправляем запрос в Rucio на регистрацию файлов с привязкой к датасету;
5) Датасет с выгруженными файлами помечается статусом TO_DELETE.

6 ПРОТОТИПИРОВАНИЕ

Прототипирование в разработке программного обеспечения предполагает создание упрощенной версии приложения, которое демонстрирует основной функционал.

Репозитории с исходным кодом сервисов расположены на внутреннем сервере МЛИТ ОИЯИ в GitLab. Их организация представлена на рис. 6.1.

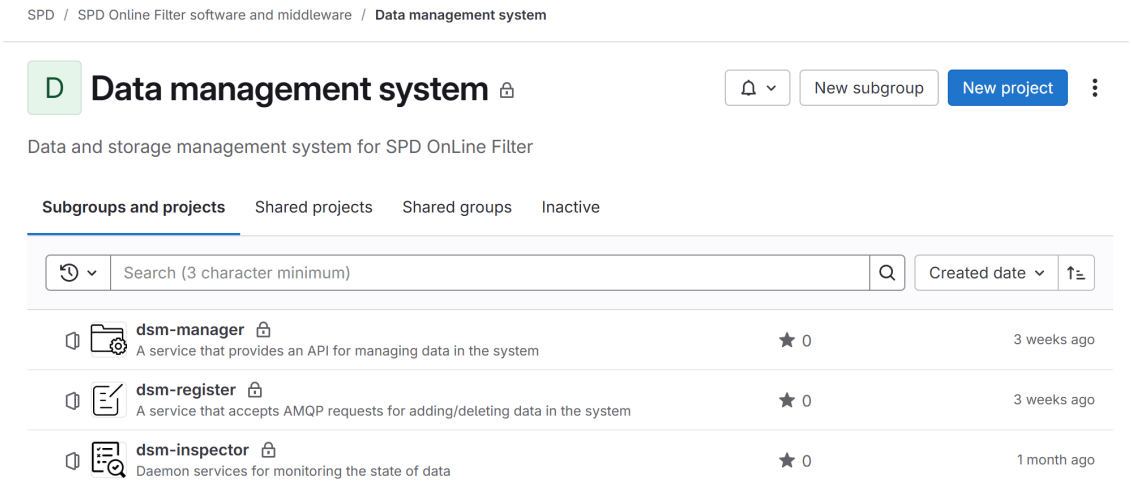


Рисунок 6.1 — Репозитории проекта с исходным кодом.

Разработка всех сервисов системы управления данными ведется на языке Python версии 3.11.

Проекты развернуты на виртуальной машине с помощью инструмента Docker Compose, каждое приложение в отдельном docker-контейнере.

6.1 DSM-MANAGER

Структура проекта представлена на рисунке 6.2. Отличительной особенностью данного сервиса является работа с базой данных. Помимо этого, он должен предоставлять API [29] для управления данными.

В качестве базы данных используется PostgreSQL, которая развернута на отдельной виртуальной машине.

```
├── README.md                          <- Файл верхнего уровня для разработчиков.
│
├── app
│   ├── migrations                     <- Модуль миграций для SQL Базы данных.
│   │   └── versions                   <- Скрипты миграции в соответствии с версией приложения.
│   ├── api                            <- Главный модуль приложения.
│   │   ├── main.py                    <- Скрипт построения приложения.
│   │   ├── config                     <- Модуль настройки всех компонент приложения.
│   │   ├── dao                        <- Модуль работы с данными.
│   │   │   ├── models                 <- Модуль с ORM моделями Базы данных.
│   │   │   ├── repositories           <- Модуль с с репозиториями операций к таблицам БД.
│   │   │   └── database.py            <- Объект доступа к Базе данных.
│   │   ├── domain                     <- Модуль с доменными моделями данных.
│   │   ├── dto                        <- Модуль с моделями данных для их отправки клиенту.
│   │   ├── mappers                    <- Модуль с функциями сопоставления моделей данных.
│   │   ├── services                   <- Модуль с сервисами бизнес логики.
│   │   ├── endpoints                  <- Модуль с точками входа в приложение.
│   │   │   ├── v1                     <- Функциональные точки входа в приложение.
│   │   │   ├── info_endpoints.py      <- Служебные точки входа в приложение.
│   │   │   └── exception_handlers.py  <- Обработчики исключений.
│   │   ├── exceptions                 <- Модуль с исключениями приложения.
│   │   └── constants                  <- Модуль с константами приложения.
│   ├── cli
│   │   └── docs.py                    <- CLI команды для генерации файла API.
│   └── settings                       <- Модуль с настройками приложения.
│
├── Dockerfile                         <- Файл с настройками Docker образа приложения.
│
├── docker-compose.yml                 <- Docker Compose файл с настройкой инфраструктуры.
│
└── pyproject.toml                     <- Poetry файл с зависимостями приложения.
```

Рисунок 6.2 — Описание структуры проекта dsm-manager.

Схемы таблиц задаются из кода с помощью ORM (Object-Relational Mapping) [30] моделей библиотеки SQLAlchemy [31]. С помощью ORM можно работать с данными из БД как с объектами, что значительно упрощает разработку. Пример такой модели представлен в приложении листинг 1.

Для непосредственного создания таблиц в базе данных используется библиотека Alembic [32], которая позволяет управлять миграциями. Миграция описывает последовательность изменений, которые необходимо применить к базе данных. Эти изменения могут включать создание новых таблиц, изменение существующих, добавление или удаление столбцов и так далее. Пример скрипта миграции представлен в приложении листинг 2.

После применения скриптов миграции соответствующие изменения появляются в базе данных (рис. 6.3).

Доступ к данным осуществляется с помощью паттерна «репозиторий». Основная цель данного паттерна – обеспечить стандартизированный способ доступа к данным и управления ими, абстрагируясь от технологий хранения данных. Пример репозитория представлен в приложении листинг 3.

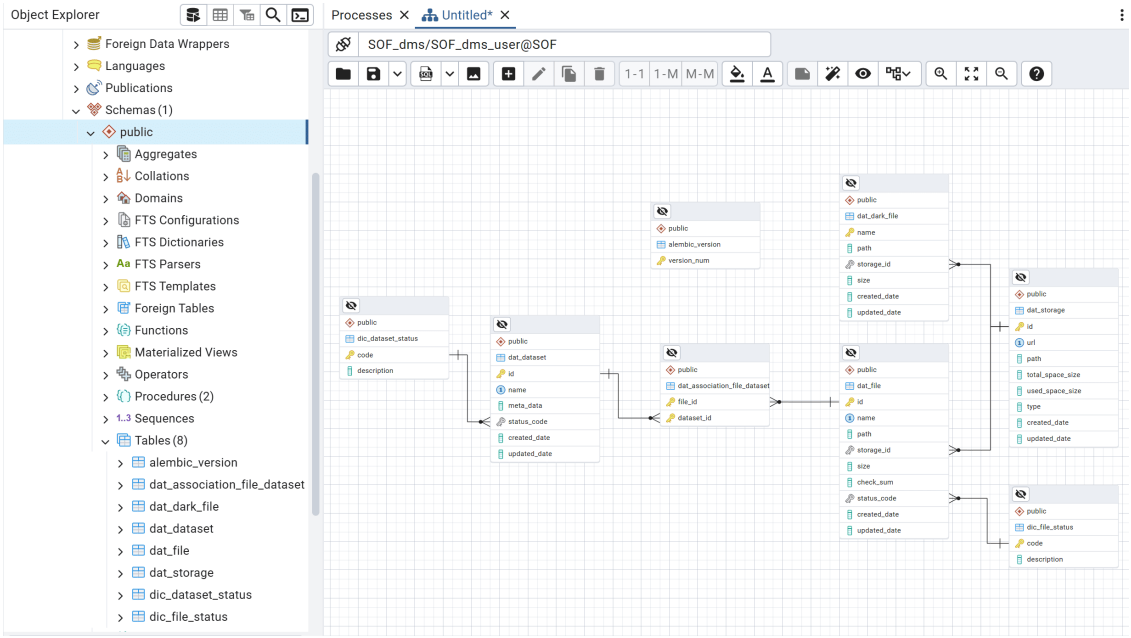


Рисунок 6.3 — Развернутая схема таблиц в БД.

В качестве фреймворка для создания API используется FastAPI [33]. Он имеет ряд преимуществ по сравнению с Flask и Django, таких как автоматическая валидация данных и встроенная интерактивная документация. Сгенерированную документацию можно посмотреть в интерфейсе Swagger UI по адресу <адрес сервера>/docs (рис. 6.4).

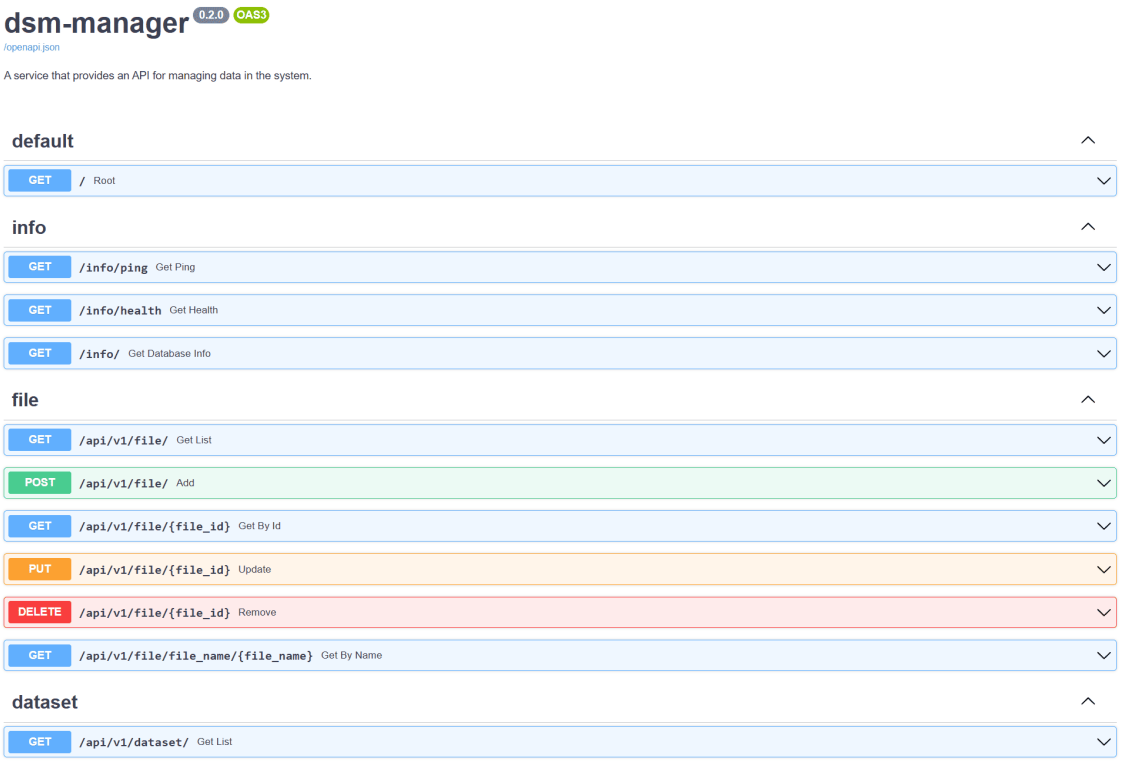


Рисунок 6.4 — Описание API сервиса dsm-manager.

6.2 DSM-REGISTER

Описание структуры проекта представлено на рисунке 6.5. Особенность данного сервиса заключается в взаимодействии с RabbitMQ.

```
├── README.md                          <- Файл верхнего уровня для разработчиков.
│
├── app
│   ├── executor
│   │   ├── main.py                    <- Точка входа (построение приложения).
│   │   ├── config                     <- Настройки всех компонент приложения.
│   │   ├── domain                     <- Модели данных приложения.
│   │   ├── dto
│   │   │   ├── rabbit                 <- DTO модели сообщений RabbitMQ.
│   │   │   └── manager                <- DTO модели DSM-Manager.
│   │   ├── rabbit                     <- Модуль работы с RabbitMQ.
│   │   │   ├── consumers              <- Модуль приёма сообщений из очередей RabbitMQ.
│   │   │   ├── producers              <- Модуль отправки сообщений в очереди RabbitMQ.
│   │   │   ├── reply                  <- Модуль отправки ответов в очереди RabbitMQ.
│   │   │   ├── rabbit_connector.py    <- Объект доступа к серверу RabbitMQ.
│   │   │   └── rabbit_server.py       <- Старт компоненты работы с RabbitMQ.
│   │   ├── services
│   │   │   ├── processor              <- Бизнес логика обработки сообщений.
│   │   │   └── manager                <- Adapter к сервису DSM-Manager.
│   │   └── utils                      <- Общие методы.
│   ├── settings                       <- Настройки приложения.
│   └── resources                      <- Примеры сообщений.
│
├── Dockerfile                         <- Файл с настройками Docker образа приложения.
│
├── docker-compose.yml                 <- Docker Compose файл с настройкой инфраструктуры.
│
└── pyproject.toml                     <- Poetry файл с зависимостями приложения.
```

Рисунок 6.5 — Описание структуры проекта dsm-register.

RabbitMQ – брокер сообщений с открытым исходным кодом. Он поддерживает несколько протоколов взаимодействия, гарантирует доставку сообщений получателю (если получатель не подтверждает получение сообщения в течение некоторого времени, RabbitMQ повторяет отправку) и реализует пуш-модель получения сообщений, т.е. он позволяет отправлять сообщения клиенту без подтверждения со стороны последнего [34].

В сервисе dsm-register потребовалось реализовать настройку очередей сообщений, а также их обработчиков. Для этого используется библиотека pika, которая позволяет взаимодействовать с RabbitMQ. Сконфигурированные очереди сообщений представлены на рисунке 6.6 (пример их построения см. в приложении листинг 4).

Также на сервисе была предусмотрена возможность отклонения некорректных сообщений. При этом, чтобы такого рода сообщения не были пол-

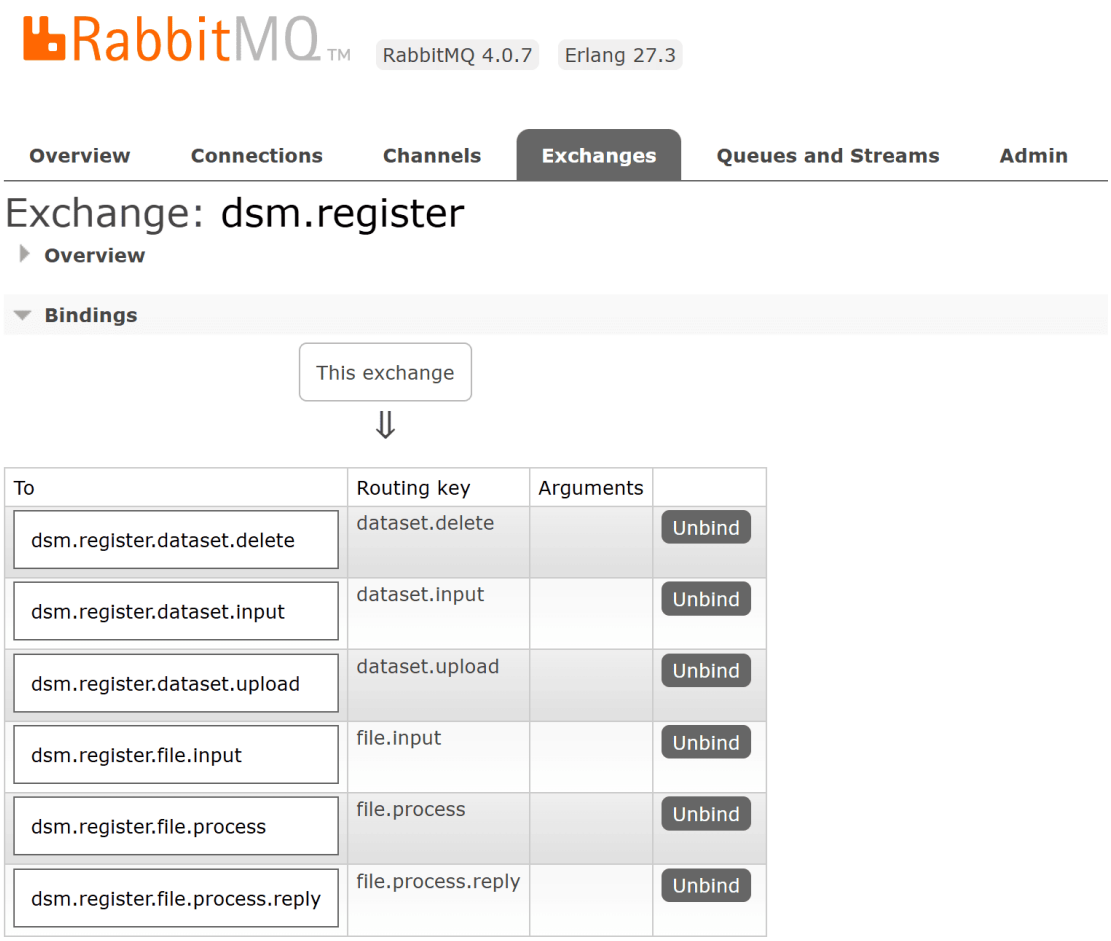


Рисунок 6.6 — Сконфигурированные очереди RabbitMQ.

ностью утеряны, для каждой очереди реализована дополнительная очередь «мертвых» сообщений [35] (рис. 6.7).

Помимо того, что очередь «мертвых» сообщений позволяет хранить сообщения, при обработке которых возникла ошибка, она также гарантирует, что эти ошибки не нарушат общую функциональность системы.

6.3 DSM-INSPECTOR

На рисунке 6.8 представлена структура проекта dsm-inspector.

В отличие от двух других сервисов, dsm-inspector работает непосредственно с файлами на хранилище, а не только с записями в базе данных.

Чтобы у сервиса был доступ к хранилищу, необходимо вмонтировать его в файловую систему внутри docker-контейнера. Это задаётся внутри docker-compose.yml файла для каждого сервиса отдельно.

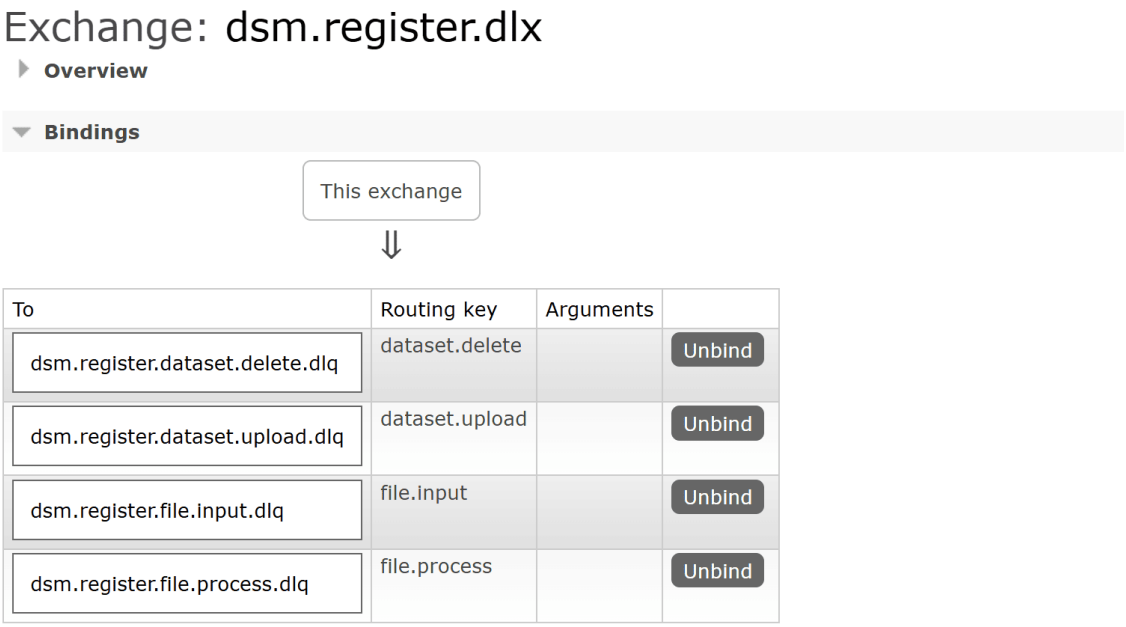


Рисунок 6.7 — Сконфигурированные очереди «мертвых» сообщений RabbitMQ.

```
├── README.md                          <- Файл верхнего уровня для разработчиков.
│
├── base
│   ├── executor
│   │   ├── config                     <- Настройки общих компонент приложения.
│   │   ├── domain                     <- Модели данных приложения.
│   │   ├── dto                        <- DTO модели DSM-Manager.
│   │   └── services                   <- Adapter к сервису DSM-Manager.
│   └── settings                       <- Настройки приложения.
│
├── files_deleting_inspector           <- Модуль, отвечающий за удаление данных.
│   ├── main.py                        <- Точка входа (построение приложения).
│   ├── config                         <- Настройки компонент приложения.
│   ├── inspectors                     <- Логика работы инспекторов.
│   ├── inspector_start.py             <- Старт компонент приложения.
│   └── Dockerfile                     <- Файл с настройками Docker образа приложения.
│
├── files_integrity_inspector          <- Модуль, отвечающий за проверку целостности данных.
│   └── ...
│
├── monitoring_inspector               <- Модуль, отвечающий за мониторинг хранилища.
│   └── ...
│
├── docker-compose.yml                 <- Docker Compose файл с настройкой инфраструктуры.
│
└── pyproject.toml                     <- Poetry файл с зависимостями приложения.
```

Рисунок 6.8 — Описание структуры проекта dsm-inspector.

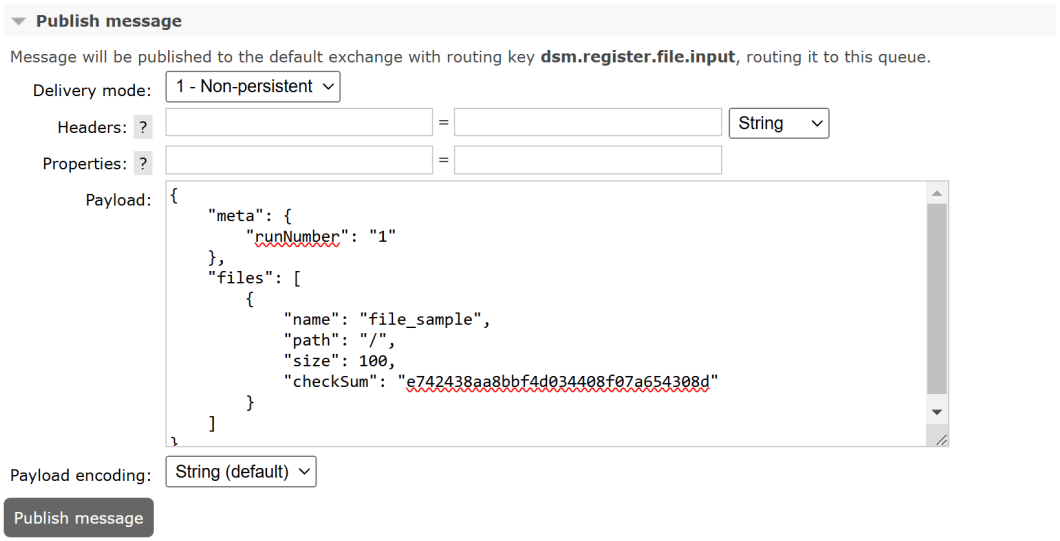
7 ТЕСТИРОВАНИЕ

В ходе работы были выполнены следующие задачи:

1) Расширение функционала dsm-register для взаимодействия с другими компонентами промежуточного ПО:
   (а) Реализация приема заявок на регистрацию промежуточных данных и механизма оповещения WMS о статусе регистрации файлов (см. пример в приложении листинг 5);
   (б) Настройка механизма оповещения WFMS о входных датасетах;
   (в) Реализация приема заявок на удаление датасета.
2) Реализация сервиса dsm-inspector:
   (а) Разработка сервиса проверки целостности файлов;
      – Дополнительно в dsm-manager добавлена возможность получения списка наборов, к которым принадлежит файл с данным ID.
   (б) Разработка сервиса контроля использования хранилища:
      i. Реализация сервиса мониторинга хранилища на предмет «темных» файлов;
         – Дополнительно в базе данных создана таблица для «темных» файлов и реализовано API к ней;
         – Добавлен поиск файла по имени в dsm-manager.
      ii. Реализация сервиса мониторинга заполненности хранилища.
   (в) Разработка сервиса удаления данных:
      i. Реализация сервиса исполнения заявок на удаление датасетов (см. пример в приложении листинг 6):
         – Дополнительно в dsm-manager добавлена возможность получения списка файлов по статусу.
      ii. Реализация сервиса удаления «темных» файлов.
3) Добавление дополнительного функционала в сервис dsm-manager для взаимодействия с WMS и WFMS.

7.1 РЕГИСТРАЦИЯ ВХОДНЫХ ФАЙЛОВ

Сперва проверим механизм регистрации входных файлов и оповещения WFMS о входных наборах. Для этого в очередь *dsm.register.file.input* отправим сообщение с информацией о входном файле (рис. 7.1а).



(а) Сообщение о входном файле.

```
app-1          | 2025-05-19 14:54:42 INFO: [DSM-REGISTER-I004] [CONSUMER-HANDLER] Received message FilesInputDto(meta={'runNumbe
r': '1'}, files=[FileInputDto(name='file_sample', path='/', size=100, check_sum='e742438aa8bbf4d034408f07a654308d')]) from queue
'.dsm.register.file.input' with delivery tag 1 [in /src/app/executer/rabbit/consumers/base/consumer_handler.py:50
app-1          | 2025-05-19 14:54:42 INFO: Processing msg [in /src/app/executer/services/processor/file_input_processor.py:42
app-1          | 2025-05-19 14:54:42 INFO: Registered dataset ID=b21d78e9-8f4d-4649-96a2-887f20404311. [in /src/app/executer/ser
vices/processor/file_input_processor.py:56
app-1          | 2025-05-19 14:54:42 INFO: Registered file ID=1929896f-089a-41d0-8fa7-8a9d6f13d553 in dataset ID=b21d78e9-8f4d-4
649-96a2-887f20404311. [in /src/app/executer/services/processor/file_input_processor.py:70
app-1          | 2025-05-19 14:54:43 INFO: Finish registering files. Dataset ID=b21d78e9-8f4d-4649-96a2-887f20404311 CLOSED! [in
```

(б) Лог о завершении регистрации входных файлов и закрытии созданного датасета.

(в) Сообщение с информацией о созданном датасете.

Рисунок 7.1 — Проверка работоспособности механизма регистрации первичных файлов и отправки сообщений о входных наборах.

Сервис dsm-register получает данное сообщение, создает новый набор и регистрирует файл в каталоге с привязкой к данному набору. После завершения регистрации набор закрывается (рис. 7.1б). Информация об этом наборе направляется в очередь *dsm.register.dataset.input* (рис. 7.1в) для информирования системы управления процессами обработки.

7.2 РЕГИСТРАЦИЯ ВЫХОДНЫХ ФАЙЛОВ

Проверим теперь работоспособность механизма регистрации выходных файлов.

Для начала отправим в очередь *dsm.register.file.process* сообщение с информацией о промежуточном файле (рис. 7.2).

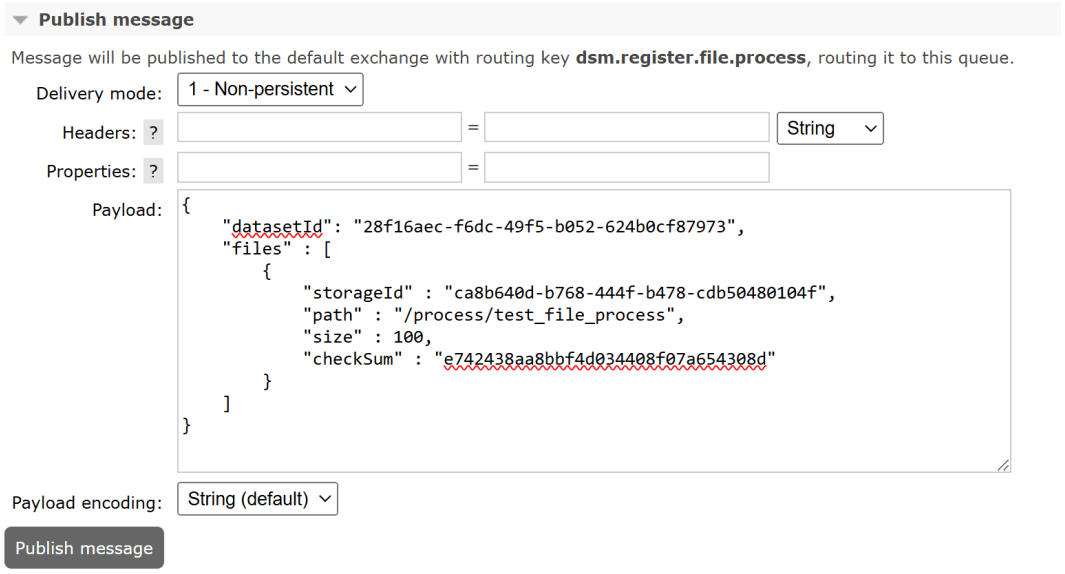


Рисунок 7.2 — Сообщение о выходном файле.

Проверим, что файл появился в системе с привязкой к нужному набору (рис. 7.3) и что в очередь dsm.register.file.process.reply было отправлено сообщение об успешной регистрации файла (рис. 7.4).

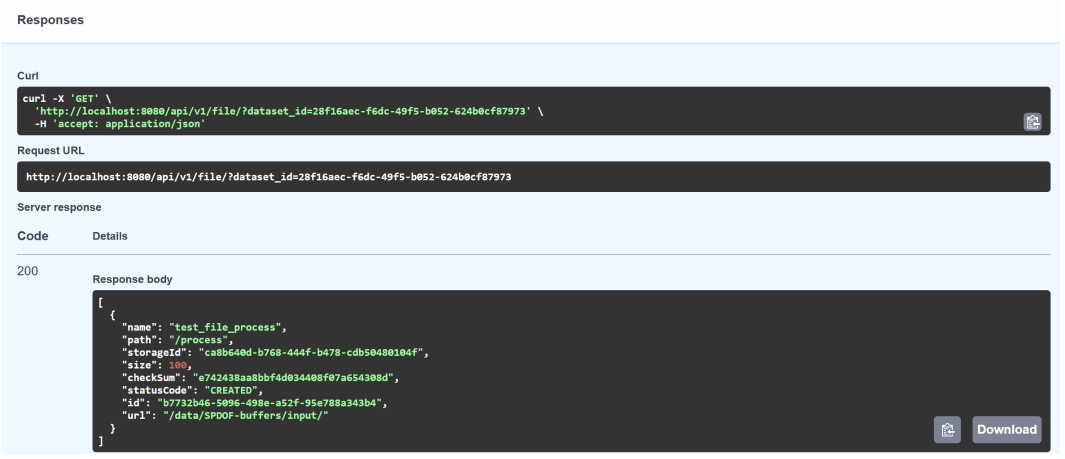


Рисунок 7.3 — Информация о файле в БД.

Теперь закроем набор, в котором был зарегистрирован последний файл, и попытаемся зарегистрировать новый файл в этом же датасете. В таком случае

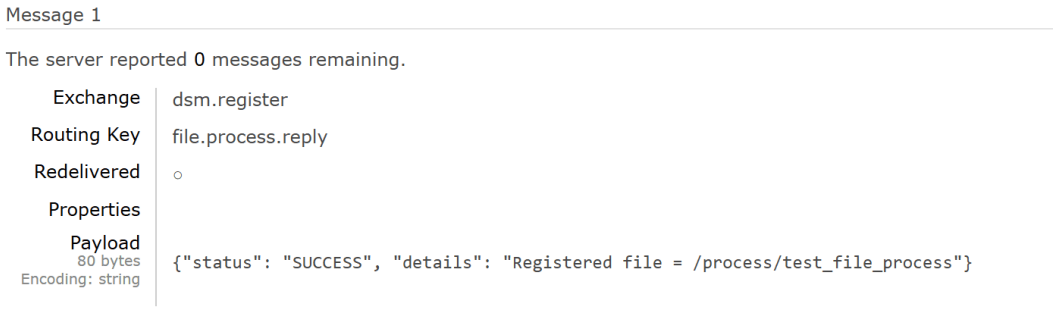


Рисунок 7.4 — Сообщение с информацией о статусе регистрации файла.

должна возникнуть ошибка при регистрации файла, т.к. добавление файлов возможно только в открытый датасет.

Проверим, что в очередь *dsm.register.file.process.reply* поступило сообщение об ошибке при регистрации файла (рис. 7.5).

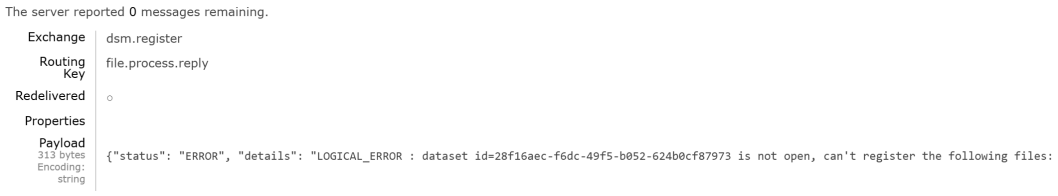
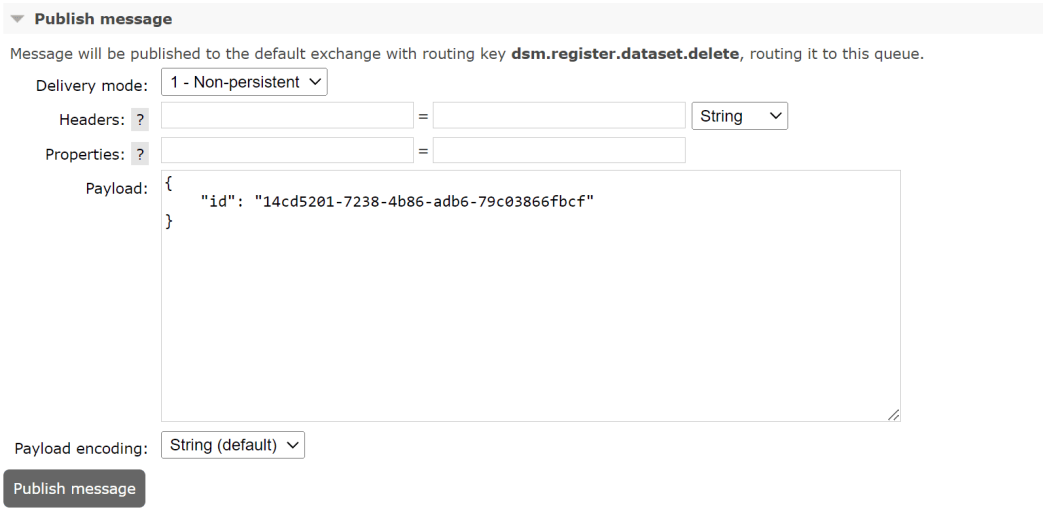


Рисунок 7.5 — Сообщение с информацией о статусе регистрации файла.

7.3 ПРИЕМ ЗАЯВОК НА УДАЛЕНИЕ ДАТАСЕТОВ

	id [PK] uuid	name character varying (255)	meta_data json	status_code character varying (20)
1	05d4c366-6d5a-468b-9d80-b21ae2ba0ce0	input.test.c40a08fe-4a75-4157-a04d-92946d5c0967.raw.output...	{"task_id": 19}	OPEN
2	0859c70b-2c4d-45c8-93e2-a2e804d4f53c	input.test.c40a08fe-4a75-4157-a04d-92946d5c0967.raw.log.1	{"task_id": 19}	OPEN
3	14cd5201-7238-4b86-adb6-79c03866fbcf	input.test.c1471f0f-4b55-45d8-9d66-97c42ff7adff.raw	{"run_number": 96, "files": 10}	CLOSED

(а) Информация о датасете в статусе CLOSED.



(б) Заявка на удаление датасета в статусе CLOSED.

```
app-1  | 2025-01-18 16:26:06 INFO: [DSM-REGISTER-I004] [CONSUMER-HANDLER] Received message DatasetDeleteDto(id=UUID('14cd5201-7238-4b86-
adb6-79c03866fbcf')) from queue '.dsm.register.dataset.delete' with delivery tag 56 [in /src/app/executer/rabbit/consumers/base/consumer
_handler.py:50
app-1  | 2025-01-18 16:26:06 INFO: Processing msg. [in /src/app/executer/services/processor/dataset_delete_processor.py:23
app-1  | 2025-01-18 16:26:06 INFO: Dataset ID=14cd5201-7238-4b86-adb6-79c03866fbcf status changed to TO_DELETE. [in /src/app/executer/se
rvices/processor/dataset_delete_processor.py:35
```

(в) Лог с информацией о получении сообщения и изменения статуса набора на TO_DELETE.

	id [PK] uuid	name character varying (255)	meta_data json	status_code character varying (20)
1	05d4c366-6d5a-468b-9d80-b21ae2ba0ce0	input.test.c40a08fe-4a75-4157-a04d-92946d5c0967.raw.output...	{"task_id": 19}	OPEN
2	0859c70b-2c4d-45c8-93e2-a2e804d4f53c	input.test.c40a08fe-4a75-4157-a04d-92946d5c0967.raw.log.1	{"task_id": 19}	OPEN
3	14cd5201-7238-4b86-adb6-79c03866fbcf	input.test.c1471f0f-4b55-45d8-9d66-97c42ff7adff.raw	{"run_number": 96, "files": 10}	TO_DELETE

(г) Проверка изменения статуса набора в базе данных.

Рисунок 7.6 — Проверка механизма приема и обработки заявок на удаление датасета в статусе CLOSED.

Далее необходимо было реализовать прием заявок на удаление набора. Все подобные заявки проходят через очередь *dsm.register.dataset.delete*.

Сервис dsm-register при получении заявки должен проверять статус датасета, который мы хотим удалить. Если статус OPEN, то заявка должна быть возвращена в очередь для отложенной обработки.

Выберем какой-нибудь набор в статусе CLOSED (рис. 7.6а) и через панель администрирования RabbitMQ отправим заявку на удаление выбранного датасета (рис. 7.6б).

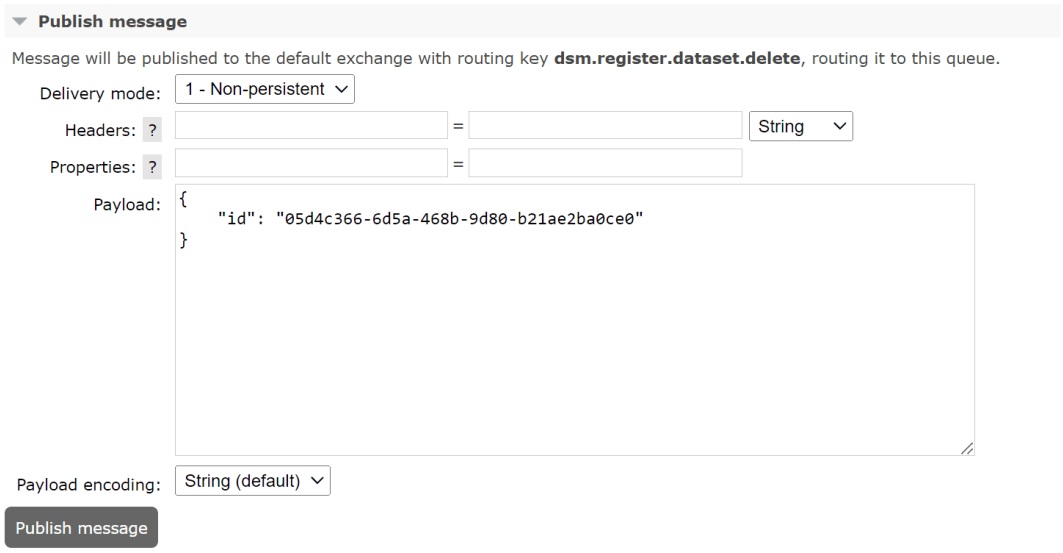
Сервис dsm-register получает сообщение из очереди, обрабатывает его и

изменяет статус набора на TO_DELETE (рис. 7.6в-7.6г).

Если мы теперь захотим отправить заявку на удаление открытого набора (рис. 7.7а-7.7б), то увидим, что после проверки статуса набора сервис возвращает сообщение обратно в очередь (рис. 7.7в).

	id [PK] uuid	name character varying (255)	meta_data json	status_code character varying (20)
1	05d4c366-6d5a-468b-9d80-b21ae2ba0ce0	input.test.c40a08fe-4a75-4157-a04d-92946d5c0967.raw.output...	{"task_id": 19}	OPEN

(а) Информация о датасете в статусе OPEN.



(б) Заявка на удаление датасета в статусе OPEN.

```
app-1  | 2025-01-18 16:34:40 INFO: [DSM-REGISTER-I004] [CONSUMER-HANDLER] Received message DatasetDeleteDto(id=UUID('05d4c366-6d5a-468b-
9d80-b21ae2ba0ce0')) from queue '.dsm.register.dataset.delete' with delivery tag 2547 [in /src/app/executer/rabbit/consumers/base/consum
er_handler.py:50
app-1  | 2025-01-18 16:34:40 INFO: Processing msg. [in /src/app/executer/services/processor/dataset_delete_processor.py:23
app-1  | 2025-01-18 16:34:40 INFO: Dataset ID=05d4c366-6d5a-468b-9d80-b21ae2ba0ce0 OPEN. The message returned to the queue for deferred
processing. [in /src/app/executer/services/processor/dataset_delete_processor.py:28
```

(в) Лог с информацией о получении сообщения и возвращения его в очередь для отложенной обработки.

Рисунок 7.7 — Проверка механизма приема и обработки заявок на удаление датасета в статусе OPEN.

7.4 ПРОВЕРКА ЦЕЛОСТНОСТИ ДАННЫХ

Теперь проверим, как работает в dsm-inspector проверка целостности файлов. Данный сервис раз в 5 часов должен проходить по всем закрытым наборам и проверять по каждому файлу его наличие на хранилище, размер и контрольную сумму.

Выберем какой-нибудь файл (рис. 7.8а-7.8б), принадлежащий набору в статусе CLOSED, и в базе данных изменим его размер (рис. 7.8в).

	id [PK] uuid	name character varying (255)	path character varying (255)	storage_id uuid	size integer
1	0fb1e1af-5c02-4d17-87a9-64debf5e6a17	input.test.a976a020-3de5-44e2-91ee-319e426eda2f.raw	input_40	b3307ad4-f2b3-4f3a-a390-4f4e2762c620	50

(а) Корректная информация о файле в БД.

(б) Полная информация о выбранном файле в БД.

	id [PK] uuid	name character varying (255)	path character varying (255)	storage_id uuid	size integer
1	0fb1e1af-5c02-4d17-87a9-64debf5e6a17	input.test.a976a020-3de5-44e2-91ee-319e426eda2f.raw	input_40	b3307ad4-f2b3-4f3a-a390-4f4e2762c620	100

(в) Измененная информация (размер) о файле в БД.

Рисунок 7.8 — Изменение информации о размере файла в базе данных.

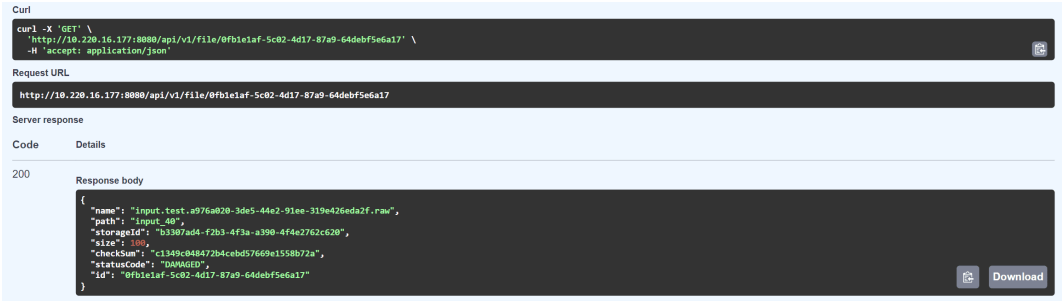
Проверим также, что выбранный файл действительно принадлежит закрытому набору (рис. 7.9).

Рисунок 7.9 — Информация о наборе, которому принадлежит выбранный файл.

После проверки целостности всех файлов видим, что выбранный нами файл оказался повреждён и что соответствующий набор переведён в статус FROZEN (рис. 7.10).

```
integrity-inspector-1  | 2025-01-19 13:31:47 INFO: File /data/SPDOF-buffers/input/input_40/input.test.a976a020-3de5-44e2-91ee-319e426eda
2f.raw DAMAGED! [in /src/files_integrity_inspector/file_integrity_inspector.py:77
integrity-inspector-1  | 2025-01-19 13:31:47 INFO: HTTP Request: PUT http://app:8080/api/v1/dataset/f61828be-64b5-44e8-9d18-a1a22068094d
 "HTTP/1.1 200 OK" [in /src/.venv/lib/python3.11/site-packages/httpx/_client.py:1038
integrity-inspector-1  | 2025-01-19 13:31:47 INFO: Dataset ID=f61828be-64b5-44e8-9d18-a1a22068094d FROZEN [in /src/files_integrity_inspe
ctor/file_integrity_inspector.py:89
```

Рисунок 7.10 — Лог с информацией о повреждении файла и изменении статуса набора на FROZEN.



(а) Информация о выбранном файле.

(б) Информация о соответствующем наборе.

Рисунок 7.11 — Проверка изменения статусов файла и датасета после отработки сервиса проверки целостности файлов.

Далее через API, которое предоставляет dsm-manager к базе данных, проверяем, действительно ли произошло изменение статуса у файла и набора. Как видно из рисунка 7.11, механизм действительно отработал.

Здесь рассматривался только случай несовпадения фактического размера файла с его каталожной записью, но это справедливо и в случаях контрольной суммы файла и его наличия на хранилище.

7.5 ИСПОЛНЕНИЕ ЗАЯВОК НА УДАЛЕНИЕ ДАТАСЕТОВ

Проверим теперь часть сервиса dsm-inspector, которая отвечает за удаление датасетов и файлов по заявкам.

Запрос | История запросов

```
SELECT dat_file.id, dat_file.name, dat_file.status_code, dat_file.path FROM dat_dataset
JOIN dat_association_file_dataset fd on fd.dataset_id = dat_dataset.id
JOIN dat_file on fd.file_id = dat_file.id
WHERE dat_dataset.status_code = 'TO_DELETE'
```

Data Output | Сообщения | Notifications

	id [PK] uuid	name character varying (255)	status_code character varying (20)	path character varying (255)
1	c81794fe-c57c-485c-89c7-0ace02f81883	input.test.41dad14a-e58c-4cd0-979a-16ee3e1cdb03.r...	CREATED	input_4
2	aa20e90d-6892-4d1a-aad1-1f3c5a7e4dc2	input.test.08850bb9-ae56-460e-875e-486f166c7533.r...	CREATED	input_4
3	b41ef691-3a5d-4d17-90d9-715a01e109a7	input.test.297eb0ea-c6e2-4261-acdf-b0d6f1c87482.ra...	CREATED	input_4
4	baa79005-1115-4bef-a82c-3b342c2213f0	input.test.8e607ad1-c88a-4e95-ba61-b1d539282662.r...	CREATED	input_4
5	6fa4d091-f983-4e21-92b8-ce68ea020732	input.test.d8a5c878-1ebe-4828-84b7-b37056bf6ac5.r...	CREATED	input_4
6	4255668b-26f2-4be7-98d0-64d82832c3ac	input.test.faa4e27f-ae2a-49d8-8934-5bfaa3102709.raw	CREATED	input_4
7	ec5ba1bf-b30b-47dc-8bc7-c76689bb58e1	input.test.8c04f7d8-2f26-4cc3-a81e-1b0dc796239e.ra...	CREATED	input_4
8	1e87276a-0462-4964-803e-7b6f96ea018d	input.test.3ea2562e-aa73-4c5e-8911-576db19d7fc5.r...	CREATED	input_4
9	8e238086-b493-459e-a567-627e00ab08e4	input.test.0aec3382-efe0-4986-8237-f07bc545248c.ra...	CREATED	input_4
10	760ecdb0-6f41-4515-b880-c7c21890ed81	input.test.451393ef-5ecb-4ed8-afd4-454acecc92d6.raw	CREATED	input_4

(а) Информация о файлах, принадлежащих набору в статусе TO_DELETE.

Запрос | История запросов

```
SELECT dat_file.id, dat_file.name, dat_file.status_code, dat_file.path FROM dat_dataset
JOIN dat_association_file_dataset fd on fd.dataset_id = dat_dataset.id
JOIN dat_file on fd.file_id = dat_file.id
WHERE dat_dataset.status_code = 'TO_DELETE'
```

Data Output | Сообщения | Notifications

	id [PK] uuid	name character varying (255)	status_code character varying (20)	path character varying (255)
1	c81794fe-c57c-485c-89c7-0ace02f81883	input.test.41dad14a-e58c-4cd0-979a-16ee3e1cdb03.r...	TO_DELETE	input_4
2	aa20e90d-6892-4d1a-aad1-1f3c5a7e4dc2	input.test.08850bb9-ae56-460e-875e-486f166c7533.r...	TO_DELETE	input_4
3	b41ef691-3a5d-4d17-90d9-715a01e109a7	input.test.297eb0ea-c6e2-4261-acdf-b0d6f1c87482.ra...	TO_DELETE	input_4
4	baa79005-1115-4bef-a82c-3b342c2213f0	input.test.8e607ad1-c88a-4e95-ba61-b1d539282662.r...	TO_DELETE	input_4
5	6fa4d091-f983-4e21-92b8-ce68ea020732	input.test.d8a5c878-1ebe-4828-84b7-b37056bf6ac5.r...	TO_DELETE	input_4
6	4255668b-26f2-4be7-98d0-64d82832c3ac	input.test.faa4e27f-ae2a-49d8-8934-5bfaa3102709.raw	TO_DELETE	input_4
7	ec5ba1bf-b30b-47dc-8bc7-c76689bb58e1	input.test.8c04f7d8-2f26-4cc3-a81e-1b0dc796239e.ra...	TO_DELETE	input_4
8	1e87276a-0462-4964-803e-7b6f96ea018d	input.test.3ea2562e-aa73-4c5e-8911-576db19d7fc5.r...	TO_DELETE	input_4
9	8e238086-b493-459e-a567-627e00ab08e4	input.test.0aec3382-efe0-4986-8237-f07bc545248c.ra...	TO_DELETE	input_4
10	760ecdb0-6f41-4515-b880-c7c21890ed81	input.test.451393ef-5ecb-4ed8-afd4-454acecc92d6.raw	TO_DELETE	input_4

(б) Измененная информация о файлах, подлежащих удалению.

Рисунок 7.12 — Проверка изменения статусов файлов после отработки процедуры, помечающей файлы, подлежащие удалению.

Для начала проверим корректность работы процедуры, которая для каждого набора, подлежащего удалению, помечает файлы статусом TO_DELETE, если они принадлежат только данному набору.

Посмотрим на список файлов, которые привязаны к набору в статусе

TO_DELETE (рис. 7.12а). Далее запускаем соответствующую процедуру и видим, что у всех файлов из списка изменился статус (рис. 7.12б).

Далее сервис получает список всех файлов, подлежащих удалению, а затем в многопоточном режиме удаляет каждый файл с хранилища и изменяет его статус на DELETED. Это мы видим на рисунке 7.13.

```
deleting-inspector-1  | 2025-01-19 14:17:13 INFO: Foarm file list with status TO_DELETE [in /src/files_deleting_inspector/file_del
ete_inspector.py:32
deleting-inspector-1  | 2025-01-19 14:17:13 INFO: HTTP Request: GET http://app:8080/api/v1/file/?file_status=TO_DELETE "HTTP/1.1 2
00 OK" [in /src/.venv/lib/python3.11/site-packages/httpx/_client.py:1038
deleting-inspector-1  | 2025-01-19 14:17:13 INFO: HTTP Request: GET http://app:8080/api/v1/storage/b3307ad4-f2b3-4f3a-a390-4f4e276
2c620 "HTTP/1.1 200 OK" [in /src/.venv/lib/python3.11/site-packages/httpx/_client.py:1038
deleting-inspector-1  | 2025-01-19 14:17:13 INFO: File /data/SPDOF-buffers/input/input_4/input.test.8c04f7d8-2f26-4cc3-a81e-1b0dc7
96239e.raw DELETED! [in /src/files_deleting_inspector/file_delete_inspector.py:44
```

(а) Лог с информацией об удалении файлов, находящихся в статусе TO_DELETE.

Запрос | История запросов

```
1 SELECT dat_file.id, dat_file.name, dat_file.status_code, dat_file.path FROM dat_dataset
2 JOIN dat_association_file_dataset fd on fd.dataset_id = dat_dataset.id
3 JOIN dat_file on fd.file_id = dat_file.id
4 WHERE dat_dataset.status_code = 'TO_DELETE'
```

Data Output | Сообщения | Notifications

	id [PK] uuid	name character varying (255)	status_code character varying (20)	path character varying (255)
1	c81794fe-c57c-485c-89c7-0ace02f81883	input.test.41dad14a-e58c-4cd0-979a-16ee3e1cdb03.r...	DELETED	input_4
2	aa20e90d-6892-4d1a-aad1-1f3c5a7e4dc2	input.test.08850bb9-ae56-460e-875e-486f166c7533.r...	DELETED	input_4
3	b41ef691-3a5d-4d17-90d9-715a01e109a7	input.test.297eb0ea-c6e2-4261-acdf-b0d6f1c87482.ra...	DELETED	input_4
4	baa79005-1115-4bef-a82c-3b342c2213f0	input.test.8e607ad1-c88a-4e95-ba61-b1d539282662.r...	DELETED	input_4
5	6fa4d091-f983-4e21-92b8-ce68ea020732	input.test.d8a5c878-1ebe-4828-84b7-b37056bf6ac5.r...	DELETED	input_4
6	4255668b-26f2-4be7-98d0-64d82832c3ac	input.test.faa4e27f-ae2a-49d8-8934-5bfaa3102709.raw	DELETED	input_4
7	ec5ba1bf-b30b-47dc-8bc7-c76689bb58e1	input.test.8c04f7d8-2f26-4cc3-a81e-1b0dc796239e.ra...	DELETED	input_4
8	1e87276a-0462-4964-803e-7b6f96ea018d	input.test.3ea2562e-aa73-4c5e-8911-576db19d7fc5.r...	DELETED	input_4
9	8e238086-b493-459e-a567-627e00ab08e4	input.test.0aec3382-efe0-4986-8237-f07bc545248c.ra...	DELETED	input_4
10	760ecdb0-6f41-4515-b880-c7c21890ed81	input.test.451393ef-5ecb-4ed8-afd4-454acecc92d6.raw	DELETED	input_4

(б) Измененная информация об удаленных файлах.

Рисунок 7.13 — Проверка изменения статусов файлов после отработки сервиса по удалению файлов.

И в конце переходим к процедуре, которая меняет статус датасетов, подлежащих удалению, на DELETED, если все файлы в них удалены. На рисунке 7.14 представлена информация о наборе до и после срабатывания процедуры.

	id [PK] uuid	name character varying (255)	meta_data json	status_code character varying (20)
1	bf04dfe1-db61-4195-b584-45de6ea8f04d	input.test.d95edab2-8f40-4b67-9fd7-01565512228f.raw	{"run_number": 3, "files": 10}	TO_DELETE

(а) Информация о наборе, к которому привязаны удаленные файлы.

	id [PK] uuid	name character varying (255)	meta_data json	status_code character varying (20)
1	bf04dfe1-db61-4195-b584-45de6ea8f04d	input.test.d95edab2-8f40-4b67-9fd7-01565512228f.raw	{"run_number": 3, "files": 10}	DELETED

(б) Обновленная информация о наборе.

Рисунок 7.14 — Проверка изменения статуса набора после отработки процедуры, удаляющей датасеты.

7.6 МОНИТОРИНГ ХРАНИЛИЩА НА ПРЕДМЕТ ТЕМНЫХ ФАЙЛОВ

В отличие от предыдущего пункта, «темные» файлы удаляются не по заявкам, а по истечении срока их хранения. Если в течение 24 часов файлы не были зарегистрированы в системе, то они удаляются из хранилища и из базы данных.

(а) Лог с информацией об обнаружении «тёмного» файла.

(б) Информация о «тёмном» файле в БД.

Рисунок 7.15 — Механизм обнаружения «тёмных» файлов на хранилище.

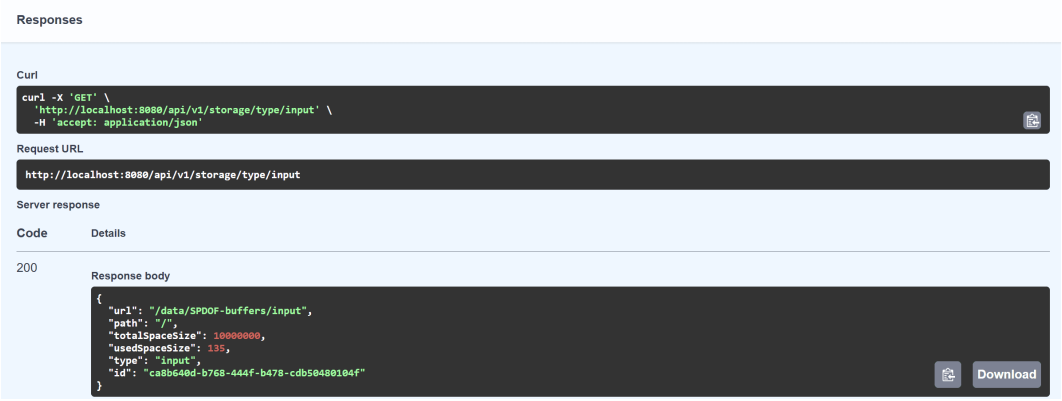
Создадим на хранилище новый файл, но информацию о нем в dsm-register отправлять не будем. Таким образом, файл будет для системы «темным». Сервис по мониторингу хранилища обнаруживает файл и записывает информацию о нем в специальную таблицу (рис. 7.15).

Далее, по прошествии 24 часов, файл удаляется из системы и из хранилища.

7.7 МОНИТОРИНГ ЗАПОЛНЕННОСТИ ХРАНИЛИЩА

Последнее, что было сделано, это обновление информации о занятом месте на хранилище.

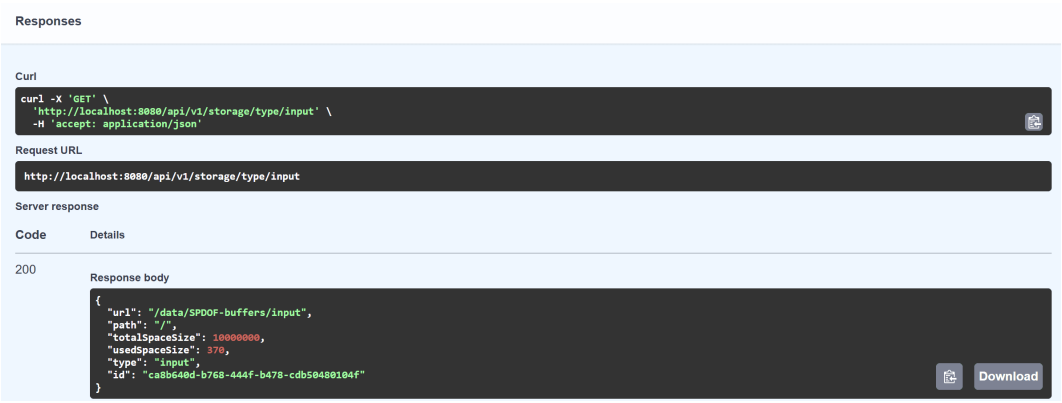
На рисунке 7.16а представлена текущая информация о входном хранилище. Создадим в нем несколько новых файлов. После работы сервиса в БД появляется обновленная информация о хранилище (рис. 7.16б - 7.16в).



(а) Информация о входном хранилище.

```
monitoring-inspector-1  | 2025-05-20 11:54:10 INFO: HTTP Request: GET http://app:8080/api/v1/storage/ "HTTP/1.1 200 OK" [in /src/
.venv/lib/python3.11/site-packages/httpx/_client.py:1038
monitoring-inspector-1  | 2025-05-20 11:54:10 INFO: HTTP Request: PUT http://app:8080/api/v1/storage/ca8b640d-b768-444f-b478-cdb5
0480104f "HTTP/1.1 200 OK" [in /src/.venv/lib/python3.11/site-packages/httpx/_client.py:1038
monitoring-inspector-1  | 2025-05-20 11:54:10 INFO: input storage uses 370 bytes. [in /src/monitoring_inspector/inspectors/storag
```

(б) Лог с информацией об обновлении данных о хранилище.



(в) Обновленная информация о хранилище.

Рисунок 7.16 — Обновление информации о занятом месте на хранилище.

ЗАКЛЮЧЕНИЕ

При проведении экспериментов на коллайдере чрезвычайно важно решить проблему обработки и хранения огромного объема данных, получаемого в результате столкновений.

В связи с этим в контексте данной работы рассмотрен вычислительный комплекс SPD Online Filter, который предназначен для быстрой реконструкции событий и сокращения объема выходных данных без потери их физической ценности. Это является важным шагом в обеспечении эффективной обработки и анализа экспериментальных данных.

Немаловажным является построение надежной системы. Под надежностью здесь понимается способность безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью. При этом недежное программное обеспечение не исключает возникновение в нем ошибок, а лишь сводит их появление к минимуму.

Система, разбитая на несколько небольших частей, более надежна по сравнению с одним большим приложением. Таким образом, SPD Online Filter разбит на три системы, каждая со своей зоной ответственности.

В работе была рассмотрена система управления данными, которая обеспечивает контроль над жизненным циклом данных. Она, в свою очередь, разбита на три микросервиса: dsm-register, dsm-manager и dsm-inspector. Dsm-register принимает заявки на добавление/удаление данных в системе, dsm-manager предоставляет доступ к каталогу данных, а dsm-inspector отвечает за мониторинг состояния хранилища.

Цель данной работы – построить надежную систему. Система управления данными является неотъемлемой частью онлайн-фильтра и без нее была бы невозможна корректная работа всего приложения.

Оценить, является ли программное обеспечение надежным, можно при его испытании путем тестирования или при практическом применении. В связи с этим было важно провести интеграцию всех компонентов онлайн-фильтра. В

системе управления данными за взаимодействие с другими системами отвечают сервисы dsm-register и dsm-manager.

В ходе работы была реализована основная часть функционала сервисов dsm-register и dsm-manager. Также была реализована значительная часть сервиса dsm-inspector.

Помимо этого, было проведено первое нагрузочное тестирование всего промежуточного ПО комплекса SPD Online Filter. На данный момент одновременно работают 100 pilot-ов, которые непосредственно занимаются обработкой файлов. При таком раскладе одна задача выполняется за 15 секунд (одна задача соответствует двум файлам). В ходе тестирования все операции с данными были выполнены успешно, что подтверждает работоспособность разработанной системы.

В дальнейшем планируется:

1) Для сервиса dsm-register реализовать обработку сообщений из очереди, предназначенной для приема заявок на выгрузку данных;
2) Реализовать механизм выгрузки файлов во внешнее хранилище в сервисе dsm-inspector;
3) Закончить этап интеграционного тестирования между системами.

СПИСОК ЛИТЕРАТУРЫ

1. A Measurement of the Spin Asymmetry and Determination of the Structure Function g(1) in Deep Inelastic Muon-Proton Scattering / J. Ashman [и др.] // Phys. Lett. B / под ред. V. W. Hughes, C. Cavata. — 1988. — Т. 206. — С. 364.
2. Conceptual design of the Spin Physics Detector / V. M. Abazov [и др.]. — 2021. — arXiv: 2102.00442 [hep-ex].
3. Комплекс NICA. — URL: https://nica.jinr.ru/ru/complex.php (дата обр. 29.05.2025).
4. *Voronyuk V.*, *Tsegelnik N. S.*, *Kolomeitsev E. E.* Hyperon global polarization in heavy-ion collisions at energies available at the JINR Nuclotron-based Ion Collider fAcility: Feed-down effects and the role of Σ0 hyperons // Phys. Rev. C. — 2025. — Т. 111, № 3. — С. 034907. — arXiv: 2305.10792 [nucl-th].
5. *Iritani T.*, *Cossu G.*, *Hashimoto S.* Partial restoration of chiral symmetry inside hadrons // AIP Conf. Proc. / под ред. A. Andrianov [и др.]. — 2016. — Т. 1701, № 1. — С. 100009.
6. Prospects of studying the production of hypernuclei in heavy-ion interactions at the NICA collider at JINR / V. Kireyeu [и др.] // PoS. — 2022. — Т. PANIC2021. — С. 220.
7. *Zinchenko A.* Techniques for Reconstruction of Strange Objects at MPD // Particles. — 2021. — Т. 4. — С. 178—185.
8. On the physics potential to study the gluon content of proton and deuteron at NICA SPD / A. Arbuzov [и др.] // Prog. Part. Nucl. Phys. — 2021. — Т. 119. — С. 103858. — arXiv: 2011.15005 [hep-ex].
9. Экспериментальная установка SPD. — URL: https://nica.jinr.ru/ru/projects/spd.php (дата обр. 29.05.2025).
10. The MINOS Near Detector front end electronics / T. Cundiff [и др.] // Nuclear Science, IEEE Transactions on. — 2006. — Т. 53. — С. 1347—1355.

11. A new scheduling algorithm for the LHCb upgrade trigger application / E. Govorkova [и др.] // Journal of Physics: Conference Series. — 2020. — Т. 1525. — С. 012052.

12. LHCb Trigger and Online Upgrade Technical Design Report. — 2014.

13. ALICE Central Trigger System for LHC Run 3 / J. Kvapil [и др.] // EPJ Web of Conferences / под ред. C. Biscarat [и др.]. — 2021. — Т. 251. — С. 04022.

14. Technical Design Report of the Spin Physics Detector at NICA / V. Abazov [и др.] // Natural Sci. Rev. — 2024. — Т. 1. — С. 1. — arXiv: 2404.08317 [hep-ex].

15. *Oleynik D.* SPD Online filter. Первичная обработка экспериментальных данных эксперимента SPD. — URL: https://indico.jinr.ru/event/4273/contributions/24346/attachments/17880/30442/19-10_05_JINR_IT_school_SPD_Online_Filter_2023_.pdf (дата обр. 02.06.2025).

16. *Buncic P.*, *Krzewicki M.*, *Vande Vyvre P.* Technical Design Report for the Upgrade of the Online-Offline Computing System. — 2015.

17. *Collaboration L.* Computing and software for LHCb Upgrade II. — 2025. — arXiv: 2503.24106 [hep-ex].

18. *Barisits M.*, *Beermann T.*, *Berghaus F.* Rucio: Scientific Data Management // Computing and Software for Big Science. — 2019. — Т. 3, № 1.

19. О микросервисной архитектуре простыми словами. — URL: https://skillbox.ru/media/code/o-mikroservisnoy-arkhitekture-prostymi-slovami/ (дата обр. 02.06.2025).

20. Микросервисная архитектура: что это, кому подойдёт, с чего начать. — URL: https://yandex.cloud/ru/blog/posts/2022/03/microservice-architecture?utm_referrer=https%3A%2F%2Fyandex.ru%2F (дата обр. 02.06.2025).

21. 26 основных паттернов микросервисной разработки. — URL: https://cloud.vk.com/blog/26-osnovnyh-patternov-mikroservisnoj-razrabotki/ (дата обр. 02.06.2025).

22. REST API: для чего нужен и как работает. — URL: https://yandex.cloud/ru/docs/glossary/rest-api?utm_referrer=https%3A%2F%2Fyandex.ru%2F (дата обр. 02.06.2025).

23. PostgreSQL 12.15 Documentation. — URL: https://www.postgresql.org/docs/12/index.html (дата обр. 02.06.2025).

24. Связи между таблицами базы данных. — URL: https://habr.com/ru/articles/488054/ (дата обр. 02.06.2025).

25. Триггеры в PostgreSQL: основы. — URL: https://habr.com/ru/companies/otus/articles/857396/ (дата обр. 02.06.2025).

26. Партиционирование. — URL: https://docs.arenadata.io/ru/ADB/current/concept/data-model/tables/partitioning.html (дата обр. 02.06.2025).

27. RabbitMQ Documentation. — URL: https://www.rabbitmq.com/docs (дата обр. 02.06.2025).

28. FTS3 Documentation. — URL: https://fts3-docs.web.cern.ch/fts3-docs/docs/overview.html (дата обр. 02.06.2025).

29. Understanding APIs... Everything you need to know about APIs. — URL: https://medium.com/star-gazers/understanding-apis-everything-you-need-to-know-about-apis-b0bf53db6adf (дата обр. 02.06.2025).

30. Object-Relational Mapping (ORM) Explained with Examples. — URL: https://altexsoft.medium.com/object-relational-mapping-orm-explained-with-examples-ade460cd48a6 (дата обр. 02.06.2025).

31. SQLAlchemy 2.0 Documentation. — URL: https://docs.sqlalchemy.org/en/20/intro.html (дата обр. 02.06.2025).

32. Alembic 1.15.2 documentation. — URL: https://alembic.sqlalchemy.org/en/latest/ (дата обр. 02.06.2025).

33. Что такое FastAPI и зачем он нужен. — URL: https://practicum.yandex.ru/blog/fastapi-chto-eto-i-zachem-nuzhen/ (дата обр. 02.06.2025).

34. RabbitMQ: что это такое и как работает. — URL: https://blog.skillfactory.ru/rabbitmq-chto-eto-takoe-i-kak-rabotaet/ (дата обр. 02.06.2025).

35. Dead Letter Queue - System Design. — URL: https://www.geeksforgeeks.org/dead-letter-queue-system-design/ (дата обр. 02.06.2025).

А ПРИЛОЖЕНИЯ

```python
class DatDarkFile(DatBase):
    """
    Table with info of dark files.

    Fields:
        - name - name of dark file
        - path - path to dark file on storage
        - storage_id - identifier of storage
        - size - size of dark file
    """

    __tablename__ = "dat_dark_file"

    name = Column(String(length=255), primary_key=True, index=True)
    path = Column(String(length=255), nullable=False)
    storage_id = Column(
        UUIDType(binary=False), ForeignKey("dat_storage.id"), nullable=False
    )
    size = Column(BigInteger, nullable=False)
```

Листинг 1 — Пример ORM модели.

```
def upgrade() -> None:
    op.create_table(
        "dat_dark_file",
        sa.Column("name", sa.String(length=255), nullable=False),
        sa.Column("path", sa.String(length=255), nullable=False),
        sa.Column(
            "storage_id",
            sqlalchemy_utils.types.uuid.UUIDType(binary=False),
            nullable=False,
        ),
        sa.Column("size", sa.Integer(), nullable=False),
        sa.Column(
            "created_date",
            sa.DateTime(timezone=True),
            server_default=sa.text("now()"),
            nullable=False,
        ),
        sa.Column(
            "updated_date",
            sa.DateTime(timezone=True),
            server_default=sa.text("now()"),
            nullable=False,
        ),
        sa.ForeignKeyConstraint(
            ["storage_id"],
            ["dat_storage.id"],
        ),
        sa.PrimaryKeyConstraint("name"),
    )
```

Листинг 2 — Скрипт миграции для таблицы с темными файлами.

```
class DatDarkFileRepository(BaseSqlRepository):
    """Repository of dat_dark_file."""

    def __init__(
        self, session_factory: Callable[..., AbstractContextManager[Session]]
    ) -> None:
        super(DatDarkFileRepository, self).__init__(session_factory, DatDarkFile)
        self.session_factory = session_factory

    def get_by_name(self, file_name: str) -> DatDarkFile:
        """Get dark file by name."""
        with self.session_factory() as session:
            entity = (
                session.query(DatDarkFile)
                .filter(DatDarkFile.name == file_name)
                .first()
            )
            if not entity:
                raise NotFoundException(
                    ENTITY_NAME_NOT_FOUND_ERROR.format(
                        entity_name=self.entity_class.__name__, name=file_name
                    )
                )
            return entity

    def delete_by_name(self, entity_name: str) -> None:
        """Delete dark file by name."""
        with self.session_factory() as session:
            entity: self.entity_class = (
                session.query(self.entity_class)
                .filter(self.entity_class.name == entity_name)
                .first()
            )
            if not entity:
                raise NotFoundException(
                    ENTITY_NAME_NOT_FOUND_ERROR.format(
                        entity_name=self.entity_class.__name__, name=entity_name
                    )
                )
            session.delete(entity)
            session.commit()
```

Листинг 3 — Пример репозитория для темных файлов.

```
def __rabbit_bootstrap_consume_queues(
    channel: Channel, settings: RabbitSettings
) -> None:
    """Initialize rabbit consume queues."""
    general_args: dict = {RABBIT_DLX_HEADER: settings.dead_letter_exchange()}
    channel.queue_declare(
        queue=settings.queue_file_process(),
        durable=True,
        arguments={
            **general_args,
            RABBIT_DLQ_ROUTING_KEY_HEADER: settings.CONSUMING_ROUTING_KEY_FILE_PROCESS,
        },
    )
    channel.queue_bind(
        exchange=settings.CONSUMING_EXCHANGE,
        queue=settings.queue_file_process(),
        routing_key=settings.CONSUMING_ROUTING_KEY_FILE_PROCESS,
    )
```

Листинг 4 — Построение очереди для промежуточных файлов в RabbitMQ.

```
class FileProcessProcessor:
    ...
    def process(self, message: FilesProcessDto) -> None:
        logger.info("Processing msg")
        ...
        for file_info in message.files:
            try:
                file: FileResponse = self._file_manager.create_file(
                    FileRequest(
                        name=get_file_name(file_info.path),
                        path=get_path(file_info.path),
                        storage_id=file_info.storage_id,
                        size=file_info.size,
                        check_sum=file_info.check_sum,
                        dataset_ids=[message.dataset_id],
                        status_code=FileStatus.CREATED.code,
                    )
                )
            except httpx.RequestError as exc:
                error_msg = FileProcessReplyDto(
                    status=ReplyStatus.ERROR.code,
                    details=f"Error occurs while registering file = {file_info.path}. "
                    + f"HTTP Exception for {exc.request.url} - {exc}",
                )
                self._file_process_reply.send(error_msg)
            else:
                logger.info(
                    f"Registered file ID={file.id} in dataset ID={message.dataset_id}"
                )
                success_msg = FileProcessReplyDto(
                    status=ReplyStatus.SUCCESS.code,
                    details="Registered file = " + file_info.path,
                )
                self._file_process_reply.send(success_msg)

            try:
                dark_file: DarkFileResponse = self._dark_file_manager.get_file(file.name)
                if dark_file.path == file.path and dark_file.size == file.size:
                    self._dark_file_manager.delete_file(dark_file.name)
                    logger.info(f"File {dark_file.name} is no more dark.")
            except ValidationError:
                pass
```

Листинг 5 — Часть функции класса, отвечающего за обработку сообщений из очереди для промежутоных данных.

```python
class FileDeleteInspector:

    def __init__(
        self,
        dataset_manager: DatasetManager,
        file_manager: FileManager,
    ) -> None:
        self._dataset_manager = dataset_manager
        self._file_manager = file_manager


    def form_file_list_to_delete(self) -> list[FileResponse]:
        """Form file list with status TO_DELETE"""
        logger.info(f"Form file list with status TO_DELETE")
        return self._file_manager.get_files_by_status(
            FileStatus.TO_DELETE.code
        )

    def delete_file(self, file: FileResponse) -> None:
        """Delete file from storage"""

        if os.path.exists(f"{file.url}{file.path}/{file.name}"):
            os.remove(f"{file.url}{file.path}/{file.name}")
            logger.info(f"File {file.name} DELETED!")
        else:
            logger.info(f"File ID={file.id} doesn't exist!")

        dataset: DatasetResponse = self._dataset_manager.get_datasets_by_file_id(file.id)[0]

        self._file_manager.update_file(
            file.id,
            FileRequest(
                name=file.name,
                path=file.path,
                storage_id=file.storage_id,
                size=file.size,
                check_sum=file.check_sum,
                dataset_ids=[dataset.id,],
                status_code=FileStatus.DELETED.code,
            )
        )

        logger.info(f"File ID={file.id} mark as DELETED")
```

Листинг 6 — Класс, отвечающий за удаление файлов.