МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВЕРО-ОСЕТИНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. К.Л. ХЕТАГУРОВА»

Факультет: Физико-технический

Кафедра: Физики конденсированного состояния

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка каталога физических событий эксперимента SPD на ускорители частиц NICA: создание программного и пользовательского интерфейса с использованием RESTful технологий»

Исполнитель:
Бакалавр, 4 курс, очная форма обучения,
Направление подготовки: 03.03.02 Физика
Гурциев Ричард Зурабович

Научный руководитель:
к.ф.-м.н., доцент кафедры физики конденсированного состояния
Тваури Инга Васильевна

Научный консультант:
к.ф.-м.н., с.н.с. Лаборатории ядерных проблем ОИЯИ
Прокошин Федор Валерьевич

«Допущен к защите»

Зав. кафедрой_____(профессор, доктор физико-математических наук, заведующий кафедрой физики конденсированного состояния Тамерлан Таймуразович Магкоев)

«_____»_____2024г

Владикавказ 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Эксперимент SPD представляет собой научно-исследовательский проект, направленный на изучение физики элементарных частиц и поиск новых фундаментальных знаний и структуре материи и взаимодействии ее элементов. Целью этого проекта является разработка и создание экспериментальной установки SPD на ускорителе частиц NICA и анализ полученных в ходе эксперимента данных.

NICA (Nuclotron-based Ion Collider fAcility) – это новый ускорительный комплекс, который создается на базе Объединенного института ядерных исследований в городе Дубна Московской области. Его основным объектом является кольцевой ускоритель частиц (коллайдер) на котором будут сталкиваться тяжелые ядра (до AU включительно) при энергиях 4 - 11 ГэВ либо поляризованные протоны и дейтроны с энергиями до 27 ГэВ. На коллайдере будут расположены два детектора: MPD (Multi-Purpose Detector) и SPD (Spin Physics Detector). В данный момент идет строительство ускорительного комплекса и разработка физических и инженерных решений для различных частей детекторных установок.

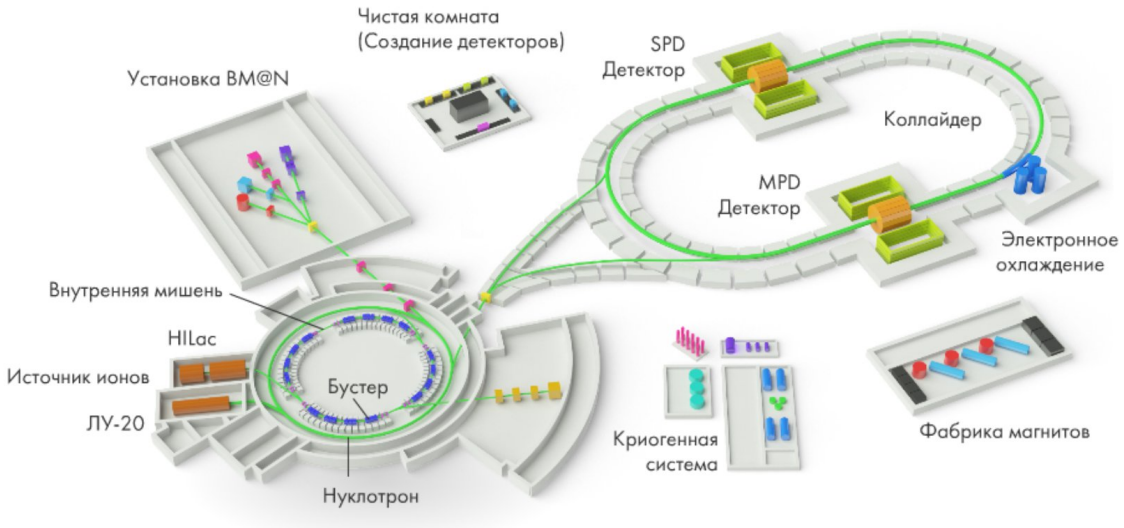


Рисунок 1. Ускорительный комплекс NICA

Для обработки и хранения больших объемов данных, полученных в ходе эксперимента SPD, требуется создание сложных информационных систем. Одним из таких элементов должен стать «EventIndex» - каталог физических событий, полученных от детектора или смоделированных для последующего анализа и обработки данных.

В ходе реализации эксперимента SPD планируется набрать до триллиона событий (записей результатов столкновений), для хранения и обработки которых потребуются сотни петабайт данных. Эта информация будет распределена между несколькими хранилищами данных в компьютерных центрах. Для исключения потери данных и повышения производительности записи, относящиеся к одному событию, будут дублироваться. Для эффективного доступа ко всем экземплярам событий необходима информационная система, а именно - разрабатываемый SPD EventIndex, каталог всех событий, полученных от детектора или смоделированных, постоянно хранящихся во всех форматах и версиях.

SPD EventIndex разрабатывается как комплексная информационная система, которая должна обеспечить:

- получение информации о событиях эксперимента и моделированных данных путем индексирования файлов данных, содержащих информацию о этих событиях;
- передачу этой информации и запись в базы данных;
- доступ к информации для программ обработки и анализа данных через API и приложения;
- доступ к информации пользователям через интерактивные и асинхронные интерфейсы.

На данном этапе создания эксперимента SPD информационные и вычислительные системы находятся в начальной стадии разработки. Создание EventIndex планируется начать с частей, которые не зависят от других компонент.

Объектом выпускной квалификационной работы является каталог физических событий эксперимента SPD на ускорители частиц NICA.

Предметом выпускной квалификационной работы является программное обеспечение для каталога физических событий с использованием Restful технологий.

Целью работы является разработка программного обеспечение для каталога физических событий.

В соответствии с целью определены следующие задачи:

1. Проанализировать исходные данные и создать архитектуру сервиса.
2. На основе выбранной архитектуры разработать сервис, уделяя внимание его визуальным и функциональным компонентам.

Результаты проведённых исследований были представлены в виде доклада на 10-ой Международной конференции “Распределенные вычисления и GRID технологии в науке и образовании” 2023, Весенней школе по информационным технологиям ОИЯИ 2024, а также опубликованы в статье “SPD EventIndex” в журнале “Физика элементарных частиц и атомного ядра” [1, 2, 3]. Журнал «Физика элементарных частиц и атомного ядра» входит в перечень рецензируемых научных изданий ВАК России, рекомендованных для публикаций статей, содержащих материалы кандидатских и докторских диссертаций.

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ ПРОЕКТА SPD EVENTINDEX

1.1. Проект «SPD EventIndex»

Система EventIndex разрабатывается для предоставления глобального каталога событий и ограниченных метаданных на уровне событий для эксперимента SPD на сверхпроводящем коллайдере NICA. Установка SPD представляет собой масштабную систему детекторов, способных регистрировать и анализировать различные события, возникающие при столкновениях ядер и элементарных частиц с высокой энергией. Создание установки планируется завершить к моменту запуска ускорительного комплекса в режиме поляризованных пучков (планируется в 2028 году). По ходу реализации проект адаптируется в соответствии с развитием детекторных технологий и средств обработки и анализа данных [2].

В ходе эксперимента SPD предположительно будут получены сотни петабайт данных, и ожидается, что в будущем их будет на порядок больше. Эти данные будут распределены по множеству серверов в компьютерных сетях ОИЯИ и других вычислительных центров. «EventIndex» будет хранить информацию о расположении основных элементов этих данных: реальных и смоделированных событий. Он предоставляет средства для выбора данных о событиях в распределенной системе хранения SPD и доступа к ним, обеспечивает поддержку проверок полноты и согласованности данных а также обнаружение дублированных событий. Проект должен быть интегрирован с другими информационными системами SPD, включая менеджеры инфраструктуры распределенных вычислений и хранения данных (Rucio и PanDA), а также ИС метаданных. На текущий момент основным хранилищем данных EventIndex выбрана СУБД PostgreSQL, возможно использование другой платформы если это окажется целесообразным [16].

1.2. Идея реализации каталога событий эксперимента SPD

При реализации каталога событий эксперимента SPD предполагалось широкое использование опыта, накопленного при разработке аналогичных систем для существующих экспериментов. В частности, в эксперименте ATLAS на ускорителе LHC используется аналогичный каталог событий — система «ATLAS EventIndex». Каждый год в ходе эксперимента генерировалось около 7 миллиардов событий. Записи о событиях хранились в файлах, расположенных на узлах распределенных вычислительных системах CERN. Наличие такого огромного объема информации определило необходимость создания глобального каталога, который позволил бы определить местоположение каждой записи для обработки и анализа [1].

В эксперименте SPD ожидаются объемы данных того же порядка, что и в ATLAS, с еще большим количеством событий. Несмотря на различия в задачах, поставленных в этих экспериментах, их системы обработки данных имеют схожие характеристики, что позволяет использовать аналогичные решения. Предполагается, что SPD будет использовать схожую модель данных и файловую организацию, а также распределенную систему хранения и обработки данных. Группы статистически эквивалентных событий хранятся в файлах на диске или магнитной ленте. Каждый файл обычно содержит от 1000 до 10000 событий, в зависимости от формата. Файлы сгруппированы в наборы данных, обычно содержащие события, относящиеся к одному сеансу сбора данных (Run). Основным отличием экспериментов является отсутствие триггера, вместо этого первоначальный отбор данных будет осуществляться с помощью онлайн-фильтра, основанного на методах машинного обучения.

В ходе разработки стало ясно, что дальнейшая реализация SPD EventIndex требует интеграции с другими ИС установки. Это проистекает из опыта разработки аналогичной системы «ATLAS EventIndex».

1.3. Отбор событий в эксперименте

Основная цель проекта SPD - проведение экспериментов по изучению спиновой структуры нуклонов и других процессов, зависящих от спина, путем измерений асимметрий в процессе Дрелла-Яна, процессы рождения J/Ψ частиц и прямых фотонов в столкновениях поляризованных протонов и дейтронов. В процессе анализа данных экспериментов в физике частиц важную роль играет отбор событий в которых могут проявляться физические процессы интересующие исследователя [5].

Отбор производится в несколько этапов. Непосредственно при наборе данных происходит фильтрация сигналов с детекторов по амплитудным и временным параметрам, затем данные поступают на этапе онлайн отбора. Это могут быть либо триггерные системы, отбирающие детекторные объекты и события в целом до записи в системы хранения, либо, при бестриггерной организации системы набора данных онлайн фильтры анализирующие данные за определенные временные промежутки. В обоих случаях данные проверяются на соответствие определенным наборам критериев и результаты проверки сохраняются для дальнейшего использования.

Как правило, экспериментальные данные используются для нескольких физических анализов, для которых представляют интерес разные физические процессы с разной вероятностью проявления в условиях эксперимента. Отбор «сигнальных» событий которые могут содержать такие процессы — сложная задача, составляющая существенную часть работы в рамках физического анализа. В конкретном анализе из миллионов событий могут быть отобраны как относительно большие наборы десятки и сотни тысяч, так и достаточно мелкие десятки и сотни. Данные об этих событиях могут храниться среди множества файлов, разбросанных по серверам распределенных систем хранения. Поиск набора событий путем сканирования петабайт данных будет требовать больше затрат вычислительных мощностей. В ходе анализа доступ к этим событиям может понадобится множество раз.

1.4. Архитектура и операции проекта

EventIndex имеет секционированную архитектуру, соответствующую потоку данных (Рис. 2). Для получения реальных данных индексируются все наборы данных, содержащие события в формате RAW (результаты реконструкции), и некоторые наборы данных в формате AOD (производные RAW, т.е. выбранные события с сокращенной информацией для конкретных анализов). Информация об индексации, извлеченная из наборов данных RAW, также содержит ссылки на соответствующие необработанные данные, поэтому впоследствии можно извлекать события также в формате необработанных данных. Для смоделированных данных индексируются все выходные данные генератора событий, а также все наборы данных в форматах RAW и некоторые из них в форматах AOD, аналогично реальным данным. Правила по которым будут отбираться типы данных для индексирования будут уточнятся в ходе подготовки и проведения эксперимента [16].

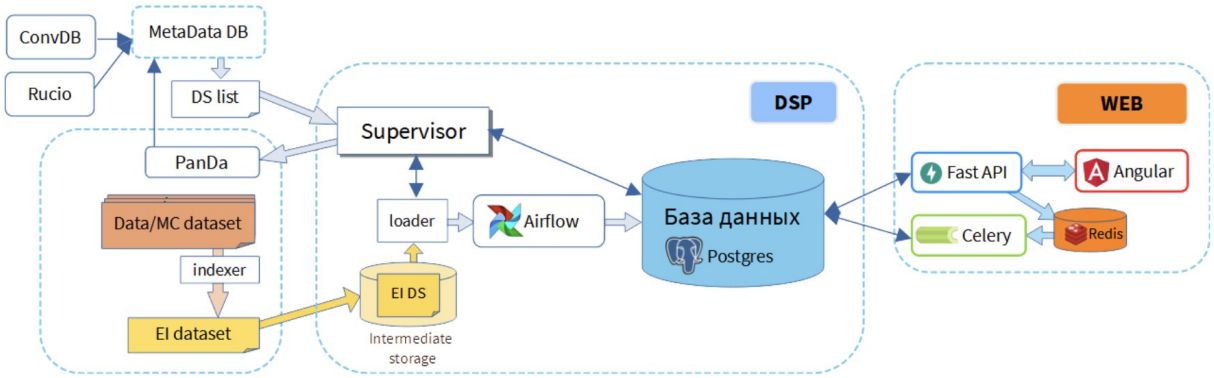


Рисунок 2. Архитектура системы «SPD EventIndex»

Для автоматического индексирования новых датасетов предлагается следующая процедура: При появлении новых датасетов БД метаданных передает супервизору список новых датасетов. Супервизор осуществляет проверку и дополнительную фильтрацию списка, и запускает программы индексации отобранных датасетов через систему распределенных вычислений PanDA. PanDA предоставляет средства для отслеживания выполнения задач

сбор логов а также может производить автоматический перезапуск в случае системных сбоев. Супервизор отслеживает выполнение задач, в случае неуспешного завершения задачи отрабатывает возникшие ошибки, перезапускает задачи при необходимости. При успешном завершении задачи, производит анализ лога задачи и отработка ошибок в ходе выполнения программы [3].

Полученная в результате работы программы записывается в новый датасет. Супервизор запускает средствами Rucio и FTS передачу датасета с индексированными данными с удаленного сервера на сервер EventIndex. Если передача прошла успешно – полученные данные ставятся в очередь на запись в БД. При успешной записи данных в БД супервизор сообщает ИС метаданных об успешной проверке данных и о количестве индексированных событий, в противном случае датасет помечается в ИС метаданных как дефектный [17].

ВЫВОДЫ ПО ПЕРВОЙ ГЛАВЕ

Была рассмотрена система эксперимента «SPD EventIndex» и его цели. Проект планируется завершить в 2028 году. При наличии набора детекторов на коллайдере NICA, необходимых для регистрации и анализа информации о различных событий, создающих в результате столкновений атомных ядер и элементарных частиц с высокой энергией, возникает необходимость эффективного хранения такого объема данных о событиях и обеспечения доступа к этим данным через интерфейс пользователя. Именно это стало основной причиной для начала реализации проекта «SPD EventIndex».

«EventIndex» — это система, предназначенная для индексации данных, хранение информации о событиях и обеспечения доступа к этим данным посредством программных интерфейсов и интерфейсов пользователя.

Разрабатываемая для эксперимента SPD система будет имеет ряд областей применения:

- подсчет и предварительный отбор событий для анализа по результатам работы онлайн-фильтра и набора важных параметров события (каких именно — предстоит определить группе физического анализа);
- доступ по идентификаторам к наборам событий, полученным по результатам процедуры отбора в рамках физических анализов;
- создание «виртуальных датасетов» из отобранных событий;
- проверка целостности наборов события в датасетах;
- обнаружение не читаемых/поврежденных либо потерянных событий;
- обнаружение дублирующихся событий внутри файлов и датасетов.

ГЛАВА 2. ОПИСАНИЕ РАЗРАБОТАННОГО СЕРВИСА

2.1. Архитектура сервиса

Архитектура — это структурное описание системы или приложения, включающее в себя компоненты системы, их свойства, а также отношения и взаимодействия между этими компонентами. В контексте программного обеспечения, архитектура определяет общий дизайн инфрастуктуры, включая распределение функциональности между компонентами, способы связи между ними, а также основные принципы, лежащие в основе ее построения [4].

В ходе реализации проекта была определена архитектура, включающая в себя выбор оптимальных технологий, а также определение основных принципов организации системы для создания высокопроизводительной технологий. Типичная структура веб-сервиса (Рис.3) состоит из трех главных компонент: Frontend, Backend и DevOps. Далее более подробно будут рассмотрены эти компоненты, но предварительно рассмотрим основные данные, с которым пользователь будет взаимодействовать.

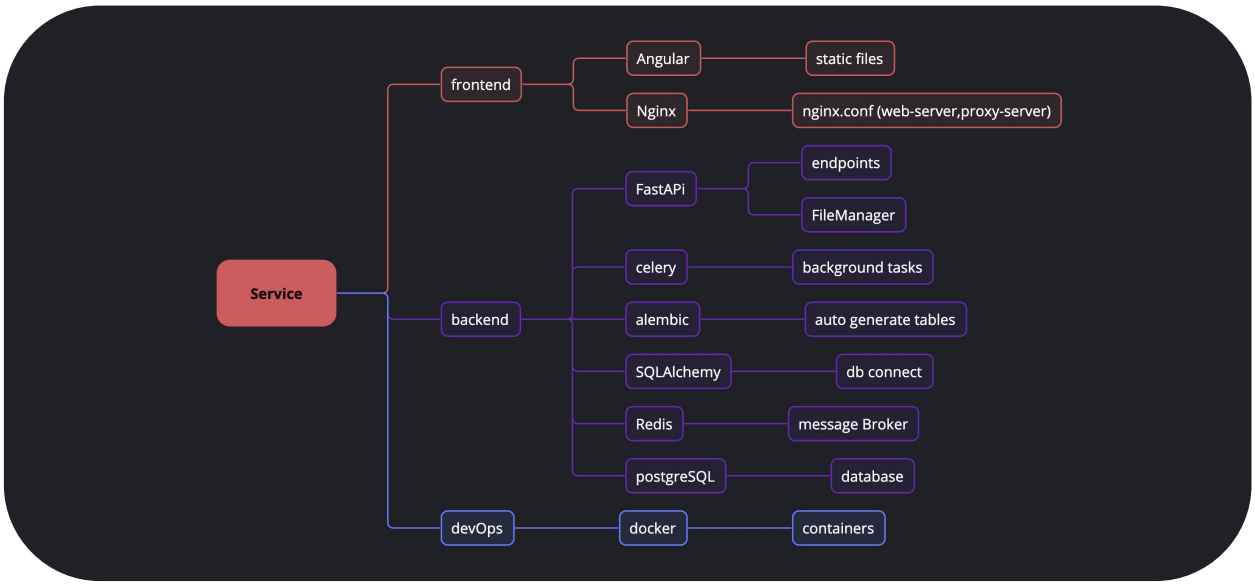


Рисунок 3. Архитектура сервиса

2.2. Описание основных данных сервиса

При разработке приложения учитывают данные с которыми пользователь непосредственно взаимодействует. Помимо этого, стоит заранее выбрать систему управления базами данных (СУБД), которая будит хранить всю информацию, предоставляемая сервисом, по нескольким параметрам:

1. Масштабируемость: Система должна быть способна масштабироваться с увеличением объема данных и нагрузки. Это может потребовать выбора соответствующей архитектуры базы данных и использование механизмов масштабирования, таких как репликация и шардинг.
2. Безопасность данных: Важно обеспечить защиту данных от несанкционированного доступа, взломов и утечки информации. Это включает в себя управление доступом, шифрование данных и другие меры безопасности.
3. Управление данными: СУБД данных должна обеспечить эффективное хранение, поиск, обновление и удаление данных. Разработка соответствующих запросов и процедур хранения может значительно повысить производительность и надежность сервера.

На основание предыдущих рассуждений, было принято решение использовать PostgreSQL для управления базой данных EventIndex. Это свободная объектно-реляционная система управления базами данных, которая предоставляет надежное хранение данных, поддерживает широкий спектр функций, включая сложные запросы, индексы, триггеры и многое другое. Она является одной из самых популярных и широко используемых СУБД, особенно в разработке веб-приложений [11].

Основными таблицами в системе, где хранятся параметры данных, являются events и datasets. Для организации работы интерфейса добавлены таблицы users, verify и pipelines. В дальнейшем, они последовательно будут подробно рассмотрены.

Таблица событий содержит подробное данные, описывающие события. Текущие параметры каталога:

- id — индекс каждой строки;
- run_number — номер run;
- event_number — номер события в run;
- type — тип события;
- version — версия события;
- dsid — идентификатор датасета, связывающая с таблице datasets;
- fuid_raw — UUID для файла в котором хранится событие.

id	run_number	event_number	type	version	dsid	fuid_raw
1	280308124	127	RAW	t0001	1	057ff062-4d76-4283-ab8f-abcc45d1ba2f
2	280308124	33	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
3	280308124	35	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
4	280308124	332	RAW	t0001	1	047fd062-2d76-4283-ab8f-accc45d1ba2f
5	280308124	338	RAW	t0001	1	047fd062-2d76-4283-ab8f-accc45d1ba2f
6	280308124	342	RAW	t0001	1	047fd062-2d76-4283-ab8f-accc45d1ba2f
7	280308124	371	RAW	t0001	1	047fd062-2d76-4283-ab8f-accc45d1ba2f
8	280308124	311	RAW	t0001	1	047fd062-2d76-4283-ab8f-accc45d1ba2f
9	280308124	20	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
10	280308124	21	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
11	280308124	290	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
12	280308124	190	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661
13	280308124	184	RAW	t0001	1	52c90fe2-bf0b-4b23-9856-098e27c35661

Рисунок 4. Поля таблиц «events»

Стоит более подробно рассмотреть fuid_raw, поскольку к этому параметру будем непосредственно возвращаться. Это универсальный идентификатор для файла позволяющий получить к нему доступ в системе распределенного хранения данных Rucio. Каждый фаил содержит от 1 тыс. до 10 тыс. событий.

Важной таблицей также является datasets, в которой хранятся информация о датасетах, содержащих события, принадлежащие таблице events.

В этой таблице содержатся такие параметры как:

- id — индекс каждого датасета;
- dsn — наименование датасета;
- date —дата добавления датасета.

dsid	dsn	date
2	data28_pp_9p4GeV.280308061.RAW.t0001	2024-03-22 12:41:12.216749+00
3	data28_pp_9p4GeV.280308042.RAW.t0001	2024-03-22 12:41:26.911751+00
1	data28_pp_9p4GeV.280308124.RAW.t0001	2024-03-22 11:24:35.83516+00
4	data28_pp_9p4GeV.280308124.RAW.t0002	2024-03-22 13:07:20.803789+00
5	data28_pp_9p4GeV.280308124.RAW.t0003	2024-03-22 13:07:30.883915+00
6	data28_pp_9p4GeV.280308124.RAW.t0004	2024-03-22 13:07:33.303554+00
7	data28_pp_9p4GeV.280308061.RAW.t0003	2024-03-22 13:48:44.551638+00
8	data28_pp_9p4GeV.280308061.RAW.t0002	2024-03-22 13:53:36.670303+00

Рисунок 5. Поля таблиц datasets

Таблица «users» (Рис. 5) необходима для хранения метаданных пользователя, по которым можно следить за статусом определенного лица, открывая доступ к дополнительным компонентам.

Основные атрибуты данных являются:

- id — индекс каждого объекта каталога users;
- first_name — имя пользователя;
- last_name — фамилия пользователя;
- username — название пользователя;
- email — почта пользователя;
- hashed_password — закэшированный пароль пользователя;
- role — статус пользователя;
- is_active — значение определяющее активность пользователя;
- is_verified — значение определяющее верифицированность; пользователя;
- created_at — создание объекта;
- updated_at — обновление объекта.

```
 id | first_name | last_name  |  username  |        email        |
     |          updated_at
----+------------+------------+------------+---------------------+
-----+-------------------------------
  2 | Igor       | Igolkin    | igol       | i.igolkin@gmail.com |
1+03 | 2024-01-20 22:05:49+03
 37 | Timur      | Tomsky     | tima       | t.tomsky@gmail.com  |
9+03 | 2024-02-06 09:54:05.680333+03
 35 | David      | Kondratiev | dava       | d.kondr@gmail.com   |
7+03 | 2024-02-06 18:41:23+03
 51 | Misha      | Koval      | m.koval    | m.koval@gmail.com   |
6+03 | 2024-02-10 12:38:47.452539+03
```

Рисунок 6. Первая часть поля таблицы «users»

```
  hashed_password   | role  | is_active | is_verified |          created_at
--------------------+-------+-----------+-------------+-------------------------
 $2b$12$CIawELoLrqIf| admin | t         | t           | 2024-01-04 06:07:20.74282
 $2b$12$GpAaK1QiFx8F| user  | t         | t           | 2024-02-06 09:54:05.68032
 $2b$12$jecNFGCxOCd3| admin | t         | t           | 2024-02-06 08:25:14.50961
 $2b$12$99DDfb8k4uRg| user  | t         | t           | 2024-02-10 12:38:47.45253
 $2b$12$QrALkmskfgq8| user  | t         | t           | 2024-01-31 15:03:36.94327
 $2b$12$PIFGjwcMmLLn| user  | t         | t           | 2024-01-31 15:22:55.89467
```

Рисунок 7. Вторая часть поля таблицы «users»

Таблица «verify» предназначена для хранения проверок верификации пользователя. Исходя из значений параметров, принимается решение о доступе пользователя к сервису или его ограничении.

Каталог содержит такие поля как:

- id — индекс каждого объекта таблицы;
- user_id — идентификатор пользователя;
- code — верификационный код;
- created_at — дата добавления объекта;
- is_active — значение определяющее активность пользователя.

id	user_id	code	created_at	is_active
1	1	785487	2024-01-03 12:50:58.644521+03	f
2	2	914186	2024-01-04 06:07:20.780979+03	f
55	43	303514	2024-02-07 10:08:37.236206+03	f
5	5	861512	2024-01-05 13:33:38.384656+03	f
6	6	418418	2024-01-05 15:55:56.717851+03	f
7	7	747224	2024-01-20 20:35:06.372916+03	f
57	45	971411	2024-02-07 17:06:20.86451+03	f
9	9	601136	2024-01-31 15:03:36.99448+03	f
58	46	367377	2024-02-07 17:07:45.961752+03	f
11	11	218257	2024-01-31 15:22:55.922935+03	f

Рисунок 8. Поля таблицы «verify»

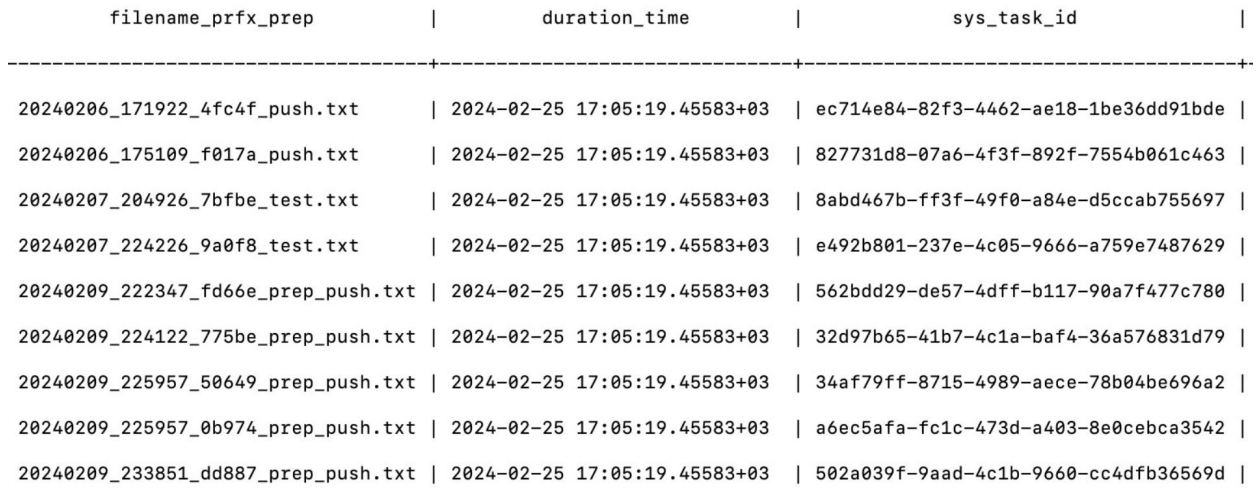
В «pipelines» хранятся данные, относящиеся к файлам, которые разбиваются на задачи и далее классифицируются на исходные и обработанные файлы.

В реляционной базе данных таблицы содержит такие столбцы, как:

- id — индекс каждой строки;
- user_id — идентификатор пользователя;
- status — состояние задачи в процессе выполнения;
- task_id — идентификатор задачи;
- filename — название исходного файла;
- filename_prfx_prep — название обработанного файла с уникальным префиксом;
- duration_time — время выполнения задачи;
- sys_task_id — системный идентификатор для передачи данных.

id	user_id	status	task_id	filename	filename_prfx_raw
492	36	success	15ac5b86-8ee7-4d5f-8b09-bf83b078cf0d	push.txt	c4f_raw_push.txt
530	1	success	db7a078a-6a1e-441b-8567-27149c748b19	push.txt	17a_raw_push.txt
543	49	pending	028f4342-ef87-4c4c-9eef-9dac541a9fb1	test.txt	fbe_raw_test.txt
545	49	success	85c74415-8e75-450c-a1dd-abc748c53177	test.txt	0f8_raw_test.txt
550	1	success	dc6cb617-082e-457a-9ac5-f7a3131a7e5d	push.txt	66e_raw_push.txt
551	1	success	c9f5fe88-cacf-4509-bdfc-7f1137cefcfd	push.txt	5be_raw_push.txt
553	1	success	a5e29d59-8075-4bc4-be1d-ad59d5b1ca5e	push.txt	649_raw_push.txt
555	1	success	10d746dd-29e0-432f-ac53-d35e7887156f	push.txt	974_raw_push.txt
557	50	success	df00c774-4d8a-472c-8c49-787191038358	push.txt	

Рисунок 9. Первая часть поля таблицы «pipelines»



filename_prfx_prep	duration_time	sys_task_id
20240206_171922_4fc4f_push.txt	2024-02-25 17:05:19.45583+03	ec714e84-82f3-4462-ae18-1be36dd91bde
20240206_175109_f017a_push.txt	2024-02-25 17:05:19.45583+03	827731d8-07a6-4f3f-892f-7554b061c463
20240207_204926_7bfbe_test.txt	2024-02-25 17:05:19.45583+03	8abd467b-ff3f-49f0-a84e-d5ccab755697
20240207_224226_9a0f8_test.txt	2024-02-25 17:05:19.45583+03	e492b801-237e-4c05-9666-a759e7487629
20240209_222347_fd66e_prep_push.txt	2024-02-25 17:05:19.45583+03	562bdd29-de57-4dff-b117-90a7f477c780
20240209_224122_775be_prep_push.txt	2024-02-25 17:05:19.45583+03	32d97b65-41b7-4c1a-baf4-36a576831d79
20240209_225957_50649_prep_push.txt	2024-02-25 17:05:19.45583+03	34af79ff-8715-4989-aece-78b04be696a2
20240209_225957_0b974_prep_push.txt	2024-02-25 17:05:19.45583+03	a6ec5afa-fc1c-473d-a403-8e0cebca3542
20240209_233851_dd887_prep_push.txt	2024-02-25 17:05:19.45583+03	502a039f-9aad-4c1b-9660-cc4dfb36569d

Рисунок 10. Вторая часть поля таблицы «pipelines»

Таким образом PostgreSQL для всей инфраструктуры веб-приложения является системным менеджером, позволяющий хранить и обрабатывать огромное количество данных, содержащихся в ранее перечисленных таблицах [11].

2.3. Интерфейс и взаимодействие с пользователем (Frontend)

Фронтенд — это часть веб-приложения, которая отвечает за взаимодействие с пользователем. Это то, что видит и с чем взаимодействует пользователь в браузере, включая дизайн и пользовательский интерфейс. Frontend обычно разрабатывается с использованием HTML, CSS и JavaScript, а также различных фреймворков и библиотек, таких как React, Angular и Vue.js [18].

Основой визуальной части сервиса является платформа — Angular. Это фреймворк от Google, совместимый с большинством распространенных редакторов кода. Angular предназначен для создания динамических одностраничных и прогрессивных веб-приложений с помощью веб-разметки и функциональной части. Фреймворк больше всего ценится за надежную архитектуру, обширную экосистему библиотек и инструментов, позволяющие создавать современные и масштабируемые приложения. Angular является одним из самых популярных фронтенд-фреймворков [8].

В ходе разработки визуальных элементов сервиса были созданы основные компоненты, облегчающие взаимодействие с пользователем. Визуальные аспекты сервиса будут разделены на две отдельные части для удобства ориентирования: «авторизация» и «панель управления». Более подробная информация по этим компонентам будет представлено далее.

2.3.1. Авторизация

Авторизация — это предоставление определенному лицу или группе лиц прав на выполнение определённых действий, а также процесс проверки этих прав при попытке выполнения этих действий. С точки зрения любой информационной системы это процесс принятия решения о предоставлении доступа субъекту на выполнение операции на основании каких-либо знаний о субъекте. К этому моменту субъект, как правило, уже должен быть идентифицирован и аутентифицирован [20].

Рассмотрим основные элементы авторизации, которые охватывают систему, включающую такие компоненты, как регистрация, верификация и аутентификация.

При загрузке сервиса пользователю будет представлена регистрационная форма с несколькими полями для заполнения. После отправки данных на сервер по протоколу HTTP, сервер создаст новую запись в базе данных. Если процесс завершится успешно, пользователь будет зарегистрирован в системе и сможет пройти дальнейшую идентификацию.

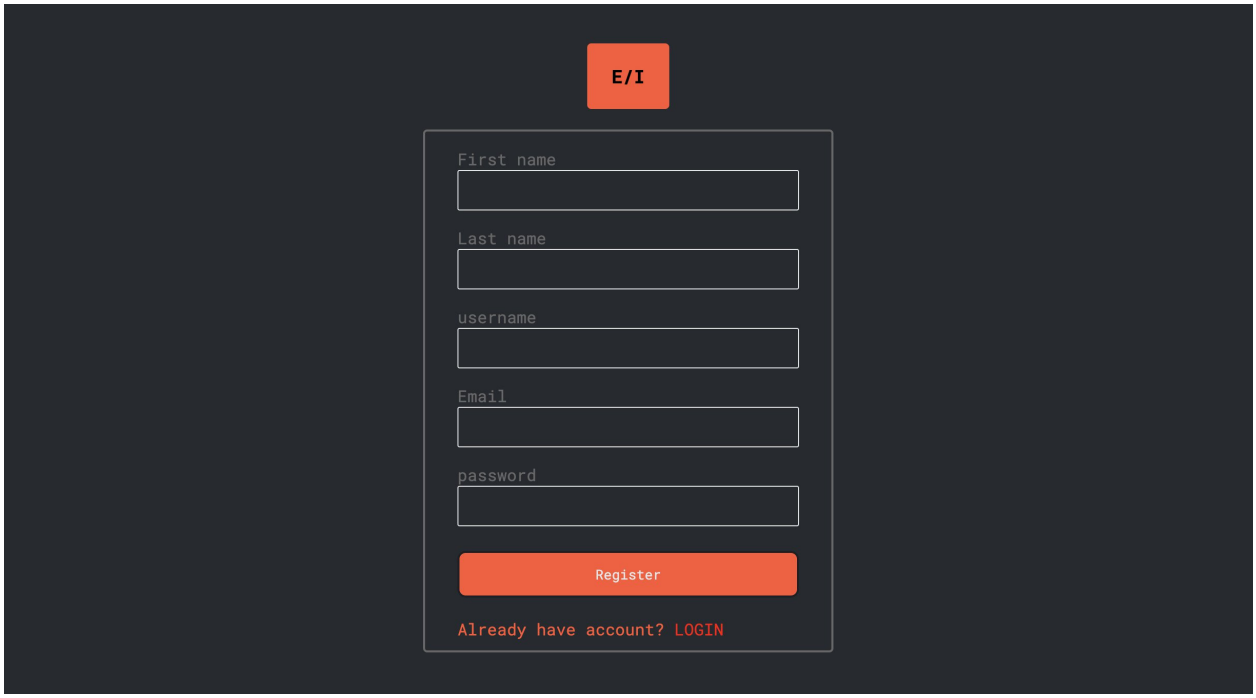


Рисунок 11. Регистрационная часть

Однако выполнения этого шага недостаточно для доступа к услугам, предоставляемым сервисом. Поэтому пользователю необходимо пройти этапы проверки, завершив их подтверждением кода верификации, который будет отправлен на почту по протоколу SMTP.

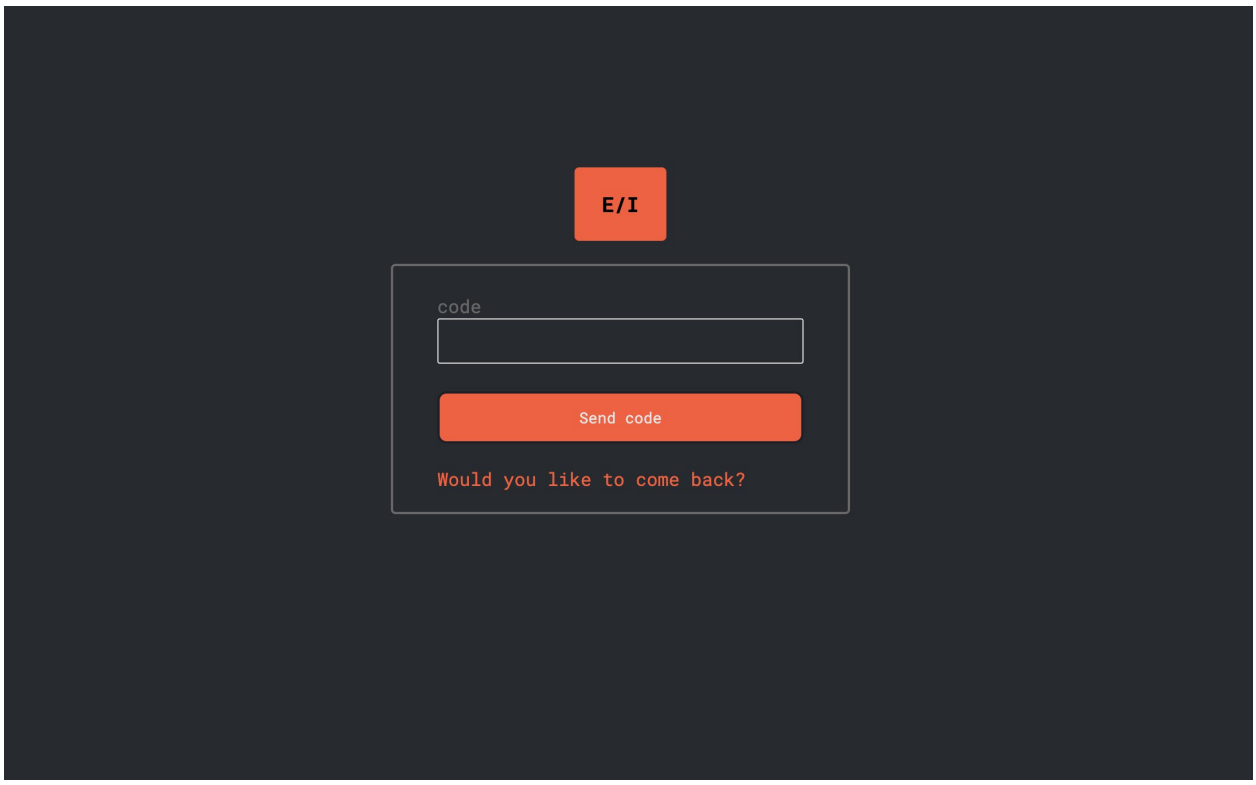


Рисунок 12. Часть верификации

После успешной верификации система перенаправит на страницу аутентификации, где будет предложено ввести метаданные пользователя для доступа к сервисной обслуживаемой зоне.

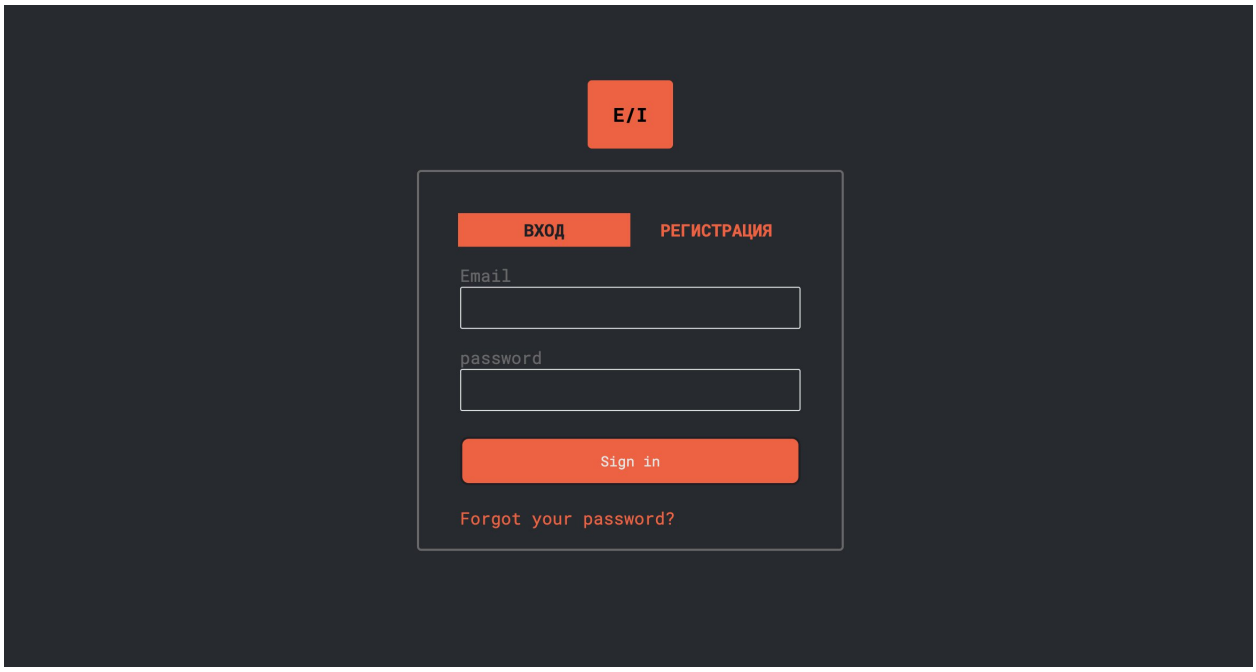


Рисунок 13. Аутентификационная часть

Таким образом, авторизация играет ключевую роль в обеспечении безопасности информационных систем. Она контролирует доступ пользователей к ресурсам системы, защищая конфиденциальные данные и предотвращая несанкционированный доступ. Это позволяет гарантировать надежную защиту системы и предотвращать возможные нарушения безопасности.

2.3.2. Панель обслуживания

Панель обслуживания представляет собой часть интерфейса для управления и мониторинга различных аспектов сервиса. Говоря о ключевых компонентах серверного обслуживания, которые охватывают основную часть системы, стоит рассмотреть такие элементы, как «Search», «Pipeline» и «AdminView».

Компонент Search включает в себя три основных метода получения универсального идентификатора, ранее рассматриваемого. Первым подходом является получение «fuid_raw» путем указания идентификаторов события. Вторым методом является получение этого параметра, с помощью текстового файла, который предварительно должен содержать описания каждого события. И как результат, после выполнения запроса, пользователь получить на почту или в личный кабинет как обработанный файл, так и исходный. Третий подход заключается в извлечении выборки событий из набора данных, которая будет предварительно определена пользователем на основе, например, временного периода текущих данных, количества событий в процентном соотношений.

Для выполнения всех упомянутых запросов предварительно необходимо знать данные принадлежащие конкретному событию. Поэтому было принято решение создать таблицу с подробным описанием событий, которая будет содержать все рассмотренные ранее параметры: run_number, event_number, type, version.

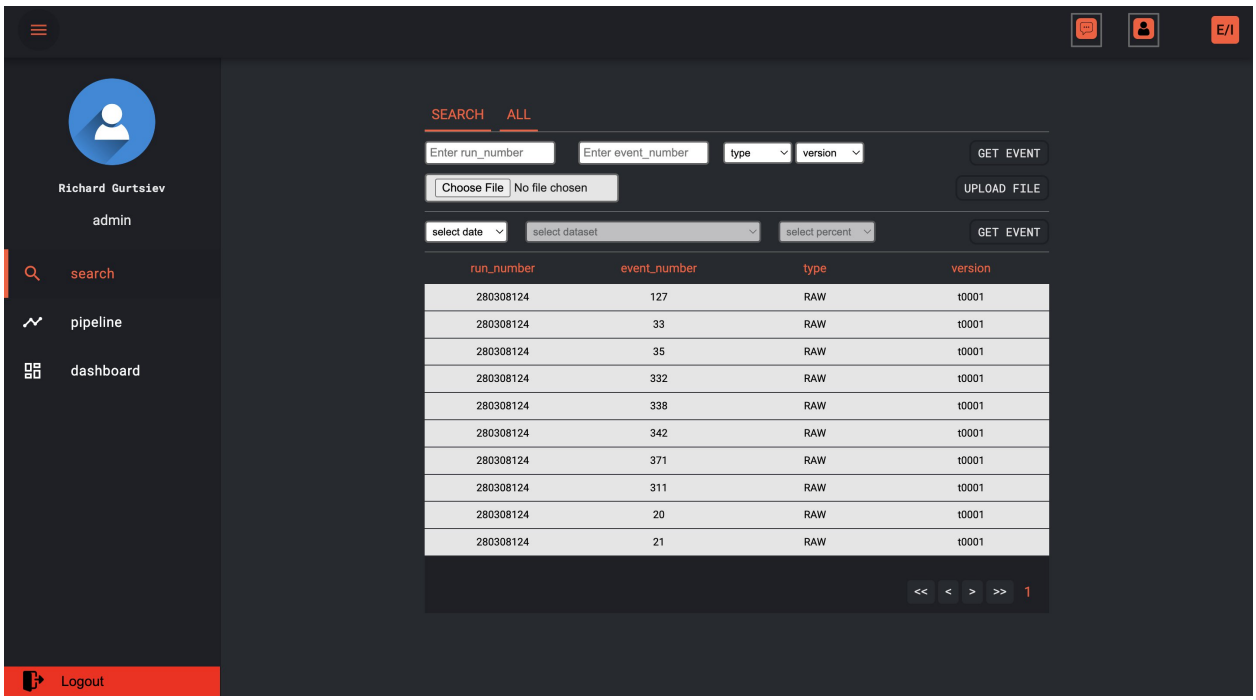


Рисунок 14. Поисковая система для получения универсального идентификатора

Поскольку существует вероятность того, что пользователю по различным причинам потребуется обратиться к обработанным файлам, которые были отправлены им самим, был создан конвейер (Pipeline). Этот компонент содержит информацию о ранее запрошенных файлах и предоставляет возможность ориентироваться в процессе обработки файлов, проверять состояние запроса, а также обращаться и получать как исходные, так и обработанные файлы.

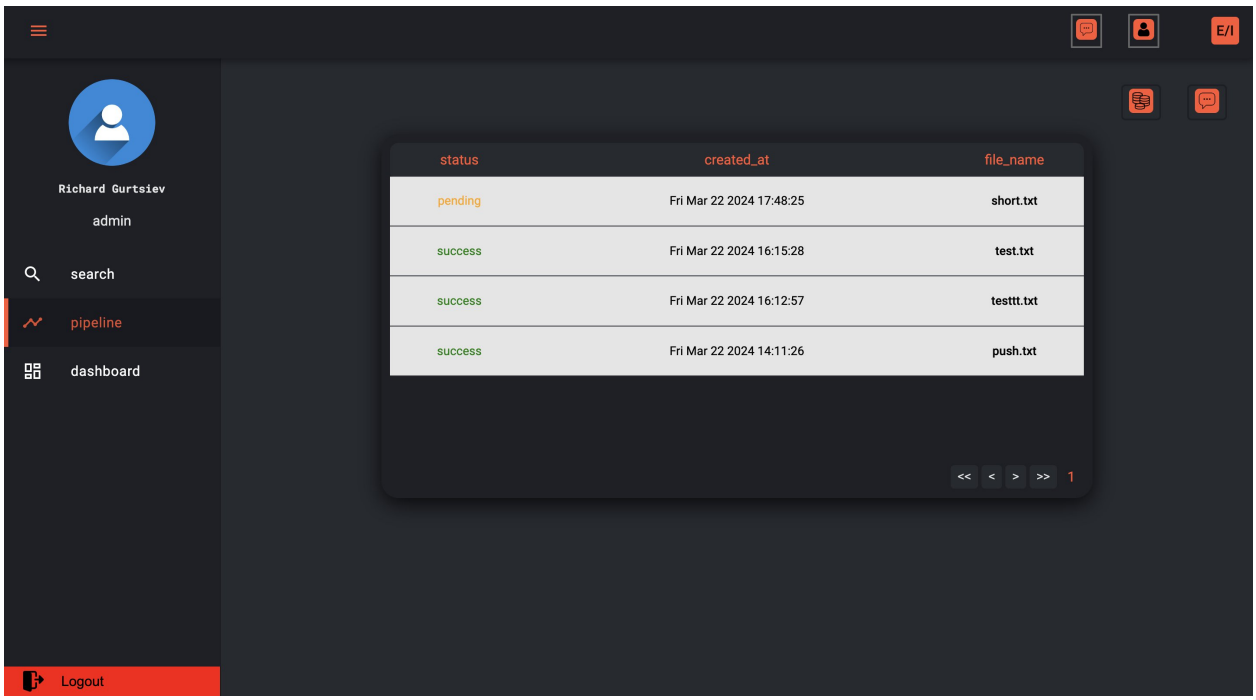


Рисунок 15. Конвейерная система для обработанных файлов

Для получения как исходного, так и обработанного файла предусмотрены два метода, доступных через диалоговое окно. В первом случае пользователь может указать название файла для получения. Однако возможно, что пользователь не сможет вспомнит название отправленного файла. Для таких случаев предусмотрен метод, при котором пользователю потребуется указать примерный период отправки запроса. В результате будут получены все обработанные файлы за указанный временной интервал.

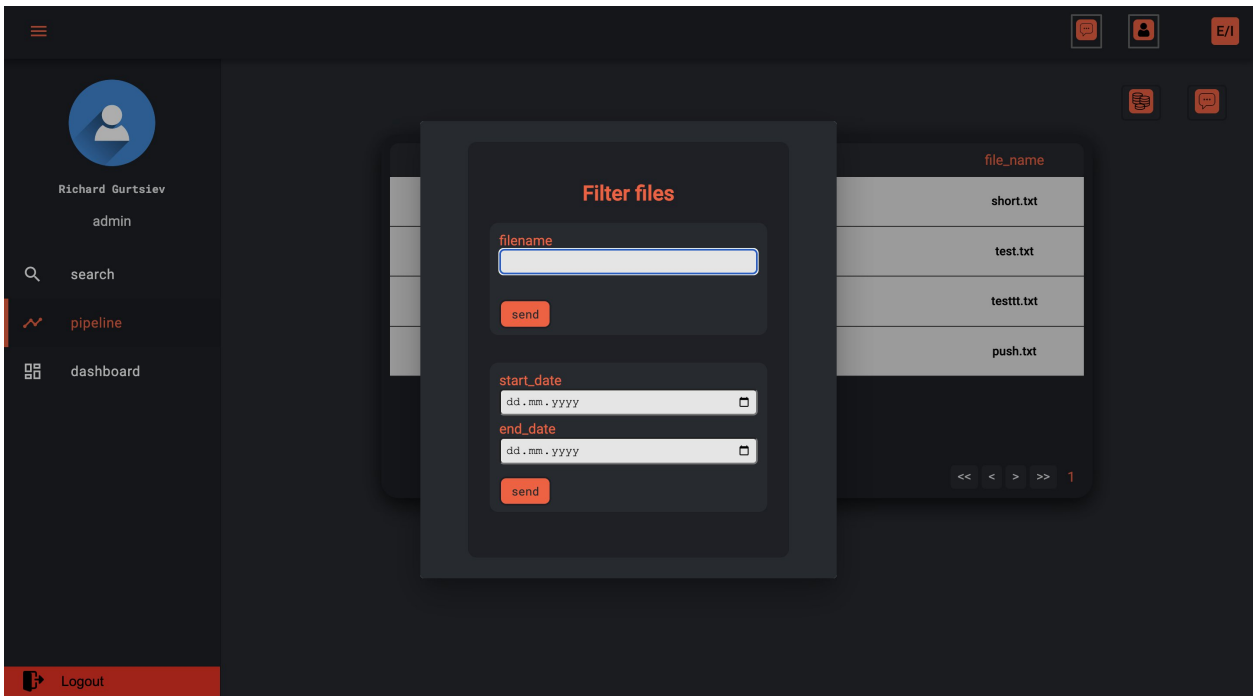


Рисунок 16. Поисковая система для отбора событий

Последним этапом обслуживания является учетная запись администратора (AdminView). С ее помощью администратор может управлять правами доступа участников сервиса, предоставляя или ограничивая доступ к компонентам интерфейса.

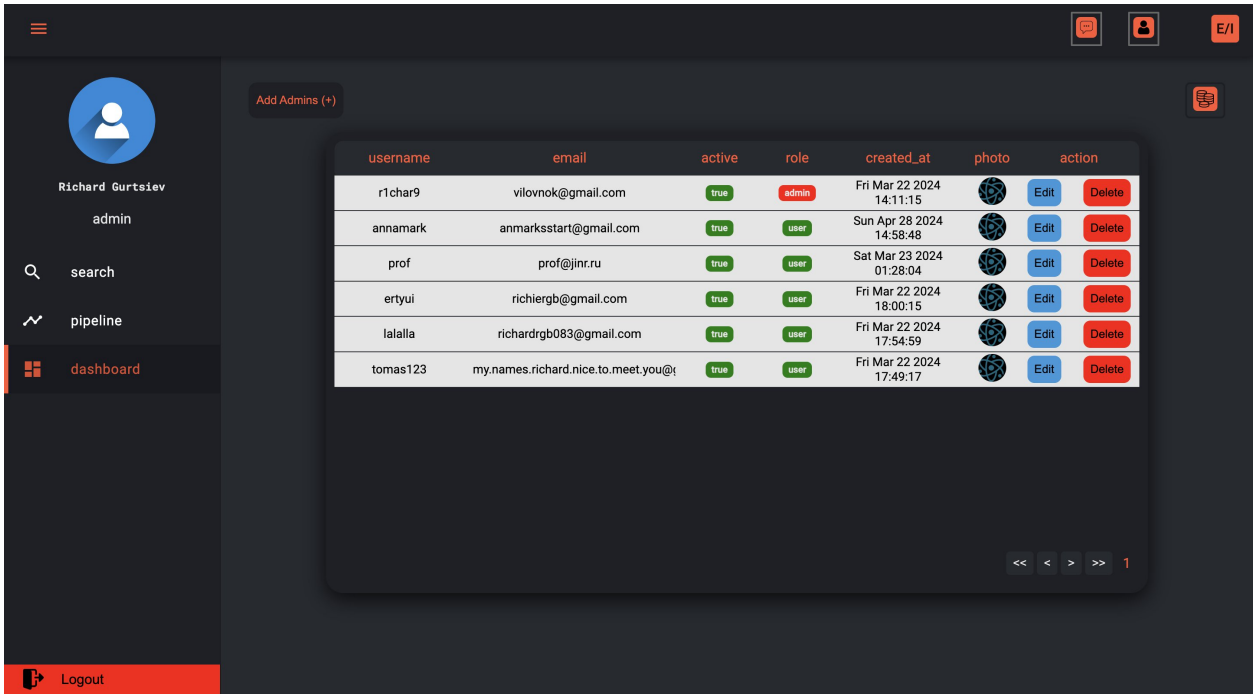


Рисунок 17. Система администрирования пользователей

Поскольку ожидается большое количество пользователей в сервисе, для оперативного взаимодействия с ними была создана поисковая система. Она представляет собой диалоговое окно, где можно использовать методы поиска по электронной почте или имени пользователя.

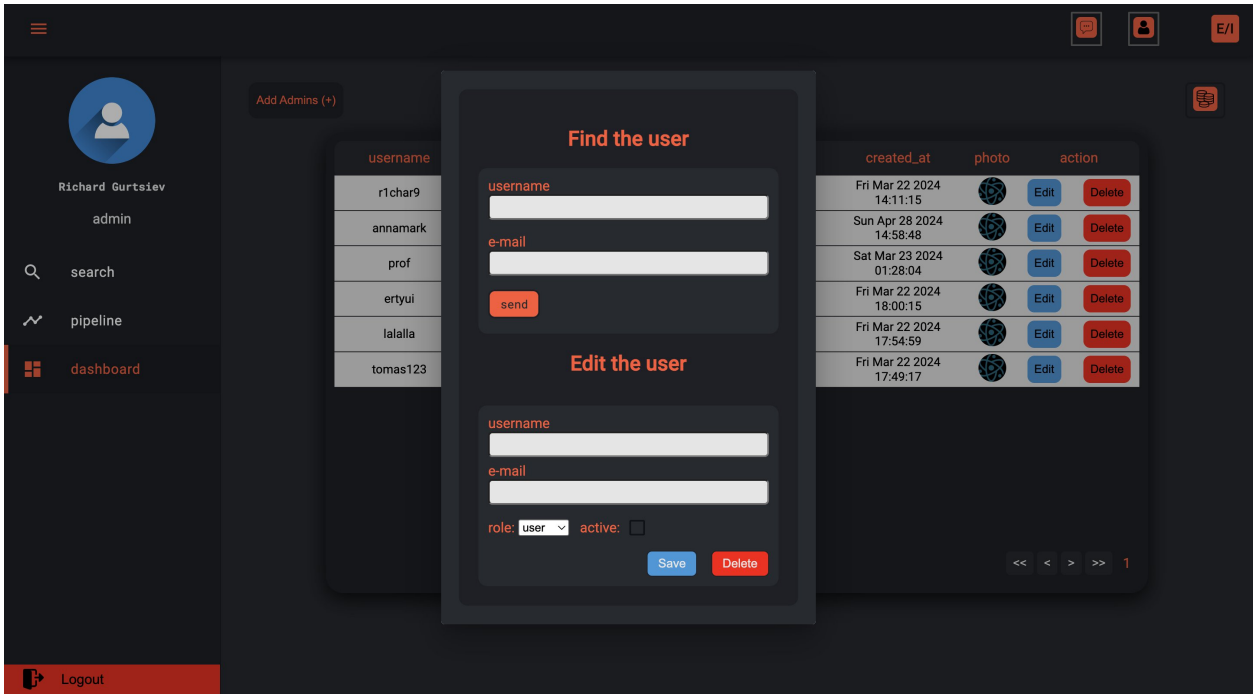


Рисунок 18. Поисковая система для отбора пользователей

В настоящее время «панель управления» позволяет пользователю взаимодействовать с тремя компонентами сервиса, однако в будущем будут добавлены другие компоненты, выполняющие различные методы, что сделает систему более функциональной.

2.4. Архитектура и разработка серверной части

Бэкенд — это та часть веб-приложения или программного обеспечения, которая отвечает за обработку данных, взаимодействие с базой данных, бизнес-логикой, аутентификацией и авторизацией пользователей, а также другие аспекты функциональности, которые не видны конечному пользователю напрямую. В отличие от фронтенда, отвечающего за пользовательский

интерфейс и визуальное взаимодействие, бекенд представляет собой скрытую часть приложения, работающую на серверной стороне. Он обычно написан на языках программирования, таких как Python, Java, PHP, Ruby, Node.js, и может использовать различные фреймворки и библиотеки для упрощения разработки, обеспечения безопасности и масштабируемости [19].

При разработке функциональных элементов сервиса, подразумевалось создание высокопроизводительного RESTful API, обеспечивающие взаимодействие с пользователем. Для лучшего понимания всех процессов функциональная часть сервиса будет разделена на две секции: REST API и Database. Эти компоненты будут далее подробно описаны [7].

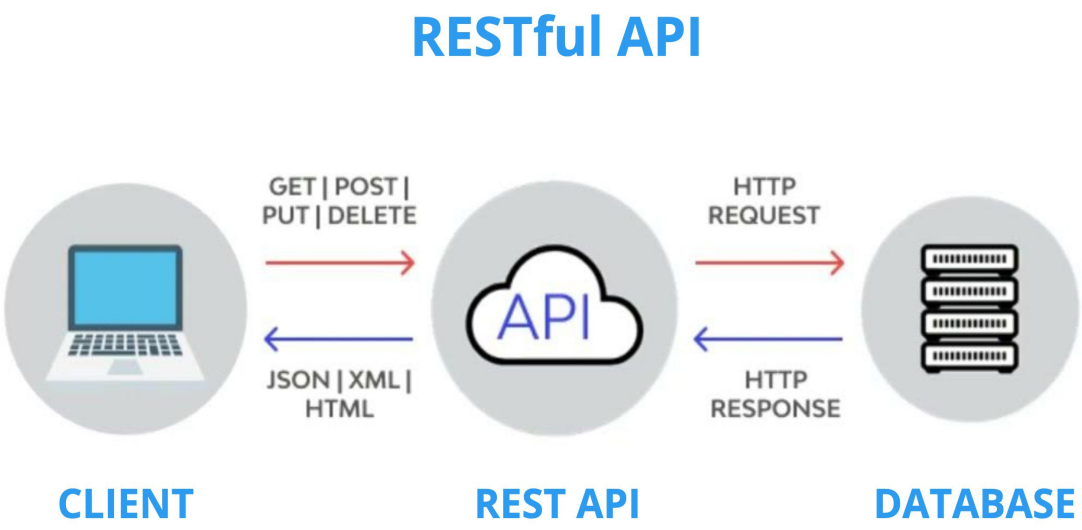


Рисунок 19. Структура RESTful API

2.4.1. Реализация и использование REST API

REST API (Application Programming Interface) — это интерфейс прикладного программирования, который следует принципам архитектуры REST (Representational State Transfer). Он позволяет взаимодействовать с веб-сервисами или приложениями через стандартные методы HTTP, такие как GET, POST, PUT и DELETE, используя уникальные URI (Uniform Resource Identifiers) для доступа к ресурсам. Основные принципы REST API включают в

себя клиент-серверную архитектуру, отсутствие состояния, кеширование, единообразный интерфейс и слои, подобно тем, что описаны для RESTful API. Он позволяет разным системам и приложениям обмениваться данными и взаимодействовать между собой, обеспечивая стандартизированный и удобный способ интеграции [7].

В рамках функциональной части проекта было решено использовать веб-фреймворк FastAPI, который за счет асинхронности и быстрой обработки запросов, обеспечивает высокую производительность и масштабируемость сервиса. FastAPI активно использует декораторы, аннотации типов и интроспекцию кода, что позволяет уменьшить количество шаблонов кода в веб-приложении. Также платформа способна автоматически генерировать и отображает документацию согласно спецификации OpenAPI. В основе FastAPI лежат две библиотеки — Starlette (ASGI-фреймворк) и *Pydantic* (для описания схем данных); FastAPI склеивает их и реализует некоторые дополнительные возможности — регистрацию представлений через внедрение зависимостей, работу с аутентификацией и авторизацией [12].

В рамках бэкенда была внедрена архитектура, которая распределила логику системы на модули, что повысило ее эффективность.

Такой подхода был обусловлен следующими причинами:

- Повышение читаемость и управляемости кода: каждый модуль выполняет отдельную задачу или группу связанных задач;
- Упрощение масштабирования: модульная структура облегчает добавление новых функций и компонентов в проекте;
- Облегчение тестирования: тестирование происходить независимо друг от друга, что упрощает процесс выявления и исправления ошибок.
- Изоляция изменений: изменение в одной секции минимально влияет на другие, что снижает риск непредвиденных ошибок и упрощает управления версиями.

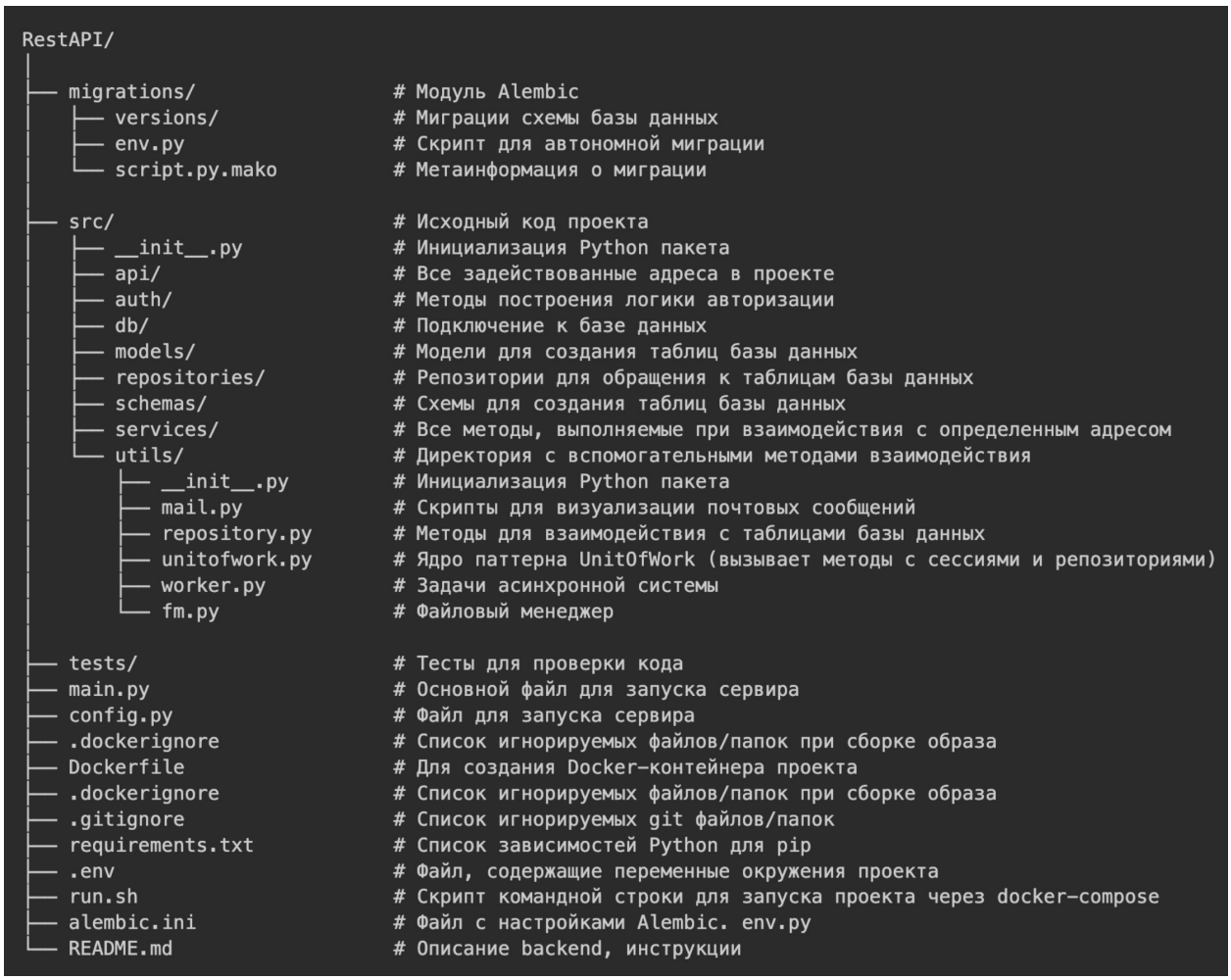


Рисунок 20. Структура распределенной логики backend системы

2.4.1.1. Файловый менеджер

Одним из важных элементов backend части является файловый менеджер. Обработчик имеет большое применение в фронтенд части, особенно для компонентов «search» и «pipeline», которые выполняют процессы поиска событий по файлам и его хранение. Файловый менеджер представляет собой класс, содержащий методы для обработки файла.

```
class FileManager:

    """
        :param path_dir_dataset: путь к директории всех хранимых и передаваемых данных
        :param path_dir_attachments: путь к передаваемым zip файлам
        :param path_dir_trash: путь к временно хранимых файлам
        :param path_zf_raw: путь к исходным zip файлу
        :param path_zf_prep: путь к преобразованным zip файлу
        :param date_now: момент записи данных для unique prefix
        :param unique_prefix: уникальные префикс для названия файла
        :param filename: название обрабатываемого файла
    """

    def __init__(self) -> None:
        self.path_dir_dataset=PATH_DATASET
        self.path_dir_attachments=PATH_ATTACHMENTS
        self.path_dir_trash=PATH_TRASH
        self.path_zf_raw=os.path.join(PATH_DATASET,NAME_RAW_ZIP)
        self.path_zf_prep=os.path.join(PATH_DATASET,NAME_PREP_ZIP)
        self.date_now=None
        self.unique_prefix=None
        self.filename=None
        ...
```

Рисунок 21. Файловый менеджер

Основным методом обработки файлов является поиск паттернов. Когда файл попадает в функциональную зону, он проходит несколько этапов проверки. После этого файл считывается, и в нем ищутся предварительные паттерны, которые затем делятся на четыре категории (run_numbers, event_numbers, types, versions). На их основе создаются персональные векторы для описания универсального идентификатора.

```
def pattern_search(input_text) -> tuple:
        pattern1 = r"(\d+),(\d+),([A-Z]+),([a-zA-Z0-9]+)"
        pattern2 = r"(\d+) (\d+) ([A-Z]+) ([a-zA-Z0-9]+)"
        matches = re.findall(f"({pattern1}|{pattern2})",input_text)
        run_numbers = []
        event_numbers = []
        types = []
        versions = []
        for match in matches:
            run_number = match[1] or match[5]
            event_number = match[2] or match[6]
            type = match[3] or match[7]
            version = match[4] or match[8]
            run_numbers.append(run_number)
            event_numbers.append(event_number)
            types.append(data_format)
            versions.append(data_version)
        return run_numbers, event_numbers, types, versions
```

Рисунок 22. Метод для поиска паттернов в файле

2.4.1.2. Система асинхронной выдачи результатов

При разработке микросервиса с использованием вышеупомянутого фреймворка были созданы API точки доступа (endpoints), которые упрощают взаимодействие с определенными ресурсами или выполнение заданных действий в веб-сервисе. Иными словами, endpoint представляет собой процедуру, запускаемое в ответ на конкретные запросы, поступающие из внешнего интерфейса (визуальной части сервиса) [7].

Во время проектирования конечных точек возникла проблема с основным потоком запросов, которая существенно повлияла на производительность сервиса. Причиной такого поведения являлось продолжительность выполнения каждого запроса. В большинстве своем запросы довольно быстро отрабатываются, однако некоторые из них в особенности методы, занимающиеся обработкой файлов, занимают рабочий узел продолжительное время. Поэтому для балансировки запросов было решено создать систему асинхронной выдачи результатов.

Рисунок 23. Все задействованные запросы сервиса

Асинхронная выдача результатов с использованием таких технологий, как Celery и Redis, позволила выполнять длительные задачи в фоновом режиме, не блокируя основной поток запросов [11].

Celery — это библиотека для управления асинхронными задачами и распределенными вычислениями, позволяющая выполнять задачи в фоновом

режиме и получать результаты по мере их завершения [10].

В связке с Celery используется Redis, высокопроизводительное хранилище ключ-значение, которое служит брокером сообщений. Redis быстро обрабатывает большие объемы данных и хорошо масштабируется [9].

Таким образом, что Redis будет последовательно сохранять запросы пользователей, а Celery будет извлекать эти запросы и обрабатывать их через рабочие узлы в фоновом режиме. Это поможет сбалансировать нагрузку на основной поток запросов и повысить производительность сервиса.

2.4.2. Управление базой данных сервиса

База данных хранит всю необходимую информацию, и позволяет пользователю получить к ней доступ . Поэтому обеспечение эффективности конкретных методов работы с базой данных является важным аспектом сервиса. Для обеспечения связи и удобной интеграции с таблицами PostgreSQL были применены технологии, такие как SQLAlchemy и Alembic.

SQLAlchemy — это библиотека для работы с базами данных, предоставляющая мощный API, предназначенный для реляционных баз данных, таких как PostgreSQL, MySQL, SQLite и другие. Она упрощает создание и управление объектами баз данных, позволяя пользователям определять структуры таблиц с помощью конструкций, называемых «моделями» [13].

Alembic является утилитой для мониторинга миграции баз данных, легко взаимодействующая с SQLAlchemy. Она предоставляет надежные функциональные возможности для автоматизации изменений в схеме базы данных, включая такие задачи, как введение новых таблиц, изменение существующих столбцов, создание индексов и многое другое [14].

```
from typing import AsyncGenerator
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy import MetaData

from config import DB_HOST, DB_NAME, DB_PASS, DB_PORT, DB_USER

DATABASE_URL = f"postgresql+asyncpg://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
Base = declarative_base()
metadata = MetaData()
engine = create_async_engine(DATABASE_URL)
async_session_maker = sessionmaker(engine, class_=AsyncSession, expire_on_commit=False)

async def get_async_session() -> AsyncGenerator[AsyncSession, None]:
    async with async_session_maker() as session:
        yield session
```

Рисунок 24. Скрипт для подключения к базе данных

Таким образом, SQLAlchemy будет взаимодействовать с ранее определенными таблицами в PostgreSQL, а Alembic, при необходимости, будет обновлять всю базу данных в случае развития сервиса, что сделает систему более автоматизированной и надежной.

2.4.3. Оркестрационная сторона сервиса: DevOps

DevOps — это методология, направленная на ускорение программного обеспечения, повышение его качества и надежности. Система автоматизирует процессы разработки, тестирования, и развертывания сервисов [6].

Основные аспекты DevOps включают:

1. Непрерывную интеграцию и непрерывное развертывание (CI/CD): Применение методологии непрерывной интеграции и непрерывной доставки для автоматического тестирования, сборки и развертывания приложений с целью быстрой и надежной поставки изменений в продакшн.
2. Мониторинг и логирование: Организация системы мониторинга и сбора логов для наблюдения за работой приложений и инфраструктуры, а также быстрого выявления и решения проблем.

3. Безопасность: Интеграция практик безопасности во все этапы разработки и эксплуатации приложений для обеспечения защиты данных и систем от угроз.
4. Автоматизация процессов: Использование инструментов для автоматизации различных этапов разработки, тестирования, развертывания и управления инфраструктурой.
5. Непрерывное улучшение: Постоянное совершенствование процессов, инструментов и практик разработки и эксплуатации на основе обратной связи и анализа производительности.

Одним из значимых технологий в рамках оркестрации сервиса является Docker. Более подробно данная платформа будет рассмотрена далее.

2.4.3.1. Контейнеризация и развертывание

При разработке проекта необходимо было предварительно определить способ интеграции в выбранную виртуальную машину. Одним из них является контейнеризация, которая «упаковывает» части проекта в «контейнеры». Каждый контейнер обеспечивает изоляцию, что позволяет им безотказно выполнять персональные процессы в любом окружение, будь то производственная среда, локальный компьютер или сервер. Сами контейнеры легковесны, так как они используют ядро хостовой операционной системы. Самым распостраненным инструментом контейнеризации является Docker [15]. Главными элементами этой платформы и их значения:

1. Docker Host — операционная система, на которую устанавливают Docker и на которой он работает.
2. Docker Daemon — служба, которая управляет Docker-объектами: сетями, хранилищами, образами и контейнерами.
3. Docker Client — консольный клиент, при помощи которого пользователи взаимодействуют с Docker daemon и отправляют ему команды, создают контейнеры и управляют ими.

4. Docker Image — неизменяемый образ, из которого разворачивается контейнер.
5. Docker Container — развёрнутое и запущенное приложение.
6. Docker Registry — репозиторий, в котором хранятся образы.
7. Dockerfile — файл-инструкция для сборки образа.
8. Docker Compose — инструмент для управления несколькими контейнерами.

Когда компоненты, отвечающие за функциональность и визуальную составляющую сервиса, переходят из разработки в тестирование, важно сохранить поддерживающие инструменты каждого элемента проекта при его внедрении на виртуальную машину. Вспоминая архитектуру сервиса важно учитывать, что компоненты этой структуры будут иметь локальные окружения, и для большинства из них образы будут браться с Docker Registry.

Однако элементы являются уникальными, поэтому для них необходимо создавать персональные Docker-образы. Это осуществляется с помощью Dockerfile, который содержит инструкцию для сборки образа [15].

Основные команды в Dockerfile для визуальной и функциональной части сервиса:

1. FROM: Определяет базовый образ.
2. RUN: Выполняет команду внутри контейнера во время сборки образа.
3. COPY / ADD: Копирует файлы и директории из локальной файловой системы внутри контейнера.
4. WORKDIR: Устанавливает рабочую директорию.
5. EXPOSE: Экспортирует порты.
6. ENV: Устанавливает переменные среды внутри контейнера.
7. ARG: Определяет аргументы сборки.
8. CMD / ENTRYPOINT: Определяет команду, которая будет выполнена при запуске.

```
Dockerfile RestAPI

FROM python:3.10
ENV PYTHONUNBUFFERED=1
WORKDIR /restapi
COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt
COPY . /restapi
RUN chmod a+x *.sh
EXPOSE 8000
```

```
Dockerfile Angular

FROM node:18.17.1-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
RUN npx ngcc --properties es2023 browser module main --firts-only --create-ivy-entry-points
COPY . .
RUN npm run build
FROM nginx:stable
COPY ./nginx/nginx.conf /etc/nginx/nginx.conf
COPY --from=build /app/dist/event-index/ /usr/share/nginx/html
EXPOSE 80
```

Рисунок 25. Dockerfiles для создания frontend и backend образов

2.4.3.2. Многоконтейнерная система

После подготовки всех образов требуется организовать их группировку, и в этом может помочь Docker Compose. Этот инструмент для определения и запуска многоконтейнерных Docker приложений. Система буквально одной командой, собирает все образы, необходимые для работы сервиса. Чтобы воспользоваться этой системой нужно создать специальный yaml файл, который будет содержит в некотором смысле «описание сервиса», включая информацию о контейнерах, сетях, томах и других параметрах, необходимых для его запуска и работы. То есть данный файл определяет конфигурацию сервиса, включая настройки каждого контейнера, их зависимости и параметры запуска [15].

```yaml
version: "3.8"
services:
  web:
    container_name: web
    restart: always
    build:
      context: ./event-index
    ports:
      - 80:80
    expose:
      - 80

  db:
    container_name: db
    restart: unless-stopped
    image: postgres:14-alpine
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASS}
      - POSTGRES_DB=${DB_NAME}
    expose:
      - 5432

  redis:
    image: redis:6.2-alpine
    container_name: redis

  api:
    container_name: api
    restart: always
    build:
      context: ./restapi
    ports:
      - "8000:8000"
    command: ["/restapi/app.sh","app"]
    depends_on:
      - db
      - redis
    environment:
      - DATABASE_URL=postgresql+asyncpg://${DB_USER}:${DB_PASS}@${DB_HOST}:${DB_PORT}/${DB_NAME}
      - CELERY_BROKER_URL=redis://redis:6379/0

  celery:
    container_name: celery
    restart: always
    build:
      context: ./restapi
    command: ["/restapi/celery.sh","celery"]
    depends_on:
      - redis
    environment:
      - CELERY_BROKER_URL=redis://redis:6379/0

  flower:
    container_name: flower
    restart: always
    build:
      context: ./restapi
    command: ["/restapi/celery.sh","flower"]
    depends_on:
      - redis
      - api
    ports:
      - 9999:5555
    environment:
      - CELERY_BROKER_URL=redis://redis:6379/0
```

Рисунок 26. Yaml файл для многоконтейнерной системы

Таким образом, использование Docker в контексте полноценного веб-сервиса при помощи Docker Compose обеспечивает гибкость и удобство. Docker Compose позволяет определять и управлять множеством контейнеров как единым приложением, что упрощает локальную разработку, тестирование и развертывание приложения [15].

Благодаря возможности определения всех компонентов сервиса и их взаимодействий в одном конфигурационном файле, Docker Compose значительно снижает затраты времени на настройку и управление окружением разработки.

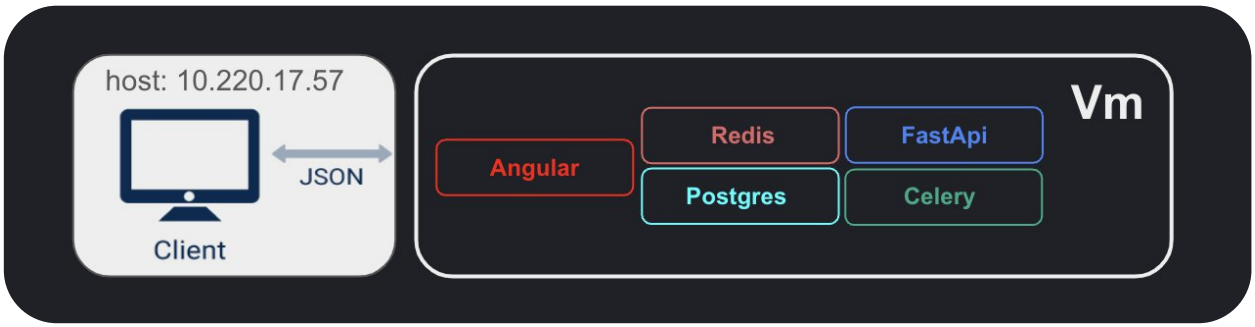


Рисунок 27. Многоконтейнерная система

ВЫВОДЫ ПО ВТОРОЙ ГЛАВЕ

В этой главе была подробно рассмотрена архитектура веб-сервиса эксперимента SPD EventIndex, состоящая из трех основных компонентов: frontend, backend и devOps. Каждый компонент играет важную роль в общей функциональности пользовательского интерфейса и работы системы в целом.

Ключевыми технологиями компонент данного проекта являются Angular, FastAPI и Docker. Angular используется для разработки пользовательского интерфейса, обеспечивая прямое взаимодействие с пользователем. Запросы, поступающие от пользователей, обрабатываются на сервере с помощью FastAPI, обеспечивая эффективную обработку данных. Docker применяется для обеспечения надежного развертывания, интеграции и поддержки всей системы.

Основными структурами данных, лежащие в основе проекта, сосредоточено на сводных таблицах, размещенных в PostgreSQL. Таблицы наборов данных служат для организации и категоризации наборов данных, облегчая эффективный поиск данных и манипулирование ими. Этими таблицами, где хранятся главные параметры данных, являются verify, users, pipelines, events, datasets. Из них ключевыми «каталогами данных» являются events и datasets, поскольку именно они будут содержат основную информацию о событиях.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано программное обеспечение, позволяющее пользователю получить информацию о данных, передаваемых с детектора SPD большого сверхпроводящего коллайдера NICA, способного регистрировать и анализировать различные события, возникающие при столкновениях ядер и элементарных частиц с высокой энергией.

Предварительно была реализована архитектура, которая учитывала аспекты высокопроизводительности:

1. Масштабируемость — Распределения нагрузки и увеличения мощности серверов.
2. Оптимизация базы данных — Использование индексации для ускорения поиска данных, кэширование запросов для уменьшения нагрузки.
3. Асинхронная обработка — Использование очередей задач и асинхронного программирования для выполнения долгих задач в фоновом режиме.
4. Оптимизация кода — Профилирование для выявления узких мест и оптимизация критически важных участков кода.
5. Мониторинг и логирование — Мониторинг производительности и использование логов для оптимизации.

На основе данной структуры сервис был разработан и интегрирован на виртуальную машину с операционной системой «AlmaLinux-9».

Главными компонентами системы составляют фреймворки Angular и FastAPI. Вместе эти фреймворки создают мощную и гибкую инфраструктуру, которая обеспечивает высокую производительность, масштабируемость и удобство в использовании, что делает конечный сервис эффективным и надежным для пользователей.

Базой данных сервиса является PostgreSQL, и эта СУБД играет важную роль, храня всю необходимую информацию:

- идентификационные данные пользователей;

- описание событий;
- результаты обработки файлов.

Также существенный вклад в инфраструктуру внес Docker, который локально изолирует компоненты сервиса, обеспечивая высокую производительность и легкость интеграции в различных виртуальных окружениях.

В качестве среды разработки самого сервиса была выбрана среда Visual Studio Code, который обладает необходимыми инструментами для реализации программного обеспечения.

В рамках веб-приложения были созданы следующие экранные формы:

- Login — форма входа в систему;
- Registration — форма регистрации в систему;
- Verify — форма проверки верификации пользователя;
- AdminView — форма учетной записи администратора;
- Pipeline — форма учета запрошенных файлах;
- Search — форма учета событий.

Разработанный проект соответствует техническим требованиям. Очевидно, что текущий результат является одним из этапов разработки для успешной эксплуатации сервиса на виртуальных машинах. Для обеспечения необходимого уровня производительности, функциональности и удобства использования требуются дополнительные улучшения и настройки, которые могут быть достигнуты путем применения соответствующих технологий и алгоритмов в системе «SPD EventIndex».

Таким образом, поставленные цели выпускной квалификационной работы достигнуты. В дальнейшем будут продолжены доработки и усовершенствования интерфейсных компонентов, чтобы оптимизировать взаимодействие с пользователем и максимально эффективно использовать сервис «SPD EventIndex» в научных исследований и анализа данных.

СПИСОК ЛИТЕРАТУРЫ

1. The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of data [Электронный ресурс]: монография: Д. Барберис, А. Фаварето, Д. Малон, М. Новак — Режим доступа: https://pos.sissa.it/210/023/pdf
2. Мегапроект NICA [Электронный ресурс] — Режим доступа: https://nica.jinr.ru/ru/
3. PanDa: Production and Distributed Analysis System [Электронный ресурс]: монография / А. Алексеев, Т. Маено, Т. Корчуганова и др. — Режим доступа: https://link.springer.com/article/10.1007/s41781-024-00114-3
4. Сервис-ориентированная архитектура (SOA) [Электронный ресурс] — Режим доступа: https://habr.com/ru/companies/vk/articles/342526/
5. Conceptual Design of the Spin Physics Detector [Электронный ресурс]: монография / Л. Афонасиев, Р. Ахунзянов — Режим доступа: https://www.researchgate.net/publication/348959622_Conceptual_design_of_the_Spin_Physics_Detector
6. Зачем нужен DevOps и кто такие DevOps-специалисты [Электронный ресурс] — Режим доступ: https://habr.com / ru / companies / epam_systems / articles / 465601 /
7. Введение в REST API — RESTful веб-сервисы [Электронный ресурс] — Режим доступа: https://habr.com/ru/articles/483202/
8. Введение в концепции Angular [Электронный ресурс] — Режим доступа: https://angular.io/guide/architecture
9. Redis документация [Электронный ресурс] — Режим доступа: https://redis.io/
10. Celery – Распределенная очередь задач [Электронный ресурс] — Режим доступа: https://docs.celeryq.dev/
11. Официальный сайт PostgreSQL [Электронный ресурс] — Режим доступа: https://www.postgresql.org/

12. FastAPI’s документация [Электронный ресурс] — Режим доступа: https://fastapi.tiangolo.com/
13. SQLAlchemy документация [Электронный ресурс] — Режим доступа: https://www.sqlalchemy.org/
14. Alembic документация [Электронный ресурс] — Режим доступа: https://alembic.sqlalchemy.org/
15. Docker документация [Электронный ресурс] — Режим доступа: https://www.docker.com/
16. SPD Event Index [Электронный ресурс]: статья / Ф. Прокошин, И. Тваури, Р. Гурциев, А.-Б. Газзаев, З. Будтуева — Режим доступа: https://link.springer.com/article/10.1134/S1063779624030699?utm_source=rct_congratemailt&utm_medium=email&utm_campaign=nonoa_20240606&utm_content=10.1134%2FS1063779624030699
17. Rucio: Scientific Data Manager [Электронный ресурс]: монография / М. Бариситс, Т. Бирманн, Ф. Бергхаус, Б. Бокельман — Режим доступа: https://link.springer.com/article/10.1007/s41781-019-0026-3
18. Фронтенд-разработка: ключевые технологии и понятия [Электронный ресурс] — Режим доступа: https://habr.com/ru/companies/otus/articles/674748/
19. Бэкенд-разработчик: кто это и чем он занимается [Электронный ресурс] — Режим доступа: https://ru.hexlet.io/blog/posts/backend-developer
20. Обзор способов и протоколов аутентификации в веб-приложениях [Электронный ресурс] — Режим доступа: https://habr.com/ru/companies/dataart/articles/262817/

ПРИЛОЖЕНИЕ

Скрипт для формы учета событий

```
<div class="container"
  [ngStyle]="!this.loadingHandler.isLoading ? {} : {'display': 'flex', 'justify-content': 'center', 'align-items': 'center', 'height':'100%','width': 'auto'}">
  <div *ngIf="!this.loadingHandler.isLoading">
    <div class="navbar_container">
      <div>
        <ul>
          <li>SEARCH</li>
          <li>ALL</li>
        </ul>
        <div class="items">
          <div>
            <div class="part_1_block">

              <input type="text" name="run_number" [(ngModel)]="run_number" placeholder="Enter run_number">

      <input type="text" name="event_number" [(ngModel)]="event_number" placeholder="Enter event_number">
            </div>
            <div class="part_2_block">
              <select name="category" [(ngModel)]="type">
                <option value="type">type</option>
                <option value="RAW">RAW</option>
                <option value="AOD">AOD</option>
              </select>
```

```
            <select name="category" [(ngModel)]="version">
              <option value="version">version</option>
              <option value="t0001">t0001</option>
              <option value="t0002">t0002</option>
              <option value="t0003">t0003</option>
              <option value="t0004">t0004</option>
            </select>
          </div>
          <button (click)="GetDataFuid()">GET EVENT</button>
        </div>
        <div>

    <div class="FileSelected">
            <input type="file" (change)="onFileSelected($event)" accept=".txt">
          </div>
          <button type="submit" (click)="UploadFile()">UPLOAD FILE</button>
        </div>
        <div style="border-top: 1px solid #686868;">
          <div class="part_3_block">

      <select name="category" [(ngModel)]="date" (change)="CheckDatasets(date)">
            <option>select date</option>

            <option *ngFor="let date of dataByDate"
[ngValue]="date.date">{{date.date}}</option>

</select>
```

```
            <select name="category" [(ngModel)]="dataset" [disabled]="date=='select
date'">
              <option>select dataset</option>
              <option *ngFor="let dataset of datasets"
value="{{dataset.dsid}}">{{dataset.dsn}}</option>
            </select>

            <select name="category" [(ngModel)]="percent"
[disabled]="dataset=='select dataset'">
              <option>select percent</option>

              <option *ngFor="let item of percents"
[ngValue]="item">{{item}}</option>

            </select>
          </div>
          <button (click)="gonextpage()">GET EVENT</button>
        </div>
      </div>
      <div class="metrics">
        <div>
          <table>
            <thead>
              <tr>
                <td>run_number</td>
                <td>event_number</td>
                <td>type</td>
                <td>version</td>
              </tr>
```

```
            </thead>
            <tbody>

   <tr *ngFor="let event of paginatedData">
                <td>{{event.run_number}}</td>
                <td>{{event.event_number}}</td>
                <td>{{event.type}}</td>
                <td>{{event.version}}</td>
              </tr>
            </tbody>
          </table>

 <br>
          <div class="pagination_section">
            <button (click)="navigateToFirstPage()"><<<</button>
            <button (click)="navigateBackward()"><</button>
            <button (click)="navigateForward()">></button>
            <button (click)="navigateToLastPage()">>></button>
            <span

      style="color: #ff5533; vertical-align:middle; display: flex; align-items:
center;">{{currentPage}}</span>
          </div>
        </div>
       </div>
      </div>
    </div>
  </div>
```

```
  <app-loading *ngIf="this.loadingHandler.isLoading"></app-loading>
</div>
```