Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторному проекту

По дисциплине «Архитектурные решения для обработки больших объёмов информации»

По теме «Микросервисная multi-tenancy архитектура для новостного агрегатора»

Выполнил:

магистрант гр. 056241

Хлопцев А.А.

Проверил:

доцент кафедры информатики

Стержанов М.В.

Минск 2022

Оглавление

[Введение 3](#_Toc96268618)

[Список нововведений 4](#_Toc96268619)

[Переход к микросервисам 5](#_Toc96268620)

[Общая архитектура 5](#_Toc96268621)

[Шина передачи сообщений (Kafka+Faust) 10](#_Toc96268622)

[Источники новостей (Feeders) 13](#_Toc96268623)

[Обработчики новостей (Processors) 16](#_Toc96268624)

[Классификаторы новостей (Predictors) 17](#_Toc96268625)

[Клиенты кластера (Clients) 19](#_Toc96268626)

[Multi-tenancy веб приложение 23](#_Toc96268627)

[Общая архитектура 25](#_Toc96268628)

[Анализ 27](#_Toc96268629)

[1. Solution modularity 27](#_Toc96268630)

[2. Observability 30](#_Toc96268631)

[3. Content vs. feature de-coupling 34](#_Toc96268632)

[4. “Shift Right Ops” readiness 36](#_Toc96268633)

[5. Tenant provisioning easiness 38](#_Toc96268634)

[6. Deployment patterns 41](#_Toc96268635)

[Вывод 42](#_Toc96268636)

[Список источников 42](#_Toc96268637)

# Введение

В качестве основы для лабораторного проекта был использован проект по МиМОБОД с прошлого года. Проект можно найти по ссылке: https://github.com/AndrewKhl/WNews/tree/master

Проект состоял из 3 частей:

1. Расширяемый модуль, в который можно добавить обработчики открытых API интернет-ресурсов. Данный модуль необходим для сбора информации.

2. Модуль сортировки и хранения собранной и обработанной информации.

3. Модуль, реализующий алгоритмы машинного обучения, необходимые для классификации полученной информации и функции обработки полученной информации.

Данный проект необходимо было перевести на мильтисервисную архитектуру с элементами multi-tenancy, добавить новую функциональность, исправить найденные ошибки и внести ряд правок в уже имеющиеся модули.

# Список нововведений

Вторая версия приложения включает следующие нововведения и функции:

* Архитектура приложения была переделана из монолитной в микросервисную
* Веб часть приложения было переделано под multi-tenancy архитектуру
* Добавлена логика работы с пользователями
* Новости теперь поддерживают несколько тегов (раньше выбиралась наиболее вероятная)
* Добавлен новый источник новостей – TheNewYorkTimes (<https://www.nytimes.com>)
* Разделение логики хранения новостей и остальной информации
* Оптимизация производительности и исправление ошибок.

# Переход к микросервисам

## Общая архитектура

Базовый проект был написан на языке программирования Python 3.8 с использованием фраемворка Django 3.1 для веб представления приложения.

Серверная часть представляла собой монолитное приложение, состоящее из следующих частей (рис. 1):

1. TheGuardianParser – модуль, считывающий новые новости с использованием TheGuardian Api
2. TextManager – модуль, отслеживающий все рабочие парсеры новостей
3. TextProcessor – модуль, обрабатывающий тексты и подготавливающий их к классификации
4. SvmAdapter – модуль, выполняющий классификацию новостей с помощью метода опорных векторов по одной из заданных тем
5. SvmManager – модуль, хранящий в себе и управляющий списком Svm адаптеров
6. DatabaseManager – модуль, отвечающий за сохранение данных в базе данных MySQL
7. ArticleModel – модель представления информации о новости

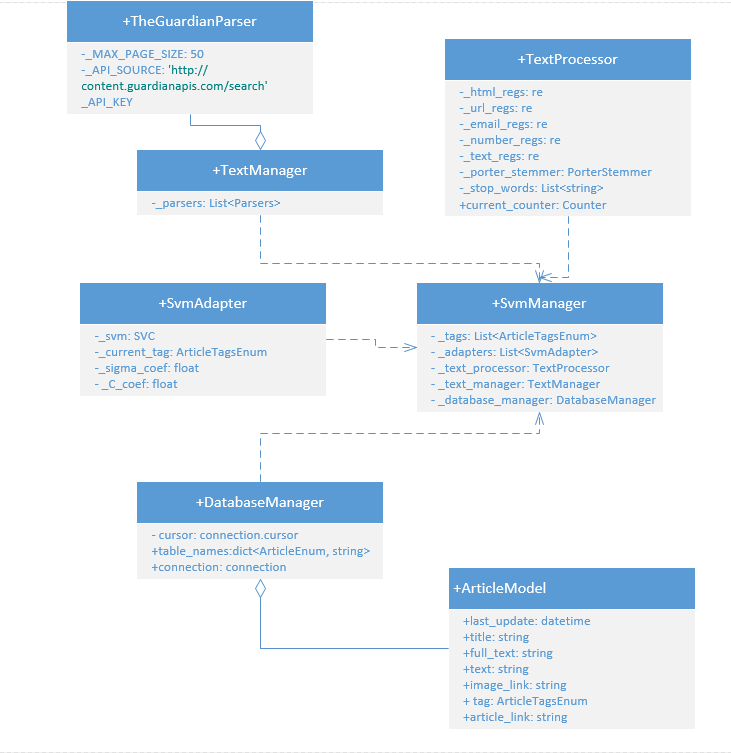


Рис. 1 – Архитектура серверной части первой версии приложения

Для второй версии приложения был совершён переход к микросервисной архитектуре, состоящей из следующих частей:

1. Feeders – группа сервисов, отвечающих за поставку новостей из различных источников
2. Шина сообщений – модуль, отвечающий за передачу сообщений между сервисами
3. Processors – группа сервисов, отвечающих за обработку полученных текстов
4. Predictors – группа сервисов, отвечающих за классификацию новостей
5. Clients – группа сервисов, обрабатывающая классифицированные новости

С новой архитектурой приложения можно познакомиться на рис. 2.

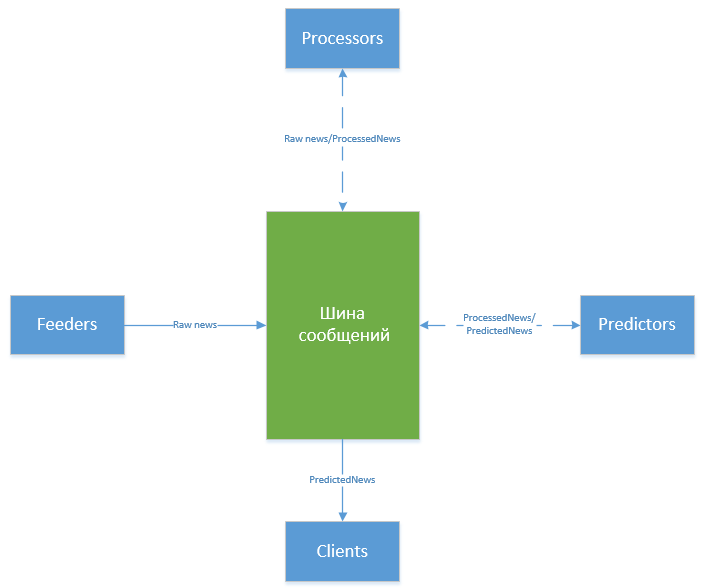


Рис. 2 – Общая архитектура 2 версии приложения

Общий пусть данных таков:

1. Фидеры скачивают новости, преобразуют их в общий формат сообщения (RawNews) и отправляют на шину.
2. Шина пересылает RawNews к процессорам на обработку, после обработки процессоры возвращают обработанные новости (ProcessedNews) обратно на шину.
3. Шина отправляет обработанные новости ProcessedNews к сервисам классификации (Predictors), где ProcessedNews преобразуются в PredictedNews (классифицированные новости), и возвращаются обратно на шину.
4. Шина отправляет PredictedNews клиентам

Такие образом сервисы отдельных групп ничего не знают о остальных группах, являются полностью независимыми, могут быть развёрнуты на разных машинах и масштабированы без остановки всей системы.

На рис.3 можно увидеть более детальную схему реализованных сервисов, модулей и их связи. Подробнее каждая группа сервисов будет рассмотрена далее.

Обозначения по цветам:

* Жёлтый – части Веб приложения
* Зелёный – элементы шины сообщений
* Синий – базовые Python сервисы

Обозначения по фигурам:

* Круг – Веб приложение
* Залитый прямоугольник – сервис
* Цилиндр – база данных
* Прямоугольник с градиентом – семейство сервисов
* Прямоугольник без заливки – Kafka topics



Рис. 3 – Подробная архитектура 2 версии приложения

## Шина передачи сообщений (Kafka+Faust)

В качестве шины сообщений было решено использовать Apache Kafka version 3.1.0.

**Apache Kafka** — распределённый программный [брокер сообщений](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%BE%D0%BA%D0%B5%D1%80_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B9), проект с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), разрабатываемый в рамках фонда [Apache](https://ru.wikipedia.org/wiki/Apache_Software_Foundation" \o "Apache Software Foundation). Написан на языках программирования [Java](https://ru.wikipedia.org/wiki/Java" \o "Java) и [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)" \o "Scala (язык программирования)).

Спроектирован как [распределённая](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), [горизонтально масштабируемая](https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D1%80%D0%B8%D0%B7%D0%BE%D0%BD%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C) система, обеспечивающая наращивание пропускной способности как при росте числа и нагрузки со стороны источников, так и количества систем-подписчиков. Подписчики могут быть объединены в группы. Поддерживается возможность временного хранения данных для последующей [пакетной обработки](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0). Одной из особенностей реализации инструмента является применение техники, сходной с [журналами транзакций](https://ru.wikipedia.org/wiki/%D0%96%D1%83%D1%80%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%B9), используемыми в [системах управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [1].

Для понимания работы с Kafka, необходимо дать определение 4 базовым понятиям:

1. Топик (topics) – это способ группировки поток сообщений по категориям. Производители публикуют сообщения определённой категории в топик, а потребители подписываются на него.
2. Раздел (partitions) – это последовательность сообщений топика, которые упорядочены в порядке их поступления [2].
3. Производитель (producer) – приложение-источник данных. Оно подключается к кластеру Kafka и записывает сообщения с данными в определённые топики и разделы.
4. Потребитель (consumer) – приложение, которое принимает и обрабатывает данные от производителя [3].



Рис. 4 – Строение Kafka кластера

Kafka кластер в приложении WNews содержит 3 основных топика:

1. RawNews Topic – хранит новости, полученные из фидеров
2. ProcessedNews Topic – хранит обработанные новости, полученные из процессоров
3. PredictedNews Topic – хранит классифицированные новости, полученные из предикторов

В данной архитектуре можно заметить ещё 2 дополнительных модуля:

1. Zookeper - это распределенная служба для хранения информации о конфигурации основного сервера приложений. Zookeeper также выполняет функцию управления реакцией сервера на реакцию клиента.
2. Kafka Magic – веб приложение для мониторинга состояния кластера и zookeeper (рис. 5)

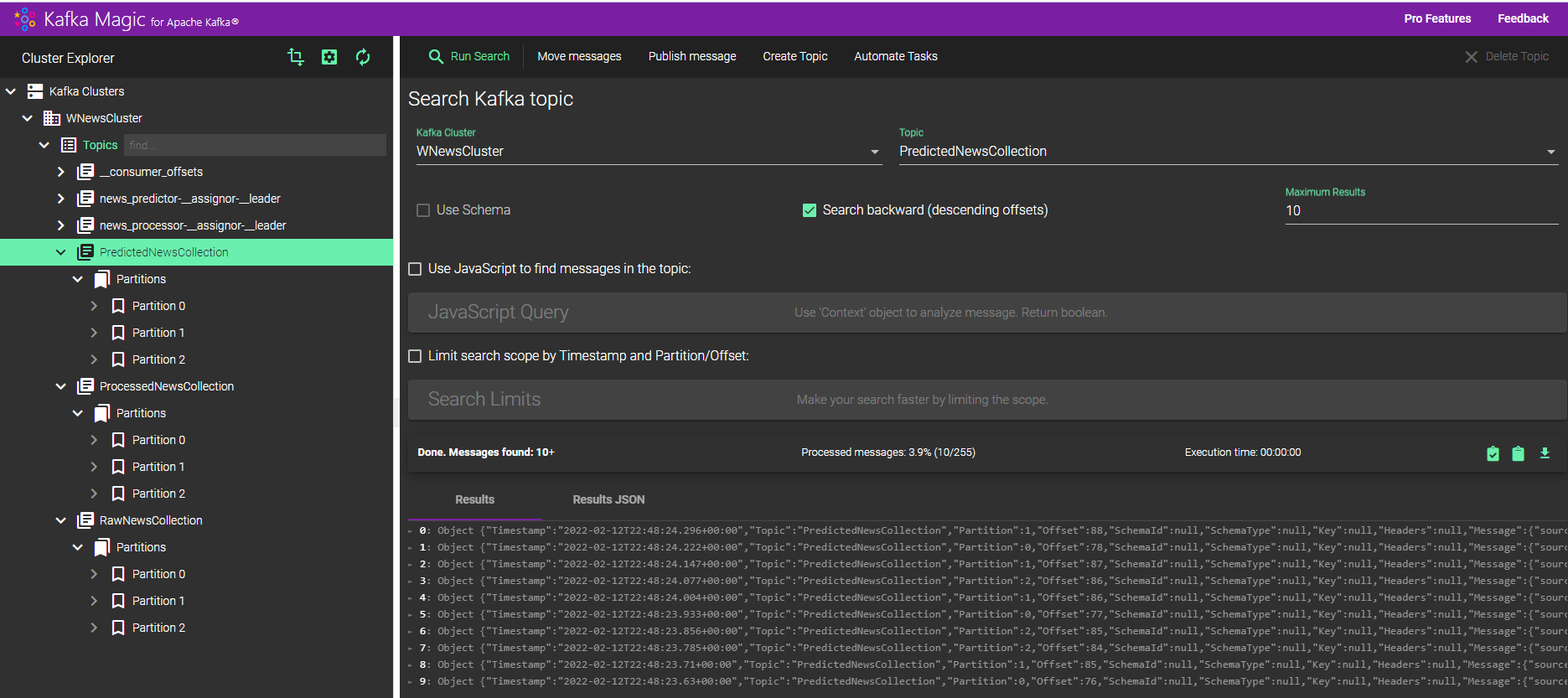


Рис. 5 – Интерфейс Kafka Magic

Также необходимо упомянуть об ещё одном понятии – Stream. В Kafka streams - это последовательность событий, которые получают из темы, над которой можно выполнять определенные операции, трансформации и затем результат отдать далее на обработку. В данном приложении kafka-streams используются в сервисах по обработке и классификации новостей, т.к. данные сервисы должны не только получить данные на обработку, но и вернуть обработанные данные обратно в кластер.

**Faust.** Приложение написано на языке программирования Python 3.8 с использованием библиотеки kafka-python v.2.0.2, однако в данной библиотеке нету нативной поддержки kafka-streams, поэтому для получения функциональности kafka-streams была использована библиотека Faust (https://faust.readthedocs.io/en/latest/), которая и добавляет необходимую отсутствующую логику в приложение.

## Источники новостей (Feeders)

Feeder-сервисы выступают источниками новостей в приложении и являются производителями для Kafka кластера. Эти сервисы выполняют 2 основные функции: запрашивают данные из источника и отправляют полученные на кластер.



Рис. 6 – Структура Feeder-сервисов

Текущая версия приложения реализует 2 источника (рис. 6):

1. TheGuardianFeeder – данные запрашиваются с помощью Api сайта https://www.theguardian.com
2. TheNYTimesFeeder – данные запрашиваются с помощью Api сайта <https://www.nytimes.com>

Для упрощения реализации новых фидеров был создан базовый класс сервиса, который содержит все необходимые методы для обработки и отправки данных на кластер (рис. 7)



Рис. 7 – Структура BaseFeeder



Рис. 8 – Пример использования BaseFeeder в TheNYTimes фидере

Т.к. группа Feeders является независимой, то отключение любого из источников никак не повлияет на всю систему в целом. Также при добавлении новых фидеров не нужно перезапускать или заново настраивать всю систему.

## Обработчики новостей (Processors)

Processor-сервисы отвечают за обработку полученных новостей и преобразование их к виду, с которым могут работать сервисы-классификаторы. Данные сервисы являются Faust-агентами, т.к. одновременно могут и принимать новые сообщения из RawNewsTopic, так и отправлять их обратно в ProcessedNewsTopic.



Рис. 9 – Структура Processor-сервисов

В текущей версии приложения процессор-сервис выполняет следующие действия, чтобы подготовить текст к классификации:

1. переводит текст в нижний регистр;

2. удаляет HTML тэги (если они есть в тексте);

3. заменяет URL на одно слово (“httpaddr”);

4. заменяет email-адреса на одно слово (“emailaddr”);

5. заменяет числа на одно слово (“number”);

6. заменяет знаки доллара ($) на слово “dollar”;

7. заменяет формы слов на исходное слово (такой подход называется stemming);

8. удаляет остальные символы и заменяет их пробелами;

9.  удаляет и заменяет "стоп-слова" (слова, не имеющие никакой информации) в тексте;

Для реализации этих операций были использованы регулярные выражения и Python библиотека nltk с английским корпусом текстов.

После обработки группы статей, объединённых одной темой, на выходе мы получаем набор векторов, каждый из которых состоит только из информационных слов в их начальной форме и словарь самых популярных слов по данной теме.

После этого каждая статья кодируется в виде бинарного вектора X, длина которого равна длине словаря. 0 на i-м месте означает, что i-e слово в словаре не встречалось в данной статье, 1 – наоборот.

Благодаря мультисервисной архитектуре при увеличении потока поступающих на обработку новостей (например, при увеличении чиста фидеров) можно без проблем развернуть новые processor-сервисы остановки всей системы.

## Классификаторы новостей (Predictors)

Predictor-сервисы необходимы для классификации новостей по 6 основным темам: экономика, фильмы, музыка, политика, наука и спорт. Если новость не подошла ни к одной из данных тем, то она будет классифицирована как общая (all). Данные сервисы также являются Faust-агентами, т.к. получают данные из ProcessedNewsTopic и возвращают их после классификации в PredictedNewsTopic.



Рис. 10 – Структура Predicted-сервисов

В данной реализации за классификацию новостей отвечает модуль, состоящий из 6 подряд идущих SVM ядер. Каждое ядро отвечает за определённую тему, и если оно предсказывает, что текущая новость подходит, то такая новость будет помечена специальным тегом. SVM ядра были импортированы из первой версии библиотеки и не подвергались никаким изменениям.

Обучение моделей происходило так: для каждого ядра был подготовлен свой словарь самых популярных слов (1200 слов) в статьях данной темы, и набор обработанных статей, состоящий из 12000 векторов (по 2000 на каждую тему).

Прежде чем обучать модели, необходимо выбрать правильные коэффициенты C и sigma для тренировки. Для этого был создан небольшой набор данных 3000 статей (по 500 с каждой темы) в качестве тренировочного набора и 600 (по 100 с каждой темы) в качестве валидационного. После этого алгоритмом полного перебора (brute-force) были выявлены коэффициенты для каждого ядра, на котором наблюдается наименьшее число отклонений.

* Спорт: С=100, sigma=0.0209
* Политика: С=100, sigma=0.0023
* Наука: С=100, sigma=0.0082
* Фильмы: С=100, sigma=0.0167
* Музыка: С=100, sigma=0.0182
* Экономика: С=100, sigma=0.0137

После получения коэффициентов, ядра были обучены на выборке из 12000 статей. Далее точность обучения была проверена на новой тестовой выборке в 2400 (по 400 статей каждой темы). Полученная точность на тестовой выборке:

* Спорт: 89.83333333333333%;
* Политика: 96.95833333333333%;
* Наука: 99.08333333333333%;
* Фильмы: 92.08333333333333%;
* Музыка: 92.66666666666666%;
* Экономика: 98.95833333333334%.

При текущем мультисервисной подходе можно использовать не только данную модель, но и любую другую, которая будет помечать новости определёнными тегами. Также данные модели можно включать/выключать/заменять/перенастраивать в режиме реального времени, не нанося вреда остальной системе.

## Клиенты кластера (Clients)

Клиенты-сервисы являются завершающей ступенью в обработке сообщений в серверной части приложения. Данные сервисы являются потребителями Kafka-кластера и предназначены для различных целей, которые будут рассмотрены далее.

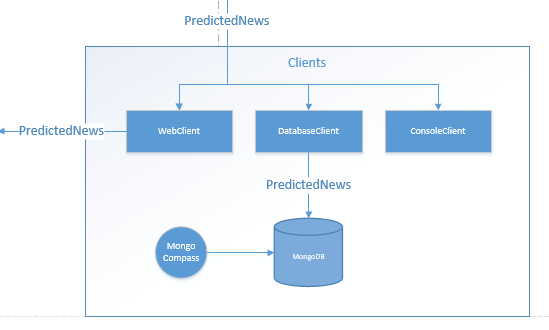


Рис. 10 – Структура Client-сервисов

В данной версии программного продукта существует 3 вида клиентов:

1. ConsoleClient – представляет собой обычной консольное приложение, необходимое для мониторинга общего состояния системы и его работоспособности. Позволяет следить за видом приходящих сообщений и упрощает поиск и отладку ошибок системы.
2. WebClient – является частью django сервера приложения, пересылает полученные сообщения подключенным клиентам.
3. DatabaseClient – является сервисом, который сохраняет классифицированные сообщения в базу данных.

Далее разберём структуру сохраняемых сообщений и способы отслеживания статистики по приходящим данным.

В прошлой версии продукта все данные хранились в MySQL базе данных, но, т.к. по своей структуре новости это неизменяемые объекты, то для этих целей лучше использовать NoSQL базу данных. Мой выбор пал на MongoDB, т.к. она показывает хорошие результаты для чтения/записи больших объёмов однотипной информации и у меня был опыт работы с ней.

MongoDB — документоориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Применяется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN [4].

Также в интернете можно найти сравнения производительностей MongoDB по сравнению с другими базами данных. Вот одна из статей [5], сравнивающая между собой MongoDB и MySQL (база, которая раньше использовалась в приложении). Некоторые графики сравнения приведены ниже (рис. 11, 12).



Рис. 11 – Сравнение скорости операции добавления

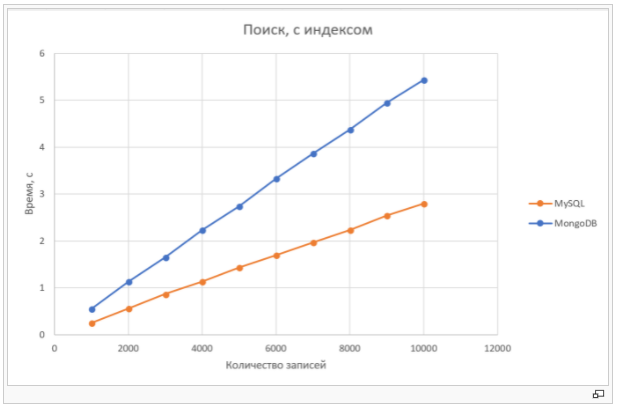


Рис. 12 – Сравнение скорости операции поиска

Для управления базой данных используется официальная графическая утилита MongoDB Compass — это интерактивный инструмент для запросов, оптимизации и анализа данных MongoDB. Пример интерфейса приложения можно увидеть на рис. 13.

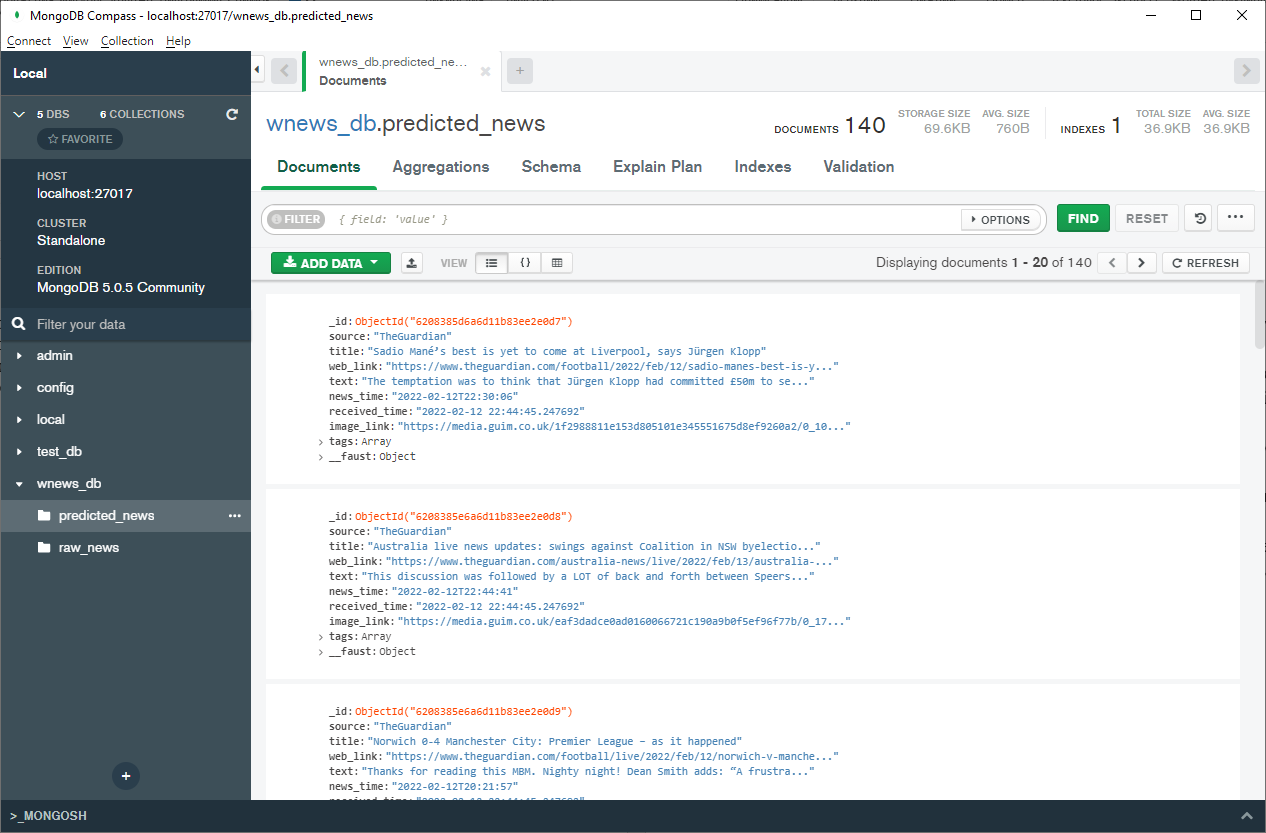


Рис. 13 – Интерфейс MongoDBCompass.

Классифицированные новости приходят как JSON объекты вида:

* \_id - уникальный номер статьи;
* source – имя фидера, который отправил статью;
* title – строка содержащая заголовок статьи;
* text – строка, содержащая обрезанный текст статьи (около 100 символов);
* web\_link – ссылка, по которой можно перейти к оригиналу статьи;
* image\_link – ссылка, по которой будет грузится превью статья;
* news\_time – дата, когда данная статья была размещена;
* receive\_time – время, когда статья была отправлена в кластер;
* tags – массив тегов, к которым может принадлежит данная статья

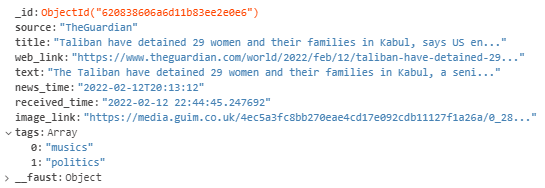


Рис. 14 – Пример классифицированной новости

# Multi-tenancy веб приложение

Первая версия веб приложения была написана с помощью фраемворка Django 3.1 и представляла собой одностраничное приложение со списком новостей и кнопками выбора тем (рис. 14). В качестве дальнейшего развития проекта было решено перевести веб часть приложения на multi-tenancy архитектуру.

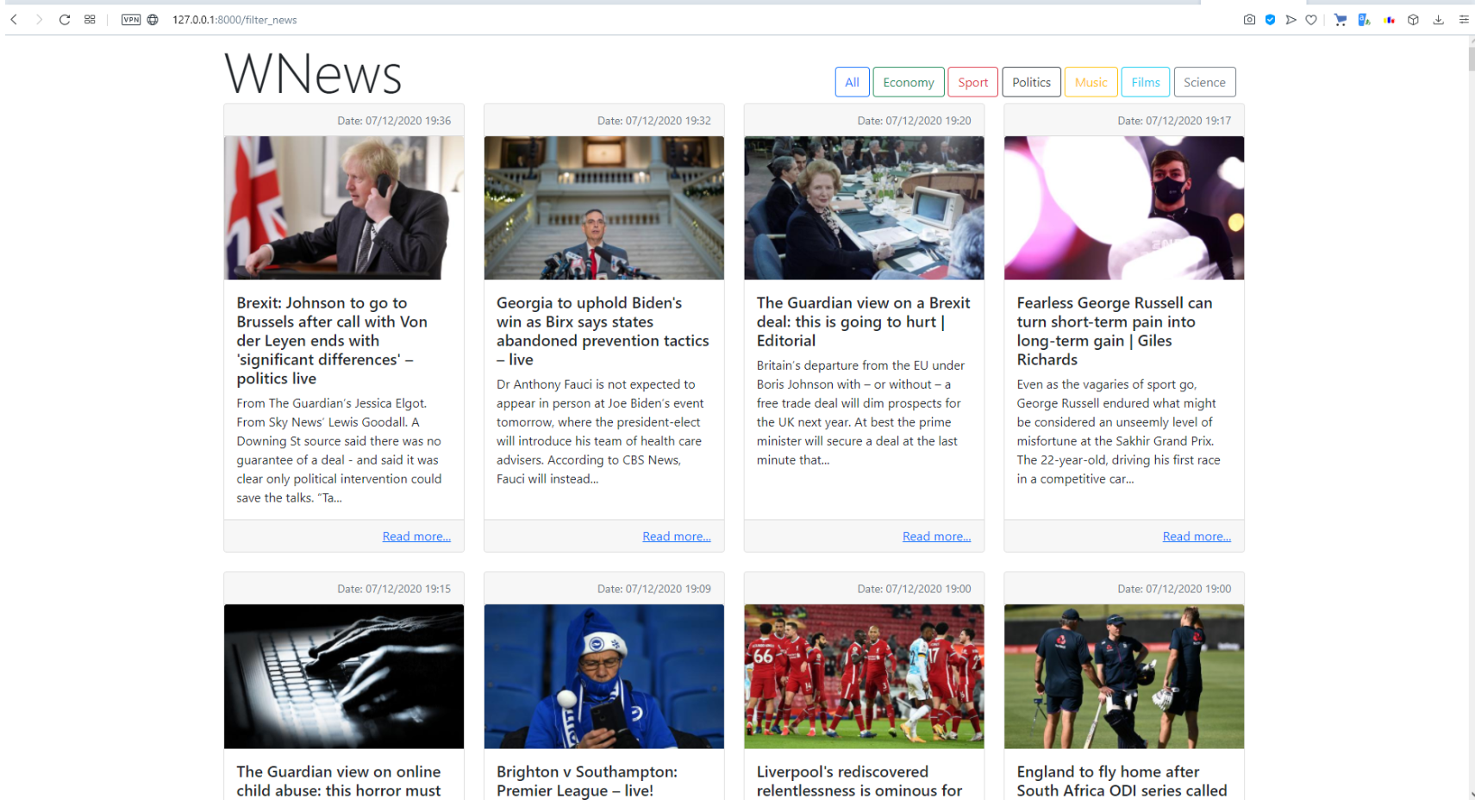


Рис. 14 – Интерфейс первой версии приложения

**Multi-tenancy (мультиарендность)** — элемент архитектуры программного обеспечения, где единый экземпляр приложения обслуживает множество организаций-клиентов («арендаторов»). Мультиарендность противопоставляется архитектуре из множественных экземпляров (англ. multiinstance), где для каждой организации-клиента создаются отдельные программные экземпляры. В мультиарендной архитектуре программные приложения работают одновременно с несколькими конфигурациями и наборами данных нескольких организаций, а каждая организация-клиент работает со своим экземпляром виртуального приложения, видя только свою конфигурацию и свой набор данных [6].

Также существует несколько сценариев по созданию multi-tenancy приложений (рис. 15):

1. **У каждого арендатора есть собственная инсталляция ПО, собственный пул ресурсов, собственное хранилище.** Наименее экономичный вариант, требует больше всего ресурсов поддержки. По сути своей, каждый арендатор получает собственную копию приложения. Так что и разработчикам нужно развивать каждую версию отдельно, и поддержка у них строится независимо. Если компании нужно контролировать всю ситуацию целиком, нужно строить некое интеграционное решение. Такой сценарий подходит для приложений, которые разрабатывались без расчёта на мультиарендное использование. Это не мультитенантность в полном смысле этого слова – скорее, способ добиться части её возможностей, когда переделывать продукт оказывается слишком дорого или нецелесообразно по другим причинам.
2. **Единое хранилище данных, одна инсталляция ПО, пул ресурсов у каждого тенанта свой.** В этом случае разделение между арендаторами происходит на уровне инфраструктуры. Каждая группа пользователей заходит на свой URL или подключается к собственному серверу. В этом случае ресурсы используются менее экономично, зато у каждого арендатора есть гарантированные мощности – ведь он использует собственный пул.
3. **Единый пул ресурсов, единое хранилище данных, одна инсталляция, разделение на уровне софта.** Такое приложение само понимает, куда пустить каких пользователей и какие функции им требуются. Этот сценарий обеспечивает оптимальное использование ресурсов, простое администрирование и развитие ПО. Это мультиарендность на уровне ядра [7].

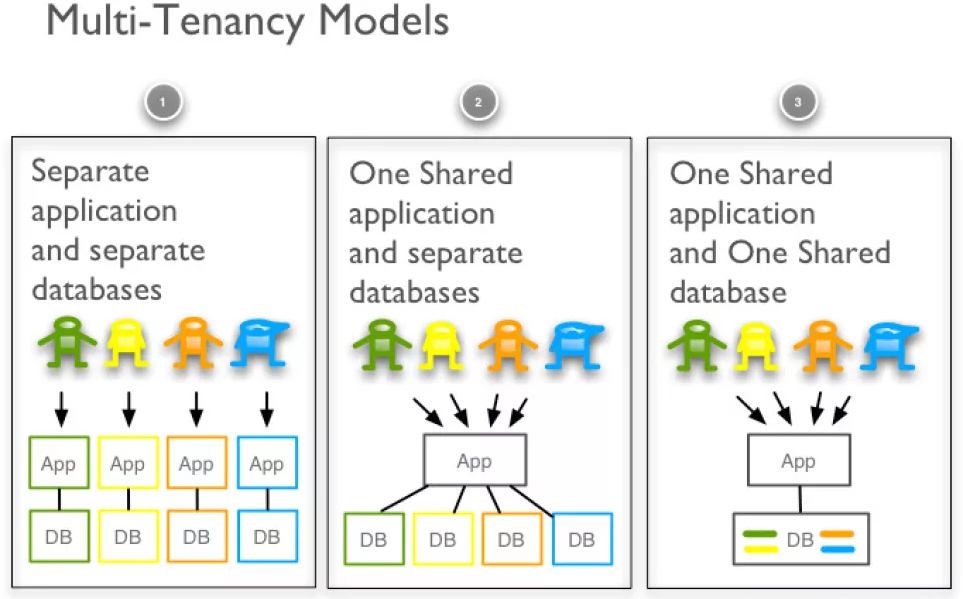


Рис. 15 – Виды мультитенантных архитектур

## Общая архитектура

Для второй версии приложения было решено выбрать 2 подход: сделать общее приложения, которое будет разворачивать несколько отдельных сайтов по различным адресам с собственными настройками. Разделение данных реализовано с помощью базы данных PostgreSQL и её технологии Scheme (отсутствие аналогичной логики в MySQL привело к тому, что от неё пришлось отказаться). В общем виде новая архитектура выглядит так (рис. 16):



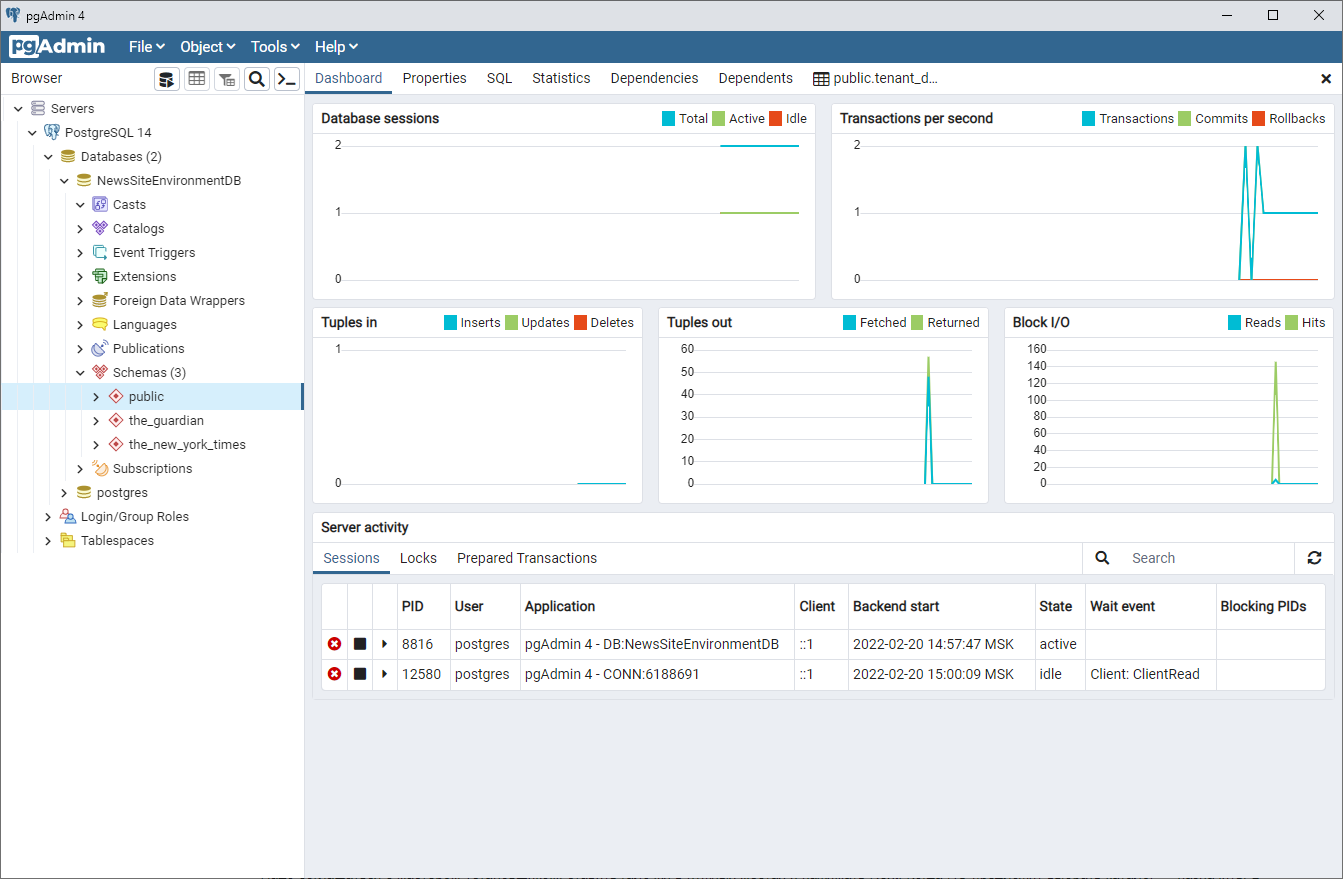
Рис. 16 – Общая архитектура второй версии веб части

Новая реализация веб версии приложения состоит из (рис. 17):

1. DjangoServer – сервер, написанный на фраемворке Django 3.2. Для реализации multi-tenant архитектуры использована библиотека django-tenant v3.4.2
2. В качестве базы данных была использована PostgreSQL 14
3. В для демонстрации мультиарендности архитектуры были развёрнуты следующие тенанты:
   1. theguardian.wnews.com - реализация сайта с новостями только TheGuardian фидера. Информация о пользователях и остальные данные хранятся в TheGuardianShema
   2. nyt.wnews.com - реализация сайта с новостями только TheNewYorkTimes фидера. Информация о пользователях и остальные данные хранятся в TheNewYorkTimesSchema
   3. wnews.admin.com – доступ на сайт доступен только администраторам, хранит всю информацию о инфраструктуре тенантов, пользователей и приложении. Связан с PublicShema.
4. PgAdmin — это программа кроссплатформенного типа для работы с PostgreSQL-серверами. Подключив ПО, пользователи смогут создавать SQL-скрипты, отслеживать процессы и оперировать несколькими БД (рис 18).



Рис. 17 – Подробная архитектура второй версии веб части

 Рис. 18 – Интерфейс PgAdmin

## Анализ

### Solution modularity

Способность каждого компонента быть модульно интегрированным и, таким образом, не зависеть от внутренней реализации внешнего компонента.

#### Flexibility

Система должна быть гибка для добавления или замены компонентов. Отдельные модули независимы по отношению к другим компонентам.

Как было показано на рисунках 2 и 3, вся система разбита на сервисы, которые общаются между собой передавая сообщения по шине. Выключение/удаление любого модуля (кроме самой шины или zookeeper) никак не повлияет на систему. При увеличении нагрузки можно без проблем и каких-либо внутренних изменений поднять ещё пару сервисов, на проблемном участке архитектуры.

#### Decoupled

Компоненты взаимодействуют через стандартизированные интерфейсы, автономны и не знают друг о друге.

Фидеры и Процессоры общаются между собой посредством объектов в формате JSON следующего вида (рис. 19):

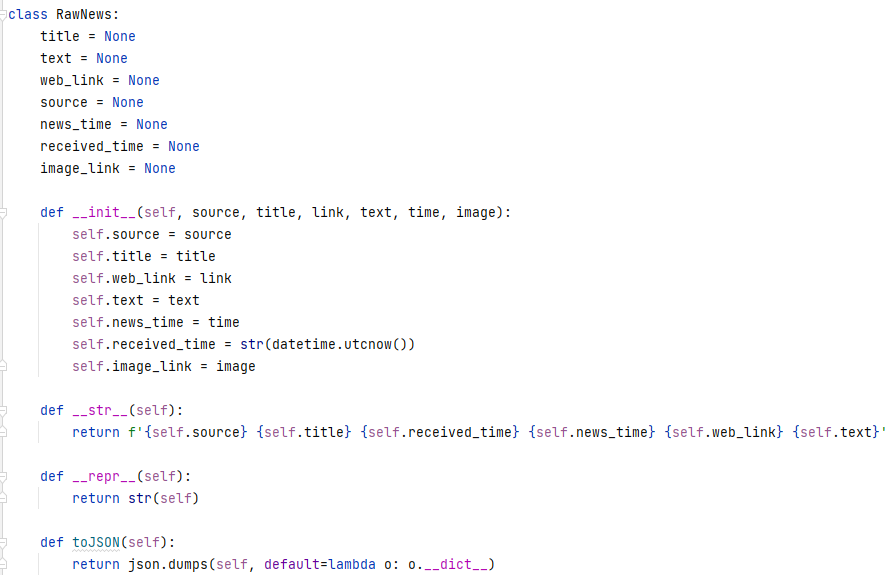


Рис. 19 – Формат RawNews

Процессоры и предикторы общаются между собой посредством объектов в формате JSON такого вида (рис. 20):

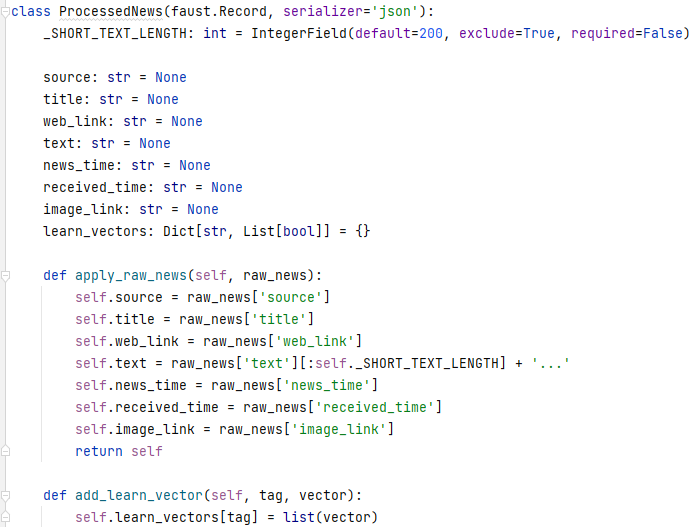


Рис. 20 – Формат ProcessedNews

А процессоры и предикторы общаются между собой объектами в формате JSON (рис. 21):

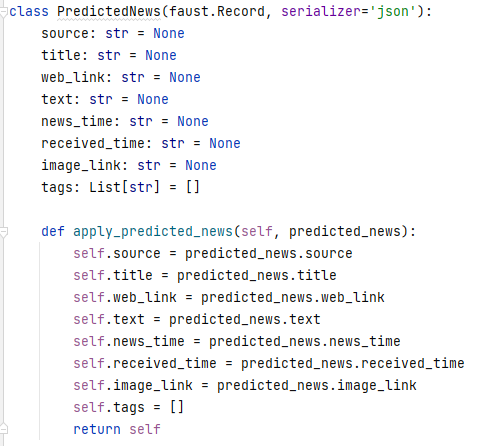


Рис. 21 – Формат PredictedNews

Таким образом, любой компонент, который знает, по какому адресу и к какому топику подключиться + работает с сообщениями данных форматов может быть легко интегрирован в систему без изменения логики работы других модулей.

### Observability

Насколько хорошо существующие компоненты и их внутренние состояния могут быть выведены из знаний о его различных выходах, чтобы получить обзор текущих действий в системе.

#### Monitoring

Существующие функции мониторинга работают и могут обнаруживать системные аномалии

Для мониторинга состояния системы могут использоваться различные инструменты. Так, для мониторинга состояния веб части приложения используется pgAdmin (рис. 22).

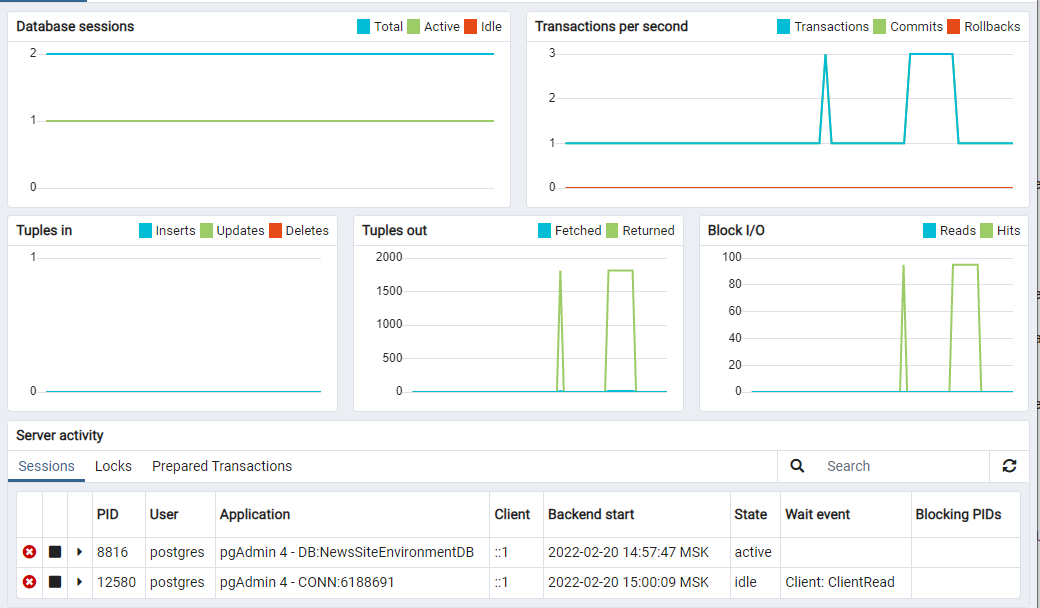


Рис. 22 – Мониторинг состояния веб части с помощью PgAdmin

Для мониторинга состояния серверной части используется MongoCompass (рис. 23):



Рис. 23 – Мониторинг состояния серверной части с помощью PgAdmin

#### Transactional tracing

Возможность сквозного отслеживания события через различные компоненты

Для сквозного отслеживания системы используются консольные окна с данными о состоянии системы в реальном времени. Отслеживание текущего состояния Kafka кластера выглядит следующим образом (рис. 24).

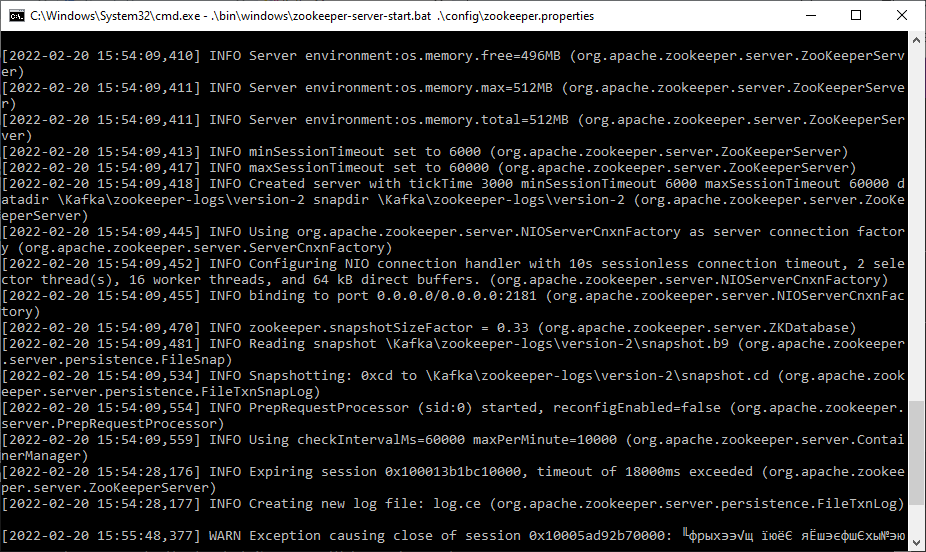


Рис. 24 – Состояние Kafka кластера в окне Zookeper

Состояние Faust агентов также можно отслеживать в специальных консольных окнах (рис. 25):

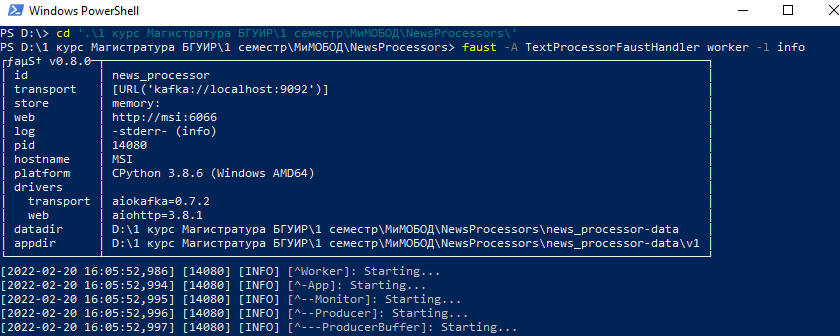


Рис. 25 – Состояние Faust агента одного из процессоров

#### Log aggregation

Журналы всей системы объединены, организованы и доступны для поиска.

Kafka кластер по умолчанию хранит и агрегирует данные в одном месте и текущая инфраструктура выглядит как-то так (рис. 26):

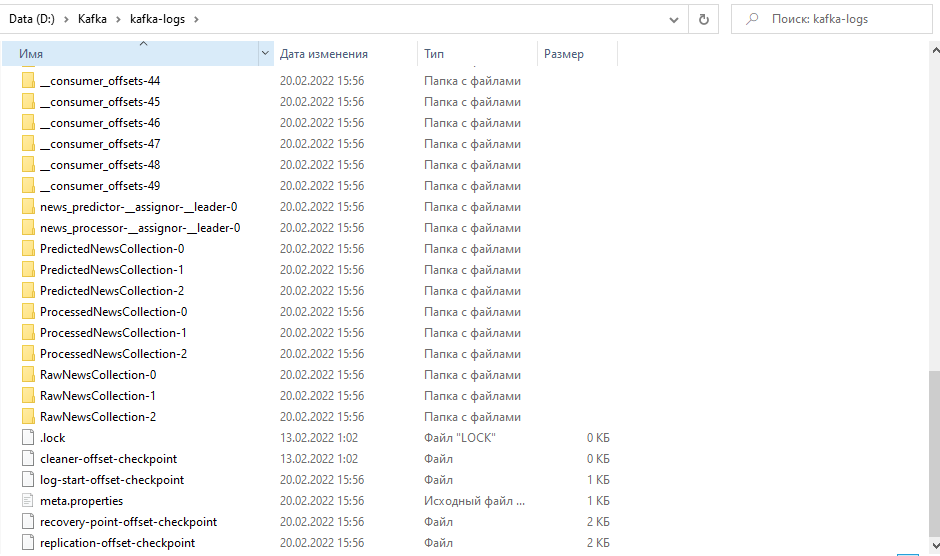


Рис. 26 – Хранение агрегированых логов Kafka кластера

При необходимости поиска и визуализации логов можно воспользоваться KafkaMagic (рис. 27):

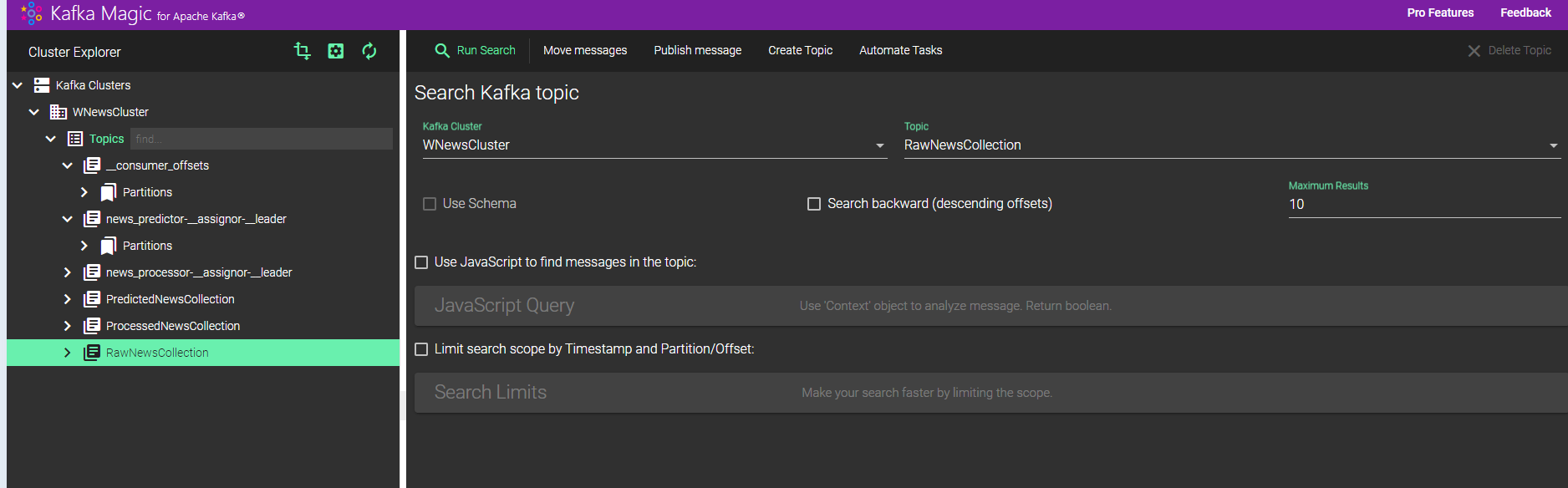


Рис. 27 – KafkaMagic для отслеживания состояния кластера

### Content vs. feature de-coupling

Способность системы адаптировать контент независимо от разработки функций.

#### Independent content creation

Данные и предопределенное поведение можно настроить и включить независимо для каждого арендатора без необходимости изменения кода.

По своей логики текущее приложения является довольно простым и в данный момент нет необходимости включать/выключать определённую логику. Однако, данная возможность была предусмотрена в администраторской версии приложения. Так, например, я есть возможность добавления/изменения источника данных для тенанта, добавление новых пользователей, администраторов, добавления новых доменов для тенантов и их группировка (рис. 28)

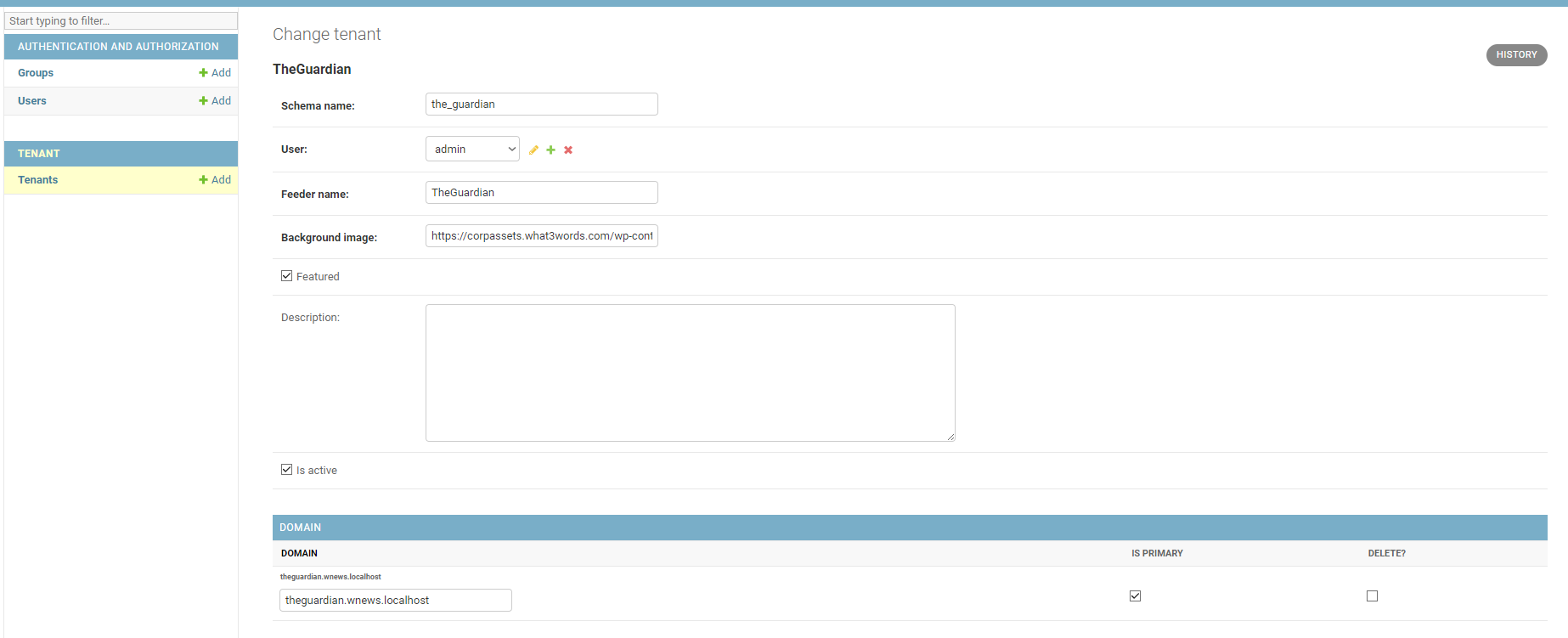


Рис. 28 – Интерфейс WNews admin

#### Separation of concerns

Различные задачи разделены в архитектуре, что делает ее более удобной в сопровождении.

Текущая версия приложения содержит ещё не много компонентов, однако, они могут быть легко разделены и интегрированы в администраторскую часть, для управления. В качестве примера можно привести логику управления тенантами (рис. 29) и код, отвечающий за данный доступ (рис. 30):

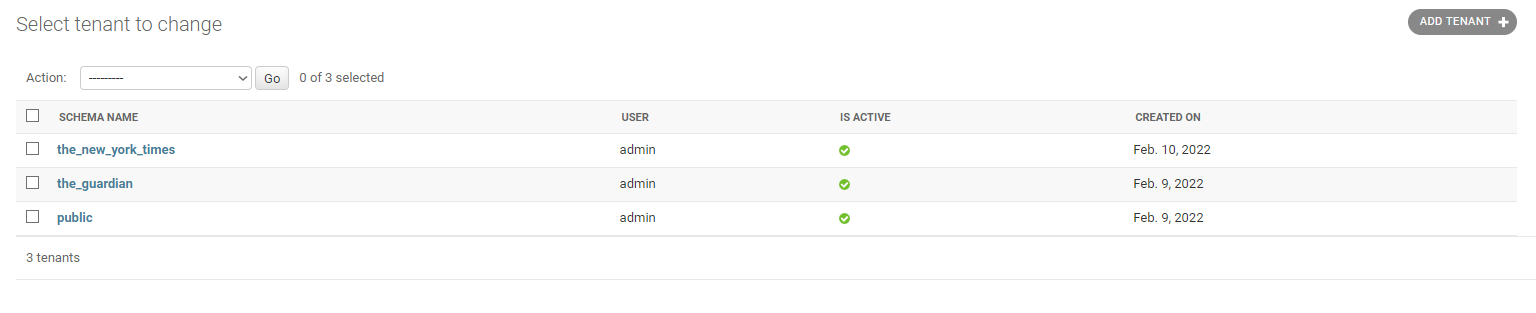


Рис. 29 – Логика управления тенантами в администраторской части приложения

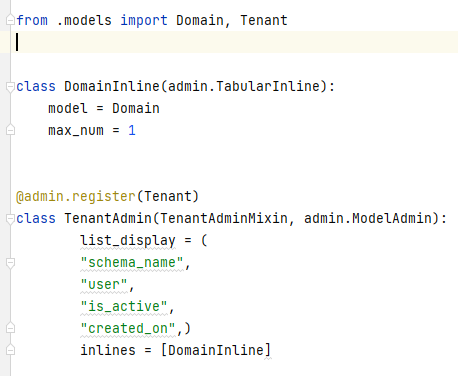


Рис. 30 – Код регистрации и управления логики тенантов из рис. 29

### “Shift Right Ops” readiness

Проверьте способность системы обрабатываться рынком/брендом («сдвиг вправо»), т. е. как работает самообслуживание и как рынки/бренды могут внедряться.

#### Configuration options

Рынки/бренды могут персонализировать и настраивать компоненты в зависимости от потребностей.

В качестве примера конфигурации тенантов можно привести логику изменения фонового изображения на сайтах (рис. 31, 32):



Рис. 31 – Страница, доступная по адресу http://theguardian.wnews.localhost:8000/login



Рис. 31 – Страница, доступная по адресу <http://nyt.wnews.localhost:8000/login>

Настройка ссылки на фоновое изображение доступно в администраторской части приложения.

#### Component responsibilities

Ответственность за управление тенантом возлагается на администратора тенанта. Доступ к администрированию конкретного тенанта доступен по адресу <http://theguardian.wnews.localhost:8000/admin> (для TheGuardian тенанта). В данный момент доступно управление следующими частями тенанта (рис. 33).

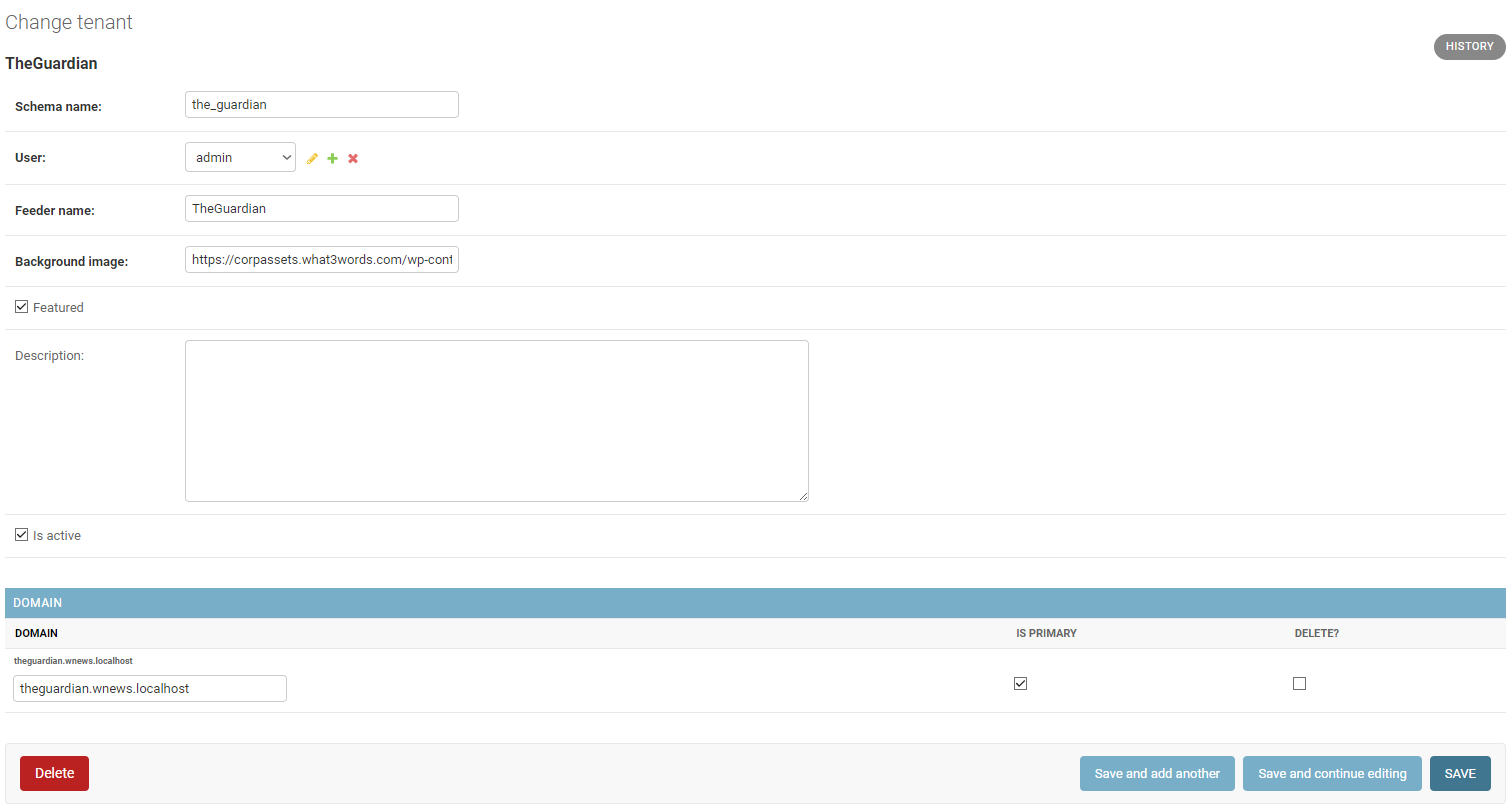


Рис. 33 – Управление тенанта TheGuardian

### Tenant provisioning easiness

Проверьте способность системы работать с несколькими арендаторами, т. е. допускать нескольких арендаторов в пределах одного экземпляра сервера.

#### Tenant management

Имеются достаточные варианты интеграции арендатора (например, интеграции данных)

Для добавления нового тенанта достаточно иметь права администратора wnews и зайти по адресу /admin/tenant/tenant/add/. Окно добавления нового тенанта выглядит так (рис. 34):

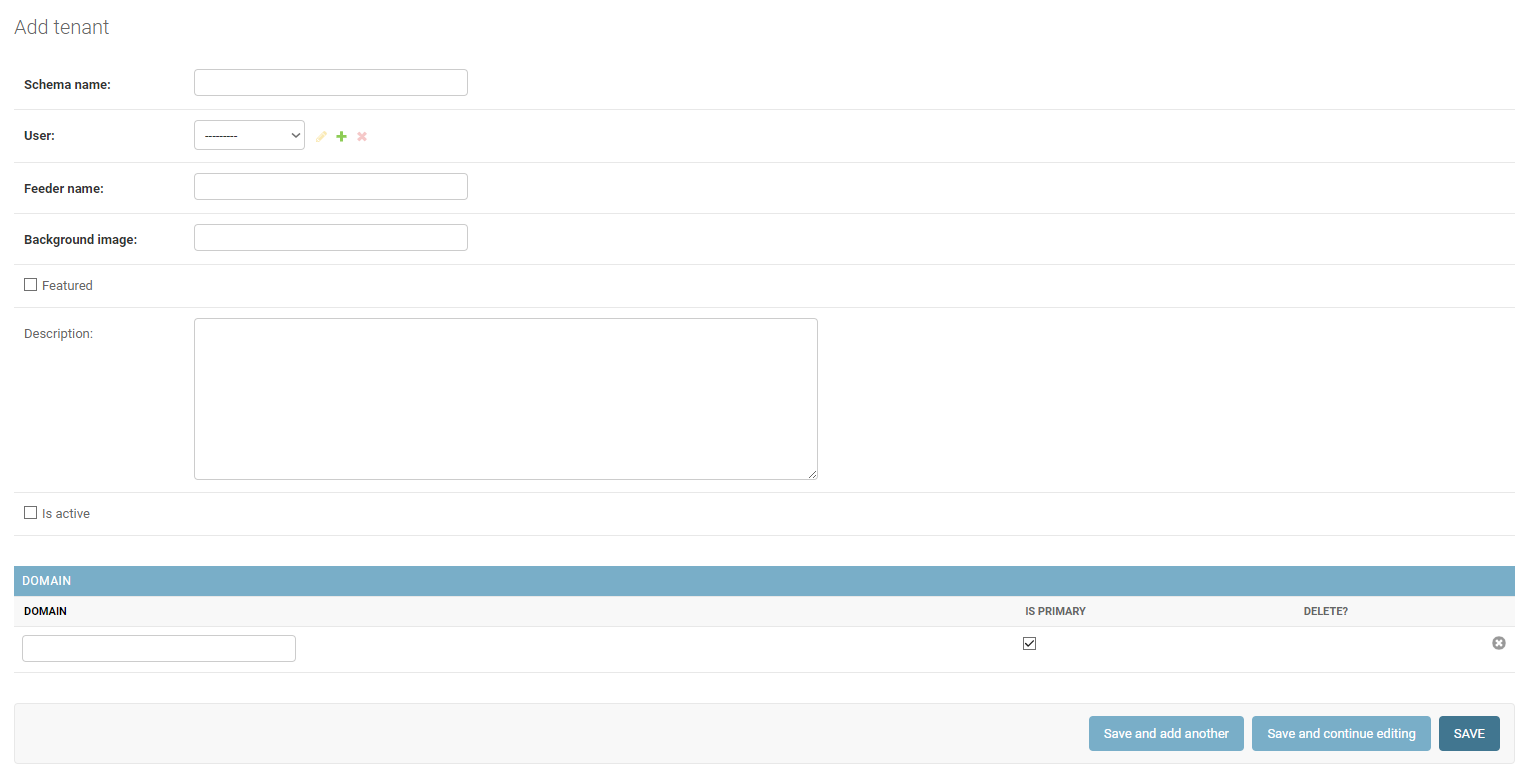


Рис. 34 – Добавление нового тенанта в приложение

#### Tenant & data segregation

Система может разделять данные, принадлежащие одному объекту (арендатору).

После добавления нового тенанта автоматически в PostgreSQL добавится новая схема со всеми необходимыми для работы настройками и данными (рис. 35). Структура данной схемы будет аналогична схемам других тенантов, за исключением public схемы. Данная схема является родительской для остальных схем, и её администраторы являются полными администраторами дочерних схем тоже. Также данная схема хранит метоинформацию дочерних схем, для возможности управления ими (рис. 36)

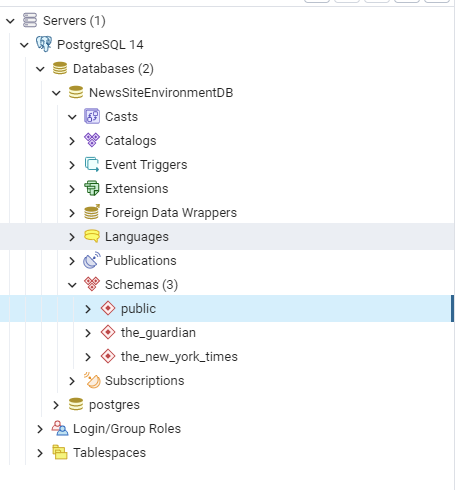


Рис. 35 – Расположение схем тенантов в одной базе

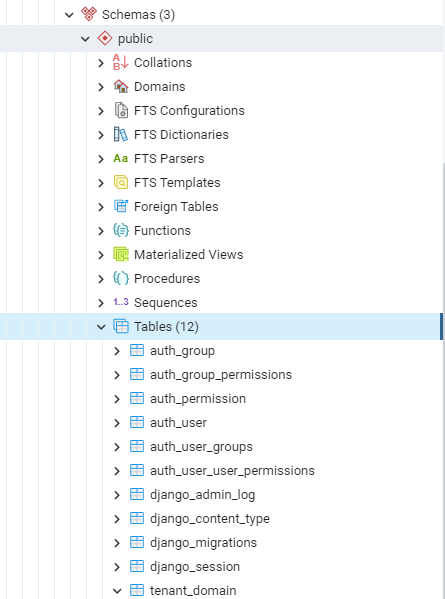


Рис. 36 – Структура таблиц в public схеме

### Deployment patterns

Обеспечьте беспроблемное развертывание и управление выпусками для всех компонентов, чтобы обеспечить бесперебойную работу пользователей.

#### Pattern strategy

Система использует стратегию развертывания, которая лучше всего подходит для ее вариантов использования (синий/зеленый, канареечный или скользящий).

Т.к. данное приложение в текущей реализации невелико, то для обновления версии сервера подойдёт и обычный подход грубой силы, когда сервер отключается на некоторое время, на него устанавливается новая версия и после он опять доступен для пользователей.

Однако, при росте числа пользователей, мне кажется более логичным перейти на сине-зелёную стратегию выпуска версий, т.к. для остальных стратегий (канареечной и скользящей) необходим довольно крупный пул серверов, пользователи в приложении разделены по тенантам инфраструктурным, т.е. они не используют 2 независимых версии продукта. Таким образом, наиболее логичным вариантом, я бы обновлял все сервера по сине-зелёной стратегии (заранее подготовив сервер с новой версией и переключив трафик на него).

# Вывод

Таким образом, для второй версии приложения была проделана большая работа по переводу системы из монолитной архитектуры в мультисервисную. Были изучены технологии Kafka и Faust. Веб часть приложения была переведена на мультиарендную архитектуру. Были добавлены новые возможности и исправлены старые ошибки. В конце работы был проведён анализ по новым возможностям системы.

# Список источников

1. Определение Apache Kafka [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Apache_Kafka#cite_note-5>
2. Топики и разделы Apache Kafka [Электронный ресурс] – Режим доступа: https://cloud.yandex.ru/docs/managed-kafka/concepts/topics
3. Производители и потребители Apache Kafka [Электронный ресурс] – Режим доступа: https://cloud.yandex.ru/docs/managed-kafka/concepts/producers-consumers
4. MongoDB [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/MongoDB
5. MongoDB vs MySQL [Электронный ресурс] – Режим доступа: https://ru.bmstu.wiki/Сравнение\_производительности\_MongoDB\_vs\_MySQL
6. Мультиарендность [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Мультиарендность
7. Мультитенантность: как вырастить из одного приложения линейку независимых продуктов [Электронный ресурс] – Режим доступа: https://vc.ru/u/99035-true-engineering/167174-multitenantnost-kak-vyrastit-iz-odnogo-prilozheniya-lineyku-nezavisimyh-produktov