

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение  
высшего образования

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ  
имени первого Президента России Б.Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра высокопроизводительных компьютерных технологий

РАЗРАБОТКА И ВНЕДРЕНИЕ СИСТЕМЫ МОНИТОРИНГА НОВОСТНЫХ  
РЕСУРСОВ: РЕАЛИЗАЦИЯ ФИЛЬТРАЦИОННОГО СЛОЯ,  
РАЗВЕРТЫВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ И  
ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ КОМПОНЕНТОВ

Направление подготовки 02.03.02 Фундаментальная информатика и  
информационные технологии

Заведующий кафедрой:  
д. ф.-м. н., М. Ю. Филимонов

Выпускная квалификационная  
работа бакалавра  
**Шишкина Александра  
Евгеньевича**

Нормоконтролер:  
А. Ю. Берсенов

Научный руководитель:  
А. Ю. Берсенов

Научный соруководитель:  
к. ф.-м. н., проф., В. С. Зверев

Екатеринбург  
2025

## РЕФЕРАТ

Шишкин А. Е. РАЗРАБОТКА И ВНЕДРЕНИЕ СИСТЕМЫ МОНИТОРИНГА НОВОСТНЫХ РЕСУРСОВ: РЕАЛИЗАЦИЯ ФИЛЬТРАЦИОННОГО СЛОЯ, РАЗВЕРТЫВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ И ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ КОМПОНЕНТОВ, выпускная квалификационная работа.

Объём работы: 36 страниц, 2 иллюстраций, 14 источников.

Ключевые слова: фильтрационный слой, фильтрация новостей, мониторинг новостей, дедупликация, ключевые слова, анализ тональности, отказоустойчивость, Docker, ElasticSearch, RabbitMQ, MongoDB, репликация, мониторинг компонентов, логирование.

Цель работы: разработка фильтрационного слоя системы мониторинга новостных ресурсов и внедрение микросервисной архитектуры с обеспечением отказоустойчивости её компонентов.

Методы работы: В работе применяются методы построения микросервисных систем, организации отказоустойчивых инфраструктурных решений, алгоритмы обработки текстовых данных и анализа тональности, методы дедупликации и выявления аномалий. Используются технологии Docker, MongoDB, RabbitMQ, ElasticSearch и языки программирования Python и C#.

Результат работы: система демонстрирует повышение надёжности и отказоустойчивости мониторинга новостей, а также улучшение качества фильтрации данных. Разработанная система может быть использована для мониторинга репутации в СМИ и социальных сетях.

Новизна работы: В условиях стремительного роста объёмов информационного потока в СМИ и социальных сетях особенно актуальной становится задача оперативной фильтрации и анализа новостей. Разработанная система предлагает комплексное решение, объединяющее интеллектуальный фильтрационный слой с микросервисной архитектурой.

## СОДЕРЖАНИЕ

<b>ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ</b> . . . . .	5
<b>ВВЕДЕНИЕ</b> . . . . .	8
<b>ОСНОВНАЯ ЧАСТЬ</b> . . . . .	10
1 Анализ методов фильтрации новостного контента и построения отказоустойчивых систем . . . . .	10
1.1 Анализ существующих решений мониторинга новост- ного контента . . . . .	10
1.2 Анализ существующих подходов к построению отказо- устойчивых систем . . . . .	12
1.3 Анализ существующих решений сбора телеметрии си- стемы и хранения логов . . . . .	13
1.4 Выводы по главе 1 . . . . .	13
2 Постановка задачи . . . . .	16
2.1 Общая характеристика решаемой задачи . . . . .	16
2.2 Цель работы . . . . .	16
2.3 Задачи исследования . . . . .	16
2.4 Ожидаемые результаты . . . . .	17
3 Реализация фильтрационного слоя новостей . . . . .	18
3.1 Обоснование выбора технологий . . . . .	18
3.2 Общая архитектура системы мониторинга новостей . .	19
3.3 Архитектура фильтрационного слоя . . . . .	20
3.4 Реализация сервиса Filtrator . . . . .	22
3.5 Микросервис TextProcessor . . . . .	24
3.6 Микросервис Classifier . . . . .	25
4 Развёртывание и обеспечение отказоустойчивости . . . . .	28

4.1	Общая схема развёртывания . . . . .	28
4.2	Настройка Docker Swarm . . . . .	28
4.3	Настройка MongoDB . . . . .	29
4.4	Настройка RabbitMQ . . . . .	29
4.5	Настройка Elasticsearch, Kibana и Filebeat . . . . .	29
4.6	Обеспечение отказоустойчивости . . . . .	30
4.7	Логирование и мониторинг . . . . .	31
4.8	Гибкая конфигурация сервисов . . . . .	31
4.9	Безопасность . . . . .	31
4.10	Тестирование устойчивости . . . . .	32
<b>ЗАКЛЮЧЕНИЕ . . . . .</b>		<b>33</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ</b>		<b>35</b>

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей работе применяются следующие обозначения и сокращения:

**APM (Application Performance Monitoring)** — система мониторинга производительности приложений.

**BangerProbability** — вероятность широкой огласки (вирусности) новостного сообщения, определяемая с помощью модели машинного обучения.

**Batched inference** — обработка группы сообщений за один вызов модели.

**CI/CD** — непрерывная интеграция и доставка, автоматизирующая процесс сборки и деплоя.

**Classifier** — микросервис, оценивающий вирусный потенциал новостей (BangerProbability).

**Docker Swarm** — встроенный оркестратор контейнеров Docker, применяемый для автоматизированного масштабирования и управления микросервисами.

**Docker** — платформа контейнеризации, используемая для изоляции и развёртывания компонентов системы.

**ElasticSearch** — полнотекстовый поисковый движок, применяемый для индексирования новостей и логов.

**FastAPI** — веб-фреймворк для создания REST API на Python, применяемый в сервисах TextProcessor и Classifier.

**Filebeat** — агент для сбора логов из контейнеров и их отправки в ElasticSearch.

**Filtrator** — основной фильтрационный микросервис, реализующий все этапы обработки новостей.

**Healthcheck** — механизм автоматической проверки «здоровья» контейнеров.

**Kibana** — инструмент визуализации данных из ElasticSearch (в частности — логов и метрик).

**MongoDB** — документо-ориентированная база данных, используемая для хранения новостей, метаданных и пользовательских настроек.

**Named Entity Recognition (NER)** — извлечение именованных сущностей из текста (планируется внедрение).

**NewsFilterService** — компонент для семантического сравнения новостей при дедупликации.

**NewsToSend** — коллекция в MongoDB, содержащая новости, готовые к отправке пользователям.

**REST API** — архитектурный стиль взаимодействия между компонентами системы по протоколу HTTP.

**RabbitMQ** — брокер сообщений, обеспечивающий асинхронное взаимодействие между микросервисами системы.

**Retry policy (Политика повторных попыток)** — механизм обработки временных сбоев во взаимодействии сервисов.

**Telegram-бот (TG-бот)** — компонент, осуществляющий доставку отобранных новостей пользователям через мессенджер Telegram.

**TextProcessor** — микросервис, отвечающий за векторизацию текста и определение его эмоциональной окраски.

**Vectorization** — процесс преобразования текста в вектор признаков с помощью предобученных моделей.

**WireGuard** — VPN-протокол, обеспечивающий безопасное соединение между сервером и внешними узлами.

**Анализ тональности (Sentiment Analysis)** — метод обработки текстов, определяющий эмоциональную окраску сообщения (положительную, отрицательную или нейтральную).

**Дедупликация** — процесс выявления и удаления повторяющихся или семантически схожих новостных сообщений.

**Ключевые слова** — заранее определённые слова или фразы, служащие фильтром релевантного новостного контента.

**Репликация MongoDB** — механизм обеспечения отказоустойчивости и доступности данных путём создания копий базы данных.

**Фильтрационный слой** — компонент системы, выполняющий интеллектуальный отбор, анализ и обработку новостных данных по заданным критериям (дедупликация, анализ тональности, фильтрация по ключевым словам).

## **ВВЕДЕНИЕ**

### **Актуальность**

В современном мире, характеризующемся высоким темпом появления и распространения информации, актуальными становятся задачи мониторинга и анализа новостных ресурсов. Большие объёмы данных, поступающих из различных источников (RSS-ленты, социальные сети, мессенджеры), требуют использования автоматизированных систем обработки и фильтрации контента. Одной из ключевых проблем является не только своевременное получение данных, но и их качественная фильтрация по критериям релевантности, достоверности и значимости для пользователя.

Повышенные требования к надёжности и стабильности таких систем обуславливают необходимость использования отказоустойчивых решений и развертывания компонентов в распределённой среде. Для этого применяются микросервисные архитектуры с обеспечением мониторинга и высокой доступности всех сервисов.

**Цель выпускной квалификационной работы** – разработка фильтрационного слоя системы мониторинга новостных ресурсов и внедрение микросервисной архитектуры с обеспечением отказоустойчивости её компонентов.

### **Задачи выпускной квалификационной работы:**

1. Реализация фильтрационного слоя с поддержкой функций:
  - дедупликации новостных сообщений;
  - фильтрации по ключевым словам;
  - определения тональности публикаций;
  - определения потенциальной вирусности публикации;
2. Развертывание компонентов системы на базе контейнерной платформы Docker.
3. Обеспечение отказоустойчивости системы на уровне микросервисов и инфраструктуры.



4. Настройка систем логирования.
5. Проведение тестирования отказоустойчивости системы.

**Объект исследования** — система мониторинга новостных ресурсов и потоков данных из различных источников.

**Предмет исследования** — архитектура отказоустойчивой системы обработки данных с реализацией фильтрационного слоя.

### **Методы исследования**

В работе применяются методы построения микросервисных систем, организации отказоустойчивых инфраструктурных решений, алгоритмы обработки текстовых данных и анализа тональности, методы дедупликации и выявления аномалий. Используются технологии Docker, MongoDB, RabbitMQ, Elasticsearch и языки программирования Python и C#.

### **Практическая значимость**

Результатом исследования является внедрённая система мониторинга новостей, обеспечивающая высокую отказоустойчивость и достоверность фильтрации информации. Решение может быть применено в системах корпоративной аналитики, службах мониторинга репутации и СМИ, а также в государственных информационных системах.

### **Структура работы**

Работа состоит из введения, трёх глав, заключения, списка использованных источников и приложений.

1. В первой главе рассматриваются существующие подходы к мониторингу новостей и фильтрации данных, а также архитектурные решения для построения отказоустойчивых систем.
2. Во второй главе выполняется постановка задачи
3. Во третьей главе описывается реализация фильтрационного слоя новостей.
4. В четвертой главе приведены решения по развертыванию компонентов системы и обеспечению их отказоустойчивости.

## ОСНОВНАЯ ЧАСТЬ

### 1 Анализ методов фильтрации новостного контента и построения отказоустойчивых систем

#### 1.1 Анализ существующих решений мониторинга новостного контента

##### 1.1.1 Дедупликация

Дедупликация — ключевой этап в обработке новостного контента, направленный на устранение повторяющихся или схожих сообщений. Существуют различные методы дедупликации:

- **Хеширование:** использование хеш-функций для идентификации идентичных текстов.
- **Сравнение по ключевым словам:** выделение и сравнение ключевых слов в текстах для определения схожести.
- **Семантический анализ:** применение методов обработки естественного языка для выявления смысловой близости текстов.

Современные исследования предлагают адаптивные методы дедупликации, учитывающие контекст и структуру данных, что повышает точность фильтрации.

##### 1.1.2 Фильтрация по ключевым словам

Фильтрация по ключевым словам является основным инструментом для отбора релевантного новостного контента. Существуют различные подходы:

- **Прямое сопоставление:** поиск точных совпадений ключевых слов в тексте.
- **Использование регулярных выражений:** позволяет учитывать различные формы слов и фраз.
- **Семантический анализ:** учет синонимов и контекста для более гибкой фильтрации.

Инструменты, такие как Elasticsearch, предоставляют мощные возможности для реализации фильтрации по ключевым словам с использованием

различных методов анализа текста.

### 1.1.3 Определение эмоционального окраса

Анализ эмоционального окраса (сентимент-анализ) позволяет классифицировать новостные сообщения по тональности: положительной, отрицательной или нейтральной. Существуют различные методы:

- **Словарные методы:** использование заранее составленных словарей с оценкой эмоциональной нагрузки слов.
- **Машинное обучение:** обучение моделей на размеченных данных для определения тональности.
- **Гибридные подходы:** сочетание словарных методов и машинного обучения для повышения точности.

Современные исследования подчеркивают эффективность гибридных методов, особенно при анализе коротких текстов, таких как заголовки новостей.

### 1.1.4 Вычисление потенциальной вирусности

Оценка потенциальной вирусности новостного контента представляет собой задачу предсказания вероятности быстрого и широкого распространения материала в информационной среде. Для этого используются методы из областей машинного обучения, анализа графов и анализа текстов.

Наиболее распространённые подходы включают:

- **Регрессионные модели (Logistic Regression, Random Forest):** Используются для предсказания бинарного события — станет ли материал вирусным или нет, на основе метаданных (время публикации, количество слов, источник, первые реакции). Источник: Ma, J., Gao, W., & Wong, K.-F. (2018). Rumor detection on Twitter with tree-structured recursive neural networks. ACL 2018.
- **Глубокие нейронные сети (CNN, RNN, Transformer):** Модели, такие как BERT или LSTM, обучаются на текстах новостей с учетом их предыдущей популярности (например, число репостов или лайков). Это

позволяет учитывать не только содержание, но и контекст публикации. Источник: Nguyen, D. T., Sugiyama, K., Nakov, P., & Kan, M.-Y. (2020). FANG: Leveraging social context for fake news detection using graph representation. CIKM.

- **Графовые модели распространения (Graph Propagation Models):** Контент рассматривается как узел в сети, а пользователи и связи между ними — как ребра. Применяются модели, такие как Independent Cascade или Linear Threshold для симуляции распространения. Источник: Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. KDD.
- **Векторизация контента с обучением на исторических данных (TF-IDF, Doc2Vec, BERT embeddings):** Содержимое новости преобразуется в вектор признаков, и на его основе оценивается схожесть с уже известными вирусными публикациями. Источник: Bandari, R., Asur, S., & Huberman, B. A. (2012). The Pulse of News in Social Media: Forecasting Popularity. ICWSM.
- **Гибридные модели (мультимодальные):** Используются одновременно текст, метаданные и поведенческие данные (время просмотра, вовлечённость). Такие модели комбинируют разные типы входных признаков и обычно реализуются на основе ансамблей или нейросетей с несколькими входами. Источник: Tatar, A., Antoniadis, P., De Amorim, M. D., & Fdida, S. (2014). A Survey on Predicting the Popularity of Web Content. Computer Communications, 36(11-12), 1132-1144.

Таким образом, вычисление вирусности представляет собой комплексную задачу, в которой используются как традиционные алгоритмы машинного обучения, так и современные методы анализа графов и нейросетевые архитектуры. Выбор подхода зависит от доступных данных и требуемой точности модели.

## 1.2 Анализ существующих подходов к построению отказоустойчи-

## **вых систем**

Отказоустойчивость — способность системы продолжать функционировать при возникновении сбоев. Современные подходы к построению отказоустойчивых систем включают:

- **Микросервисная архитектура:** разделение системы на независимые сервисы, что позволяет локализовать сбои и облегчает масштабирование.
- **Использование оркестраторов:** инструменты, такие как Docker Swarm, управляют развертыванием, масштабированием и восстановлением сервисов.
- **Паттерны отказоустойчивости:** применение шаблонов проектирования, таких как Circuit Breaker и Retry, для обработки сбоев.

### **1.3 Анализ существующих решений сбора телеметрии системы и хранения логов**

Эффективный сбор телеметрии и логов необходим для мониторинга и диагностики систем. Современные решения включают:

- **Системы сбора метрик:** инструменты, такие как Prometheus, собирают и хранят метрики производительности.
- **Системы агрегации логов:** инструменты, такие как Filebeat, собирают, обрабатывают и передают логи в хранилища.
- **Платформы визуализации:** инструменты, такие как Kibana, предоставляют визуальное представление метрик и логов для анализа.

Интеграция этих инструментов обеспечивает полную наблюдаемость системы, позволяя оперативно выявлять и устранять проблемы.

### **1.4 Выводы по главе 1**

В ходе анализа существующих решений в области мониторинга новостного контента и построения отказоустойчивых систем были сделаны следующие выводы:

1. Методы фильтрации контента демонстрируют значительное разнообра-

зие как по глубине анализа, так и по вычислительным затратам. Простейшие подходы, такие как хеширование и фильтрация по ключевым словам, обеспечивают высокую производительность, но ограничены в выявлении смысловых дубликатов или релевантных публикаций в изменённой формулировке. Более продвинутые методы — семантический анализ и гибридные модели — требуют больших вычислительных ресурсов, но дают более точные результаты и способны учитывать контекст.

2. Определение эмоционального окраса и оценка вирусности основаны на широком спектре методов: от словарных и регрессионных до графовых и нейросетевых. Эффективное использование этих методов позволяет не только фильтровать контент по тону, но и прогнозировать его дальнейшее распространение в сети, что критично для новостного мониторинга в высоконагруженных информационных средах.
3. Современные подходы к построению отказоустойчивых систем, такие как микросервисная архитектура в сочетании с Docker Swarm, обеспечивают гибкость масштабирования и изоляцию сбоев. Использование паттернов обработки ошибок, включая политику повторных попыток (retry), позволяет значительно повысить надёжность работы сервисов.
4. Системы наблюдаемости и логирования, основанные на связке Filebeat, Elasticsearch и Kibana, являются промышленным стандартом для обеспечения прозрачности работы компонентов системы. Они позволяют своевременно выявлять отклонения в поведении сервисов и проводить диагностику на основе собранных логов и метрик.

Таким образом, анализ показал, что эффективная система мониторинга новостного контента должна сочетать в себе современные методы фильтрации, механизмы предсказания распространения информации, архитектурные решения для отказоустойчивости и развитую систему телеметрии. Эти аспекты легли в основу проектирования и реализации предлагаемого решения,

рассмотренного в следующих главах работы.

## **2 Постановка задачи**

### **2.1 Общая характеристика решаемой задачи**

Современные информационные системы ежедневно генерируют огромные объёмы новостных данных. Одной из ключевых задач является не только оперативный сбор такой информации, но и её эффективная фильтрация с целью предоставления конечным пользователям только актуального, достоверного и релевантного контента.

Решение данной задачи требует реализации высокоэффективных алгоритмов фильтрации и анализа тональности сообщений. Дополнительно возникает необходимость в обеспечении отказоустойчивости всей системы для её непрерывного функционирования в условиях возможных сбоев отдельных компонентов.

### **2.2 Цель работы**

Целью настоящей работы является разработка фильтрационного слоя системы мониторинга новостных ресурсов, а также внедрение архитектурных решений, обеспечивающих отказоустойчивую работу компонентов системы.

### **2.3 Задачи исследования**

В рамках работы решаются следующие задачи:

1. Разработать фильтрационный слой, обеспечивающий:
  - удаление дублирующихся сообщений (дедупликация);
  - фильтрацию контента по ключевым словам;
  - определение тональности публикаций (положительная, отрицательная, нейтральная);
  - определение потенциальной вирусности публикации.
2. Развернуть инфраструктуру системы мониторинга новостей, включающую:
  - MongoDB в режиме репликации для повышения доступности данных;
  - RabbitMQ в кластерной конфигурации для обеспечения надёжной



передачи сообщений между сервисами;

- Elasticsearch для обеспечения поиска и аналитики новостных данных;
- Сервисов парсинга, обработки данных, API и Telegram-бот.

3. Организовать развёртывание сервисов в контейнерной среде Docker для упрощения масштабирования и управления жизненным циклом компонентов.
4. Реализовать механизмы мониторинга и логирования компонентов системы с целью обеспечения их отказоустойчивости и быстрого обнаружения сбоев.
5. Провести тестирование системы на предмет надёжности и стабильности функционирования при различных нагрузках.

#### **2.4 Ожидаемые результаты**

Ожидается, что результатом работы станет развернутая система мониторинга новостных ресурсов с реализованным фильтрационным слоем и архитектурой, обеспечивающей отказоустойчивость её основных компонентов.

Система должна демонстрировать высокую устойчивость к сбоям, обеспечивать фильтрацию данных в реальном времени и предоставлять пользователям только актуальную и релевантную информацию.

### 3 Реализация фильтрационного слоя новостей

#### 3.1 Обоснование выбора технологий

Выбор технологий для реализации системы мониторинга новостных ресурсов и, в частности, фильтрационного слоя, базируется на результатах анализа, представленного в главе 1. В рамках анализа были выделены ключевые требования к системе: поддержка масштабируемой микросервисной архитектуры, обеспечение отказоустойчивости, высокая наблюдаемость, а также применение современных методов фильтрации контента, включая дедупликацию, анализ тональности и оценку потенциальной вирусности.

- **Микросервисный подход** в совместной работе был выбран как базовая архитектурная парадигма ввиду его соответствия принципам отказоустойчивости и масштабируемости (см. п. 1.2). Он позволяет изолировать критически важные компоненты, упростить развертывание и ускорить развитие системы.
- В качестве **инструмента оркестрации контейнеров** выбран **Docker Swarm**. Это решение обеспечивает автоматизированное развертывание и поддержку отказоустойчивых сценариев при сбоях отдельных сервисов.
- Для **фильтрации контента** реализован ряд механизмов, описанных в разделе 1.1: дедупликация на основе семантического сходства (см. п. 1.1.1), фильтрация по ключевым словам (см. п. 1.1.2), анализ тональности с использованием предобученной модели для русского языка (см. п. 1.1.3), а также вычисление вирусности с применением *batched inference* через REST API (см. п. 1.1.4).
- Для **сбора логов и телеметрии** выбрано решение на базе **Filebeat + Elasticsearch + Kibana** (см. п. 1.3), обеспечивающее полную наблюдаемость за всеми компонентами системы, включая анализ ошибок, метрик загрузки и отслеживание состояния очередей.
- Во всех синхронных взаимодействиях между микросервисами реализо-



Telegram) и **RSS-лент**. Каждый демон осуществляет непрерывный мониторинг доверенных источников, извлекает новые публикации и помещает их в очередь сообщений *filtration*. Эти демоны разрабатывались параллельно с данной работой и в её рамках не рассматривались детально, однако их функциональность важна для полноценного функционирования всей системы.

Следующим этапом обработки является **фильтрационный слой**, который принимает на вход собранные новости и выполняет интеллектуальную фильтрацию, включая анализ содержания и пользовательские предпочтения. Результаты работы фильтрационного слоя сохраняются в базе данных и формируют коллекцию новостей, готовых к отправке пользователям.

Последняя стадия обработки — **доставка новостей конечным пользователям**, которая реализуется с помощью **Telegram-бота**. Бот периодически запрашивает через Web Api подготовленные сообщения из коллекции NewsToSend и отправляет их в соответствующие каналы или личные чаты в Telegram. Отправка производится с учётом индивидуальных настроек пользователей, включая предпочтения по тематикам, ключевым словам и режиму уведомлений.

Общая архитектура системы представлена на рисунке (см. рис. 3.2.1).

### 3.3 Архитектура фильтрационного слоя

Фильтрационный слой системы мониторинга новостей реализован как связка специализированных микросервисов, объединённых единой очередью сообщений и REST-интерфейсами для вызова вспомогательных компонентов. Основной задачей фильтрационного слоя является интеллектуальная обработка новостных сообщений: от базовой предобработки до персонализированной фильтрации под конкретных пользователей.



- **Микросервис Classifier** получает батчи новостей и возвращает значение вероятности широкой огласки для каждой. Вызов осуществляется синхронно через REST API с политикой повторных попыток в случае сетевых или внутренних ошибок.
- **Финальная обработка** включает:
  - сохранение новостей в MongoDB и индексацию в ElasticSearch;
  - персонализированную фильтрацию под конкретных пользователей (учёт ключевых слов, дедупликации, пользовательских настроек);
  - формирование объектов в коллекции NewsToSend, с метаданной для Telegram-бота (включая флаг replyMessageId для формирования цепочек).

Взаимодействие между компонентами осуществляется асинхронно (через RabbitMQ) либо по REST (между Filtrator и вспомогательными сервисами). Архитектура слоя представлена на рисунке (см. рис. 3.3.1).

### 3.4 Реализация сервиса Filtrator

Микросервис **Filtrator** реализует основную логику фильтрационного слоя и отвечает за обработку новостных сообщений, поступающих из очереди filtration. Внутри сервиса реализовано два основных этапа обработки: **предобработка (pre-process)** и **постобработка с участием модели (ml-process)**.

#### Предобработка

На первом этапе сервис подписывается на очередь filtration и обрабатывает каждое входящее сообщение в отдельном воркере. Обработка включает следующие действия:

1. **Проверка времени публикации:** если формат времени некорректен, сообщение отбрасывается.
2. **Проверка на дублирование по идентификатору:** если новость с тем же ID уже существует в MongoDB, она также отбрасывается.
3. **Векторизация текста:** новость передаётся в микросервис

`text_processor`, где рассчитывается векторное представление текста. Этот вектор добавляется в объект новости.

4. **Определение эмоций:** осуществляется попытка получить эмоциональную окраску текста через тот же `text_processor`. В случае ошибки значение по умолчанию — `unknown`.
5. **Определение дубликатов по смыслу:** новость сравнивается с ранее обработанными сообщениями с помощью компонента `NewsFilterService`. Дедупликация производится с помощью векторного сравнения (`knn`-поиск) на основе предварительно рассчитанного вектора (см. п. 1.3). Если новость признана дубликатом, в неё добавляется ссылка на оригинал и соответствующая метка. Также планируется дальнейшее улучшение дедупликации с помощью машинного обучения, т.е. если новости векторно похожи, дополнительно сравниваем их с помощью предобученной модели.

После завершения предобработки сообщение отправляется во вторую очередь `filterMlQueue` для последующего этапа.

### Батчинг и классификация

Сервис параллельно подписан на очередь `filterMlQueue`. Все новости, поступившие на этом этапе, накапливаются в **батч конфигурируемого размера** (по умолчанию 10 штук) или обрабатываются по **таймауту**, если новые сообщения не поступают. Когда батч готов, он передаётся в микросервис **Classifier**, где для каждой новости рассчитывается **вероятность огласки** (показатель `BangerProbability`). Классификация осуществляется через REST API с политикой повторных попыток в случае ошибок.

### Сохранение и фильтрация по пользователям

После получения результата классификации каждая новость:

- сохраняется в коллекции `News` `MongoDB`;
- индексируется в `ElasticSearch`;
- проверяется по настройкам пользователей, подписанных на соответ-

ствующий источник.

Фильтрация выполняется с учётом:

- включена ли опция дедупликации;
- наличие совпадений по ключевым словам;
- другие параметры профиля пользователя.

Если новость удовлетворяет критериям, она сохраняется в коллекции `NewsToSend`. Для дубликатов предусмотрен механизм “ответа на оригинал”: если оригинальная новость уже была отправлена пользователю, новая помечается как обновление с `replyMessageId`, что позволяет боту сформировать цепочку сообщений. Таким образом, сервис **Filtrator** выполняет полную обработку новостных сообщений от входной очереди до формирования пользовательских задач на отправку, интегрируя в себе дедупликацию, машинное обучение и персонализированную фильтрацию.

### 3.5 Микросервис **TextProcessor**

Микросервис **TextProcessor** отвечает за векторизацию текста новостных сообщений и определение их эмоциональной окраски. Он используется на этапе предобработки в рамках фильтрационного слоя и вызывается асинхронно из микросервиса **Filtrator**.

#### Архитектура и технологии

Сервис реализован с использованием языка **Python** и фреймворка **FastAPI**. Для векторизации применяется предобученная модель `paraphrase-multilingual-MiniLM-L12-v2` из библиотеки `sentence-transformers`, обеспечивающая получение универсального векторного представления текста на разных языках, включая русский.

Для анализа тональности используется модель `blanchefort/rubert-base-cased-sentiment`, предназначенная для работы с русскоязычными текстами. Она классифицирует эмоциональную окраску как **положительную**, **отрицательную** или **нейтральную**, что позволяет проводить базовую эмоциональную фильтрацию новостей.



Сервис предоставляет два HTTP-эндпоинта:

- `POST /vectorize` — возвращает векторное представление текста;
- `POST /emotion` — возвращает эмоциональную окраску текста.

### **Взаимодействие с другими компонентами**

Сервис вызывается асинхронно из микросервиса **Filtrator** при получении новой новости:

- сначала происходит векторизация текста;
- затем определяется его тональность;
- полученные данные записываются в объект новости и используются на следующих этапах обработки (дедупликация, фильтрация, классификация).

В ходе нагрузочного тестирования (см. п. 4.10) было выявлено, что одиночная обработка запросов на векторизацию и определение тональности является узким местом. В дальнейшем планируется переход на батчевую обработку.

Таким образом, **TextProcessor** выполняет ключевую роль в подготовке новостей к дальнейшему анализу, обеспечивая семантическое представление и базовую эмоциональную характеристику сообщений.

### **3.6 Микросервис Classifier**

Микросервис **Classifier** отвечает за определение вероятности широкого распространения новостного сообщения — метрики, обозначенной как **вероятность огласки** (`BangerProbability`). Он используется после этапа предобработки и векторизации новостей, когда они уже прошли первичный фильтр и накапливаются в батчи в микросервисе **Filtrator**.

#### **Архитектура и назначение**

Сервис реализован как API-приложение на основе **FastAPI**. Основная задача — принимать батчи новостей и возвращать список вероятностей, отражающих степень “вирусности” или потенциальной значимости каждой публикации. Обработка осуществляется асинхронно по HTTP-протоколу.

## Модели и обработка

Разработка и обучение моделей машинного обучения в рамках данного микросервиса **не входила в зону ответственности автора**. В данной работе были реализованы:

1. Сбор данных для обучения модели на основе ранее обработанных новостей
2. Инфраструктура API, обеспечивающая:
  - приём входных данных (тексты новостей или их векторные представления);
  - вызов предобученной модели и получение прогнозов;
  - возврат предсказаний в виде вероятностей от 0 до 1 (интерпретируемых как процент вероятности огласки).

Определение вероятности огласки выполняется на основе следующих данных:

- Текст публикации
- Уровень влияния источника, рассчитывается на основе количества подписчиков и территориального уровня (окружной, региональный, федеральный)

Основной рабочий эндпоинт:

- `POST /predict_batch` — принимает список новостей и возвращает массив значений `probs`, соответствующих вероятностям распространения для каждой новости.

Взаимодействие с **Filtrator**-ом происходит асинхронно: батчи формируются на стороне фильтрационного слоя и отправляются в **Classifier** по мере достижения заданного размера или по таймауту. Встроенные механизмы

Сервис включает:

- базовую валидацию входных данных;
- регистрацию ошибок с понятными HTTP-ответами;
- фоновые задачи (например, отложенное дообучение через

BatchAccumulator, если потребуется доработка модели в будущем).

Таким образом, **Classifier** обеспечивает масштабируемую точку входа для применения моделей оценки новостей и интегрируется с фильтрационным слоем в рамках общей архитектуры микросервисной системы.

## 4 Развёртывание и обеспечение отказоустойчивости

### 4.1 Общая схема развёртывания

Система мониторинга новостных ресурсов развёрнута в виде микросервисной архитектуры с применением Docker Swarm. Все компоненты упакованы в отдельные контейнеры, что обеспечивает изоляцию, гибкость конфигурации и масштабируемость. Основные микросервисы включают:

- `filtrator` — основной фильтрационный слой;
- `text_processor` — сервис векторизации текста и определения эмоции;
- `classifier` — сервис определения вероятности широкого распространения новости;
- `vk_daemon`, `rss_daemon`, `telegram_parser`, `telegram_subscriber`, `master_daemon` — демоны, получающие данные из внешних источников;
- `webapi` — API-интерфейс;
- `tg_bot` — Telegram-бот, отвечающий за отправку новостей пользователю.

Также в состав системы входят:

- MongoDB;
- RabbitMQ;
- ElasticSearch + Kibana;
- Filebeat для сбора логов.

Развёртывание выполнено на домашнем сервере с Ubuntu, подключённом к облачному VPS по WireGuard, что обеспечивает внешний доступ к внутренней инфраструктуре без необходимости статического IP-адреса.

### 4.2 Настройка Docker Swarm

Контейнеры управляются с помощью Docker Swarm. Это позволило:

- обеспечить оркестрацию микросервисов;
- централизованно управлять конфигурацией и переменными окружения;
- использовать общие `docker-compose` файлы с возможностью пере-

определения хостов, портов и путей.

Для повышения устойчивости контейнеров используется политика `restart: always`. Однако в перспективе планируется переход на более надёжный механизм — `healthcheck`-мониторинг, который позволит выявлять не только падения контейнеров, но и внутренние ошибки приложения, влияющие на его корректную работу.

### 4.3 Настройка MongoDB

MongoDB изначально разворачивалась в режиме репликации (`ReplicaSet`), с полным набором скриптов и `keyfile`-аутентификацией. Однако в силу ограничений по ресурсам репликация была временно отключена.

При этом:

- подготовлена инфраструктура для быстрого возврата к отказоустойчивому режиму;
- все коллекции (пользователи, новости, подписки и т.д.) создаются автоматически;
- добавлены необходимые индексы (например, по `timestamp`, `vector`, `user_id`, `status`), повышающие производительность запросов и устойчивость системы к нагрузкам.

### 4.4 Настройка RabbitMQ

RabbitMQ развёрнут в виде одиночного экземпляра с сохранением состояния через `volume`. Все очереди создаются автоматически с помощью специального `bash`-скрипта, который:

- создаёт пользователя и виртуальный хост;
- настраивает права доступа;
- инициализирует очереди (`news_queue_vk`, `news_queue_rss`, `filter_queue`).

В будущем возможен переход к кластерной конфигурации RabbitMQ для обеспечения отказоустойчивости и балансировки нагрузки.

### 4.5 Настройка Elasticsearch, Kibana и Filebeat

ElasticSearch используется для хранения и поиска новостей и логов. Настроена работа с TLS-сертификатами, сгенерированными вручную. Kibana предоставляет визуальный интерфейс для работы с данными.

Filebeat конфигурирован на сбор логов из всех контейнеров Docker, включая:

- структурирование логов (декодирование JSON, парсинг полей);
- фильтрацию лишних сообщений (например, от MongoDB);
- автоматическую разбивку логов по индексам: `logs-service_name-prod-date`.

Всё это обеспечивает удобную отладку и анализ системы в реальном времени.

Потенциал развития:

- подключение Alerting-механизмов в Kibana;
- внедрение Elastic APM для трассировки вызовов между сервисами.

#### 4.6 Обеспечение отказоустойчивости

Были предприняты следующие шаги:

- Внедрены **ретрай**-политики (через Polly и собственные механизмы) при обращении к внешним сервисам: MongoDB, RabbitMQ, TextProcessor, Classifier.
- Используется batch-обработка сообщений, позволяющая сглаживать нагрузку и минимизировать потери при сбоях.
- Весь обмен сообщениями между сервисами организован через **устойчивые очереди** RabbitMQ (`durable: true`).
- Добавлено **обработка ошибок и логирование** в ключевых точках.
- Используется **асинхронная модель** выполнения в Python-сервисах (RSS, Telegram, TextProcessor), что обеспечивает высокую отзывчивость при одновременной работе с десятками источников.

В будущем планируется интеграция healthcheck-механизмов, позволяющих более точно управлять перезапуском контейнеров и обнаружением

«тихий сброс».

#### 4.7 Логирование и мониторинг

Система логирования реализована на базе Filebeat + Elastic + Kibana. Все логи собираются в формате JSON, что обеспечивает возможность автоматической агрегации и анализа.

На данный момент отсутствует метрик-ориентированный мониторинг (Prometheus, Grafana), но архитектура системы предусматривает лёгкое подключение данных инструментов в будущем.

#### 4.8 Гибкая конфигурация сервисов

Особенностью реализации является удобная система конфигурации:

- Все переменные окружения вынесены в `.env` и `hosts.env`, что позволяет быстро переключаться между окружениями (dev/prod).
- Возможен **гибридный режим** разработки — часть сервисов запускается в Docker, часть — напрямую из IDE. Это существенно ускоряет отладку.
- Все пути к основным сервисам (Mongo, Elastic, RabbitMQ, WebAPI) задаются параметрами, что упрощает развертывание в различных инфраструктурах.

#### 4.9 Безопасность

Для обеспечения безопасности предприняты следующие шаги:

- Все соединения с Elasticsearch и Kibana защищены с помощью SSL-сертификатов.
- Используется WireGuard для подключения внутреннего домашнего сервера к облачному VPS, обеспечивая защищённый канал доступа к сервисам.
- Для RabbitMQ и MongoDB настроены отдельные пользователи и роли с минимальными правами.

В будущем планируется реализация:

- ограничений по IP;
- централизованного управления правами доступа.

#### **4.10 Тестирование устойчивости**

В рамках разработки были проведены следующие виды тестирования:

- перезапуск отдельных контейнеров и всей системы в целом;
- отключение ключевых компонентов (TextProcessor, Classifier) с последующим восстановлением;
- стресс-тестирование системы с симуляцией массового поступления новостей.

Результаты подтвердили способность к восстановлению без потерь. Также было выявлено узкое место при определении тональности новостей. Из-за того, что обработка происходит поочередно, без применения батчевой обработки, при больших нагрузках (более 50 сообщений в секунды) новости обрабатываются медленнее, чем поступают в очередь на предобработку. Однако стоит отметить, что потери данных не происходит, и после снижения нагрузки состояние системы приходит в нормальное состояние.



## ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была разработана и частично внедрена система мониторинга новостных ресурсов, ориентированная на высокую надёжность, масштабируемость и качество фильтрации контента. Основное внимание уделялось созданию **фильтрационного слоя**, обеспечивающего интеллектуальную предобработку новостных сообщений, и построению **инфраструктуры**, устойчивой к сбоям.

В ходе реализации достигнуты следующие результаты:

- Разработан фильтрационный микросервис **Filtrator**, включающий:
  - предобработку новостей и асинхронную векторизацию текста;
  - анализ эмоций и определение вероятности "вирусности" публикации;
  - дедупликацию на основе векторных представлений текста;
  - фильтрацию по пользовательским настройкам и распределение новостей.
- Разработаны и внедрены два вспомогательных микросервиса:
  - **TextProcessor** — для векторизации и анализа тональности;
  - **Classifier** — для оценки вероятности широкого распространения новостей.
- Вся система развёрнута с использованием **Docker Swarm**, с удобной, параметризуемой конфигурацией, поддержкой локальной отладки и потенциальной миграцией в облако.
- Реализована начальная поддержка отказоустойчивости:
  - retry-политики в критичных местах;
  - **батч**-обработка данных;
  - `restart: always` в Docker;
  - устойчивые очереди и хранение промежуточных результатов в MongoDB.
- Настроено централизованное логирование через Filebeat и ElasticSearch

- Обеспечена базовая безопасность с помощью TLS и WireGuard.

В ходе работы было выявлено несколько направлений для развития:

- внедрение полноценного мониторинга состояния микросервисов (Prometheus, Grafana, healthcheck);
- автоматизация CI/CD и деплоя в облачные инфраструктуры;
- масштабирование брокера RabbitMQ и базы MongoDB с включением репликации;
- улучшение качества классификации тональности и расширение фильтрационного слоя (включение Named Entity Recognition, тематической категоризации и т.д.).

Разработанная архитектура и фильтрационный слой подтвердили свою эффективность в тестовых условиях и могут быть использованы в системах анализа репутации, информационных панелях и корпоративных решениях для обработки новостей и сообщений из открытых источников. Таким образом, поставленные цели и задачи выпускной работы были достигнуты, а созданная система обладает высоким потенциалом для дальнейшего развития и промышленного применения.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Docker Compose Documentation [Электронный ресурс] // Docker Compose. URL: <https://docs.docker.com/compose/>. (дата обращения: 13.05.2025).
2. Docker Documentation [Электронный ресурс] // Get Started. URL: <https://docs.docker.com/get-started/>. (дата обращения: 13.05.2025).
3. Docker Swarm Documentation [Электронный ресурс] // Docker Swarm. URL: <https://docs.docker.com/reference/cli/docker/swarm/>. (дата обращения: 13.05.2025).
4. Elastic Documentation [Электронный ресурс] // Filebeat. URL: <https://www.elastic.co/docs/reference/beats/filebeat>. (дата обращения: 13.05.2025).
5. Elastic Documentation [Электронный ресурс] // Kibana Guide. URL: <https://www.elastic.co/guide/en/kibana/8.18/index.html>. (дата обращения: 13.05.2025).
6. Elasticsearch Documentation [Электронный ресурс] // Get started. URL: <https://www.elastic.co/docs/get-started>. (дата обращения: 13.05.2025).
7. FastAPI Documentation [Электронный ресурс] // Learn. URL: <https://fastapi.tiangolo.com/learn/>. (дата обращения: 13.05.2025).
8. Microsoft .NET Documentation [Электронный ресурс] // .NET documentation. URL: <https://learn.microsoft.com/en-us/dotnet/>. (дата обращения: 13.05.2025).
9. Microsoft ASP.NET Documentation [Электронный ресурс] // ASP.NET documentation. URL: <https://learn.microsoft.com/en-us/aspnet/core/>. (дата обращения: 13.05.2025).
10. Microsoft C# Documentation [Электронный ресурс] // C# Docs. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. (дата обращения: 13.05.2025).
11. MongoDB Documentation [Электронный ресурс] // Frameworks and Integrations. URL: <https://www.mongodb.com/docs/languages/csharp/>. (дата

обращения: 13.05.2025).

12. Python 3.13.3 Documentation [Электронный ресурс] // Documentation. URL: <https://docs.python.org/3/>. (дата обращения: 13.05.2025).
13. RabbitMQ Official Documentation [Электронный ресурс] // Tutorials. URL: <https://www.rabbitmq.com/docs>. (дата обращения: 13.05.2025).
14. Yandex Cloud Rest API [Электронный ресурс] // REST API: для чего нужен и как работает. URL: <https://yandex.cloud/ru/docs/glossary/rest-api>. (дата обращения: 13.05.2025).