Product - News Aggregator

---

1. Концептуальное описание модели

Агрегатор новостей — это система, которая собирает, обрабатывает, хранит и предоставляет доступ к новостным статьям из различных источников. Она объединяет несколько баз данных для выполнения различных функций: от хранения структурированных данных до аналитики и рекомендаций. Основные компоненты системы:

Источники данных: Внешние API, RSS-ленты, а также ручной ввод новостей редакторами.

Парсер: Модуль, который собирает новости, извлекает данные (заголовок, текст, источник, дата, категория, теги) и распределяет их по соответствующим базам.

Хранилище данных: Используются семь баз данных:

- **PostgreSQL**: Хранит структурированные данные (пользователи, статьи, категории, теги).
- **MongoDB**: Хранит полуструктурированные данные (сырые тексты статей, метаданные).
- **Redis**: Кэширует часто запрашиваемые данные (например, популярные статьи).
- **Neo4j**: Хранит граф связей между статьями, авторами, тегами и пользователями для рекомендаций.
- **ClickHouse**: Хранит аналитические данные (статистика просмотров, тренды).
- **InfluxDB**: Хранит временные ряды (активность пользователей, метрики запросов).
- **Firebase**: Управляет push-уведомлениями и данными реального времени.

API-сервер: Обрабатывает запросы пользователей, взаимодействуя со всеми базами.

Frontend: Веб-интерфейс, бот или CLI для взаимодействия пользователей с системой.

Рекомендательная система: Использует данные Neo4j для персонализации контента.

Админ-панель: Позволяет редакторам и администраторам управлять контентом и анализировать статистику.

Общий процесс работы:

1. Парсер собирает новости из источников, сохраняет их в MongoDB (сырые данные) и PostgreSQL (структурированные данные).
2. Пользователь запрашивает новости через API. Redis проверяет наличие данных в кэше, если их нет — запрос идет к PostgreSQL.
3. Neo4j формирует рекомендации на основе связей между статьями и пользовательскими предпочтениями.
4. ClickHouse и InfluxDB предоставляют аналитику для админ-панели.
5. Firebase отправляет push-уведомления о новых статьях или обновлениях.

---

2. Логическое описание модели

— Базы данных и их использование —

PostgreSQL (реляционная база данных):

Хранит: Структурированные данные:

- Пользователи (Users): ID, имя, email, хэшированный пароль, роль (admin, editor, reader).
- Статьи (Articles): ID, заголовок, дата публикации, источник, ID категории.
- Категории (Categories): ID, название (например, "Политика", "Технологии").
- Теги (Tags): ID, название (например, "экономика", «спорт», т.д.).
- Связь статьи-теги (Article_Tags): таблица для связи многие-ко-многим.

Использование: Основное хранилище для структурированных данных, поддерживает транзакции и целостность данных. Используется для поиска статей, фильтрации по категориям и тегам, а также управления пользователями.

MongoDB (NoSQL, документоориентированная база):

Хранит: Полуструктурированные данные:

- Полные тексты статей и их метаданные (JSON-документы с полями: заголовок, текст, источник, дополнительные атрибуты, такие как изображения или ссылки).

Использование: Хранит сырые данные, полученные от парсера, для быстрого доступа к полным текстам и неструктурированным метаданным. Поддерживает гибкие схемы, что удобно для данных с вариативной структурой.

Redis (ключ-значение, в памяти):

Хранит: Кэшированные данные:

- Списки популярных статей (например, top_articles:day как список ID статей).
- Результаты частых запросов (например, search:технологии).

Использование: Ускоряет доступ к часто запрашиваемым данным. Кэширует результаты запросов к PostgreSQL или MongoDB, чтобы минимизировать нагрузку на основные базы.

Neo4j (графовая база данных):

Хранит: Граф связей:

- Узлы: статьи, пользователи, теги, категории.
- Ребра: связи типа RELATED_TO (между статьями), LIKES (пользователь-статья), BELONGS_TO (статья-категория).

Использование: Формирует рекомендации (например, "похожие статьи" или "статьи, понравившиеся похожим пользователям"). Поддерживает сложные запросы на основе связей.

ClickHouse (аналитическая база данных):

Хранит: Аналитические данные:

- Таблица Article_Views: article_id, views_count, date.
- Таблица Category_Stats: category_id, views_count, date.

Использование: Хранит и обрабатывает большие объемы данных для аналитики (например, статистика просмотров статей, популярные категории). Оптимизирована для агрегации и быстрого анализа.

InfluxDB (база данных временных рядов):

Хранит: Временные ряды:

- Метрики активности пользователей: user_id, timestamp, action (например, "просмотр", "поиск").
- Метрики запросов: частота запросов по категориям или тегам.

Использование: Анализирует временные тренды, например, пики активности пользователей или популярность категорий.

Firebase (платформа реального времени):

Хранит: Данные для уведомлений:

- Сообщения для push-уведомлений: user_id, message, timestamp.

Использование: Отправляет уведомления о новых статьях, обновлениях или персонализированных рекомендациях.

— Логика взаимодействия —

Сбор данных:
- Парсер собирает новости из источников (RSS, API), сохраняет сырые данные в MongoDB и структурированные данные в PostgreSQL.
- Категории и теги синхронизируются между PostgreSQL и Neo4j для поддержки рекомендаций.

Обработка запросов:
- Пользовательский запрос (например, поиск по категории) сначала проверяется в Redis.
- Если данные отсутствуют, запрос идет к PostgreSQL для получения списка статей и к MongoDB для полных текстов.
- Neo4j используется для рекомендаций (например, похожие статьи).

Аналитика:
- ClickHouse агрегирует данные о просмотрах для отчетов (например, топ-10 категорий).
- InfluxDB отслеживает временные метрики (например, активность пользователей).

Уведомления:
- Firebase отправляет push-уведомления при появлении новых статей в выбранных пользователем категориях.

Администрирование:
- Админ-панель запрашивает аналитику из ClickHouse и InfluxDB, а также позволяет редактировать данные в PostgreSQL и MongoDB.

Пример сценария:

- Пользователь ищет новости в категории "Технологии".
- API-сервер проверяет Redis на наличие кэша (search:технологии).
- Если кэша нет, PostgreSQL возвращает список статей, MongoDB предоставляет полные тексты.
- Neo4j предлагает похожие статьи на основе связей.
- ClickHouse обновляет счетчик просмотров, InfluxDB записывает действие пользователя.
- Firebase отправляет уведомление, если пользователь подписан на категорию.

3. Схематическое описание модели.

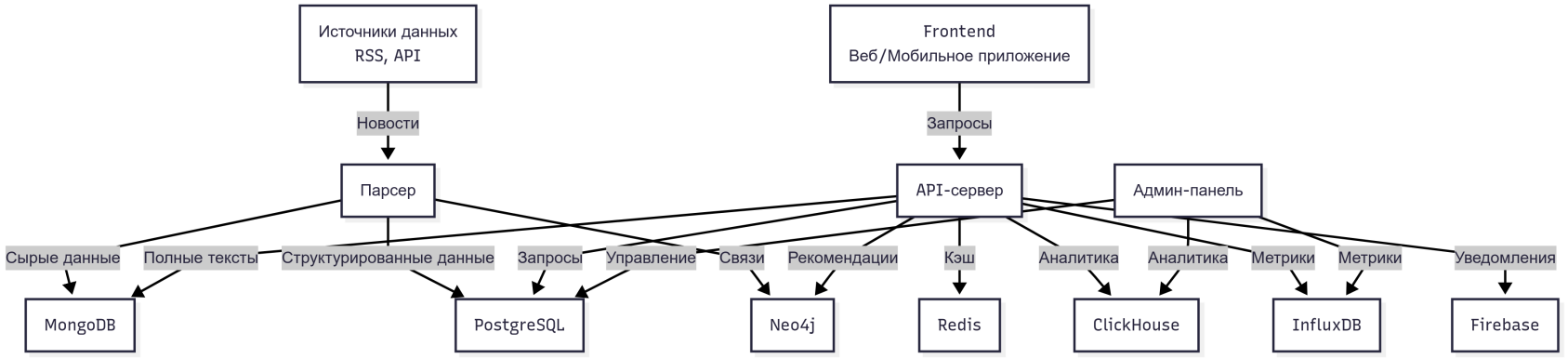

Схема 1: Общая структура агрегатора новостей

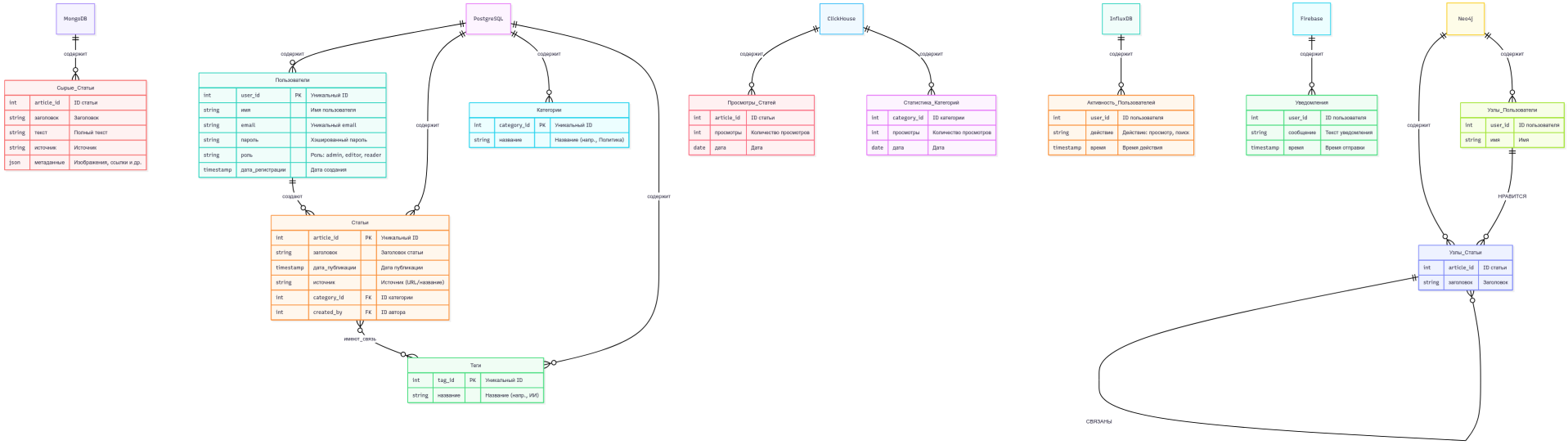

Схема 2: Базы данных и их содержимое

Схема 1 показывает поток данных от источников через парсер к базам данных, а затем через API к пользователю. Каждая база выполняет свою роль: MongoDB и PostgreSQL — для хранения, Redis — для кэширования, Neo4j — для рекомендаций, ClickHouse и InfluxDB — для аналитики, Firebase — для уведомлений.

Схема 2 детально показывает, какие данные хранятся в каждой базе данных, используемой в агрегаторе новостей, и как эти данные связаны между собой.

---

4. Требования LLM в качестве Product Owner.

1. Функциональные требования

1.1. Сбор данных

- Система должна собирать новости из заранее подготовленного файла (например, JSON или CSV) с данными о новостях (заголовок, текст, источник, дата, категория, теги).
- Поддержка ручного добавления новостей через консольное приложение (ввод данных пользователем).
- Пример структуры новости:
  - Заголовок (строка).
  - Текст (строка).
  - Источник (строка, например, "BBC").
  - Дата публикации (дата/время).
  - Категория (например, "Политика", "Спорт").
  - Теги (список строк, например, ["ИИ", "технологии"]).

1.2. Хранение данных

- **PostgreSQL**: Хранит структурированные данные:
  - Пользователи (ID, имя, роль: reader, editor).
  - Статьи (ID, заголовок, дата публикации, источник, категория).
  - Категории (ID, название).
  - Теги (ID, название).
  - Связь статьи-теги (таблица многие-ко-многим).
- **MongoDB**: Хранит полные тексты новостей в виде JSON-документов (ID, заголовок, текст, источник, дополнительные данные).
- **Redis**: Кэширует список из 5 популярных статей (ID и заголовок).
- **Neo4j**: Хранит связи между статьями (например, "похожие статьи") и пользователями (например, "пользователь прочитал статью").
- **ClickHouse**: Хранит статистику просмотров статей (ID статьи, количество просмотров, дата).
- **InfluxDB**: Хранит временные метрики (например, количество просмотров статей за день).
- **Firebase**: Отправляет простое уведомление о новой статье (например, заголовок статьи).

1.3. Основные функции

- **Поиск новостей**:
  - Поиск статей по категории или тегу через PostgreSQL.
  - Вывод результатов в консоль (ID, заголовок, категория).
- **Отображение новостей**:
  - Вывод списка всех статей (из PostgreSQL) или одной статьи с полным текстом (из MongoDB).
- **Рекомендации**:
  - Простая рекомендация: показать 3 статьи, связанные с текущей (через Neo4j).
- **Аналитика**:
  - Вывод количества просмотров статей за день (из ClickHouse).
  - Вывод количества действий пользователей за день (из InfluxDB).
- **Добавление новостей**:
  - Редактор может добавить новую статью через консоль (сохраняется в PostgreSQL и MongoDB).
- **Уведомления**:
  - Отправка уведомления о новой статье через Firebase (простое текстовое сообщение).

1.4. Интерфейс

- Консольное приложение на C++.
- Простое меню с командами:
  - Добавить новость (для редактора).
  - Найти новости по категории/тегу.
  - Показать статью по ID.
  - Показать рекомендации.
  - Показать статистику просмотров.
- Вывод результатов в текстовом формате (без сложного UI).

2. Нефункциональные требования

2.1. Производительность

- Система должна обрабатывать до 100 запросов в минуту (поиск, отображение, добавление).
- Время ответа на поиск — не более 2 секунд.

2.2. Простота реализации

- Использовать библиотеки C++ для работы с базами:
  - **libpqxx** для PostgreSQL.
  - **mongocxx** для MongoDB.
  - **redis-plus-plus** для Redis.
  - **neo4j-cpp** (или REST API) для Neo4j.
  - **ClickHouse C++ Client** для ClickHouse.
  - **InfluxDB C++ Client** для InfluxDB.
  - **Firebase C++ SDK** для Firebase.

- Допускается использование JSON-файлов для имитации данных, если подключение к базе затруднительно.

2.3. Тестирование

- Достаточное количество тестовых данных:
- Тестирование через консоль: запуск команд и проверка вывода.

3. Требования к реализации

- **Язык**: C++
- **Библиотеки**:
  - Использовать официальные или популярные библиотеки для работы с базами.
  - Для JSON-парсинга использовать nlohmann/json.
- **Документация**:
  - Краткое описание кода в комментариях.
  - Описание, как запустить программу (например, какие базы нужно установить).