**БУ ВО ХАНТЫ-МАНСИЙСКОГО АВТОНОМНОГО ОКРУГА – ЮГРЫ «СУРГУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Политехнический институт

Кафедра автоматики и компьютерных систем (АиКС)

**ОТЧЕТ**

**по учебной практике, по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности/ознакомительной\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(сроки практики)

Студента \_\_\_\_ курса, группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| ФИО: |  |

Руководитель практики:

|  |  |
| --- | --- |
| Должность, ФИО: |  |

|  |  |
| --- | --- |
| Дата: |  |

Сургут 202\_\_

**Словарь.**

**MVP -** **Минимально жизнеспособный продукт - версия продукта с минимальным набором функций, достаточным для: Тестирования гипотез на реальных пользователях; Получения обратной связи на ранней стадии; Дальнейшей итеративной разработки.**

**Тред -** **Цепочка сообщений по одной теме.**

**Топик -**

**Rest API - Архитектурный стиль для веб-сервисов, где: Ресурсы (данные) доступны по URL (эндпоинтам).**

JWT токен - это стандарт (RFC 7519) для создания токенов доступа, которые содержат закодированные JSON-данные. Используется для аутентификации и передачи информации между клиентом и сервером.

Асинхронная обработка - это подход, при котором задачи выполняются без блокировки основного потока выполнения программы. Вместо ожидания завершения операции (как в синхронном коде), система продолжает работать, а результат обрабатывается позже.

**БД-база данных**

**Kafka — это распределённая потоковая платформа для обработки событий в реальном времени. Кластер Kafka состоит из нескольких брокеров, работающих вместе для обеспечения отказоустойчивости и масштабируемости.**

**Брокер (Broker) в Kafka — это сервер, который хранит данные, обрабатывает запросы на запись и чтение сообщений, а также обеспечивает отказоустойчивость и масштабируемость системы.**

**Фронтэнд-**

**Бэкенд-**

Ручка – это публичный метод сервиса, URL'ы в API какого-нибудь сервиса.(Например: "Чтобы загрузить файл на сервис, мне надо дёрнуть ручку https://123/123/123 и получить URL для закачки".)

**По центру и чёрным цветом**

Оглавление

[**1. Введение.** 4](#_Toc198295622)

[**2. Обзор систем аналогичного назначения.** 5](#_Toc198295623)

[3. Техническое задание. 7](#_Toc198295624)

[4. Используемые программные технологии и средства. 7](#_Toc198295625)

[4.1. Frontend. 7](#_Toc198295626)

[4.2. Backend. 7](#_Toc198295627)

[4.3. Базы данных. 7](#_Toc198295628)

[4.4. Инфраструктура. 7](#_Toc198295629)

[4.5. Дизайн интерфейса. 7](#_Toc198295630)

[**5.** Работа с данными. 7](#_Toc198295631)

[6. Создание тредов и комментариев. 9](#_Toc198295632)

[7. Возможность регистрации, аутентификации. 10](#_Toc198295633)

[**8. Задачи сервисов.** 10](#_Toc198295634)

[9. Заключение. 11](#_Toc198295635)

[**10. Список литературы.** 12](#_Toc198295636)

# **Введение.**

**Цели**

Разработка веб-форума с использованием современных технологий и паттернов программирования.

**Задачи**

Реализация MVP продукта с базовым функционалом:

Создание тредов и комментариев.

Возможность регистрации, аутентификации.

Реализация асинхронной обработки запросов(с помощью kafka).

Освоить полный цикл разработки(фронтенд, бэкенд, базы данных, инфраструктура).

Изучить на практике работу и проектирование микросервисной архитектуры.

**актуальность темы:**

Изучения полного цикла разработки требуется знать каждому разработчику в той или иной мере, так как ему требуется реализовывать продукт которым будут пользоваться не только программисты его направления, важно понимать для кого и каким образом будет удобнее пользоваться продуктом.

Микросервисная архитектура также является популярной решением современной разработки так как она предлагает более гибкую, масштабируемую и ускоренную разработки продукта для крупных проектов.

Веб-форум, это простой проект для разработки и в то же время сложный для создания архитектуры, так как в нём заключены простые и понятные задачи, которые нужно укомплектовать в современную обёртку. Стоит отметить, что имиджборды как являение появились в конце 1990-х годов, но у них до сих пор есть аудитория и они остаются востребованными благодаря уникальной культуре и минималистичному подходу.

# **Обзор систем аналогичного назначения.**

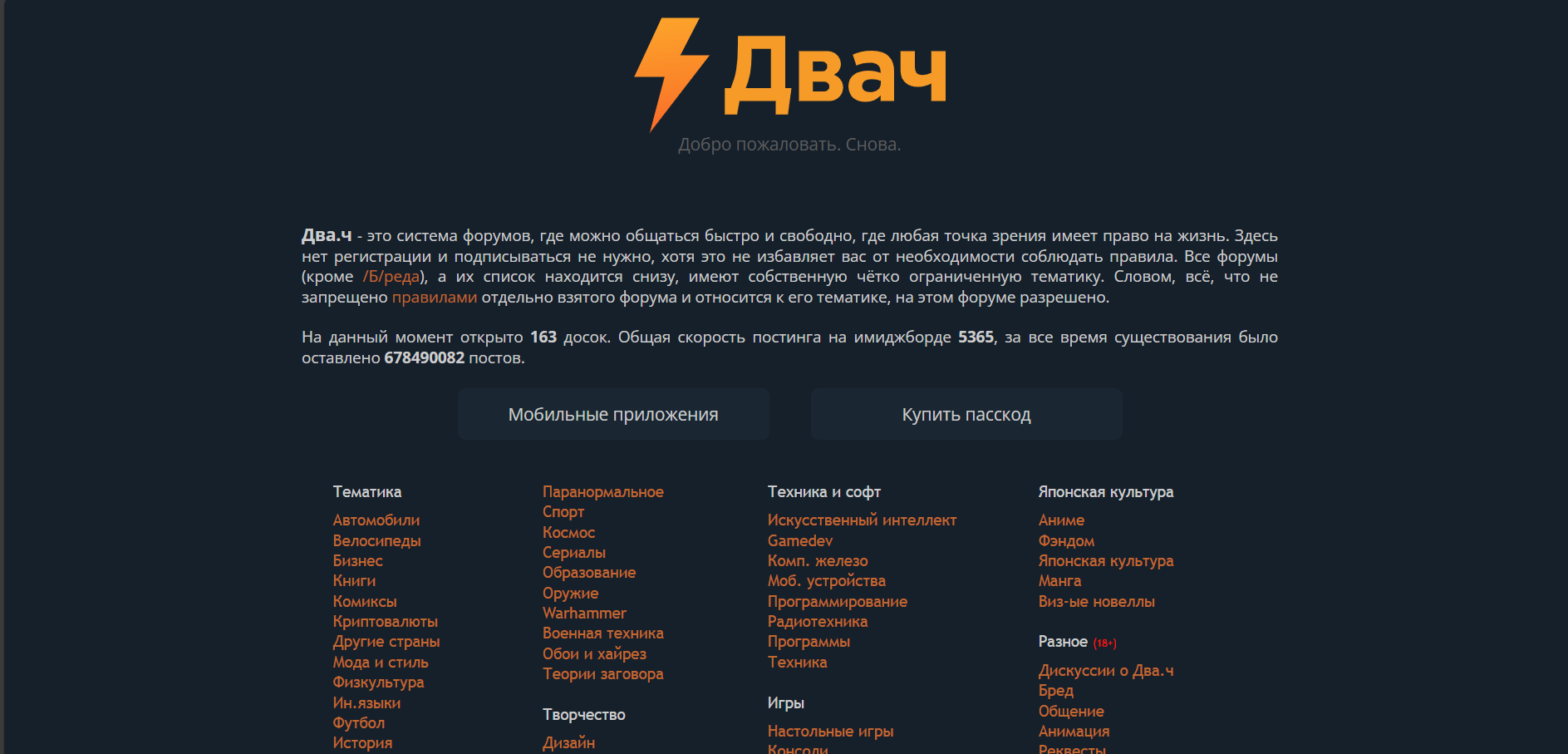


Рисунок 1 главная страница 2ch.life

Функциональность:

Создание тредов в определённой доске; Ответы на открытые темы; В ответах есть возможность прикрепления до 4 файлов или до 40мб всё сообщение или 15000 символов; Ответ отличается от обычного сообщения специальном набором символов [>>номер поста] в начале сообщения; Защита от спама в виде капчи при каждом новом сообщении. Если пользователь не хочет каждый раз вводить капчу, есть возможность купить “пасскод” который помогает пользователю не вводить CAPTHCA и увеличивает максимальный размер его сообщений; Встроенные инструменты для создания ASCII-арта и мемов. Пользователи публикуют сообщения без регистрации, идентифицируя только по временным ID или «трипкодам». Модерация осуществляется через пользовательские жалобы, а не премодерацию. Встроенная возможность форматирования текста. На главной странице есть новости сайта.

стек технологий:

Фронтенд: HTML5, CSS3, JavaScript

Бэкенд: Perl

Интерфейс:

Отсутствие лишних элементов(нет профилей, лайков). Навигация через линейные списки тредов. Монохромная цветовая схема. Упор на текстовый контент и пользовательские медиа вложения.

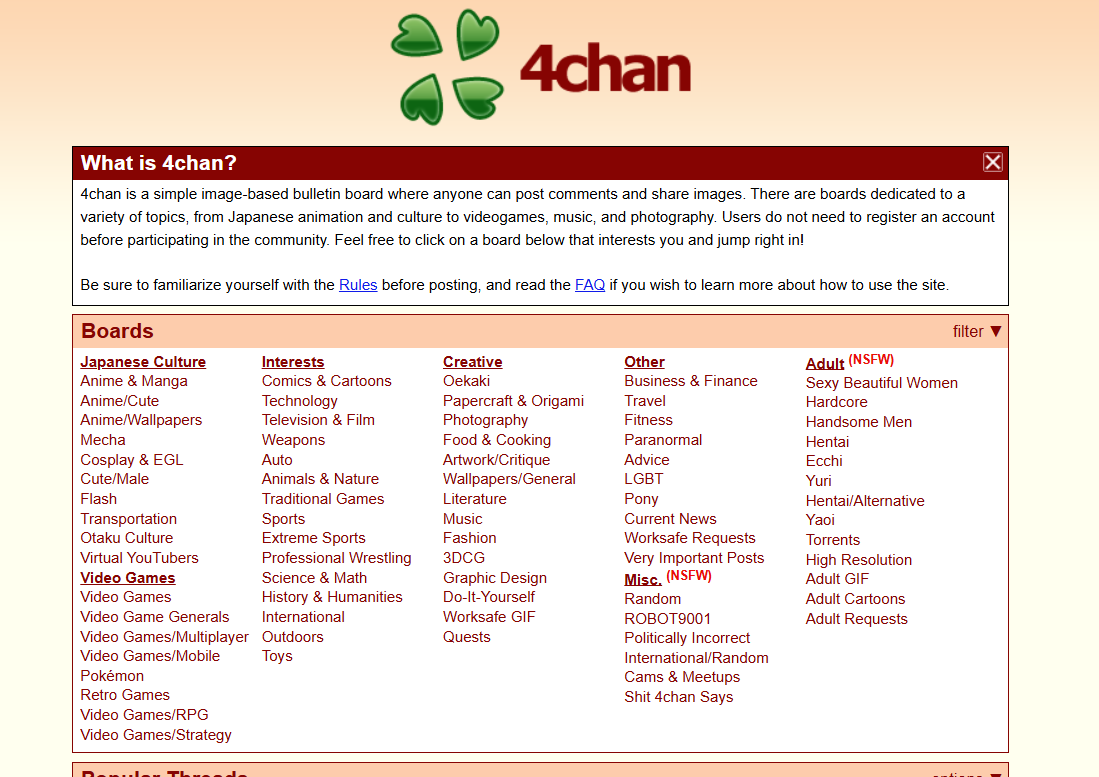


Рисунок 2 лавная страница 4chan

Функциональность:

Создание тредов в определённой доске; Ответы на открытые темы; В ответах есть возможность прикрепления файлы; Есть поле для вписывания настроек сообщения. Защита от спама в виде капчи при каждом новом сообщении; Пользователи публикуют сообщения без регистрации, идентифицируя в момент создания сообщения. Модерация осуществляется через пользовательские жалобы, а не премодерацию. Встроенная возможность форматирования текста.

стек технологий:

Фронтенд: HTML5, CSS3, JavaScript

Бэкенд: PHP + LAMP-стек в ранних версиях

Интерфейс:

Отсутствие лишних элементов(нет профилей, лайков). Навигация через линейные списки тредов. Монохромная цветовая схема. Упор на текстовый контент и пользовательские медиа вложения.

# Техническое задание.

В ходе практики было выявлены сильные и слабые стороны команды разработки и определены требования для проекта. По итогам обсуждения проект должен включать следующие требование к функционалу:

Пользовательский интерфейс должен включать в себя инструменты для просмотра и создания новых сообщений;

Регистрация (email/login + пароль);

Аутентификация (JWT token);

Создание и комментирование тредов;

Для реализация данного функционала, требуется следующая программная архитектура:

RESTful API (JSON) и kafka для синхронного и асинхронного общения сервисов;

Docker (контейнеризация сервисов);

Развёртывание: локально (Docker Compose).

# **Структура программного продукта.**

Общая струтура разрабатываемого продукта представлена на рисунке 3, где:

**Api gateway – перенаправление запросов пользователей на нужный сервис.**

**Post writer – сервис для обработки создания новых постов.**

**Post Reader – сервис на получение постов из БД.**

**Post Reader Sinchonized – отвечает за запись постов в БД ElasticSearch.**

**User Service – отвечает за создание пользователя в PostgreSQL**

**Auth Service – аутентификация пользователя и генерация JWT токена.**

**Image Service – сервис для обработки и хранения медиа файлов(не реализован).**

**WebSocket manager – сервис для отслеживания изменений и** автоматического уведомления клиента о новом посте в рамках сессии(не реализован).

Notify service – уведомление пользователя о новом посте(не реализован).

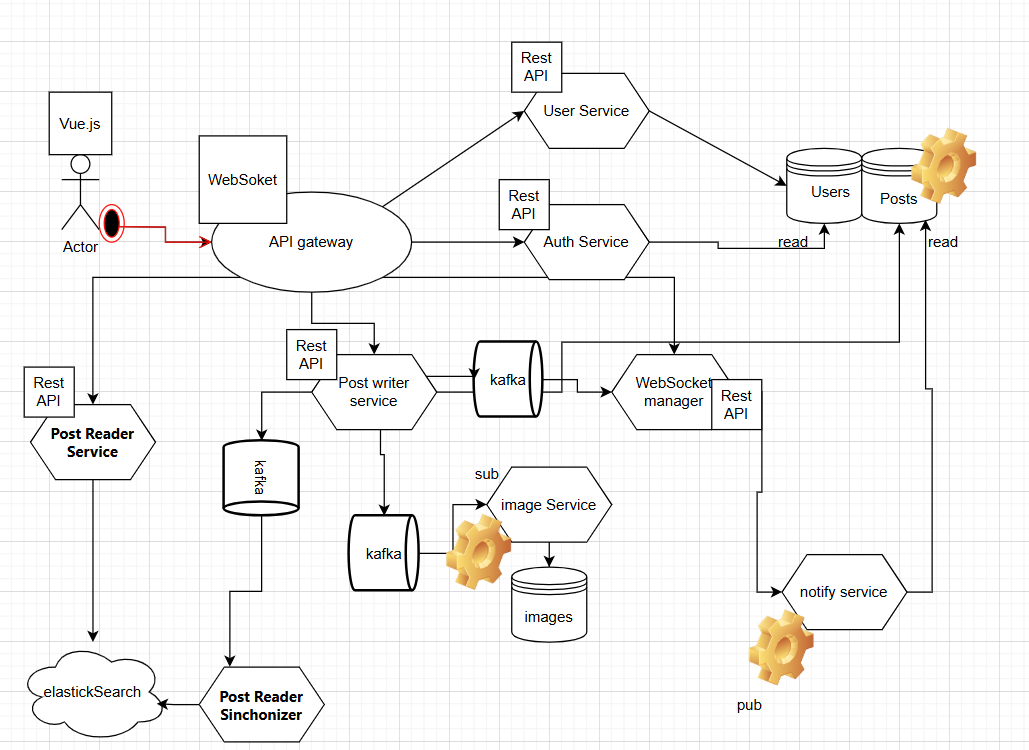


Рисунок 3 архитектура проекта

# Используемые программные технологии и средства.

## Frontend.

Vue.js – для клиентской части и создания и обработки запросов.

## Backend.

Java+Spring – Для сервисов аутентификации и регистрации , Go- для создания сообщений, тредов обработки запросов на получение всех сообщений и работу с БД, Kafka – для асинхронной обработки запросов.

## Базы данных.

ElasticSearch -для сохранение постов, PostgreSQL- для сохранении пользователей.

## Инфраструктура.

Инфраструктура реализована с помощью docker контейнеров и docker-compose файлов для совместной работы контейнеров в одной сети. Для Kafka было развёрнуто 3 брокера сообщений.

## Дизайн интерфейса.

Figma – проектирование пользовательского интерфейса.

# Работа с данными.

**Для хранения информации о пользователях применялась СУБД PostgreSQL, а для хранения сообщений и тредов использовалась нереляционная БД ElasticSearch.**

**Обоснование выбранной архитектуры:**

**PostgreSQL:**

**Идеально для структурированных данных**

**Elasticsearch:**

**Лучше для полнотекстового поиска.**

**Гибкость схемы (можно добавлять поля без миграций).**

**Быстрее в поиске и анализе текста.**

**В Elasticsearch нет ACID (как в PostgreSQL).**

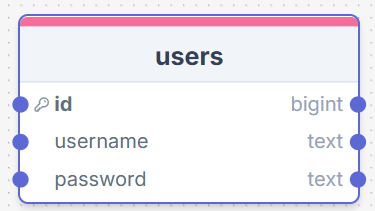
****

Рисунок 4 Таблица пользователей

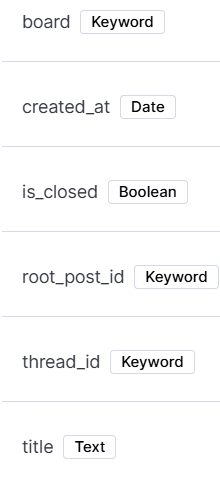
****

Рисунок 4 индекс тредов ElasticSearch

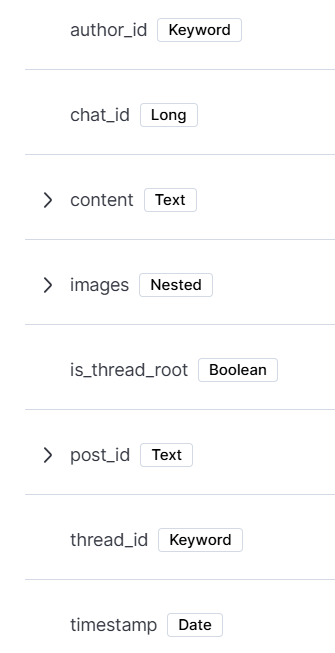


Рисунок 5 индекс сообщений в ElasticSearch

# **Распределение ролей.**

Фаттахов Максим: проектирование, разработка сервисов.

Гаврилов Егор: проектирование, Api Gateway, вёрстка, cоздание js запросов на клиентской части.

Калабина Валерия: Дизайн в Figma, вёрстка.

# Создание тредов и комментариев.

Создание тредов и комментариев происходит через одни и те же сервисы по одним и тем же ручкам, единственное отличие в том, что при создании нового треда, в табдице сообщений создаётся и сообщение порадевшее этот тред и в таблице тредов создаётся новая запись. При отправлении запроса в его теле используется поле is\_thread\_root типа bool которое впоследствии используется в сервисе Post Reader Sinchronized для определения нужно ли создавать новый тред или это просто сообщение в уже существующий тред.

Пример тела запроса для создание сообщения:

{"action":"create","model":"messages","data":{

"post\_id": "12345",

"thread\_id": "101",

"author\_id": "user-789",

"content": "Пример текста сообщения с русскими словами",

"images": [

{

"url": "https://example.com/image1.jpg",

"hash": "a1b2c3d4e5"

}],

"timestamp": "2023-05-01T12:00:00Z",

"is\_thread\_root": true

}}

Action – поле для опредение какое действие делають над запросом, у нас реализован только create, но такая запись поможет для дальнейшего расширения.

Model – над какой таблицей в бд происходит действие, также как и с полем action реализована работа только над messages.

Data – основной блок информации о сообщении.

Post\_id - id будущего поста, если не указано то будет создан автоматический случайный UUID.

thread\_id – id треда в котором опубликовано новое сообщение, в случае если это сообщение пораждающее новый тред, id(UUID)будет создан автоматически.

Author\_id – идентификатор пользователя создавшего это сообщение

Content – текст сообщения.

Images – массив изображений.

url – адресс изображения, на данном этапе нереализованная работа с изображениями

hash – хэш изображений что бы нельзя было в БД загружать одно и то же изображение если такое уже есть.

timestamp – время создания поста если не указанно то будет создано автоматически

is\_thread\_root – является ли это сообщение порождающим треды.

# Возможность регистрации, аутентификации.

Регистрация осуществляется через User Service написанный на Java с использованием Spring Boot. Api Gateway посылает POST запрос пользователя в User Service по адрессу “ /api/users/register”. Где проверяется существует ли уже такой пользователь(по логину). В случае ошибки возвращается 409(Конфликт) ошибка, в ином случае 200(Ok) и в БД PostgreSQL создаётся запись нового пользователя.

Аутентификацией занимается Auth Service написанный на Java с использованием Spring Boot. Api Gateway посылает POST запрос пользователя в User Service по адресу “ /auth/login” с телом {"username":"username ","password":"password "} – для получения JWT токена. В случае неправильного логина или пароля возвращается ошибка 401(UNAUTHORIZED). После успешного получения JWT токена – Vue.js на клиенте сохраняет этот токен в куки браузера, для дальнейшего использования. Клиент при каждом запросе добавляет токен в заголовок Authorization: Bearer <JWT>. Дальнейшие запросы будут проверяться через Auth Service “/api/data”, где проверяется существование токена, не вышел ли срок его действия, если вернулось Ok- значит токен валидный, в ином случае пользователь не аутентифицирован.

# Реализация асинхронной обработки запросов(с помощью kafka).

Реализована обработка в моменте создания нового поста, запрос доходит до сервиса Post writer, и пользователю возвращается ответ об успешном принятии сервером его запроса, затем запрос отправляется к kafka-кластер и перенаправляется в Post Reader Sinchronized который уже добавляет в базу данных Elasticserach, благодаря ассинхронной обработки запросов, в случае высокой нагрузки на базу данных, все запросы будут скапливаться в kafka-кластере в виде очереди, и постепенно разбираться сервисом.

Принципы микросервисов:

1.Один сервис = одна ответственность (например: auth-service, post-writer service);

2.Независимое развертывание (каждый сервис работает в своём контейнере). 3.Слабая связанность (общение через API, а не прямые вызовы).

4.Собственная БД у каждого сервиса (или хотя бы схема).

Выше перечислены базовые принципы микросервисной архитектуры, которые мы пытались достичь, на практике самым сложным для нашего проекта оказался принцип единственной базы данных и мы его не в полной мере реализовали, так как для этого нужно разрабатывать помимо самих бд, ещё и принцип их согласованности что бы в данные в 1ой бд не противоречили данным из другой бд.

# Заключение.

В ходе практики удалось создать рабочее MVP веб-форума (image-board) с базовым функционалом. Приобретены необходимые практические умения и навыки для понимания разработки в сфере микросервисной архитектуры.

Форум готово к запуску и тестированию в реальных условиях, что позволит выявить и устранить возможные недочеты, а также улучшить его функциональность на основе отзывов пользователей.

Принципы микросервисов:

1.Один сервис = одна ответственность (например: auth-service, post-writer service);

2.Независимое развертывание (каждый сервис работает в своём контейнере). 3.Слабая связанность (общение через API, а не прямые вызовы).

4.Собственная БД у каждого сервиса (или хотя бы схема).

Что не удалось реализовать в полной мере: Идеальная микросервисная архитектура: не все сервисы полностью независимы; Глубокая аналитика и модерация; Выше перечислены базовые принципы микросервисной архитектуры, которые мы пытались достичь, на практике самым сложным для нашего проекта оказался принцип единственной базы данных и мы его не в полной мере реализовали, так как для этого нужно разрабатывать помимо самих бд, ещё и принцип их согласованности что бы в данные в 1ой бд не противоречили данным из другой бд.

Дальнейшее развитие: Поддержка уведомлений, лучшение согласованности данных(Добавление Saga-паттерна для транзакций между микросервисами); Реализация WebSocket для live-обновлений; Система рейтингов и модерации; Переход на Kubernetes для оркестрации микросервисов.

Проект успешно реализован как MVP с возможностью масштабирования. Главный урок: микросервисы требуют тщательного проектирования взаимодействий, особенно при работе с разными типами БД.

# **Список литературы.**

1. Хабр: сообщество IT специалистов. Статья работа с Kafka в Go: практическое применение [Электронный ресурс] Режим доступа: <https://habr.com/ru/articles/894056/>
2. go.dev: официальная документация по языку Go. [Электронный ресурс] Режим доступа: [Documentation - The Go Programming Language](https://go.dev/doc/)
3. docs.spring.io: официальная документация по библиотеки Spring для Java [Электронный ресурс] Режим доступа: [Spring Framework Documentation :: Spring Framework](https://docs.spring.io/spring-framework/reference/index.html)
4. Хабр: сообщество IT специалистов. Статья REST API с использованием Spring Security и JWT [Электронный ресурс] Режим доступа: https://habr.com/ru/articles/545610/
5. 2ch: пример image-board [Электронный ресурс] Режим доступа: https://2ch.life/news/
6. 4chan: пример image-board [Электронный ресурс] Режим доступа: <https://4chan.org/index.php>
7. Хабр: сообщество IT специалистов. Статья Работа Apache Kafka на примерах. Поднимаем Kafka Cluster используя docker-compose [Электронный ресурс] Режим доступа: https://habr.com/ru/articles/738874/