High-Level Overview

This system is a distributed, event-driven microservices platform for managing user profiles and associated content blocks, with real-time search indexing. It leverages Apache Kafka for reliable event delivery, Elasticsearch for full-text search, and Spring Cloud Netflix (Eureka, Zuul) for service discovery and API gateway routing.

Core Purpose & Goals

- **User Profiles**: Create, update, delete user profiles stored in PostgreSQL.
- **Content Blocks**: Maintain per-profile collections of “blocks” (e.g. text+link items) stored in MongoDB.
- **Event Bus**: Emit `ProfileEvent` and `BlockEvent` messages on Kafka whenever changes occur.
- **Search Index**: Consume those events in a dedicated Search-Aggregator service, update an Elasticsearch index to enable low-latency, full-text search across profiles and their blocks.
- **Resilience & Scalability**: Each service scales independently, uses Kafka for decoupling, and can be replicated behind the Eureka registry.

Key Microservices & Responsibilities

1. **Profile Service**
   a. Exposes REST endpoints for CRUD operations on user profiles.
   b. Persists profiles in PostgreSQL.
   c. Emits `ProfileEvent` to Kafka on create/update/delete.
   d. Invokes Block Service to provision per-profile block storage on creation.
2. **Block Service**
   a. Manages CRUD operations on a profile’s blocks.
   b. Persists blocks in MongoDB.
   c. Emits `BlockEvent` to Kafka on any change to blocks.
3. **Search-Index-Aggregator Service**
   a. Subscribes to `ProfileEvent` and `BlockEvent` topics.
   b. For each event, fetches latest profile or block data via Feign clients.
   c. Updates or indexes a `profiles` document in Elasticsearch with current username and block list.
4. **Service Registry & API Gateway**

    a. **Eureka** for dynamic service discovery.
    b. **Zuul** (or Spring Cloud Gateway) to route external API calls to the appropriate microservice.
5. **Infrastructure Components**
    a. **Kafka Cluster**: Topics `profile-events` and `block-events` for event streaming.
    b. **Elasticsearch**: Holds a `profiles` index keyed by username, mapping nested block objects for search.
    c. **Databases**: PostgreSQL for structured profile data; MongoDB for flexible block content; Redis (optional) for caching or rate-limiting.

Data Flows & Integration

1. Client creates/updates/deletes a profile via API Gateway → routed to Profile Service.
2. Profile Service writes to PostgreSQL → publishes `ProfileEvent(kafka)` → Eureka-registered.
3. Block Service similarly manages block collections → MongoDB → publishes `BlockEvent(kafka)`.
4. Search-Aggregator Service (Kafka consumer) receives events → calls Profile & Block Services (via Feign) → writes/updates documents in Elasticsearch.
5. Downstream clients query Elasticsearch directly or via a search API for fast, aggregated profile+block search results.

External Dependencies & Configuration

- **Docker Compose** orchestrates local development, bringing up each service, Kafka (Bitnami/KRaft), and Elasticsearch.
- **Wait-for-it** scripts ensure Kafka and Elasticsearch are ready before starting consumers/producers.
- **Spring Boot** configuration files wire up Kafka serializers/deserializers, Elasticsearch repositories or Java client, and Eureka discovery.

By following this architecture, the system achieves loose coupling, event-driven synchronization, and scalable full-text search—all while maintaining clear service boundaries and data ownership.

git :
https://github.com/shyn9yskhan/ziplink-frontend.git
https://github.com/shyn9yskhan/ziplink.git