**RECOMMENDATION SYSTEM**

A recommendation system is a crucial component of many online platforms, including e-commerce, streaming services, social media, and more. It analyzes user behavior and preferences to suggest relevant content, products, or items.

**REQUIREMENTS:**

**FUNCTIONAL REQUIREMENTS:**

1. User Registrations and profile (user can update their profile preferences and interests).
2. The system tracks user interactions such as clicks, views, likes and purchases.
3. User interaction should be collected and processed in real time for up-to-date recommendations.
4. The system should generates personalized recommendations based on user behavior and preferences.
5. The system should uses collaborative filtering techniques to identify user similarities and recommend items.
6. The system addresses the "cold start" problem by providing initial recommendations for new users or items.

**OPTIONAL REQUIREMENTS**

1. Users can provide feedback on recommended items (e.g., thumbs up/down, ratings, reviews).

**NON-FUNCTIONAL REQUIREMENTS**

1. The system should be scalable.
2. The system should be fault tolerant with high accuracy and high performance.
3. Highly available to prevent downtime.

**ESTIMATIONS**

Total active users = 10Million

Daily active users = 1million

No of interactions each user take per day = 10

Daily interactions performed = 10\*1Million = 10million

Each interaction takes 100bytes

Storage per day = 100\*10\*1000000=1GB

**DATA DESIGN MODEL**

**Tables**

User(userId, userName, Email, createdAt)

Item(ItemId, title, category)

Interaction(InteractionId, userId, ItemId, InteractionType, Timestamp)

Recommendation(recommendationId, userId, itemId, score, timestamp)

* The "User" table stores user details such as user ID, username, email, and registration timestamp.
* The "Item" table represents items available for recommendation, with attributes like item ID, title, and category.
* The "Interaction" table records user interactions with items, including the interaction type (e.g., click, view) and timestamp.
* The "Recommendation" table stores generated recommendations for users, including the user ID, recommended item ID, recommendation score, and timestamp.

**API Design**

**Endpoints:**

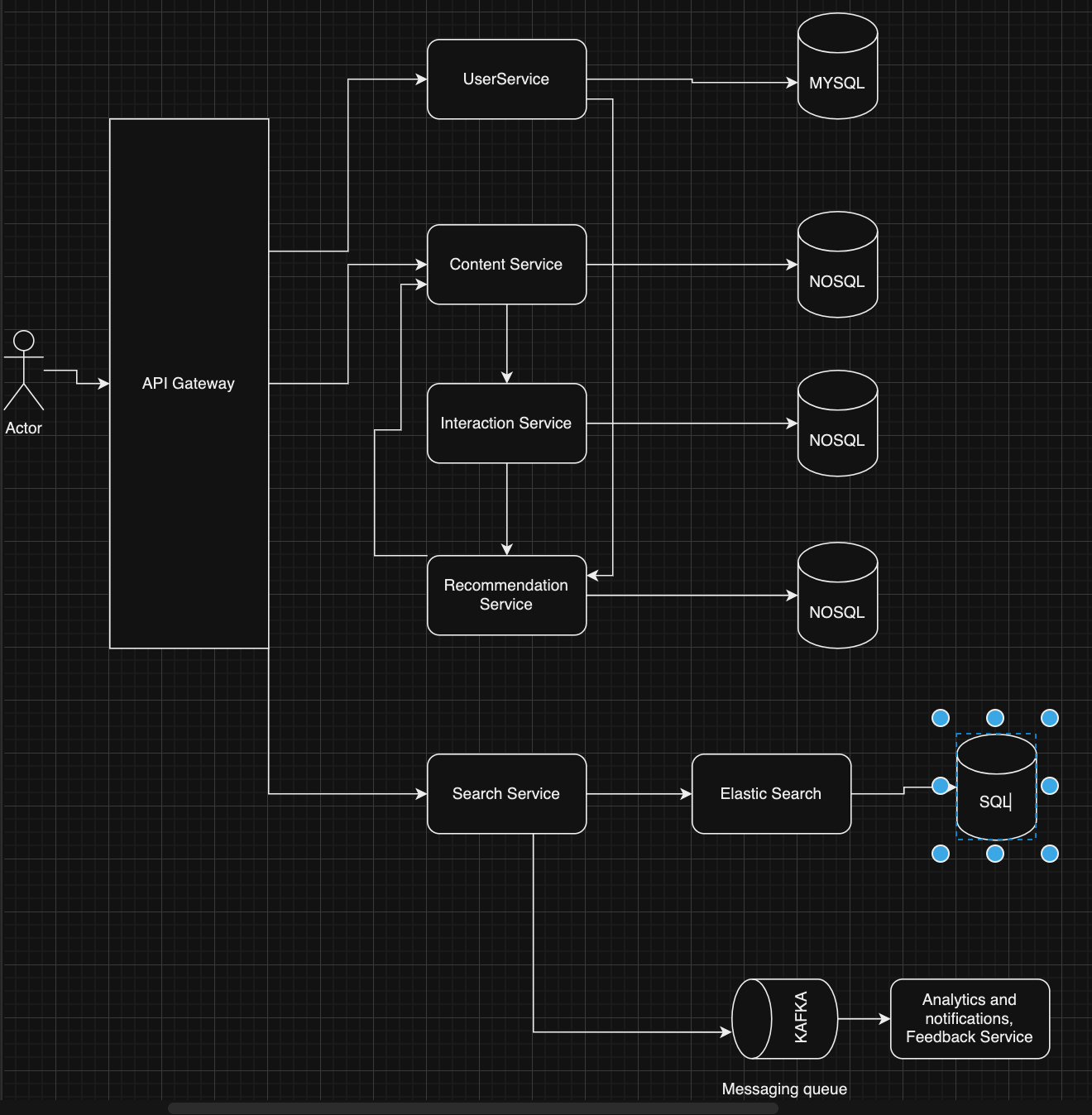
1. **User Registration:**
   * POST /users/register
   * Request Body: { "username", "email", "password" }
   * Response: { "message": "Registration successful" }
2. **User Login:**
   * POST /users/login
   * Request Body: { "email", "password" }
   * Response: { "token": "your.jwt.token" }
3. **Get User Profile:**
   * GET /users/profile
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Response: { "user\_id", "username", "email", "created\_at" }
4. **Record User Interaction:**
   * POST /interactions
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Request Body: { "item\_id", "interaction\_type" }
   * Response: { "message": "Interaction recorded" }
5. **Get Recommendations:**
   * GET /recommendations
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Response: [ { "item\_id", "title", "score" }, ... ]
6. **Search Items:**
   * GET /items/search?q=:query
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Response: [ { "item\_id", "title", "category" }, ... ]
7. **Get Item Details:**
   * GET /items/:item\_id
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Response: { "item\_id", "title", "category" }
8. **Feedback on Recommendations:**
   * POST /feedback
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Request Body: { "recommendation\_id", "feedback\_type" }
   * Response: { "message": "Feedback recorded" }
9. **User Settings:**
   * GET /users/settings
   * Headers: { "Authorization": "Bearer your.jwt.token" }
   * Response: { "notification\_preferences", "display\_preferences" }
10. **Update User Settings:**
    * PUT /users/settings
    * Headers: { "Authorization": "Bearer your.jwt.token" }
    * Request Body: { "notification\_preferences", "display\_preferences" }
    * Response: { "message": "Settings updated" }
11. **Logout:**
    * POST /users/logout
    * Headers: { "Authorization": "Bearer your.jwt.token" }
    * Response: { "message": "Logged out successfully" }

**High Level Design**

1. User Service:
   * Database Type: Relational Database (e.g., PostgreSQL, MySQL)
   * Use: Stores user profiles, authentication data, and preferences.
2. Content Service:
   * Database Type: Relational Database (e.g., PostgreSQL, MySQL) for metadata, and NoSQL Database (e.g., MongoDB) for user-item interactions.
   * Use: Stores item details (metadata) in a relational database and user interactions in a NoSQL database.
3. Interaction Service:
   * Database Type: NoSQL Database (e.g., MongoDB)
   * Use: Stores user-item interaction data, timestamps, and interaction types.
4. Recommendation Engine:
   * Database Type: NoSQL Database (e.g., MongoDB) for storing recommendation scores and user profiles.
   * Use: Stores recommendation scores, user profiles, and potentially item features.
5. Search Service:
   * Database Type: Search Index (e.g., Elasticsearch, Solr)
   * Use: Indexes item attributes and details for efficient search.
6. Feedback Service:
   * Database Type: NoSQL Database (e.g., MongoDB)
   * Use: Stores user feedback on recommendations.
7. Notification Service:
   * Database Type: NoSQL Database (e.g., MongoDB)
   * Use: Stores user notification preferences and pending notifications.
8. Analytics Service:
   * Database Type: Data Warehouse (e.g., Amazon Redshift, Google BigQuery)
   * Use: Stores aggregated and processed data for generating insights.
9. Caching Layer:
   * Database Type: In-Memory Cache (e.g., Redis)
   * Use: Caches frequently accessed data for improved response times.
10. Message Broker:
    * No specific database type, uses messaging systems like RabbitMQ or Apache Kafka for asynchronous communication.

Certainly, let's walk through how data flows through each service in your recommendation system:

1. **User Service:**
   * User data such as registration details, login credentials, and preferences flow from the client interface to the User Service.
   * User profiles and authentication tokens are stored in a Relational Database (e.g., PostgreSQL or MySQL) managed by the User Service.
   * Authentication tokens are generated and sent back to the client for future authorized requests.
2. **Content Service:**
   * Item details (metadata) like titles and categories are stored in a Relational Database (e.g., PostgreSQL or MySQL).
   * When a user interacts with content (e.g., clicks on an item), the Interaction Service records the interaction details.
3. **Interaction Service:**
   * Records of user interactions (e.g., clicks, views) with items flow from the Content Service to the Interaction Service.
   * These interactions are stored in a NoSQL Database (e.g., MongoDB) for efficient and scalable data storage.
4. **Recommendation Engine:**
   * User interactions and preferences are retrieved from the Interaction Service and the User Service.
   * Recommendation algorithms process this data to generate personalized recommendations.
   * Recommendations and associated scores are stored in a NoSQL Database (e.g., MongoDB) managed by the Recommendation Engine.
5. **Search Service:**
   * Item attributes and details are indexed and stored in a Search Index (e.g., Elasticsearch or Solr) by the Content Service.
   * When a user initiates a search, the Search Service queries the index and retrieves relevant search results.
6. **Feedback Service:**
   * User feedback on recommendations (likes, dislikes) is recorded by the Feedback Service.
   * Feedback data is stored in a NoSQL Database (e.g., MongoDB) for analysis and potential algorithm improvement.
7. **Notification Service:**
   * The Notification Service monitors user interactions and preferences.
   * Real-time notifications are triggered based on predefined rules and user preferences.
   * User notification preferences are stored in a NoSQL Database (e.g., MongoDB).
8. **Client Response:**
   * Responses from various services are collected by the API Gateway.
   * The API Gateway structures the responses and sends them back to the client interface for display to the user.

****