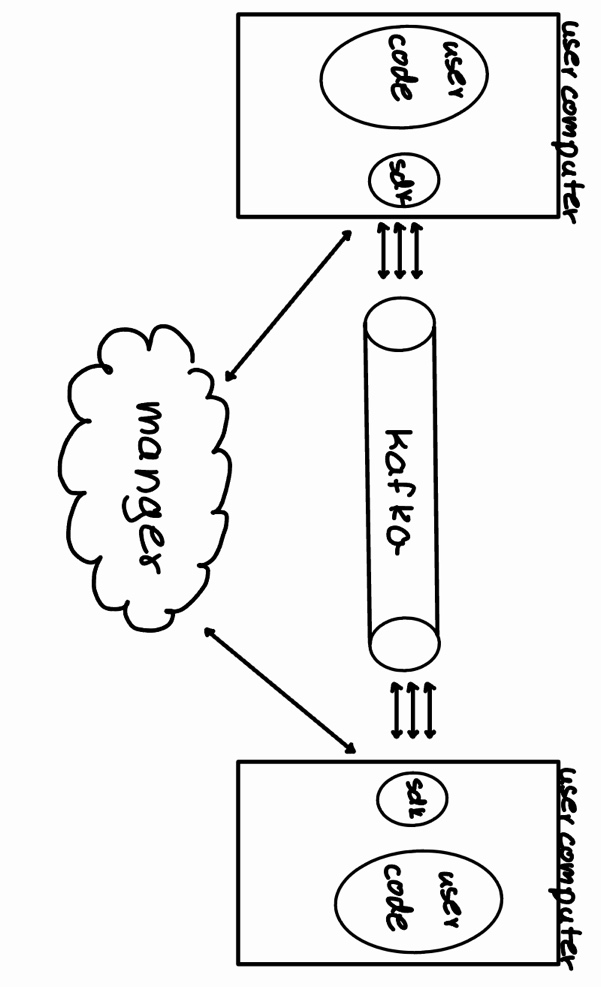
**Kafka Boost High Level Architecture**

**1. System Main Components**

The proposed system is a wrapper for Kafka designed to manage priority-based requests in distributed systems. It introduces a structured way to handle topics and groups, with the main components detailed below:

**1.1 SDK (Software Development Kit):**

* **Functionality:**
  + Routes requests from a group to a specific priority, represented as a Kafka topic.
  + Sends requests to Kafka while listening to topics within the appropriate group.
  + Sorts incoming requests by priority based on predefined metadata.
* **Deployment:** Installed on the client side for every user utilizing the system.

**1.2 Manager:**

* **Functionality:**
  + Converts groups into Kafka topics for each system using the product.
  + Manages metadata defining priorities (e.g., user type, request type).
  + Allows users to define and update priorities via metadata.
  + Metadata configuration occurs during initial setup and can only be modified by restarting the SDK.

**1.3 API:**

* Provides interfaces for:
  + System configuration.
  + Updating priorities and metadata.
* Simplifies the integration process for client applications.

**2. Main User Use Cases**

**2.1 Use Case 1: Submitting a Request**

1. The client application sends a request with priority metadata to the SDK.
2. The SDK routes the request to the appropriate Kafka topic within the group.
3. Kafka processes the request and forwards it to the consumers.

**2.2 Use Case 2: Listening to Requests**

1. The SDK listens to the relevant Kafka topics in the group.
2. It retrieves and sorts incoming requests based on priority.
3. The requests are processed by the client application in the correct order.

**2.3 Use Case 3: Updating Metadata**

1. The user modifies priority definitions in the Manager.
2. The Manager updates the metadata for the relevant groups and topics.
3. The SDK requires a restart to apply the updated metadata.

**3. Back-End Technology**

* **Core Messaging System:** Kafka, extended with the custom wrapper for priority handling.
* **Manager Implementation:**
  + Programming Language: Python or
  + Database: PostgreSQL for metadata storage.
* **API:** RESTful API for configuration and updates.
* **Deployment:** Dockerized services for scalability and flexibility.

**4. Front-End Technology**

**4.1 Technologies**

* **Framework:** React.js for building a user-friendly dashboard.
* **State Management:** Redux or Context API for managing system configurations.
* **Styling:** Tailwind CSS for responsive and modern design.

**4.2 Draft for Main Screens**

1. **Login Screen:**
   * Fields for username and password.
   * "Forgot Password" link.
2. **Dashboard:**
   * Displays groups and their associated Kafka topics.
   * Visual indicators for priority levels and request statuses.
3. **Metadata Configuration Screen:**
   * Allows users to define or modify metadata for groups and topics.
   * Includes fields for user types, request types, and priorities.
4. **API Testing Panel:**
   * Provides tools to test API endpoints.
   * Displays request and response logs.

**5. Additional Considerations**

**5.1 Scalability**

* Horizontal scaling supported via Docker and Kubernetes.
* Kafka’s inherent scalability ensures high throughput.

**5.2 Fault Tolerance**

* Metadata changes require SDK restarts to avoid misalignment during updates.
* Redundant Manager instances ensure high availability.

**5.3 Security**

* Authentication using OAuth 2.0.
* Encrypted communication channels (HTTPS).

**6. Conclusion**

The proposed Kafka Boost enhances distributed systems by introducing priority management to Kafka topics. The SDK, Manager, and API components collectively create a robust system for efficiently handling requests based on predefined priorities. By leveraging Kafka’s strengths and adding an intelligent priority layer, this solution streamlines communication and ensures high-priority tasks are processed first.