SYSTEM ARCHITECTURE

The platform ‘DonHub’ will be used by students and faculty to buy, sell, or exchange academic resources, or even products. Based on that, here are proposed system architectures:

1. **Event-Driven Architecture:** It is a software architectural pattern in which the flow of the system is determined by events. In EDA, events are occurrences or notifications that represent changes in the state of a system or the environment. These events are used as triggers to initiate actions, processes, or communications between different parts of a software system or between multiple systems. EDA is particularly useful in scenarios where real-time responsiveness, scalability, and decoupling of components are essential. EDA can be implemented in ‘DonHub’ by identifying the key events, event producers and event consumers.
2. **Key Events:** The critical events that drive interactions within the application need to be identified. These events could include user actions such as posting a listing, sending a message, making an offer, updates to listings, user interactions (e.g., posting comments), and system events such as notifications, recommendations.
3. **Event Producers:** The event producers are responsible for generating events based on user actions, system updates, or external data integration. These producers need to be implemented throughout the platform. For instance, when a user posts a new listing, an event is generated.
4. **Event Consumers:** Each consumer is designed to perform particular actions or updates based on the events they receive. For example:

* A listing detail page might subscribe to events related to the specific listing and update its view when changes occur.
* A recommendation system might subscribe to user interaction events to refine its recommendations.

1. **Event Broker/Bus:** Implements an event broker or event bus that acts as a central hub for managing and routing events. This component facilitates event distribution from producers to consumers efficiently.
2. **Asynchronous Processing:** Event driven interactions should be asynchronous, allowing components to operate independently. Events are published and consumed without waiting for immediate responses, improving system responsiveness. For example, when listings are created or updated, their corresponding data in the search index can be updated asynchronously. This ensures that search results remain up to date without impacting the user experience.
3. **Microservices or Service Components:** Utilize microservices or service components that can act as both event producers and consumers. These components can generate events based on their functionality and subscribe to events relevant to their responsibilities.
4. **Data Synchronization:** Leverage events to maintain data consistency across different parts of the application. For instance, when a user updates their profile information, related events can trigger updates in messaging, notifications, and search indexes.

