**Event Driven Architecture with RabbitMQ and Springboot Microservices**

Instead of making synchronous API call will make a use of RabbitMQ for Asynchronous communication

A white rectangular object with black text

AI-generated content may be incorrect.

* We have producer and consumer microservices in between Queue where in messages are stored until they are processed.
* In terms of our application once the activity saved into the data base ActivityController/ActivityService can publish the activity that has been saved into the database and that can be publish that particular message into RabbitMQ Queue
* Consumer can consume the message from the Queue one by one and it can process it and it can store recommendation into database or broadcasting it whatever logic you want to define there after the recommendations are generated
* Consumer in our case is the AIService which will be generating the AI recommendation for our users.
* ***Queue*** is first in first out so basically the message that enters first is the first one to be picked up on the consumer side
* The benefit of doing this Queue here is that even if the consumer service/ AI model is down producer just keeps on publishing the message over here let’s there are 10 activities that got registered and consumer was not able to process because of some technical issues or some network delays over there. So there will be 10 messages/activities in the QUEUE that were published and consumer can come in and do the job of processing it one by one
* ***Exchange***: Exchange is basically a decides where the message should go, it takes the producer and routes them to the correct QUEUE.
* We have concept of routing key there will be a key that which will help RabbitMQ know where to send the message .

We have website springAmqp so this applies the core spring concepts to the development of asynchronous communication