1. **Contextualização do cenário:**
   a. O cenário escolhido para desenvolvimento do trabalho consiste em um sistema de chat com diversos usuários, onde cada usuário pode estar em diferentes grupos. Neste contexto, cada usuário poderá enviar mensagens e visualizá-las em tempos indefinidos após terem sido enviadas (comunicação assíncrona). Além disso, é necessário garantir que, mesmo que haja falha no recebimento de uma mensagem que foi enviada, esta possa ser recebida depois pelo usuário destinatário, sendo armazenada até que o recebimento se torne possível.

   b. O uso da comunicação assíncrona se dá pela possibilidade de consumo da mensagem em tempo diferente ao tempo de envio. E o uso da mensageria se dá pela necessidade do chat ser feito de forma distribuída.

   c. No cenário citado acima, o RabbitMQ torna-se muito útil, uma vez que facilita a escalabilidade, tanto de produtores quanto de consumidores de forma independente, além de garantir a entrega das informações de forma assíncrona, com persistência das mensagens, tratamento e prevenção de falhas.

2. **Arquitetura da solução:**
   a. Os componentes produtores serão:
      - Aplicativos Clientes (quando os usuários enviam mensagens);

   b. Os componentes consumidores serão:
      - Aplicativos Clientes (quando os usuários recebem mensagens);
      - RabbitMQ (para armazenar e recuperar as mensagens diretamente nas filas).

   c. Definição do fluxo de mensagens:

      Mensagem de Envio: Usuário envia uma mensagem para um grupo ou para outro usuário.

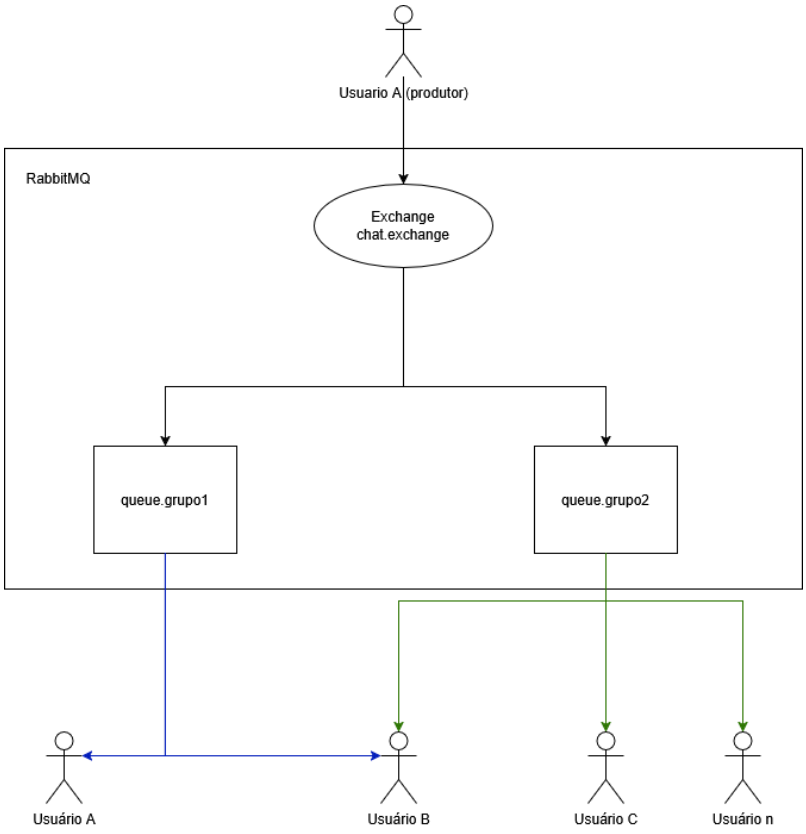
      - Exemplo: Usuário A envia uma mensagem para o grupo X.
      - Tipo: SEND_MESSAGE.
      - Payload: ID do usuário remetente, ID do grupo (ou usuário destinatário), conteúdo da mensagem, timestamp.

Mensagem de Recebimento: Usuário recebe uma mensagem.

- Exemplo: Usuário B recebe a mensagem enviada pelo Usuário A.
- Tipo: RECEIVE_MESSAGE.
- Payload: : ID do usuário destinatário, ID da mensagem recebida, status de leitura.

d. Fluxo de mensagens:

Será realizado conforme imagem abaixo:



e. Estratégias de escalabilidade:

- Escalabilidade Horizontal: A arquitetura pode escalar horizontalmente adicionando mais instâncias de consumidores para cada tipo de serviço:
  - Múltiplos consumidores para a fila de mensagens de envio podem processar mensagens simultaneamente.
  - Cada fila do RabbitMQ pode ser consumida por múltiplos consumidores
- Filas Independentes: Cada tipo de mensagem e cada serviço tem sua própria fila (ou tópico) no RabbitMQ, permitindo que cada serviço escale independentemente, sem afetar os outros.

f. Estratégias de confiabilidade:
   - Persistência das Mensagens: Usar a persistência de mensagens no RabbitMQ, ou seja, configurar as filas como "duráveis" e as mensagens como "persistentes" para garantir que as mensagens não se percam, mesmo em caso de falha do sistema.
   - Confirmação de Recebimento (Acknowledgments): Configurar o RabbitMQ para exigir confirmação de recebimento (ack) para garantir que as mensagens foram processadas corretamente pelos consumidores. Caso uma mensagem não seja confirmada, o RabbitMQ pode reenviar a mensagem.

g. Estratégias de tolerância a falhas:
   - Reenvio de Mensagens Não Recebidas: Caso um usuário não consiga receber uma mensagem em tempo real (devido à desconexão ou falha), a mensagem será armazenada até que o usuário esteja disponível para recebê-la. Isso pode ser feito usando o sistema de persistência de mensagens e garantindo que as mensagens não sejam descartadas até serem entregues ao usuário.
   - Reinício Seguro: O RabbitMQ oferece garantias de que as mensagens não serão perdidas durante reinicializações do servidor, pois elas são persistidas no disco.