

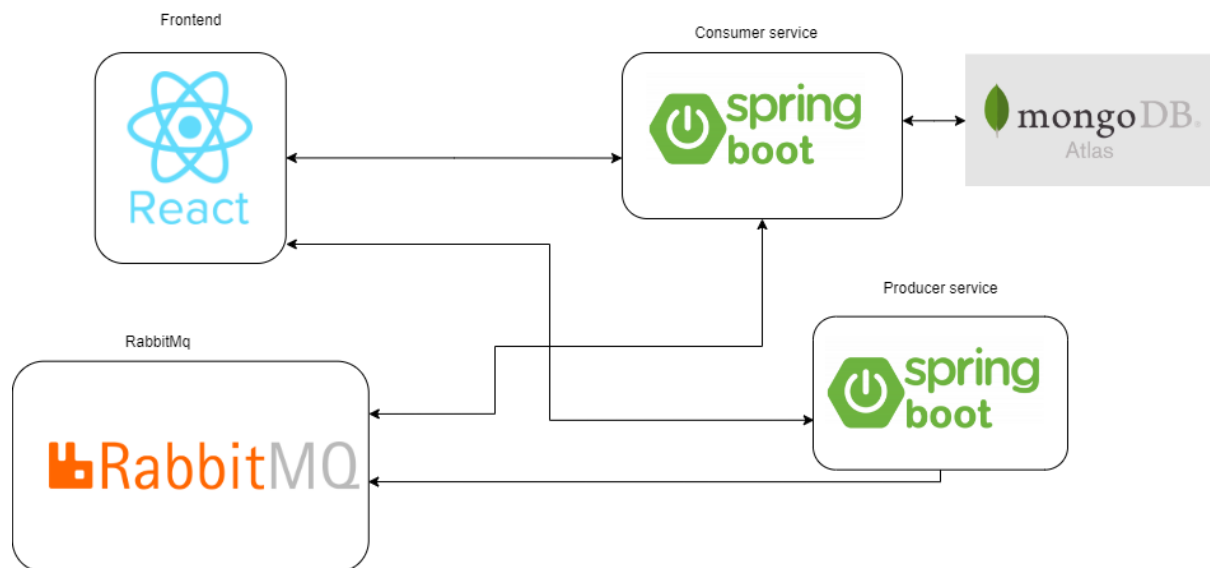
groupe 17

- Gregoire Lequippe
- Ali Loudagh
- Nahel Kini
- Roman Sabechkine

M1 DEV MANAGER FULL STACK (GRP 2)

Rapport de Conception du projet RabbitMq

Tech stack :



Description de l'Architecture

L'architecture de cette application de messagerie comprend plusieurs composants intégrés pour fournir une solution complète et réactive. Voici une description détaillée de chaque composant et de leurs interactions :

1. Frontend (React) :

- Technologie : React
- Rôle : Interface utilisateur interactive.
- Interactions :
 - Avec le service consommateur : Utilise les API REST pour l'authentification, la récupération des données utilisateur et la récupération de messages.
 - Avec le service producteur : L'envoi de messages.
 - Avec MongoDB Atlas : Indirectement, via le service consommateur pour la récupération et le stockage des données utilisateur.
 - Avec WebSocket : Maintient une connexion WebSocket pour la réception de messages en temps réel.

2. Service Consommateur (Spring Boot) :

- Technologie : Spring Boot
- Rôle : Gestion de l'authentification des utilisateurs et de la session, point de terminaison API REST, gestion des connexions WebSocket.
- Interactions :
 - Avec MongoDB Atlas : Stocke et récupère les données utilisateur et les messages.
 - Avec RabbitMQ : Consomme les messages des queues RabbitMQ et effectue les traitements nécessaires.
 - Avec le Frontend : Fournit des API REST et des connexions WebSocket pour les opérations utilisateur et la messagerie en temps réel.

3. Service Producteur (Spring Boot) :

- Technologie : Spring Boot
- Rôle : Traitement et routage des messages.
- Interactions :
 - Avec RabbitMQ : Publie les messages à la queue.

4. Message Broker (RabbitMQ) :

- Technologie : RabbitMQ
- Rôle : Intermédiaire de message, gestion des queues pour la communication entre services.
- Interactions :

- Avec le Service Consommateur : Reçoit les messages publiés et les distribue aux queues appropriées.
- Avec le Service Producteur : Délivre les messages aux services appropriés pour traitement.

5. Base de Données (MongoDB Atlas) :

- Technologie : MongoDB Atlas
- Rôle : Stockage des données utilisateur, des messages et autres informations persistantes.
- Interactions :
 - Avec le Service Consommateur : Stocke et récupère les données utilisateur et les messages.
 - Avec le Service Producteur : (Si nécessaire) Peut être utilisé pour le stockage/récupération des données liées aux messages.

Cette architecture modulaire permet une gestion efficace des opérations utilisateur, une messagerie en temps réel et une communication asynchrone robuste entre les différents composants de l'application. Elle assure également une scalabilité et une maintenabilité élevée, adaptées aux applications web modernes.

RabbitMq :

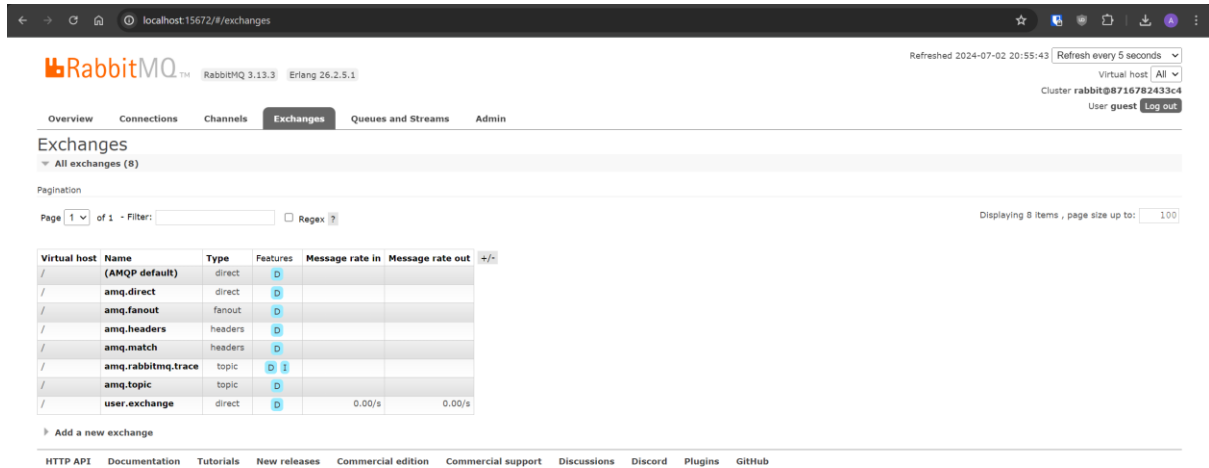
Depuis l'interface de management de RabbitMq, on peut voir qu'une queue « chat.queue » a été créée :

The screenshot shows the RabbitMQ management interface at localhost:15672/#/queues. The 'Queues and Streams' tab is active, displaying a table of queues. The table has columns for Overview, Messages, and Message rates. The queue 'chat.queue' is listed with a classic type, running state, and zero messages.

Overview					Messages			Message rates		
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	Incoming	deliver	get ack
/	chat.queue	classic		running	0	0	0	0.00/s	0.00/s	0.00/s

Search

Ainsi qu'un échange de type direct :



The screenshot shows the RabbitMQ web interface at localhost:15672/#/exchanges. The 'Exchanges' tab is selected, showing a table of 8 exchanges. The table columns are Virtual host, Name, Type, Features, Message rate in, and Message rate out. The exchanges listed are: (AMQP default) [direct], amq.direct [direct], amq.fanout [fanout], amq.headers [headers], amq.match [headers], amq.rabbitmq.trace [topic], amq.topic [topic], and user.exchange [direct]. The user.exchange is the only one with message rates (0.00/s in, 0.00/s out). Below the table is a link to 'Add a new exchange'. At the bottom, there is a navigation bar with links: HTTP API, Documentation, Tutorials, New releases, Commercial edition, Commercial support, Discussions, Discord, Plugins, and GitHub.

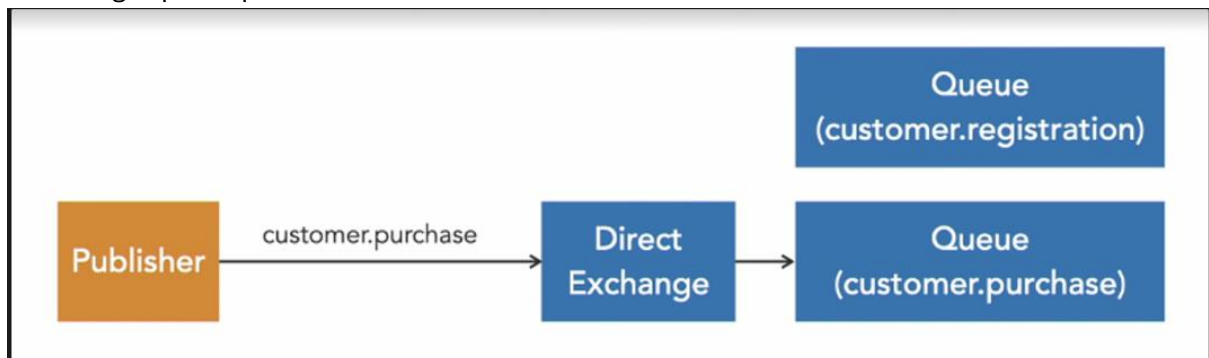
Virtual host	Name	Type	Features	Message rate in	Message rate out
/	(AMQP default)	direct	D		
/	amq.direct	direct	D		
/	amq.fanout	fanout	D		
/	amq.headers	headers	D		
/	amq.match	headers	D		
/	amq.rabbitmq.trace	topic	D, I		
/	amq.topic	topic	D		
/	user.exchange	direct	D	0.00/s	0.00/s

⤵ Add a new exchange

HTTP API | Documentation | Tutorials | New releases | Commercial edition | Commercial support | Discussions | Discord | Plugins | GitHub

Un échange de type direct dans RabbitMQ est un composant qui achemine les messages vers les files d'attente (queues) en fonction d'une clé de routage (routing key) précise. Lorsque le producteur envoie un message à l'échange avec une clé de routage spécifique, l'échange achemine ce message vers toutes les files d'attente qui sont liées à cet échange avec exactement la même clé de routage. C'est un mécanisme de correspondance exacte, où les messages sont livrés uniquement aux queues dont les clés de liaison (binding keys) correspondent parfaitement à la clé de routage du message.

En résumé, un échange direct permet une distribution précise des messages basée sur des clés de routage spécifiques.



Démo :

Page de login :

Chat App

Login

Username

Password

Login

Don't have an account? [Sign up](#)

Page de création de compte :

Signup

Username

Email

Password

Signup

Je vais créer deux comptes avec les logins :

User1 :

Username : Toto

Password : test1234

User2 :

Username : Fafa

Password : test1234

Signup

User created successfully. Redirecting to login...

Username

Toto

Email

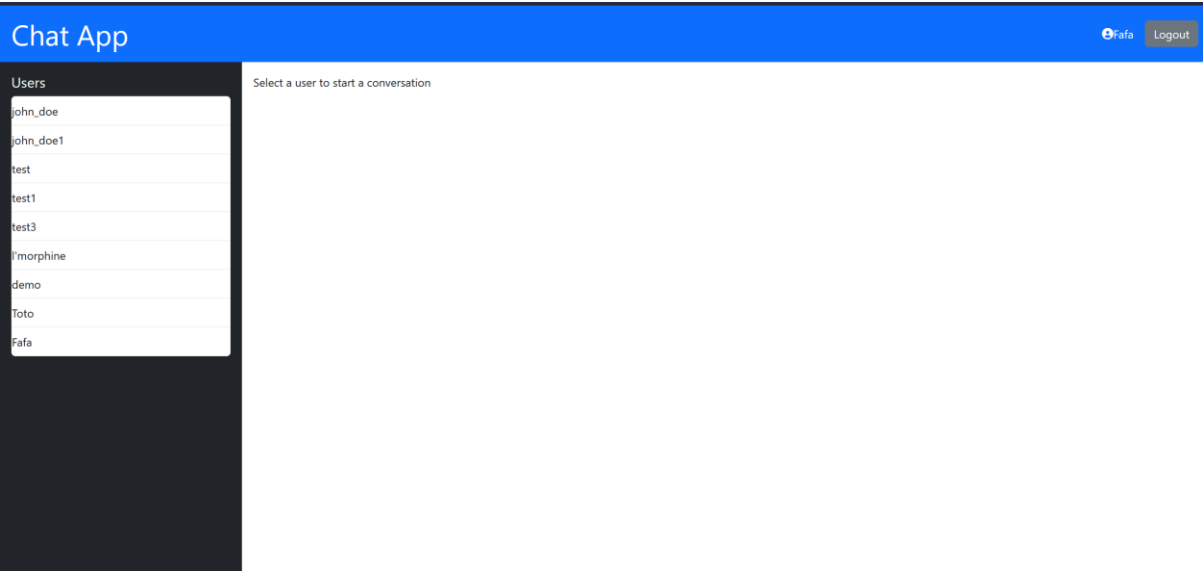
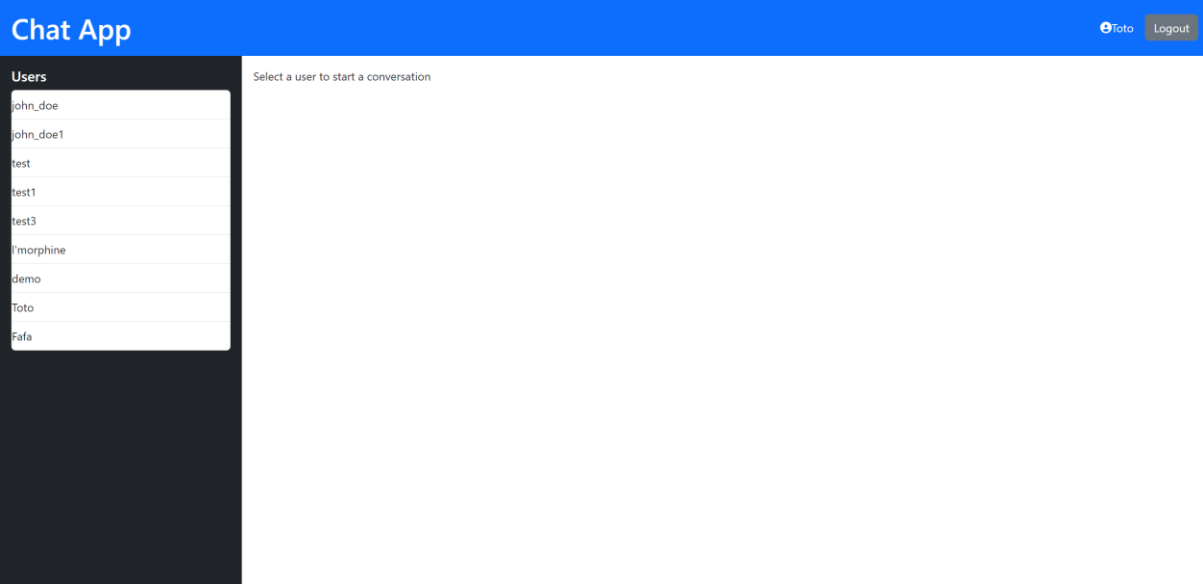
toto@test.com

Password

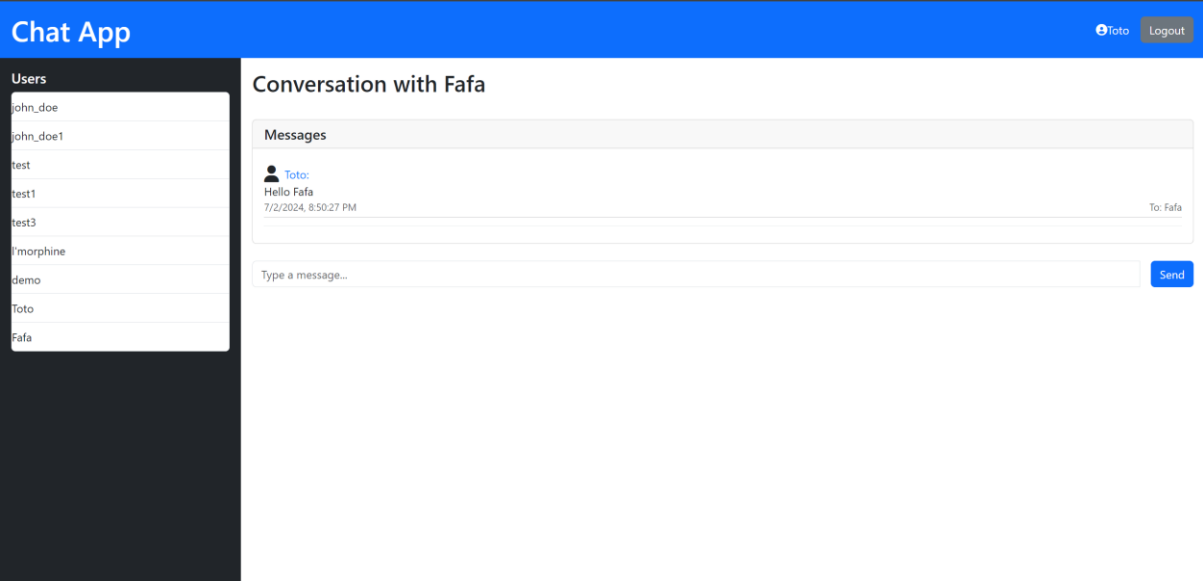
••••••••

Signup

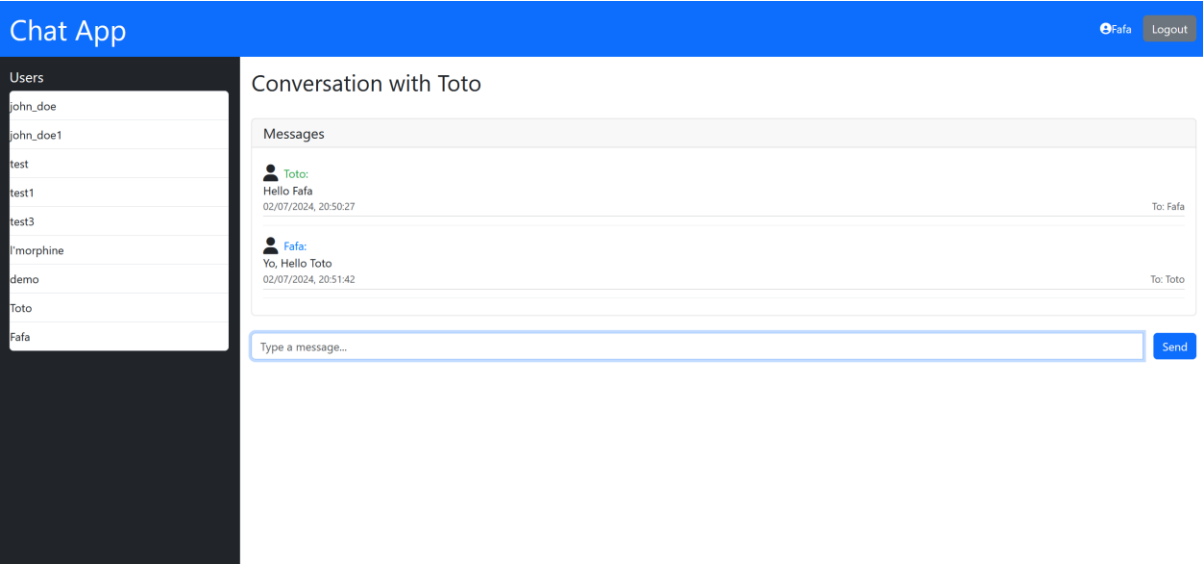
Je me connecte aux deux comptes avec deux navigateurs différents :



Depuis le compte de Toto, j’ouvre la conversation avec Fafa et j’envoie un message :



Je réponds depuis le compte de Fafa :



Logs :

```
2024-07-02 20:50:27 Received and saved message: Message[id='66844bf33ea8ba2a3f0cb3a5', senderId='66844b003ea8ba2a3f0cb3a2', receiverId='66844b233ea8ba2a3f0cb3a3', content='Hello Fafa', timestamp= Tue Jul 02 18:50:27 UTC 2024]
2024-07-02 20:51:42 Received and saved message: Message[id='66844c3e3ea8ba2a3f0cb3a6', senderId='66844b233ea8ba2a3f0cb3a3', receiverId='66844b003ea8ba2a3f0cb3a2', content='Yo, Hello Toto', timestamp= Tue Jul 02 18:51:42 UTC 2024]
```