



Compte Rendu
LICENCE-PROFESSIONNELLE

Filière : IDBD

Matière : BIG DATA ET CLOUD COMPUTING

KafkaTalk

École nationale des sciences appliquées de Kénitra

Réalisé par :

YASSIR EL GHRISSI

Oualid El Ouadoudi

ABDERRAHMANE SLIMANI EL IDRISSE

Sous l'encadrement de :

DR . CHAIMAE KISSI

Année universitaire

2024/2025

Rapport de projet : ChatApp avec Kafka

1. Introduction

- **Contexte du projet** : Besoin de créer une application de messagerie privée dans un environnement distribué.
 - **Objectif** : Permettre la communication fiable et sécurisée entre plusieurs utilisateurs connectés à un serveur.
 - **Problématique** : Comment assurer la synchronisation des messages dans un système distribué, éviter la perte de données et garantir l'ordre d'arrivée des messages ?
-

2. Outils et technologies utilisés

2.1 Apache Kafka

- Rôle : transmission des messages entre les utilisateurs.
- Fonctionnalités clés :
 - Basé sur le modèle **publish/subscribe**.
 - Haute performance et faible latence.
 - Gestion de topics pour organiser les conversations privées.

2.2 Apache ZooKeeper

- Rôle : coordination du cluster Kafka.
- Fonctionnalités clés :
 - Synchronisation des brokers Kafka.
 - Gestion de la configuration et suivi de l'état du cluster.
 - Assurer la cohérence dans le système distribué.

2.3 Python

- Utilisé pour développer le **backend**.
- Responsabilités :
 - Création des **producers** et **consumers** Kafka.
 - Gestion des API REST pour l'envoi/réception des messages.
 - Traitement des données (ex : formatage, stockage si nécessaire).

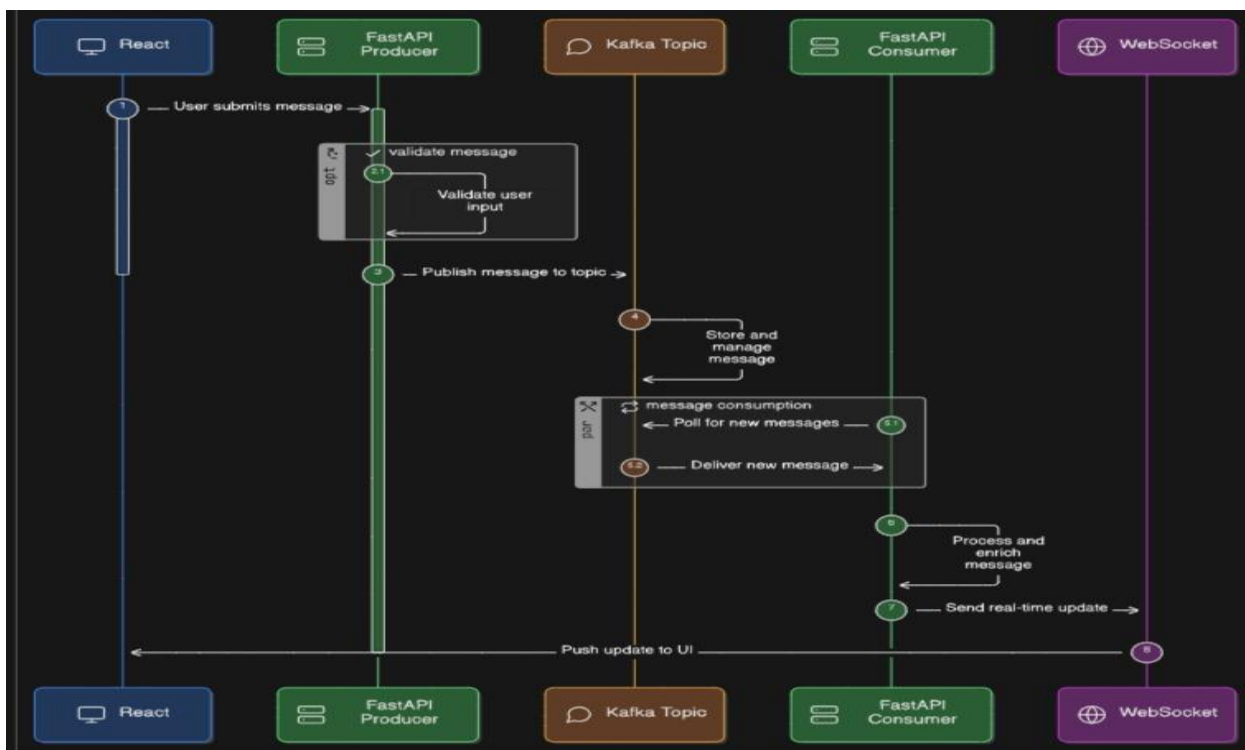
2.4 React

- Utilisé pour développer le **frontend**.
- Responsabilités :
 - Interface utilisateur conviviale pour les conversations.
 - Affichage en temps réel des messages reçus.
 - Intégration avec l'API Python pour envoyer/recevoir des messages

3. Architecture du système

Le système est basé sur une architecture distribuée :

1. **Utilisateur A** envoie un message via l'interface React.
2. Le message est transmis au **backend Python**.
3. Python agit comme **producer Kafka** et envoie le message dans un **topic** spécifique (conversation privée).
4. **Kafka** distribue le message aux **consumers** (utilisateurs concernés).
5. L'interface React de l'**Utilisateur B** reçoit le message en temps réel via le backend Python (consumer).
6. **ZooKeeper** assure la coordination et la fiabilité du cluster Kafka.



4. Fonctionnalités principales

- Création de conversations privées entre deux utilisateurs.
 - Transmission des messages en temps réel.
 - Interface utilisateur moderne et réactive (React).
 - Backend robuste et extensible (Python).
-

5. Déroulement du projet

- **Phase 1 : Conception** → choix des technologies, schéma d'architecture.
 - **Phase 2 : Mise en place de l'environnement** → installation de Kafka et ZooKeeper, création de topics.
 - **Phase 3 : Développement backend** → Python producers/consumers + API REST.
 - **Phase 4 : Développement frontend** → interface React pour la messagerie.
 - **Phase 5 : Tests** → envoi/réception de messages, tolérance aux pannes.
-

6. Résultats obtenus

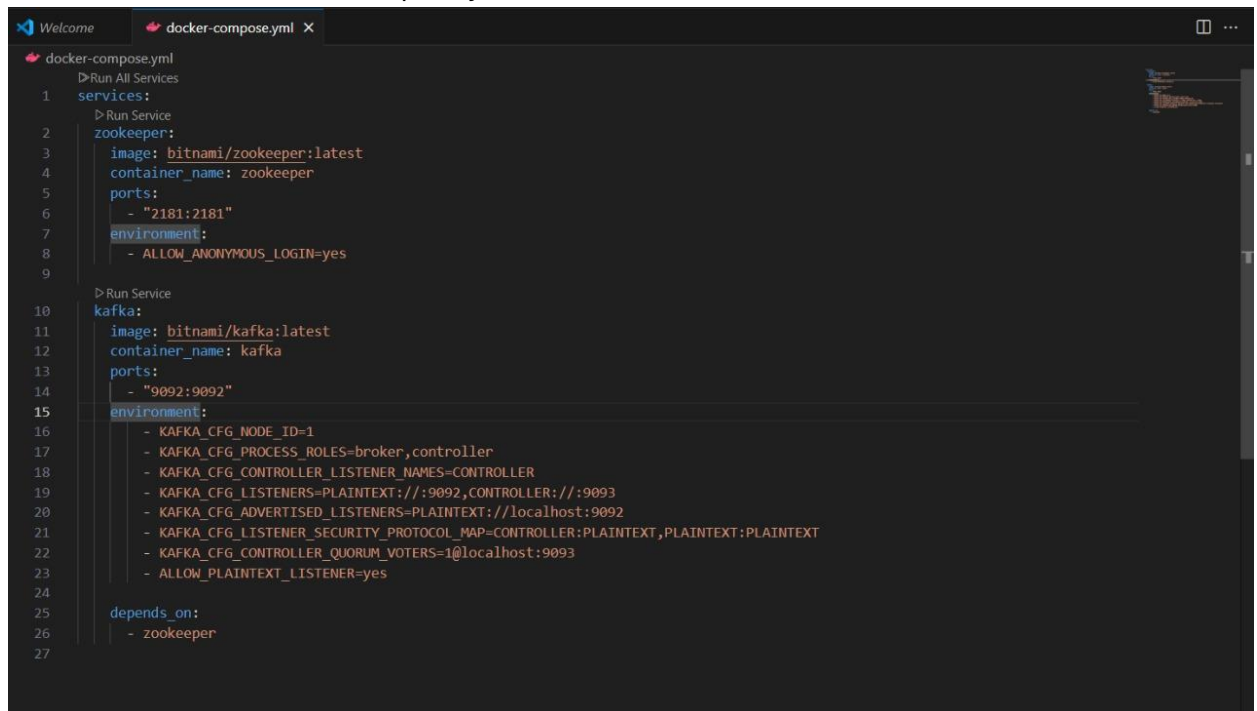
- Messages échangés de façon fluide entre deux utilisateurs.
- Synchronisation assurée grâce à Kafka + ZooKeeper.
- Interface React intuitive et interactive.
- Backend Python performant et facile à maintenir.

Les Etapes de Realisation du Projet :

BACKEND

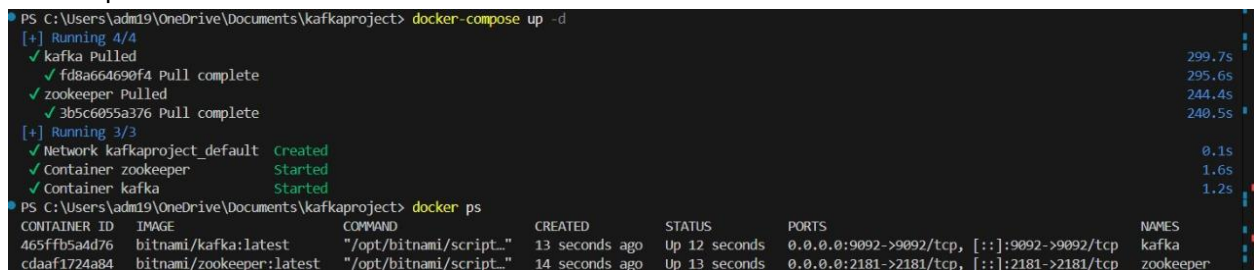
1. Creation du fichier docker-compose.yml :

- Creation du fichier docker-compose.yml



```
1 services:
2   zookeeper:
3     image: bitnami/zookeeper:latest
4     container_name: zookeeper
5     ports:
6       - "2181:2181"
7     environment:
8       - ALLOW_ANONYMOUS_LOGIN=yes
9
10  kafka:
11    image: bitnami/kafka:latest
12    container_name: kafka
13    ports:
14      - "9092:9092"
15    environment:
16      - KAFKA_CFG_NODE_ID=1
17      - KAFKA_CFG_PROCESS_ROLES=broker,controller
18      - KAFKA_CFG_CONTROLLER_LISTENER_NAMES=CONTROLLER
19      - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092,CONTROLLER://:9093
20      - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092
21      - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT
22      - KAFKA_CFG_CONTROLLER_QUORUM_VOTERS=1@localhost:9093
23      - ALLOW_PLAINTEXT_LISTENER=yes
24
25    depends_on:
26      - zookeeper
```

- Demarrer le fichier pour Puller l'image du kafka et Zookeeper
- Docker ps : lister l'ensemble des conteneur



```
PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker-compose up -d
[+] Running 4/4
  ✓ kafka Pulled                                299.7s
  ✓ fd8a664690f4 Pull complete                  295.6s
  ✓ zookeeper Pulled                            244.4s
  ✓ 3b5c6055a376 Pull complete                  240.5s
[+] Running 3/3
  ✓ Network kafkaproject_default Created         0.1s
  ✓ Container zookeeper Started                 1.6s
  ✓ Container kafka Started                     1.2s

PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS                               NAMES
465ffb5a4d76   bitnami/kafka:latest "/opt/bitnami/script..." 13 seconds ago Up 12 seconds 0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp kafka
cdaaf1724a84   bitnami/zookeeper:latest "/opt/bitnami/script..." 14 seconds ago Up 13 seconds 0.0.0.0:2181->2181/tcp, [::]:2181->2181/tcp zookeeper
```

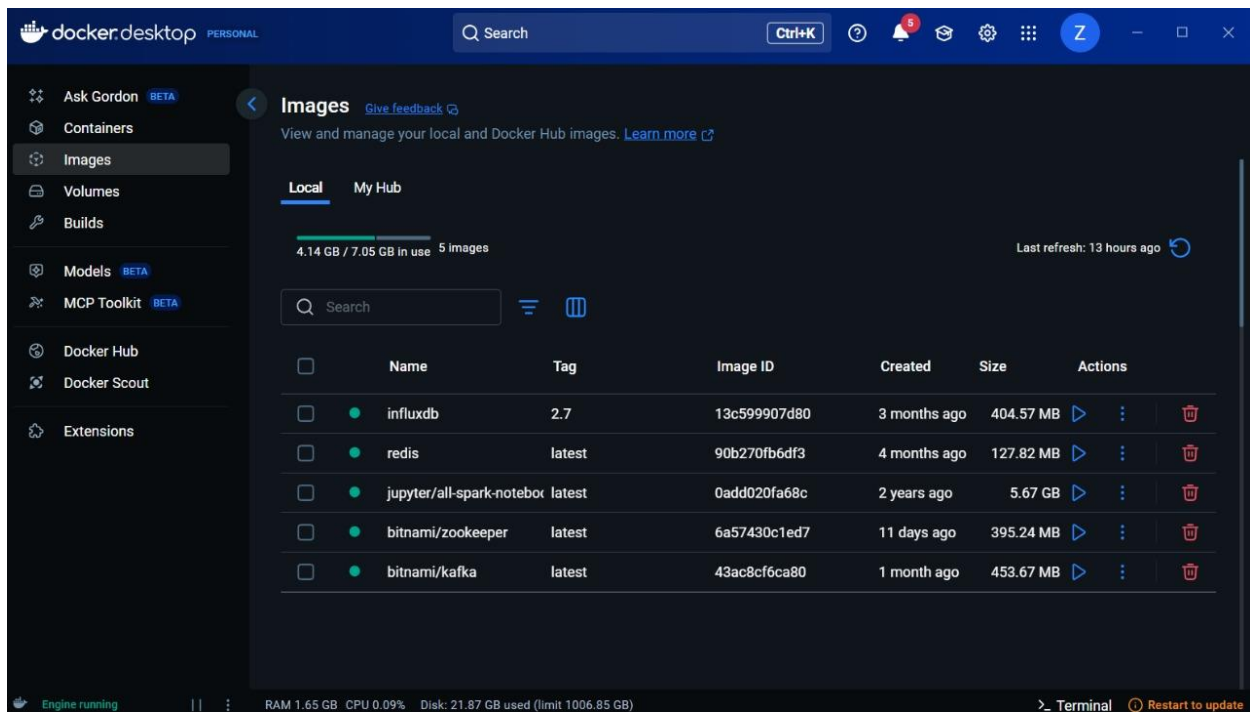
- Lister les topics dans le conteneur kafka
- La creation d un topic 'chat-messages'
- Lexecution de conteneur kafka avec topic 'chat-messages'

```

PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
465ffb5a4d76   bitnami/kafka:latest               "/opt/bitnami/script..." 13 seconds ago Up 12 seconds 0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp kafka
cdaaf1724a84   bitnami/zookeeper:latest           "/opt/bitnami/script..." 14 seconds ago Up 13 seconds 0.0.0.0:2181->2181/tcp, [::]:2181->2181/tcp zookeeper
PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker exec -it kafka kafka-topics.sh --bootstrap-server localhost:9092 --list
>>
>>
PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker exec -it kafka kafka-topics.sh --bootstrap-server localhost:9092 --create --topic chat-messages --partitions 1 --replication-factor 1
PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker exec -it kafka kafka-topics.sh --bootstrap-server localhost:9092 --create --topic chat-messages --partitions 1 --replication-factor 1
>>
>>
Created topic chat-messages.
PS C:\Users\adm19\OneDrive\Documents\kafkaproject> docker exec -it kafka kafka-topics.sh --bootstrap-server localhost:9092 --list
>>
chat-messages
PS C:\Users\adm19\OneDrive\Documents\kafkaproject>

```

- Verifier le demarage du conteneur :



2. Creation du fichier Main.py:

- Creation du code de FastAPI :

```

from fastapi import FastAPI, WebSocket
from kafka import KafkaProducer, KafkaConsumer
import threading
from aiokafka import AIOKafkaConsumer
from fastapi.middleware.cors import CORSMiddleware
producer = KafkaProducer(bootstrap_servers="localhost:9092")

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"], # ton frontend
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

```

- Creation du code de Producer

```

# Kafka Producer

from pydantic import BaseModel

class Message(BaseModel):
    username: str
    message: str

@app.post("/send/")
async def send_message(msg: Message):
    data = f"{msg.username}: {msg.message}"
    producer.send("chat-messages", data.encode("utf-8"))
    return {"status": "sent", "message": data}
import asyncio

```

- Creation du code de Consumer

```
# Kafka Consumer
@app.websocket("/ws/chat")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    consumer = AIOKafkaConsumer(
        "chat-messages",
        bootstrap_servers="localhost:9092",
        group_id=None, # chaque websocket reçoit tous les messages
        auto_offset_reset="latest"
    )

    await consumer.start()
    try:
        async for msg in consumer:
            await websocket.send_text(msg.value.decode("utf-8"))
    finally:
        await consumer.stop()
```

Lancement du backend :

```
PS C:\Users\AGX\Desktop\Kafka_Project\backend> python -m uvicorn app.main:app --reload --port 8000
INFO: Will watch for changes in these directories: ['C:\\Users\\AGX\\Desktop\\Kafka_Project\\backend']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [32568] using WatchFiles
INFO: Started server process [5680]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

FRONEND

Lancement du Frontend :

Npm start

```
Compiled successfully!

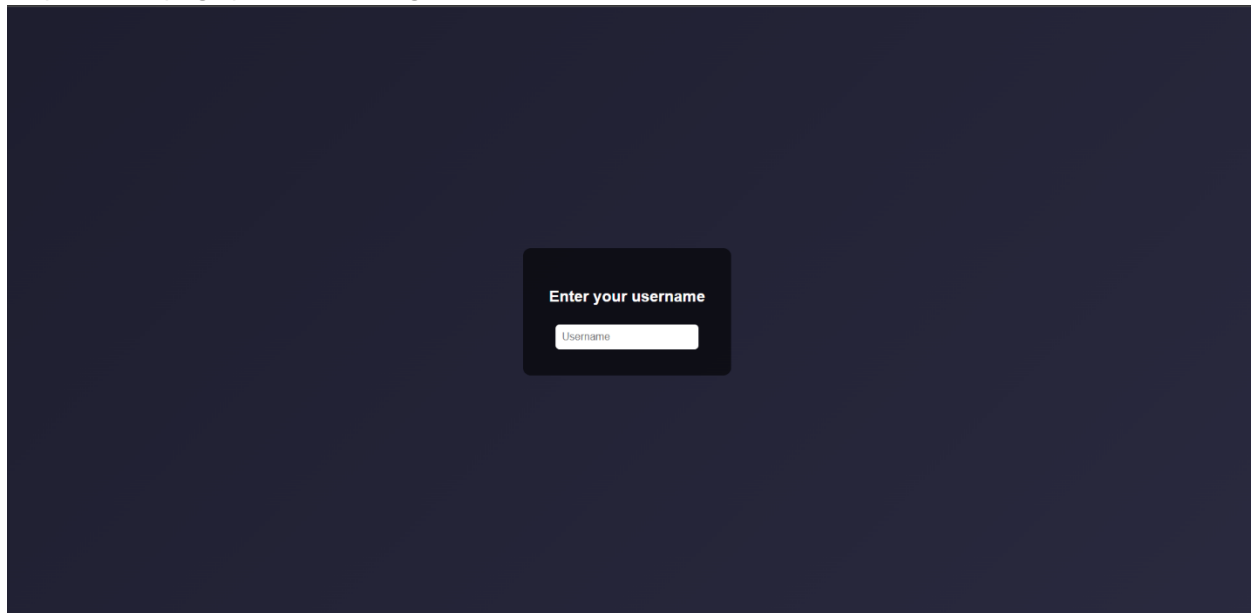
You can now view frontend in the browser.

Local:      http://localhost:3000
On Your Network:  http://172.20.48.1:3000

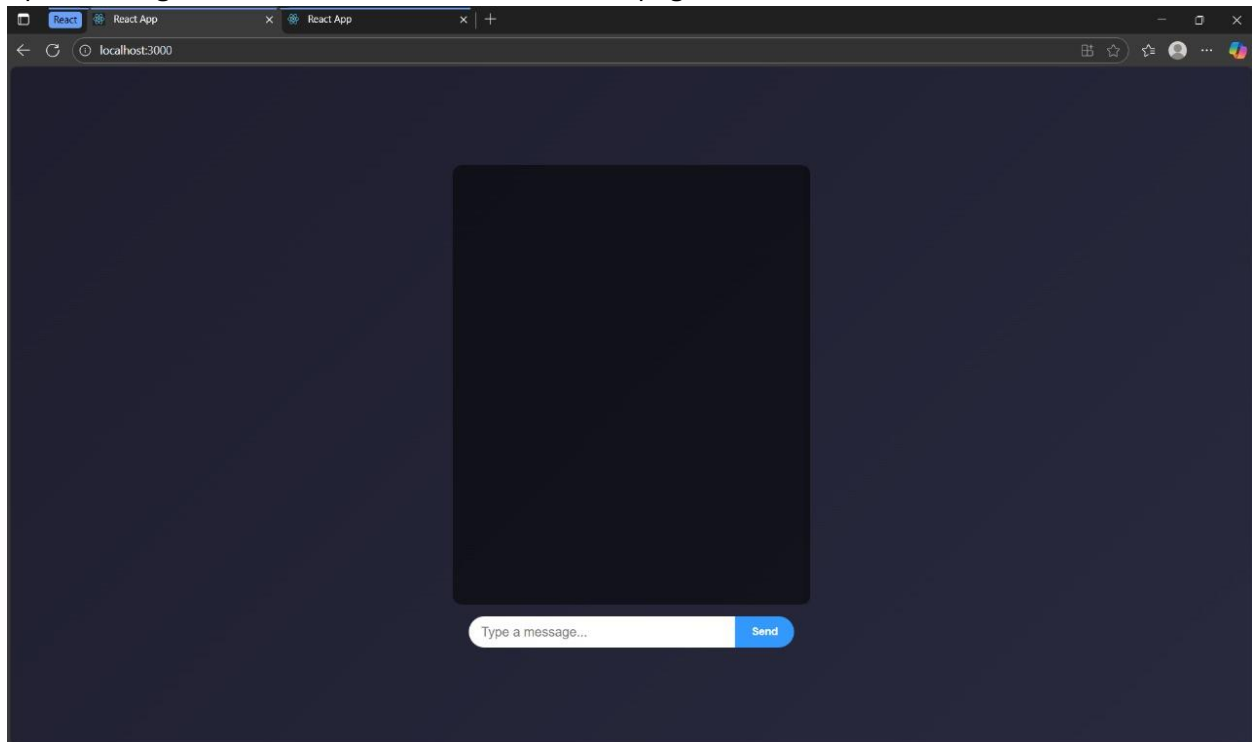
Note that the development build is not optimized.
To create a production build, use npm run build.

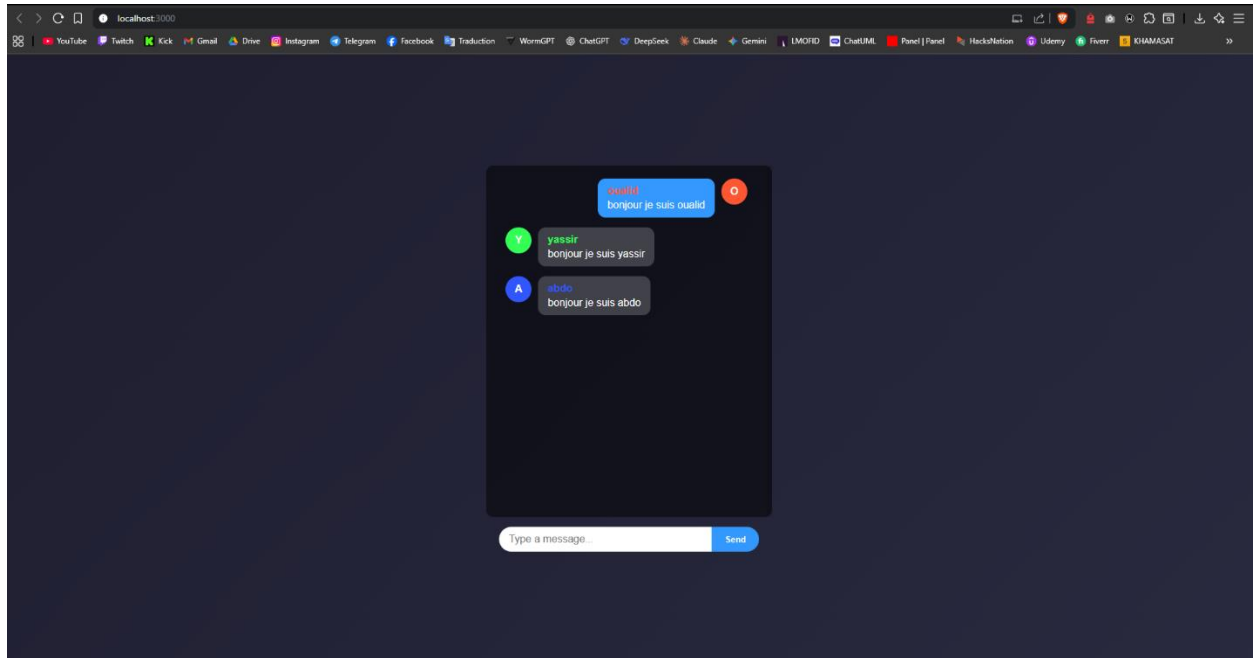
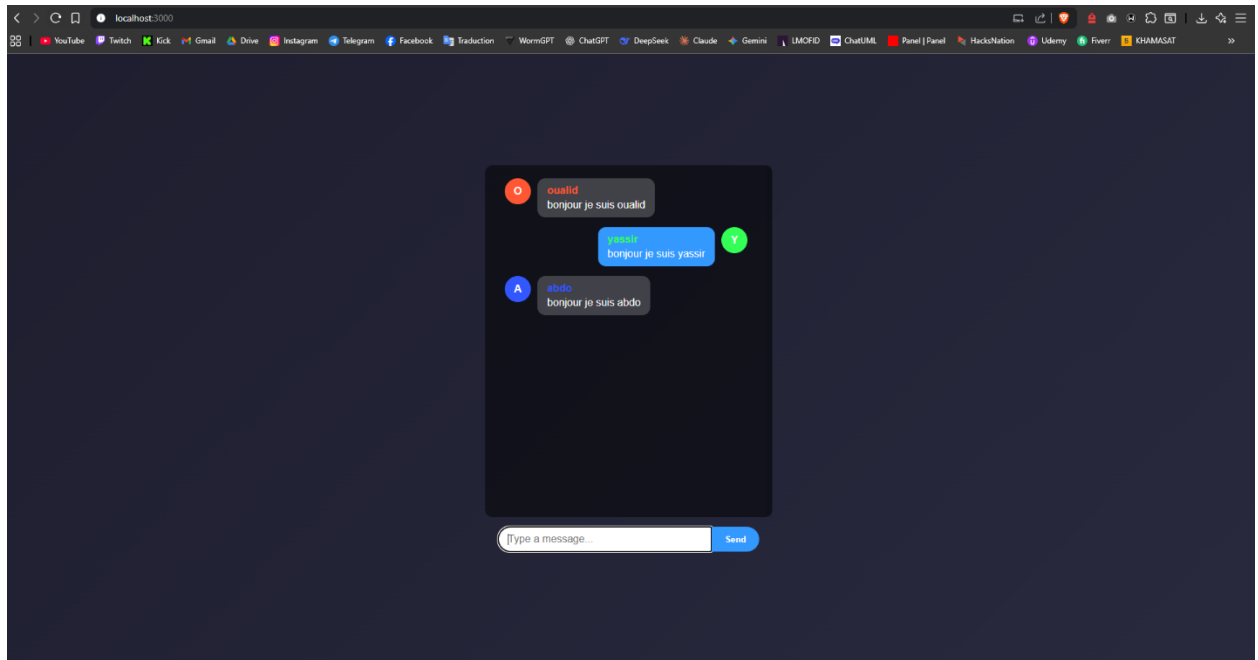
webpack compiled successfully
```

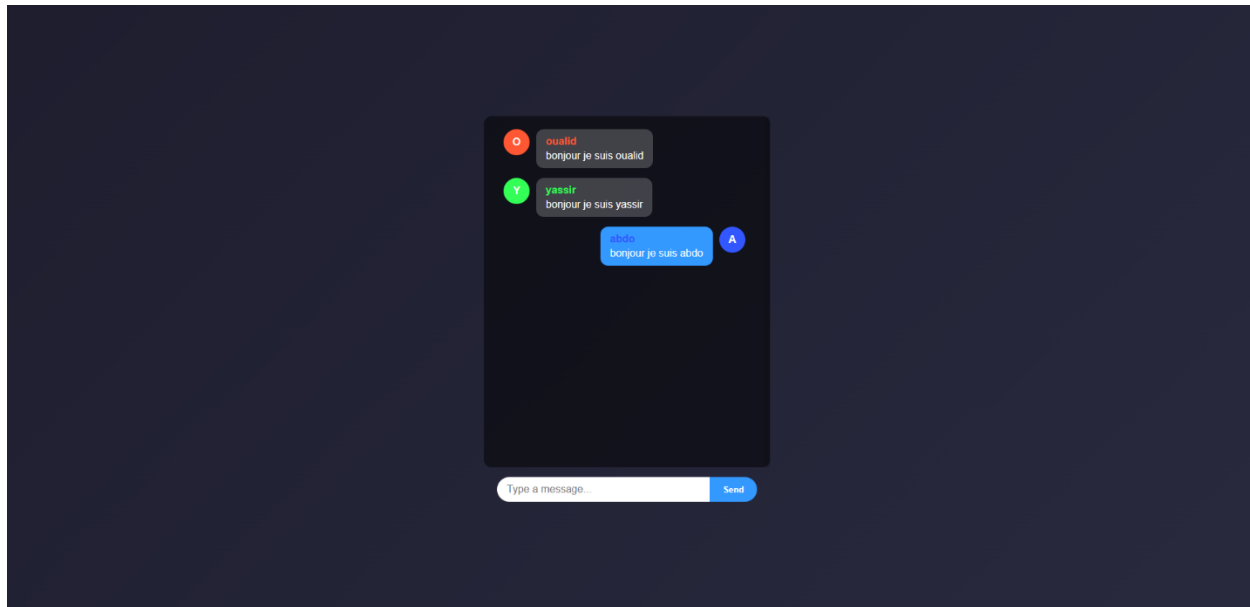

La premiere page permet d'enregister username de visiteurs :



Apres l'enregistrement du username s'affiche une page de conversation :







Conclusion

Ce projet démontre la faisabilité d'une **application de messagerie privée distribuée** en s'appuyant sur Kafka et ZooKeeper pour la fiabilité, Python pour la logique serveur, et React pour l'expérience utilisateur.

Nous, tenons à vous adresser nos sincères remerciements pour la qualité de votre enseignement et pour votre engagement constant à nos côtés. Votre pédagogie, votre disponibilité et votre passion pour la transmission du savoir nous ont profondément marqués et ont grandement enrichi notre parcours académique.

Nous vous sommes très reconnaissants pour vos efforts, votre patience et l'inspiration que vous nous apportez à travers vos cours.