

# Projet ALBC

*Membres de groupe:*

## **KOTBI Abderrahmane & El HAFI Abdessamad & Salah Eddine QATAB**

*La première partie contient les éléments suivants:*

- *Liste des fonctionnalités recensé par module.*
- *Les inputs et outputs pour chaque fonctionnalité .*
- *Les interconnexions éventuelles entre Fonctionnalités.*

*La deuxième partie contient les éléments suivants:*

- *Une description de comment les services sont reliés.*
- *Une manifestation des différents cas d'interactions avec l'utilisateur .*
- *Une description détaillée de chaque cas.*

*La troisième partie contient les éléments suivants:*

- *Décomposants des technologies utilisées par niveaux.*
- *Une présentation des différents drivers pour chaque niveaux.*
- *Une description détaillée de chaque choix technique.*

## Modules, Fonctionnalités, et Inputs/Outputs

- 1) Tweet Service
- 2) Search service
- 3) User TimeLine Service
- 4) Home TimeLine service
- 5) Social Graph Service
- 6) Direct messages Service

## Les interconnections

### Architecture Fonctionnelle

- 1) Service direct messages
- 2) Service du Tweet
- 3) Service home timeline
- 4) Service Social Graph
- 5) Service search
- 6) Service User Time Line

### Architecture Technique

- 1) Service direct messages
- 2) Service social graph
- 3) Service home timeline
- 4) Service user timeline
- 5) Service search
- 6) Service tweet
- 7) Service twitter gateway

## Cas d'interactions

### Drivers fonctionnels

- 1) Base de données
- 2) Back-end
- 3) Front-end
- 4) Déploiement et automation
- 5) Orchestration

### Implémentation (Proof of Concept)

- 1) Twitter home page
- 2) Sign in page
- 3) User TimeLine (mes tweets)
- 4) Voir un tweet
- 5) Créer un tweet
- 6) Editer un tweet
- 7) Supprimer un tweet
- 8) Afficher la liste des following
- 9) Ajouter un follow à un user
- 10) Ajouter un follow à un user
- 11) Modifier le statut Follow/Unfollow à un user
- 12) Consulter la liste de tous les messages.
- 13) Les services sur docker
- 14) Base de données

## Modules, Fonctionnalités, et Inputs/Outputs

### 1) Tweet Service

*le service qui gère tous les tweets publiés, ainsi que toutes les données associées. Il permet également la publication, la suppression et la modification des tweets.*

Fonctionnalités	Inputs	Outputs
Publication des tweets	Texte, images, personnes, dates, localisations...	post sur le User Timeline
Modification des tweets	Tweet, nouveau contenu	même post sur le User Timeline
Suppression des tweets	Tweet, validation de choix (car c'est irréversible)	suppression de tweet

### 2) Search service

*la fonctionnalité de recherche gérée par la barre de recherche Twitter, elle permet de rechercher des personnes, des sujets et mots-clés.*

Fonctionnalités	Inputs	Outputs
Recherche des Personnes	mots-clés ou nom d'une personne	liste des sujets relatives aux mots-clés
Historique des recherches	User	liste des recherches faites précédemment

### 3) User TimeLine Service

*pour afficher son propre TimeLine, qui ne contient que les Tweets publiés par l'utilisateur et leurs retweets.*

Fonctionnalités	Inputs	Outputs
Affichage des postes	User/Page Id	Mure de la personne
Possibilité d'interagir	User/Page Id, post	réactions
Possibilité de commenter	User/Page Id, post	commentaires
Possibilité de Retweeter	User/Page Id, post	Le même post apparaît sur le mur de la personne
Possibilité de consulter les commentaires	User/Page Id, post	Lecture des commentaires

#### 4) Home TimeLine service

*pour afficher le Home TimeLine où toutes les publications des personnes que l'utilisateur suit apparaissent. Ça permet également d'afficher des rubriques de publicités.*

Fonctionnalités	Inputs	Outputs
Affichage des postes	User/Page Id	Mure de la personne
Possibilité d'interagir	User/Page Id, post	réactions
Possibilité de commenter	User/Page Id, post	commentaires
Possibilité de Retweeter	User/Page Id, post	Le même post apparaît sur le mur de la personne
Possibilité de consulter les commentaires	User/Page Id, post	Lecture des commentaires
Publicités	Préférences personnelles	Publicités personnalisées

## 5) Social Graph Service

*pour gérer les relations entre les utilisateurs (Follow, Block...)*

Fonctionnalités	Inputs	Outputs
S'abonner	User	Voir un contenu relative à la personne sur le home timeline, voir ses activités...
Bloquer	User	La personne ne peut ni suivre l'autre personne en question, ni lui envoie un message, ni voir son timeLine...
Se désabonner	User	La personne ne peut continuer à voir les postes de la personne en question mais moins fréquemment, et elle peut voir les siennes
Être suivi	User	Le contenu relative à la personne va être promu pour l'autre personne en question sur le home timeline, ainsi que ses activités...

## 6) Direct messages Service

*Pour la gestion des conversations entre les utilisateurs*

Fonctionnalités	Inputs	Outputs
envoyer une demande d'envoi de message	message (text,image,lien..)	accusé de réception de la demande
envoyer un message	message(text, image, lien)	accusé de réception du message
recevoir la notification d'un message	User	affichage de notification
recevoir un message	User	affichage d'un nouveau message
voir la historique des messages	User	liste des messages
réagir à un message	User / message	réaction afficher sur le message

# Les interconnexions

Plusieurs fonctionnalités peuvent voir des interconnexions. Ceci se voit clairement par des dépendances, des partages de données, etc... Il est important de prendre en considérations toutes ces dernières. Puisque elles permettent de concevoir toute forme de partage et dépendances entre les fonctionnalités.

## 1. Pour les fonctionnalités du Tweet Service

Les fonctionnalités de publication, de modification, et de suppression ont une partie commune entre eux car elles opèrent sur le même composant qui est un post. Notamment, la modification et la suppression d'un post dépend de la création de ce dernier.

## 2. Pour les fonctionnalités du TimeLine Service

Les fonctionnalités du Timeline Service, qui sont dans l'affichage des postes, la réaction, la création des commentaires et le retweet opèrent toutes ensemble sur un post. En effet, les réactions et les commentaires n'ont aucun sens s'il ne sont pas liés à un post.

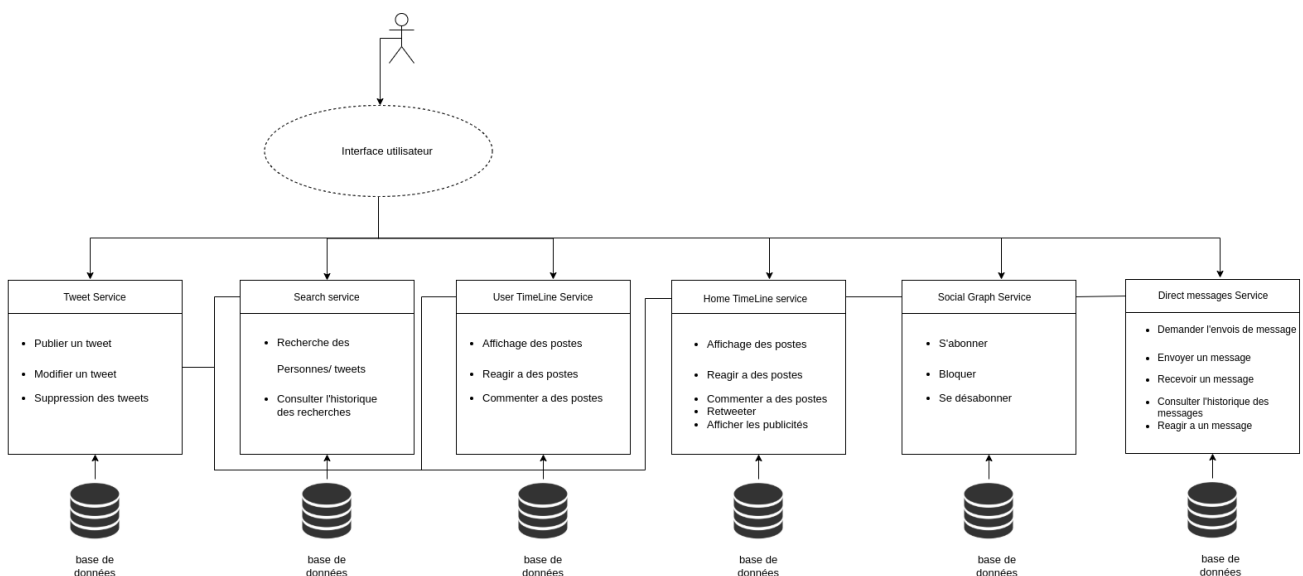
## 3. Pour les fonctionnalités du Direct messages Service

Les fonctionnalités du Direct messages services qui se résument dans l'envoi et la réception des messages sont liées entre eux. En effet, un message envoyé est destiné à être lu.

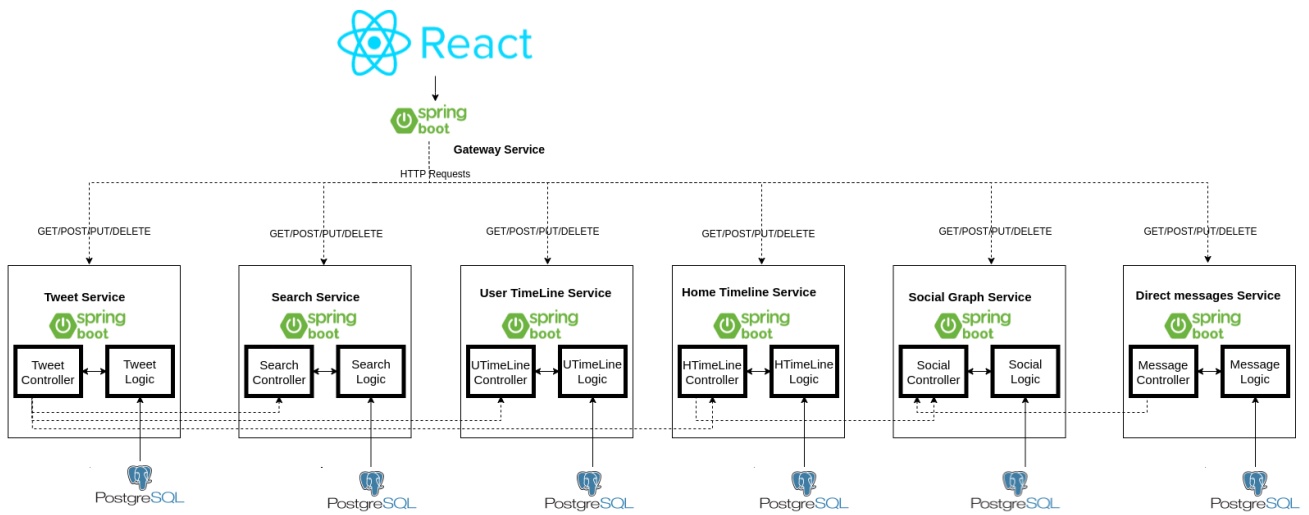
# Architecture Fonctionnelle

La partie suivante décrit les différents services de l'application séparément, ainsi que les cas d'interactions possibles entre eux.

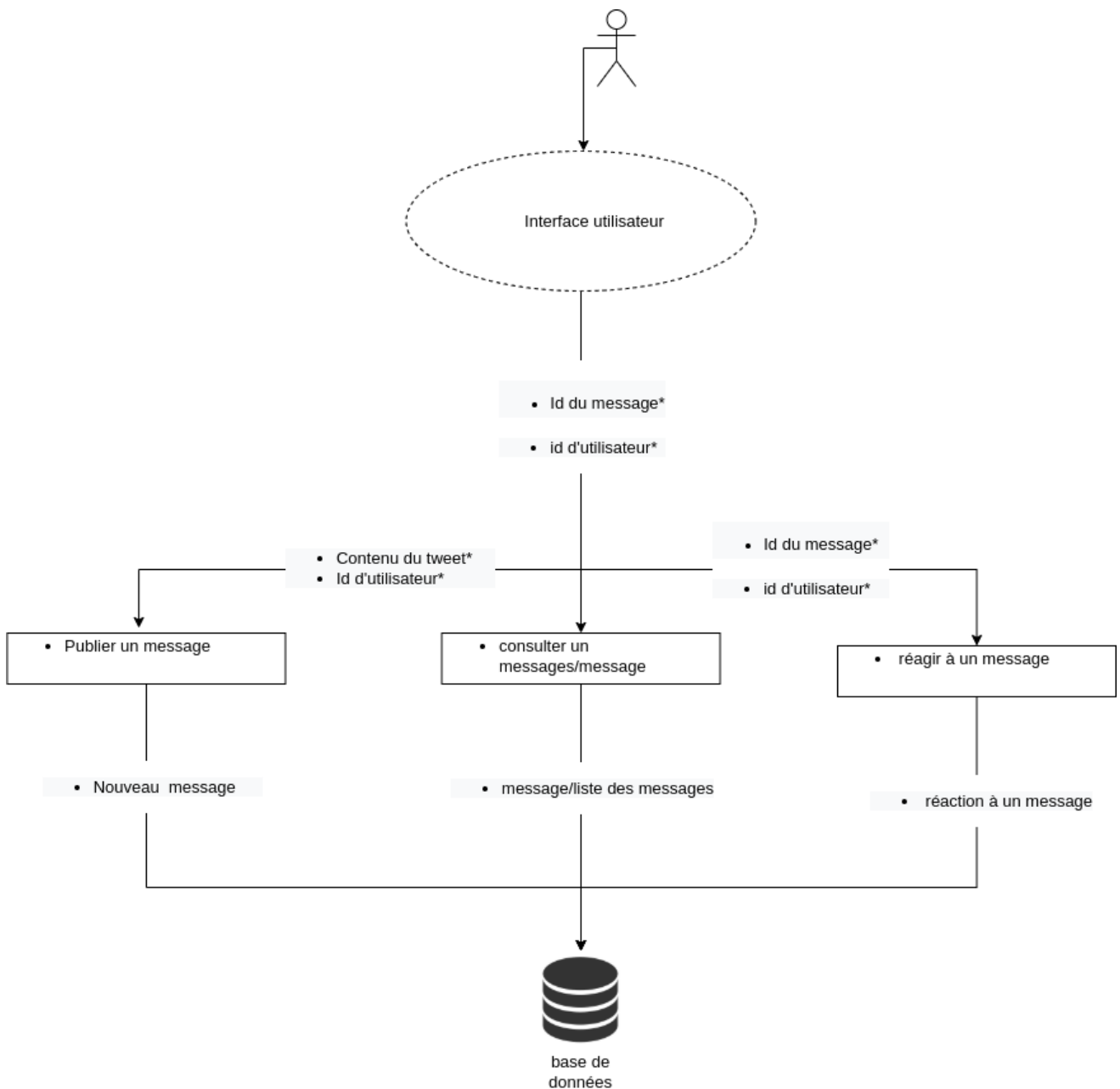
- En prenant en considération les fonctionnalités:



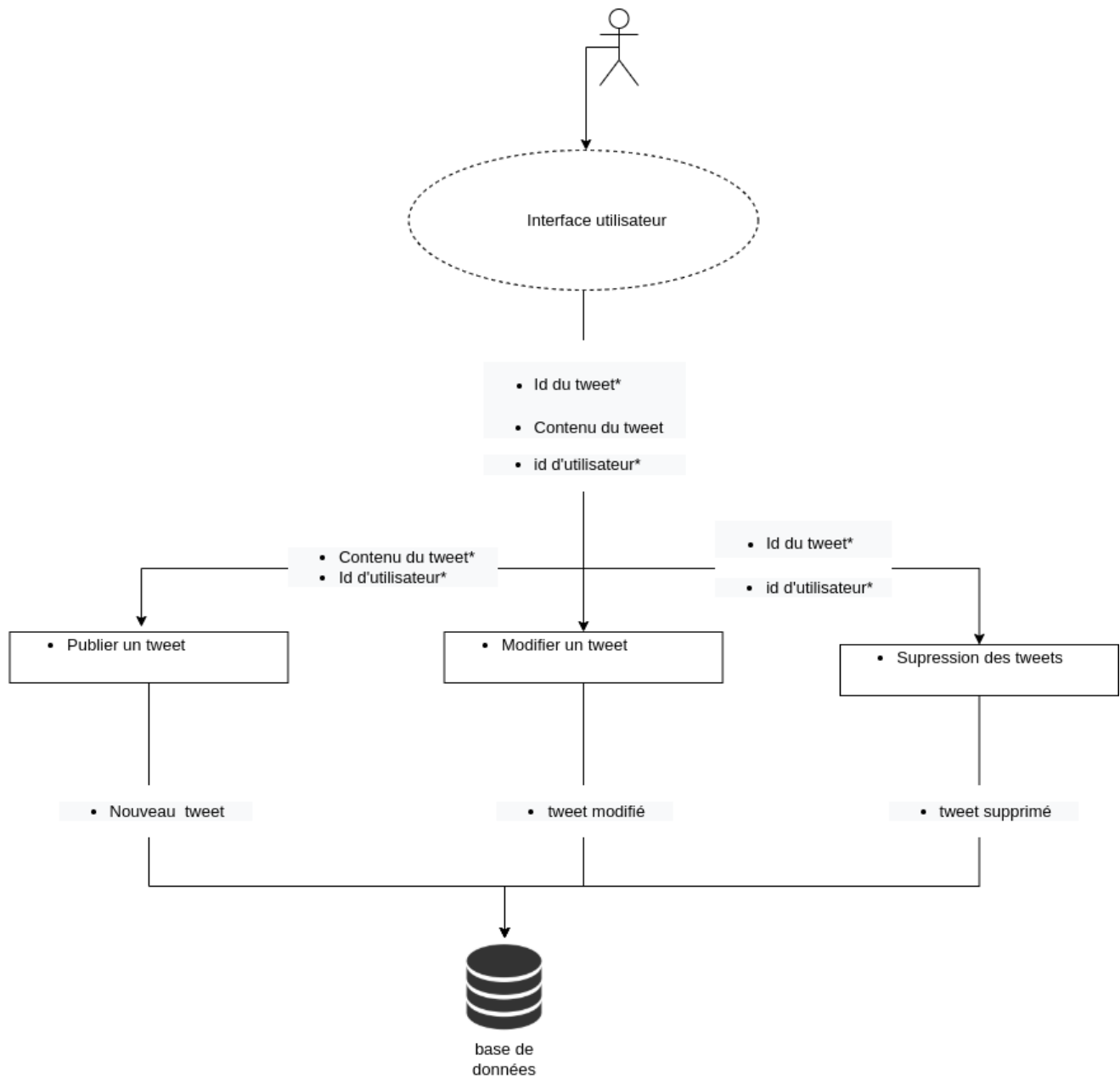
- En prenant en considération les choix techniques:



## 1) Service direct messages

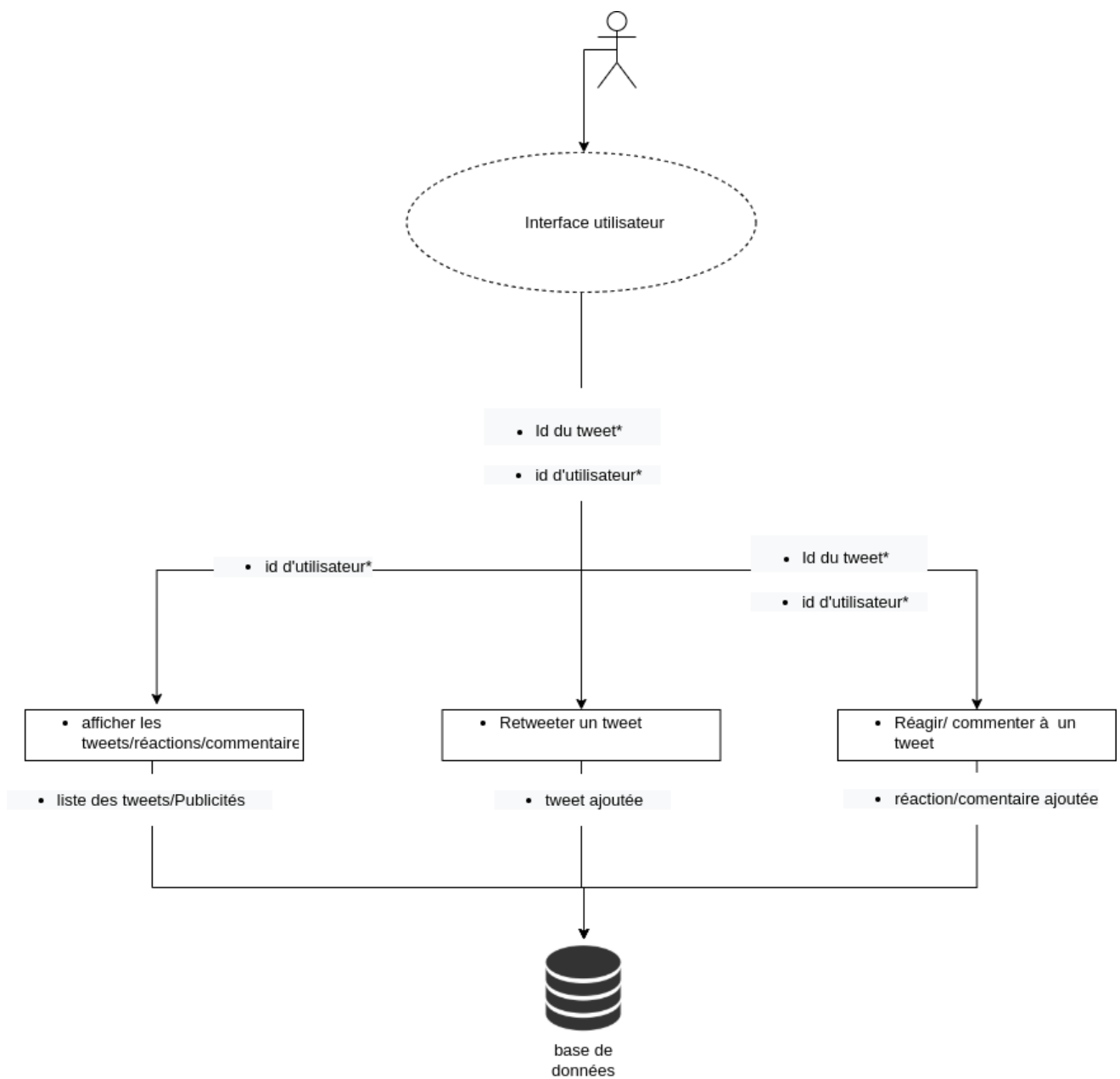


## 2) Service du Tweet

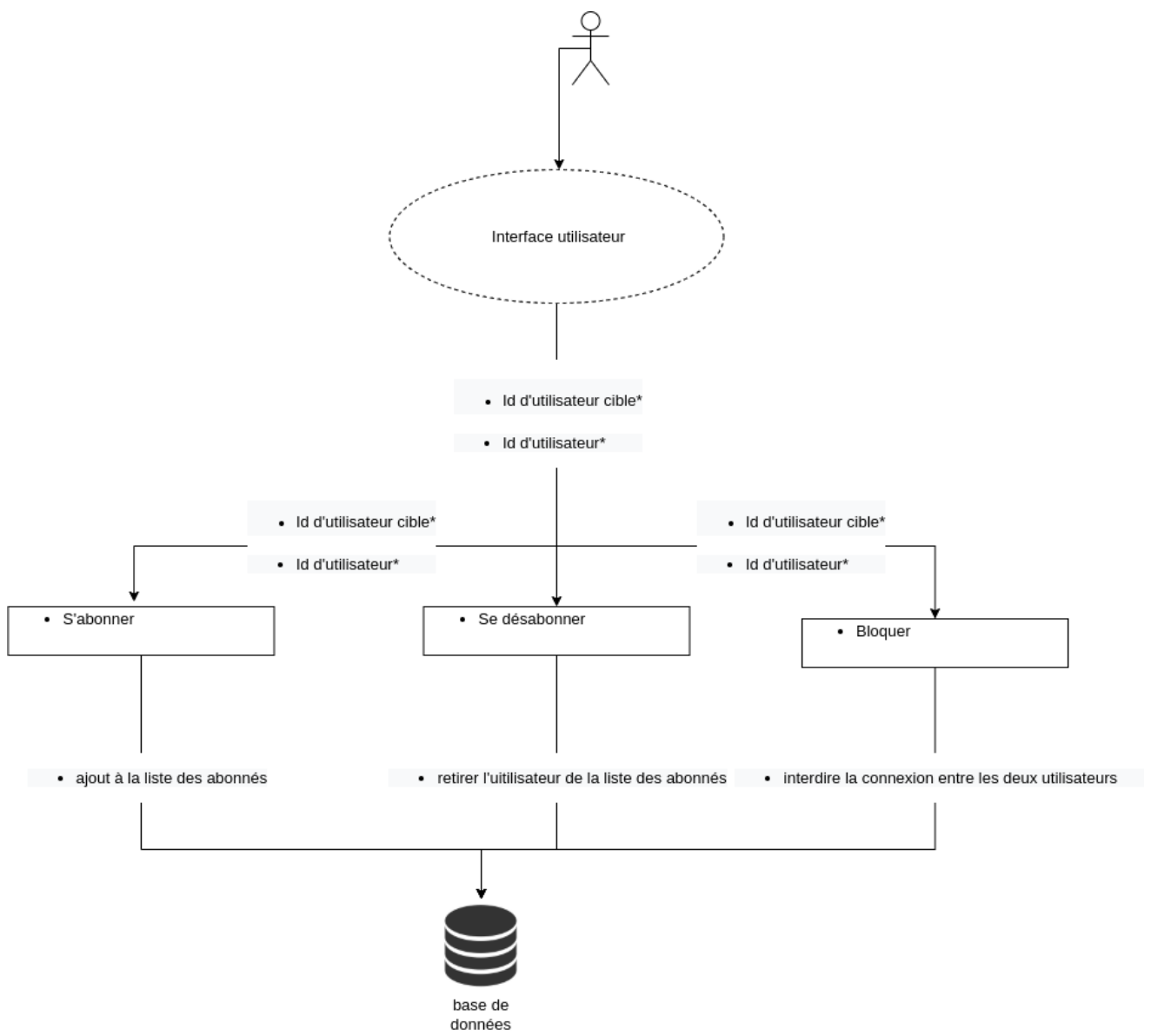


## 3) Service home timeline

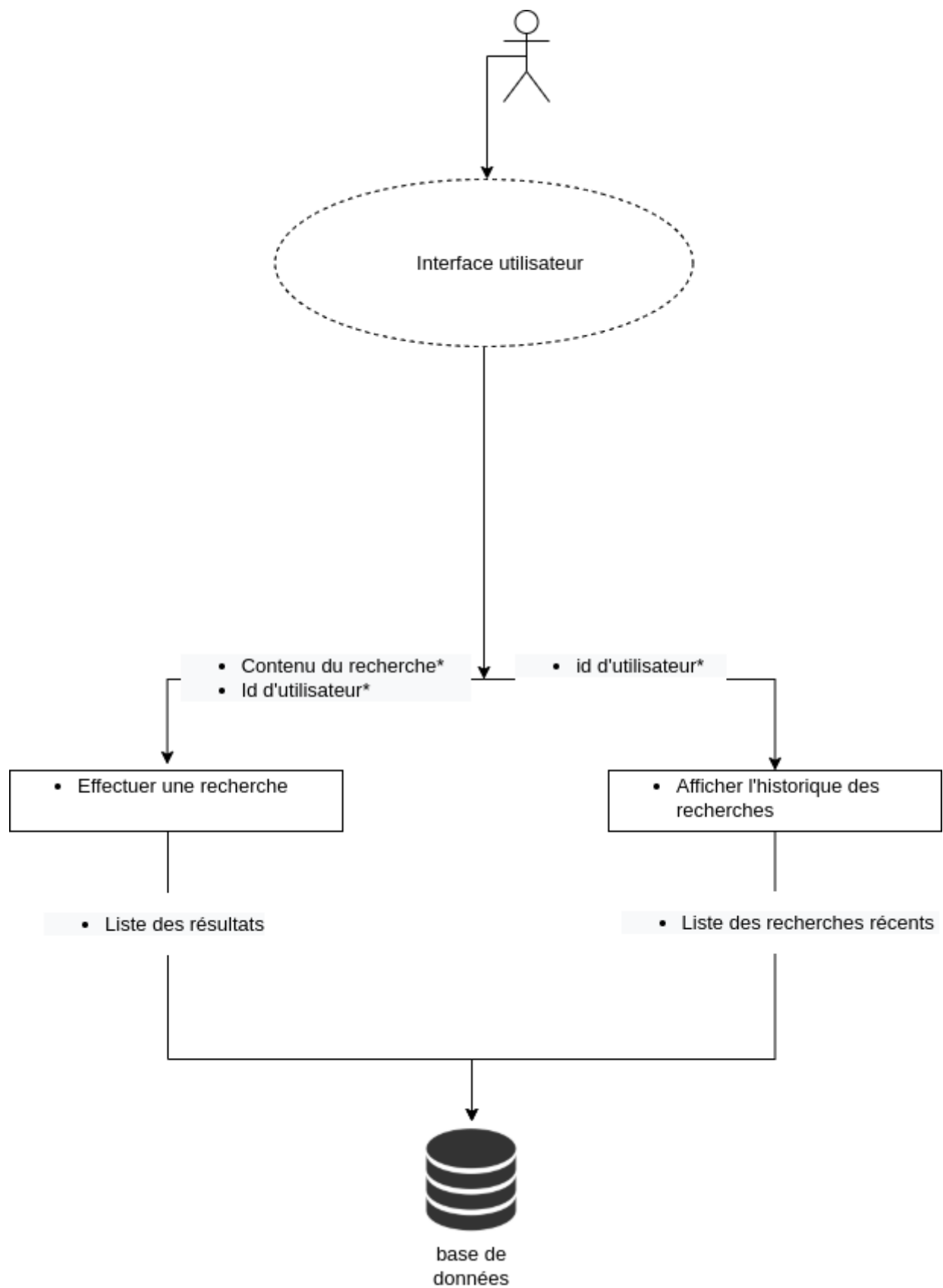




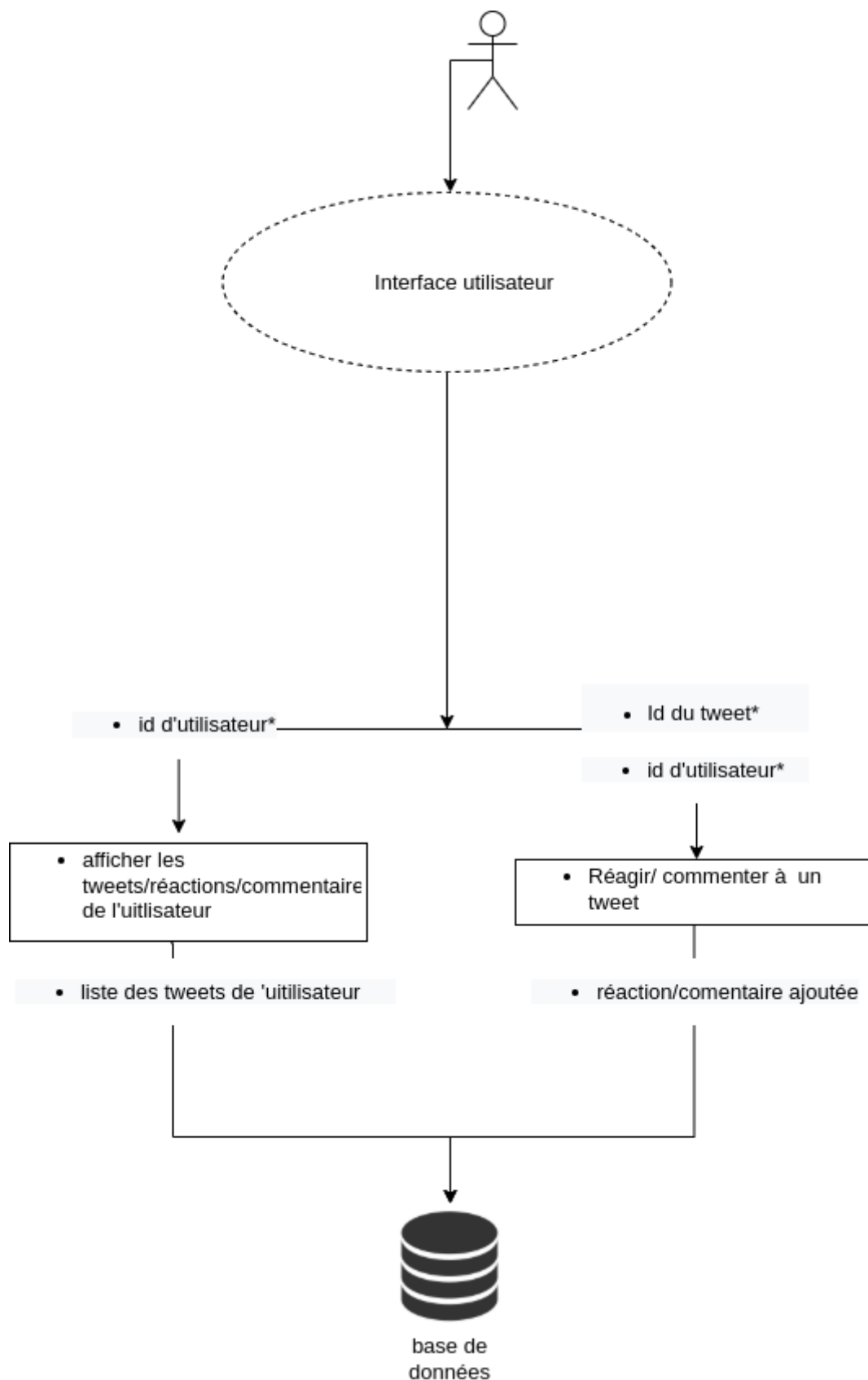
#### 4) Service Social Graph



## 5) Service search

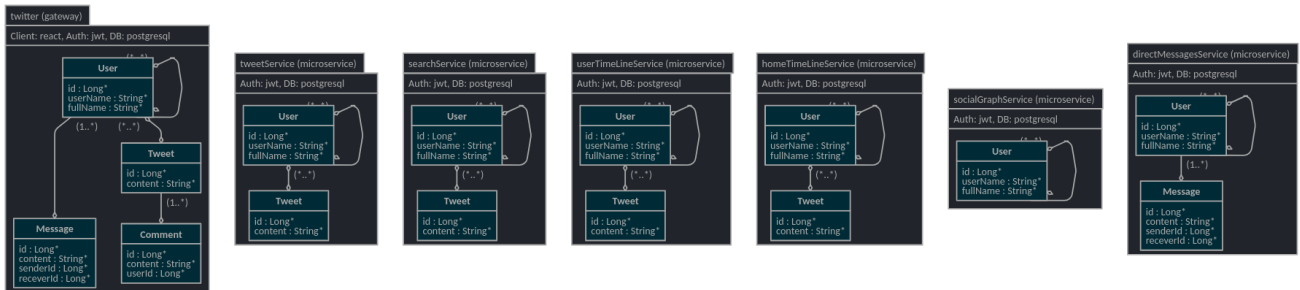


## 6) Service User Time Line

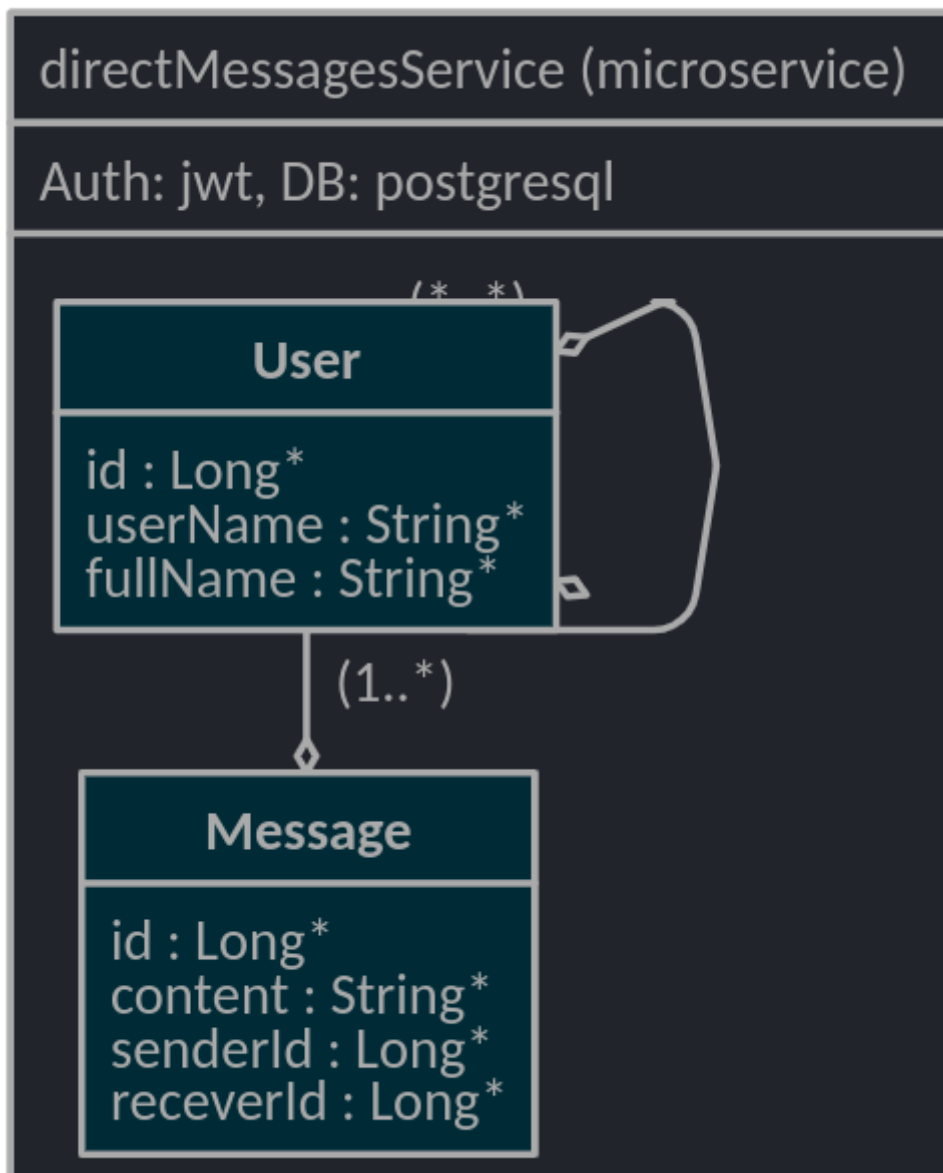


## Architecture Technique

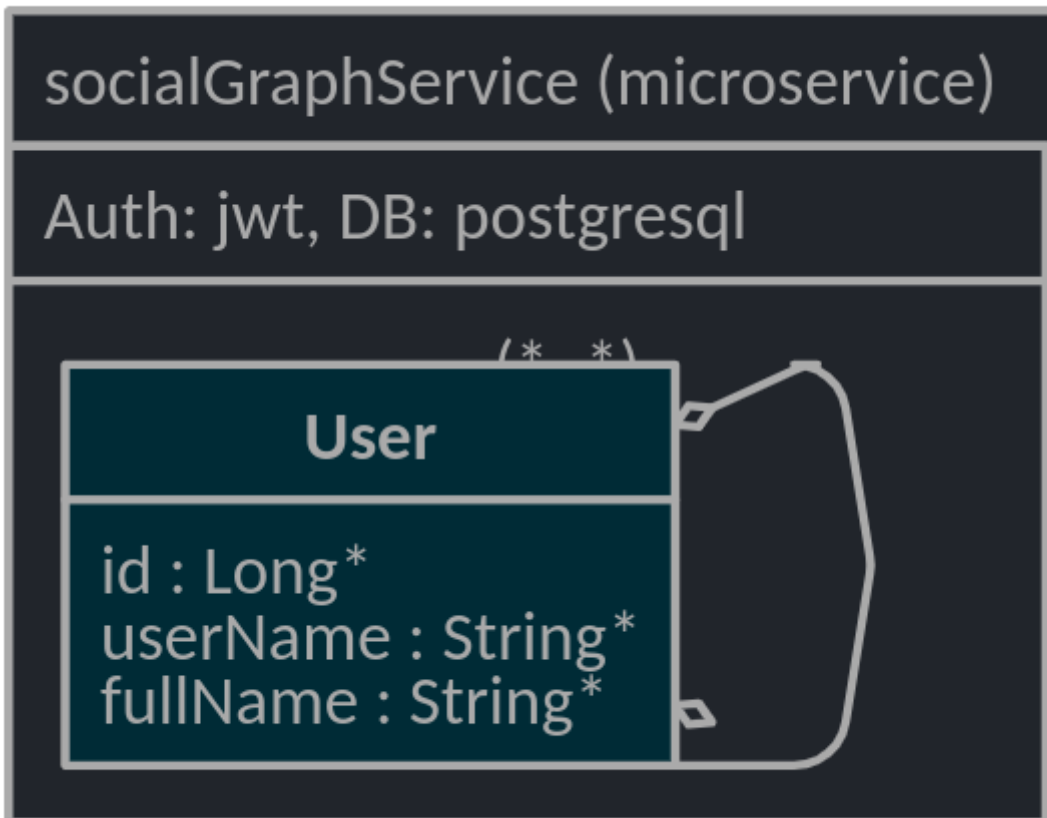
La partie suivante décrit les différents services de l'application séparément, ainsi que les cas d'interactions possibles entre les entités englobées, et les technologies utilisées dans chaque service.



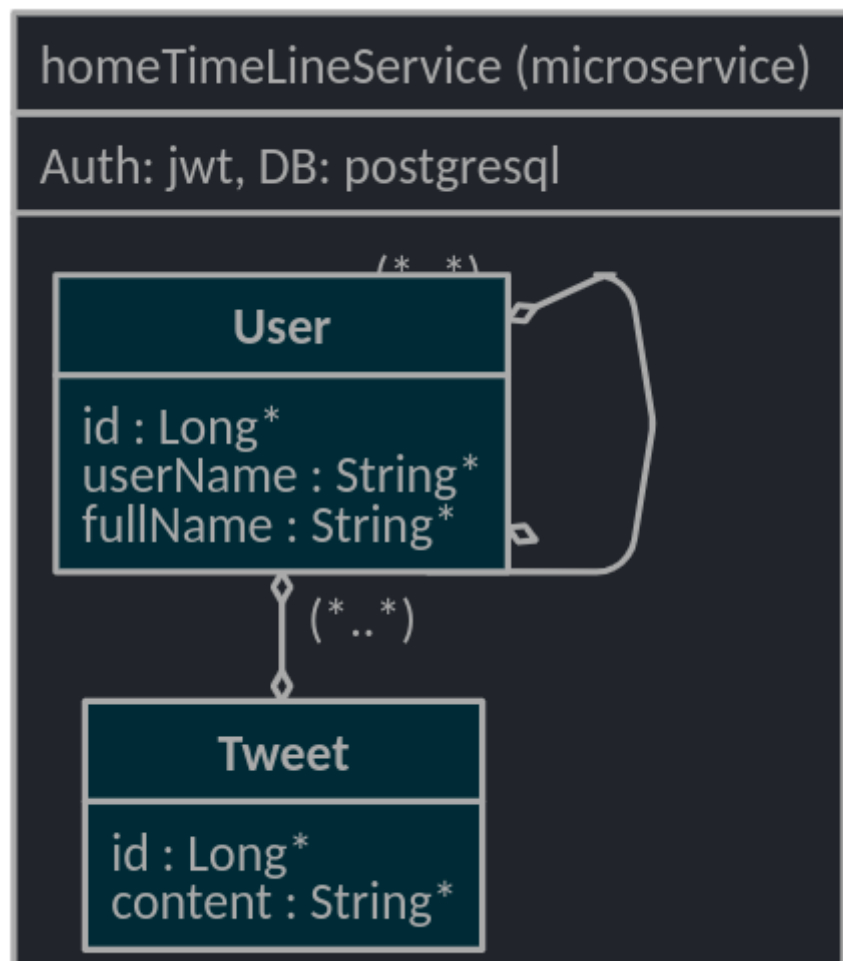
## 1) Service direct messages



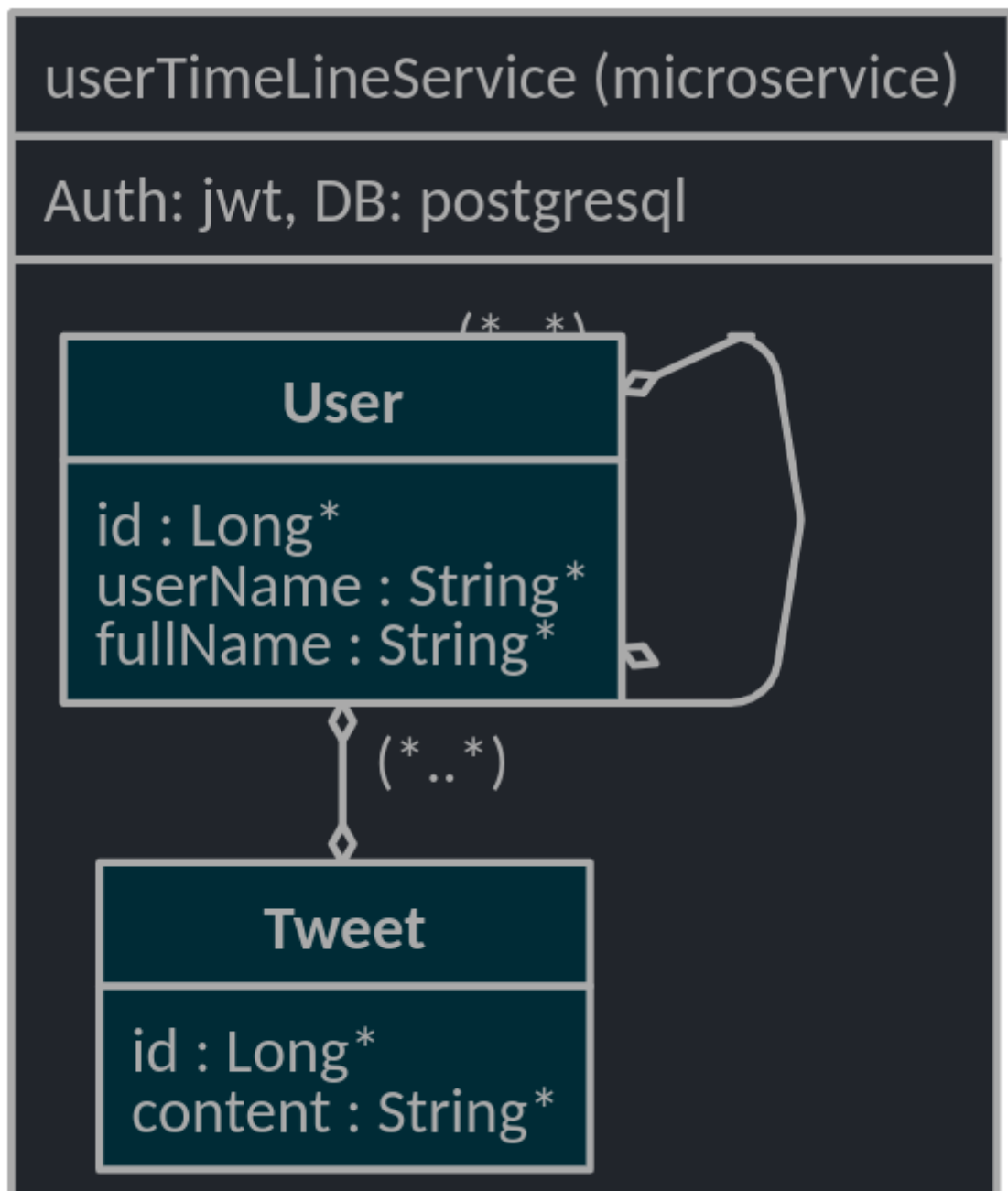
## 2) Service social graph



### 3) Service home timeline



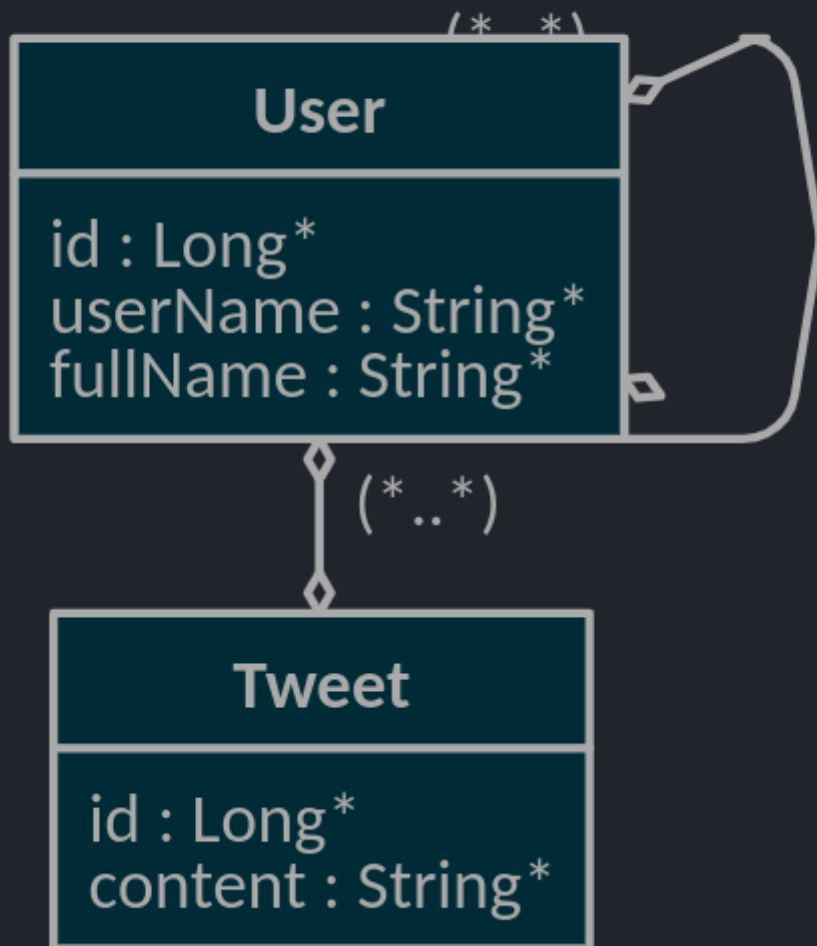
#### 4) Service user timeline



#### 5) Service search

searchService (microservice)

Auth: jwt, DB: postgresql

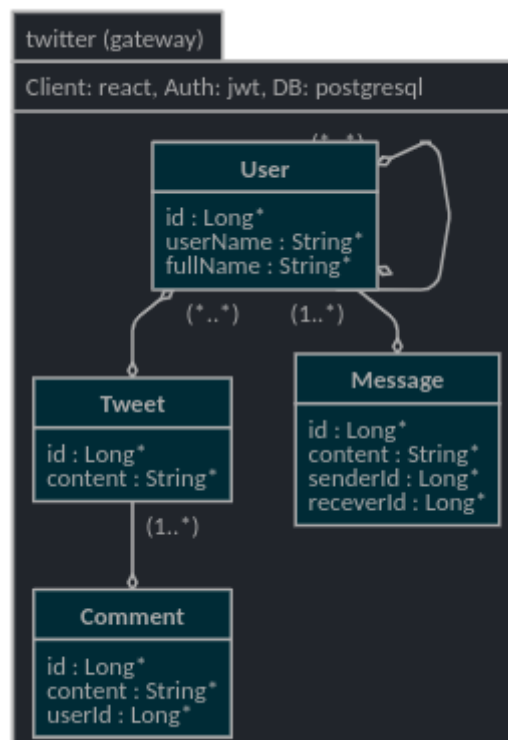


## 6) Service tweet





## 7) Service twitter gateway



# Cas d'interactions

- Description:

Numéro de cas d'interaction	Description
1	L'utilisateur consulte son propre TimeLine, qui ne contient que les Tweets qu'il a publié et leurs retweets.
2	L'utilisateur publie, supprime et modifie ses propres tweets.
3	L'utilisateur recherche des personnes, des sujets et des mots clés.
4	L'utilisateur consulte toutes les publications des personnes qu'il suit. Ça permet également d'afficher des rubriques de publicités.
5	L'utilisateur peut entamer, consulter, ou supprimer une conversation.
6	Les fonctionnalités de publication, de modification, et de suppression des tweets pour un utilisateur affectent le userTimeLine.
7	Si l'utilisateur cherche une information sur son TimeLine, le userTimeLine service serait appelé par le search service.
8	Si l'utilisateur cherche une information, le homeTimeLine service serait appelé par le search service.
9	Home timeline service aura besoin de l'ensemble des twitte des personnes follower par le user.

## Drivers fonctionnels

*Dans cette partie on va essayer de présenter en globalité l'ensemble des technologies utilisées dans ce projet. En outre, on va justifier ces choix > techniques. La justification importée pour ces choix serait conforme aux aspects suivants:*

- Performance de la technologie.
- Compatibilité de la technologie de s'intégrer avec d'autres technologies.
- Extensibilité pour des nouveaux changements.
- Disponibilité des services.

## 1) Base de données

La base de données qu'on a choisi à utiliser dans ce projet est PostgreSQL avec LiquidBase. C'est une base de données relationnelle objet, tandis que les autres bases de données telles que MySQL sont purement relationnelles. Cela signifie que PostgreSQL offre des types de données plus complexes et permet aux objets d'hériter des propriétés, mais cela rend également le travail avec PostgreSQL plus complexe. Cela se traduit par:

Aspect	Description
<b>Performance</b>	Même que PostgreSQL est très performantes et permet de gérer plusieurs types de données autres que les types traditionnelles de SQL, elle permet d'optimiser les requêtes en utilisant des outils (tel que ANALYZE) offerts par la communauté active qui la supporte.
<b>Compatibilité</b>	En utilisant Spring Data JPA et Hibernate ORM, on peut aisément connecter notre back-end à la base de données et effectuer toutes les opérations nécessaires.
<b>Extensibilité</b>	PostgreSQL est facilement extensible puisque elle peut intégrer d'autres services. À l'instar des services de métriques et de discovery. Ceci généralement est réalisable en moyennant des APIs.
<b>Disponibilité</b>	PostgreSQL permet de bien assurer l'archivage des données. En utilisant aussi LiquidBase on peut bien assurer la disponibilité des données.

## 2) Back-end

En ce qui concerne la partie back-end, on a opté à utiliser Spring. Ce dernier, est une technologie bien connue et ayant un grand support. Elle est bien connue par le nombre énorme des modules qui englobe. Cette richesse technique va permettre de réaliser beaucoup de choses sans même s'ennuyer par la configuration. Ses avantages nombreux se manifestent par:

Aspect	Description
<b>Performance</b>	Spring offre deux modèles différents de travail selon le nombre des requêtes reçues: Spring MVC et Spring WebFlux. En ce sens, la performance de Spring est indiscutable. En outre, ces différents modules sont bien puissants et capables de satisfaire leurs besoins. D'ailleurs, le support à l'architecture microservice, le rend un choix technique très intéressant.

Aspect	Description
<b>Compatibilité</b>	<code>Spring</code> est compatible avec le choix de la base de données pré-précisée. La spécification JPA qu'il intègre permet de bien gérer les données sous forme d'objet.
<b>Extensibilité</b>	L'adoption du modèle REST dans <code>Spring</code> le rend très extensible. Un point de plus qu'on peut utiliser aussi pour gérer nos APIs REST est OPEN API specification. Même l'utilisation du GraphQL ne pose plus un soucis.
<b>Disponibilité</b>	En utilisant un discovery service tel que Eureka, on peut toujours assurer l'état de nos services.

### 3) Front-end

Pour bien présenter l'information à l'utilisateur final, on choisi d'utiliser une technologies très intéressante qui'esr `React`. Elle se base sur `JSX`, et supporte `TypeScript`. En plus, sa souplesse lors du développement, permet de suivre plusieurs organisation des composants de la partie front-end. Cette partie front-end sera contenu dans le service gateway. On peut clairement voire l'importance de ce choix en:

Aspect	Description
<b>Performance</b>	<code>React</code> permet de réaliser le UI en travaillant sur plusieurs composants séparément. Ceci, permet de réutiliser les composants. En addition, il utilise le virtual dom, donc on ne va plus surcharger en travaillant sur le dom. La diversité des librairies qui entrent dans l'écosystème de <code>React</code> offre plusieurs choix pour faciliter le travail.
<b>Compatibilité</b>	<code>React</code> s'intègre facilement avec <code>Spring</code> en utilisant REST ou GraphQL.
<b>Extensibilité</b>	On peut toujours ajouter de nouvelles composants en réact ou d'autres librairies. Aussi, on peut utiliser <code>Gatsby static site generator</code> pour générer des pages web statiques et optimiser le UI.
<b>Disponibilité</b>	La disponibilité de la partie front-end ne va pas poser un problèmes si on minimise l'utilisation des CDN externes.

## 4) Déploiement et automation

Pour l'automation et le déploiement, on a choisi d'utiliser `git` avec `husky`, `JUnit` et `Jest`, `docker` et `docker-compose`, et `Github Actions`. Le process qu'on va suivre est le suivant: on utilise `docker` et `docker-compose` pour contenir l'application et exécuter les tests en `JUnit` et `Jest`. Par la suite, on utilise `git` pour le controle des versions et `husky` pour les executer les `git-hooks`. Ces derniers seront appelés par un `Github Actions Workflow`. Le résultats finals de ce workflow est un ensemble des logs permettant de vérifier que le realise de nouveau livrable était bien, et les images `docker` qui peuvent être déployées par le même workflow. On peut remarquer l'importance de ce process dans les points suivants:

Aspect	Description
<b>Performance</b>	Ces technologies sont très performantes et faciles (un tant soit peu) à implémenter. L'execution de ce process se fait dans le cloud. Il aura autant de ressource que le necessaire.
<b>Compatibilité</b>	Ces technologies s'intègrent bien sans aucun problèmes. Il suffit de bien gérer les versions des dépendances, et les ports des containers <code>docker</code> pour assurer que tout marcherait bien.
<b>Extensibilité</b>	La configuration de ce process se fait par des fichier <code>yaml</code> en générale. Ce qui permet de modifier le workflow facilement.
<b>Disponibilité</b>	Les métriques de ces process serait toujours disponibles pour les vérifier même après.

## 5) Orchestration

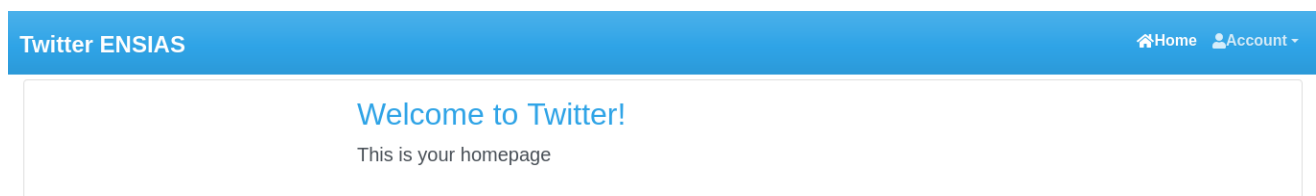
Pour l'utilisation local on peut utiliser `docker swarm` mode pour l'orchestration des noeux et les réseaux des images. Ce mode intégré en `docker` permet d'avoir un modèle minmaliste de la production. Ce modèle contient toutes les images nécessaires pour que l'application tourne. Chaque service est dans sa propre container. Le tout est dans le même réseaux. On peut avoir plusieurs répliques du même réseaux. Certes, ceci permet de tester l'application de un petit context de test. Mais, il faut penser vraiment à d'autre alternatives pour une production à grande échelle tel que `kubernetes`. Néanmoins, ce choix technique se justifie par:

Aspect	Description
<b>Performance</b>	Il permet de voir l'état des répliques et de controller leurs nombres et de redresser les noeux tombants. Tous se gère automatiquement, en se basant sur un fichier décrivant le stack très proches au fichier utiliser pour <code>docker-compose</code> .

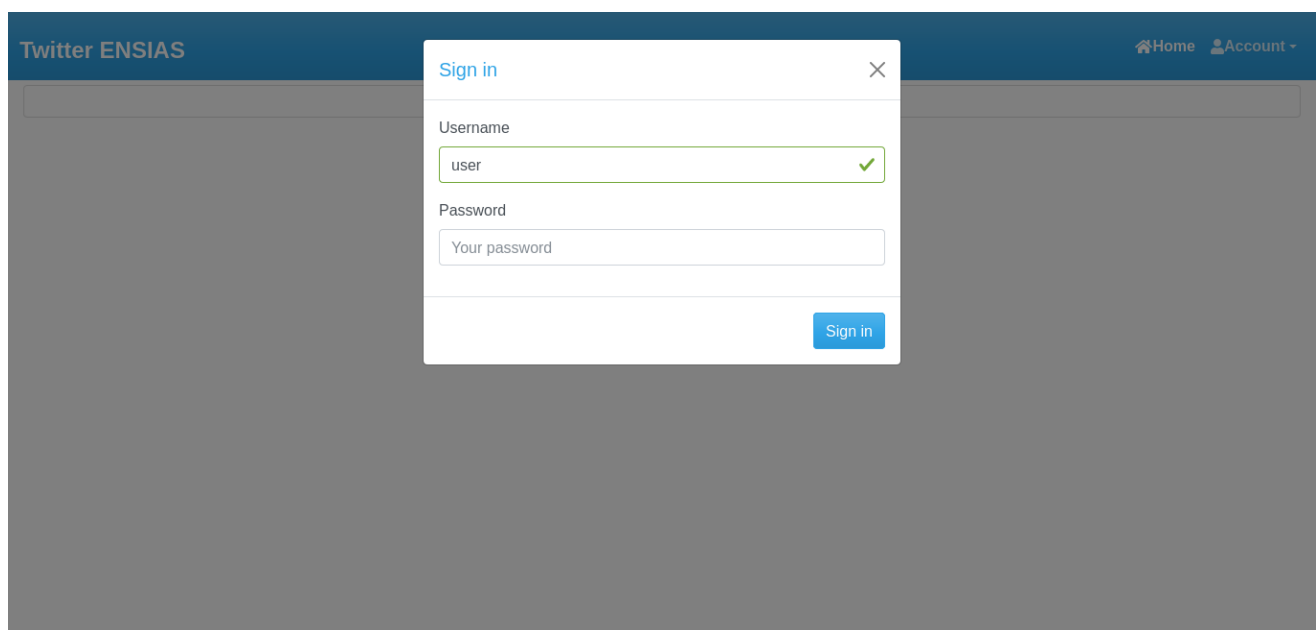
Aspect	Description
Compatibilité	Il très compatible avec docker - compose puisque leurs configurations ne se diffèrent pas beaucoup.
Extensibilité	Ce choix technique permet de bien penser les autres choix en cas de production.
Disponibilité	Ce service serait toujours disponibles et offre des logs décrivant l'état de l'application.*

## Implémentation (Prof of Concept)

### 1) Twitter home page



### 2) Sign in page



### 3) Uset TimeLine (mes tweet)

Twitter ENSIAS

[Home](#) [Actions](#) [Account](#)

Tweets

Refresh List

Create new Tweet

ID	Content	
<a href="#">1001</a>	pfijskgd	<div><div>View</div><div>Edit</div><div>Delete</div></div>

### 4) Voir un tweet

Twitter ENSIAS

[Home](#) [Actions](#) [Account](#)

Tweet

ID

1051

Content

tweet from tweeter

Back

Edit

### 5) Créer un tweet

Twitter ENSIAS

Home Actions Account

Create or edit a Tweet

Content

Back

Save

## 6) Editer un tweet

- Edition du tweet

Twitter ENSIAS

Home Actions Account

Create or edit a Tweet

ID

1051

Content

tweet from tweeter

Back

Save

## 7) Supprimer un tweet



Twitter ENSIAS

[Home](#) [Actions](#) [Account](#)

Confirm delete operation

Are you sure you want to delete this Tweet?

Cancel

Delete

Tweets

ID

Content

[1001](#)

pijskgd

View

Edit

Delete

[1051](#)

tweet from tweeter

View

Edit

Delete

Refresh List

Create new Tweet

## 8) Afficher la list des following

Twitter ENSIAS

[Home](#) [Actions](#) [Account](#)

Follows

Refresh List

Create new Follow

ID

Status

User

[1001](#)

FOLLOW

Change Status

## 9) Ajouter un follow a un user

Twitter ENSIAS

Home Actions Account

Create or edit a Follow

Status

FOLLOW

Follow

user

Back Save

## 10) Ajouter un follow a un user

Twitter ENSIAS

Home Actions Account

Create or edit a Follow

Status

FOLLOW

Follow

user

Back Save

## 11) Modifier le statut Follow/Unfollow a un user

## Create or edit a Follow

ID

1001

Status

UNFOLLOW



Follow

user

[← Back](#)[Save](#)

## 12) Consulter la liste de tous les messages.

## Messages

[Refresh List](#)[+ Create new Message](#)

ID

Message Text

[1001](#)

this is a new message with no receiver

[View](#)[Edit](#)[Delete](#)

## 13) Les services sur docker

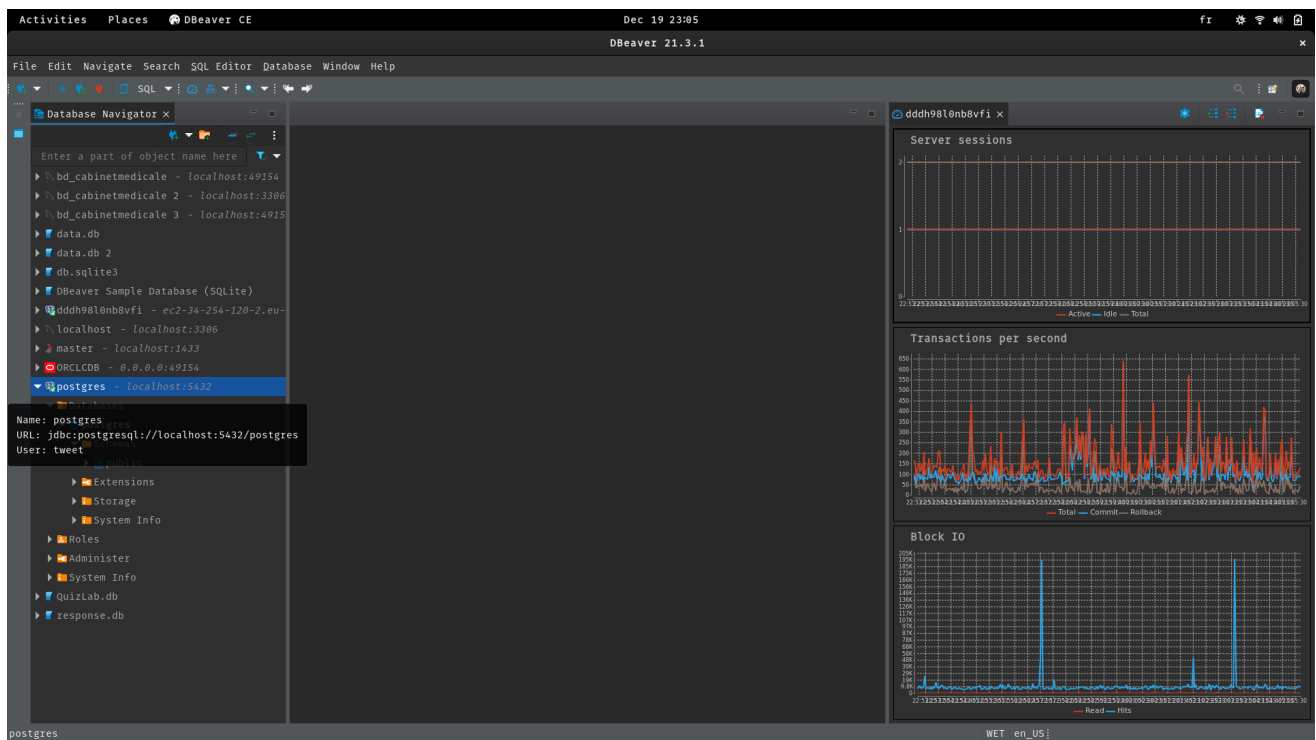
- Après l'exécution du `docker-compose up`

```
docker-compose_gatewayservice-postgresql_1 is up-to-date
docker-compose_tweetsservice_1 is up-to-date
docker-compose_usertimelineservice_1 is up-to-date
docker-compose_messagingsservice_1 is up-to-date
Recreating 6076f7f7c772_docker-compose_gatewayservice_1 ...
docker-compose_usertimelineservice-postgresql_1 is up-to-date
docker-compose_messagingsservice-postgresql_1 is up-to-date
docker-compose_searchservice_1 is up-to-date
docker-compose_socialgraph-postgresql_1 is up-to-date
docker-compose_jhipster-registry_1 is up-to-date
docker-compose_socialgraph_1 is up-to-date
docker-compose_tweetsservice-postgresql_1 is up-to-date
docker-compose_hometimelineservice_1 is up-to-date
Recreating 6076f7f7c772_docker-compose_gatewayservice_1 ... done
docker-compose_hometimelineservice-postgresql_1 is up-to-date
```

- Après l'exécution du `docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bcedb5094224	kotbymo/gateway	"bash -c /entrypoint..."	3 hours ago	Up 3 hours	0.0.0.0:8080->8060/tcp, ::8080->8060/tcp	docker-compose_gateway
yservice_1	jhipster/jhipster-registry:v7.1.0	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	0.0.0.0:8761->8761/tcp, ::8761->8761/tcp	docker-compose_jhipst
er-registry_1	kotbymo/hometimelineservice	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	8066/tcp	docker-compose_hometi
3fe007891b88	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_messag
melineservice_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_social
74f0330f011a	postgres:13.5	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	8065/tcp	docker-compose_messag
ingreservice-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_gateway
bc6d3698ec9e	postgres:13.5	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	8062/tcp	docker-compose_tweets
graph-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_tweets
7e08d58b5f54	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_userti
ingreservice_1	postgres:13.5	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	8061/tcp	docker-compose_userti
f0f6d6c701c9	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_search
yservice-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_tweets
263a2cb03694	postgres:13.5	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	8064/tcp	docker-compose_userti
ervice_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	8063/tcp	docker-compose_social
09c09088d783	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
melineservice_1	postgres:13.5	"bash -c /entrypoint..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
65121be6ad1a	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
service-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
bf80a3a14803	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
ervice-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
8b2326062355	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
melineservice-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
1b96c2b313f0	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
lbservice-searchservice_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
c69031075f0d	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
graph_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
93b9a1bbbead	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi
melineservice-postgres_1	postgres:13.5	"docker-entrypoint.s..."	18 hours ago	Up 3 hours	5432/tcp	docker-compose_hometi

## 14) Base de données



## 15) Swagger APIs

