



Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes – RABAT

Rapport Projet ALBC : Proof of Concept - Twitter Architecture

Réalisée par :

El Mehdi OUDAOUD
Fatima Ezzahra LAHNINE
Ikrame EL ARFAOUI
Asmaa EL AZHAR

Encadré par :

Abdelkarim REMLI
Bouchra EL ASRI

1- Stratégie d'Entreprise & Vision

1. Vision :

Ce qu'on voit diffère selon nos expériences et notre savoir. Ce qu'on va présenter n'est que notre vision pour une application similaire à twitter :

Un espace où les communautés de toutes tailles coexistent, discutent et partagent leurs points de vue. Un espace spécial où chacun peut trouver une communauté bienveillante là où il se sent connecté avec autrui.

Notre but est de stimuler la créativité et le sentiment d'appartenance et distinction ainsi que permettre aux communautés de s'ouvrir facilement au public ou de rester inclusives.

2. Stratégie, challenges et drivers :

Challenges :

- La concurrence est féroce vu qu'il a des réseaux sociaux géants qui monopolise le marché (Facebook, Tiktok, Redit...) donc notre produit doit se distinguer pour pouvoir survivre.
- Une stratégie marketing doit être mis en place ainsi qu'une infrastructure assez robuste pour pouvoir supporter le réseau mais aussi d'être abordable avec un petit budget
- Les Challenges technique comporte :
 - Optimisation des routes selon la zone géographique
 - Management des fichiers partagés.
 - Nombre de fichiers et comment faciliter, optimiser et sécuriser l'accès.
 - Style personnalisé sans faille de sécurité.
 - Contrôle sur les publications et le statut de la communauté.
 - Classement des postes (Trending)
 - Lien vers différents services (3ier)
 - Publicités créatives

Drivers :

Les défis à surmonter inspire les drivers, Voici ce les axes conducteurs qui pour nous doivent être accentué :

- 1- La sécurité : pour pouvoir créer un espace customisable nous devons être sûr que ce ne soit pas exploitable par des parties externes via XSS ou autre. Comme ça tout le monde peut se sentir en sécurité et peuvent laisser leurs esprits créatifs s'épanouir.

- 2- Disponibilité : une application SM (social media) frustrant à ouvrir et toujours en panne est une application que personne n'utilise, la preuve est Facebook qui a perdu des milliards de dollars juste parce que ses services ont tombé pendant 6 heures.
- 3- Scalabilité : Notre solution doit être capable d'évoluer en taille rapidement selon les exigences du marché.
- 4- Performance (VIP) : Nos abonnés, et les comptes qui gèrent les communautés doivent avoir le meilleur service possible (les artistes avec la meilleure qualité d'image, les streamers avec la meilleure connectivité et qualité de Stream possible ...)

Stratégie :

Promouvoir l'application auprès des communautés qui servent à des services tels que discord pour la communication et Reddit pour les liens et notre application twitter pour une expérience personnalisée pour la communauté.

Adapter notre interface et notre backend aux objectifs de la communauté (comme Stackoverflow pour les développeurs, DeviantArts avec des arts non fragmentés pour les artistes, des audios de haute qualité avec des lecteurs vidéo/audio pour les musiciens ...).

Pour la monétisation, nous faisons en sorte que nos expériences personnalisées soient payées par le créateur ou ses sponsors et nous restreignons l'utilisation des publicités pour une expérience plus saine.

2- Cahier des charges et étude de l'existant

Afin d'obtenir une architecture solution convenable à notre problème, on doit tout d'abord spécifier les fonctionnalités clés ainsi que les exigences via une étude de l'application existante Twitter.

N.B: tous les services présentés en bas requiert une authentification.

1. Tweet Service :



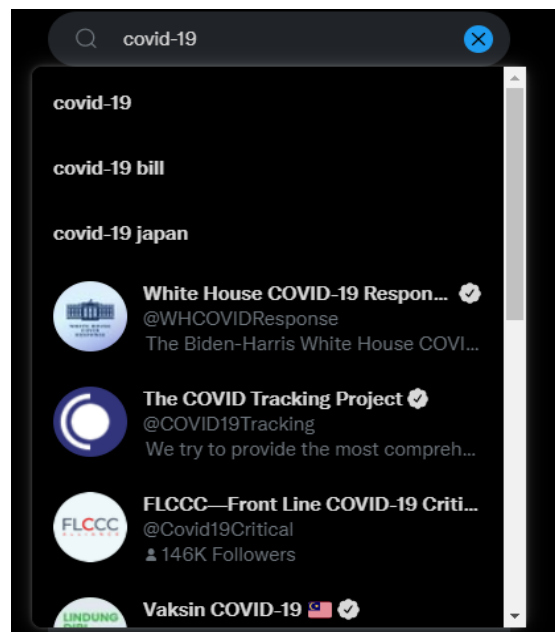
- L'utilisateur a la possibilité de faire un tweet qui peut inclure en plus du texte du tweet une photo ou une vidéo
- L'utilisateur peut réagir aux tweets des autres par des commentaires, des Likes, des partages, ou des retweets.
- Chaque tweet a un nombre de commentaires, de likes et de retweets.

Pour créer un nouveau tweet on doit entrer comme Input :

- Le texte du tweet
- les fichiers médias à joindre
- Date et heure actuels
- Les hashtags et les personnes/communauté à inclure dans le tweet

Comme Output on aura un message de sauvegarde à partir du backend

2. Search service :



Le search service nous aide à faire de la recherche de telle sorte que:

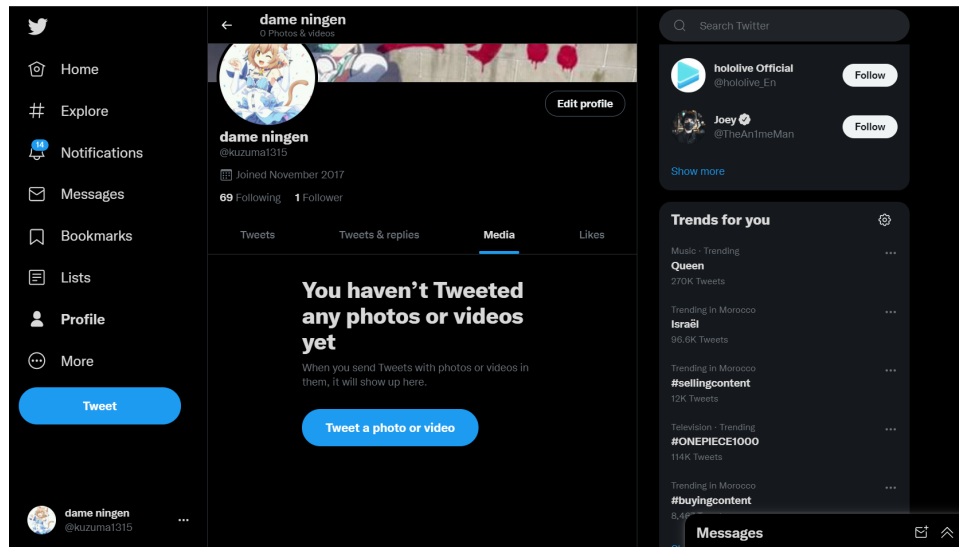
- Les utilisateurs peuvent chercher des tweets en fonction de mots-clés ou hashtags relatifs à un sujet.
- Les résultats affichés comportent à la fois des Tweets, des photos, des personnes
- Les résultats peuvent être filtrés mais pas en grande quantité.

Une autre fonctionnalité clé du search service est de capturer l'influence des tweets, personnes... pour déterminer si il peuvent être considéré comme 'trending' ou en tendance via un score qui fluctue selon les recherches et les retweets.

On peut lister les inputs et outputs du Search service comme suit :

- Input :
 - o l'expression clé à rechercher
 - o les paramètres de filtre
- Output : une liste filtré ordonné selon le niveau trending, cette liste contient des
 - o Utilisateurs
 - o Communauté
 - o Tweets
 - o Hashtags

3. User timeline service :



Une timeline est une représentation linéaire d'événements et de tweet auquel un utilisateur (personne ou communauté) a créé, retweeté ou réagit.

La principale chose à savoir sur les timelines de Twitter est que vous pouvez interagir avec chaque message en cliquant dessus. Il se développe pour vous montrer tout média associé, comme une vidéo ou une photo, les personnes qui ont répondu ou retweeté le message, ou d'autres conversations pertinentes liées à ce tweet.

- Renvoyer à l'utilisateur sa timeline qui contient tous les tweets de l'utilisateur dans l'ordre décroissant du temps, soit personnelle ou bien d'un autre utilisateur.
- Lorsqu'un utilisateur publie un tweet, insérer le tweet en tête de la liste de tweet de la timeline de l'utilisateur.
- Les tweets sont stockés dans l'ordre décroissant de l'heure de création.

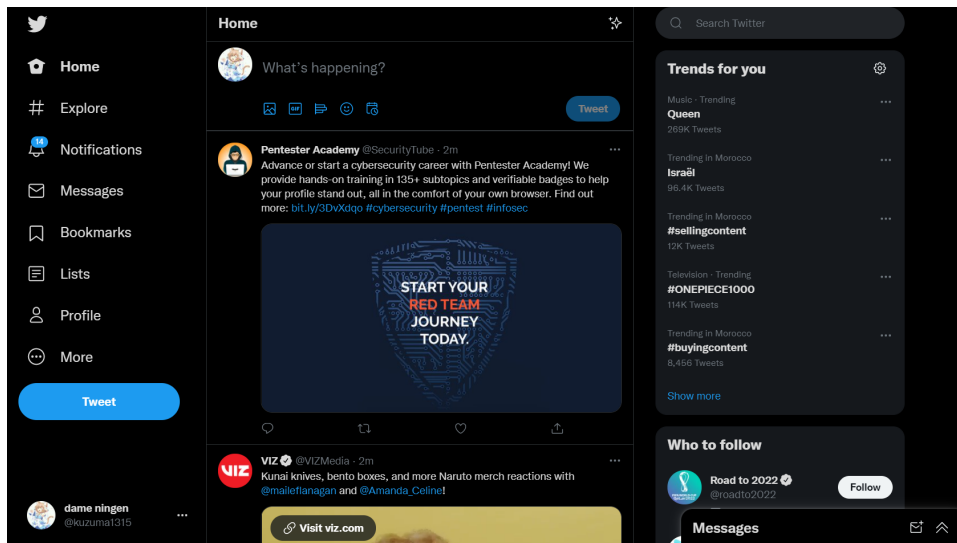
Pour accéder à la timeline d'utilisateur qui contient ses derniers tweets, on doit avoir comme input :

- Input : des informations permettant l'identification de l'utilisateur ainsi que les filtres à appliquer sur les tweets.

On reçoit comme output :

- Output : la page de la timeline

4. Home timeline service :



- Afficher un flux de Tweets provenant des comptes choisis à suivre sur Twitter.
- Voir des suggestions de contenu basées sur divers signaux.
- Répondre, retweeter ou aimer un tweet depuis la page d'accueil.
- Choisir d'afficher d'abord les Tweets les plus importants dans Accueil ou les Tweets les plus récents.
- Voir des Tweets pour les sujets suivis, des Tweets pour les suggestions de sujets, et un résumé des Tweets les plus intéressants qui peuvent ne pas être consultés, intitulé "Au cas où vous l'auriez manqué".
- En cliquant ou en tapant n'importe où sur un Tweet dans la timeline, la page de détails du Tweet s'affiche, ce qui permet de voir des photos, des vidéos et d'autres informations liées à ce Tweet.
- La fonction Favoris pour mettre en favoris les Tweets qui peuvent être consultés ultérieurement.
- Afficher les suggestions des sujets qui peuvent intéresser l'utilisateur dans la rubrique "Tendances pour vous", et les comptes similaires à ceux qu'il avait suivis dans "Qui suivre".

Les inputs et outputs de la Home timeline service sont donc :

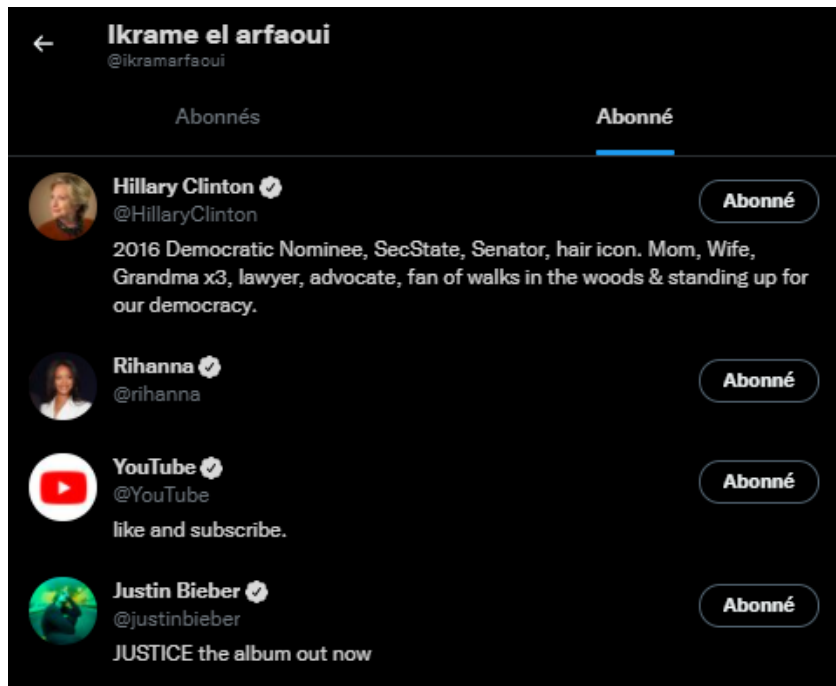
- Input : aucune.
- Output : ID, texte, date et heure de création, nombre de likes, les commentaires et le nombre de partage de tous les tweets les plus récents des utilisateurs auquel l'utilisateur actuel est abonné.

5. Social graph service :

Le service de graphes sociaux garde une trace des utilisateurs qui suivent quels utilisateurs.

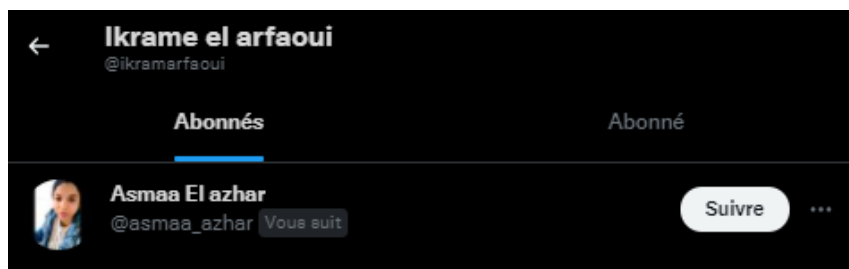
Ainsi, un utilisateur doit avoir accès à :

- La liste de ses abonnements.



Il peut se désabonner d'un utilisateur ou visiter son profil en cliquant sur son nom.

- La liste des abonnées



Il peut de même suivre l'un de ses abonnées, se désabonner d'eux, ou visiter leurs profils en cliquant sur le nom.

Pour réaliser cette fonctionnalité, on doit avoir comme input :

- Input :aucune.

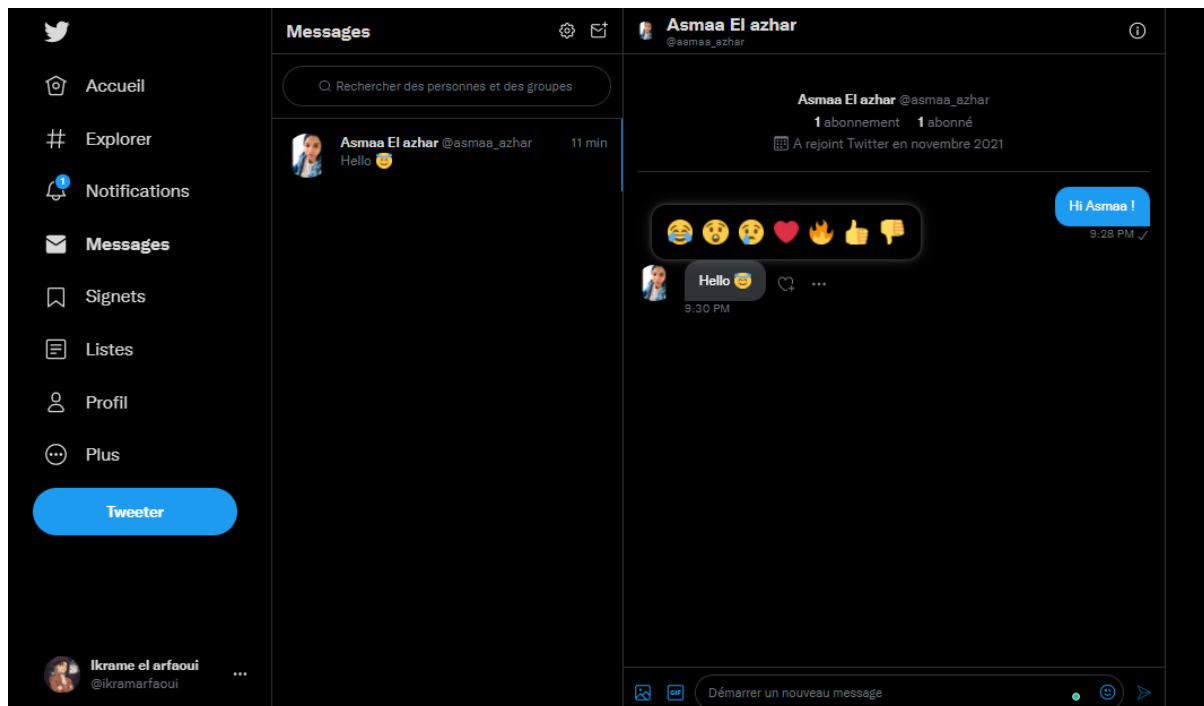
Les résultats obtenus sont les suivant :

- Output :

- Les noms, identifiant et bio de tous les utilisateurs auxquels l'utilisateur actuel est abonné.
- Les noms, identifiant et bio de tous les utilisateurs qui suivent l'utilisateur actuel.

6. Direct messages service :

C'est le service qui permet à un utilisateur d'envoyer et recevoir des messages entre les utilisateurs qui se suivent mutuellement.



Un utilisateur peut :

- Explorer la liste des utilisateurs avec lesquels il discute.
- Ecrire un nouveau message.
- Réagir à un message.
- Supprimer/signaler/copier un message.
- Insérer des médias, des émojis et des GIF.
- Envoyer un message à un groupe d'utilisateurs.

Des points additionnels à notre Architecture Solution :

- Possibilité de sauvegarder les données côté serveur à l'aide d'un middleware de compression qui examine les paramètres de la communauté et argumente ou fragmente les fichiers et choisit de les placer dans des pôles de haute priorité ou de priorité moyenne.
- Couche transparente entre le public et les communautés où les créateurs peuvent faire de la publicité pour eux-mêmes via des posts.

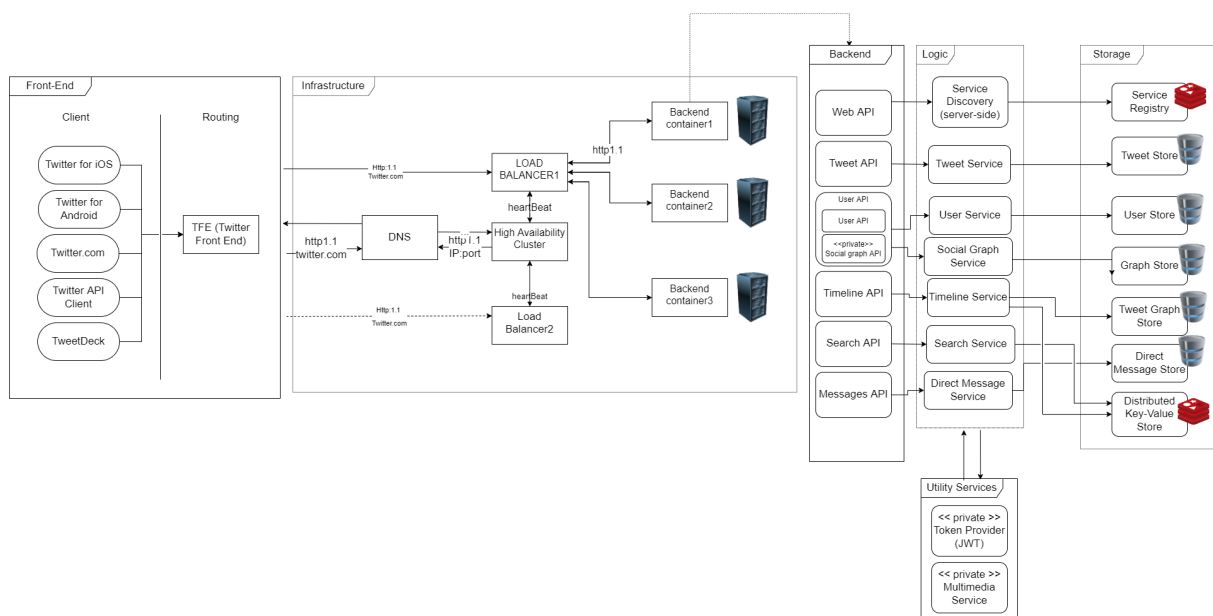
- Refaire le mécanisme de partage où vous n'indiquez pas votre nom à côté du message mais permettez que le message soit montré à votre cercle.
- Une couche front-end personnalisable par les créateurs en fonction du profil de la communauté qu'ils souhaitent (comme Wix, etc.).

Ainsi pour être conforme à ces fonctionnalités, on suivra le format input output suivant :

- Input :
 - id à partir de la session.
 - Id de l'utilisateur avec lequel on envoie le message
 - Texte du message
 - Date et heure actuels
 - Fichiers média à joindre
- Output :
 - Un message d'envoi du backend.

3- Architecture

Voici notre architecture solution (abstraite) :



Cette architecture de Twitter comporte cinq couches principale qu'on va déborder en plus de détails :

La couche de présentation, infrastructure, Backend, une couche logique métier, et enfin le stockage.

La couche de présentation contient la partie frontale des différentes interfaces de nos clients. Le routage des tweets est assuré par un service nommé TFE pour "twitter Front End".

La couche infrastructure contient un load balancer qui assure la disponibilité de notre application en assurant sa duplication sur plusieurs container backend. Et puis un High Availability cluster qui rend le load balancer accessible à tout moment.

Au sein de la logique métier, des services assurent la constitution des timelines, la gestion du social graph, les messages directs, etc. Par exemple, lorsqu'un utilisateur écrit un simple tweet, c'est un ensemble d'appels qui est déclenché. Le tweet passe par le tweet API qui demande le "Tweet Service".

La couche Storage assure le stockage des données de notre système.

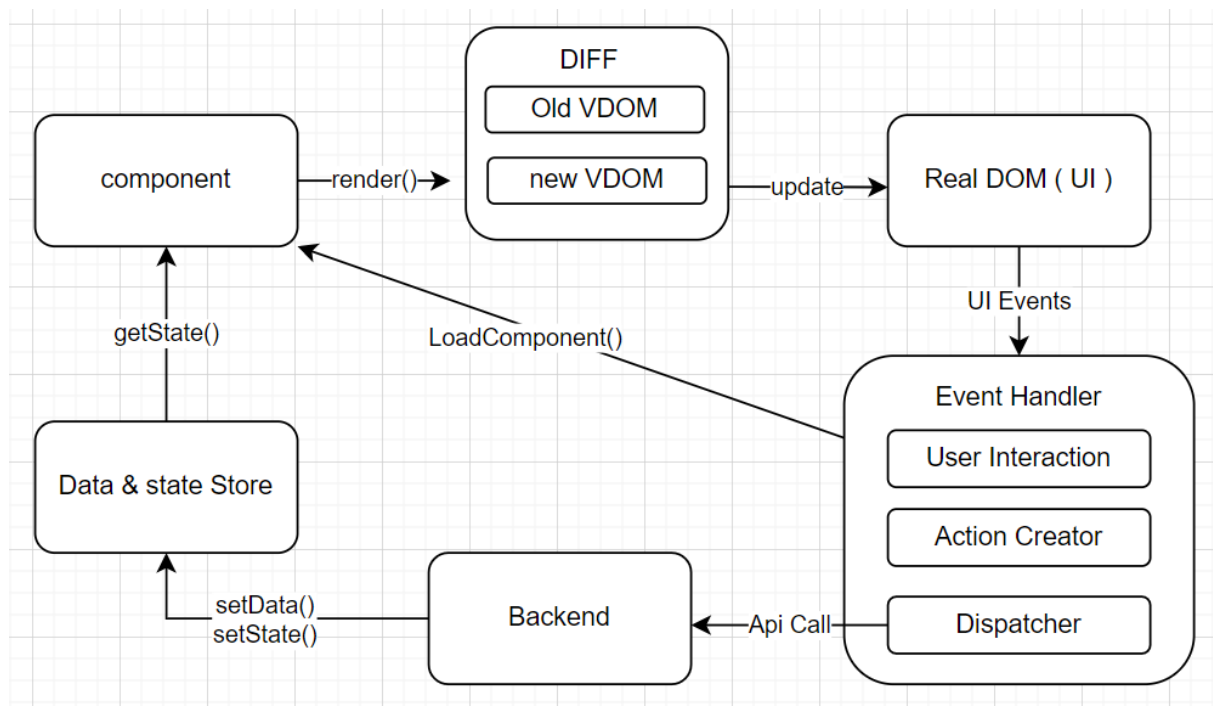
Finalement, on a la partie utility services : qui contiens les services de support que la couche logique et backend utilisé pour gérer les opérations les plus répétitive plus précisément l'authentification et l'upload de fichiers. Le Token Provider (JWT) qui permet l'échange sécurisé des tokens qui assure l'intégrité et l'authenticité des données, et le multimédia service qui permet le stockage et la gestion de tous les fichiers circulés dans notre application.

4- Description des composants de l'architecture

NB: l'utilisation de liens API est juste une simplification pour donner des exemples concrets qui épanouissent la description des composants dans la partie architecture abstraite comparé à ce qui a été vu dans la partie analyse du produit. On va élaboré encore plus sur la partie API sur la partie architecture technique

4.1. Le frontend

La partie frontend de notre système aura comme Inputs les différents interactions de l'utilisateur avec l'interface graphique. Cependant ceci doit être organisé de telle sorte que la maintenance et la mise à jour de l'interface soit rapide et intuitive pour les développeurs , afin de répondre aux demandes des clients ressenties sur le product backlog via des users stories qu'on extraction depuis leurs requêtes au support. Donc on va adopter une architecture basée sur des composants sur tous nos devices. Avec quelques différences de style.



pour les interfaces de type terminal (notamment Twitter API Client) l'architecture n'est pas intéressante car ce n'ai qu' une fenêtre qui traduit des commands vers des appel à l'API

On rappel que les Applications Twitter sont:

- Twitter.com : version web de l'interface. devices: ALL
- Twitter for IOS: version IOS de l'interface. devices: IOS suit
- Twitter for android : version Android de l'interface. devices: Android
- Twitter API Client : version Terminal de l'interface pour tester et automatiser les appels à l'API.
- TwitterDeck : une version pour les utilisateurs VIP qui possède une interface customisable et du chargement en realtime des données lié au tweets et plein d'autre features

4.2. les API

4.2.1. Tweet API

La création d'un nouveau tweet se fait par une requête POST en faisant appel à l'API suivante : `/api/v1/tweets/`

Dans le corps de la requête on aura :

- user mentions: { @Said, @Trump, ... } qui seront des identifiants des utilisateurs qu'on veut mentionner
- Les Hashtags : { #freedom, #tech}
- Le timestamp : le date de création

- La Destination du tweet : données sur l'utilisateur qui a créer le tweet

Dans les Headers on aura :

- Le User token : pour identifier l'utilisateur
- Le Refresh token

Comme output , on aura une réponse HTTP qui indique le succès/échec de la création du tweet.

La suppression d'un tweet se fera par une requête DELETE en faisant appel à l'API suivante : `/api/v1/tweets/{t_id}`

input au niveau URL :

t_id : tweet identifier

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse HTTP qui indique le succès/échec de la suppression du tweet.

L'affichage d'un tweet specific se fera par une requête GET en faisant appel à l'API suivante : `/api/v1/tweets/{t_id}`

input au niveau URL :

t_id : tweet identifier

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse HTTP qui contient :

- user mentions: { @Said, @Trump, ... } qui seront des identifiants des utilisateurs qu'on veut mentionner
- Les Hashtags : { #freedom, #tech}
- Le timestamp : la date de création
- modification timestamp : la date de la dernière modification
- La Destination du tweet : données sur l'utilisateur qui a créer le tweet

L'affichage de tous les tweets se fera par une requête GET en faisant appel à l'API suivante : `/api/v1/tweets/`

Dans le corps de la requête HTTP on aura un ensemble de critère de filtrage :

- Les Hashtags : { #albc, #GL, }
- La Timeline Destination
- un Compteur : pour choisir la limite des tweets qu'on veut recevoir

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON sous la forme:

```
{ "data" : [ tweet1 ....] }
```

chaque tweet ayant cette structure :

- user mentions: { @Said, @Trump, ... } qui seront des identifiants des utilisateurs qu'on veut mentionner
- Les Hashtags : { #freedom, #tech }
- Le timestamp : la date de création
- modification timestamp : la date de la dernière modification
- Interaction : {numéro de likes , retweets ... }
- La Destination du tweet : données sur l'utilisateur qui a créer le tweet

L'interaction avec les tweets se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/tweets/{t_id}/interact`

input au niveau URL :

t_id : tweet identifier

Dans le corps de la requête HTTP on aura une description de l'interaction :

- Type : (like, retweet , comment , share : via DM, copy link, bookmark)

Dans les Headers on aura :

- Le User token
- Le Refresh token

4.2.2. User API

La création d'un nouvel utilisateur se fait par une requête POST en faisant appel à l'API suivante : `/api/v1/user/`

Dans le corps de la requête on aura :

- Le nom d'utilisateur
- Le mot de passe

- L'adresse email
- Le numéro de téléphone

Comme output , on aura une réponse JSON qui indique le succès/échec de la création de l'utilisateur.

La validation d'un nouvel utilisateur se fait par une requête POST en faisant appel à l'API suivante : `/api/v1/user/ validate`

Dans le corps de la requête on aura :

- Le code de validation de l'adresse email
- Le code de validation du numéro de téléphone

Comme output , on aura une réponse JSON qui indique le succès/échec de la validation des coordonnées de l'utilisateur.

La modification du profile de l'utilisateur courant se fait par une requête PUT en faisant appel à l'API suivante : `/api/v1/user/ :u_id`

Dans le corps de la requête on aura :

- Le nom complet
- Le numéro de téléphone
- Bio : la description qui s'affiche lorsqu'on accède au user timeline
- L'image : le nom du fichier sous la forme filename.ext
- Les préférences de l'utilisateur { art, music, news ...}

Dans les Headers on aura :

- L'image : { profil.png}
- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec de la modification du profil de l'utilisateur.

Le blocage d'un utilisateur se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/user/block/{user_id}`

input au niveau URL :

user_id : target user identifier

Dans les Headers on aura :

- Le User token

- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec du blocage de l'utilisateur. Cette réponse se propage vers le social graph API via l'appel à l'url : `/api/v1/sociograph/block?source={source_id}&target={target_id}`

L'abonnement à un utilisateur se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/user/{user_id}/sub`

Input au niveau URL :

user_id : target user identifier

Comme output , on aura une réponse JSON qui indique le succès/échec du blocage de l'utilisateur.

Dans les Headers on aura :

- L'utilisateur courant : { user id d'après la session courante }
- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec du blocage de l'utilisateur. Cette réponse se propage vers le social graph API via l'appel à l'url : `/api/v1/sociograph/subscribe?source={source_id}&target={target_id}`

Le désabonnement d'un utilisateur à partir de la liste des abonnements se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/user/{user_id}/unsub`

Input au niveau URL :

user_id : target user identifier

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec de la suppression de l'utilisateur. Cette réponse se propage vers le social graph API via l'appel à l'url : `/api/v1/sociograph/unsub?source={source_id}&target={target_id}`

Le joint d'une communauté fermé se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/user/{com_id}/join`

Input au niveau URL :

com_id: target community identifier

selon les demandes de la communauté un utilisateur doit parfois répondre à plusieurs question

Dans le corps de la requête (format JSON) on aura :

- Réponses { "question 1":réponse 1 ,...}

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec du rejoint de la communauté. Cette réponse se propage vers le social graph API via l'appel à l'url : `/api/v1/sociograph/join?source={source_id}&target={com_id}`

L'abandon d'une communauté se fera par une requête POST en faisant appel à l'API suivante : `/api/v1/user/{com_id}/leave`

Input au niveau URL :

com_id: target community identifier

Dans le corps de la requête (format JSON) on aura :

- Raison : un paragraphe sur lequel l'utilisateur décrit pourquoi il abandonne une communauté.

Dans les Headers on aura :

- Le User token
- Le Refresh token

Comme output , on aura une réponse JSON qui indique le succès/échec d'abandon de la communauté. Cette réponse se propage vers le social graph API via l'appel à l'url : `/api/v1/sociograph/leave?source={source_id}&target={com_id}`

4.2.3. Timeline API

La récupération des Tweets et Retweets les plus récents publiés par l'utilisateur qui s'authentifie et les utilisateurs qu'il suit. Cela se fait par une requête GET en faisant appel à l'API suivante : `/api/v1/statues/{user_id}`

Input au niveau URL :

user_id: target community or user identifier

Dans le corps de la requête (format JSON) on aura :

- un Compteur : pour choisir la limite des tweets à récupérer

Dans les Headers on aura :

- Le User token
- Le Refresh token

L'output sera sous la forme JSON suivante :

```
{ "data" : [ tweet1 ....] }
```

Dans le timeline, on distingue entre le timeline d'un utilisateur spécifique et le timeline d'accueil. Cette différence figuré au sein de la requête GET est exprimée comme suit :

- */api/v1/statuses/user_timeline : pour avoir les tweet propres à un utilisateur spécifié.*
- */api/v1/statuses/home_timeline : pour avoir la collection des tweets les plus récents soit de l'utilisateur ou ses abonnements.*

4.2.4. Search API

La collecte des éléments recherchés ou trouver d'autres utilisateurs se fait par une requête GET en faisant appel à l'API suivante : */api/v1/search/{keyword}*

Input au niveau URL :

keyword: target topic or user name

Dans le corps de la requête (format JSON) on aura :

- query : contient le mot clé de la recherche

Dans les Headers on aura :

- L'autorisation token

L'output sera sous la forme JSON suivante :

```
{ "data" : [ { "id" : , "text": }, ....] }
```

4.2.5. Messages API

L'envoi d'un message personnel à un utilisateur spécifique se fait par une requête POST en faisant appel à l'API suivante :

`/api/v1/direct_messages/events/new(message_create)`

La publication d'un nouvel événement `message_create` donne lieu à un message direct envoyé à un utilisateur spécifié par l'utilisateur authentifié. Un événement est renvoyé en cas de succès. La publication de messages directs avec option de réponse rapide et de pièces jointes multimédia est prise en charge.

La réception des messages personnels se fait par une requête GET en faisant appel à l'API suivante : `/api/v1/direct_messages/events/list`

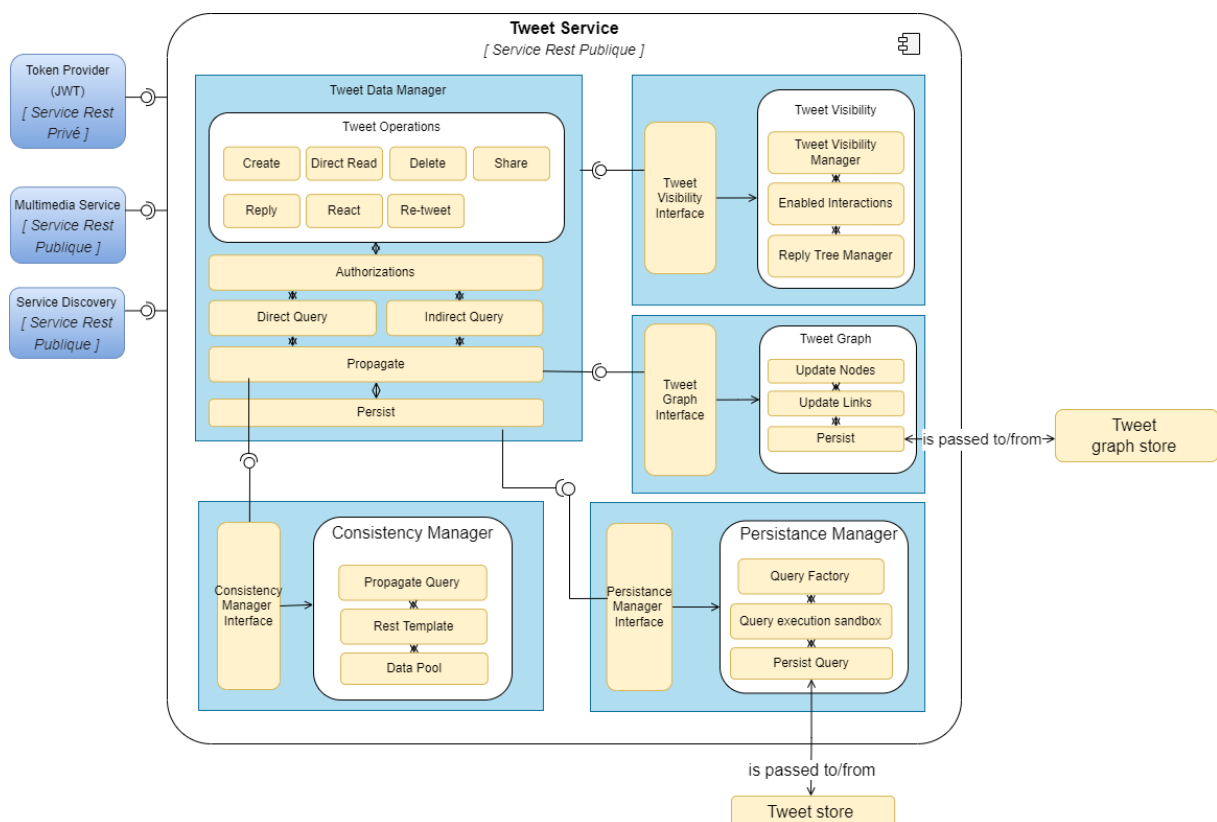
L'output sera sous la forme JSON suivante :

```
{ "event" : { "type" : , "message_create" : , "message_data" : }, .... }
```

4.3. Les Services (Logique métier)

4.3.1 Tweet Service

4.3.1.1 Architecture Solution:

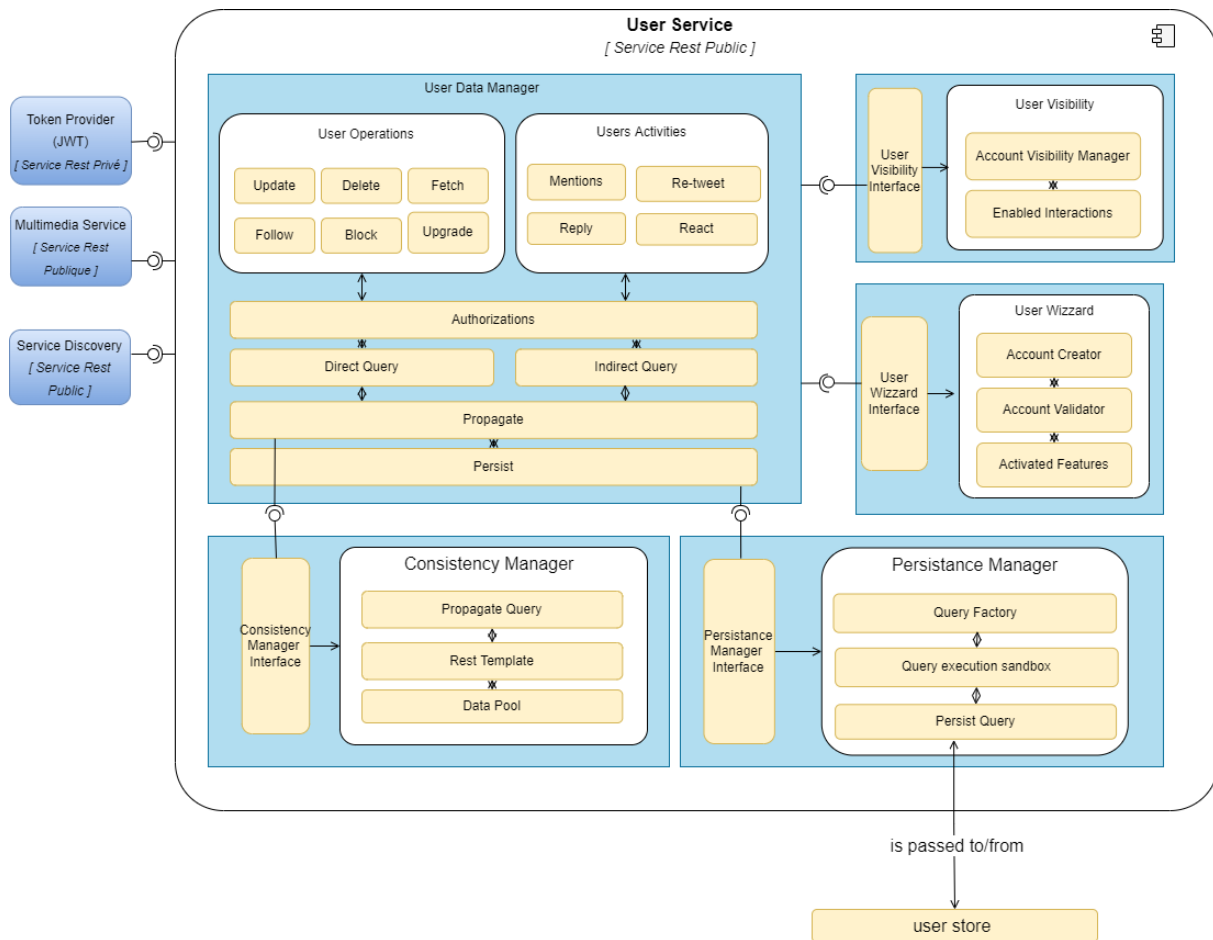


4.3.1.2 Description des composants:

- Tweet operations : ce composant contient les opérations crud : create, read et delete. Il ne contient pas l'opération update, car un tweet publié ne peut pas être modifié. Puis, on a les opérations de repartage, commenter, réagir et retweet.
- les autorisations: ils contrôlent l'accès des utilisateurs aux opérations sur les tweets.
- Direct Query : c'est l'ensemble des requêtes qu'on envoie directement à notre base de données.
- Indirect Query : ce sont des requêtes qui s'exécutent de sources multiples comme la mémoire cache, Il est fortement recommandé lorsque la taille des données n'est pas très grande et que les données ne changent pas continuellement. Ainsi, on profite de la haute performance du moteur de requête.
- Propagate : permet de propager un traitement vers d'autres composants en faisant appel au consistency manager.
- Persist : permet le stockage et la persistance des données.
- tweet visibility : ils contrôlent l'accès des utilisateurs aux tweets publiés. Par exemple : un utilisateur peut désactiver les commentaires d'un tweet.
- Tweet graph : il passe / récupère les données du tweet graph store pour gérer l'ensemble des retweets et des partages.
- Persistance manager: il communique avec le tweet store pour garantir la persistance des tweets.
- Consistency manager : il propage les traitements vers les autres composants. Par exemple : supprimer un utilisateur implique la suppression de tous les tweets qu'il a publié.

4.3.2 User Service

4.3.2.1 Architecture Solution:

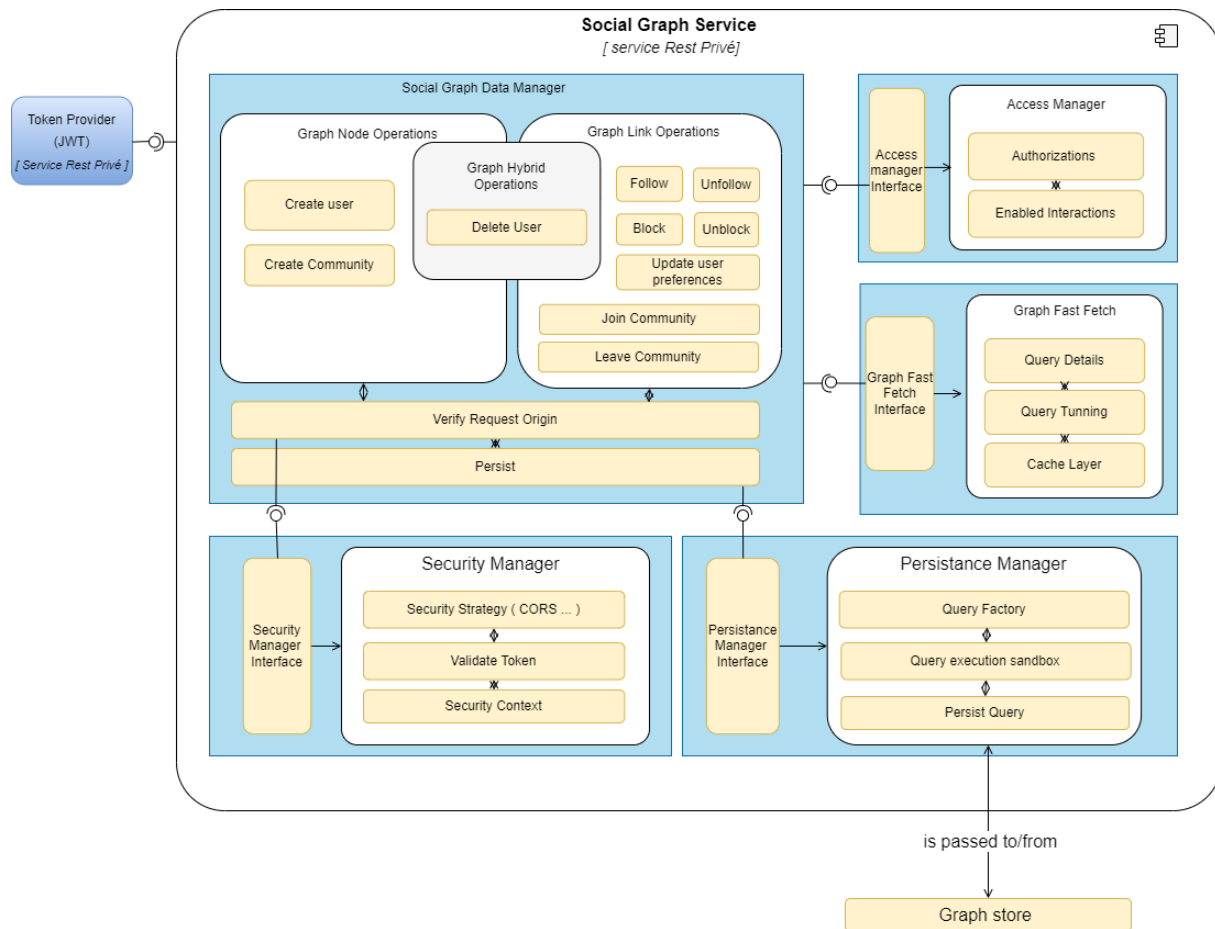


4.3.2.2 Description des composants:

- **User operations** : ce composant contient les opérations crud : update et delete. Puis, Fetch pour trouver un utilisateur, Follow pour abonner à un utilisateur, Block pour bloquer et Upgrade pour rendre un utilisateur premium.
- **User Activities** : il contient les activités qu'un utilisateur peut effectuer : mentionner un utilisateur, faire un retweet, commenter et réagir.
- **User visibility** : Contrôle la visibilité d'un utilisateur pour les autres utilisateurs.
- **User Wizzard** : c'est le processus de création d'un nouvel utilisateur, qui contient les différentes étapes de validation.
- **Persistence manager**: il communique avec le user store pour garantir la persistance des données des utilisateurs.
- **Consistency manager** : il propage les traitements vers les autres composants. Par exemple : supprimer un utilisateur implique la suppression de toutes les données de cet utilisateur(images, posts, ...).

4.3.3 Social Graph Service

4.3.3.1 Architecture Solution:

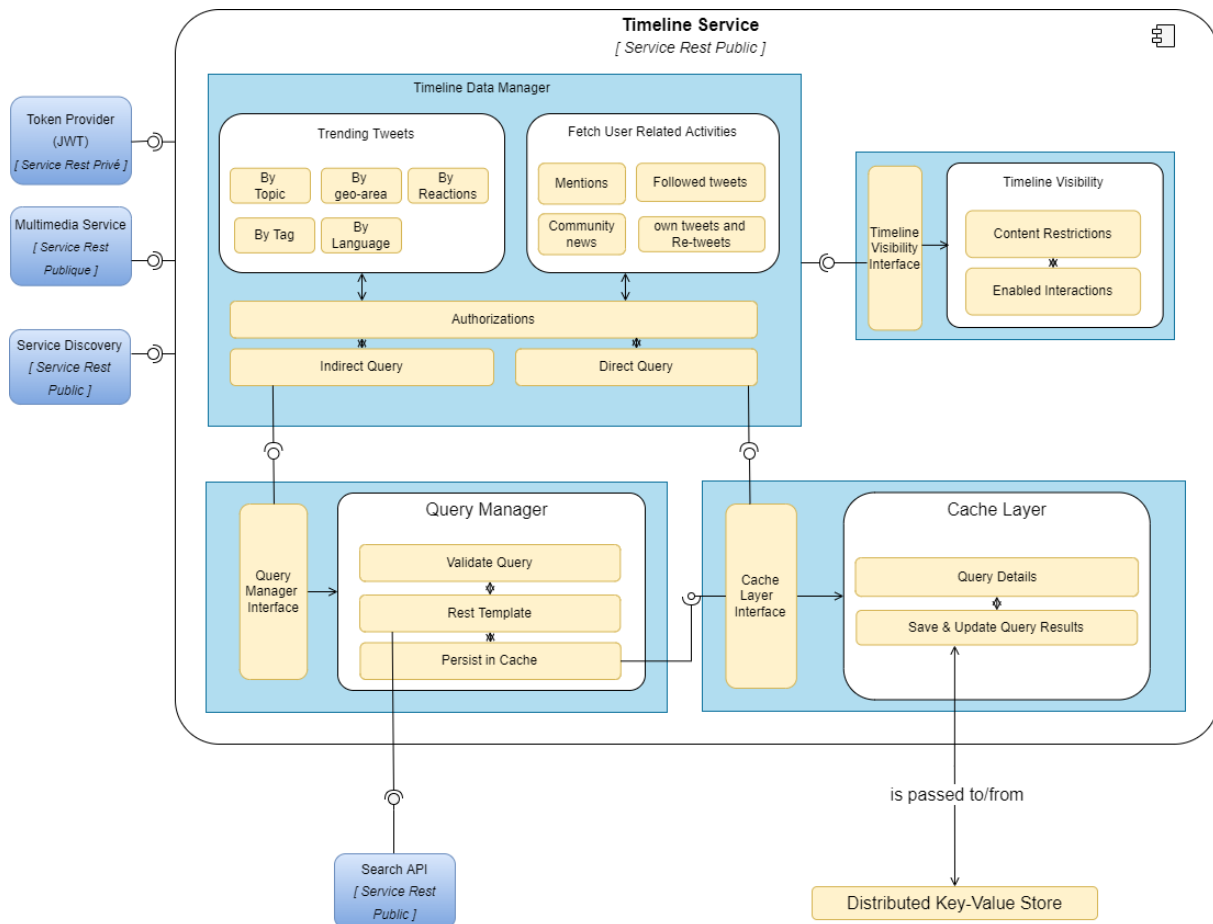


4.3.3.2 Description des composants:

- Graph node opérations : s'occupe de la création des nœuds du social graph : utilisateurs et groupes.
- Graph link opérations : il gère les différents liens qu'on pourra avoir entre les utilisateurs et les groupes : abonner , désabonner , bloquer, débloquer, modifier les préférences, rejoindre un groupe et quitter un groupe.
- Graph fast fetch : s'occupe des différentes recherches qu'on effectue en se basant sur le social graph, il a un moteur de recherche très performant qui se base sur une mémoire cache.

4.3.4 Timeline Service

4.3.4.1 Architecture Solution:

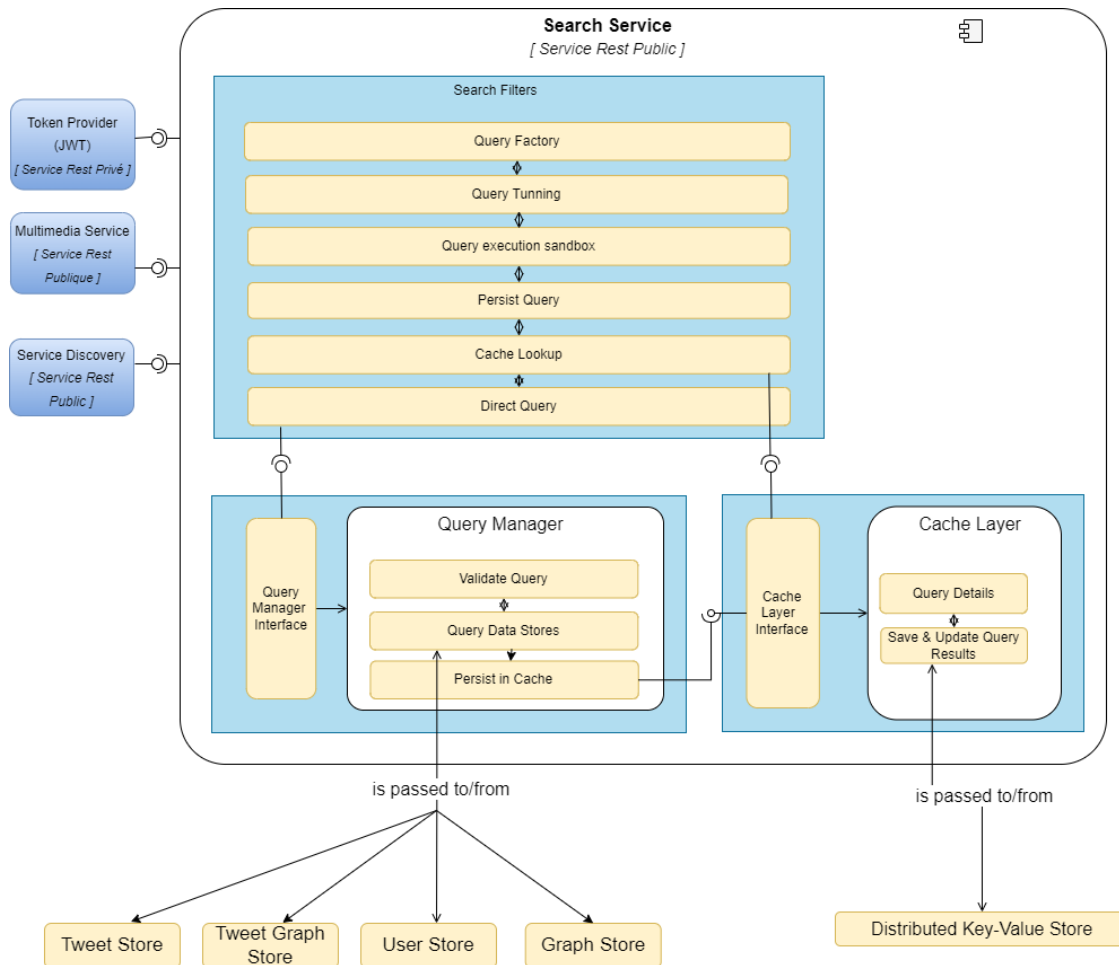


4.3.4.2 Description des composants:

- Trending tweets : il récupèrent les tweets en se basant sur : la localisation, les réactions, les tags, le langage....
- Fetch User Related Activities : il se charge d'obtenir les informations à propos des activités et des interactions de l'utilisateur authentifié tel que ses propres tweets, les mentions, les nouveautés de ses communautés, les tweets qu'il suit ...
- Timeline Visibility : son rôle est de gérer les limitation du contenu qui s'affiche sur la timeline d'un utilisateur, et autoriser les interactions;
- Query Manager : il planifie l'exécution des requêtes et dirige les requêtes vers les tables appropriées dans l'entrepôt de données.
- Cache Layer : stocke les résultats et les détails des requêtes de sorte que si une requête identique est reçue à l'avenir, les résultats peuvent être rapidement renvoyés.

4.3.4 Search Service

4.3.4.1 Architecture Solution:

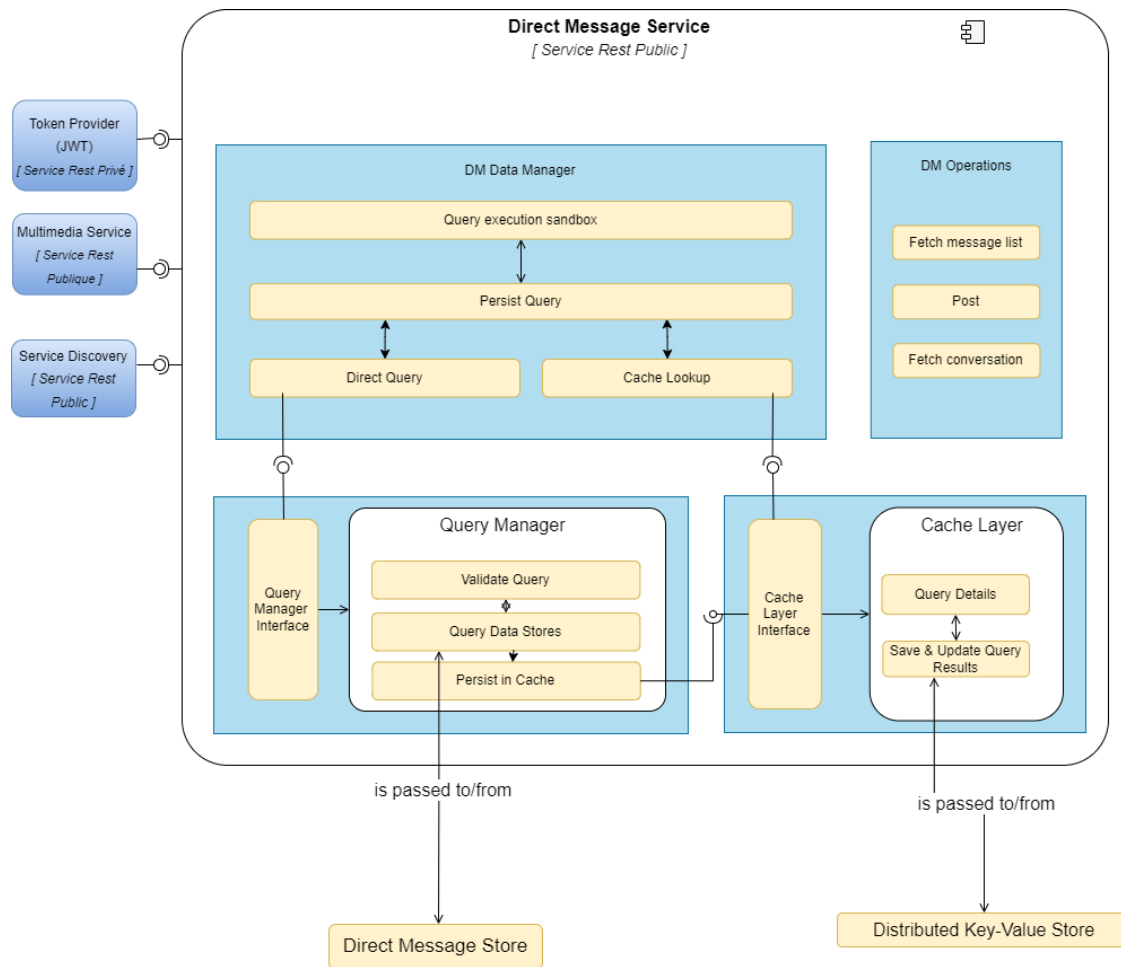


4.3.4.2 Description des composants:

- **Search Filters** : il regroupe les modules de création de requêtes, ses méthodes d'optimisation, les instructions d'exécution, la vérification de la persistance, la recherche et la récupération des résultats dans le cache, la gestion des requêtes directes.

4.3.4 Direct Message Service

4.3.4.1 Architecture Solution:



4.3.4.2 Description des composants:

- Direct Message Data Manager: il contient les outils et les instructions de l'exécution des requêtes, et vérifie la persistance des requêtes.
- Direct Message Operations : il récupère la liste des messages pour un utilisateur authentifié, lui permet d'envoyer des messages, et de voir la list de ses conversations

5-Description technique de l'architecture

5.1. Le frontend

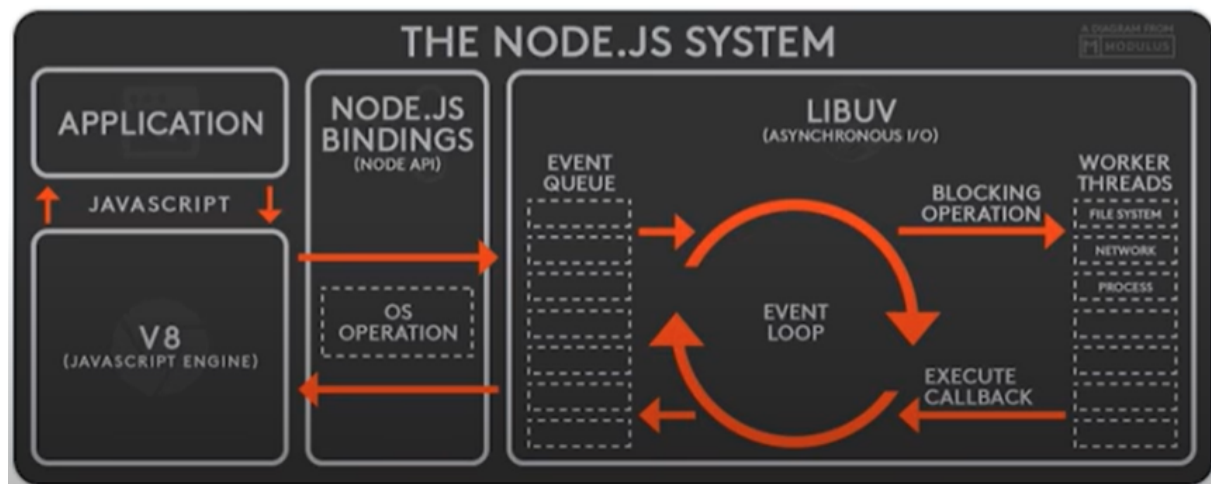
Pour le front-end nous avons décidé de nous concentrer sur la partie web. Cette dernière requiert le choix d'un framework front-end avec un state manager et une restTemplate pour qu'on puisse envoyer des requêtes Http à nos serveurs.

	Performance	Compatibilité	Extensibilité	Disponibilité
ReactJs	0.8	1	1	1
AngularJS	0.6	1	1	1
VueJS	0.8	0.8	1	1
VanillaJS	1	0.5	0.5	1

De plus React est une librairie light-weight comparé à angular et elle est supporté par facebook est la documentation est très intuitive, de plus il possède des fonctionnalités PWA permettant ainsi une intégration web-mobile facile. Notre choix de technologie est donc **React.Js**

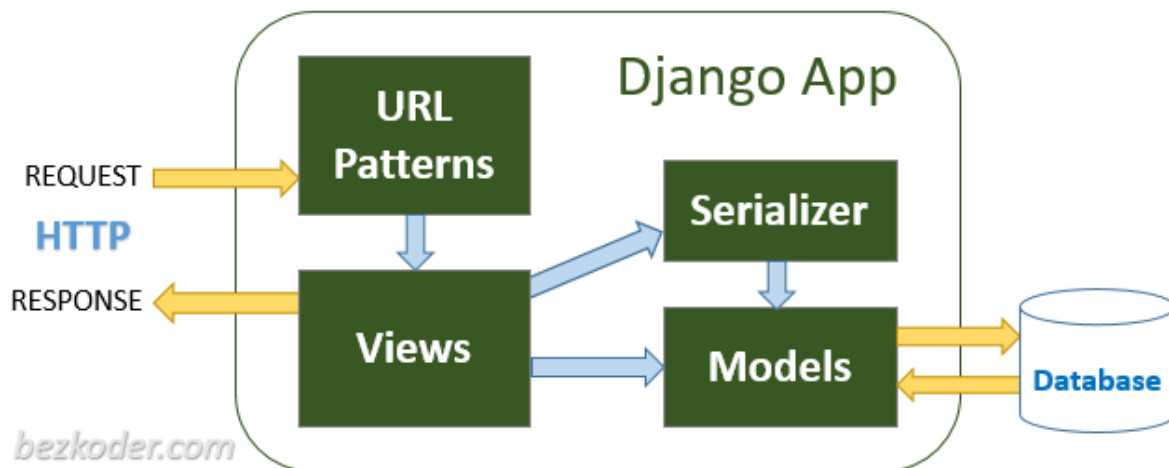
5.2. Les API

Pour le front-end nous avons décidé d'utiliser node.js et express.js pour tous les composants sans partie filtre ou query factory, vu que c'est la technologie la plus portable, non bloquante ainsi on peut créer avec des web services et des "real-time applications". De plus node est compatible avec Reactjs notre choix au front-end. Voici l'implémentation de notre API .



	Performance	Compatibilité	Extensibilité	Rapidité de dev
nodeJS	1	1	1	0.5

Par contre lorsqu'on essaye de créer des query performante avec des filtres on optera pour l'utilisation de Django un web framework robuste et sécurisé via une ORM qui possède un ensemble de librairies qui facilite le développement notamment Django Rest Framework qu'on utilisera pour générer nos API de recherche.



	Performance	Compatibilité	Extensibilité	Rapidité de dev
Django	0.8	1	1	1

5.2.1. Tweet API

- Gestion des Tweets

Nous disposons de deux méthodes pour gérer les Tweets, POST et DELETE. La méthode POST nous permet de publier des sondages, de citer des Tweets, de Tweet avec des paramètres de réponse, de Tweet avec géo, de Tweet avec des médias et de taguer les utilisateurs, et de Tweet aux Super Followers, en plus d'autres fonctionnalités. De même, la méthode DELETE sert à supprimer un Tweet spécifique. Pour la méthode POST, il est possible de passer les paramètres que recherche l'utilisateur pour lui permettre de personnaliser davantage sa requête.

POST /v1/tweets

Crée un Tweet de la part d'un utilisateur authentifié

Endpoint URL :

<https://api.twitter.com/v1/tweets>

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du corps JSON :

Nom	Type	Description
direct_message_deep_link	string	Tweeté un lien direct vers une conversation par message direct avec un compte.
geo	object	Un objet JSON qui contient des informations de localisation pour un Tweet. Les Tweets ne peuvent être localisés que si la fonction géographique est activée dans les paramètres de profil. Si cette option n'est pas activée, le paramètre de localisation peut toujours être ajouté au corps de la requête, mais il ne figurera pas dans le Tweet.
geo.place_id	string	Identifiant de lieu attaché au Tweet pour la géolocalisation.
media	object	Un objet JSON qui contient des informations sur le média attaché au Tweet créé.
media.media_ids	array	Une liste d'identifiants de médias attachés au Tweet. Ceci n'est requis que si la requête inclut les tagged_user_ids.
reply_settings	string	Paramètres permettant d'indiquer qui peut répondre au Tweet. Les options comprennent "mentionedUsers" et "following". Si le champ n'est pas spécifié, il est défini par défaut sur tout le monde.
text	string	Texte du Tweet en cours de création. Ce champ est obligatoire si media.media_ids n'est pas présent.

Exemple de requête :

```
curl -X POST https://api.twitter.com/v1/tweets -H  
"Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:  
application/json" -d '{"text": "Hello World"}
```

Exemple de réponse :

```
{  
  "data": {  
    "id": "1445880548472328192",  
    "text": "Hello world!"  
  }  
}
```

Champs de la réponse :

Nom	Type	Description
id	string	Identifiant du Tweet fraîchement créé.
text	string	Texte du Tweet fraîchement créé.

DELETE /v1/tweets/:t_id :

Permet à un utilisateur ou à un utilisateur authentifié de supprimer un Tweet.

Endpoint URL :

```
https://api.twitter.com/v1/tweets/:t_id
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
id	string	L'identifiant du tweet à supprimer.

Exemple de requête :

```
curl -X DELETE
https://api.twitter.com/v1/tweets/1445880548472328192 -H
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": {
    "deleted": true
  }
}
```

Champs de la réponse :

Nom	Type	Description
deleted	boolean	Indique si le Tweet spécifié a été supprimé à la suite de cette demande. La valeur renvoyée vaut true pour une demande de suppression réussie.

- Affichage des tweets :

Bien qu'il existe une variété de méthodes HTTP, de sélection et de livraison qui peuvent livrer, publier et agir sur les Tweets, ce groupe de points terminaux REST renvoie simplement un Tweet ou un groupe de Tweets, spécifié par un identifiant de Tweet. Bien que simples, ces points terminaux peuvent être utilisés pour recevoir des informations actualisées sur un tweet, vérifier qu'un tweet est disponible ou mettre à jour des informations stockées à la suite d'un événement de conformité.

Les points terminaux utilisent la méthode GET HTTP et renvoient un ou plusieurs objets Tweet, qui fournissent des champs tels que le texte du Tweet, la date de création, les URL incluses, etc.

GET /v1/tweets - Django

Renvoie une variété d'informations sur le Tweet spécifié par l'ID ou la liste d'ID demandée.

Endpoint URL :

`https://api.twitter.com/v1/tweets`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres de la requête :

Nom	Type	Description
ids	string	Une liste d'identifiants de Tweet séparés par des virgules. Jusqu'à 100 sont autorisés dans une seule demande.
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête <code>expansions=attachments.media_keys</code> a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête <code>expansions=geo.place_id</code> a été inclus dans la requête.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer

	reply_settings, source, text)	l'expansion expansions=referenced_tweets.id pour renvoyer les champs spécifiés à la fois pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets?ids=126132639932071526,1278347468690915330" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "1261326399320715264",
      "text": "Tune in to the @MongoDB @Twitch stream featuring our very own @suhemparack to learn about Twitter Developer Labs - starting now https://t.co/fAWpYi3o5O"
    },
    {
      "id": "1278347468690915330",
      "text": "Good news and bad news: nn2020 is half over"
    }
  ]
}
```

Champs de réponse :

Nom	Type	Description
id (default)	string	Identifiant unique de ce Tweet. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.

text (default)	string	Le contenu du Tweet. Pour renvoyer ce champ, il faut ajouter tweet.fields=text dans le paramètre de requête de la demande.
created_at	date (ISO 8601)	Temps de création du Tweet. Pour renvoyer ce champ, il est nécessaire d'ajouter tweet.fields=created_at dans le paramètre de requête de la demande.
attachments	object	Spécifie le type de pièces jointes (si elles existent) présentes dans ce Tweet. Pour renvoyer ce champ, il faut indiquer tweet.fields=attachments dans le paramètre de requête de la demande.
geo	object	Contient des détails sur le lieu marqué par l'utilisateur dans ce tweet, s'il en a spécifié un. Pour renvoyer ce champ, il s'agit d'ajouter tweet.fields=geo dans le paramètre de requête de la demande.
lang	object	Langue du Tweet, si elle est détectée par l'application. Renvoie une balise de langue BCP47. Pour renvoyer ce champ, il faut ajouter tweet.fields=lang dans le paramètre de requête de la demande.
reply_settings	string	Indique qui peut répondre à ce Tweet. Les champs retournés sont everyone, mentionedUsers, et following. Pour renvoyer ce champ, il suffit de préciser tweet.fields=reply_settings dans le paramètre de requête de la demande.
includes	object	Lorsque le paramètre d'expansion est inclus, les objets référencés sont retournés s'ils sont disponibles.
errors	object	Contient des détails sur les erreurs qui ont affecté l'un des Tweets demandés.

GET /v1/tweets/:t_id

Renvoie une variété d'informations sur un seul Tweet spécifié par son identifiant.

Endpoint URL :

`https://api.twitter.com/v1/tweets/:t_id`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	Identifiant unique du Tweet à rechercher.

Paramètres de la requête :

Nom	Type	Description
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête <code>expansions=attachments.media_keys</code> a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête <code>expansions=geo.place_id</code> a été inclus dans la requête.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être

	possibly_sensitive, referenced_tweets, reply_settings, source, text)	spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer l'expansion expansions=referenced_tweets.id pour renvoyer les champs spécifiés à la fois pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets/126132639932071526" -H
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "1278347468690915330",
      "text": "Good news and bad news: nn2020 is half over"
    }
  ]
}
```

Champs de réponse :

Nom	Type	Description
id (default)	string	Identifiant unique de ce Tweet. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text (default)	string	Le contenu du Tweet. Pour renvoyer ce champ, il faut ajouter tweet.fields=text dans le paramètre de requête de la

		demande.
created_at	date (ISO 8601)	Temps de création du Tweet. Pour renvoyer ce champ, il est nécessaire d'ajouter tweet.fields=created_at dans le paramètre de requête de la demande.
attachments	object	Spécifie le type de pièces jointes (si elles existent) présentes dans ce Tweet. Pour renvoyer ce champ, il faut indiquer tweet.fields=attachments dans le paramètre de requête de la demande.
geo	object	Contient des détails sur le lieu marqué par l'utilisateur dans ce tweet, s'il en a spécifié un. Pour renvoyer ce champ, il s'agit d'ajouter tweet.fields=geo dans le paramètre de requête de la demande.
lang	object	Langue du Tweet, si elle est détectée par l'application. Renvoie une balise de langue BCP47. Pour renvoyer ce champ, il faut ajouter tweet.fields=lang dans le paramètre de requête de la demande.
reply_settings	string	Indique qui peut répondre à ce Tweet. Les champs retournés sont everyone, mentionedUsers, et following. Pour renvoyer ce champ, il suffit de préciser tweet.fields=reply_settings dans le paramètre de requête de la demande.
includes	object	Lorsque le paramètre d'expansion est inclus, les objets référencés sont retournés s'ils sont disponibles.
errors	object	Contient des détails sur les erreurs qui ont affecté l'un des Tweets demandés.

PUT /v1/tweets/:t_id/hidden

Permet de masquer ou de démasquer une réponse à un Tweet.

Endpoint URL :

`https://api.twitter.com/v1/tweets/:t_id/hidden`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	Identifiant unique du Tweet à masquer ou à démasquer. Le Tweet doit appartenir à une conversation initiée par l'utilisateur authentifié.

Paramètre du corps JSON :

Nom	Type	Description
hidden	boolean	Indique l'action à effectuer. Pour le cacher, indiquez true et pour le dévoiler, false. Si un tweet est déjà masqué (ou si un tweet n'est pas masqué), l'appel aboutira.

Exemple de requête :

```
curl -XPUT 'https://api.twitter.com/v1/tweets/:t_id/hidden'  
-H "Authorization: Bearer $ACCESS_TOKEN"  
-H "Content-type: application/json"  
-d '{"hidden": true}'
```

Exemple de réponse :

```
{  
  "data": {  
    "hidden": true  
  }  
}
```

Champs de réponse :

Nom	Type	Description
hidden	boolean	Indique si le Tweet a été masqué ou non avec succès.

GET /v1/tweets/:t_id/retweeted_by

Permet d'obtenir des informations sur les personnes qui ont retweeté un Tweet.

Endpoint URL :

https://api.twitter.com/v1/tweets/:t_id/retweeted_by

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	Identifiant du Tweet dont il faut demander le retweet aux utilisateurs.

Paramètres de la requête :

Nom	Type	Description
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque Tweet épinglé retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs de Tweet ne seront affichés que si l'utilisateur a épinglé un Tweet et si le paramètre de requête <code>expansions=pinned_tweet_id</code> a été incluse à la requête.
user.fields	enum (created_at,	Une sélection des champs spécifiques de

	description, id, location, name, profile_image_url, protected, url, username, verified)	l'utilisateur qui seront livrés avec chaque objet utilisateur retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.
--	---	---

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets/126132639932071526/retweeted_by" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "783214",
      "name": "Twitter",
      "username": "Twitter"
    },
    {
      "id": "1526228120",
      "name": "Twitter Data",
      "username": "TwitterData"
    }
  ],
  "meta": {
    "result_count": 2
  }
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans

		son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
errors	object	Contient des détails sur les erreurs qui ont affecté l'un des utilisateurs demandés.

GET /v1/tweets/:t_id/liking_users

La première permet d'obtenir des informations sur les utilisateurs qui aiment un Tweet. Les 100 derniers utilisateurs qui ont aimé le Tweet spécifié sont affichés.

Endpoint URL :

`https://api.twitter.com/v1/tweets/:t_id/liking_users`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	Identifiant du Tweet pour lequel la réaction Like des utilisateurs a été faite.

Paramètres de la requête :

Nom	Type	Description
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête

		expansions=attachments.media_keys a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête expansions=geo.place_id a été inclus dans la requête.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque Tweet épinglé retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs de Tweet ne seront affichés que si l'utilisateur a épinglé un Tweet et si le paramètre de requête expansions=pinned_tweet_id a été incluse à la requête.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs spécifiques de l'utilisateur qui seront livrés avec chaque objet utilisateur retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets/126132639932071526/liking_users" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "783214",
      "name": "Twitter",
```

```
    "username": "Twitter"
  },
  {
    "id": "1526228120",
    "name": "Twitter Data",
    "username": "TwitterData"
  },
  {
    "id": "2244994945",
    "name": "Twitter Dev",
    "username": "TwitterDev"
  },
  {
    "id": "6253282",
    "name": "Twitter API",
    "username": "TwitterAPI"
  }
]
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.

username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
errors	object	Contient des détails sur les erreurs qui ont affecté l'un des utilisateurs demandés.

5.2.2. Users API

- Interaction avec les tweets :

Retweeter et aimer des Tweets sont l'une des principales fonctions que les gens utilisent pour participer à la conversation publique sur Twitter. D'une part, avec les points terminaux de recherche des likes, il est possible d'afficher une liste des comptes qui ont aimé un tweet donné, ou les tweets qu'un compte particulier a aimés. Ces critères peuvent être utilisés pour comprendre quel type de contenu un compte ou un groupe de comptes spécifique préfère. D'autre part, avec les terminaux de consultation des retweets, la liste des comptes qui ont retweeté un Tweet donné s'affiche. En outre, les nouveaux paramètres de gestion des retweets permettent de retweeter un tweet ou d'annuler un retweet.

GET /v1/users/:user_id/liked_tweets

Permet d'obtenir des informations sur les Tweets aimés d'un utilisateur.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/liked_tweets`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	Identifiant de l'utilisateur pour lequel il faut collecter des tweets aimés.

Paramètres de la requête :

Nom	Type	Description
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête <code>expansions=attachments.media_keys</code> a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête <code>expansions=geo.place_id</code> a été inclus dans la requête.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque Tweet épinglé retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs de Tweet ne seront affichés que si l'utilisateur a épinglé un Tweet et si le paramètre de requête <code>expansions=pinned_tweet_id</code> a été incluse à la requête.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs spécifiques de l'utilisateur qui seront livrés avec chaque objet utilisateur retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/2244994945/liked_tweets"
-H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "1294346980072624128",
      "text": "I awake from five years of slumber
https://t.co/OEPVyAFcfB"
    },
    {
      "id": "1283153843367206912",
      "text": "@wongmjane Wish we could tell you more, but I'm only a
teapot"
    }
  ]
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.

POST /v1/users/:user_id/likes

Fait en sorte qu'un identifiant utilisateur désigné dans le paramètre de path réagit avec un Like au Tweet cible.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/likes`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	L'identifiant de l'utilisateur pour lequel nous aimons un Tweet. Il doit correspondre à notre propre ID d'utilisateur ou à celui d'un utilisateur qui s'authentifie, ce qui signifie que nous devons transmettre les tokens d'accès associés à l'ID d'utilisateur lors de la demande d'authentification.

Paramètre du corps JSON:

Nom	Type	Description
tweet_id	string	Identifiant du Tweet qui doit être aimé par l'identifiant de l'utilisateur.

Exemple de requête :

```
curl -X POST
"https://api.twitter.com/v1/users/2244994945/likes/" -H
"Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:
application/json" -d '{"tweet_id": "1228393702244134912"}'
```

Exemple de réponse :

```
{
  "data": {
    "liked": true
  }
}
```

```
}  
}
```

Champs de réponse :

Nom	Type	Description
liked	boolean	Indique si l'utilisateur a liké le Tweet spécifié comme résultat de cette requête.

DELETE /v1/users/:user_id/likes/:t_id

Permet à un utilisateur ou à un identifiant d'utilisateur authentifié de ne plus liker un Tweet. La demande finit sans action lorsque l'utilisateur envoie une demande à un utilisateur qui n'a pas réagi au Tweet ou qui a déjà annulé son Like pour le Tweet.

Endpoint URL :

https://api.twitter.com/v1/users/:user_id/likes/:t_id

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
id	string	L'identifiant de l'utilisateur pour lequel nous supprimons un Like d'un Tweet. Il doit correspondre à notre propre ID d'utilisateur ou à celui d'un utilisateur qui s'authentifie, ce qui signifie que nous devons transmettre les tokens d'accès associés à l'ID d'utilisateur lors de la demande d'authentification.
tweet_id	string	L'ID du Tweet que l'on souhaite unliker.

Exemple de requête :

```
curl -X DELETE
"https://api.twitter.com/v1/users/2244994945/likes/12283937022
44134912" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": {
    "liked": false
  }
}
```

Champs de réponse :

Nom	Type	Description
liked	boolean	L'ID du Tweet qu'on souhaite indiquer si l'utilisateur n'aime pas ce Tweet comme résultat de cette requête. La valeur renvoyée est false en cas de succès de la demande de unlike.

POST /v1/users/:user_id/retweets

Fait en sorte que l'ID utilisateur identifié dans le paramètre path retweeté le Tweet cible.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/retweets
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
-----	------	-------------

id	string	L'identifiant de l'utilisateur pour lequel nous re-tweetons un Tweet.
----	--------	---

Paramètre du corps JSON:

Nom	Type	Description
tweet_id	string	Identifiant du Tweet qui doit être retweeté par l'identifiant de l'utilisateur.

Exemple de requête :

```
curl -X POST
"https://api.twitter.com/v1/users/2244994945/retweets/" -H
"Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:
application/json" -d '{"tweet_id": "1228393702244134912"}'
```

Exemple de réponse :

```
{
  "data": {
    "retweeted": true
  }
}
```

Champs de réponse :

Nom	Type	Description
retweeted	boolean	Indique si l'utilisateur a retweeté le Tweet spécifié comme résultat de cette requête.

DELETE /v1/users/:user_id/retweets/:t_id

Permet à un utilisateur ou à un identifiant d'utilisateur authentifié de supprimer un Retweet. La demande finit sans action lorsque l'utilisateur envoie une demande à un utilisateur qui n'a pas retweeté ou qui a déjà annulé son Retweet.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/retweets/:t_id
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre du Path :

Nom	Type	Description
user_id	string	L'identifiant de l'utilisateur pour lequel nous supprimons le Retweet. Il doit correspondre à notre propre ID d'utilisateur ou à celui d'un utilisateur qui s'authentifie, ce qui signifie que nous devons transmettre les tokens d'accès associés à l'ID d'utilisateur lors de la demande d'authentification.
tweet_id	string	L'ID du Tweet que l'on souhaite supprimer le Retweet.

Exemple de requête :

```
curl -X DELETE  
"https://api.twitter.com/v1/users/2244994945/retweets/12283937  
02244134912" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{  
  "data": {  
    "retweeted": false  
  }  
}
```

Champs de réponse :

Nom	Type	Description
retweeted	boolean	L'ID du Tweet qu'on souhaite indiquer si l'utilisateur a supprimé ce Retweet comme résultat de cette

		requête. La valeur renvoyée est false en cas de succès d'une demande de suppression d'un Retweet.
--	--	---

- Affichage et recherche des utilisateurs :

Le point terminal RESTful utilise la méthode GET pour renvoyer des informations sur un utilisateur ou un groupe d'utilisateurs, spécifié par un ID utilisateur ou un nom d'utilisateur. La réponse comprend un ou plusieurs objets utilisateur, qui fournissent des champs tels que le nombre de suiveurs, la localisation, l'ID du Tweet épinglé et la biographie du profil. Les réponses peuvent également inclure des expansions pour renvoyer l'objet Tweet complet pour le Tweet épinglé d'un utilisateur, y compris le texte du Tweet, l'auteur et d'autres champs du Tweet.

Ce point terminal est généralement utilisé pour recevoir des informations actualisées sur un utilisateur, pour vérifier qu'un utilisateur existe ou pour mettre à jour les informations stockées à la suite d'un événement de conformité.

GET /v1/users

Renvoie une variété d'informations sur un ou plusieurs utilisateurs spécifié par l'ID ou la liste d'ID demandée.

Endpoint URL :

<https://api.twitter.com/v1/users>

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres de la requête :

Nom	Type	Description
ids	string	Une liste d'identifiants de Tweet séparés par des virgules. Jusqu'à 100 sont autorisés dans

		une seule demande.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données incluses.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users?ids=126132639932071526,1278347468690915330" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "126132639932071526",
      "username": "TwitterDev",
      "name": "Twitter Dev"
    },
    {
      "id": "1278347468690915330",
      "username": "Twitter",
      "name": "Twitter"
    }
  ]
}
```

```
}  
  ]  
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
errors	object	Contient des détails sur les erreurs qui ont affecté l'un des utilisateurs demandés.

GET /v1/users/user_id

Renvoie une variété d'informations sur un seul utilisateur spécifié par l'ID demandée.

Endpoint URL :

https://api.twitter.com/v1/users/:user_id

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
-----	------	-------------

Id	String	L'identifiant de l'utilisateur qu'on cherche.
----	--------	---

Paramètres de la requête :

Nom	Type	Description
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données includes.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/2244994945" -H  
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{  
  "data": {  
    "id": "2244994945",  
    "name": "Twitter Dev",  
    "username": "TwitterDev"  
  }  
}
```

```
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.

GET /v1/users/me

Renvoie une variété d'informations sur l'utilisateur authentifié.

Endpoint URL :

```
https://api.twitter.com/v1/users/me
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres de la requête :

Nom	Type	Description
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront

	reply_settings, source, text)	retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données incluses.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/me" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": {
    "id": "2244994945",
    "name": "Twitter Dev",
    "username": "TwitterDev"
  }
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.

name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.

GET /v1/users/by/username/:username

Renvoie une variété d'informations sur un ou plusieurs utilisateurs spécifié par leurs noms.

Endpoint URL :

<https://api.twitter.com/v1/users/by/username/:username>

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
username	String	Le nom Twitter (handle) de l'utilisateur qu'on cherche.

Paramètres de la requête :

Nom	Type	Description
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors

		que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données includes.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/by/username/TwitterDev"
-H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": {
    "id": "2244994945",
    "name": "Twitter Dev",
    "username": "TwitterDev"
  }
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet

		utilisateur.
errors	object	Contient des détails sur les erreurs qui concernent l'un des utilisateurs demandés.

- Follows des utilisateurs :

Suivre des utilisateurs est l'une des actions les plus fondamentales sur Twitter. D'une part, on peut explorer et analyser les relations entre les utilisateurs, ce qui est parfois appelé analyse de réseau. Plus précisément, il existe deux endpoints REST qui renvoient des objets utilisateur représentant les personnes suivies par un utilisateur donné ou celles qui suivent un utilisateur donné. D'autre part, on peut suivre ou s'abandonner des utilisateurs. Puisque nous faisons une demande au nom d'un utilisateur, nous devons authentifier ces endpoints et utiliser les tokens d'accès associés à l'utilisateur pour lequel nous souhaitons que l'action de follow / unfollow soit effectuée.

GET /v1/users/:user_id/following

Renvoie une liste de comptes que l'ID d'utilisateur spécifié a suivis.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/following`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
Id	String	L'identifiant de l'utilisateur dont on souhaite récupérer le suivi.

Paramètres de la requête :

Nom	Type	Description
max_results	integer	Le nombre maximum de résultats à renvoyer par page. Il peut être compris entre 1 et 1000. Par défaut, chaque page renvoie 100 résultats.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données incluses.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/2244994945/following?max_results=10" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "6253282",
      "name": "Twitter API",
      "username": "TwitterAPI"
    },
    {
```

```
    "id": "2244994945",
    "name": "Twitter Dev",
    "username": "TwitterDev"
  },
  {
    "id": "783214",
    "name": "Twitter",
    "username": "Twitter"
  },
  {
    "id": "1526228120",
    "name": "Twitter Data",
    "username": "TwitterData"
  }
],
"meta": {
  "result_count": 4,
  "next_token": "DFEDBNRET3MHCZZZ"
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours, ainsi que des détails sur la pagination.
meta.result_count	integer	Le nombre d'utilisateurs retournés dans cette

		requête. Ce nombre peut être inférieur à celui spécifié dans le paramètre de requête max_results.
--	--	---

GET /v1/users/:user_id/followers

Renvoie une liste d'utilisateurs qui sont des followers de l'ID utilisateur spécifié.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/followers`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
Id	String	L'identifiant de l'utilisateur dont on veut récupérer les followers.

Paramètres de la requête :

Nom	Type	Description
max_results	integer	Le nombre maximum de résultats à renvoyer par page. Il peut être compris entre 1 et 1000. Par défaut, chaque page renvoie 100 résultats.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront

	reply_settings, source, text)	retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données incluses.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/2244994945/followers?max_results=10" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "6253282",
      "name": "Twitter API",
      "username": "TwitterAPI"
    },
    {
      "id": "2244994945",
      "name": "Twitter Dev",
      "username": "TwitterDev"
    },
    {
      "id": "1526228120",
      "name": "Twitter Data",
      "username": "TwitterData"
    }
  ]
}
```

```
1,  
  "meta": {  
    "result_count": 3,  
    "next_token": "DFEDBNRFT3MHCZZZ"  
  }  
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours, ainsi que des détails sur la pagination.
meta.result_count	integer	Le nombre d'utilisateurs retournés dans cette requête. Ce nombre peut être inférieur à celui spécifié dans le paramètre de requête <code>max_results</code> .
meta.next_token	string	Token de pagination pour la prochaine page de résultats. Cette valeur est renvoyée lorsqu'il y a plusieurs pages de résultats, car la requête actuelle peut ne renvoyer qu'un sous-ensemble de résultats. Pour récupérer la liste complète, il suffit de passer la valeur de ce champ dans le paramètre de requête <code>pagination_token</code> . Lorsque ce champ n'est pas renvoyé dans la réponse, cela signifie que la dernière page de résultats a été atteinte et qu'il n'y a pas d'autres pages.

POST /v1/users/:user_id/following

Permet à un ID utilisateur de suivre un autre utilisateur. Si l'utilisateur cible n'a pas de Tweets publics, ce endpoint enverra une demande de suivi. La demande aboutit sans action lorsque l'utilisateur authentifié envoie une demande à un utilisateur qu'il suit déjà, ou s'il envoie une demande de suivi à un utilisateur qui n'a pas de Tweets publics.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/following
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
Id	String	L'identifiant de l'utilisateur authentifié pour lequel il faut lancer le suivi. En authentifiant la requête, il faut passer les tokens d'accès qui correspondent à cet utilisateur.

Paramètres du corps JSON :

Nom	Type	Description
target_user_id	String	Identifiant de l'utilisateur que doit suivre cet identifiant (l'utilisateur actuellement connecté).

Exemple de requête :

```
curl -X POST  
"https://api.twitter.com/v1/users/6253282/following" -H  
"Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:  
application/json" -d '{"target_user_id": "2244994945"}'
```

Exemple de réponse :

```
{
```

```
"data": {  
  "following": true,  
  "pending_follow": false  
}
```

Champs de réponse :

Nom	Type	Description
following	boolean	Indique si l'identifiant suit l'utilisateur spécifié comme résultat de cette demande. Cette valeur est fausse si l'utilisateur cible n'a pas de Tweets publics, car il devra approuver la demande de suivi.
pending_follow	boolean	Indique si l'utilisateur cible doit approuver la demande de suivi. L'utilisateur authentifié ne suivra l'utilisateur cible que lorsqu'il aura approuvé la demande de suivi entrante.

DELETE /v1/users/:user_id/following/:target_user_id

Permet à un ID utilisateur de suivre un autre utilisateur. Si l'utilisateur cible n'a pas de Tweets publics, ce endpoint enverra une demande de suivi. La demande aboutit sans action lorsque l'utilisateur authentifié envoie une demande à un utilisateur qu'il suit déjà, ou s'il envoie une demande de suivi à un utilisateur qui n'a pas de Tweets publics.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/following/:target_user_id
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur pour lequel on souhaite initier l'unfollow. Lors de l'authentification de la requête, il faut passer les tokens d'accès qui se réfèrent à cet utilisateur.
target_user_id	String	L'ID de l'utilisateur que l'on souhaite que le user_id unfollow.

Exemple de requête :

```
curl -X DELETE
"https://api.twitter.com/v1/users/2244994945/following/6253282"
-H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": {
    "following": false
  }
}
```

Champs de réponse:

Nom	Type	Description
following	boolean	Indique si l'user_id est en train d'unfollower l'utilisateur spécifié à la suite de cette requête. Cette valeur vaut false pour une demande unfollow réussie.

- Blocks des utilisateurs :

Le blocage d'utilisateurs est une fonction de sécurité essentielle sur Twitter. Nous pouvons créer des expériences plus sécurisées pour les utilisateurs de Twitter ou bloquer des utilisateurs de manière programmée selon les engagements précédents. Nous pouvons également débloquer des utilisateurs après une période donnée, selon les critères que nous déterminons. Grâce à la recherche de blocks, nous pouvons voir quels utilisateurs sont

bloqués par un utilisateur authentifié. Cela peut être utile pour déterminer comment les utilisateurs peuvent interagir avec un compte donné. Le endpoint GET de recherche de blocs nous permet de voir quels comptes sont bloqués par un utilisateur authentifié. Pour gérer les blocks, il existe deux méthodes : POST et DELETE. La méthode POST nous permet de bloquer un utilisateur, et la méthode DELETE nous permet de le débloquent.

GET /v1/users/:user_id/blocking

Renvoie une liste d'utilisateurs qui sont bloqués par l'ID utilisateur spécifié.

Endpoint URL :

https://api.twitter.com/v1/users/:user_id/blocking

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
user_id	String	L'ID de l'utilisateur dont on souhaite récupérer les utilisateurs bloqués. L'ID de l'utilisateur doit correspondre à l'ID de l'utilisateur qui s'authentifie

Paramètres de la requête :

Nom	Type	Description
max_results	integer	Le nombre maximum de résultats à renvoyer par page. Il peut être compris entre 1 et 1000. Par défaut, chaque page renvoie 100 résultats.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être

	possibly_sensitive, referenced_tweets, reply_settings, source, text)	spécifiés dans une liste séparée par des virgules. Les champs Tweet ne seront retournés que si l'utilisateur a un Tweet épinglé et si nous avons également inclus le paramètre de requête expansions = pinned_tweet_id dans notre requête. Alors que l'ID du Tweet référencé se trouve dans l'objet Tweet d'origine, nous retrouverons cet ID et tous les champs Tweet supplémentaires dans l'objet de données incluses.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules. Ces champs utilisateur spécifiés s'afficheront directement dans les objets de données utilisateur.

Exemple de requête :

```
curl "https://api.twitter.com/v1/users/2244994945/blocking" -H
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse:

```
{
  "data": [
    {
      "id": "1065249714214457345",
      "name": "Spaces",
      "username": "TwitterSpaces"
    },
    {
      "id": "783214",
      "name": "Twitter",
      "username": "Twitter"
    },
    {
      "id": "6253282",
      "name": "Twitter API",
      "username": "TwitterAPI"
    }
  ]
}
```

```
}  
  ]  
}
```

Champs de réponse :

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
name	string	Le nom commun de cet utilisateur, tel qu'il apparaît dans son profil.
username	string	L'identifiant Twitter (nom de page de profil) de cet utilisateur.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours, ainsi que des détails sur la pagination.
meta.result_count	integer	Le nombre d'utilisateurs retournés dans cette requête. Ce nombre peut être inférieur à celui spécifié dans le paramètre de requête max_results.

POST /v1/users/:user_id/blocking

Renvoie une liste d'utilisateurs qui sont bloqués par l'ID utilisateur spécifié.

Endpoint URL :

https://api.twitter.com/v1/users/:user_id/blocking

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI

Taux limité	OUI
-------------	-----

Paramètres du Path :

Nom	Type	Description
user_id	String	L'ID de l'utilisateur qui initialise la demande du block. L'ID de l'utilisateur doit correspondre à l'ID de l'utilisateur qui s'authentifie

Paramètres du corps JSON :

Nom	Type	Description
target_user_id	String	L'identifiant de l'utilisateur que l'on souhaite bloquer. Le corps doit contenir une chaîne de l'identifiant de l'utilisateur dans un objet JSON.

Exemple de requête :

```
curl -X POST
"https://api.twitter.com/v1/users/6253282/blocking" -H
"Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:
application/json" -d '{"target_user_id": "2244994945"}'
```

Exemple de réponse :

```
{
  "data": {
    "blocking": true
  }
}
```

Champs de réponse :

Nom	Type	Description
blocking	boolean	Indique si l'utilisateur bloque l'utilisateur spécifié comme résultat de cette requête.

DELETE /v1/users/:user_id/blocking/:target_user_id

Permet à l'ID utilisateur authentifié de débloquent un autre utilisateur. La demande aboutit sans action lorsque l'utilisateur envoie une demande à un utilisateur non bloqué ou déjà débloquent.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/blocking/:target_user_id
```

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur pour lequel on souhaite initier l'Unblock. Lors de l'authentification de la requête, il faut passer les tokens d'accès qui se réfèrent à cet utilisateur.
target_user_id	String	L'ID de l'utilisateur que l'on souhaite que le user_id unblock.

Exemple de requête :

```
curl -X DELETE  
"https://api.twitter.com/v1/users/2244994945/blocking/6253282"  
-H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{  
  "data": {  
    "blocking": false  
  }  
}
```


Champs de réponse:

Nom	Type	Description
blocking	boolean	Indique si l'user_id est en train d'unblocker l'utilisateur spécifié comme résultat de cette requête. Cette valeur vaut false pour une demande unblock réussie.

- Activation des interaction :

L'interaction avec les tweets et les post constitue l'une des plus importantes fonctionnalités de Twitter. Pour permettre à l'utilisateur d'avoir une bonne expérience, cette fonctionnalité lui permet de gérer les interactions avec ses posts. Par exemple désactiver la possibilité d'ajouter un commentaire, ne pas permettre le retweet de son post ...

POST /v1/users/:user_id/interactions/enable/:interaction

Permet à l'ID utilisateur authentifié d'activer une interaction. La demande aboutit sans action lorsque l'utilisateur essaie d'activer une interaction déjà active.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/interactions/enable/:interaction
```

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur authentifié
interaction	String	L'identifiant de l'interaction que l'on souhaite que le user_id active.

Exemple de requête :

```
curl -X POST
"https://api.twitter.com/v1/users/6253282/interactions/enable"
-H "Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:
application/json" -d '{"interaction": "like"}'
```

POST /v1/users/:user_id/interactions/disable/:interaction

Permet à l'ID utilisateur authentifié de désactiver une interaction. La demande aboutit sans action lorsque l'utilisateur essaie de désactiver une interaction déjà inactive.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/interactions/disable
/:interaction
```

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur authentifié
interaction	String	L'identifiant de l'interaction que l'on souhaite que le user_id désactive.

Exemple de requête :

```
curl -X POST
"https://api.twitter.com/v1/users/6253282/interactions/disable
" -H "Authorization: Bearer $ACCESS_TOKEN" -H "Content-type:
application/json" -d '{"interaction": "like"}'
```

POST /v1/users/:user_id/visibility/

Permet à l'ID utilisateur authentifié de changer son statut de visibilité : online ou non.

Endpoint URL :

```
https://api.twitter.com/v1/users/:user_id/visibility/
```

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur authentifié

- Gestion de compte utilisateur :

Cette fonctionnalité se charge de la gestion des comptes des utilisateurs. Elle regroupe les modules de création des comptes et de validations des informations et des données. De plus, pour les utilisateurs qui souhaitent passer à la version améliorée de leur compte afin de bénéficier de fonctions premium, ils peuvent activer la fonction vip dans les paramètres de leur compte.

POST /v1/users/create

Permettre de créer un nouvel utilisateur .

Endpoint URL :

```
https://api.twitter.com/v1/users/create
```

POST /v1/users/validate/:validation_token

Permettre de valider un compte d'utilisateur avec un token d'accès. Si la vérification échoue, le token est considéré comme non valide et la demande doit être rejetée avec le résultat 401 Unauthorized.

Endpoint URL :

```
https://api.twitter.com/v1/users/validate/:validation_token
```

Paramètres du Path :

Nom	Type	Description
validation_token	String	le token d'accès

Exemple de requête :

```
curl -X POST  
"https://api.twitter.com/v1/users/validate/AZD34_586DVK47"
```

La réponse :

Une réponse réussie à ce message se traduira par une réponse HTTP 200 OK et une structure JSON dans le corps de la réponse.

POST /v1/users/activate/:user_id

Permettre à un utilisateur de passer à la version premium.

Endpoint URL :

```
https://api.twitter.com/v1/users/activate/:user_id
```

Paramètres du Path :

Nom	Type	Description
user_id	String	L'identifiant de l'utilisateur authentifié.

Exemple de requête :

```
curl -X POST  
"https://api.twitter.com/v1/users/activate/6253282"
```

5.2.3. Timeline API

Le endpoint Tweet timeline de l'utilisateur est un endpoint REST qui reçoit un paramètre de chemin unique pour indiquer l'utilisateur souhaité (par l'ID utilisateur). Le endpoint peut renvoyer les 3 200 Tweets, Retweets, réponses et Tweets de citation les plus récents publiés par l'utilisateur. Les Tweets sont livrés dans l'ordre chronologique inverse, en commençant par le plus récent. Les résultats sont paginés jusqu'à 100 Tweets par page. Des tokens de pagination sont fournis pour parcourir de grands ensembles de Tweets. Les ID des Tweets les plus récents et les plus anciens inclus dans la page donnée sont également fournis sous forme de métadonnées, qui peuvent également être utilisées pour rechercher les Tweets récents dans la timeline ou pour naviguer dans la timeline. La chronologie des Tweets de l'utilisateur permet également de spécifier les paramètres start_time et end_time pour recevoir les Tweets qui ont été créés dans une certaine marge de temps. Le endpoint de la timeline Tweet de l'utilisateur prend en charge les paramètres de champs et d'expansions, et renvoie le nouveau format de données JSON.

GET /v1/users/:user_id/tweets

Renvoie les Tweets composés par un seul utilisateur, spécifié par l'ID utilisateur demandé. Par défaut, les dix derniers Tweets sont retournés par cette requête. En utilisant la pagination, les 3200 derniers Tweets peuvent être récupérés.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/tweets`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
id	string	Identifiant unique du compte Twitter (ID utilisateur) pour lequel il faut renvoyer des résultats. L'ID utilisateur peut être référencé à l'aide du endpoint user/lookup.

Paramètres de la requête:

Nom	Type	Description
end_time	date(ISO 8601)	AAAA-MM-DDTHH:mm:ssZ (ISO 8601/RFC 3339). L'horodatage UTC le plus récent ou le plus récent à partir duquel les Tweets seront fournis. Seuls les 3200 Tweets les plus récents sont disponibles. L'horodatage a une granularité de seconde et est inclusif (par exemple, 12:00:01 inclut la première seconde de la minute). L'heure minimale autorisée

		est 2010-11-06T00:00:01Z
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	<p>Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer l'expansion <code>expansions=referenced_tweets.id</code> pour renvoyer les champs spécifiés à la fois pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.</p>
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	<p>Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.</p>
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	<p>Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête <code>expansions=attachments.me</code></p>

		dia_keys a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête expansions=geo.place_id a été inclus dans la requête.

Exemple de requête :

```
curl "https://api.twitter.com/v1/user/2244994945/tweets" -H  
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{  
  "data": [  
    {  
      "id": "1334987486343299072",  
      "text": "console.log('Happy birthday, JavaScript!');" ,  
    },  
    {  
      "id": "1334920270587584521",  
      "text": "Live now!nJoin the first ever @Twitch stream"  
    },  
    {  
      "id": "1334564488884862976",  
      "text": "Hello world"  
    }  
  ],  
  "meta": {  
    "oldest_id": "1334564488884862976",
```

```
"newest_id": "1338971066773905408",  
"result_count": 3,  
"next_token": "7140dibdnow9c7btw3w29grvxfcgvpb9n9coehpk7xz5i"  
}  
}
```

Champs de réponse:

Nom	Type	Description
id	string	Identifiant unique de cet utilisateur. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text	string	Le contenu du Tweet. Pour renvoyer ce champ, il suffit d'ajouter <code>tweet.fields=text</code> dans le paramètre de requête de la demande.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours et les détails de la pagination.
meta.count	integer	Le nombre de résultats de Tweet retournés dans la réponse.
meta.newest_id	string	L'identifiant du tweet le plus récent est renvoyé dans la réponse.
meta.oldest_id	string	L'identifiant du Tweet le plus ancien renvoyé dans la réponse.
meta.next_token	string	Une valeur qui encode la prochaine "page" de résultats qui peut être demandée, via le paramètre de requête <code>pagination_token</code> .

GET /v1/users/:user_id/mentions

Renvoie les Tweets mentionnant un seul utilisateur spécifié par l'ID utilisateur demandé. Par défaut, les dix derniers Tweets sont retournés par requête. En utilisant la pagination, jusqu'à 800 Tweets les plus récents peuvent être récupérés.

Endpoint URL :

`https://api.twitter.com/v1/users/:user_id/mentions`

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres du Path :

Nom	Type	Description
id	string	Identifiant unique de l'utilisateur auquel il faut retourner les Tweets qui le mentionnent.

Paramètres de la requête:

Nom	Type	Description
end_time	date(ISO 8601)	AAAA-MM-DDTHH:mm:ssZ (ISO 8601/RFC 3339). L'horodatage UTC le plus récent ou le plus récent à partir duquel les Tweets seront fournis. Seuls les 3200 Tweets les plus récents sont disponibles. L'horodatage a une granularité de seconde et est inclusif (par exemple, 12:00:01 inclut la première seconde de la minute). L'heure minimale autorisée est 2010-11-06T00:00:01Z
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets,	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être

	reply_settings, source, text)	spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer l'expansion expansions=referenced_tweets.id pour renvoyer les champs spécifiés à la fois pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête expansions=attachments.media_keys a été inclus dans la requête.
place.fields	enum (contained_within, country, country_code, full_name, geo, id, name, place_type)	Une sélection des champs de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs

		souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête <code>expansions=geo.place_id</code> a été inclus dans la requête.
--	--	---

Exemple de requête :

```
curl "https://api.twitter.com/v1/user/2244994945/mentions" -H  
"Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{  
  "data": [  
    {  
      "id": "1375152598945312768",  
      "text": "@LeBraat @TwitterDev @LeGuud There's enough @twitterdev  
love to go around, @LeBraat"  
    },  
    {  
      "id": "1375152449594523649",  
      "text": "Apparently I'm not the only one (of my test accounts)  
that loves @TwitterDev nn@LeStaache @LeGuud"  
    },  
    {  
      "id": "1375152043455873027",  
      "text": "Can I join this @twitterdev love party?!"  
    },  
  ],  
  "meta": {  
    "oldest_id": "1375151827189137412",  
    "newest_id": "1375152598945312768",  
    "result_count": 4,  
    "next_token": "7140dibdn0w9c7btw3w3y5b6jqjnk3k4g5zkmfkvqwa42"  
  }  
}
```

```
}
```

Champs de réponse:

Nom	Type	Description
id	string	Identifiant unique de ce Tweet. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text	string	Le contenu du Tweet. Pour renvoyer ce champ, il suffit d'ajouter tweet.fields=text dans le paramètre de requête de la demande.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours et les détails de la pagination.
meta.count	integer	Le nombre de résultats de Tweet retournés dans la réponse.
meta.newest_id	string	L'identifiant du tweet le plus récent est renvoyé dans la réponse.
meta.oldest_id	string	L'identifiant du Tweet le plus ancien renvoyé dans la réponse.
meta.next_token	string	Une valeur qui encode la prochaine "page" de résultats qui peut être demandée, via le paramètre de requête pagination_token.

5.2.4. Search API

La recherche de tweets est une fonctionnalité importante utilisée pour faire apparaître les conversations Twitter sur un sujet ou un événement spécifique. Bien que cette fonctionnalité soit présente dans Twitter, ces points d'accès offrent une plus grande souplesse et une plus grande puissance lors du filtrage et de l'ingestion de tweets. On peut ainsi trouver plus facilement des données pertinentes pour nos recherches, créer des applications d'écoute en temps quasi réel ou, plus généralement, explorer, analyser et/ou agir sur les tweets liés à un sujet d'intérêt.

GET /v1/tweets/search/all

Renvoie l'historique complet des Tweets publics correspondant à une requête de recherche.

Endpoint URL :

<https://api.twitter.com/v1/tweets/search/all>

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre de la requête :

Nom	Type	Description
query	string	Une requête pour les Tweets correspondants. On peut utiliser tous les opérateurs disponibles et créer des requêtes d'une longueur maximale de 1 024 caractères.
end_time	date(ISO 8601)	AAAA-MM-DDTHH:mm:ssZ (ISO 8601/RFC 3339). Utilisé avec start_time. Le timestamp UTC le plus récent auquel les Tweets seront fournis. Le timestamp est en granularité seconde et est exclusif (par exemple, 12:00:01 exclut la première seconde de la minute). S'il est utilisé sans start_time, les Tweets de 30 jours avant end_time seront retournés par défaut. Si cette option n'est pas spécifiée, la valeur par défaut de end_time est [maintenant - 30 secondes].
tweet.fields	enum (attachments, author_id, created_at,	Une sélection des champs Tweet spécifiques qui seront

	entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer l'expansion expansions=referenced_tweets.id pour renvoyer les champs spécifiés à la fois pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.
media.fields	enum (duration_ms, height, media_key, preview_image_url, type, url, width)	Une sélection des champs de média spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra des champs multimédia que si le Tweet contient des médias et si le paramètre de requête expansions=attachments.media_keys a été inclus dans la requête.
place.fields	enum (contained_within,	Une sélection des champs

	country, country_code, full_name, geo, id, name, place_type)	de lieu spécifiques qui seront livrés dans chaque Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Le Tweet ne renverra les champs de lieu que si le Tweet contient un lieu et si le paramètre de requête <code>expansions=geo.place_id</code> a été inclus dans la requête.
--	--	--

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets/search/all?query=from%3Atwitterdev%20new%20-is%3Aretweet&max_results=10" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "1373001119480344583",
      "text": "Looking to get started with the Twitter API but new to APIs in general? @jessicagarson will walk you through everything you need to know in APIs 101 session. She'll use examples using our v2 endpoints, Tuesday, March 23rd at 1 pm EST.nnJoin us on Twitchnhttps://t.co/GrtBOXyHmB"
    },
    {
      "id": "1364984313154916352",
      "text": "\"I was heading to a design conference in New York and wanted to meet new people,\" recalls @aaronykng, creator of @flocknet. \"There wasn't an easy way to see all of the designers in my network, so I built one.\" Making things like this opened the doors for him to the tech industry."
    }
  ],
}
```

```
"meta": {  
  "newest_id": "1373001119480344583",  
  "oldest_id": "1364275610764201984",  
  "result_count": 2  
}
```

Champs de réponse:

Nom	Type	Description
id	string	Identifiant unique de ce tweet. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text	string	Le contenu du Tweet. Pour renvoyer ce champ, il suffit d'ajouter tweet.fields=text dans le paramètre de requête de la demande.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours et les détails de la pagination.
meta.count	integer	Le nombre de résultats de Tweet retournés dans la réponse.
meta.newest_id	string	L'identifiant du tweet le plus récent est renvoyé dans la réponse.
meta.oldest_id	string	L'identifiant du Tweet le plus ancien renvoyé dans la réponse.
meta.next_token	string	Une valeur qui encode la prochaine "page" de résultats qui peut être demandée, via le paramètre de requête pagination_token.

GET /v1/tweets/search/recent

Renvoie les Tweets des sept derniers jours qui correspondent à la requête de recherche.

Endpoint URL :

<https://api.twitter.com/v1/tweets/search/recent>

Authentification et limites de débit :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètre de la requête :

Nom	Type	Description
query	string	Une requête pour les Tweets correspondants.
end_time	date(ISO 8601)	AAAA-MM-DDTHH:mm:ssZ (ISO 8601/RFC 3339). Utilisé avec start_time. Le timestamp UTC le plus récent auquel les Tweets seront fournis. Le timestamp est en granularité seconde et est exclusif (par exemple, 12:00:01 exclut la première seconde de la minute). Par défaut, une requête renverra des tweets datant d'il y a 30 secondes si ce paramètre n'est pas inclus.
tweet.fields	enum (attachments, author_id, created_at, entities, geo, id, lang, possibly_sensitive, referenced_tweets, reply_settings, source, text)	Une sélection des champs Tweet spécifiques qui seront livrés dans chaque objet Tweet retourné. Les champs souhaités doivent être spécifiés dans une liste séparée par des virgules. Il est aussi possible de passer l'expansion expansions=referenced_tweets.id pour renvoyer les champs spécifiés à la fois

		pour le Tweet original et les Tweets référencés inclus. Les champs Tweet demandés s'afficheront à la fois dans l'objet de données Tweet d'origine et dans l'objet de données étendu Tweet référencé qui sera situé dans l'objet de données inclus.
user.fields	enum (created_at, description, id, location, name, profile_image_url, protected, url, username, verified)	Une sélection des champs utilisateur spécifiques qui seront livrés dans chaque Tweet retourné. Il suffit de spécifier les champs souhaités dans une liste séparée par des virgules.

Exemple de requête :

```
curl "https://api.twitter.com/v1/tweets/search/recent?query=from%3Atwitterdev%20new%20-is%3Aretweet&max_results=10" -H "Authorization: Bearer $ACCESS_TOKEN"
```

Exemple de réponse :

```
{
  "data": [
    {
      "id": "1373001119480344583",
      "text": "Looking to get started with the Twitter API but new to APIs in general? @jessicagarson will walk you through everything you need to know in APIs 101 session. She'll use examples using our v2 endpoints, Tuesday, March 23rd at 1 pm EST.nnJoin us on Twitchnhttps://t.co/GrtBOXyHmB"
    },
    {
      "id": "1364984313154916352",
      "text": "\"I was heading to a design conference in New York and wanted to meet new people,\" recalls @aaronykng, creator of @flocknet."
```

```
"There wasn't an easy way to see all of the designers in my network, so I built one." Making things like this opened the doors for him to the tech industry."
```

```
    }  
  ],  
  "meta": {  
    "newest_id": "1373001119480344583",  
    "oldest_id": "1364275610764201984",  
    "result_count": 2  
  }  
}
```

Champs de réponse:

Nom	Type	Description
id	string	Identifiant unique de ce tweet. Il est renvoyé sous forme de chaîne de caractères afin d'éviter les complications avec les langages et les outils qui ne peuvent pas gérer les grands nombres entiers.
text	string	Le contenu du Tweet. Pour renvoyer ce champ, il suffit d'ajouter tweet.fields=text dans le paramètre de requête de la demande.
meta	object	Cet objet contient des informations sur le nombre d'utilisateurs renvoyés dans la requête en cours et les détails de la pagination.
meta.count	integer	Le nombre de résultats de Tweet retournés dans la réponse.
meta.newest_id	string	L'identifiant du tweet le plus récent est renvoyé dans la réponse.
meta.oldest_id	string	L'identifiant du Tweet le plus ancien renvoyé dans la réponse.

5.2.5. Media API

Un objet multimédia représente une photo, une vidéo ou un GIF animé unique. Les objets multimédias sont utilisés par de nombreux points de terminaison de l'API Twitter et peuvent être inclus dans des Tweets, des messages directs, des profils d'utilisateurs, des créations publicitaires et autres. Chaque objet multimédia peut avoir plusieurs variantes d'affichage ou de lecture, avec des résolutions ou des formats différents.

POST /v1/media/upload

Cet endpoint permet de charger des images sur Twitter. Cet endpoint renvoie un `media_id` par défaut et peut éventuellement renvoyer un `media_key` lorsqu'un `media_category` est spécifié. Ces valeurs sont utilisées par les endpoints Twitter qui acceptent les images.

La requête :

Les requêtes doivent être au format POST multipart/form-data ou application/x-www-form-urlencoded.

URL de ressource :

`https://upload.twitter.com/v1/media/upload.json`

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres :

Nom	Requis	Description
media	requis	Le contenu brut du fichier binaire téléchargé. Ne peut pas être utilisé avec <code>media_data</code> .
media_category	facultatif	La catégorie qui représente la façon dont le média sera utilisé. Ce champ est obligatoire lorsque le média est utilisé avec les valeurs API Possible Ads : <code>amplify_video</code> , <code>tweet_gif</code> , <code>tweet_image</code> , et <code>tweet_video</code> .

media_data	requis	Le contenu du fichier codé en base64 qui est téléchargé. Ne peut pas être utilisé avec media.
additional_owners	facultatif	Une liste d'identifiants d'utilisateurs séparés par des virgules à définir comme propriétaires supplémentaires autorisés à utiliser le media_id retourné dans les Tweets ou les Cartes. Jusqu'à 100 propriétaires supplémentaires peuvent être spécifiés.

Exemple de requête :

POST

```
https://upload.twitter.com/v1/media/upload.json?media_category=tweet_image/jpeg
```

Exemple de réponse :

```
{
  "media_id": 710511363345354753,
  "media_id_string": "710511363345354753",
  "media_key": "3_710511363345354753",
  "size": 11065,
  "expires_after_secs": 86400,
  "image": {
    "image_type": "image/jpeg",
    "w": 800,
    "h": 320
  }
}
```

POST /v1/media/upload (INIT)

La requête de la commande INIT est utilisée pour initier une session de téléchargement de fichiers. Elle renvoie un media_id qui doit être utilisé pour exécuter toutes les requêtes suivantes.

La requête :

Les requêtes doivent être au format POST multipart/form-data ou application/x-www-form-urlencoded.

La réponse :

La réponse fournit un identifiant de média dans les champs `media_id` (entier de 64 bits) et `media_id_string` (chaîne de caractères). La chaîne `media_id_string` fournie dans la réponse de l'API doit être utilisée par JavaScript et d'autres langages qui ne peuvent pas représenter avec précision un nombre entier long. L'ensemble du fichier doit être téléchargé avant les secondes `expires_after_secs`. Le champ `additional_owners` permet de télécharger des médias en tant qu'utilisateur A, puis de les utiliser pour créer des Tweets en tant qu'utilisateur B.

URL de ressource :

`https://upload.twitter.com/v1/media/upload.json`

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres :

Nom	Requis	Description
<code>command</code>	requis	doit avoir la valeur INIT (case sensitive)
<code>total_bytes</code>	requis	La taille du média téléchargé en octets.
<code>media_type</code>	requis	Le type MIME du média chargé. Par exemple, video/mp4.
<code>media_category</code>	parfois	Une valeur d'énumération de type string qui identifie un cas d'utilisation de média. Cet identifiant est utilisé pour appliquer des contraintes spécifiques au cas d'utilisation (par exemple, la taille du fichier, la durée de la vidéo) et activer des

		fonctions avancées.
additional_owners	facultatif	Une liste d'identifiants d'utilisateurs séparés par des virgules à définir comme propriétaires supplémentaires autorisés à utiliser le media_id retourné dans les Tweets ou les Cartes. Jusqu'à 100 propriétaires supplémentaires peuvent être spécifiés.

Exemple de requête :

POST

```
https://upload.twitter.com/v1/media/upload.json?command=INIT&total_bytes=10240&media_type=image/jpeg
```

Exemple de résultat :

```
{
  "media_id": 710511363345354753,
  "media_id_string": "710511363345354753",
  "size": 11065,
  "expires_after_secs": 86400,
  "image": {
    "image_type": "image/jpeg",
    "w": 800,
    "h": 320
  }
}
```

GET /v1/media/upload (STATUS)

La commande STATUS est utilisée pour demander périodiquement des mises à jour de l'opération de traitement des médias. Une fois que la réponse à la commande STATUS est réussie, on peut passer à l'étape suivante, qui consiste généralement de créer Tweet avec media_id.

La requête :

Il s'agit d'une requête HTTP GET avec des paramètres d'URL.

La réponse :

Le corps de la réponse contient le champ `processing_info` qui fournit des informations sur l'état actuel de l'opération de traitement des médias. Il contient un champ `state` qui présente un flux de transition : `"pending" -> "in_progress" -> ["failed" | "succeeded"]`. On ne peut pas utiliser le `media_id` pour créer un Tweet ou d'autres entités avant que le champ d'état ne prenne la valeur `"succeeded"`.

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres :

Nom	Requis	Description
<code>command</code>	requis	doit avoir la valeur <code>STATUS</code> (case sensitive)
<code>media_id</code>	requis	La catégorie qui représente la façon dont le média sera utilisé. Ce champ est obligatoire lorsque le média est utilisé avec les valeurs API Possible Ads : <code>amplify_video</code> , <code>tweet_gif</code> , <code>tweet_image</code> , et <code>tweet_video</code> .

Exemple de requête :

```
GET
https://upload.twitter.com/v1/media/upload.json?command=STATUS
&media_id=710511363345354753
```

Exemple de résultat :

```
// Example of an in_progress response:

{
  "media_id":710511363345354753,
```



```
"media_id_string":"710511363345354753",
"expires_after_secs":3595,
"processing_info":{
  "state":"in_progress", // state transition flow is pending ->
in_progress -> [failed|succeeded]
  "check_after_secs":10, // check for the update after 10 seconds
  "progress_percent":8 // Optional [0-100] int value. Please don't
use it as a replacement of "state" field.
}
}

// Example of a failed response:

{
  "media_id":710511363345354753,
  "media_id_string":"710511363345354753",
  "processing_info":{
    "state":"failed",
    "progress_percent":12,
    "error":{
      "code":1,
      "name":"InvalidMedia",
      "message":"Unsupported video format"
    }
  }
}

// Example of a succeeded response:

{
  "media_id":710511363345354753,
  "media_id_string":"710511363345354753",
  "expires_after_secs":3593,
  "video":{
    "video_type":"video/mp4"
  },
  "processing_info":{
    "state":"succeeded",
```

```
"progress_percent":100,  
}  
}
```

5.2.6. Direct Messages API

Ces API nous permettent de créer des expériences client mieux personnalisées à grande échelle ainsi que d'autres interactions innovantes. Pour nous aider à créer des expériences de service client, de marketing et de participation plus engageantes pour les utilisateurs dans les messages directs, nous pouvons accéder à des endpoints pour publier des messages avec des réponses rapides, des médias, et plus encore.

GET /v1/direct_messages/events/show

Renvoie un seul événement de message direct pour l'identifiant donné.

URL de ressource :

https://api.twitter.com/v1/direct_messages/events/show.json

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres :

Nom	Requis	Description
id	requis	L'identifiant de l'événement de message direct qui doit être renvoyé.

Exemple de requête :

```
curl
"https://api.twitter.com/v1/v1/direct_messages/events/show.js
n?id=101"
```

Exemple de réponse :

```
{
  "event":
    "id": "101",
    "created_timestamp": "5300",
    "type": "message_create",
    "message_create": {
      ...
    }
}
```

POST /v1/direct_messages/events/new

Diffuse un nouvel événement message_create résultant en un message direct envoyé à un utilisateur spécifié par l'utilisateur d'authentification. Renvoie un événement en cas de succès. Prend en charge la publication de messages directs avec option de réponse rapide et de pièces jointes multimédia. Nécessite un corps POST JSON et un header Content-Type défini sur application/json. La définition de Content-Length peut également être requise si elle n'est pas automatique.

URL de ressource :

```
https://api.twitter.com/v1/direct_messages/events/new.json
```

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Event Object :

Nom	Requis	Description
type	requis	Le type d'événement à publier. Pour les messages directs, il faut utiliser message_create.
message_create.target.recipient_id	requis	L'identifiant de l'utilisateur qui doit recevoir le message
message_create.message_data	requis	Le Message Data Object définissant le contenu à délivrer au destinataire.

Message Data Object :

Nom	Requis	Description
text	requis	Le texte de votre message direct. L'URL est codée si nécessaire. Longueur maximale de 10 000 caractères.
attachment.type	facultatif	Le type de pièce jointe. Peut être média ou localisation.
attachment.media.id	facultatif	Un identifiant de média associé au message. Un message direct ne peut faire référence qu'à un seul media_id.

Exemple de requête :

```
curl -X POST
https://api.twitter.com/v1/direct_messages/events/new.json
--header 'content-type: application/json' --data '{"event":
{"type": "message_create", "message_create": {"target":
{"recipient_id": "RECIPIENT_USER_ID"}, "message_data":
{"text": "Hello World!"}}}}'
```

Exemple de réponse :

```
{
  "event": {
```

```
{
  "type": "message_create",
  "message_create": {
    "target": {
      "recipient_id": "RECIPIENT_USER_ID"
    },
    "message_data": {
      "text": "Hello World!"
    }
  }
}
```

GET /v1/direct_messages/events/list

Renvoie tous les événements de messages directs (envoyés et reçus) au cours des 30 derniers jours. Triés par ordre chronologique inverse.

URL de ressource :

https://api.twitter.com/v1/direct_messages/events/list.json

Information de ressource :

Formats de réponse	JSON
Nécessite une authentification	OUI
Taux limité	OUI

Paramètres :

Nom	Requis	Description
count	facultatif	Nombre maximum d'événements à renvoyer. 20 par défaut. 50 maximum.
cursor	facultatif	Pour parcourir des ensembles de résultats de plus d'une page, on utilise la propriété "next_cursor" de la requête précédente.

Exemple de requête :

```
curl
"https://api.twitter.com/v1/direct_messages/events/list.json"
```

Exemple de réponse :

```
{
  "next_cursor": "AB345dkfC",
  "events": [
    { "id": "110", "created_timestamp": "5300", ... },
    { "id": "109", "created_timestamp": "5200", ... },
    { "id": "108", "created_timestamp": "5200", ... },
    { "id": "107", "created_timestamp": "5200", ... },
    { "id": "106", "created_timestamp": "5100", ... },
    { "id": "105", "created_timestamp": "5000", ... },
    ...
  ]
}
```

DELETE /v1/direct_messages/events/destroy

Supprime le message direct spécifié dans le paramètre ID requis. L'utilisateur authentifiant doit être le destinataire du message direct spécifié. Les messages directs sont uniquement supprimés de l'interface du contexte utilisateur fourni. Les autres membres de la conversation peuvent toujours accéder aux messages directs. Une suppression réussie renvoie un code de réponse http 204 sans contenu.

URL de ressource :

```
https://api.twitter.com/v1/direct_messages/events/destroy.json
```

Information de ressource :

Formats de réponse	204 - sans contenu
Nécessite une authentification	OUI

Taux limité	OUI
-------------	-----

Paramètres :

Nom	Requis	Description
id	requis	L'identifiant de l'événement de message direct qui doit être supprimé.

Exemple de requête :

```
curl -X DELETE  
"https://api.twitter.com/v1/v1/direct_messages/events/destroy.  
json?id=1010101010101010101"
```

5.3. L'infrastructure

- Notre choix pour orchestrer les conteneurs Backend est docker-compose vu sa simplicité et le besoin que notre application soit highly available
- Pour avoir un cluster de très haute Availability nous allons utiliser KeepAlived parce que c'est l'un des outils les mieux documenté et supporter pour faire du heartbeat check entre serveurs
- Pour la partie load balancer et serveur Web on utilise Nginx puisqu'il possède des capacité de sécurité contre les attaques slow loris, DDOS... ainsi que des timeouts pour protéger notre backend

	Performance	Compatibilité	Extensibilité	Disponibilité
Docker-compose	0	1	1	1
KeepAlived	0	0	0	1
Ngnix	1	0	0	1

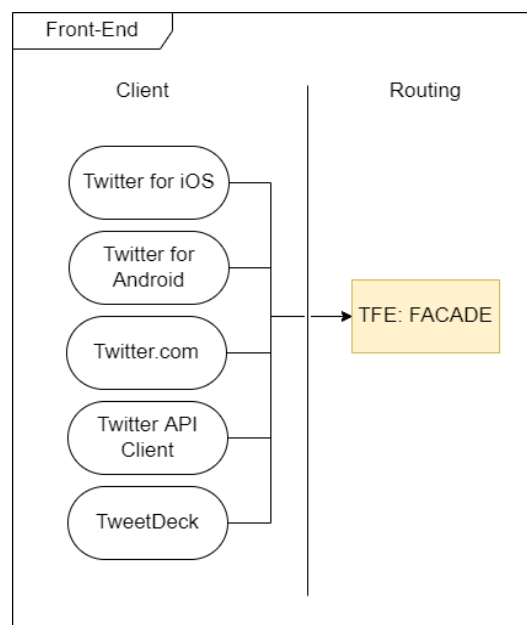
5.4. Le Stockage

- Pour la partie stockage relationnel nous allons utiliser PostgreSQL une base de données relationnelle open source puisque c'est une solution qui possède des capacités en full-text Search, SIG , Data visualisation ainsi que CRON jobs et beaucoup plus de features
- On supportera notre backend par des couches cache assuré par Redis une technologie universelle pour faire du cache et du Key Value Store.

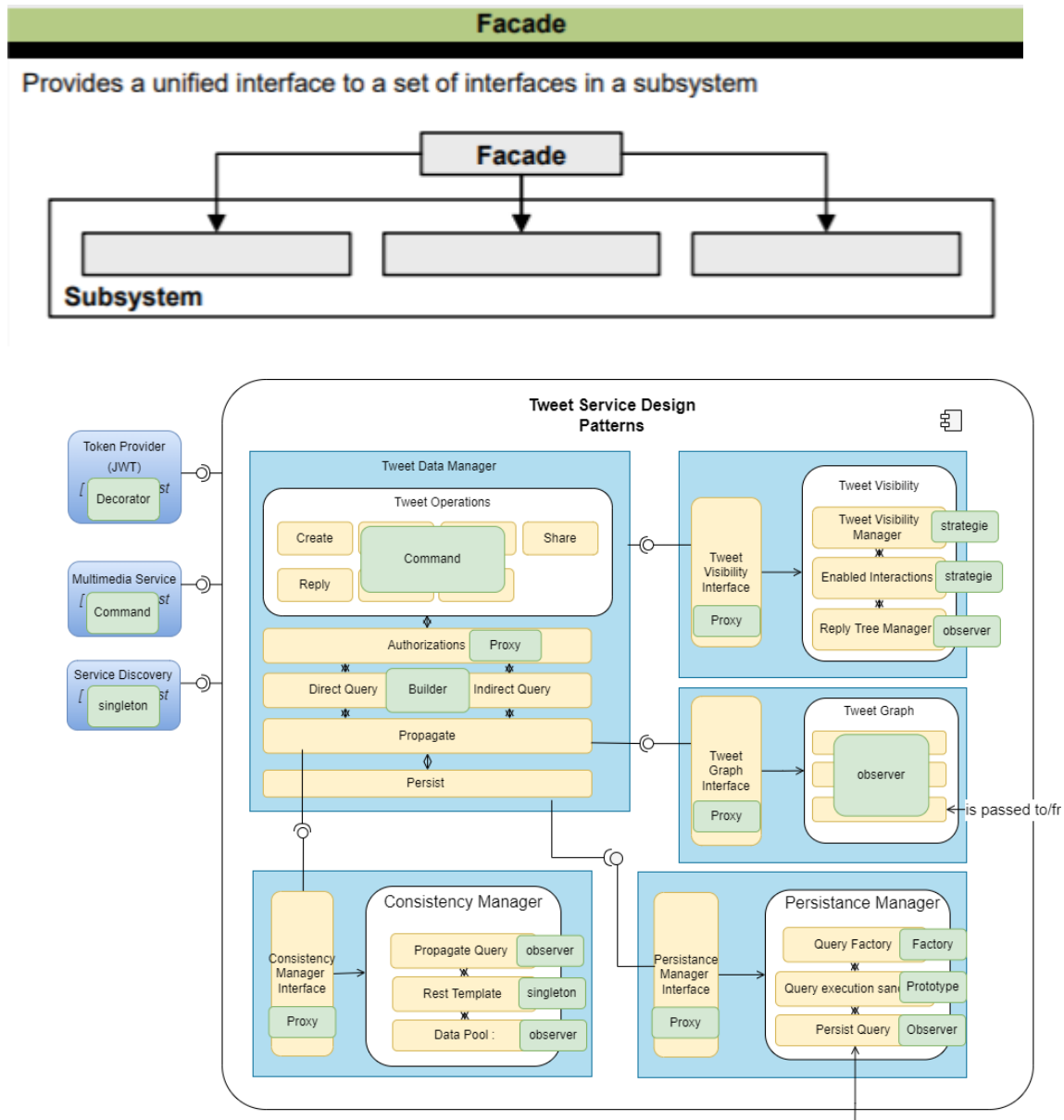
	Performance	Compatibilité	Extensibilité	Disponibilité
Postgresql	1	1	0	0
Redis	1	1	0	1

5.5. Les Design Patterns

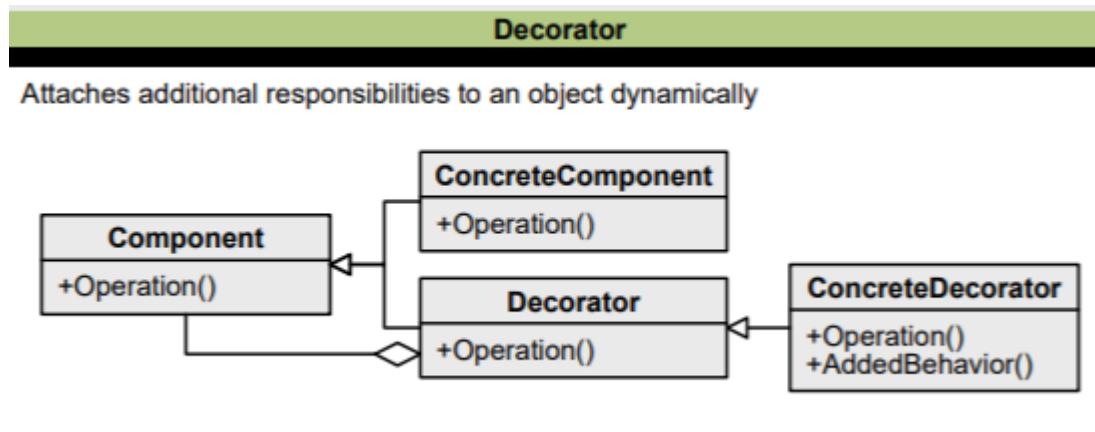
Les modèles de conception(design pattern) sont des solutions typiques à des problèmes courants dans la conception de logiciels. Chaque modèle est comme un plan que vous pouvez personnaliser pour résoudre un problème de conception particulier dans votre code.



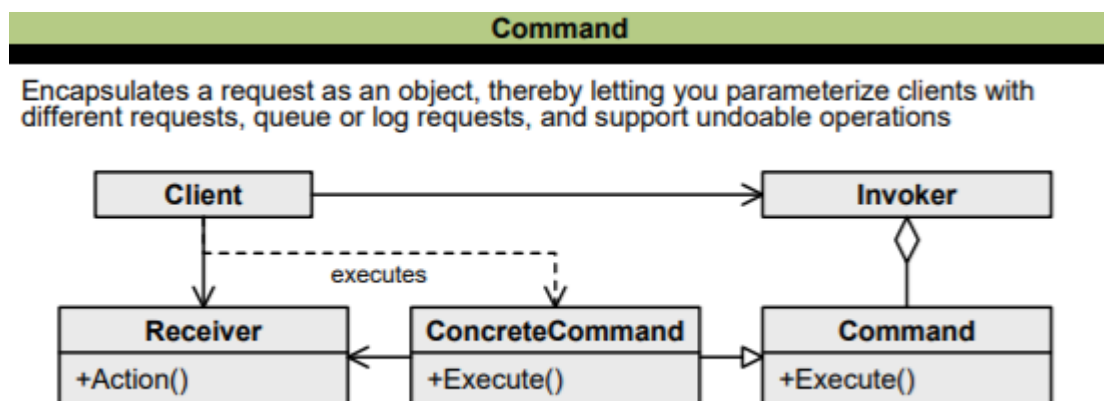
Design pattern : Facade



Design pattern : Decorator



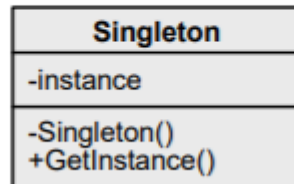
Design pattern : Command



Design pattern : Singleton

Singleton

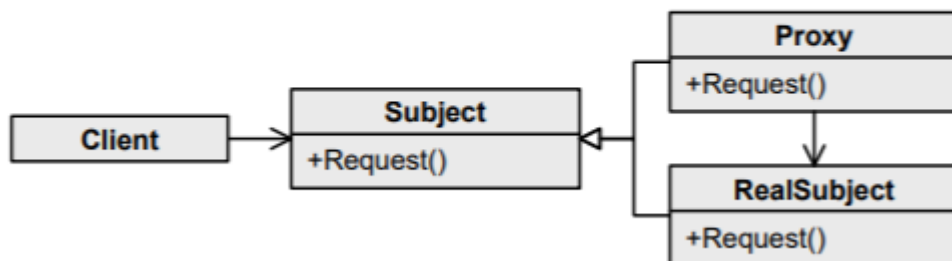
Ensure a class only has one instance and provide a global point of access to it



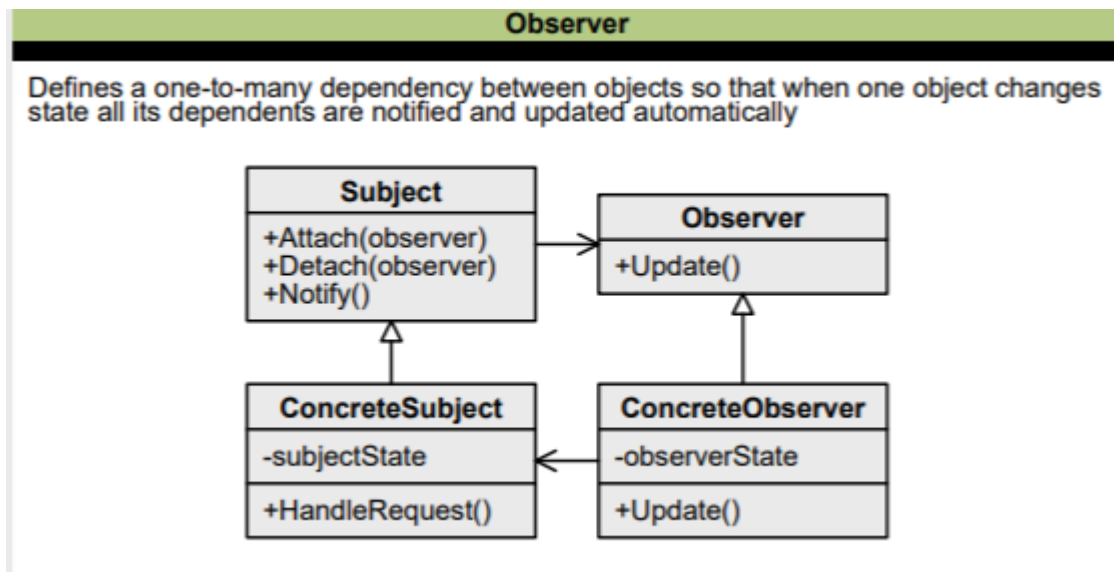
Design pattern : Proxy

Proxy

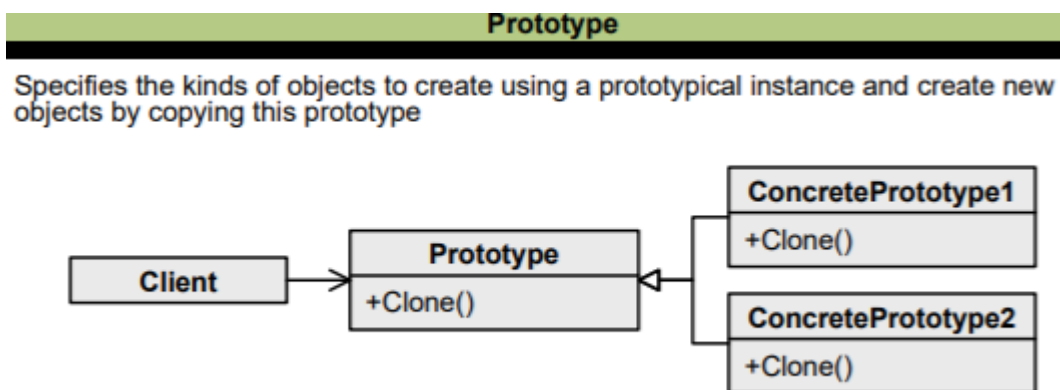
Provides a surrogate or placeholder for another object to control access to it



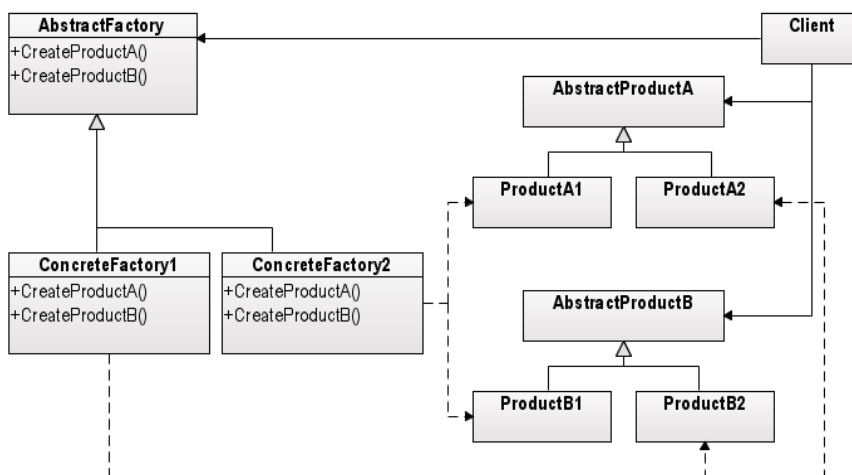
Design pattern : Observer



Design pattern : Prototype



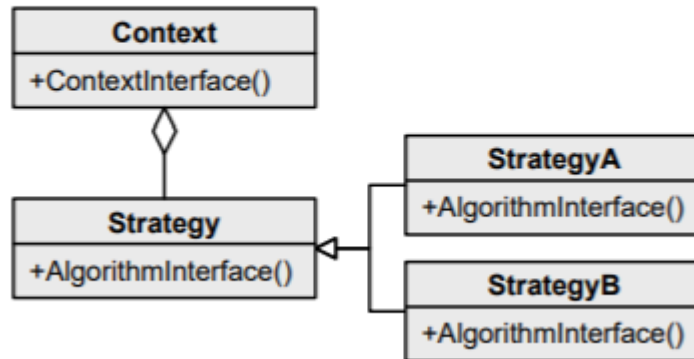
Design pattern : Factory



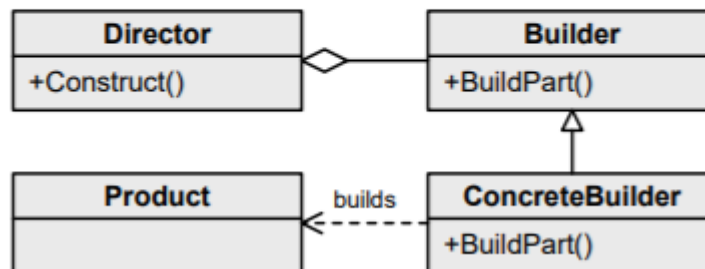
Design pattern : Strategy

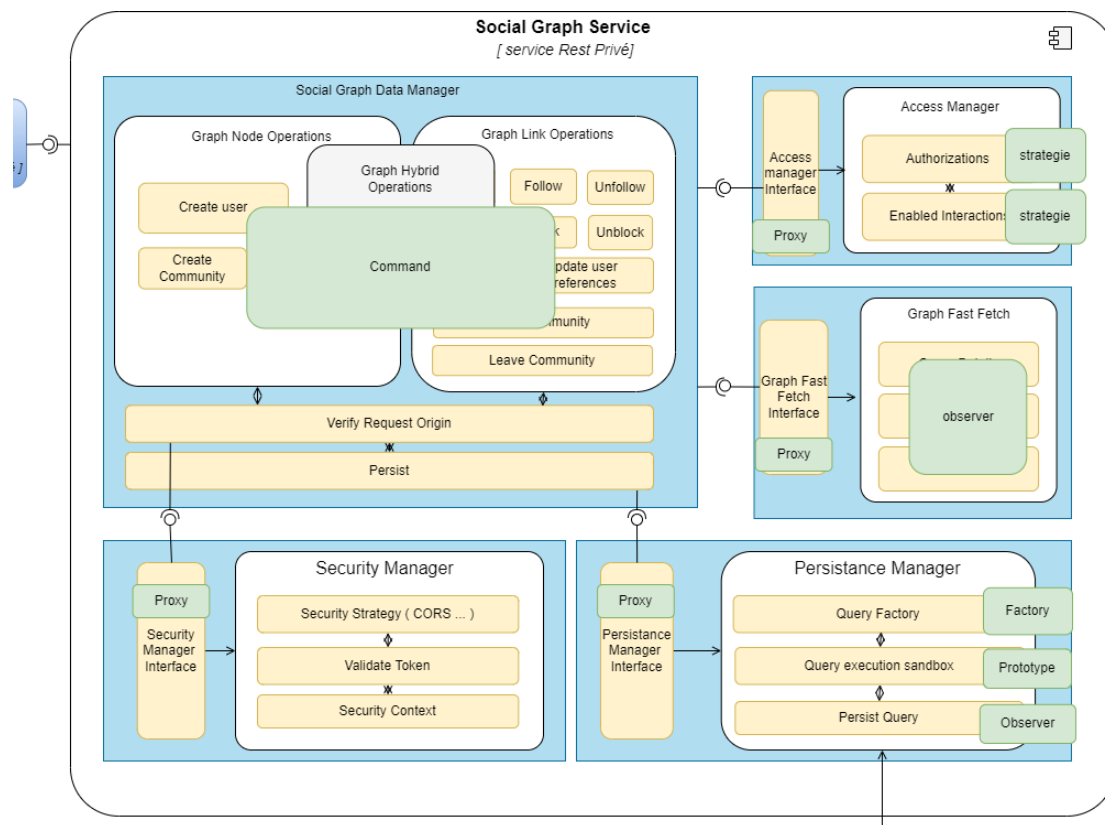
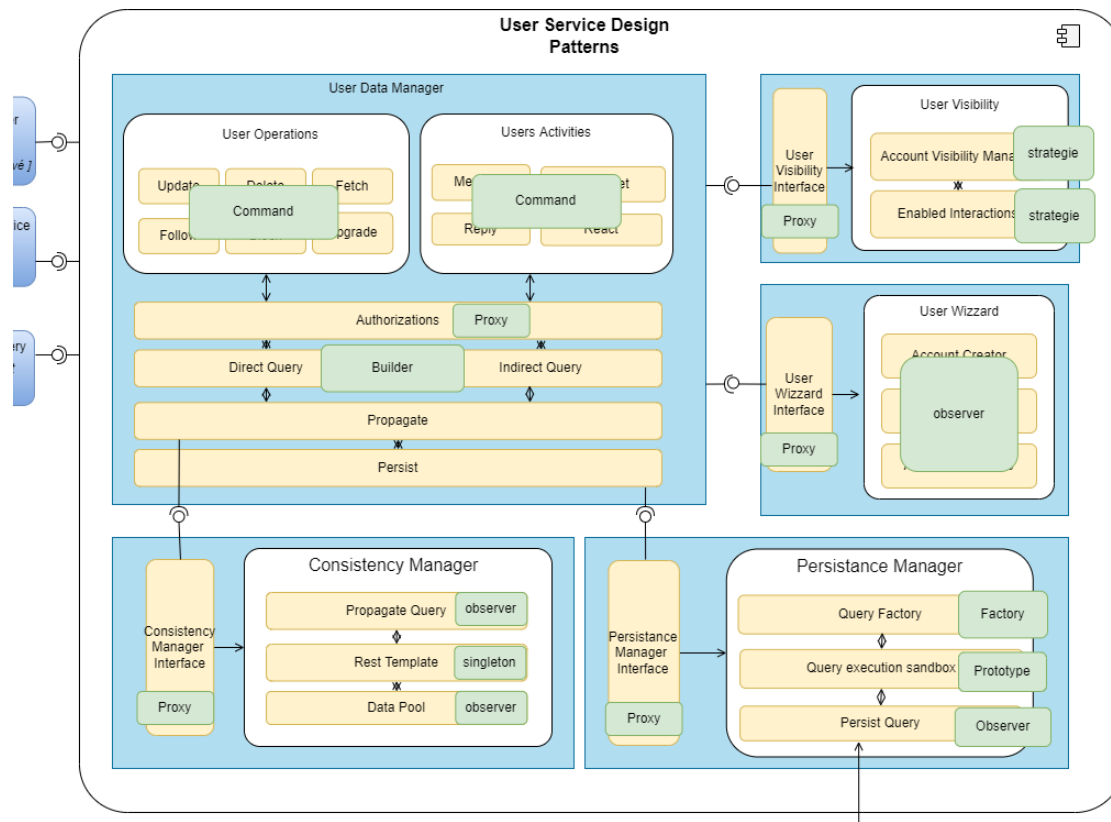
Strategy

Defines a family of algorithms, encapsulate each one, and make them interchangeable

**Design pattern : Builder****Builder**

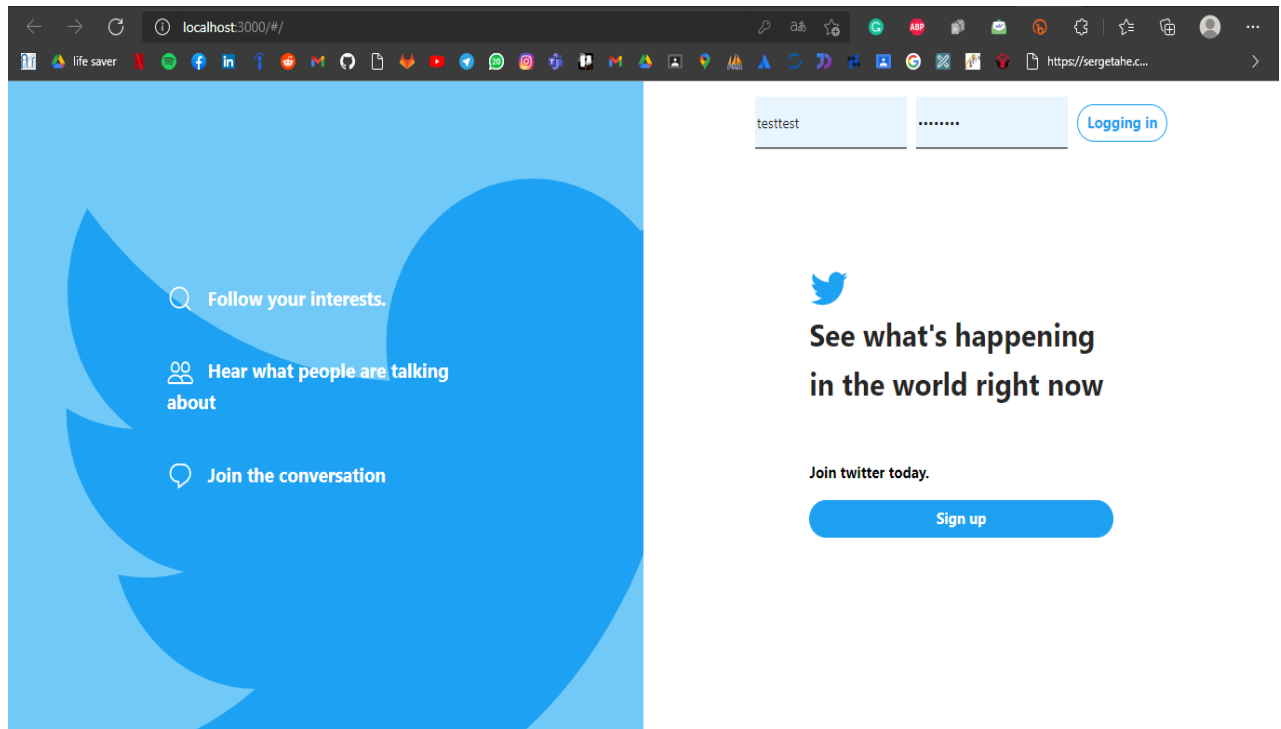
Separates the construction of a complex object from its representation so that the same construction process can create different representations.



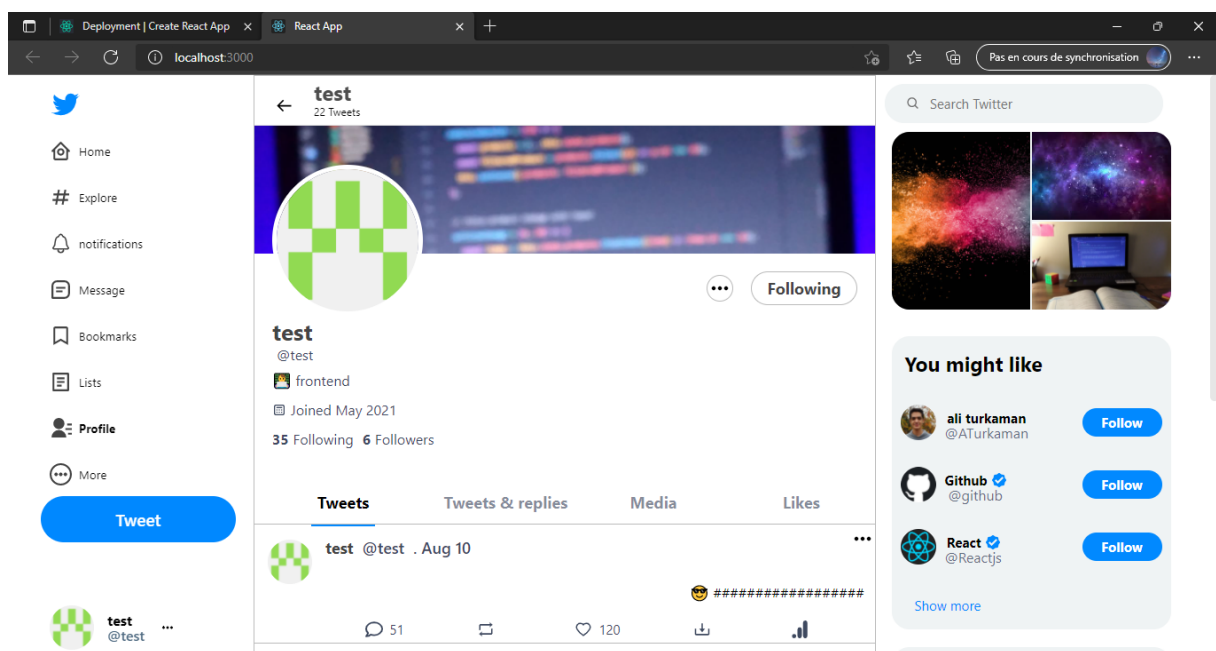


6- Réalisation et mise en oeuvre

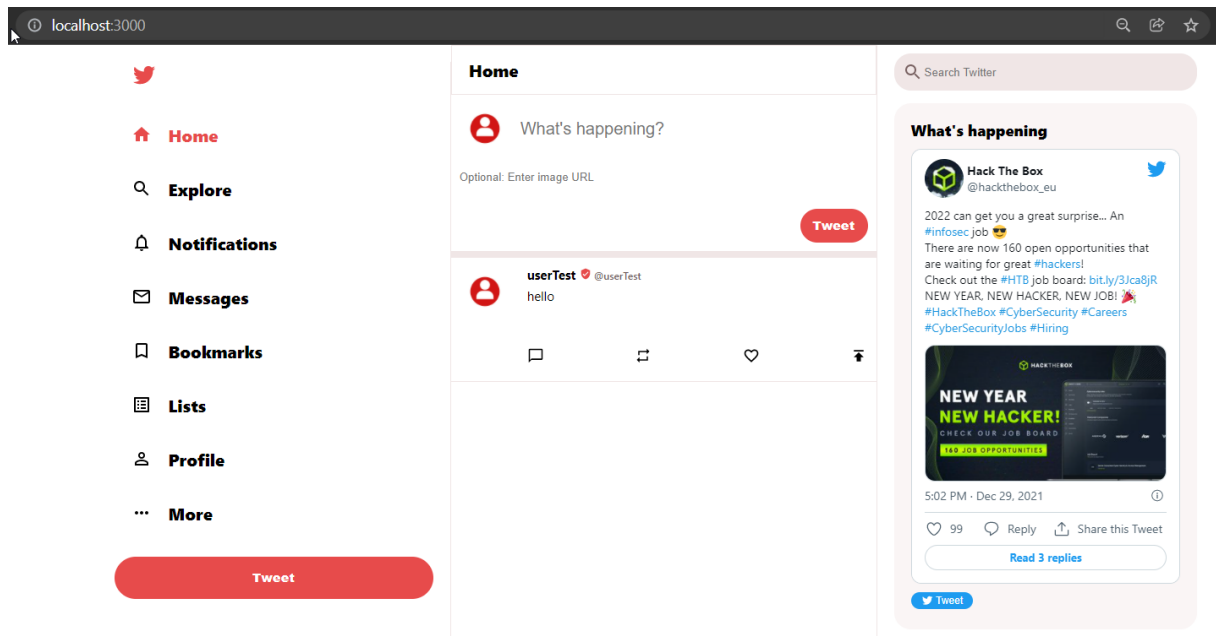
- Login page



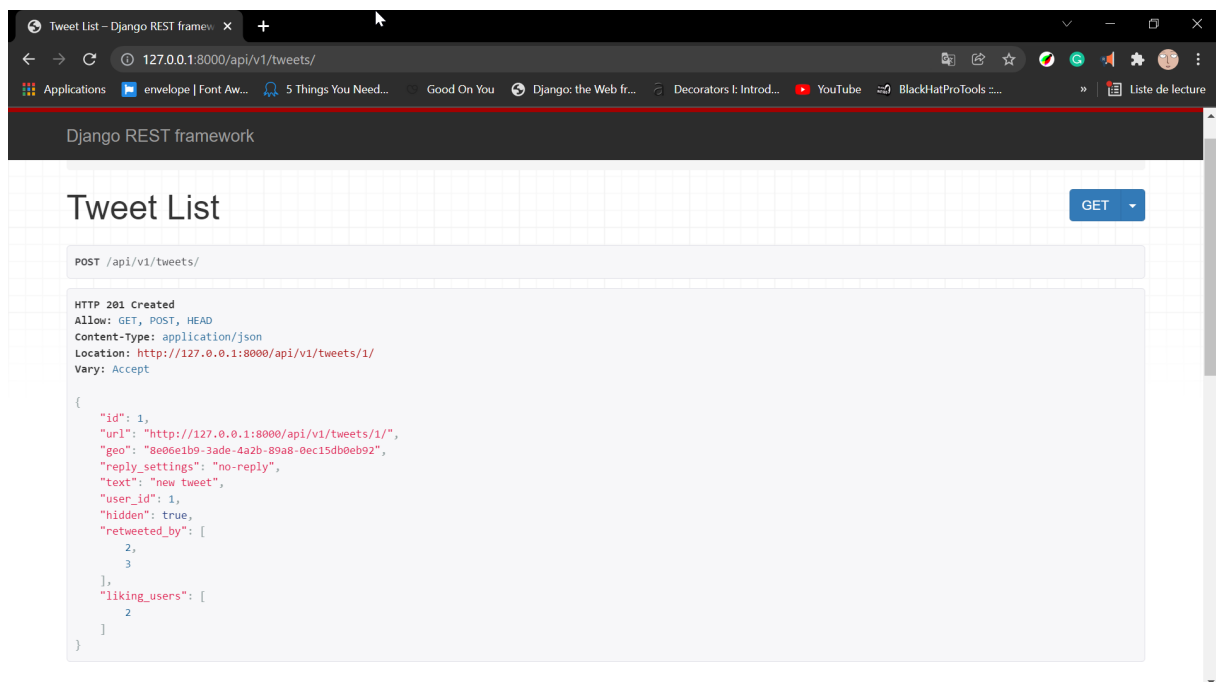
- User profil page



- User timeline page :



- Tweet API - Django:



Raw data HTML form

Geo

morroco(20.66,10.5)

Reply settings

no-reply

Text

new tweet

User id

ikram50

Hidden

☒

Retweeted by

ikram50
fulful60
oudaoud02

Liking users

ikram50
fulful60
oudaoud02

POST

- User API - Django:

127.0.0.1:8001/api/v1/users/

Applications | envelope | Font Aw... | 5 Things You Need... | Good On You | Django: the Web fr... | Decorators I: Introd... | YouTube | BlackHatProTools | Liste de lecture

Django REST framework

User List

Filters GET

GET /api/v1/users/


HTTP 200 OK
Allow: GET, POST, HEAD
Content-Type: application/json
Vary: Accept

[
 {
 "id": 1,
 "username": "fulful69",
 "name": "fatima ezzahra",
 "birthday": "1999-08-19",
 "bio": "hello me fulful",
 "liked_tweets": [],
 "following": [],
 "retweets": [],
 "blocked": [],
 "account_visibility": "",
 "disabled_interactions": ""
 }
]

Raw data HTML form

Raw data

HTML form

Username	<input type="text"/>
Name	<input type="text"/>
Birthday	<input type="text" value="jj/mm/aaaa"/> 
Bio	<input type="text"/>
Liked tweets	<div>TweetLink:52a92fd2-6de9-4a81-ab1a-344ea20f8f0a TweetLink:50462693-55a1-43e5-911d-8b0cb116d115</div>
Following	<div>fulful69</div>
Retweets	<div>TweetLink:52a92fd2-6de9-4a81-ab1a-344ea20f8f0a TweetLink:50462693-55a1-43e5-911d-8b0cb116d115</div>

Sachant que le service User ne persiste qu'un cache sur disque et RAM des tweets (sans relations) afin d'optimiser le lookup time. Dans le terminal vous pouvez voir les appels vers l'API tweet afin de consommer les tweets .

```
User Service
Calling Token Provider...
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
[30/Dec/2021 16:39:57] "GET /api/v1/users/ HTTP/1.1" 200 215
User Service
Calling Token Provider...
Not Found: /api/v1
[30/Dec/2021 16:40:00] "GET /api/v1 HTTP/1.1" 404 2913
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
[30/Dec/2021 16:40:00] "GET /api/v1/users/ HTTP/1.1" 200 215
[30/Dec/2021 16:40:03] "GET /api/v1/ HTTP/1.1" 200 5455
User Service
Calling Token Provider...
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
[30/Dec/2021 17:04:44] "GET /api/v1/users/ HTTP/1.1" 200 215
User Service
Calling Token Provider...
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
[30/Dec/2021 17:04:46] "GET /api/v1/users/ HTTP/1.1" 200 215
User Service
Calling Token Provider...
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
User Service
Calling Token Provider...
Waiting for Token Provider...
Token Provider: request TimedOut
Token Provider: call failed
```

on remarque le token provider n'a pas démarré donc on essayera de le spawner

- Token Provider - Node :

Le choix de Node ici est un peu particulier car on a dit que le drivers du choix Django est la sécurité et le Token Provider doit être sécurisé , par contre l'utilisation de l'API OAuth est plus facile et plus efficace sur Node que sur Django.


POST ▼ http://localhost:3000/generate

Params Authorization Headers (11) **Body** ● Pre-request Script Test

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ Gr

```
1 {
2   ... "userId": 96
3 }
```

Body Cookies (1) Headers (6) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1 {
2   "token": "ef0085ce40.a44e3dde9d.89f7db8572"
3 }
```

```
Token Provider
-----
Calling OAuth API...
Generating token for user 96
Encrypting the Payload...
Token: ef0085ce40.a44e3dde9d.89f7db8572 is ready!
```

```
Token Provider
{ jwt: '131ae04585.f65f42d0ea.e7fea00576' }
-----
Calling OAuth API...
Decrypting Token...
Invalid Token
```

POST

⌵

http://localhost:3000/validate

Params

Authorization

Headers (11)

Body ●

Pre-request Script

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

```
1 {
2   ... "jwt": "ef0085ce40.a44e3dde9d.89f7db8572"
3 }
```

Body

Cookies (1)

Headers (6)

Test Results

Pretty

Raw

Preview

Visualize

JSON ⌵

↻

```
1 {
2   "token": "7cadaea228.7549c70027.3b25106582",
3   "user": 96
4 }
```

```
Token Provider
{ jwt: 'ef0085ce40.a44e3dde9d.89f7db8572' }
-----
Calling OAuth API...
Decrypting Token...
Token is Valid current user is: 96
Refreshing Token
□
```

- **Connections:**