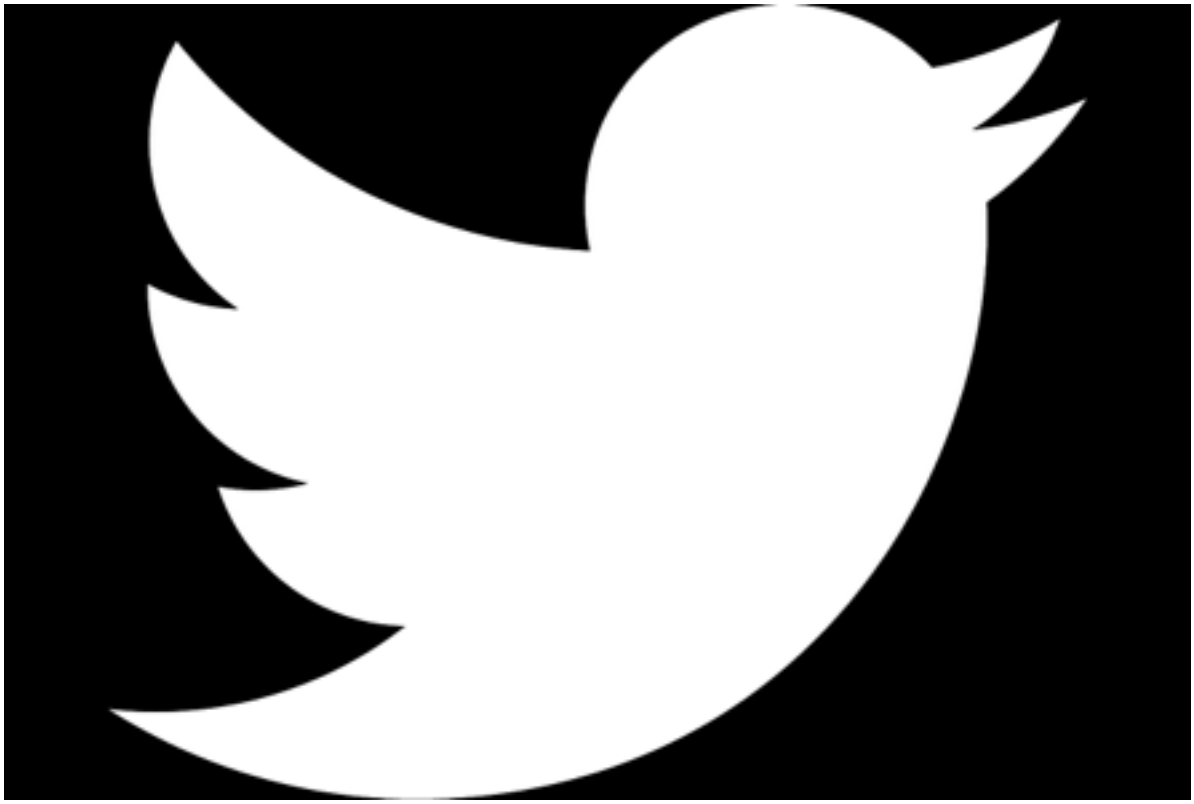
Compte rendu Touiteur

TP noSQL - Twitter-Like avec Redis



Maxime Bertheau

Pour le 23 avril 2015

http://github.com/maxoumime/Touiteur

Touiteur

Twitter-Like avec Redis

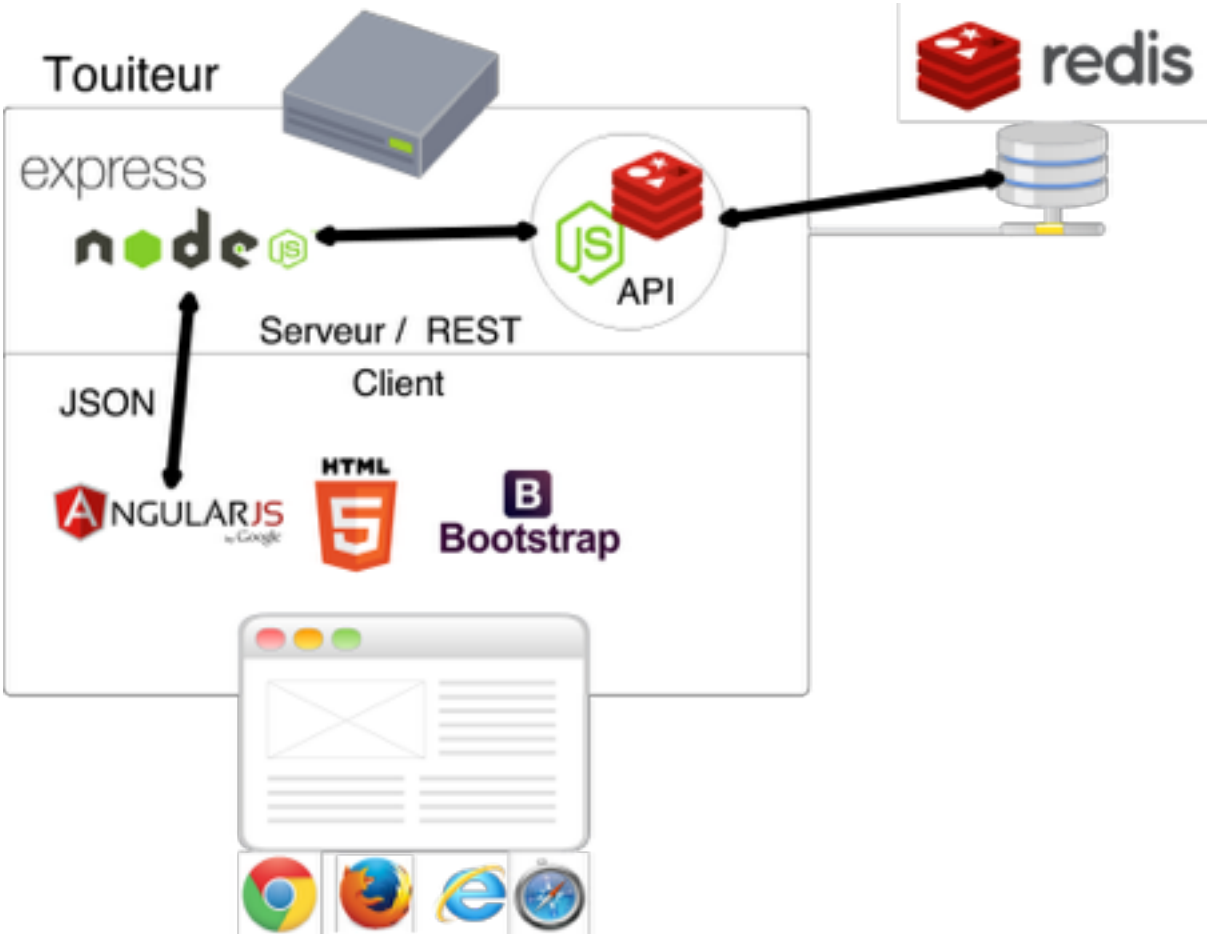
Présentation de l'application

Le but du TP était de développer une application permettant le même fonctionnement global que le réseau social Twitter, en utilisant la base de données clef-valeur Redis.

L'application Touiteur est le résultat de cette implémentation, elle est composée d'un back-end en NodeJS, et d'un front-end en AngularJS. Elle dispose des principales fonctionnalités tirées du réseau social de l'oiseau bleu:

- Création d'un utilisateur.
- Connexion d'un utilisateur enregistré.
- Déconnexion d'un utilisateur.
- Envoi d'un « touite », message à destination de la touitosphère.
- « Stalk » et « Un-stalk » d'un autre utilisateur, correspond au « follow » et le « unfollow » de Twitter.
- Listage des touites envoyés par les utilisateurs stalkés (stalkings) et par l'utilisateur courant, triés par date et utilisant un système de pagination.
- Affichage d'un résumé d'un utilisateur, avec ses touites, son nombre de stalkers, de stalking.
- Modification d'un utilisateur connecté.
- Suppression d'un utilisateur connecté.
- Recherche par mot-dièse # .
- Affichage d'utilisateur aléatoire, basé sur les stalking de l'utilisateur connecté.
- Affichage d'un mot-dièse aléatoire.
- Stockage des informations en base de données.
- Génération de tokens d'authentification.
- Chiffrage des mots de passe.

L'application est divisée en deux parties, le client et le serveur. Pour le client, on retrouve le framework Javascript MVC AngularJS, créé par Google, pour le serveur, on trouve aussi du Javascript avec NodeJS, utilisant le framework Express. La base de données utilisée étant Redis, on utilise un client Redis pour NodeJS.



L'intégralité des échanges entre le client (AngularJS) et le serveur (NodeJS) se fait par des appels REST sur HTTP, faisant transiter des données au format JSON. Voici les différents appels REST utilisables:

Tous les appels HTTP, sauf les appels utiles à la création et à la connexion d'utilisateurs doivent passer un token d'authentification.

POST /login

Authentifie l'utilisateur par son nom d'utilisateur et son mot de passe.

Pas de token.

GET /login/:token

Vérifie la connexion de l'utilisateur en passant le token stocké. Détermine si le token est encore valide.

POST /logout

Déconnecte l'utilisateur en utilisant son token.

GET /touite

Récupère les touites personnalisés d'un utilisateur, comportant ses touites ainsi que ceux de ses stalking.

Renvoie les 10 premiers résultats par défaut, il est possible de passer le paramètre « pagination » comportant un numéro de page pour récupérer la suite.

GET /touite/:idTouite

Récupère les détails d'un touite en particulier, en se basant sur ses identifiants.

POST /touite

Envoi d'un touite, dont le body est composé d'un authorId et d'un content.

DELETE /touite/:idTouite

Suppression du touite d'id idTouite.

GET /motdiese/:motdiese

Récupération des mots-dièse en rapport avec le motdiese donné.

GET /motdiese/random

Récupération d'un mot-dièse aléatoire.

POST /stalk/:idUser

Stalk d'un utilisateur d'id idUser.

DELETE /stalk/:idUser

Un-stalk d'un utilisateur d'id idUser.

GET /stalkers/:idUser

Récupération des stalkers (followers) de l'utilisateur d'id idUser.

GET /stalking/:idUser

Récupération des stalking de l'utilisateur d'id idUser.

GET /user

Récupération de l'utilisateur connecté.

GET /user/:idUser

Récupération de l'utilisateur d'id idUSer.

GET /user/touites/:idUser

Récupération des touites de l'utilisateur d'id idUser.

GET /user/available/:idUser

Détermine la validité du nom d'utilisateur idUser.

Pas de token.

POST /user

Ajoute un utilisateur d'username « id », de mot de passe « password », de mail « email » et de nom « name ».

Pas de token.

PUT /user

Mise à jour de l'utilisateur courant, en utilisant les mêmes informations que la requête précédente (sauf le username qui ne change pas).

DELETE /user

Suppression de l'utilisateur courant.

Les différents codes erreur retournés par NodeJS sont:

- Pas de résultat: **204**
- Formulaire / informations reçues invalide: **400**
- Accès non autorisé à une donnée: **401**
- Accès refusé à cette partie de l'application: **403**
- Élément introuvable: **404**
- Conflit (ex: tentative de stalk d'un utilisateur déjà stalké): **406**
- Email invalide (utilisé lors de l'inscription): **418**
- Erreur d'accès à la base de donnée: **500**
- Longueur trop importante (utilisé lors de l'envoi de touite): **509**

Structure du projet

Le projet contient à la fois la partie client et serveur. Voici l'arborescence du projet:

`.`	
`├── app.js`	Chargement des toutes principales de NodeJS
`├── bin`	Contient le fichier d'exécution du serveur NodeJS
`├── bower.json`	Fichier contenant les différentes librairies Javascript pour le client
`├── package.json`	Fichier contenant les différentes librairies Javascript pour NodeJS
`├── node_modules`	Dossier contenant les librairies pour NodeJS
`├── public`	**Dossier contenant la partie client de l'application (HTML)**
`│   ├── images`	Images utilisées dans l'application
`│   ├── index.html`	Layout principal de l'application, chargement de toutes les librairies
`│   ├── javascript`	Dossier contenant tout le code Javascript d'AngularJS
`│   │   ├── libs`	Dossier contenant les librairies Javascript du client
`│   │   ├── app.js`	Fichier de configuration principal de l'application AngularJS
`│   │   ├── directives`	Dossier contenant les directives de l'application (balises custom)
`│   │   ├── filters`	Dossier contenant les filtres utilisés dans l'application
`│   │   ├── login`	Dossier contenant les fichiers nécessaires au module de login (contrôleur, service, configuration du module, template HTML)
`│   │   ├── motdiese`	Module de recherche des mots-dièse
`│   │   ├── register`	Module d'enregistrement d'utilisateurs
`│   │   ├── touitetimeline`	Module correspondant à la timeline de l'application (accueil)
`│   │   └── user`	Module de gestion/affichage des utilisateurs
`│   └── stylesheets`	Feuilles CSS
`├── routes`	**Dossier contenant la partie serveur (NodeJS)**
`│   ├── http_constants.js`	Contient les différents codes HTTP utilisés
`│   ├── login.js`	Configuration de la route de login "/login"
`│   ├── logout.js`	Configuration de la route de logout "/logout"
`│   ├── motdiese.js`	Configuration de la route de gestion des mots-dièse "/motdiese"
`│   ├── stalk.js`	Configuration de la route d'actions de stalking "/stalk"
`│   ├── stalkers.js`	Configuration de la route de recherche de stalkers "/stalkers"
`│   ├── stalking.js`	Configuration de la route de recherche de stakning "/stalking"
`│   ├── touite.js`	Configuration de la route de gestion des Touites "/touite"
`│   └── user.js`	Configuration de la route de gestion des utilisateurs "/user"
`└── services`	**Dossier contenant les différents services d'interfaçage avec Redis**
`    ├── authService.js`	Service gérant toute la partie d'authentification
`    ├── db`	**Dossier contenant les fichier de gestion de la BDD**
`    │   ├── db.js`	Fichier de configuration principal de Redis, appels génériques, types
`    │   ├── hashdb.js`	Service de gestion des HASHes
`    │   └── setdb.js`	Service de gestion des SETs
`    ├── motdieseService.js`	Service gérant toute la partie mots-dièse
`    ├── touiteService.js`	Service gérant toute la partie Touites
`    └── userService.js`	Service gérant toute la partie utilisateurs

Structure des données stockées dans Redis

Utilisateurs

La représentation des utilisateurs est représentée comme suit:

On dispose d'un SET nommé « user », il contient les usernames (uniques) des différents utilisateurs.

```
"user" : ["username1", "username2"]
```

Ce SET permet de retrouver facilement les différents utilisateurs existants. L'utilisation des noms d'utilisateur comme suffixe évite l'utilisation d'UUID car l'on part du principe qu'ils sont uniques.

On dispose ensuite d'un HASH par utilisateur.

```
"user:username1":   {
  id: "username1", //Nom d'utilisateur
  password: "Z1ES768D9GH1F", //Mot de passe chiffré en AES256
  email: "username1@botler.me",
  name: "Utilisateur 1" //Nom affiché
  idTouites: "[\"e30469fd-fa30-4d44-8cb9-65a9ccdb7845\"]",
  idStalkers: "[\"username2\"]",
  idStalking: "[\"username2\"]",
}
```

idTouites, idStalkers et idStalking sont des tableaux qui ont étés « stringifyés » afin de pouvoir être stockés dans le HASH. Cette solution fonctionne bien qu'elle ne soit pas très correcte. Un autre solution aurait été de créer des clefs de la forme user:username1:idTouites qui auraient étés un SET.

Touite

De la même façon que le stockage des utilisateurs, le stockage des Touites comporte un set qui lui contient les IDs de tous les touites.

```
"touite":[
  "e30469fd-fa30-4d44-8cb9-65a9ccdb7845",
  "b2c161d1-a3fc-4c5b-9ebc-2de6d19681cb"
]
```

Contrairement aux utilisateurs, l'ID associé aux Touites est un UUID, généré spécialement pour un Touite, et permettant d'éviter le système d'auto-incrément, lourd en terme de temps et d'appels.

Concernant le Touite lui-même, il est représenté par un HASH:

```
"touite:b2c161d1-a3fc-4c5b-9ebc-2de6d19681cb": {
  "id": "b2c161d1-a3fc-4c5b-9ebc-2de6d19681cb", //ID du Touite
  "content": "Hey Touiteur #MyFirstTouite",//Contenu du Touite
  "authorId": "username1", //Auteur du Touite (username)
  "time": "1429532041962", //Timestamp d'envoi du Touite
  "motsdiese": "[\"myfirsttouite\"]" //Mots-dièses présents
}
```

Motdiese

Encore une fois, un SET nommé "motdiese" contient tous les mots-dièse présents:

```
"motdiese": ["myfirsttouite"]
```

Un SET est ensuite créé pour chaque Touite contenant le mot-dièse X. Ici chaque mot-dièse est unique.

```
"motdiese:myfirsttouite": [
  "b2c161d1-a3fc-4c5b-9ebc-2de6d19681cb"
]
```

Token

Les tokens servent pour l'authentification des utilisateurs, lors de la connexion de l'un d'entre eux, un token UUID est alors généré puis renvoyé. Une entrée est alors créée dans Redis, sous la forme token:UUID. Il s'agit d'une simple entrée clef-valeur, ayant pour valeur le nom d'utilisateur de la personne connectée. Chaque token a une durée de vie gérée par Redis, égale à 1 heure.

```
"token:2cd2f7f8-4c6e-4dfd-bb73-27c1ebe58f47": "username1"
```