Projet IHM

Étudiant :
NIANG Abdoul Aziz

Année : 2018-2019

15 janvier 2019

Table des matières

Introduction

Ce projet s'inscrit dans le cadre du module *IHM-visualisation de données* de notre formation. Il consiste à récupérer une liste de Tweets, puis de les stocker dans une base de données *MongoDB*. Et pour finir, il fallait proposer des fonctionnalités d'export et des visualisations sur cette liste de tweets. Dans le corps de ce rapport, nous allons en premier lieu aborder les différents outils utilisés. En deuxième lieu, les réalisations seront présentées. En dernier lieu, nous ferons un bilan sur l'outil mis en place.

I Outils utilisés

I.1 MongoDB

Le système de gestion de base de données **MongoDB** est une système orienté documents. Contrairement aux bases de données traditionnelles, cet outil ne nécessite pas un schéma prédéfini des données. Il propose une partition très large de ses données. En effet, ses données peuvent être réparties sur un nombre quelconque de machines. Il fait partie des catégories de SGBD qu'on appelle *NoSQL*.

I.2 Graphql

Graphql est un langage de requêtes développé par Facebook en 2012 et publié en 2015. Il permet d'interroger une base de données sans nécessairement connaître la structure des données de la base. La réponse obtenue suite à une requête est au format JSON . Ce langage est implémenté pour de nombre langages tels que Python, JavaScript, etc.

I.3 NodeJS

NodeJS est une plateforme logicielle libre et événementielle en JavaScript. Il a été créé en 2009 par Ryan Dahl . Il permet d'exécuter du JavaScript coté serveur pour effectuer de opérations. Il permet donc de se passer de serveurs web tels que Nginx ou Apache lors du déploiement de sites et d'applications web développés avec NodeJS.

I.4 ReactJS

ReactJS est une bibliothèque JavaScript développé en 2013 par Facebook. Il représente dans une application ce que la vue représente au modèle MVC. Il vise principalement à faciliter la création d'application web monopage en créant des composants qui dépendent de l'état général de la page. Il est utilisé par bon nombre d'applications connues comme Netflix , Yahoo , etc.

I.5 Bootstrap

Bootstrap, qui a été développé par Twitter, est un framework côté client. Ce framework est l'un des plus populaire de ceux associés au langages HTML, CSS et JavaScript. Il permet de structurer le design d'un site web de manière

à ce que ce dernier s'adapte à tout type d'écran. Il comporte un système de grille simple et efficace pour mettre en ordre l'aspect visuel d'une page web.

I.6 CharteJS

Chart.js est bibliothèque open source JavaScript. Elle permet de représenter des données sous forme de graphes statistiques.Elle est crée par *Nick Downie*.C'est une bibliothèque simple d'utilisation avec beaucoup d'option de personnalisation. Elle est compatible avec tous les navigateurs modèles supportant HTML5.

I.7 D3JS

La bibliothèque graphique JavaScript **D3.js** permet l'affichage de données sous une forme dynamique et numérique. Pour la visualisation de données, elle utilise généralement les technologies *SVG*, *JavaScript* et *CSS*.

I.8 Quelques autres langages

Quelques autres langages ont aussi été utilisés pour réaliser l'application. Ce sont les langages **HTML**, **CSS** et **JavaScript**.

II Réalisations

II.1 Mise en place des données

En se servant du script *query.sh* de l'archive *tweetEndPoint* fournit sur Ecampus, on récupérer 1500 Tweets de l'utilisateur qui a pour nom d'utilisateur *FIFAWorldCup* au cours de la période du 14/06/2018 au 15/07/2018. Ce qui se traduit par la ligne de commande ci-après.

```
1 $ ./ query.sh username FIFAWorldCup until 2018 -07 -15 since
    2018 -06 -14 maxtweets 1500
```

Remarque :

Il faut préalablement être placé dans la dossier *tweetEndPoint* fournit sur ecampus avec l'invide de commande, pour pouvoir faire fonctionner la commande.

Après cette opération, on obtient un fichier *JSON* composer d'un élément de type *JsonArray*. Cet élément contient les 1500 Tweets. Chaque Tweet est constitué des attributs suivants :

- username
- geo
- emoticons
- date
- mentions
- emoticonsStr
- replies
- hashtags
- replyings
- retweets
- id
- replyingTo
- likes
- permalink
- text

Puis, après avoir créer une base de données *MongoDB* qui s'intitule *worldCupTweetsDB*, et créer une collection nommée *tweets*, on importe les données précédemment récupérer dans ladite collection.

```
$ mongoimport --db <nom_BD> --collection <nom_collection> --
    file <nom_fichier> --jsonArray
```

On aura ainsi, à disposition, une collection dans notre base de données contenant les 1500 tweets qui sera prête à être exploitée.

II.2 Mise en place du serveur

A fin de communiquer et récupérer des données depuis la base de données, un serveur *NodeJS* a été mise en place. En effet, ce serveur utilise l'objet *MongoClient* de la dépendance *mongodb* pour se connecter à la base de données. On a ci-dessous le code qui permet d'établir la liaison vers notre base de données.

```
var mongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
var Tweets;
mongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("worldCupTweetsDB");
  dbo.collection("tweets").find({}).toArray(function(err,
     result) {
    if (err) throw err;
    Tweets = result;
    db.close();
  });
});
```

Remarque :

On observe que l'URL de connexion vers la base de données est *mongodb ://localhost :27017/* et qu'on se utilise la collection *tweets* de la base de

données *worldCupTweetsDB*.

On crée ainsi un résolveur nommé *tweets* qui va schématiser le résultat des requêtes faites vers la base de données. La structure des données de ce résolveur est le suivant :

```
type TweetQuery{
    id: String
    username: String
    date: String
    replies: String
    retweets: String
    likes: String
    geo: String
    mentions: String
    hashtags: String
    permalink: String
    emoticons: String
    emoticonsStr: String
    replyings: String
    replyingTo: String
    text: String
}
```

II.3 Mise en place du front-end de l'application

L'application comporte quatre pages à savoir une page qui liste les différents tweets avec certains de leurs données, une autre qui permet de visualiser le nombre de Tweets par jour, une autre page qui visualise les différents Tweets en tenant compte de leurs retweets et une dernière qui montre le nombre d'émoticônes utilisés par jour. La navigation entre les différentes page de l'application est gérée par les éléments *BrowserRouter* et *Route* de la dépendance *react-router-dom*.

```
  <Router>
    <ApolloProvider client={client}>
        <Route exact path="/" component={List} />
        <Route path="/Statistics" component={Statistics} />
        <Route path="/retweets" component={Retweets} />
        <Route path="/emoticons" component={Emoticons} />
      </ApolloProvider>
    </Router>
```

Les lignes de code ci-dessus représente le rendu du composant ("component") *App* qui est sollicité au démarrage de l'application. Ainsi, suivant le lien souhaité par l'utilisateur, un composant sera appelé.
On note aussi l'utilisation du composant *ApolloProvider* de *react-apollo* qui

permet aux autres composants de l'application de bénéficier des services de l'objet *ApolloClient* qui rend possible l'utilisation des données de la base de données.

```
const client = new ApolloClient({
    uri: "http://localhost:3002/graphql"
});
```

II.3.1 Le composant List

Ce composant récupère la liste de tous les tweets de la base en servant de la requête Graphql ci-dessous et du composant *Query* de *react-apollo*.

```
const TWEETS = gql`
        {
          tweets{
          id
          date
          replies
          retweets
          likes
          hashtags
          replyings
          text
          }
        }
      `;
```

Puis il les affiche partant du Tweet le plus récent au Tweet le moins récent. Ci-dessous, on a un exemple d'affichage de Tweets de la base de données.

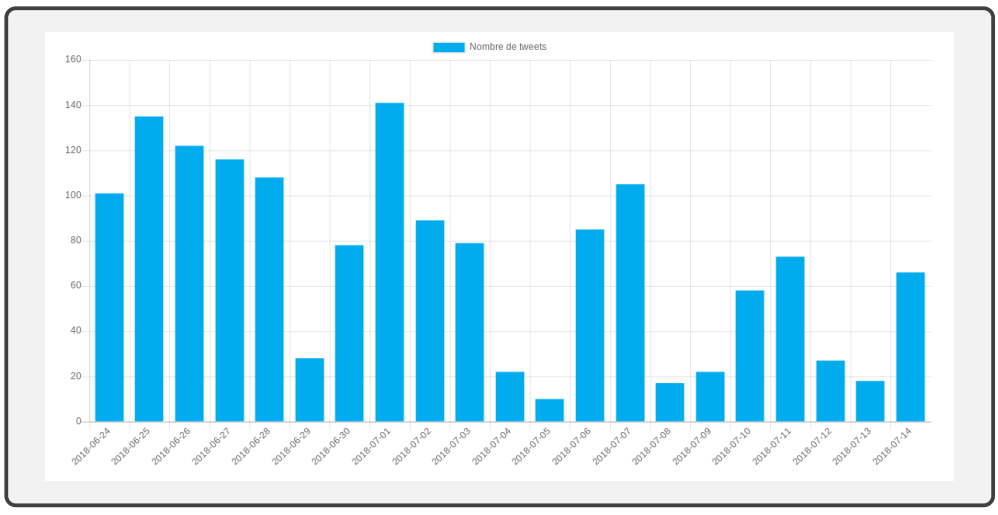
FIGURE 1 – Liste de Tweets

II.3.2 Le composant Statistique

Ce composant permet de visualiser le nombre de Tweets par jour. En effet, il récupère la liste les différents Tweets avant de transformer cette liste en calculant pour chaque date le nombre de Tweets associés. Les résultats obtenus seront visualisés sous forme d'histogramme.
Le composant *Statistique* utilise le composant *Barchart* qui se charge de constituer l'histogramme en servant de l'outil JavaScript *Chart.js* une fois que les données sont chargées. Pour ce ledit composant fait appel à l'objet **Chart** et le modèle de représentation de type **bar** comme suit :

```
1     let barChart = new Chart(ctx, {
2       type : 'bar',
3       data : {
4         labels : this.getDays(dailyTweets).reverse(),
5         datasets : [
6           {
7             label : "Nombre de tweets",
8             backgroundColor : "#00ACED",
9             data : this.getDailyNumberOfTweets(dailyTweets).
                  reverse()
10          }
11        ]
12
13      },
14      options : {}
15    });
```

On obtient ainsi le résultat ci-dessous.



II.3.3 Le composant Retweets

Ce composant gère l'affichage des Tweets en tenant compte du nombre retweets de chaque Tweet. Après récupération de la liste de Tweets, on invoque un autre composant nommé **BubbleChart** qui, comme son nom l'indique, dessine un Bubble Chart. Chaque bulle représente un tweet et le rayon d'une bulle est proportionnel au nombre de retweets du Tweet qu'il représente. Le composant **BubbleChart** utilise la technologie *D3.js*.

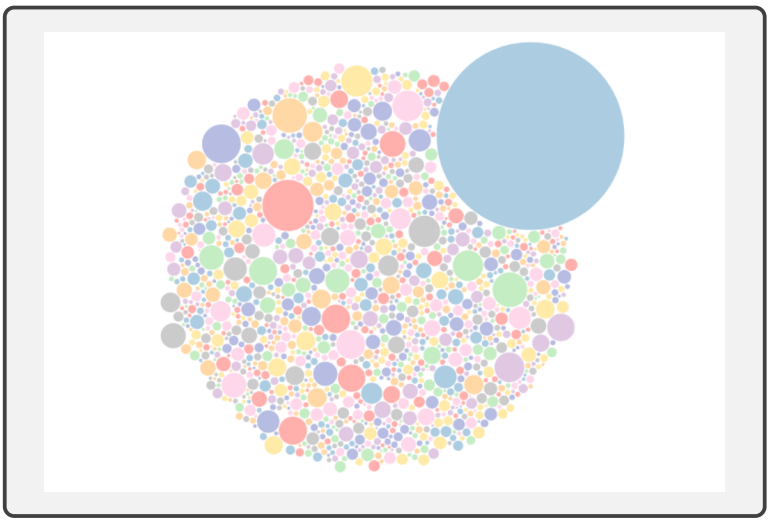
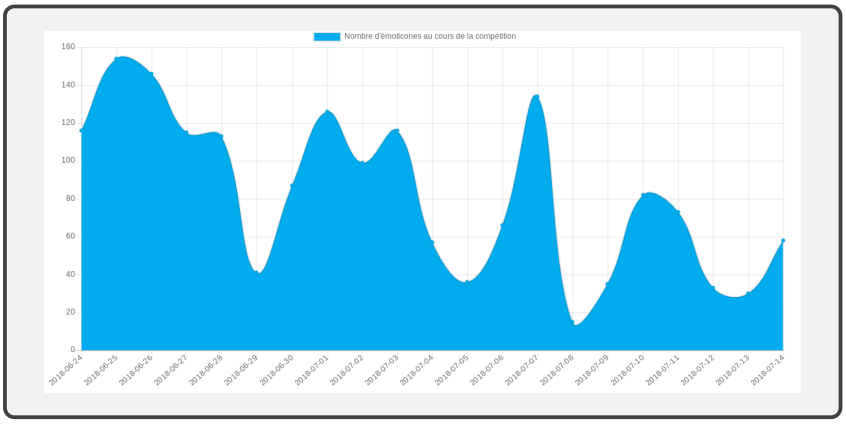


FIGURE 2 – Différents tweets en fonction de leurs retweets

II.3.4 Le composant Emoticons

Ce composant se charge de visualiser le nombre d'émoticônes lors des Tweets. Il récupère les données et les transmet à un **LinChart** qui utilise aussi la technologie *Chart.js* pour dessiner un *'line chart'*. la figure ci-dessous représente le résultat qu'on va obtenir.



III Bilan

Dans la version actuelle de l'application, on peut la lancer et naviguer entre les différentes pages. Toutes les technologies utilisées fonctionne comme il se doit de fonctionner. On a ci-dessous les différentes captures des différentes pages de l'application.

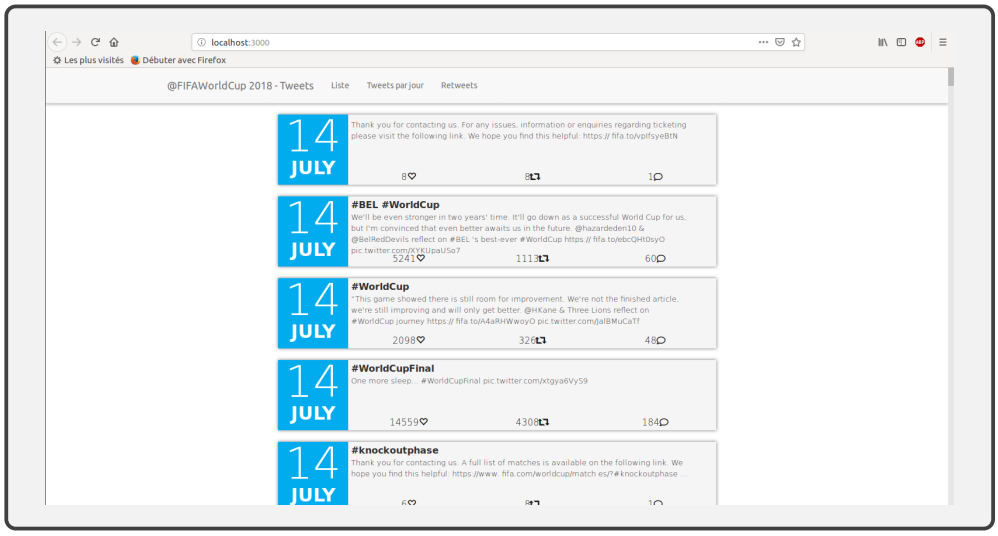


FIGURE 3 – Page listant les tweets

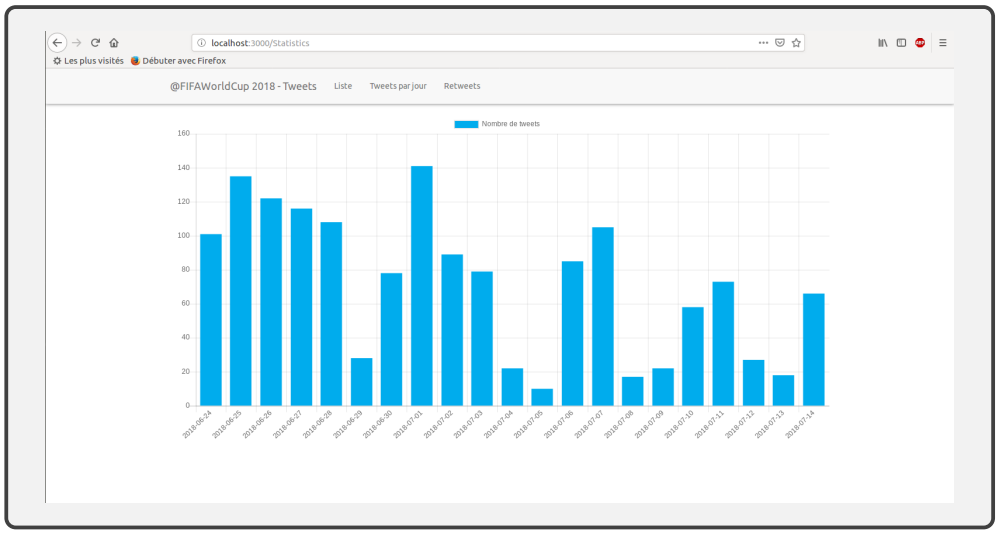


FIGURE 4 – Page visualisant le nombre de tweets par jour

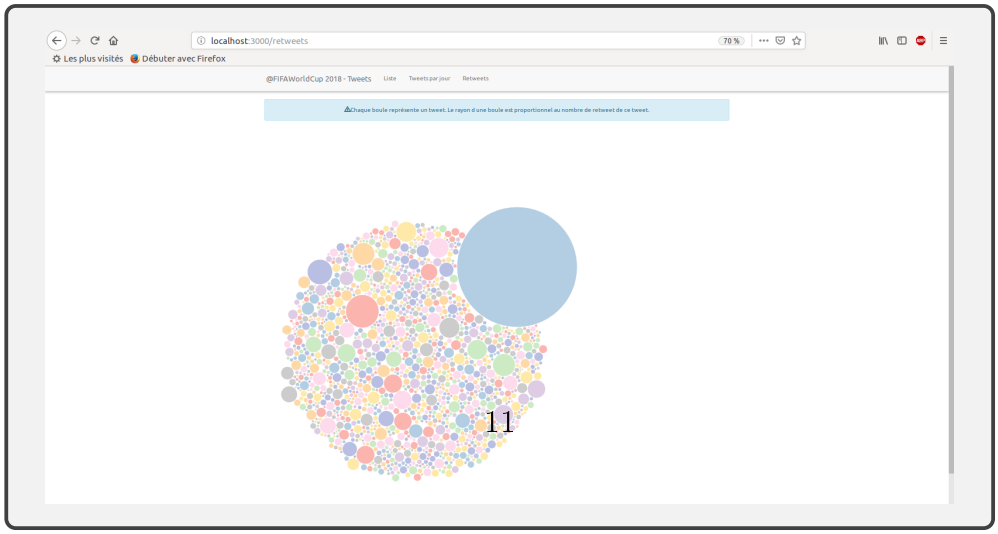


FIGURE 5 – Page représentant les tweets en tenant compte des retweets

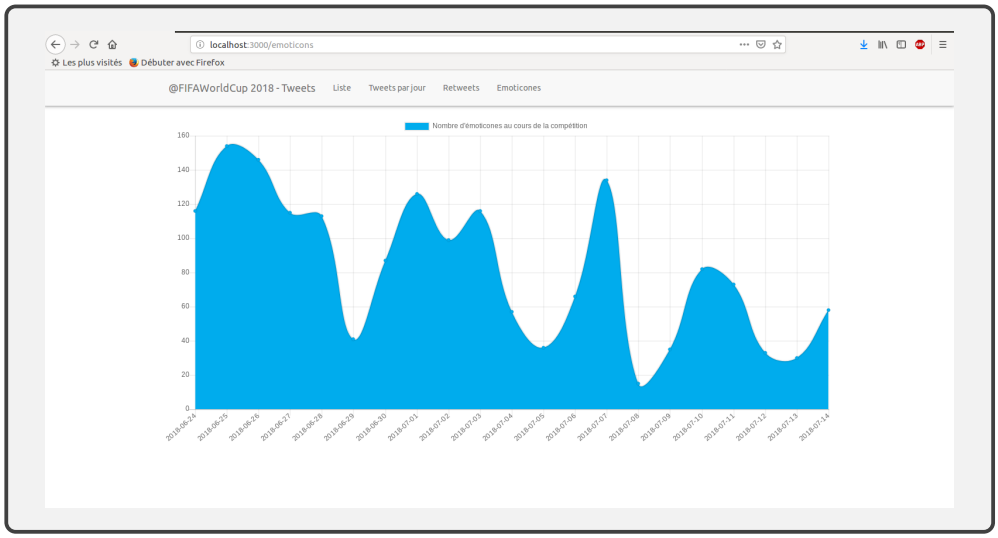


FIGURE 6 – Page représentant les tweets en tenant compte des retweets

Conclusion

En définitive, ce projet été très enrichissant pour moi. En effet, il m'a permis de gagner en compétences sur les technologies telles *MongoDB*, *ReactJS*, *NodeJS*, *D3JS* et *ChartJS* qui sont, de nos jours, des technologies très utilisées dans le monde du développement web.

Webographie

https://fr.wikipedia.org/
https://reactjs.org/
https://www.apollographql.com/
https://www.chartjs.org/
https://d3js.org/
https://openclassrooms.com/fr/courses/1915371-guide-de-demarrage-pour-utiliser-m