SCALA - Tweet Analyser

Professeur : Nastaran Fatemi

Assistants :
Miguel Santamaria
Maxime Lovito

Auteurs :
Dejvid Muaremi
Mentor Reka
Xavier Vaz Afonso

Département TIC

8 juin 2019

Table des matières

Chapitre 1

Introduction

Tweet Analyser est une application web qui permet d ?analyser les sentiments provenant des tweets sur Twitter. Il permettrait par exemple à une entreprise d ?analyser les réponses du tweet le plus récent et d ?avoir un sentiment général vis-à-vis de celui-ci. Il est également possible de visualiser ces résultats avec l'aide d'un graphe. Réalisé lors du cours à option de SCALA, qui a lieu lors du dernier semèstre pour les étudiants en TIC. La durée de développement du projet fût de 5 semaines avec un délai de remise au 10.06.2019. Dans le cadre de la phase de projet, l'équipe de développement est composée par Dejvid Muaremi, Mentor Reka, Xavier Vaz Afonso.

Chapitre 2

Objectifs

Créé le 21 mars 2006 et lancé en juillet de la même année, Twitter est une application permettant à ses utilisateurs de publier gratuitement des messages de 280 caractères, également appelé tweets, sur internet dont la durée de vie est, en moyenne, d'une heure. En 2018, Twitter avait plus de 300 millions d'utilisateurs actif par mois et plus de 500 millions de tweets envoyés par jour.

Actuellement, ce sont les personnalités publique qui envoient le plus de tweets. On retrouve principalement des chefs d'états, des artiste, mais aussi des chefs d'entreprise, des entreprises, et des chaînes de télévision. Twitter n'est désormais plus utilisé uniquement pour envoyé de simples messages mais aussi en tant que plateforme d'information et d'échanges lors de certains événements comme par exemple l'Eurovision qui a fait couler beaucoup de bits. Aussi, il existe déjà de nombreuses entreprises qui proposent un service de support passant par ce médium.

Une entreprise qui publie énormément aura forcément besoin de connaître l'avis général et la réaction de ses abonnés et autres utilisateurs par rapport à leurs tweets. Il lui serait également intéressant de connaître la manière dont sont accueilli leurs nouveaux produits mais aussi les réactions aux tweets des membres de l'entreprises.

Chapitre 3

Conception et architecture du projet

3.1 Téchnologies et outils utilisés

3.1.1 React

La bibliothèque JavaScript libre développée par Facebook et une communauté de développeur indépendant. Le but principal de React est de faciliter la création de composants pour les interfaces utilisateur. Ces composants dépendent d'un état et lorsque celui-ci change, une nouvelle page HTML est générée. En plus d'avoir eu le cours de TWEB lors du premier semèstre de troisième année, certains de nos membres avaient déjà de l'expérience et de la facilité avec cette bibliothèque.

3.1.2 Scala

Conçu à l'École polytechnique fédérale de Lausanne (EPFL), Scala est un langage de programmation multi-paradigme, il combine la programmation orientée objet et la programmation fonctionnelle. Son but est d'exprimer les modèles de programmation courants dans une forme concise et élégante.

3.1.3 Scala-Play

Basé sur Akka, Play est un framework web open source qui permet de rapidement et facilement créer des applications web basée sur Java. La particularité de Play est de ne pas être basé sur le moteur Java, ce qui en fait un moteur plus simple et plus puissant lors d'une application web. Depuis la version 2.0, le framework a subit une refonte et a été écrit en Scala, le build et le déploiement ont été migré sur du SBT.

3.1.4 Slick (database)

Scala Language-Integrated Connection Kit est une librairie de type Functional Relational Mapping (FRM) qui permet de facilement accéder à une base de données avec du Scala. Cette librairie permet de traiter les données comme s'il s'agissait d'une collections Scala tout en laissant aux développeurs, un contrôle sur les accès à la base de données. Les données sont traitées de manière asynchrone ce qui permet de facilement l'intégrer dans un projet Scala-Play. L'un des avantages principal de Slick est qu'il est Typesafe et permet donc un traitement correcte des données selon le modèle.

3.1.5 MySQL

Le système de gestion de bases de données relationnelles (SGDBR), faisant partie des logiciels de gestion des bases de données les plus utilisés au monde et que l ?entièreté de notre équipe connaît avec suffisamment de détails pour être facilement utilisé dans un projet de cette envergure. De plus, il est recommandé de l'utiliser dans un projet utilisant Slick.

3.1.6 Docker

Le logiciel libre qui automatise le déploiement d ?application dans des conteneurs logiciels. Ces conteneurs sont isolés et peuvent être exécutés sur n ?importe quel système qui prend en charges Docker. Principalement utilisé lors de la phase de développement, il permet de créer un conteneur MySQL qui sera le même pour chaque membre du groupe et ce facilement.

3.1.7 Git

Le gestionnaire de version décentralisé libre, que l'on a choisi d'utiliser afin de gérer la totalité du projet ainsi que ses différentes versions.

Nous avons créé utilisés des branches afin d'implémenter les nouvelles fonctionnalités une fois celle-ci prête elles sont envoyée dans une copie temporaire de la branche Master afin de s'assurer du bon fonctionnement du projet. Une fois les modifications acceptées, celle-ci sont envoyée sur la branche Master.

3.1.8 GitHub

Le service web permettant de parcourir graphiquement l'historique Git du client et du serveur et qui nous a également permis d'héberger nos sources en ligne. GitHub offre également de nombreux outils de gestion de projet qui sont intéressantes lorsque l'on doit travailler en équipe.

3.2 Architecture du client

Nous avons choisi d'utiliser React pour le client, de par nos expériences personnelle avec la librairie. React, l'oblige, le projet client est structuré en composant.

Un répertoire page qui contient les composants des pages de **Login**, **Register**, **Home**, **Analyse**, et le **Header**.

Un répertoire Components qui contient les **Canvas** utilisant la librairie permettant de créer des graphes, le composant **Graph**, qui contiendra le Canvas. et pour finir le composant **Post** qui lui contient les rapport d'analyse.

Un répertoire utils qui contient un composant qui utilise le contexte de React pour partager des informations entre les composant comme nous l'avons appris lors du cours de TWEB. Pour ce qui est des requêtes Axios,elles sont contenues dans le fichier **user.service**.

Le fichier App.js le programme principal, s'occupe de rediriger l'utilisateur selon ses autorisations de la page courantes vers les pages désirées tout en faisant le lien parmi les différents composants vu ci-dessus.

3.3 Architecture du serveur

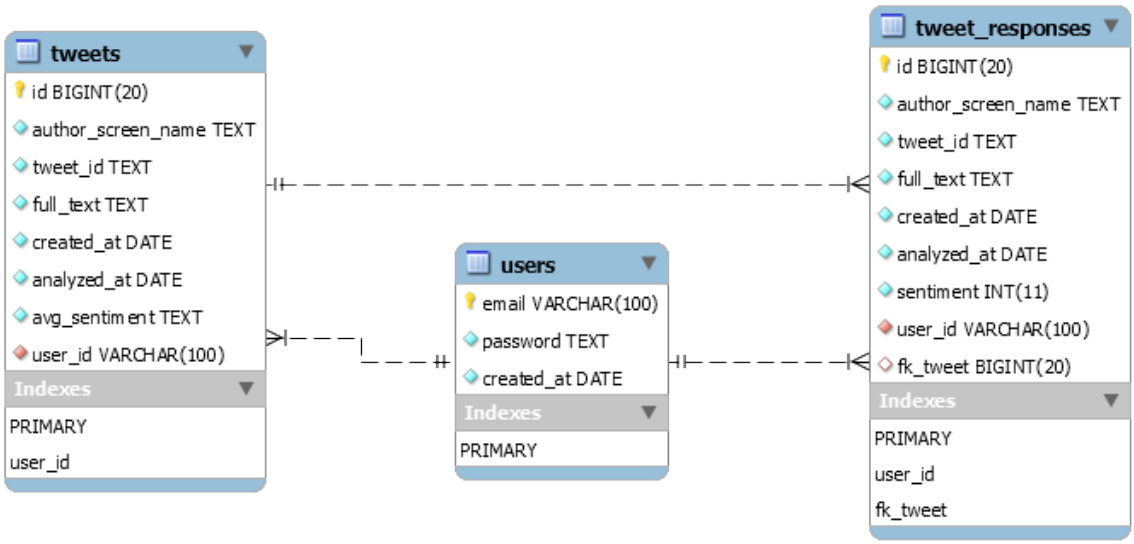
Pour ce qui est du serveur, nous avons gardé la structure de base du projet tel qu'il a été fournis par Miguel Santamaria en y intégrant nos fonctionnalités, à savoir, pour l'analyse de sentiments. Plus précisément, nous retrouvons les éléments suivants :

- **controllers** : Comme son nom l'indique, il contient les différents contrôleurs du projet.
- **filters** : Contient les différents filtres qui peuvent être appliqué au données.
- **models** : Tout simplement les modèles de données qui seront stocké dans la base de données que nous traiterons un peu plus loin.
- **repositories** : Permet de faire le lient entre les données et la base de données.
- **services** : Les différents services offerts par notre serveur, dont l'analyse de sentiments.
- **views** : Les vues HTML, offertes par notre serveur.

3.3.1 Définition de la base de données

Schéma de la base de données

En nous basant sur les données fournies par l'API de Twitter, nous avons construits nos modèles en ciblant la simplicité et la clarté.



Dans l'objectif d'avoir les données nécessaire pour récupérer les sentiments d'un Tweet nous avons besoin uniquement des tables ci-dessus.

User

Cette table contient la liste des utilisateurs pouvant utiliser notre application. Chaque utilisateur est identifié par un email unique et possède un mot de passe ainsi qu'une date de création.

Tweet

Cette table contient la liste des Tweets qu'un utilisateur souhaite analyser, un utilisateur peut analyser un à plusieurs Tweets et chaque Tweets est analysé par un seul et unique utilisateur. Chaque Tweets est défini par un id unique et possède un auteur, le texte du Tweets, une date de création, une date d'analyse, et un ressentiment moyen.

TweetResponse

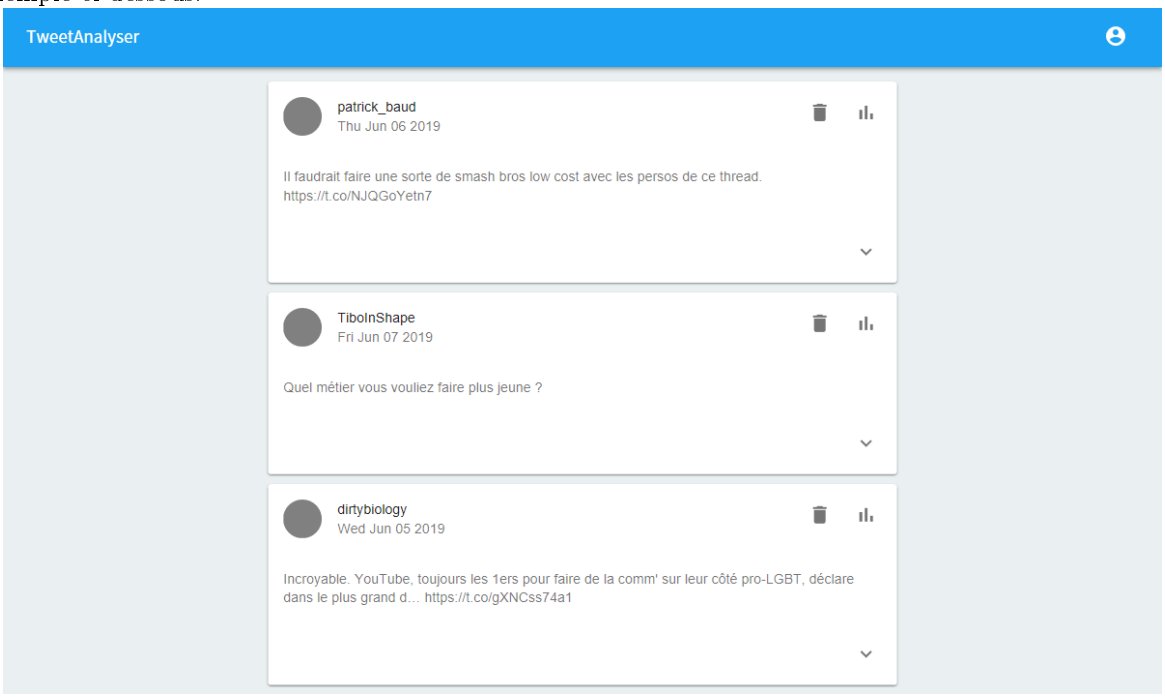
Cette table contient la liste des réponses à un Tweets, leur texte sera utilisé dans l'analyse de sentiment. Un Tweets peut posséder entre une et plusieurs réponses et chaque réponses est lié à un et un seul Tweets. Tout comme les Tweets, les réponses possèdent un id, un auteur, un texte qui sera analysé, une date de création, une date d'analyse, et un ressentiment moyen. De la même manière qu'un Tweet peut intéresser une entreprise, les réponses aux Tweets d'autrui peuvent aussi avoir un intérêt.

Chapitre 4

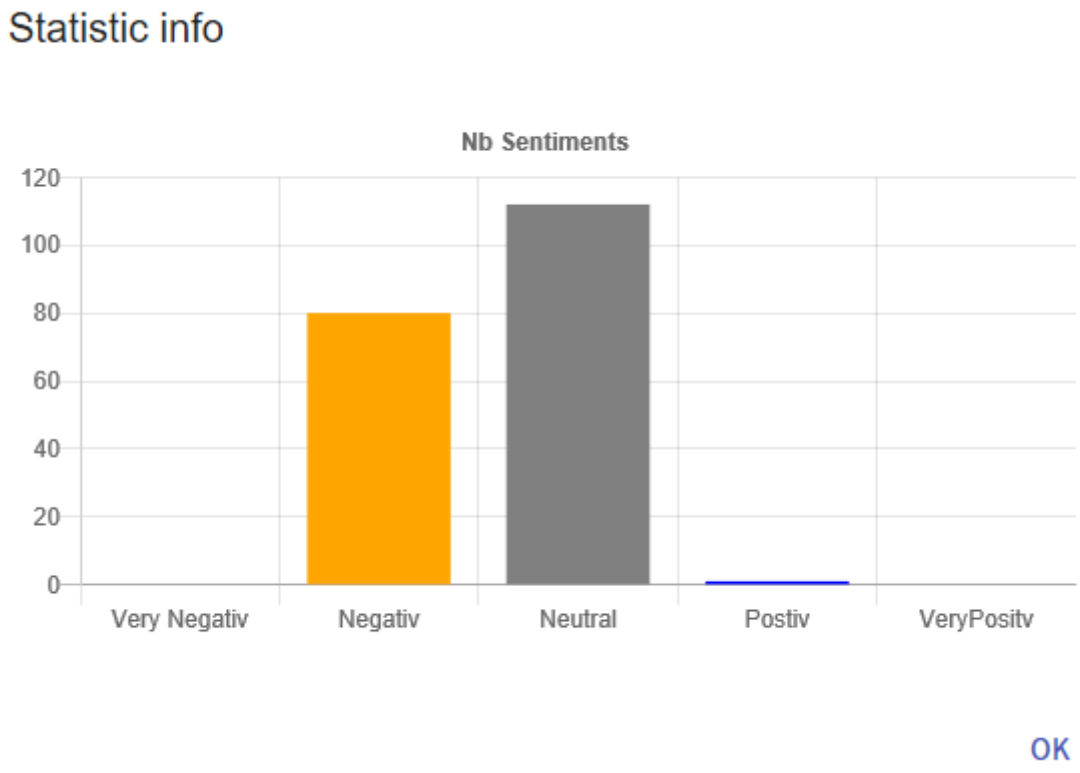
Implémentations détaillées

4.1 Détails sur le client

Pour ce qui est du client, il s'agit d'une application web en React relativement la simple. Nous avons des postes contenant le résultat de notre analyse que l'utilisateur peut consulter comme dans l'exemple ci-dessous.



La seule particularité notable est que le serveur va lui fournir une valeur représentant un sentiment pour chaque Tweet analysé et celui-ci va calculer la moyenne et l'afficher dans un graphe à barre comme on peut le voir ci-dessous.



4.2 Détails sur le serveur

La logique de notre projet se déroule du côté du serveur. En passant outre les modules dont le seul intérêt est la récupération des données, il nous reste les contrôleurs et les services qui ont demandé la majeur partie du travaille sur le serveur.

```
1 GET     /user/login                          controllers.UserController.login(email:String, password:String)
2 GET     /user/register                       controllers.UserController.register(email:String, password:String)
3 GET     /tweet                               controllers.TwitterAnalysisController.analyze(screenName:String, mode:Long)
4 GET     /tweets                              controllers.TwitterAnalysisController.getTweets()
5 GET     /tweetResponses                      controllers.TwitterAnalysisController.getTweetResponses(id:Long)
6 GET     /message                             controllers.AsyncController.message
7 GET     /assets/$file<.+>                    controllers.Assets.versioned(path:String = "/public", file:Asset)
8 GET     /tweet/delete                        controllers.TwitterAnalysisController.deleteTweet(id:Long)
```

4.2.1 Les contrôleurs

Les contrôleurs permettent de définir les actions que le serveur doit entreprendre lorsque une requêtes HTTP lui est transmise. Ici, nous avons séparé la gestions des utilisateurs de notre application, car seul les utilisateurs authentifié auprès du serveur doivent être capable de l'utiliser, et des Tweets.

Utilisateurs

Du côté des utilisateurs, nous avons mis en place deux routes utilisant le protocole GET.

register Passant par l'endpoint `/user/register`, l'utilisateurs peut se créer un compte et en échange il obtient un token JWT. À partir de ce moment, l'utilisateurs est capable d'analyser des Tweets et retrouver d'ancien résultat.

login Passant par l'endpoint `/user/login`, l'utilisateurs peut se connecter à son compte et en échange il obtient un token JWT. À partir de ce moment, l'utilisateurs est capable d'analyser des Tweets et retrouver d'ancien résultat.

Tweets

Du côté des Tweets, nous avons bien plus de routes à disposition des utilisateurs.

Tweet Passant par l'endpoint `/tweet`, cet endpoint va lancer l'analyse d'un Tweet choisi en passant par le services correspondant que nous verrons plus tard dans le rapport. Seul les utilisateurs authentifié ont accès à cette partie.

Tweets Passant par l'endpoint `/tweets`, cet endpoint permet de lister la totalité des tweets qu'un utilisateurs a, auparavant, analyser. Ceci lui permet ensuite de demander à revoir leurs résultats.

Tweet responses Passant par l'endpoint `/tweetResponses`, cet endpoint permets de lister la totalité des réponses à un Tweet qui a été analysé. Ceci peux, par exemple, permettre aux utilisateurs de voir ce qui s'est passé lorsque l'un de leurs Tweets obtient une très mauvaise évaluation ou au contraire un très bon retour.

Tweet delete Passant par l'endpoint `/tweet/delete`, comme son nom l'indique, il permet de supprimer un Tweet dont l'analyse n'est plus nécessaire.

4.2.2 Les services

Dans cette partie, nous verrons comment sont traité les requêtes faites auprès du serveur. Nous avons deux services qui valent la peine d'être expliqué, le premier qui va calculer le ressentiment génèral d'un Tweet et le second qui va faire le lien entre le premier et le contrôleurs.

SentimentAnalyzer En cherchant une manière de résoudre le problème de récupération des sentiments dans un texte, un problème relativement complexe même pour un linguiste expérimenté, nous avons fini par trouvé un bon nombre d'outils offert par Stanford CoreNLP. Stanford CoreNLP mets a disposition des outils permettant de traiter le language humain, à partir de simple mots, de discours, de la structure d'une phrase, et de l'ensemble syntaxique utilisé, cet outil permet d'extraire un sentiments, des citations et bien plus encore. Nous avons bien évidemment utilisé leur extracteur de sentiments dans le cadre de notre projet. Lire les sentiments humain à partir de 280 caractères n'est pas une chose aisée, il faut plutôt partir de l'idée que l'algorithme va donner une indications relativement exacte du vrai sentiments contenu dans un message.

TwitterClientService Ce services va permettre de mettre en relation le message contenu dans un Tweet et les sentiments général qu'ils véhiculent. En règle génèral, lorsque l'utilisateurs demande d'analyser un compte Twitter, ce que nous faisons actuellement est de récupérer le dernier Tweet qui a été publié, les réponses à celui-ci et l'on utilise l'analyseur de sentiments de Stanford CoreNLP afin d'en ressortir la manière dont il a été perçu et les réaction qu'il a déclenché. Parmi les fonction intéressante, nous avons **getTweets** qui permet de récupérer les Tweets paginé depuis l'API de Twitter. Plus concrètement, cette fonction récupère les 100 premières réponses, puis en utilisant la valeur **next** obtiendra les 100 suivant et ainsi de suite jusqu'à que soit le compteur tombe à 0, ou qu'il y ai plus de Tweets à récolter. Le tout utilisant une fonction récursive-terminale qui comme nous l'avons vu en cours sont les meilleures et les plus optimisée lorsqu'il s'agit de traiter un grand nombre de données. Finalement, nous avons mis en place une promesse qui va nous permettre d'attendre l'obtention des résultats avant de passer à la suite.

```
def getTweets (n : Int, twitterAccountName: String):List[JsObject]= {

  val prom = Promise[List[JsObject]]()

  def loop(cpt:Int,
           request:String,
           acc:List[JsObject]):Future[List[JsObject]]={

    cpt match {

      case 0 => prom.success(acc).future
      case _ => {

        val requestSearchTweets: WSRequest = ws.url(
          "https://api.twitter.com/1.1/search/tweets.json"
          +request+"&tweet_mode=extended")

        val complexRequestSearchTweets: WSRequest =
          requestSearchTweets.addHttpHeaders("Authorization" -> headers )
            .withRequestTimeout(10000.millis)

        complexRequestSearchTweets.get().map( res => {

          val tweets: String = res.body
          val searchReponse: JsObject = Json.parse(tweets).as[JsObject]
          val tweetsUser: JsObject = searchReponse("search_metadata").as[JsObject]
          var nextResults: String = ""

          if ((tweetsUser \ "next_results").isDefined) {
            nextResults = tweetsUser("next_results").as[JsString].value
          }

          //No more next, need to stop here
          if (nextResults=="") {
            return loop(0, "", acc)
          } else {
            return loop(cpt - 1, nextResults, searchReponse :: acc)
          }
        })
      }
    }
  }

  val request : String = "?q="+twitterAccountName
                              +"&count=100&result_type=recent&tweet_mode=extended"
  loop(n,request,Nil)
  Await.result(prom.future, Duration.Inf)
```

```
}
```

Chapitre 5

Difficultés rencontrées

Dans cette parties, nous allons lister les quelques problème que nous avons rencontré lors de la réalisation de ce projet.

L'API de Twitter Toutes API qui existe possède des limitations, que ce soit en nombre de requêtes, en débit, un contrôle de saisie faible ou inexistant, des délais d'attente bien trop lent, et bien d'autres encore. L'API de Twitter ne fait pas exception à la règle, entre les limitations imposée et sa documentation éparse, nous avons bien failli changer l'objectif du projet.

Programmation concurrente Nous ne pensions pas que la concurrence nous poserait autant de soucis lorsque l'on a commencé le projet. Nous avons fini par résoudre une grande majorité d'entre eux mais il et possible que certains soient resté ou que nous ne les avons pas découvert lors de nos tests.

Le temps Malgré l'intérêt porté au projet, une journée ne dure que 24 heures. Les fins de semestres sont souvent chargée voire même très chargée. Parfois, il a été difficile de trouver le temps de travailler sur le projet. Sans compter le fait qu'il arrive parfois que l'on se retrouve bloqué par des bugs pendant plusieurs heures ce qui nous a fait perdre un temps précieux.

Le stress, la pression, et l'envie de bien faire Notre côté perfectionniste est un avantage mais aussi un défaut. Afin de créer une application dont on peut être fier, nous nous sommes mis une pression supplémentaire qui aurait pu être évitée. Mais il est toujours plus plaisant de rendre un beau projet qui fonctionne comme on l'avait souhaité.

Chapitre 6

Problèmes connu dans l'application

Notre projet est maintenant terminé et nous avons réspécté nos espérance. Toutefois, il reste quelque problème pour lesquels on aurait voulu avoir plus de temps afin de les corriger.

L'API de Twitter Comme nous l'avons déjà dit auparavant, nous avons eu beaucoup de soucis avec celle-ci. Il faut savoir que pour utiliser l'API de Twitter il faut posséder un compte développeur afin de générer des tokens pour ses applications. Dans notre cas, nous avons pu profiter des accès d'un compte développer étudiant gratuit, cependant ceci n'est pas suffisant si l'on souhaite passer le projet en phase de production. En effet, les limitations sont telle que l'application serait complètement inutilisable.

Garantie des résultats Actuellement, il nous est impossible de garantir de pouvoir lister toutes les réponses d'un tweets. Ceci est lié aux limitations que l'on subit par l'API de Twitter. Nous avons essayé de faire en sorte de minimiser ce problème autant que possible.

L'intégrité des données Pour une raison que nous n'avons pas encore réussi à trouver, nous nous sommes rendu compte qu'il était possible, et ce relativement facilement, de supprimer une analyse de Tweet depuis postman par exemple n'ayant pas le bon token ou, pire encore, sans avoir besoin d'utiliser un token d'authentification. Il semblerait qu'une condition est ignorée ou pas prise en compte, ceci peut s'expliquer par une mauvaise gestion de la concurrence, une étape du processus de vérification qui a été oubliée ou mal exécutée, si l'on avait plus de temps pour le projet, ce problème serait le premier que l'on aurait essayé de résoudre.

Chapitre 7

Possibles améliorations

La fin d'un projet ne signifie pas que l'application est terminée, si l'on veut que celle-ci vive il faut toujours l'améliorer. En ce qui concerne la nôtre, les améliorations suivantes devraient être envisagée.

L'API de Twitter Dans un premier temps, prendre un compte Entreprise qui permet une plus grande liberté pour les développeurs, lorsque ceci sera fait, il est bien probable qu'il faille également changer la récupérations des Tweets.

L'Optimisations Il est toujours possibles d'optimiser un projet d'une manière ou d'une autre et le notre n'y fait pas exception, si l'on applique les précédent points et tout particulièrement le premier, à savoir un changement d'API, nous aurons un volume de données bien plus important, chaque seconde compte surtout sur le web ou une dizaine de seconde ressemble à une éternité. Ajouter Apache Spark, un framework open source de calcul distribué,au projet serait déjà un bon point dans ce sens.

Est-ce que l'on viole la vie privée des employés ? Lors de ce projet nous nous sommes surtout posé la question de savoir si l'on pouvait faire certaine chose avec l'API de Twitter au lieu de nous demander si l'on y avait le droit. Nous sommes parti du principe que Twitter ainsi que les Tweets qui sont dessus font partie du domaine publique. Mais est-ce réellement le cas ?

L'authentification Comme nous avons pu le voir auparavant, notre système d'authentification n'est pas encore au point. L'idéal serait de le réparer ou même de le remplacer. En regardant les recommandation d'OWASP, on voit un framework qui se dégage du lot, **lift**. Bien évidemment, çela signifie qu'il faudrait abandonner Scala-Play et donc il faut bien mesurer les pour et les contres d'un tel changement et mesurer le temps nécessaire pour le faire.

Security Frameworks

The following Scala frameworks contain modules that help developers implement secure features such as Authentenciation, Authorization, CRSF or SQLInjection

Framework	Authentication	Authorization	CSRF	XSS	SQLInjection
Play	✓	✓	-	-	-
Deadbolt 2		✓	-	-	-
Play-pac4j	✓	-	-	-	-
Scala-oauth2-provider	✓	-	-	-	-
SecureSocial	✓	-	-	-	-
Silhouette - Play Framework Library	✓	-	-	-	-
Lift	✓	✓	✓	✓	✓
Akka (Akka-http)	✓	✓	-	-	-
Spray	✓	✓	-	-	-

Chapitre 8

conclusion

8.1 Projet

Ce projet, l'un des derniers de notre vie de bachelier fût très intéressant. Il nous a non seulement permis de mettre en pratique tout ce que l'on a appris lors du cours à option de Scala,mais également de l'expérience acquise lors des nombreux projets réalisé à l'HEIG-VD.

La possibilité de choisir un projet qui nous plaît a également été une motivation supplémentaire pour notre groupe.

Une gestion rigoureuse du projet sous git ainsi que la liberté de choisir la partie qui nous intéresse et que l'on maîtrise au mieux à permis à chaque membre du groupe de pleinement s'épanouir dans la tâche qui lui a été confiée.

8.2 Groupe

Chaque membre était pleinement impliqué dans ce projet. Nous avons su faire preuve d'une réelle volonté, d'un intérêt certain, et d'une remarquable résistance au stress lors de cette période trouble qu'est la fin du semestre.

La régularité et la justesse a été la clé dans le succès de ce travail. La communication entre les différents membres du groupe, les décision prise par chacun d'entre nous et le bon sens a permis de résoudre de nombreux problèmes dans les plus bréf délai ainsi que le succès de notre projet.

8.3 Avis des membres

Dejvid Muaremi Ce projet fut très intéressant à réaliser. J ?ai pu découvrir certaines technologies pour lesquelles je n'aurai pas forcément investi le temps nécessaire à leur compréhension par moi-même et j'ai pu mettre en pratique tout ce que j ?ai appris jusqu ?à maintenant. Il arrivait parfois que je prenne du retard sur mes tâches, mais, au final, j ?ai pu le rattraper.

Mentor Reka

Xavier Vaz Afonso