Documentation technique COSO

Sommaire :

Membres de l'équipe :

Mélanie BERARD
Claire DE MENDITTE
Kasra MANSOURI
Nicolas PARENT
Romain TCHAKAMIAN

1. Description du projet

a. Problème adressé

A l'aube de chaque élection, nombreux sont les sondages d'opinion et de prédiction annonçant très précisément la victoire de tel ou tel candidat. Pour autant, ces prédictions s'avèrent de plus en plus éloignées de la réalité ces dernières années, jusqu'à annoncer grand gagnant un candidat qui s'avère perdant par la suite, comme ce que l'on a pu voir lors des élections américaines 2016 voyant s'affronter Hillary Clinton et Donald Trump. Face à la difficulté des médias et instituts à sonder la réalité des élections, certains modèles informatiques sont apparus, et paraissent présenter des résultats plus justes. Ce modèle notamment avait prédit la victoire de Donald Trump face à Hillary Clinton : http://www.france24.com/fr/20161107-etats-unis-professeur-trompe-jamais-donne-donald-trump-gagnant-allan-lichtman. Par ailleurs, l'issue finale d'une élection se jouant de plus en plus dans la dernière semaine de campagne, voire les derniers jours, les données issues de l'analyse des volumes de recherche sur internet à propos des candidats les quelques jours précédant l'élection semblent se rapprocher des scores finaux.

Dans ce contexte, le projet Coso a pour objectif la donnée d'un score prédictif pour un candidat pour une élection parmi les élections présidentielles françaises (un ou deux tours de primaires et deux tours d'élections générales). Le calcul de ce score prédictif se base sur des données de volume de recherches récupérées sur Google Trends, sur des données de volume de tweets et d'opinions sur Twitter Trends, ainsi que sur les scores réels d'élections passées.

L'ensemble de ces élections depuis 2007, ainsi que les candidats associés, sont conservés en base de donnée. Ainsi, les utilisateurs disposent de deux fonctionnalités :

- la consultation du score réel et du score prédit par le modèle d'un candidat sur une élection passée;
- la consultation du score prédit par le modèle pour un candidat sur une élection imminente.

b. Profil des utilisateurs

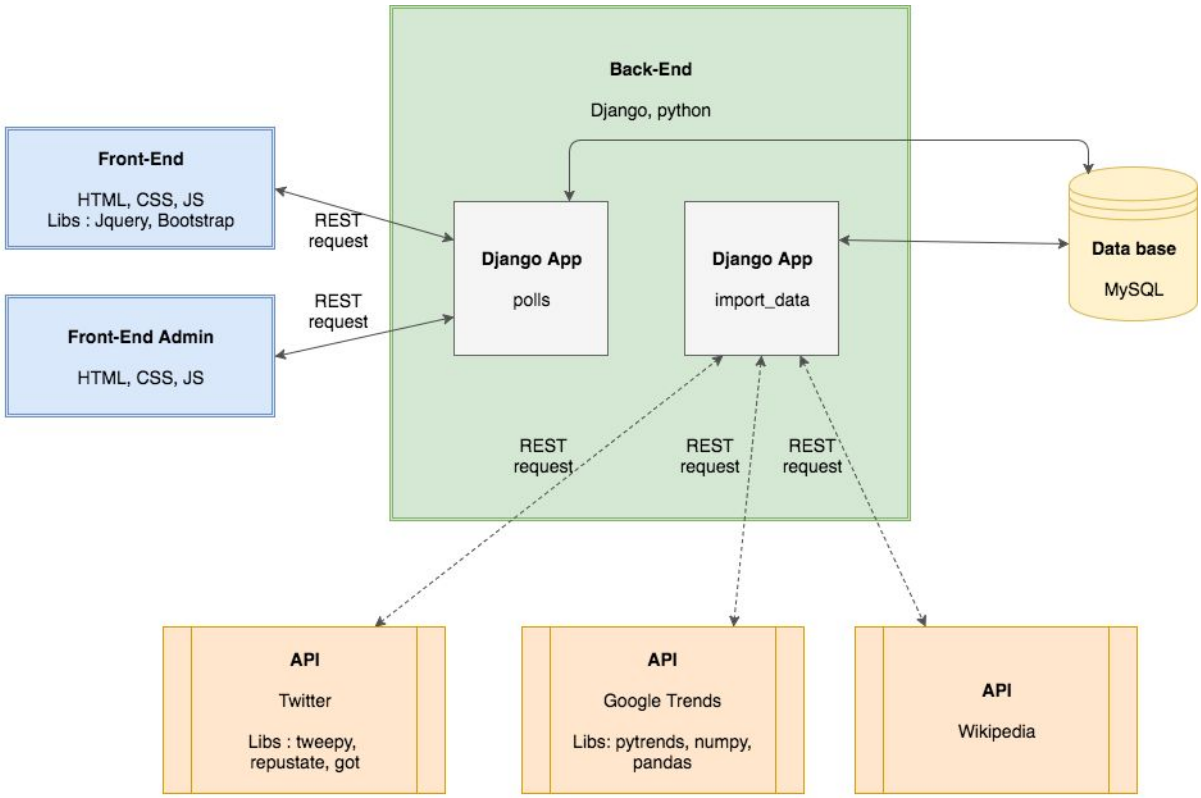
Plusieurs profils d'utilisateurs sont envisageables pour le projet Coso :

- les particuliers curieux de connaître un score prédictif pour une élection à venir;
- les médias et instituts pour appuyer ou mettre en relief des prédictions basées sur des modèles plus classiques;
- les candidats et leur staff de campagne pour connaître à la fois une estimation du volume de recherche qui leur est associé et de l'opinion qui leur est portée.

2. Vue d'ensemble de l'architecture

a. Schéma global (API, BDD, serveurs) (avec texte pour justifier les choix)

Schéma d'architecture



Notre application s'organise autour de quatre éléments : le front-end, le back-end, la base de données, et la connection entre le back-end et les API de Twitter, Google Trends et Wikipedia.

Le front-end est constitué d'un site web utilisé par nos utilisateurs et d'un site web admin permettant aux administrateurs du site, à savoir nous, de rentrer de nouvelles données dans la base de données, ou d'en modifier certaines.
Sur le site web, les utilisateurs ont la possibilité de voir quels sont les candidats et les élections disponibles sur coso, et d'exécuter une recherche sur le module statistique. Cette recherche se fait en choisissant une élection, un candidat, une date de début et une date de fin. Le résultat obtenu est le suivant :

- un détail des scores Twitter (moyenne, écart-type, valeur minimale et maximale sur l'analyse de sentiment)
- un détail des scores Google Trends (moyenne, écart-type, valeur minimale et maximale du volume observé en comparant notre candidat avec les autres candidats à l'élection)
- un score final fonction des scores Google Trends et Twitter

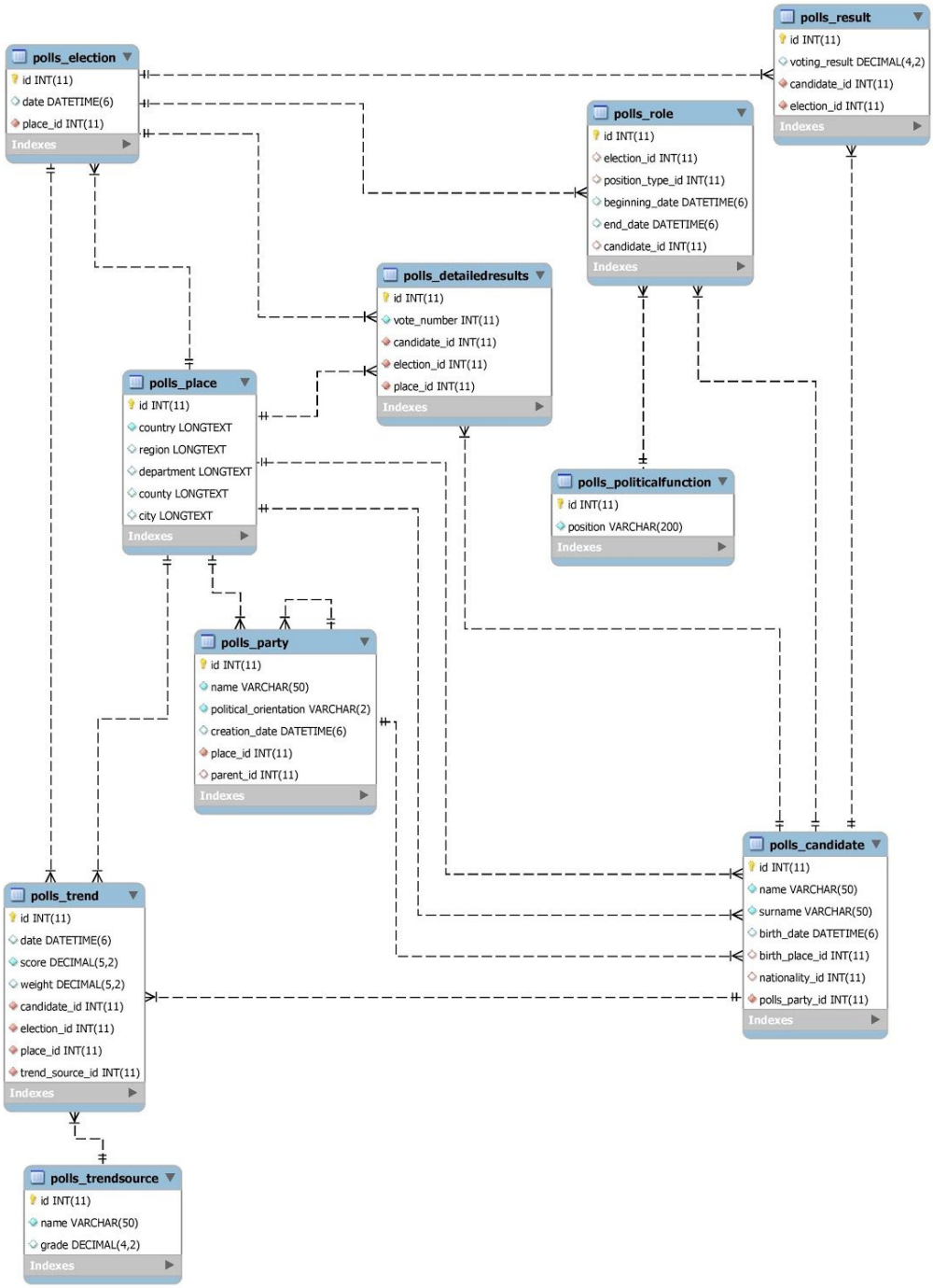
La seconde partie du front-end, la partie admin du site web, permet aux administrateurs d'ajouter, supprimer ou modifier la liste de candidat, d'élection, de score, etc.

Le back-end est organisé autours de deux applications principales : l'application polls et l'application import_data.

L'application polls, permet de modéliser comment sont enregistrés les données en base et comment celles-ci sont utilisées pour générer notre score final. Toutes les classes utilisées pour enregistrer des données dans la BDD proviennent de l'application polls.

L'application import_data sert à importer des données depuis internet. En effet, nous récupérons des données de Wikipedia, Twitter et Google Trends. Afin d'enrichir notre base de données en candidat, nous avons un module qui permet de se connecter à l'API de wikipedia pour récupérer pour chaque candidat quelques informations de base (date de naissance, lieu de naissance, postes, …). Ensuite, nous nous connectons à l'API de twitter pour récupérer des tweets correspondant à un certain tag, comme "PrimaireGauche". Enfin, nous récupérons aussi des données de Google Trends.

En ce qui concerne le modèle de la base de données, celui-ci est fortement lié aux classes de l'application polls. Voici le schéma de classe de la base de données :



Notre Backend est centré autour de 4 types de données : *Election*, *Candidate*, *Result* et *Trend*.

Les *Elections* et *Candidates* sont indépendamment définis, et les *Results* permettent de faire le lien entre les deux lorsque nécessaire, c'est-à-dire lorsqu'un *Candidate* fait bien parti des candidats à une élection. Les *Results* peuvent être créés sans score : le score que le candidat a réellement réalisé sera ajouté une fois l'élection finie.

Enfin, les *Trends* représentent notre donnée utile: il s'agit des données récupérées sur Twitter et Google Trends, et qui font office de sondage d'opinion dans notre projet.

Chaque *Trend* est lié à une unique *Election* et un unique *Candidate*, ainsi qu'à une unique date. De plus, chacun possède sa propre donnée géographique, afin de permettre une granularité plus fine sur nos données.
Le reste de nos données soutient ces 4 données centrales:

- *Place* indique un lieu, et peut être aussi bien à l'échelle d'un pays que d'une ville.
- *Role* et *PoliticalFunction* indiquent les fonctions précédemment tenues par un candidat. Dans un contexte politique où les sortants partent et les nouveaux venus revendiquent leur inexpérience de la politique, ce type de donnée peut être précieux pour l'analyse.
- *DetailedResults* permet d'avoir une granularité plus fine géographiquement sur les résultats finaux.
- *Party* indique le parti auquel appartient un *Candidate*. On peut de plus le lier à une orientation politique (Extrême Gauche, Gauche, Centre, Droite, Extrême Droite)
- Enfin *TrendSource* indique la source d'un *Trend*. Pour l'instant il n'y a que Twitter et Google

b. Limites et hypothèses de fonctionnement

Limitation liées aux données dans la BDD

Afin de faire fonctionner notre application, il faut qu'un minimum de données soient enregistrées dans la base de données. Il faut, par exemple, avoir au moins une élection et des candidats qui s'y rattachent.

Limite de fonctionnement Google Trends :

L'API Google Trends ne permet pas de récupérer la donnée "brute" qui pourrait nous intéresser. En effet, on obtient qu'une comparaison au pic de requêtes pour un sujet et une date donnée, au lieu d'un nombre de requêtes. On peut donc retravailler les données pour obtenir des pourcentages, mais aucune donnée absolue n'est disponible.
Cependant, au vu de l'évolution rapide de l'accès internet, ainsi que les stratégies numériques des candidats qui se développent, on peut supposer que le même nombre de requêtes concernant un candidat en 2007 et 2017 auraient un sens très différent. Il ne s'agit donc pas d'une limitation forte.

Pour ce qui est de la limite de requêtes, nous n'avons pas eu de problèmes venant de Google à ce sujet, la limite étant très haute (50 000 requêtes/jour). Nous avons cependant eu un problème à ce niveau avec Pytrends: automatiser plusieurs requêtes cause une erreur, que nous n'avons pas pu résoudre, le message d'erreur ne donnant aucun détail. Il ne s'agit sans doute pas de Google directement, car il y a un message d'erreur spécifique dans le cas où c'est l'API qui limite. Le problème semble être lié à une date mise par défaut à 0 (donc 1970) par Pytrends lorsqu'on formate un second appel à l'API Google Trends.

Enfin, nous avons eu un problème directement lié à l'API Google Trends: on ne peut comparer que 5 sujets à la fois, ce qui combiné à l'impossibilité d'effectuer plusieurs appels à l'API au sein d'une même requête nous oblige à restreindre nos candidats pour les élections.

Les limitations liées à l'utilisation de l'API Twitter

Pour récupérer les tweets par hashtag (#PrimaireGauche dans notre cas), nous avons d'abord utilisé tweepy. Tweepy prend en compte les limitations de twitter en terme d'appels via un paramètre "wait_on_rate_limit=True" qui gère automatiquement la latence de 15 min en cas d'erreur "API rate limit".

La méthode tweepy.Cursor(api.search, q=tag, since=day, until=next_day).items() permet de récupérer une liste sous forme d'iterator avec en input le hashtag recherché et l'étendue temporelle souhaitée. Dans notre utilisation, nous cherchons à créer des statistiques par jour, donc une étendue day à next_day permet de récupérer les tweets du jour day.

La librairie tweepy est donc parfaitement adaptée à notre problème, à un détail près : l'API Twitter sur laquelle elle repose ne permet pas de récupérer des tweets vieux de plus de 10 jours. Cette contrainte n'en est pas une si on crée un script tous les jours qui récupère les tweets du jour pour une élection actuelle, mais est majeure lorsqu'on veut récupérer des tweets d' anciennes élections qu'on souhaite analyser.

Pour palier ce problème, nous nous sommes servis du travail de Jefferson Henrique nommé GetOldTweets-python (voir ici), dont l'approche se base sur du scrapping et non sur l'API de Twitter. En utilisant son travail du dossier "got", nous sommes en mesure de récupérer les tweets correspondant à un mot-clé (et non hashtag, mais les résultats sont très similaires car une quantité négligeable d'internautes ont utilisé "PrimaireGauche" sans le hashtag : pour un jour exemple, seulement 2 sur 413).

Sur une même date, les 2 librairies renvoient un nombre différent de tweets : en retirant les retweets, qui n'apportent pas d'informations, GetOldTweets renvoie environ 25 à 30% plus de tweets, et nous n'avons pas d'explications.

Cependant, un gros désavantage de GetOldTweets est qu'il semble renvoyer un nombre de tweets qui dépend de l'étendue temporelle pour une même journée : par exemple, en faisant une recherche sur le mois de Janvier, et une autre pour uniquement le 18 janvier, on trouvera des résultats très différents pour la date du 18 Janvier. Le résultat semble complètement incohérent sur un mois, par contre cohérent sur une recherche sur un jour, ce qui correspond à notre approche.

En conclusion, il vaut mieux être à jour sur les données de l'élection et utiliser Tweepy, qui a l'air plus cohérent, même s'il renvoie moins de tweets que GetOldTweets. Il faudrait consacrer du temps aux codes sources des deux librairies pour comprendre leurs avantages et leurs inconvénients.

Une remarque : pour tweepy, il faut une clé pour s'authentifier, et nous nous en sommes créés plusieurs pour changer de clé lorsqu'une latence est lancée sur le compte d'une de ces clés et ainsi minimiser la latence globale. En pratique, nous avions suffisamment de clés pour ne pas avoir de latence.

Une fois les tweets récupérés, nous voulons réaliser une analyse de sentiment. Pour cela, nous filtrons pour garder uniquement ceux qui mentionnent un candidat. En effet, nous attribuons un score à un candidat. Un tweet peut être un score pour plusieurs candidats s'il y a plusieurs mentions.

De nombreuses librairies Python existent pour l'analyse de sentiments, certaines sont payantes, d'autres comme celle que nous avons choisie, Repustate, propose une offre gratuite. Nous avons le droit à 1000 analyses textuelles, ce qui est peu et qui a donc conduit à la création de plusieurs comptes Repustate avec une fonction qui gère le remplacement des clés désuètes.

En terme de qualité, l'analyse de sentiment donne parfois des résultats surprenants, donnant notamment beaucoup de poids aux émoticônes. L'implémentation de l'analyse de sentiments étant hors du cadre de ce projet, nous avons gardé Repustate mais notre analyse bénéficierait des progrès dans ce domaine.

A la fin de l'exécution de ce module, nous chargeons en base les 0 à 7 trends correspondants aux scores et poids des 0 à 7 candidats sur le jour. Le score est la moyenne des scores de sentiments pour un candidat divisé par le poids qui correspond au nombre total de tweets filtrés pour ce même candidat.

3. Architecture détaillée de chaque entité

a. Import de données

Dans une première phase, nous avions pour objectif d'importer des données de sondage à partir de scrapping de différents sites internet. Nous avions notamment commencé à utiliser la bibliothèque BeautifulSoup. Néanmoins, nous nous sommes rapidement rendus compte que de nombreux sites s'avèraient difficiles à scrapper et présentaient des formats de données très différents. Nous nous sommes alors tournés vers les modules de Google trends et Twitter trends afin de recueillir des données.

Dans la partie importation de données, il était également important de remplir notre base de données avec les informations des candidats, des élections et les scores réels.
Dans un premier temps, nous avons donc importé une liste de candidats (sauvegardés dans un json), puis nous avons récupéré des données de l'API de Wikipedia pour obtenir plus d'information sur les candidats.

Dans un second temps, nous souhaitions avoir des informations sur les élections. Pour cela, nous avons créé le JSON french_elections. Ce JSON résume pour chaque élection le jour, le mois, l'année, les candidats, et le score de chaque candidat pour cette élection.
Grâce à la fonction french_elections de Views, nous avons ainsi rempli la base de données à partir de ce JSON.

b. Partie Twitter

*Voir la partie sur les limitations qui présente à la fois le module Twitter et notre cheminement en fonction des contraintes auxquelles nous avons dû nous adapter.

c. Partie Google Trends

Le module Google Trends se base sur l'utilisation de l'API non officielle pour Google Trends nommée Pytrends. Cette API permet d'automatiser le téléchargement automatique de rapports venant de Google Trends.
En particulier, il est possible de choisir un pays, une date de départ, une durée et un vecteur de mots clés afin de récupérer les trends correspondantes.
La fonction import_trends du module Views réalise cette récupération de données à partir d'une requête comprenant les éléments cités précédemment.
Enfin, la fonction analysis_from_google du module Views réalise l'automatisation des appels successifs à l'API pour une élection, l'ajustement de la métrique utilisée et la sauvegarde des résultats sur notre base de données.

Django permet d'intégrer facilement ce genre de module au sein d'un projet existant. Les dépendances telles que Pytrends et Panda sont ajoutées dans l'environnement, et l'on ajoute une URL dans un fichier dédié pour permettre un appel direct au module.

Tel qu'il est présent dans les répertoires dédiés à l'import de données, le module Google Trends se construit sur 2 fonctions : import_trends qui est une fonction intermédiaire, destinée à être appelée par analyse_from_google. Cette dernière fonction est appelée directement en utilisant une URL adaptée.

- import_trends :
    - Rôle : permet de formater l'appel à l'API Google et de récupérer les données
    - Input : les données spécifiques à un appel, telles que les candidats dont on évalue le score, l'intervalle de temps sur lequel on souhaite obtenir ces données et la limitation géographique
    - Output : Un dataframe *Panda*, contenant les données de Score brut, étalonnées en pourcentage du maximum de requêtes. Il est important de remarquer que l'on obtient non pas un nombre de connexion, mais un pourcentage par rapport au pic observé au sein de l'échantillon pour un candidat et une date
    - Utilité : Possède en mémoire l'URL de base de l'API Google Trends, la référence aux identifiants Google et les données de formatage telle que la langue, la durée de l'échantillonnage ou le pseudo du projet pour Google

- analysis_from_google :
    - Rôle : prépare les données spécifiques à notre appel, les transmet à la fonction import_trends puis récupère son résultat pour créer les Trends.
    - Input : Les données de la requête telle que définies par l'URL utilisée. (exemple : http://127.0.0.1:8000/import_data/google_analysis/1/ appelle cette fonction pour l'Election d'id 1.)
    - Output : Un message confirmant l'accomplissement de la tâche
    - Utilité : Au-delà de l'output direct, cette fonction prépare la donnée utile pour import_trends, et construit nos Trends partir des données récupérées.

Les Trends ainsi créés possèdent toutes les données nécessaires à leur identification (candidat, élection, date, lieu, source), mais aussi un score et un poids : le score est le pourcentage journalier du candidat par rapport à ses concurrents, le poids est la donnée brute de Google Trends. On a ainsi à la fois un « sondage d'intérêt » journalier, mais aussi un poids relatif de cet intérêt par rapport aux autres candidats mais aussi aux autres jours. C'est cette donnée qui sert de volume d'intérêt lorsque l'on consolide les résultats.

d. Utilisation des données

Pour obtenir un score consolidé au plus proche du score réel, il faut distinguer les scores intermédiaires liés au volume de recherches sur un candidat, et ceux liés à l'opinion portée sur un candidat. Par exemple, un candidat sur une élection peut représenter 70% des recherches, mais ces recherches peuvent être soit positives, soit négatives. Or, seules les recherches positives doivent être comptabilisées pour se rapprocher du score réel. Il faut donc pondérer le volume de recherches.
C'est pourquoi les statistiques liées à une élection sont séparées selon qu'elles viennent du module Google Trends ou du module Twitter Trends, puis consolidées par une méthode plus poussée qu'une simple moyenne arithmétique.

Le module Google Trends renvoie une seule donnée pour un candidat sur une élection qui correspond au volume de recherches en pourcentage sur ce candidat pour cette élection. C'est cette donnée qui est présentée dans la partie Google Trends du récapitulatif statistique.

Le module Twitter Trends renvoie deux données pour un candidat sur une élection : le volume de tweets en pourcentage liés à ce candidat pour cette élection ainsi qu'un indice d'opinion compris entre -1 et 1 (0 étant neutre, les opinions comprises entre -1 et 0 sont négatives et celles comprises entre 0 et 1 sont positives). Ces deux données sont présentées dans la partie Twitter Trends du récapitulatif statistique.

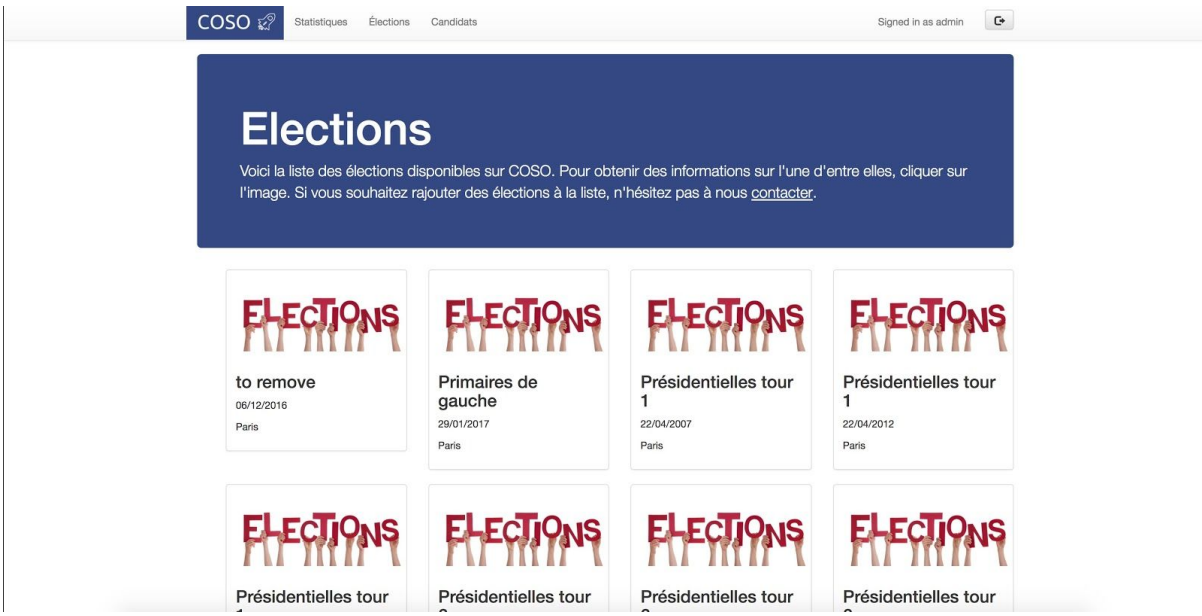
Le récapitulatif statistique présente également un score global consolidé. Le score consolidé représente la moyenne arithmétique des pourcentages de volume de recherches indiqué par Google Trends et de volume de tweets indiqué par Twitter Trends, pondérée par l'écart à la moyenne neutre de l'opinion (soit 1+K*opinion). Cette pondération permet donc d'ajuster le poids de recherche d'un candidat pour une élection à l'opinion générale qui lui est portée. Le facteur K est une constante permettant d'ajuster la pondération pour coller au plus près des scores réellement obtenus. Il est obtenu par expérimentations sur les élections déjà présentes en base de données.

e. Présentation des données sur le site web

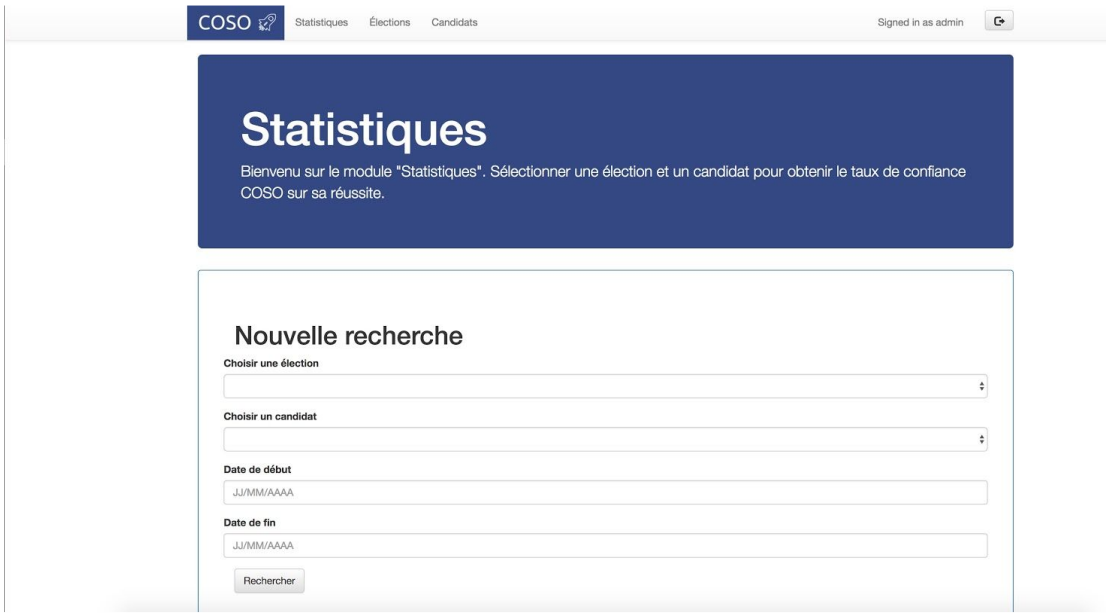
Sur le site web, nous présentons les éléments ci-dessous.

La liste des candidats inscrits :

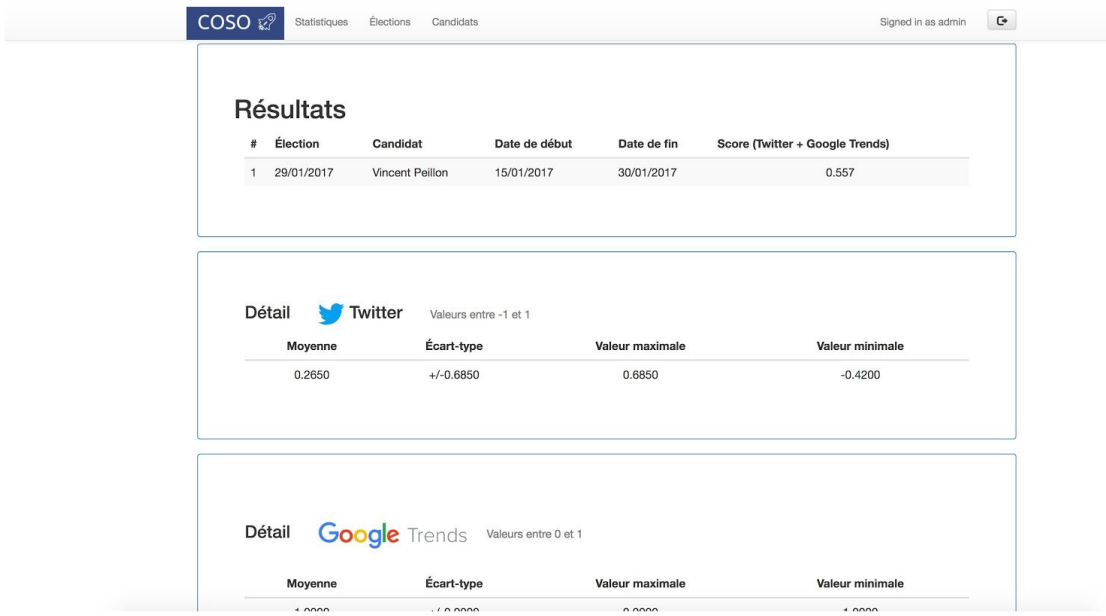
La liste des élections dont nous tenons compte:



Le formulaire de demande d’une nouvelle statistique

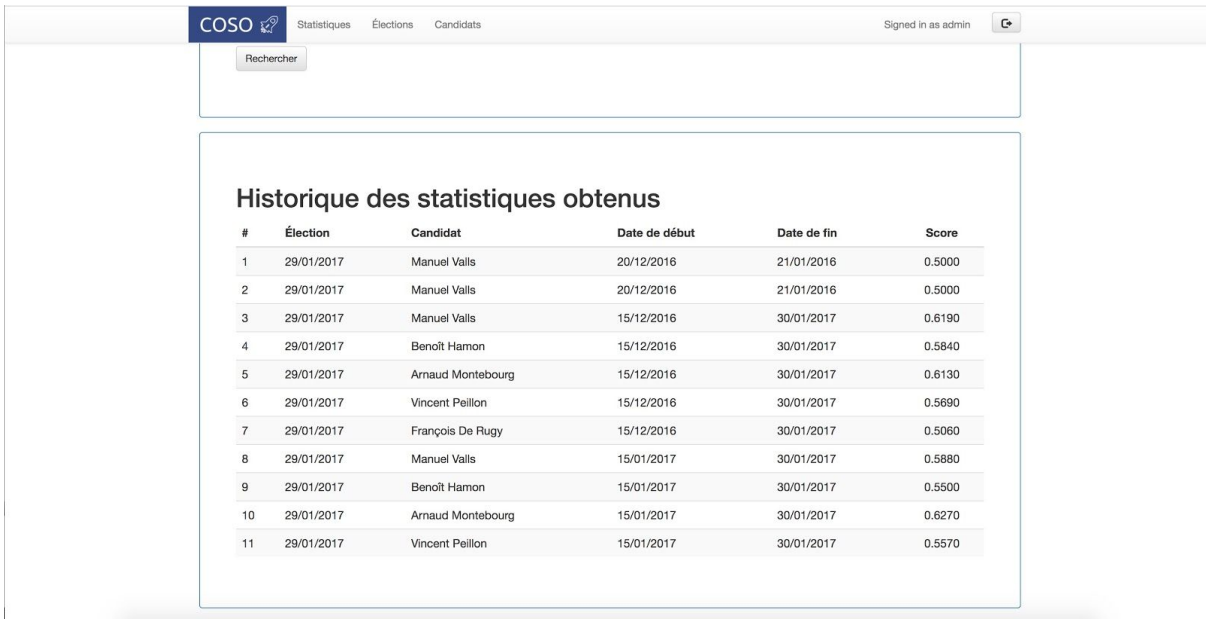


Les résultats de la recherche (résultats globaux, Twitter et GoogleTrend)



4. Résultats obtenus

Voici les résultats que nous obtenons actuellement sur les Primaires de Gauche de 2017 :



Historique des statistiques obtenus

#	Élection	Candidat	Date de début	Date de fin	Score
1	29/01/2017	Manuel Valls	20/12/2016	21/01/2016	0.5000
2	29/01/2017	Manuel Valls	20/12/2016	21/01/2016	0.5000
3	29/01/2017	Manuel Valls	15/12/2016	30/01/2017	0.6190
4	29/01/2017	Benoît Hamon	15/12/2016	30/01/2017	0.5840
5	29/01/2017	Arnaud Montebourg	15/12/2016	30/01/2017	0.6130
6	29/01/2017	Vincent Peillon	15/12/2016	30/01/2017	0.5690
7	29/01/2017	François De Rugy	15/12/2016	30/01/2017	0.5060
8	29/01/2017	Manuel Valls	15/01/2017	30/01/2017	0.5880
9	29/01/2017	Benoît Hamon	15/01/2017	30/01/2017	0.5500
10	29/01/2017	Arnaud Montebourg	15/01/2017	30/01/2017	0.6270
11	29/01/2017	Vincent Peillon	15/01/2017	30/01/2017	0.5570