**Rapport Projet Intégré**

Data Visualisation 1 :Affichage des données issues des réseaux d’informations DBLP

**Participants :**

**Scrum Master :** BEN HAJ Afaf

**Product Owner :** ELAMRAOUI Mohamed

**Equipe de développement :**

AMGHAR Ayoub, BARNAT Bochra,

BELGHIATIA Taha et MBOUP Libasse

Sommaire :

[INTRODUCTION 2](#_Toc56970730)

[I. GESTION DE PROJET 2](#_Toc56970731)

[i. L’équipe Scrum et les rôles associés 2](#_Toc56970732)

[ii. Les évènements 4](#_Toc56970733)

[iii. Les artéfacts 9](#_Toc56970734)

[II. PROJET INTEGRE 0](#_Toc56970735)

[i. Etude des données 0](#_Toc56970736)

[ii. Proposition de requetes et execution 3](#_Toc56970737)

[iii. Affichage des résultats sous forme graphe 5](#_Toc56970738)

[CONCLUSION 10](#_Toc56970739)

[ANNEXE 11](#_Toc56970740)

[BIBLIOGRAPHIE 12](#_Toc56970741)

# INTRODUCTION

Les services d’un réseau d’information sont rapidement devenus un puissant moyen de diffusion, de communication et d’échange par lesquels les utilisateurs partagent des informations en temps réel. Répondant à cette nouvelle tendance, il devient extrêmement important de visualiser cette quantité massive de données. Le champ d’application portera sur le réseau DBLP, référence en ligne en termes de bibliographies scientifiques ouvertes (https://dblp.uni-trier.de/).  
  
L’objectif de notre partie étant donc de prendre en entrée le réseau d’information cité précédemment et de présenter à l’utilisateur selon ses demandes ces informations sous forme de graphe. Afin de bien visualiser les résultats et d’avoir un affichage net des données. La structure de graphe s’avère être adaptée aux réseaux d’information.  
  
 Les données seront donc représentées sous forme d’un graphe d’information hétérogène pour une illustration plus exhaustive. Cela permettra d’analyser les données sous différents points de vue.  
  
Graphe : un graphe est un ensemble de sommets (ou points) et d'arcs ou d'arêtes liant certains couples de points. […] (Warusfel, Math. mod.,1969, p. 182, consulté sur https://www.cnrtl.fr/definition/graphe).  
  
**Notre partie abordera donc la formulation de requêtes et la visualisation dynamique des données sous forme de graphes. Dans un premier temps, nous présenterons le déroulement de la gestion de notre projet. Par la suite, notre application et nos résultats seront présentés.**

# GESTION DE PROJET

Pour la construction du projet, nous avons adopté une organisation projet agile, celle-ci nous a permis de poser un cadre méthodologique, que nous avons adaptée au contexte. La méthode SCRUM permet de répondre de manière productive au problème posé dans ce projet. Elle est constituée de plusieurs éléments que nous allons présenter dans cette partie.

## L’équipe Scrum et les rôles associés

Les rôles ont rapidement été définis après le premier CM de gestion de projet.

**Le Product Owner,** représentant du client est celui qui dans notre équipe définira les fonctionnalités du produit. Ici, on parlera plutôt d’utilisateur puisqu’à sa demande nous ajusterons les fonctionnalités afin d’afficher des résultats correspondant à ce qu’un utilisateur attendrait. Il porte aussi le rôle d’expert métier et prendra lors des réunions les décisions quant à l’avancée du projet.

Mohamed Elamraoui, Product Owner, mais aussi l’équipe de développement au complet a permis d’ajuster et d’alimenter au fur et à mesure le Product Backlog (liste des fonctionnalités du produit), notamment lors des passages au sprint suivant.

Pour visualiser le travail qui a été réalisé, nous présenterons avec détails le Product Backlog dans la partie artefacts (I. iii).

**Le Scrum Master**

Le rôle principal du Scrum Master est de s’assurer que l’équipe est pleinement opérationnelle et productive.  
Afaf Ben Haj, Scrum master du groupe a donc pris en charge ce rôle et a tenté tout au long du déroulement du projet de gérer le temps et de maintenir l’avancée du projet dans le bon sens. Le Scrum master a donc assuré le développement et la définition de la durée de chaque Sprint (cf. partie artefacts I. iii), ainsi que les réunions hebdomadaires de l’équipe.  
Dans notre gestion du projet, le Scrum Master s’ait également chargé du partage des tâches en accord avec l’équipe de développement. En effet, nous avons fait le choix de monter le Projet en formant deux équipes parallèles : une sous-équipe se chargeant de développer les requêtes et leurs exécutions, une autre se concentrant sur les librairies d’affichage pour la visualisation des graphes. Une manière de travailler qui nous a permis de ne pas perdre de temps notamment sur l’affichage des graphes où la recherche d’une bonne librairie fut fastidieuse. Pour que le projet soit tout de même homogène nous avons gardé une collaboration étroite grâce aux réunions.

**Équipe de développement**

L’équipe de développement est composée d’Ayoub Amghar, de Bochra Barnat, de Taha Belghiatia et de Libasse Mboup. Elle s’est auto-organisée dans la réalisation des fonctionnalités du produit.

Ils ont présenté les résultats de leur travail aux différentes réunions pour pouvoir en discuter avec l’équipe complète.

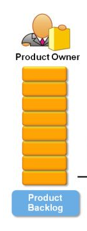
**Processus global**

**Processus de travail**

**Vision du produit,**

**Feuille de route,**

**Objectifs**



**Equipe fonction**

**Requête et exécution**

**Taha, Ayoub et Mohamed**



**Graphes**

**Bochra, Libasse et Afaf**

## 

Figure 1 – Processus de travail global

## Les évènements

**Le Sprint** est défini en amont, il représente une durée limitée pour réaliser une tâche. Il reste important de vérifier en cours de sprint que son développement répond bien à la demande du client. Les réunions permettront d’adapter la fonctionnalité aux besoins de l’utilisateur. Il permet l’optimisation du temps et permet de revoir tous les besoins du Product Backlog.  
  
Le Sprint est au cœur de la méthode SRUM puisque c’est autour de lui que les événements que nous allons présenter découlent.

À savoir que les événements suivants nous ont réellement aidé dans l’avancée et le déroulement du projet, les séances de TD chaque semaine également.

**Nous allons présenter un à un les événements de manière générale tels qu’ils ont été appréhendés par l’équipe à la fin de la partie, nous rentrerons plus dans le détail des Sprints.**

**La réunion de planification de Sprint**

La planification de Sprint est basée sur le Product Backlog, en effet le Product Owner et l’équipe de développement font une réunion afin d’échanger ensemble les objectifs discutés à partir du Product Backlog.

Nous avons défini la durée et les objectifs de chaque Sprint à la première séance de TD en sélectionnant les attendus du projet (éléments du Product Backlog). Il s’est avéré que plusieurs choses ont évolué par la suite.

Avant le sprint, nous avons défini les tâches de développement et les rôles de chacun.  
  
Nous avons procédé par logique chronologique des exigences de l’utilisateur : pour réaliser les requêtes, il fallait avant étudier les donner et pour réaliser l’affichage, il fallait en amont avoir travaillé sur le requêtage.

Les tâches de chaque Sprint ont donc été formulées en fonction des exigences prioritaire (cf. annexe 2 et 3). Le temps estimé à la réalisation de chaque Sprint et tâche à plus ou moins bien été respecté.

**La mêlée quotidienne**

Nous nous sommes réunis lors de la mêlée quotidienne afin de discuter du travail réalisé et faire ressortir les idées ou problèmes rencontrés par l’équipe, l’objectif étant de mesurer l’avancement du Sprint. Ce point dure environ 15 minutes. Dans notre cas ce point fut plus ou moins hebdomadaire puisque nous la faisions avant chaque séance de TD de projet intégré et de TD de gestion de projet.

En ce qui concerne le TD de projet intégré nous échangions sur les questions et les problèmes que nous avions rencontrés pour qu’une personne les recense et les présente s’il y en avait. Pour le TD de gestion de projet, nous faisions un point sur l’avancée du projet pour réfléchir à ce qui avait été fait et ce qui restait à produire.

Il faut savoir que pour avoir une bonne communication, nous avons créé un Discord dès le début du projet. C’est sur cette plateforme que nous échangions. Une application étant disponible sur le téléphone, nous étions sûres d’échanger un maximum, notamment pour partager des ressources pouvant être utiles aux membres du groupe. C’est par ce biais-là que nous réalisions les différentes réunions.

**La revue du Sprint**

À la fin d’un sprint, nous faisions une réunion, ces réunions ont eu lieu sur un salon Discord où le partage d’écran pour la présentation des nouvelles fonctionnalités est simple. Ces réunions ont pour la plupart duré longtemps puisque beaucoup de retours sont émis par les différentes sous-équipes.  
  
   
**La rétrospective du Sprint**

Après la revue du Sprint, un plan d’amélioration est mis en place. Nous sommes souvent passées par cette étape. Nous adaptons donc les fonctionnalités, les idées de chacun qui sont bonnes à prendre.

* **Déroulement des événements en détail**

Présentons les choses chronologiquement. A savoir que deux sprints ont été réalisés parallèlement par chaque sous équipe pendant 2 semaines.

Ce qui était fait en arrivant sur les séances de CM/TD suivantes :

**30/09 et 01/10 :** Présentation du projet et des enjeux de celui-ci (Méthode Agile/SCRUM).

**07/10 et 08/10 :** Observation des données transmis par Mme Bentayeb.

Séance gestion de projet : attribution des rôles. Comment gérer le temps en minute.

**14/10 et 15/10 :**

3 Sprints ont donc été défini à la première séance de TD :

1. Etude des données. (1 semaine)
2. Requête et exécution (Affichage simple sur des DataFrame). (3 semaines)
3. Affichage des résultats sous forme de graphe. (3 semaines)

Démarrage du Sprint 1 : Etude des données, comprendre les données que l’on a pour pouvoir faire des requêtes adaptées à l’utilisateur.

**21/10 et 22/10 :** Revue du Sprint 1 (étude des données). Qu’est-ce qu’il ressort des données et qui serait intéressant à présenter.

A partir de cette étape les tâches à réaliser ont été revues et précisées.

Le client veut (Product Backlog)

* Une interface permettant de générer des requêtes interactives, c’est dire que lui décidera du filtrage des données que lui souhaite afficher.
* Il souhaite que cet affichage soit sous forme de graphe.
* Et que ces graphes soient de préférence dynamiques.
* Il souhaite aussi que cette application soit accompagnée d’un rapport présentant les fonctionnalités.

Sprints :

1. Etude des données. (1 semaine)
   1. Lecture des fichiers sur un fichier texte (pour analyser comment elles sont séparées par exemple). (1 jour)
   2. Importer les données sur Python. (2 heures)
   3. Vérifier qu’elles soient bien nettoyées, qu’il n’y ait pas de choses incohérentes. (2 jours)
   4. Stocker les données dans des DataFrame. (1 jour)
   5. Rédaction de la partie étude des données dans le rapport. (4 heures)
2. Requête et exécution (Affichage simple sur des DataFrame). (3 semaines)
   1. Réflexion sur les requêtes intéressantes à afficher.
   2. Création de fonctions permettant de retourner des requêtes.
   3. Création d’une interface (fenêtre graphique) pour que l’utilisateur rentre les champs et affiche un sous DataFrame représentant la requête sortant de la fonction associée.
   4. Test de l’interface.
   5. Rédaction de la partie requête sur le rapport.
3. Affichage des résultats sous forme de graphe. (3 semaines)
   1. Recherche de librairies permettant l’affichage de graphe à partir d’un DataFrame.
   2. Codage requêtes en attendant les fonctions de requêtes du sous-groupe requête et exécution.
   3. Exécution des codes réalisé à partir des librairies trouvées pour l’affichage des graphes avec les données de DBLP.
   4. Rédaction de la partie graphe sur le rapport.
4. Intégration des fonctions de requête à l’affichage du graphe (1 jour)
5. Intégration des données du groupe interface. (2 jours)
6. Intégration des différentes interfaces. (2 jours)
7. Rédaction de la partie gestion de projet sur le rapport (3 jours)
8. Homogénéisation du rapport et relecture et préparation de la soutenance (2 jours)

Lors de ces séances, nous avons fait le choix qui par la suite s'est avéré le mauvais de proposer 5 requêtes chacun à l’écrit à partir de ce que l’on avait pu tirer de l’étude des données (soit 30 requêtes en tout : cf. Annexe 1).

* À la fin de cette semaine **le 24/10,** nous avons fait une réunion pour la planification du sprint 2.
* Dès le départ étant parti sur une organisation en sous-équipe, lors de la réunion, nous avons décidé que les personnes faisant partie de l’équipe requête devait proposer à la fin de la semaine de vacances les fonctionnalités attendues par l’utilisateur. Mohamed Elamraoui, Ayoub Amghar et Taha Belghiatia se sont donc chargé de coder les requêtes.

**Le 31/10** c’est l’heure de la revue du Sprint 2 : les fonctionnalités produites par l’équipe requête ne sont pas adaptées. Créer des requêtes à la volée ne permet pas à l’utilisateur/au client de faire des affichages interactifs. Tout est à refaire. Il faut permettre à l’utilisateur de d’entrer les informations qu’il veut afficher. Une fenêtre de filtre pour que l’utilisateur crée lui-même la requête. Nous faisons donc la rétrospective du Sprint et réfléchissons au plan d’amélioration.  
  
**Plan d’amélioration :** créer une « interface test » et des fonctions liées aux inputs et aux boutons permettant la création des requêtes. Pour gérer le temps, nous avons procédé de la manière suivante :

Nous poursuivons le Sprint 2 en créant les fonctions de requêtes.  
Nous savons que les résultats de ces fonctions seront stockés dans des DataFrame, nous décidons donc de lancer le Sprint 3 par l’équipe Graphes : les tâches du Sprint sont donc distribuées pour Bochra Barnat, Libasse Mboup et Afaf Ben Haj.

Sprint 2 + 3 = 6 semaines, mais deux semaines en même temps donc 4 semaines. Optimisation du temps.

**04/11 et 05/11 :** création de l’interface pour l’équipe requête, l’équipe graphe présente les libraires de graphe trouvées pour le moment seulement des graphes statiques  
  
**12/11 :** 2 fonctions de requête sur 5 sont réalisées pour l’équipe requête, en attendant les fonctions le groupe graphes teste ses librairies sur des DataFrame que nous avions du cours Data Mining dispensé par M. Rakotomalala.

**18/11 et 19/11 :** Le groupe graphes a trouvé une librairie déployant des graphes dynamiques. Test sur les DataFrame et présentation au groupe requête. Revue des Sprint 2 et 3 et planification du Sprint 4 : intégration de deux sous équipes.

**25/11 et 26/11 :** présentation du travail réalisée. Rétrospective des Sprint 4 : il faut travailler le visuel, en effet, l’interface doit être plus compréhensible et agréable pour l’utilisateur.  
  
**Le 27/11 :** une réunion a lieu avec les Scrum Master et PO du TD 1 (1 réunion comme celle-ci va être réalisé chaque semaine jusqu’à la fin du projet) pour discuter de l’intégration des données et des interfaces. Elle conforte l’idée qu’il faille changer notre interface, en effet premièrement pour une question d’esthétiques qui nous appartient, mais aussi pour l’intégration les autres groupes ayant travaillé sur une interface web pour une intégration simple et rapide nous avons fait le choix de changer même si le temps manquait plus ou moins. En effet, avant d’avoir les données du groupe nous avions 2 jours de battement où nous nous sommes mis sur la création d’une interface web avec Flask (interface web via Python.).

Entre temps, le Sprint 7 est démarré.

**Du 27/11 au 29/11** création d’une nouvelle interface et intégration des fonctions à celle-ci.  
  
 **30/11 et 01/12 :** Sprint 5 : l'intégration des données du groupe interface.

**02/12 et 03/12** : Planification du Sprint 6. Démarrage de l’intégration des interfaces avec les autres équipes du TD.

**05/12 et 06/12 :** Sprint 8, homogénéisation du rapport et relecture et préparation de la soutenance.  
  
**8/12 :** Présentation orale.

## Les artéfacts

**Product Backlog** (cf. figure 1 et annexe 2 et figure 2)

Le Product Backlog centralise la liste des exigences attendues (fonctionnalités, exigences non-fonctionnelles.

Le travail de gestion de projet s’est effectué sur GitLab, en accord avec tous les membre du groupe, il permet plusieurs fonctionnalités dont la possibilité de gérer le Product Backlog en créant une « GitLab issue lists » correspondant aux besoins de l’entreprise et des clients, ceux-ci sont classés par ordre de priorité pour faire correspondre à l’urgence et l’ordre de développement souhaité (cf. Annexe 1). Il a été constamment affiné, complété grâce aux questions soulevées et la communication au sein de l’équipe.



**Priorité**

Ajout d’un besoin

+

BESOIN #1

BESOIN #6

BESOIN #22

|

Retrait d’un besoin

BESOIN #31

Figure 2 – Product Backlog

Prenons un exemple concret qui a marqué l’avancée de la gestion du projet et reprend cette représentation du travail sur le Product Backlog :

Nous avons remis en question en se mettant face aux besoins du client sur la manière de faire les requêtes sur le Sprint requêtes et exécution que nous développerons par la suite. En effet, nous avons testé la demande de base qui était de proposer une trentaine de requêtes (cf. annexe 1) que l’équipe de développement pensaient puis codaient. Mais la conformité avec les demandes de l’utilisateur ne correspondait pas. Pour que l’utilisateur puisse afficher de manière dynamique ce qu’il voulait visualiser, nous avons créé les requêtes à l’aide de fonction se basant sur les paramètres.

**Sprint Backlog** (cf. annexe 3)

Le Sprint Backlog comporte la liste des tâches du Sprint se basant sur le Product Backlog. Dans notre cas ces données, on était rentré dans GitLab mais aussi sur papier pour être sûre et reporter correctement les dates.

**Graphique d’avancement de Sprint**

Tout au long du Sprint, n’importe quel acteur du projet peut consulter la progression de l’équipe de développement grâce au graphique d’avancement actualisé quotidiennement. Ce graphique indique l’évolution du Reste A Faire (généralement exprimé en heures) en fonction du temps.

Sur Gitlab, aucune fonctionnalité n’a été trouvée pour générer un graphique d’avancement. Il est pourtant très intéressant de visualiser l’avancement du projet. Nous avons donc cherché de modèle Excel de gestion de Projet Agile. Nous avons repris les éléments insérés sur GitLab dans le tableau Excel qui généré le graphique d’avancement. Il se base sur le tableau des tâches. Ce tableau comporte la liste des tâches du Sprint Backlog.  
  
Il nous a permis de voir, malheureusement à la fin comment avait avancé le projet en un coup d’œil, voilà ce que nous obtenons :

# PROJET INTEGRE

## Etude des données

#### En début de semestre nous avons eu des fichiers déjà traités en attendant que le groupe interface En début de semestre, nous avons eu des fichiers déjà traités en attendant que le groupe interface nous fournissent des données.

#### Description des fichiers reçus en début de semestre :

#### Les fichiers représentent les publications des années 2017, 2018 et 2019. Nous avons globalement 9 fichiers sous format csv dont 5 représentants les entités et 4 les relations entre les entités.

#### Le fichier author on y trouve le nom de l’auteur (name\_author) et le nombre de publications de l’auteur (nbr\_publication) de celui-ci notamment. Le fichier keyword qui reprennent des mots clé et le nombre de fois qu’ils ont été utilisés. Le fichier publication ici nous avons la date exacte de publication (jour/mois/année) allant du 07/12/2016 au 07/03/2020. Nous avons également le nombre d’auteur(e) de la publication, le titre de l’article et sa catégorie. Le fichier venue correspond aux conférences de manière générale. Le fichier year contient les années 1900 à 2000. Aucune donnée manquante n’a été trouvée sur ces fichiers.

#### Les fichiers représentants les liens entre les entités :

#### Le fichier publication\_author met en relation une publication et auteur par l’identifiant unique des entités. Le fichier publication\_venue met en relation une publication et sa conférence un par l’identifiant unique des entités. Le fichier publication\_keyword (id\_publication,keyword, nbr\_use\_keyword) met en relation une publication et les mots clé de celle-ci, ainsi que les nombres de fois où ce mot-clé est utilisé. Le fichier publication\_year (id\_publication,id\_year) met en relation une publication et l’année par l’identifiant unique des entités.

#### Chargement des données

#### Après ouverture de nos fichiers CSV avec un éditeur de texte, nous observons de quelle manière les données sont présentées, la séparation des données étant le caractère «, »

Une fois ouvert, nous repérons certaines informations :

* Les colonne(s) contenant des données
* Le format des données
* Le caractère par lequel les colonnes sont délimitées
* Les fichiers contiennent des en-têtes : la toute première ligne indique le nom des colonnes (on l'appelle en-tête ou header), et les suivantes correspondent donc aux données brutes.

Nous avons donc fait le choix d’utiliser le langage Python et rédiger un script sur Jupyter notebook pour avoir la visibilité sur chaque commande exécutée. Pour charger les fichiers nous utilisons la méthode read\_csv, pour cela nous avons besoin de la librairie Pandas. Enfin nous stockons ces fichiers dans un DataFrame.

*Import pandas as pd*

*Data = pd.read\_csv("fichier", …)*

*Print(data)*

C’est grâce à la librairie Pandas que l’utilisation de la fonction read\_csv, l’encodage des fichiers était le suivant ISO-8859-1.

#### Vérifier que les données sont nettoyées sur python pour avancer tranquillement sur les requêtes et les graphes :

Les données étaient en effet bien nettoyées la commande précédente n’a réalisé aucun changement sur les données. Malgré cela nous avons rencontré plusieurs problèmes avec le fichier venue que nous avons finalement résolue à l’aide de 5 paramètres :

1. **Engine** : c’est un moteur d'analyseur à utiliser. Le moteur C est plus rapide tandis que le moteur python est actuellement plus complet ce qui explique notre choix
2. **skip\_blank\_lines booléen**

Si True, ignorez les lignes vides plutôt que de les interpréter comme des valeurs NaN.

1. **error\_bad\_lines booléen**

Les lignes avec trop de champs (par exemple, une ligne csv avec trop de virgules) provoqueront par défaut le déclenchement d'une exception et aucun DataFrame ne sera retourné. Si False, alors ces « mauvaises lignes » seront supprimées du DataFrame qui est retourné.

1. **warn\_bad\_lines booléen**

Si error\_bad\_lines vaut False et que warn\_bad\_lines vaut True, un avertissement pour chaque «mauvaise ligne» sera affiché.

*Data = pd.read\_csv ("fichier",engine='python',skip\_blank\_lines=True*

*,error\_bad\_lines=False,warn\_bad\_lines=False)*

Nous avions également renommé dans le dataframe df\_publication\_venue de la colonne d\_venue en id\_venue pour plus de clarté dans les requêtes et pour pouvoir les utiliser dans le merge ( la liaison) entre les fichiers.

*df\_Publication\_venue.rename(columns={'d\_venue': 'id\_venue'}, inplace=True)*

**Passage sur les données du groupe interface :**

Lorsque nous sommes passé sur les fichiers du groupe interface nous n’avons pas rencontré les problèmes cités ci-dessus. Ils ont pu nous fournir des fichiers équivalents à ceux fournis par Mme Bentayeb. En revanche, nous avons rencontré quelques soucis lors de l’intégration des données, en effet les colonnes n’ayant pas exactement les mêmes headers, nous avons dû les recoder sur nos fonctions de requêtes. Sur ces fichiers, l’encodage des fichiers était UTF 16. Les données contenant beaucoup trop de données nous avons fait le choix de travailler sur un échantillon de ces données.  
  
Ci-dessous les nouveaux noms de colonnes qu’il a fallu mettre à jour dans nos fonctions :  
author 🡪 author , nbr\_publication  
keyword 🡪 keyword,nbr\_utilisation  
publication 🡪 id\_publication, date\_pub, article\_title, categorie, nbr\_authors  
publication\_author 🡪 author ,id\_publication  
publication\_keywords 🡪 id\_publication, keyword, nbr\_use\_keyword  
publication\_venue 🡪 id\_publication, id\_venue  
publication\_year 🡪 id\_publication, id\_year  
venue 🡪 name\_venue, id\_venue  
year 🡪 year,id\_year

## Proposition de requêtes et exécution

Requêtes proposées :

Nous avons démarré le requêtage en se questionnant sur les requêtes intéressantes à présenter une longue liste en a découlé (cf. Annexe 1).

Au départ, chacun a proposé 5 requêtes paramétrables auxquelles on devait répondre :

df\_key\_author = pd.merge(df\_Publication\_author, df\_Publication\_keywords, left\_on = 'id\_publication', right\_on = 'id\_publication', how = 'inner')

df\_key\_author.loc[df\_key\_author.id\_author=='id\_79/6633']['keyword'].drop\_duplicates()

Pour ce faire nous avons utilisé la fonction merge de la libraire pandas pour faire une liaison entre des fichiers csv.

Mais cette méthode n’était pas intéressante au vu de la limite de 30 requêtes que nous nous étions fixés.

Finalement en réfléchissant du point de vue du client nous nous sommes rendu compte qu’afficher les requêtes proposées était intéressant, mais qu’il serait d’autant plus intéressant de rendre celles-ci dynamique, interactive du point de vue du client.

En effet, cette remise en question a été possible grâce à la création d’une « Interface Test ». Celle-ci nous a permis de nous questionner sur l’exécution des requêtes. Malgré cela le travail effectué avant n’a pas totalement été perdu, mais il a fallu revoir les requêtes déjà codées.  
Pour cela, avant de coder, il a fallu modéliser ce que l’on souhaitait obtenir. L’organigramme suivant présente la procédure d’exécution des requêtes.

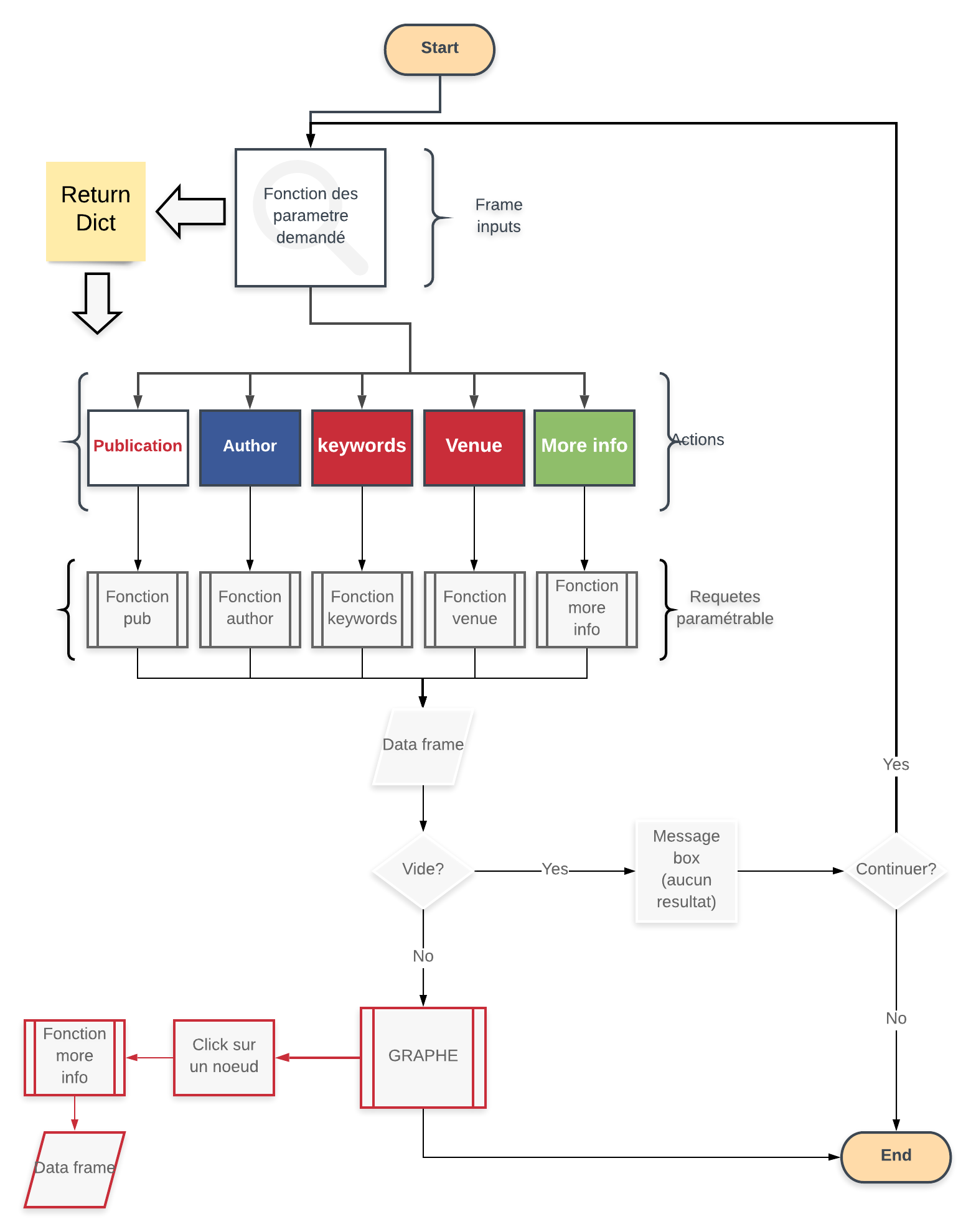


Figure 1 - Organigramme la procédure d’exécution des requêtes

Pour répondre à cette logique voici les différentes fonctions utiles aux requête :

**Fonction des paramètres demandés :**

Création d’une première fonction getinputs() :

Cette fonction vérifie s’il y a une entrée ajoutée à notre dictionnaire, puis le met à jour en le retournant à la fin.

Le nom des entrées vérifiées sont les noms des cases nommées dans notre code d’interface. Elle prend donc en entrée les données remplis dans le formulaire html

*def getinputs():*

*dict = {}*

*if request.form.get('tauthor')!="" :*

*dict.update({'author': request.form.get('tauthor')})*

*if request.form.get('tpublication')!="" :*

*dict.update({'publication': request.form.get('tpublication')})*

*if request.form.get('tvenue')!="" :*

*dict.update({'venue': request.form.get('tvenue')})*

*if request.form.get('tkeyword')!="" :*

*dict.update({'keyword': request.form.get('tkeyword')})*

*if request.form.get('tyear')!="" :*

*dict.update({'year': request.form.get('tyear')})*

*return dict*

**Fonction Get\_id :**

Nous avons créé des fonctions qui permettent de retourner l’id d’un attribut chercher selon son nom :

* Get\_id\_venue(names) : prend en parametre un ou plusieurs noms sépares avec une virgule () et split (« , ») retourne un ou plusieurs id
* Get\_id\_author(name)

En effet, il est toujours plus intéressant pour un utilisateur de chercher un nom plutôt qu’un identifiant.

**Les fonctions de recherche :**

Ces fonctions prennent en paramètre le dictionnaire remplie à l’aide des inputs :

* def Publication(dict) :
* def Keywords(dict) :
* def Author(dict) :
* def Venue(dict) :
* def More\_info(dict) :

Ces fonctions finalisent le fonctionnement du diagramme présenté ci-dessus en ce qui concerne les requêtes.





Pour exécuter une requête l’utilisateur n’a donc qu’a remplir les champs souhaités du formulaire. Et cliquer sur le bouton des informations qu’il souhaite observer.

Exemple : Afficher les publications d’une année données comportant le ou les mots-clés(s) (en utilisant la virgule) donné(s)

## Affichage des résultats sous forme de graphe

 Le graphe d'un réseau est la représentation graphique d'un réseau à éléments localisés (sommet), dans laquelle les branches (relation) sont figurées par des segments de ligne et les nœuds par des points. (Larousse)  
  
Principales caractéristiques de nos graphes : ils sont non orientés et hétérogènes. (réseaux bibliographiques auteurs / articles)  
  
Concernant les graphes, nous avons commencé à les réaliser étape par étape.  
  
Au début, nous avons voulu bien maîtriser l’affichage des graphe sur python donc nous avons décidé de se concentrer d'abord sur l'affichage des graphes statique à l’aide des bibliothèques Networkx , un package Python pour la création, la manipulation et l'étude de la structure, de la dynamique et des fonctions de réseaux complexes.  
  
Après avoir fouillé la documentation de Networkx et essayer différents tutoriels, nous avons procédé finalement comme suit :

*Import networkx as nx*

1) Importation de la librairie.

2) La construction du graphe est réalisée grâce à la fonction from\_pandas\_edgelist qui prend en entrée un DataFrame et deux attributs de ce dernier : la source et la destination. Nos fonctions de requêtes produisent des DataFrame, l’intégration des deux parties était donc simple.

*G=nx.from\_pandas\_edgelist(df\_”f”\_of\_”fichierdf”,source=”fichier”,target=”fichier”)*

3) La fonction type(G) prenant en paramètre G nous donne le type. nx.info(g) donne les informations sur le nombre de nœuds, d'arcs (ou arêtes) et le degré.

4) Les fonctions g.node() et g.edges() affichent respectivement à l'écran les nœuds et arcs du graphe .

5) L’affichage du graphe avec nx.draw. Cette fonction prend en paramètres le graphe construit dans notre exemple G, with\_labels = true permet d'avoir sur l'affichage la valeur de l'enregistrement du nœud concerné.

*nx.draw(G,with\_labels=true)*

Pour le côté esthétique, les paramètres node\_color , node\_size , edge\_size jouent sur la couleur, la taille des nœuds et des arcs a l'affichage.

*nx.draw(G,with\_labels= true,node\_color='r',node\_size = 80,edge\_size = 0.05)*

On a également une autre façon de pouvoir afficher avec Matplotlib sous forme d'image avec la fonction de Networkx suivante nx.draw\_networkx() . Cette image peut être enregistrée sur le disque avec matplotlib.save("nomdelimage.son\_extension") .

**Après avoir** réussi l’affichage des graphes statiques, nous avons commencé à chercher des librairies pour les graphes interactifs qui prendrait en entrée le graphe statique crée par NetworkX pour faciliter la tâche.

Au début, nous avons opté la librairie poltly. Avec cette dernière, nous avons réussi à réaliser des nuages de points interactifs, mais la réalisation des graphes était un peu trop compliquée : nous souhaitions une méthode plus simple. Après de longues recherches, nous avons trouvé la librairie Pyvis : une bibliothèque destinée à la génération rapide de graphes de réseau visuels avec un code python minimal.

Pour cela, il a suffi de créer un Network de la librairie Pyvis avec les options de celui-ci (taille, couleur de fond…). Ensuite, on prend en entrée du Network Pyvis le graphe statique crée auparavant. Encore une fois restant sur la base de DataFrame, l’intégration des différents développements fut simple.

Nous avons donc fait fonctionner tout ça dans une fonction graphe qui pend en entrée comme pour les requêtes : le DataFrame, la source et la destination. Dans cette fonction, nous avons pu ajouter toutes les options visuelles possibles par cette librairie.  
  
Option très intéressante les informations liées aux nœuds lorsque la souris passe dessus : elle a nécessité une fonction en plus.

**La fonction Info\_node**

*def Info\_node(ask,name):*

*if ask == "id\_author" :*

*….*

*if ask == "id\_publication":*

*.…*

Cette fonction récupère les données attachées aux choix, et renvoie l’argument « title » (éléments associés à l’id) de la fonction get\_net\_nodes de la librairie Pyvis

**Par la suite,** pour un souci de visuel, mais aussi d’intégration avec les autres groupes, nous avons pris la décision d’afficher nos résultats sur une page web. Tout serait sur la même page : le formulaire pour les utilisateurs, les DataFrame et le graphe.  
  
Pour réaliser tout ça nous somme passé par la librairie Flask. Flask est un Framework de développement web en Python, simple et c’est à nous de lui coder/rajouter ces composants à la main. Il est aussi très utilisé et ne risque pas de disparaître de sitôt.  
  
Avant de commencer faut bien savoir que nous travaillions en local c’est-à-dire que la page web sera situé sur la même machine que le code. Notre serveur HTTP intégré à Flask, fonctionne sur le port 10500, il faut donc le préciser dans l’adresse du navigateur. Ce qui donne au final l’adresse http://localhost:10500/unepage pour accéder à la page /unepage .  
  
La première chose à faire est d’importer le module Flask

*from flask import flask*

*app=flask(\_\_name\_\_)*

et donner un nom à notre application

Partie cruciale : définir une page (ou route) :

**@app.route** permet de préciser a quelle adresse ce qui suit va s’appliquer. Comme nous somme sur la page d’accueil donc nous mettons (“/”)

En ouvrant notre page web la fonction home.page va être appelé et notre Template dblp\_Projet.html s’exécute.

*def home\_page():*

*return render\_template('dblp\_Projet.html')*

La méthode FORM de l’objet REQUEST contient toutes les données du formulaire envoyé en POST. La méthode FORM est un dictionnaire qui associe la valeur à l’attribut name du champ du formulaire. Pour récupérer « les attributs » de la requête, nous avons utilisé request.form.get(‘valeur bouton’)

*@app.route('/submit', methods=['GET', 'POST'])*

*def submit():*

*#if request.method == "POST":*

*if request.form.get('author'):*

*Author(getinputs())*

*elif request.form.get('publication'):*

*Publication(getinputs())*

*elif request.form.get('venue'):*

*Venue(getinputs())*

*elif request.form.get('keyword'):*

*Keywords(getinputs())*

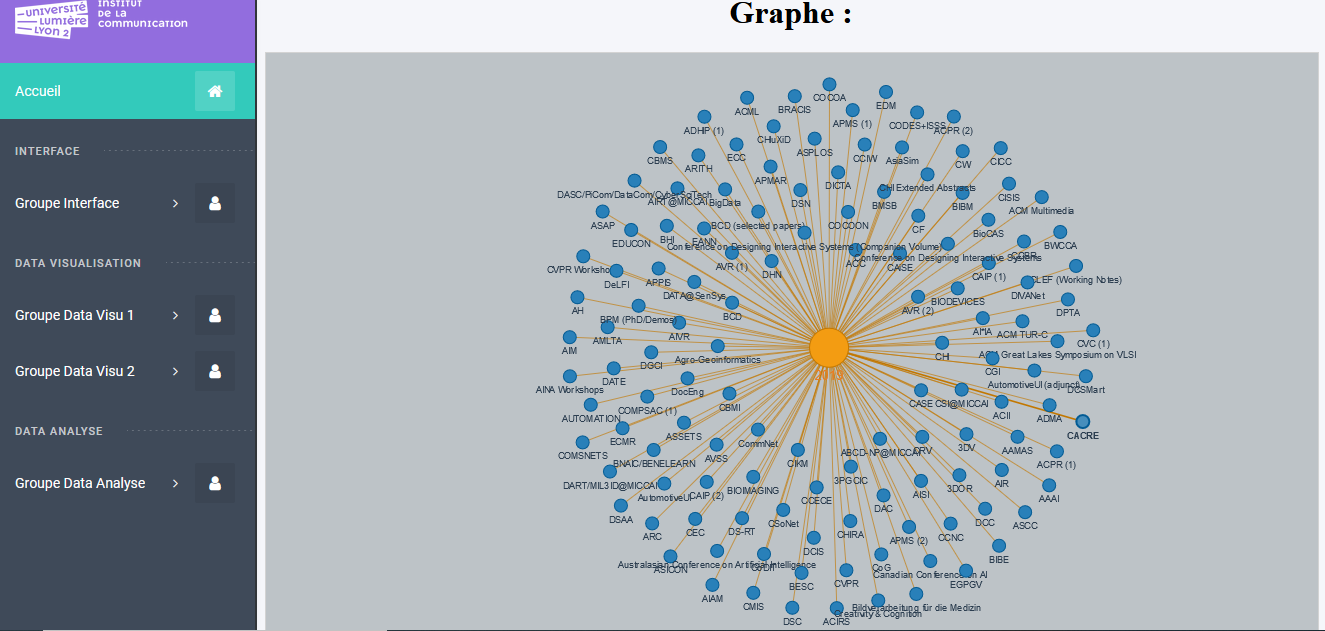
*#elif request.form.get('more\_info'):*

*# More\_info(getinputs())*

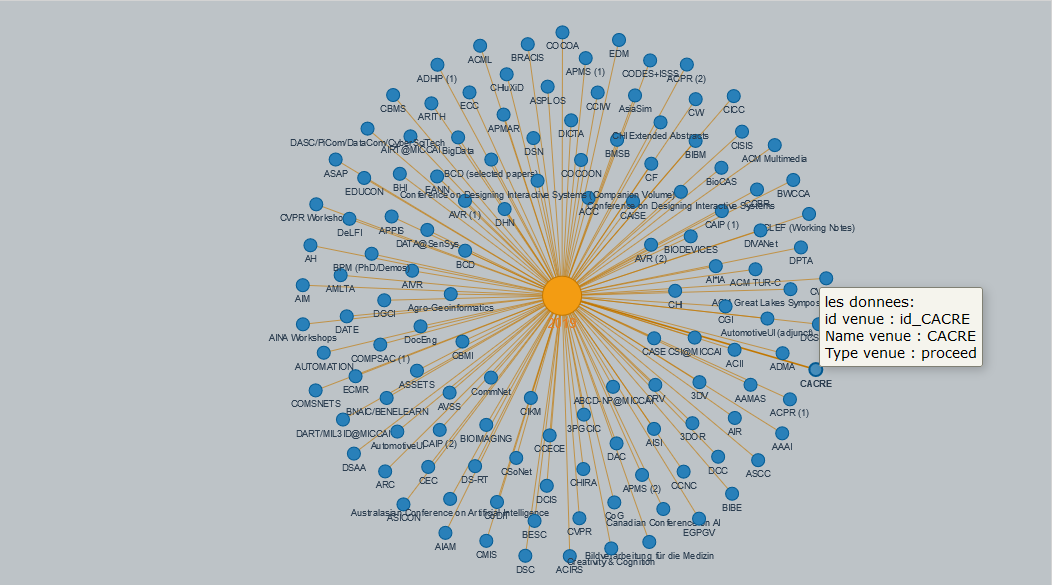
app.run(port=”10500”)

Finalement on fait tourner l’application

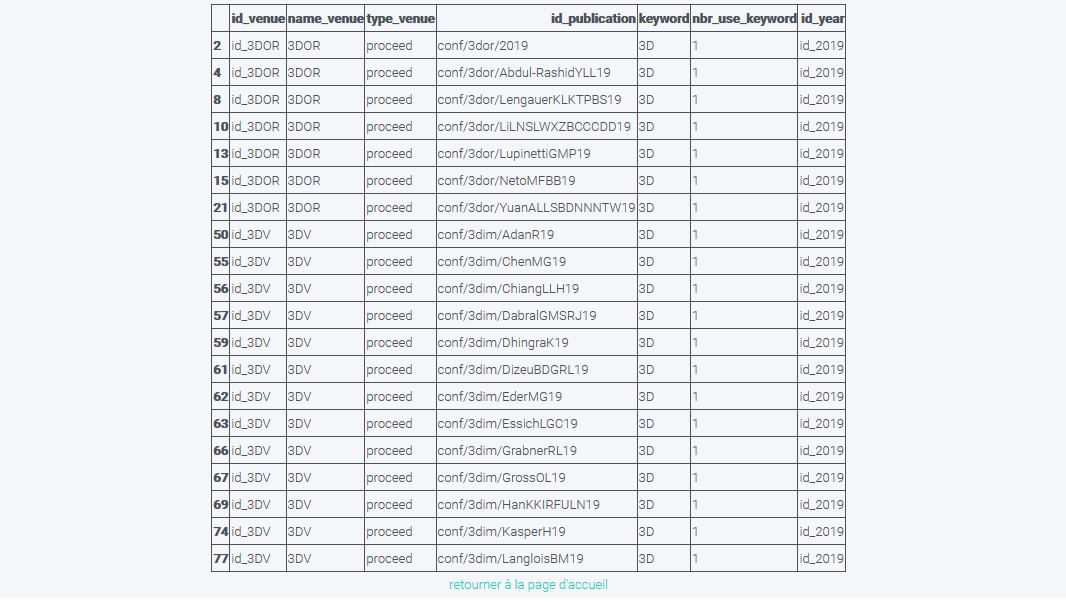
# Visualisation des options :



* Le graphe est dynamique : on peut zoomer, déplacer les nœuds, faire bouger le graphe en entier.
* Si on fait un clic droit sur le graphe nous pouvons enregistre l’image sous format .png



* Pour plus d’informations sur un nœud, poser votre souris sur le nœud en question.



* Visualisez le tableau correspondant à la requête de l’utilisateur.
* Un bouton est disponible pour retourner à la page de filtrage.

# CONCLUSION

**Conclusion et perspectives :**

Finalement nous obtenons un site web grâce à la création d’un serveur python en passant par la librairie Flask. Cette interface comprend des redirections vers les différentes interfaces des autres groupes. Dans notre menu une première page s’ouvre donc et permet à l’utilisateur de créer sa requête, puis en un clic la page des requêtes s’ouvre. Nous sommes satisfaits de tout afficher sur une page même si nous avons mis du temps à y parvenir

Finalement ce projet nous aura à tous beaucoup appris, tant par la manière de travailler que sur la programmation. En effet ce travail en groupe a permis pour certains d’entre nous de remettre en question notre manière de penser un projet, de chercher les informations, de distribuer les tâches.

Le temps a selon nous été optimisé, nous sommes fières du résultat produit même si avec plus de temps nous aurions apprécier faire plus, développer plus de fonctionnalités et d’options pour l’utilisateur.

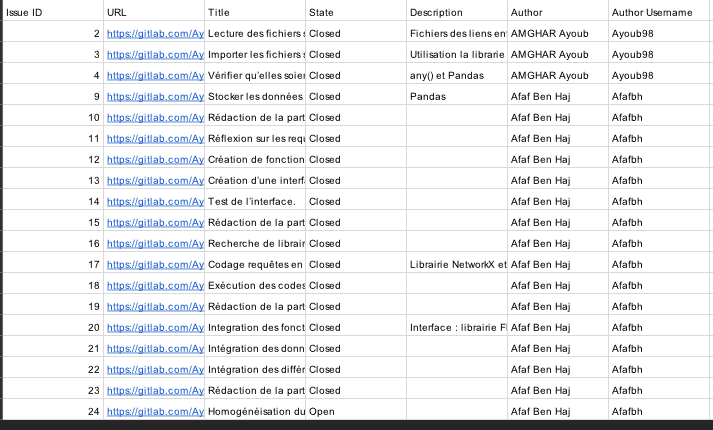
# ANNEXE

**Annexe 1 : Proposition de requêtes**

* Affichage de la liste des co-auteurs pour un auteur donné
* Affichage des mots-clés utilisés par un auteur donné
* Affichage des publications d'un auteur donné
* Affichage des auteurs d'une publication donnée
* Affichage des lieux de publication pour un auteur donné
* Affichage des 10 mots-clés les plus utilisés pour un lieu de publication donné
* Détection de communautés d’auteurs par rapport aux thèmes traités
* La publication id\_pub elle vient de quelle name\_venue ?
* L'auteur id\_autor sur quelles name\_venue il a travaillé ?
* Quels sont les keywords sortis en telle annee year ?
* Pour une pub\_Keywor combien d'auteurs ont utilisé le même keyword ?
* En year quels authors ont publié ?
* Nombre de keyword utilisé par communauté
* Publication dans une année
* Affichage des 10 mots-clés les plus utilisés pour un auteur donné
* Mots clé les plus utilisés ces 5 dernières années (tranche d’années)
  + Mots clé les plus utilisés dans une année
* Affichage des données produisant le plus grand nombre de publication pour une année
* Auteur selon une catégorie (champ catégorie de publication)
* Les auteurs les plus actifs sur DLPB
* Publication où 3 (paramètre) auteur et plus ont collaborés
* Affichage des publications d'un auteur donné
* Affichage des auteurs d’une venue
* Auteur ces 5 dernières années (tranche d’années)
  + Auteur dans une année
* Nombre de publication selon l’édition (venue)
* Type de venue selon la publication
* Communauté de keyword
* Nom d’auteur dans une date de publication
* Les publications publiées en 2018 d'un auteur donné
* Les auteurs d'une publication publié à <venue>
* Les venues des publications d"un auteur donnée en 2018
* Les années des publications utilisant les keywords suivant : <keywords..>
* Les venues des publications utilisant le keyword : <keyword>

\*Exemple de requête du cours

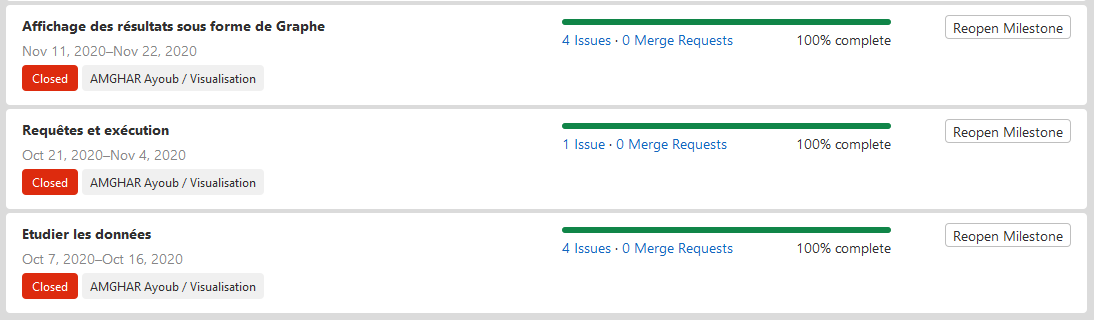
**Annexe 2 : Product Backlog**





**Annexe 3 : Sprints**





# BIBLIOGRAPHIE

Rakotomalala R., Tutoriel Tanagra, "[Python : manipulation des données avec Pandas](http://tutoriels-data-mining.blogspot.fr/2012/08/manipulation-des-donnees-avec-r.html)", 21 février 2017. Consulté sur

<http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_Data_Manipulation_Pandas.pdf>

Interactive Visualizations In Jupyter Notebook, written by 5agado Data Scientist @ Zalando Dublin - Machine Learning, Computer Vision and Everything Generativ. Consulté sur : https://towardsdatascience.com/interactive-visualizations-in-jupyter-notebook-3be02ab2b8cd

Chaîne YouTube : Alex Martinelli, 2017. Demo of Jupyer as dashboard for Fitbit sleep data. Consulté sur : <https://youtu.be/FYnM84TgzZU>

Tutorial edit on GitHub, consulté sur : https://pyvis.readthedocs.io/en/latest/tutorial.html

The pandas development team.Merge, join, concatenate and compare. 2008-2020. Consulté sur <https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html>

NetworkX Developers, from\_pandas\_dataframe, 2015. Consulter sur <https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.convert_matrix.from_pandas_dataframe.html>

[Jonathan Soma](http://jonathansoma.com/), 2017. NetworkX Graphs from Source-Target DataFrame, Algorithms, Lede 2017. Consulté sur <http://jonathansoma.com/lede/algorithms-2017/classes/networks/networkx-graphs-from-source-target-dataframe/>

Lothon F. [Exemple d’organisation projet Agile](https://agiliste.fr/exemple-dorganisation-projet-agile/) (2013) consulté sur : https://agiliste.fr/exemple-dorganisation-projet-agile/