## **TEMPS DE PAROLE 1min 🡪** Afaf : Bonjour alors pour commencer je vais présenter l’équipe rapidement, on a Mohamed qui est le Product Owner, Taha, Ayoub, Libasse et Bochra qui sont l’équipe de développement et moi-même Afaf, Scrum Master

Pour démarrer le travail nous avons chargé nos fichiers sur un interpréteur Python Après ouverture de nos fichiers CSV et les avoir stockés dans un DataFrame grâce à la librairie Pandas : librairies de base pour la data science en Python on a pris en note les différentes informations importantes comme la séparation des données, le format des données, les en-têtes des colonnes, on a trouvé l’encodage des fichiers etc. On a aussi utilisé différentes commandes lors de l’import des fichiers pour donner des résultats de données cohérentes et sans mauvaise surprise notamment sur le fichier « venue » Lors du passage sur les données du groupe interface il n’y pas grand-chose qui a changé, l’encodage n’était pas le même, le séparateur de colonnes également mais mai la data était quasiment la même

* TAHA MONTRE EN MEME TEMPS EN PARTAGE LES READ.CSV DES FICHIER DE FADILA

**TEMPS DE PAROLE 1min 🡪** Mohamed :

Nous avons démarré le requêtage en se questionnant sur les requêtes intéressantes à présenter du point de vue de l’utilisateur.

Toujours grâce à la librairie Pandas et l’utilisation de la fonction merge de celle-ci, nous avons pu faire la liaison entre des fichiers csv.

Finalement en réfléchissant du point de vue du client on s’est rendu compte qu’afficher des requêtes proposées que l’on proposé nous était moins intéressant et que se serai plus intéressant de les rendre dynamique / interactive pour le client. MOI

C’est grâce à la création d’une « Interface Test » qu’on a pu se questionner sur l’exécution des requêtes.

**TEMPS DE PAROLE 1m30 🡪** Ayoub : Fonction

La création des fichiers

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 vérifiés sont les noms des cases nommées dans notre code d’interface.

Création des fonctions get\_id :

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

Pour la fonction Get\_id\_venue(names) on prend en paramètre un ou plusieurs noms sépares avec une virgule () et split(« , ») retourne un ou plusieurs id

Création de la fonction revealcsv(df) prend en paramètre un dataframe :

Cette fonction permet de

Les fonctions de recherche : Ces fonctions prend en paramètre le dictionnaire remplie avec les inputs :

1/ def Publication(dict) :

2/ def Keywords(dict) :

3/ def Author(dict) :

4/def Venue(dict) :

5/def More\_info(dict) :

**TEMPS DE PAROLE 2min 🡪** Taha : Fonction (Graphique explication)

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.

* EXPLICATION DU GRAPHIQUES + DEMONSTRATION DE LA PAGE 1 : localhost :5000
  + - Préparation des données d’entrées et des informations à afficher

**TEMPS DE PAROLE 1min30 🡪** Libasse : NetworkX

Au début on a eu du mal à trouver une librairie qui rendrait le graphe dynamyque pour l’utilisateur donc on a pris le temps de bien maitriser l’affichage des graphe statique sur python 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 fouiller la documentation de Networkx et importé la librairie

La construction du graphe est réalisée grâce à la fonction from\_pandas\_edgelist qui prenne en entrée un dataframe ce que nous avons produit grâce aux fonction de requête, et deux attributs de ce dernier : la source et la destination

L’affichage du graphe avec nx.draw. Cette fonction prend en paramètres le graphe construit

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

Mais en affichant le graphe avec nx.draw nous obtenons un graphe statique

**TEMPS DE PAROLE 2min 🡪** Bochra : Pyvis / Flask

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.

Après de longue recherche 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 ces options (taille…) et de prendre en entrée du Network Pyvis le graphe statique crée auparavant.

Nous avons fait fonctionner tous 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 possible par cette librairie (couleur, information lié aux nœud lorsque la souris passe dessus

* TAHA MONTRE EN MËME TEMP LES FONCTIONNALITE DU GRAPHE.

Nous affichons également le tableau lié à la requête

Tout ça dans une interface web : 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.

Notre serveur HTTP Flask fonctionne sur le port 10500, il soit l’adresse [http://localhost:5000/](http://localhost:5000/unepage)

Pour commencer le développement :

definir la page (ou route). En ouvrant notre page web la fonction home.page va etre appeler et notre template dblp\_Projet.html c’est une page Html que nous avons crée qui contient note formulaire. Nous faisons ensuite appel à la page html requête de la clique sur un bouton.

**TEMPS DE PAROLE 1m30 🡪** Afaf : Avant de passer aux questions, nous avons préparé une petite publicité pour présenter le produit + Vidéo