



Projet Ingénieur

Peio LOUBIERE

2022 – 2023

Nicolas DURAND
Lucas FERNANDES
Jérémi LIOGER-BUN
Lilian MICHEL-DANSAC
Matt COSTE

CY Tech

Table des matières

I. Introduction	3
II. Notre organisation	3
III. Stockage des données	3
1. Le Modèle conceptuel de données	4
2. La base de données	4
a. Gestion de la connexion à la base de données	5
b. Récupération des données de la base de données	5
c. Modification des données de la base de données	6
IV. Les fonctionnalités implémentées	6
1. Compte utilisateur	6
a. Inscription et connexion	6
b. Profil	7
2. Administration des utilisateurs	7
3. Administration des data challenges	9
4. Gestion des data challenges	9
5. La messagerie	10
a. Contacter un nouvel utilisateur	10
b. Contacter les utilisateurs	11
6. Gestion des groupes	11
7. L'API en Java	11
a. La classe <i>CodeAnalyse</i>	11
b. Partie serveur de la classe Java	14
c. Visualisation des résultats	15
V. Problèmes rencontrés	18
1. La messagerie	18
VI. Bilan du projet	19

Liste des images

1	MCD	4
2	Base de données IAPau	5
3	Inscription	6
4	Connexion	7
5	Gestion et création d'un utilisateur	7
6	Modification des informations d'un utilisateur	8
7	Création d'un data event	9
8	Diagramme de classes temporaire	12
9	Diagramme de classes final	15
10	Visualisation des résultats 1	16
11	Visualisation des résultats 2	17
12	Visualisation des résultats 3	18

I. Introduction

Dans le cadre de notre première année du cycle ingénieur, il nous est proposé la réalisation d'un projet pour mettre en pratique nos connaissances. Ce projet consiste en la réalisation d'une application pour IA Pau permettant la création et l'administration de data challenges.

Le dépôt git se trouve au lien suivant : <https://github.com/blj64/projetIng1.git>

II. Notre organisation

Afin de réaliser ce projet, nous avons dû travailler en équipe. Ce défi peut effrayer mais avec un peu d'organisation et de travail, nous sommes arrivés à un résultat plutôt convaincant.

Le premier jour, nous avons tous déclaré nos préférences au niveau du code. Lilian et Nicolas souhaitaient travailler sur le Backend, Matt sur l'analyseur de code et l'API Java, Jérémie s'occuperait de la messagerie et Lucas du Frontend. Nous avons ensuite réalisé un mld pour la base de donnée et un diagramme afin de structurer le site.

III. Stockage des données

L'utilisation d'une base de données dans le contexte de ce projet est évident pour pouvoir stocker diverses informations comme celles concernant les utilisateurs ou les data challenges. Ainsi, nous avons réalisé un modèle conceptuel de données (MCD) pour réaliser ensuite la structure de la base de données.

1. Le Modèle conceptuel de données

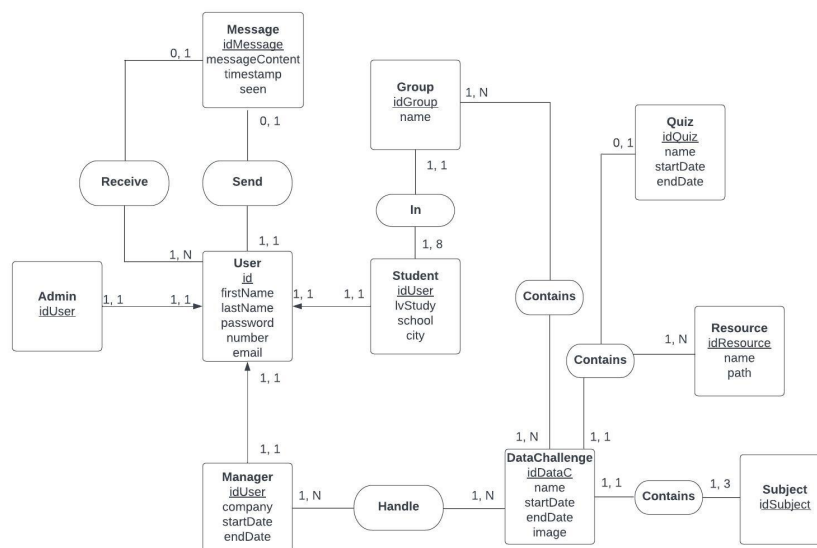


Figure 1: MCD

Un utilisateur peut être soit un administrateur, un gestionnaire ou un étudiant (un participant). Un ensemble d'utilisateurs quelconques peut recevoir un message donné et un utilisateur peut aussi envoyer un message. On peut considérer qu'un étudiant puisse envoyer un message à un administrateur sous des conditions particulières (requête pour résoudre un bug, etc) qui peuvent être gérées lors du développement de l'application. Un même groupe peut participer à plusieurs data challenge en même temps d'où la cardinalité dans le MCD.

Cependant, tous les data challenges ont des sujets différents, des ressources différentes et s'ils contiennent un quiz, un quiz différent les uns des autres. D'où la cardinalité (1, 1) du côté de la table DataChallenge. Enfin, on considère qu'un gestionnaire gère le data challenge complet (on aurait pu définir des gestionnaires qui gèrent seulement un ou des sujets d'un data challenge).

2. La base de données

Nous avons choisi d'utiliser la base de données MySQL pour stocker les informations nécessaires.

Nous avons pris parti d'inclure l'identifiant des data challenge en tant que clé étrangère dans les tables Quiz et Resource puisqu'elles sont uniques à chaque data challenge ainsi qu'à la table Group puisque le cas d'un même groupe qui participe à plusieurs data challenges en même temps reste un cas très particulier.

Ci-dessous une visualisation de la base de données utilisée :

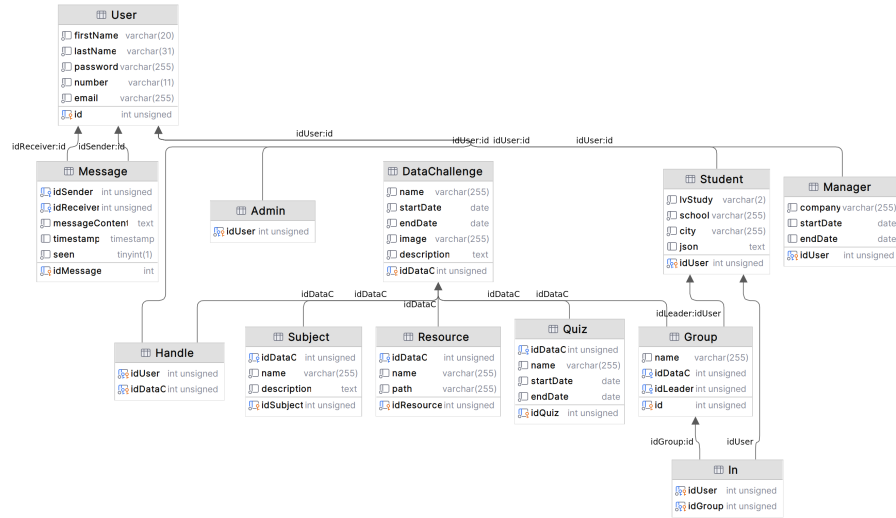


Figure 2: Base de données IAPau

a. Gestion de la connexion à la base de données

Plusieurs fonctions php pour gérer la connexion à la base de données sont présentes dans le fichier bdd.php.

- connect_db(\$host = DB_HOST, \$user = DB_USER, \$pass = DB_PASS, \$db = DB_NAME)
- is_connected_db()
- disconnect_db()

Ces fonctions servent respectivement dans le cas général à se connecter à la base de données, vérifier si on est connecté et enfin se déconnecter.

b. Récupération des données de la base de données

Pour simplifier l'aspect front et la création des diverses pages nécessaires, plusieurs fonctions ont été codées pour accéder aux informations stocker dans la base de données. Ces fonctions dans bdd.php sont identifiables par un "get" au début du nom de la fonction. De plus, une fonction request_db(\$dbRequestType, \$request = null) permet dans le cas général d'envoyer une requête à la base de données.

c. Modification des données de la base de données

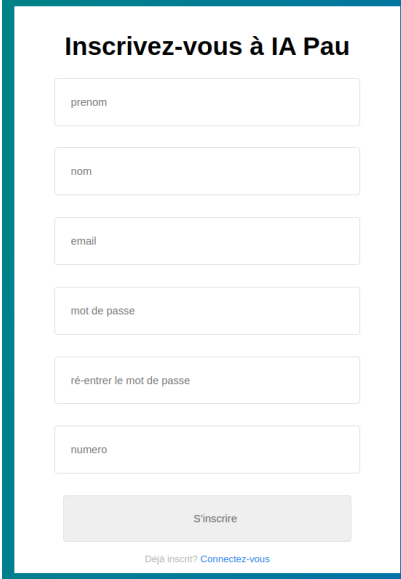
De même, de multiples fonctions ont été codées pour modifier les informations stockées dans la base de données toujours dans le but de simplifier l’aspect front de l’application. Ces fonctions dans bdd.php sont identifiables par un “alter” au début du nom de la fonction. Il existe par exemple une fonction pour modifier les informations d’un user ou data challenge.

IV. Les fonctionnalités implémentées

Les fonctionnalités qui ont pu être implémentées dans l’application sont présentées dans cette partie.

1. Compte utilisateur

a. Inscription et connexion



The image shows a registration form titled "Inscrivez-vous à IA Pau". It contains several input fields: "prenom", "nom", "email", "mot de passe", "ré-entrer le mot de passe", and "numero". Below these fields is a grey button labeled "S'inscrire". At the bottom, there is a link that says "Déjà inscrit? [Connectez-vous](#)".

Figure 3: Inscription

Le menu d’inscription pour un utilisateur de type élève. Les gestionnaires sont quant à eux à ajouter manuellement par un administrateur (voir : 2. Administration des utilisateurs).

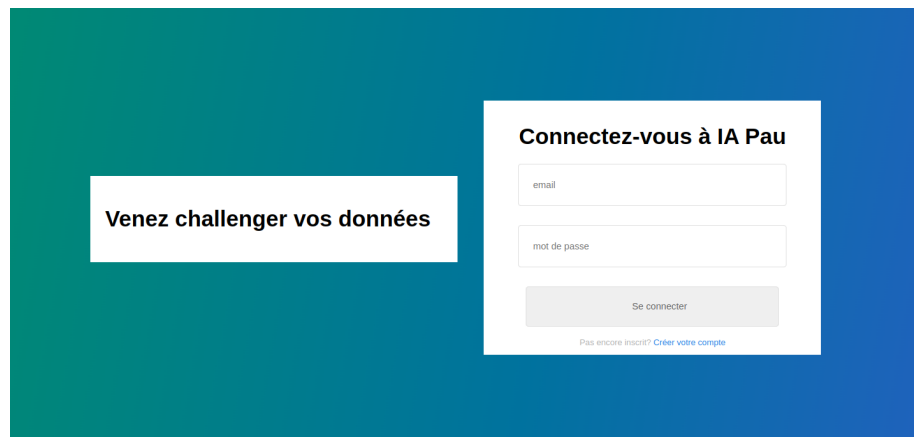


Figure 4: Connexion

Tous les utilisateurs peuvent se connecter depuis ce menu. Une redirection vers la page de connexion depuis la page d'inscription et inversement est implémentée.

b. Profil

Tous les types d'utilisateur ont accès à leur propre profil avec la possibilité de modifier certaines informations de base comme l'adresse e-mail, le mot de passe, le nom, etc.

2. Administration des utilisateurs



Figure 5: Gestion et création d'un utilisateur

L'accès à ce menu est réservé aux administrateurs. La gestion de tous les utilisateurs et la possibilité d'en créer un nouveau (menu côté droit) sont possibles.

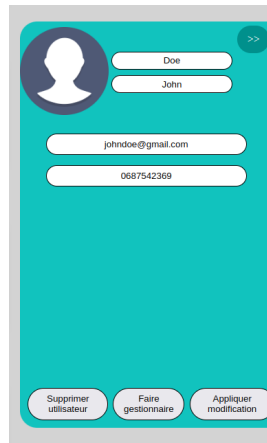
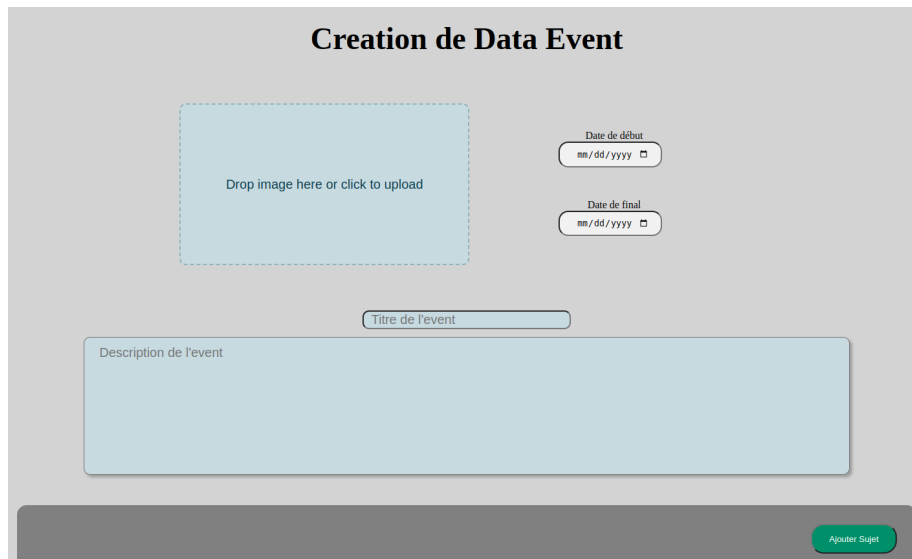
The image shows a mobile application interface for user management. It features a teal background with a white silhouette of a person's head and shoulders on the left. To the right of the silhouette are two input fields: the first contains 'Doe' and the second contains 'John'. Below these are two more input fields: the first contains 'johndoe@gmail.com' and the second contains '0687542369'. At the bottom of the interface are three buttons: 'Supprimer utilisateur', 'Faire gestionnaire', and 'Appliquer modification'. A small '>>' button is located in the top right corner.

Figure 6: Modification des informations d'un utilisateur

La modification des informations d'un utilisateur, le supprimer ainsi que la possibilité de créer un gestionnaire à partir d'un utilisateur. À noter qu'un administrateur peut tout de même être supprimé depuis cette interface.

3. Administration des data challenges



The screenshot shows a web form titled "Creation de Data Event". It features a large dashed blue box for image upload with the text "Drop image here or click to upload". To the right are two date pickers labeled "Date de début" and "Date de final", both showing the format "mm/dd/yyyy". Below these is a text input field for "Titre de l'événement" and a larger text area for "Description de l'événement". At the bottom right, there is a green button labeled "Ajouter Sujet".

Figure 7: Création d'un data event

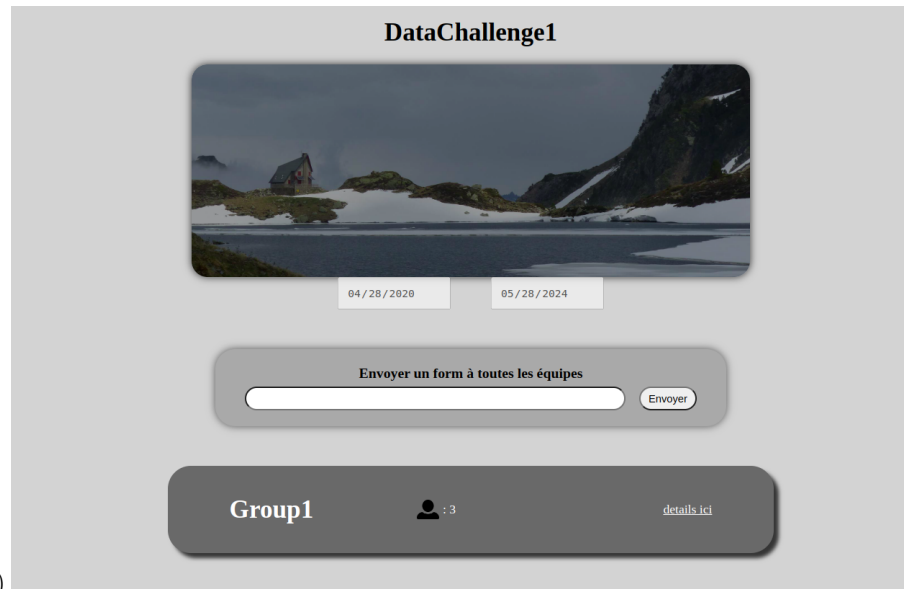
Le menu de création d'un data challenge avec ses informations de base. L'ajout de sujets est possible dans la limite de 3 sujets avec chacun son nom et sa description. La création et la suppression (depuis le menu Data Challenges) peut seulement être effectuées par un administrateur

4. Gestion des data challenges

La liste de tous les data challenges sont visibles dans le menu Data Challenges. Les administrateurs peuvent modifier les informations de tous les data challenges alors que les gestionnaires peuvent seulement le faire pour les data challenges dont ils s'occupent.

(TBA afficher l'id d'un DataC pour l'admin) 333 (TBA when updated) (Pour l'instant on a : le menu du suivi des dataC associés à un manager, les équipes pour ce dataC et la possibilité d'envoyer un quiz (form))

(data challenge côté manager, l'aspect gestion)



(update image)

(Attention non implémentée) (changer de partie si nécessaire) Cette page permet à un manager de visualiser toutes les équipes pour un data challenge donné. De plus, si nécessaire il peut aussi envoyer un quiz aux équipes et voir leurs membres.

5. La messagerie

La liste de tout les personnes avec qui l'utilisateur a une conversation est affiché en colonne dans la partie gauche. Il est possible d'afficher la conversation en cour avec eux en cliquant dessus. La liste s'actualise automatiquement. Les messages sont stockés dans la base de données dans la table message. Chaque message possède un id, l'id de celui qui envoie le message, celui qui le recoit, le message, sa date et son heure qui sont générées automatiquement ainsi qu'une variable equivalente à un type boolean qui prend 0 ou 1 en fonction de si le message a été lu ou non.

a. Contacter un nouvel utilisateur

Le bouton nouveaux message sert a contacter un nouvel utilisateur. Il inclut dans la zone ou les messages apparaissent normalement un input ou l'utilisateur sélectionne la personne qu'il souhaite contacter. A partir de la saisie de l'utilisateur, le serveur est censé proposer les noms des personnes contactables en créant une datalist dynamique mais cette fonctionnalité ne marche pas très bien. Ensuite il suffit juste de rédiger le message et de l'envoyer pour finaliser la création d'un échange avec une nouvel personne.

b. Contacter les utilisateurs

Comme dit précédemment, pour parler avec une personne déjà contactée, il suffit de sélectionner l'utilisateur dans la liste à gauche. La liste de tous les messages va donc s'afficher dans leur ordre d'envoi. Ils sont également séparés en deux parties, à gauche ceux reçus et à droite ceux envoyés. La messagerie est instantanée, chaque seconde une fonction va rechercher les messages qui n'ont pas été vus. Cette méthode est efficace mais bien trop énergivore et peu adaptable dans le cas d'un vrai site. Nous avons en revanche trouvé une méthode plus efficace: HTTP Long Polling. L'utilisateur envoie une requête au serveur et tant que le serveur ne détecte pas de changement il ne renvoie pas de requête la ou dans notre solution initiale il la renvoie immédiatement et ainsi de suite toutes les secondes. Par souci de temps, nous n'avons pas adapté la méthode HTTP Long Polling.

6. Gestion des groupes

555 (maybe available later ?)

(images du menu groupe (celui student leader suffit) avec un pour chaque partie : groupe, messagerie, paramètre, rendu avec graphe)

7. L'API en Java

Dans cette partie, il nous était demandé de réaliser un analyseur de code source Python en Java, pour ensuite créer un web service REST sous forme d'API en Java. Ainsi, nous allons expliquer comment nous nous sommes débrouillés pour répondre à cette demande en détaillant les étapes.

a. La classe *CodeAnalyse*

Nous avons tout d'abord créé une classe *CodeAnalyse* qui avait comme seuls attributs :

- *path* de type **String** correspondant au chemin vers le fichier Python
- *file* de type **File** correspondant au fichier Python
- *br* de type **BufferedReader** permettant de lire le fichier à partir d'un flux de symboles
- *nb_lignes* de type **int** correspondant au nombre de lignes du fichier
- *text[][]* de type **String**, un tableau de tableaux de **String** qui permet d'accéder au texte complet ligne par ligne et mot par mot au sein de chaque ligne

Puis, nous avons créé deux constructeurs de classe, l'un à partir d'un fichier Python de type **File** et l'autre à partir d'un chemin vers un fichier Python de type **String**. Voilà un des constructeurs :

```
public CodeAnalyse(File fileInsert) throws IOException {  
    file = fileInsert;  
}
```

```

    path = fileInsert.getPath();
    br = new BufferedReader(new FileReader(file));
    taille_texte();
    text = new String[nb_lignes] [];
    splitTab();
}

```

splitTab() permet d'initialiser l'attribut *text[][]* en séparant le texte du flux de l'attribut *br*. *taille_texte()* permet d'initialiser l'attribut *nb_lignes* au nombre de lignes du texte à partir de l'attribut *br*.

Il n'y a aucune vérification de l'extension du fichier au sein de la classe parce que la vérification est gérée du côté web lorsque l'utilisateur s'occupera d'upload son fichier. Nous n'avons qu'un seul getter qui est *getNb_lignes()* puisqu'il est le seul utile pour un utilisateur.

Suite à cela, nous avons créé plusieurs méthodes au sein de notre classe, certaines privées qui ne sont pas utiles pour l'utilisateur et d'autres publiques car essentielles pour ce dernier. Voici notre diagramme de classes que l'on avait à ce moment-là :

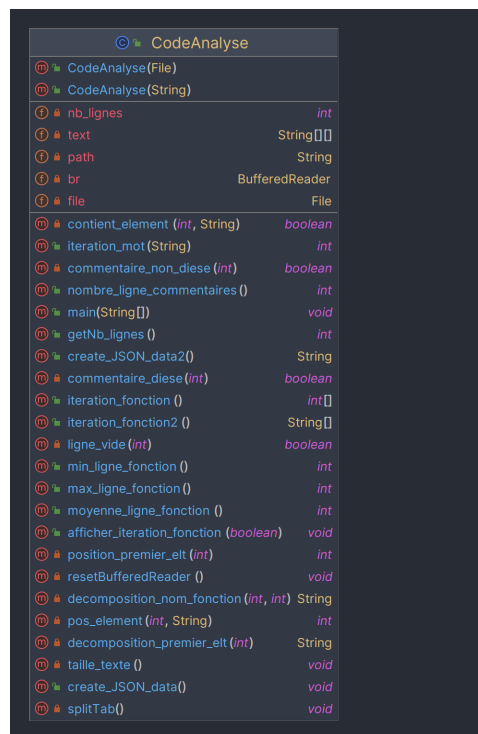


Figure 8: Diagramme de classes temporaire

Nous avons essayé de rendre le nom des méthodes le plus clair possible cependant, si vous souhaitez avoir plus d'informations sur chacune d'elles, elles sont toutes commentées et vous pouvez vous rendre sur la **Javadoc** (le lien fonctionne si vous lisez le rapport depuis le repertoire du projet, sinon il faudra s'y rendre manuellement).

Voilà un exemple de fonctionnement de notre classe (cette partie provient du **main** qui est une partie commentée pour laisser place à la gestion du serveur) :

```
/* Création de l'instance de ma classe à partir du fichier
nommé "test.py" */
CodeAnalyse code = new CodeAnalyse("src/test.py");

/* Première fonction permettant de générer le JSON avec
toutes les data nécessaires */
code.create_JSON_data();

/* Deuxième fonction permettant de prendre un mot et d'obtenir
le nombre d'itérations de celui-ci */
int iteration_mot_def = code.iteration_mot("def");

/* Démonstration de toutes les fonctions de la classe et les
différents affichages */

/* Affichage des noms de fonctions */
System.out.print("Tableau des noms de fonctions : ");
code.afficher_iteration_fonction(true);

/* Affichage des tailles de fonctions */
System.out.print("Tableau des tailles de fonctions : ");
code.afficher_iteration_fonction(false);

/* Affichage de la taille maximale */
System.out.println("Taille maximale : " + code.max_ligne_fonction());

/* Affichage de la taille minimale */
System.out.println("Taille minimale : " + code.min_ligne_fonction());

/* Affichage de la moyenne */
System.out.println("Taille moyenne : " + code.moyenne_ligne_fonction());

/* Affichage du nombre de commentaires */
System.out.println("Nombre de commentaires : " + code.nombre_ligne_commentaires());
```

```

/* Affichage du nombre de lignes totales */
System.out.println("Nombre de lignes totales : " + code.getNb_lignes());

/* Affichage du nombre de lignes de code */
int taille_code = code.getNb_lignes() - code.nombre_ligne_commentaires();
System.out.println("Nombre de lignes de code : " + taille_code);

```

Voilà l’affichage correspondant :

```

Tableau des noms de fonctions : [fRepartition ; test ; test ; graphique ; __str__ ;
variance ; esperance ; __init__ ; test ; f_inv2 ; f_inv ; S_n]
Tableau des tailles de fonctions : [7 ; 2 ; 2 ; 4 ; 2 ; 1 ; 2 ; 4 ; 24 ; 2 ; 2 ; 5]
Taille maximale : 24
Taille minimale : 1
Taille moyenne : 4
Nombre de commentaires : 17
Nombre de lignes totales : 116
Nombre de lignes de code : 99

```

b. Partie serveur de la classe Java

Après s’être occupé de la classe *CodeAnalyse* avec toutes les fonctionnalités de base pour analyser le code, nous nous sommes occupés de l’implémentation de la partie serveur. Ainsi, nous avons ajouté les attributs suivants :

- *SERVEUR* de type **String** qui représente l’url de base du service
- *PORT* de type **int** qui représente le port du serveur
- *URL* de type **String** qui représente l’url de base du service
- *LOGGER* de type **Logger** qui permet de gérer les messages émis durant l’exécution du serveur

Le *main*, qui servait à faire la démonstration de la classe et de son fonctionnement, sert désormais de boucle principale permettant au serveur de démarrer et de fonctionner jusqu’à l’arrêt du fonctionnement de la classe. Nous avons dû nous occuper également des différentes méthodes permettant de gérer les requêtes (POST, GET qui proviennent du serveur web), de les traiter et d’envoyer une réponse. Pour la méthode POST, nous récupérons un fichier Python provenant d’un form HTML et nous lui appliquons la méthode permettant de générer une chaîne de caractères en format JSON et nous envoyons à la page web cette dernière. Pour la méthode GET, nous ne la gérons pas puisque nous ne faisons aucune requête GET au sein de notre site web.

Voici donc le diagramme de classes final que nous avons :

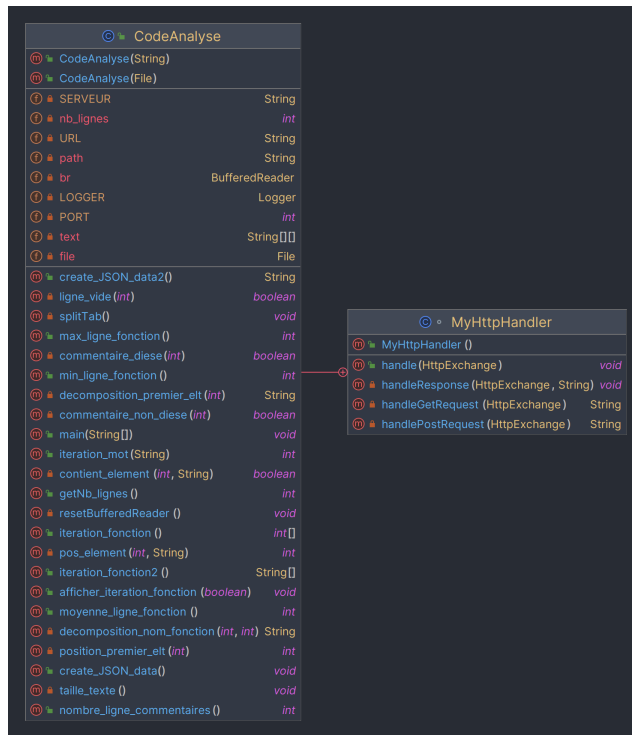


Figure 9: Diagramme de classes final

Pour finir, il y a un *README* dans le dossier *java* contenant les instructions à réaliser pour lancer le serveur Java. Vous pouvez retrouver ces mêmes instructions dans le *README* global avec les instructions pour lancer le serveur PHP.

c. Visualisation des résultats

Pour visualiser les résultats sur notre site, nous utilisons une librairie nommée ChartJS ce qui nous permet de créer des graphiques visuellement beaux et dynamiques à partir d'un JSON. Voilà à quoi ressemble la visualisation des résultats d'un fichier Python (le fichier *test.py* présent dans le dossier *java*) :

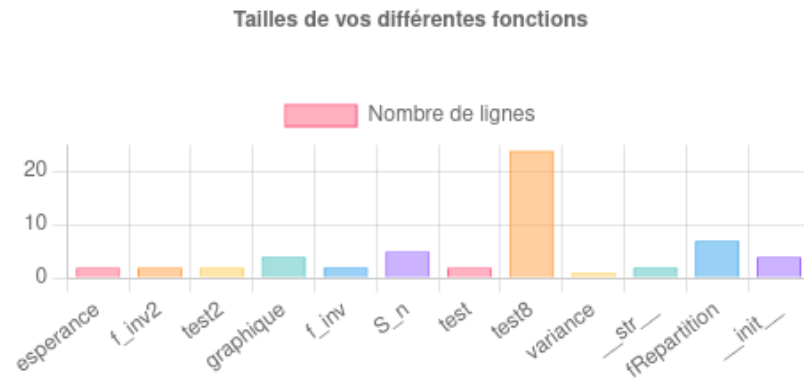


Figure 10: Visualisation des résultats 1

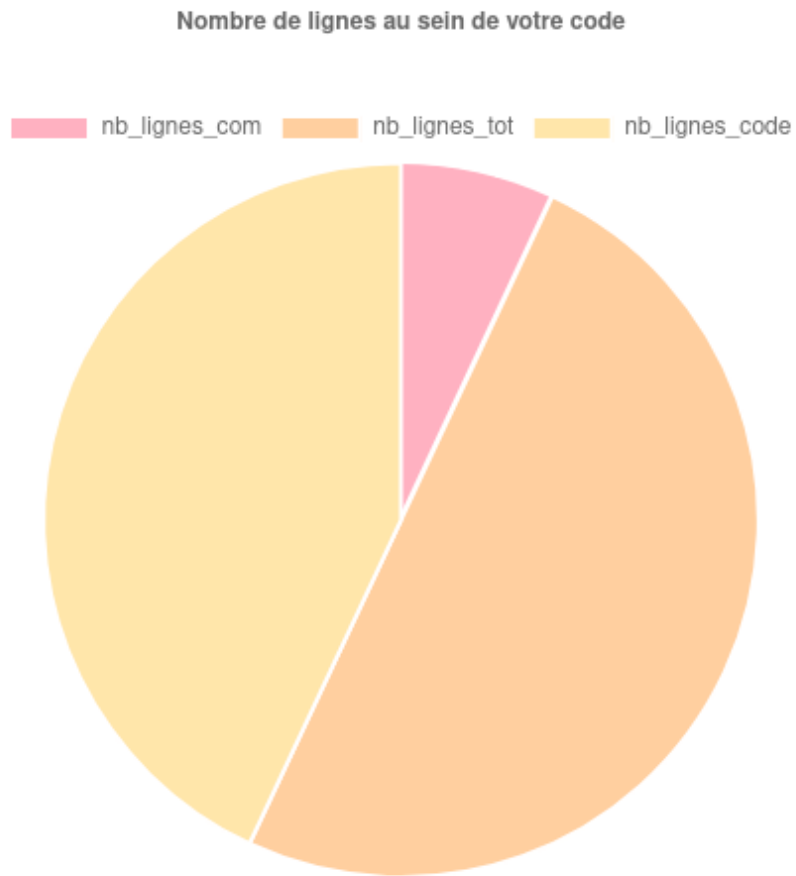


Figure 11: Visualisation des résultats 2

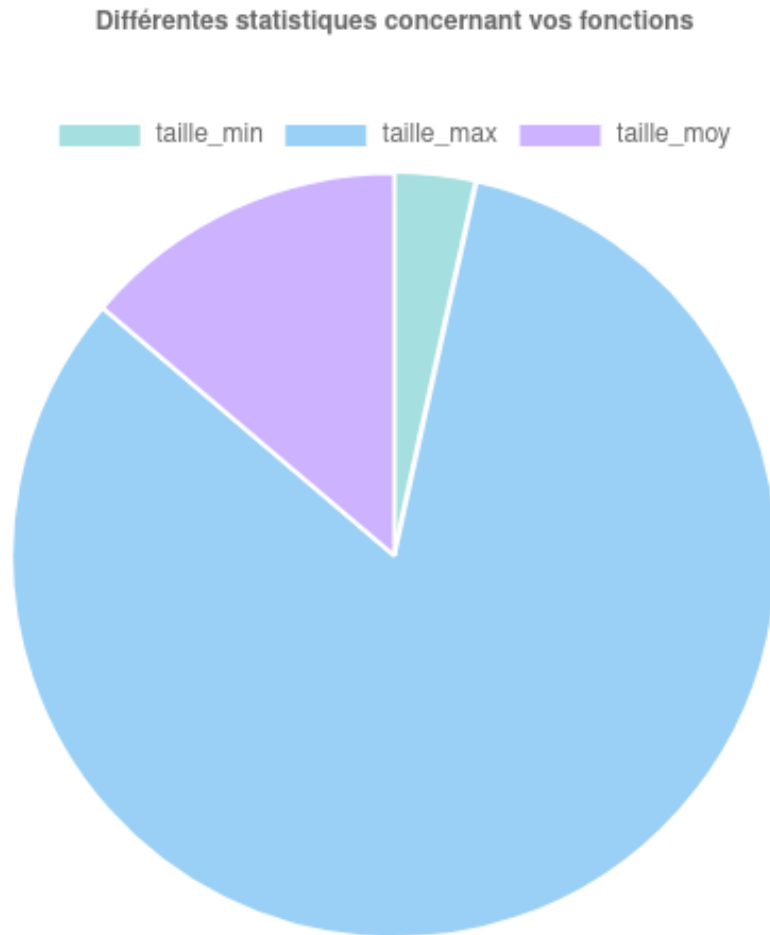


Figure 12: Visualisation des résultats 3

V. Problèmes rencontrés

Lors de ce projet nous avons fait face à plusieurs problèmes. Nous avons réussi à résoudre une bonne partie d'entre eux mais certains d'entre eux ont persistés.

1. La messagerie

Il y a eu plusieurs problèmes rencontrés dans la conception de la messagerie: -Les nouveaux messages s'envoient avec une requête AJAX mais lorsque l'utilisateur appuyait sur la touche "entre" le formulaire était envoyé et réactualisé la

page. Pour résoudre ce problème nous avons forcé l'action de touche enter pour qu'elle simule l'appui du bouton envoyer. -Comme dit précédemment lorsqu'on desire contacter un nouvel utilisateur il faut saisir son nom dans un input et des noms sont proposés en fonctions de ce que l'utilisateur saisit. Pour une raison inconnue cela ne marche pas. Pourtant si on saisie manuellement la datalist, la suggestion marche bien.

VI. Bilan du projet

(TBA)