

# RAPPORT DU PFA

ISBAINE Mohamed

LABRIJI Saad

Prof encadrant : Mme.Aziza benomar

Etablissement : INPT (Institut national des postes et télécommunications)



## *Table des matières :*

*Remerciement.*

*Introduction.*

### **I. Présentation du projet :**

#### **A. Contexte :**

1. *Problématique.*
2. *Objectif du projet.*
3. *Outils utilisés.*
4. *Quelques notions générales de programmation utilisées.*
5. *Gestion de Projet.*

#### **B. Explication théorique du fonctionnement de l'application :**

6. *Présentation.*
7. *La recherche Exclusive.*
8. *La recherche Approximative.*
9. *La recherche Minimale.*
10. *Tri par défaut des données.*

### **II. Conception :**

#### **A. Diagramme des classes.**

#### **B. Mode Graphique.**

11. *La classe Exclusive.*
12. *La classe Approximative.*
13. *La classe Minimale.*
14. *La classe Main.*
15. *Tri par défaut des données.*

### **III. Travaux effectués :**

#### **A. Tests et simulations du fonctionnement de l'application :**

#### **B. Conclusion**

#### **C. Bibliographie**

***Remerciement :***

Nous tenons à exprimer nos remerciements avec un grand plaisir et un grand respect à notre encadrant **Mme Benomar Aziza** pour ses conseils, sa disponibilité et ses encouragements qui nous ont permis de réaliser ce travail dans les meilleures conditions.

Nous exprimons de même notre gratitude envers tous ceux qui nous ont accordé leur soutien, tant par leur gentillesse que par leur dévouement.

A tous les enseignants qui nous ont aidés pendant notre première année à l'**INPT**.

A toute personne ayant contribué de près ou de loin à l'avancement de notre projet.

## Lexique :

**JavaFX :** est une collection de packages graphiques et multimédias qui permet aux développeurs de concevoir, créer, et tester des applications sur diverses plates-formes.

**Scene Builder :** est un outil de mise en page visuelle qui permet aux utilisateurs de concevoir rapidement des interfaces utilisateur d'application JavaFX, sans codage.

**JDK :** Le Java Development Kit (JDK) désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java.

**SDK :** (Software Development Kit) est une compilation d'outils permettant le développement d'applications destinées à un matériel/logiciel donné ou codées dans un langage de programmation spécifique.

**Fichier FXML :** FXML est un format de données textuelles, dérivé du format XML, qui permet de décrire une interface utilisateur pour des applications conçus avec JavaFX.

**Remarque :** l'intégralité du code source du projet, ainsi que les bibliothèques externes utilisées dans ce projet se trouvent [ICI](#).

## Introduction générale :

Ayant atteint la fin de la première année en filière **Data Engineer** au sein de l'**INPT**, un projet de fin d'année est demandé d'accomplir, Notre choix s'est rapporté à concevoir et réaliser un produit logiciel : il s'agit d'application desktop en **Java**.

Nous avons choisi les outils d'une manière cohérente avec notre philosophie de travail et en harmonie avec les connaissances acquises pendant cette première année.

Ainsi, la recherche et le tri puis le classement selon l'importance des données récoltées depuis les fichiers, s'avère des fois pénibles surtout lorsqu'il s'agit de les ouvrir et rechercher leurs contenus manuellement. Ce problème est plutôt courant chez les entreprises qui ont pour support de base de données des fichiers de type Txt, PDF, doc...

L'automatisation de la recherche à travers des algorithmes qui collectent les données et les trient selon des critères d'importance, par exemple la date de dernière modification, puis affiche les résultats sous une forme plus visible et plus parlante. Cette automatisation forme la solution idéale. De plus, on peut assurer la rapidité et la facilité de mise en œuvre grâce à une interface graphique simplifiée permettant d'adapter l'application au profil de l'entreprise.

L'application que nous développons permet de :

- ✓ Récupérer la recherche de l'utilisateur.
- ✓ Lire et récolter le contenu des fichiers (d'extension : Txt, PDF, doc et docx).
- ✓ Trier et classer les fichiers selon des critères (qu'on détaillera après).
- ✓ Afficher les fichiers de manière plus visible dans un tableau.
- ✓ Proposer un fichier d'historique qui contient le résultat de la recherche, ainsi que des données secondaires (nombre d'occurrence, lien vers le fichier... etc.).

Enfin, le présent projet intitulé **Dfinder** s'articule autour de trois grands points suivants

- Le premier point : proposer des méthodes de recherche pertinentes afin que le résultat de la recherche soit plus significatif adaptatif au besoin de l'utilisateur.
- Le deuxième point : proposer une interface graphique personnalisable (personnalisation du tri, du répertoire de sauvegarde, et bien plus...).
- Le troisième point : proposer la possibilité de sauvegarder le résultat de la recherche (contient plus de détails de la recherche) sous forme de fichier txt.

En effet, nous voulons que notre application soit extensible, évolutive (peut être mise à jour pour s'adapter aux nouvelles technologies afin qu'elle cible une zone de recherche plus large), simple tout en gardant son efficacité.

## **I. Présentation du projet :**

### **A. Contexte :**

#### **1. Problématique :**

Aujourd'hui, la méthode la plus courante pour effectuer une recherche dans un répertoire donné est l'utilisation du programme de recherche intégré avec Windows, or ce dernier n'est pas complet (car il n'effectue la recherche que sur le nom des fichiers, de plus, il est limité à une seule méthode de recherche, et il ne propose pas à l'utilisateur la possibilité de personnaliser le tri...etc.).

**Problématique principale :** la méthode existante ne permet pas de rechercher des informations se trouvant dans le fichier.

Pour remédier à ce problème et faciliter la recherche, il serait important de disposer d'une application avec interface graphique simple et personnalisable, qui permet d'effectuer des recherches au sein des fichiers.

#### **2. Objectif du projet :**

L'objectif de notre application **Dfinder** est de remédier aux problèmes cités ci-dessous, et également proposer des algorithmes de recherches différents pour que la recherche soit la plus pertinente possible et qui varie selon le besoin de l'utilisateur.

Son élaboration s'est appuyée, entre autres, sur les recommandations de notre professeur Mme Benomar Aziza.

L'application **Dfinder** est destinée à toute personne, et spécialement aux entreprises qui ont pour support de base de données des fichiers (de type Txt, PDF, doc et docx).

#### **3. Outils utilisés :**

L'application a été conçue et implémentée avec le langage **Java**, à l'aide de l'**IDE : JetBrains IntelliJ IDEA**, la version du **JDK** utilisé est **15**, l'interface graphique de l'application a été conçue avec **JavaFX** par l'intermédiaire de **Scene Builder**.

**JavaFX :** est une collection de packages graphiques et multimédias qui permet aux développeurs de concevoir, créer, et tester des applications sur diverses plates-formes.

**Scene Builder** : est un outil de mise en page visuelle qui permet aux utilisateurs de concevoir rapidement des interfaces utilisateur d'application JavaFX, sans codage.

L'application utilise plusieurs bibliothèques externes qui peuvent être divisées en deux parties principales : (nous détaillerons leur utilisation après)

- Jars chargés de lire le contenu des fichiers.
- Jars responsables de l'interface graphique (**JavaFX**)

#### 4. Quelques notions générales de programmation utilisées :

L'application utilise plusieurs notions générales de programmation, les principales sont : la **POO** qui subdivise l'application en parties et facilite son organisation et le choix des méthodes. L'**Héritage** qui permet d'éviter la répétition des méthodes entre classes et donc optimiser le code. Les **Structures de données** : (Arraylist, HashMap, TreeMap...etc.). Qui facilitent le stockage et la manipulation des données selon leur type. Le **tri** avec **comparable & comparator** qui permet de trier les données selon plusieurs critères (dans notre cas : score accumulé, dernière date de modification, taille du fichier).

#### 5. Présentation du cahier de charges :

##### 1<sup>ère</sup> Partie : Spécifications du browser

##### 1- Choix d'un nom, d'un logo du browser à développer.

##### 2- Etude sur l'accès des fichiers de données en java :

On se propose d'étudier l'accès aux fichiers textes en java en précisant les principales classes de l'API (package java.io) qui permettent de représenter des fichiers textes (InputStreamReader, BufferedReader, ...) ainsi que les méthodes d'accès en lecture à ces fichiers.

Il sera intéressant d'illustrer cette étude par des exemples simples

##### 3- Proposition d'une méthode de recherche et d'affichage des résultats :

Il s'agit de proposer une méthode simple, pertinente et personnalisée de recherche d'informations dans un fichier texte.

**Par exemple** : Soit à rechercher dans un emplacement donné de l'ordinateur local (un disque dur spécifique, un dossier, ...) les fichiers contenant les n mots d'une phrase : mot<sub>1</sub> mot<sub>2</sub> mot<sub>3</sub>...mot<sub>n</sub>. (On suppose que les mots sont séparés par des espaces.) On peut calculer par exemple pour chaque fichier se trouvant dans l'emplacement de recherche,

la somme des occurrences de  $\text{mot}_1$ ,  $\text{mot}_2$ ,  $\text{mot}_3$ , ...,  $\text{mot}_n$  dans le fichier. Les noms de fichiers et leurs chemins d'accès seront affichés par ordre décroissant du nombre d'occurrences.

On peut en plus donner des coefficients en fonction de l'ordre des mots dans la phrase (Par exemple  $\text{mot}_1$  de coefficient  $n$ , ...,  $\text{mot}_n$  de coefficient 1).

D'autres paramètres peuvent être pris en compte (par exemple on peut avoir plusieurs options de recherche (rechercher exclusivement les fichiers contenant la phrase avec les mêmes mots dans le même ordre, ou rechercher les fichiers contenant la phrase avec tous ses mots mais pas nécessairement dans le même ordre, ou rechercher les fichiers contenant au minimum un mot de la phrase (ou le 1er mot de la phrase), ou rechercher les fichiers contenant les mots de la phrase et édités à partir d'une certaine date, ...etc.)

### **2<sup>ème</sup> Partie : Conception et développement (sans interface graphique).**

Une fois la méthode de recherche d'informations et d'affichage des résultats déterminée, il faut concevoir puis développer une application java qui permet à l'utilisateur de choisir un emplacement (sur l'ordinateur), saisir une phrase (sous forme de mots séparés par des espaces) puis définir l'option de recherche souhaitée et l'application doit ainsi afficher dans un ordre spécifique et pertinent, les noms de fichiers contenant tous où une partie des mots de cette phrase en fonction de l'option choisie par l'utilisateur.

### **3<sup>ème</sup> Partie : Interface graphique interactive.**

Après avoir testé et validé l'application, il s'agit de construire une interface graphique ergonomique et pratique permettant d'illustrer le browser local développé.



## **1. Gestion de Projet :**

La gestion de projet, est de bien conduire, coordonner et harmoniser les diverses tâches réalisées dans le cadre du projet. Ceci permet d'augmenter la rentabilité et garder l'esprit d'équipe chez les membres.

Dans notre cas, le projet a été divisé en plusieurs tâches, et chacun de nous s'occupait d'une partie. Après la réalisation des tâches affectées, une réunion est faite pour discuter les difficultés rencontrées et essayer de les résoudre ensemble. Ce qui a permis de réaliser un travail collectif à cent pour cent.

Le diagramme de Gantt est un outil efficace pour représenter visuellement l'état d'avancement des différentes activités (tâches) qui constituent un projet. La colonne de gauche du diagramme énumère toutes les tâches à effectuer, tandis que la ligne d'en-tête représente les unités de temps les plus adaptées au projet (jours, semaines, mois etc.). Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin. Ce diagramme permet donc de visualiser d'un seul coup d'œil :

- Les différentes tâches à envisager
- La date de début et la date de fin de chaque tâche
- La durée escomptée de chaque tâche
- Le chevauchement éventuel des tâches, et la durée de ce chevauchement
- La date de début et la date de fin du projet dans son ensemble

Pour cela on donnera le diagramme de Gantt de notre projet dans la figure suivante :

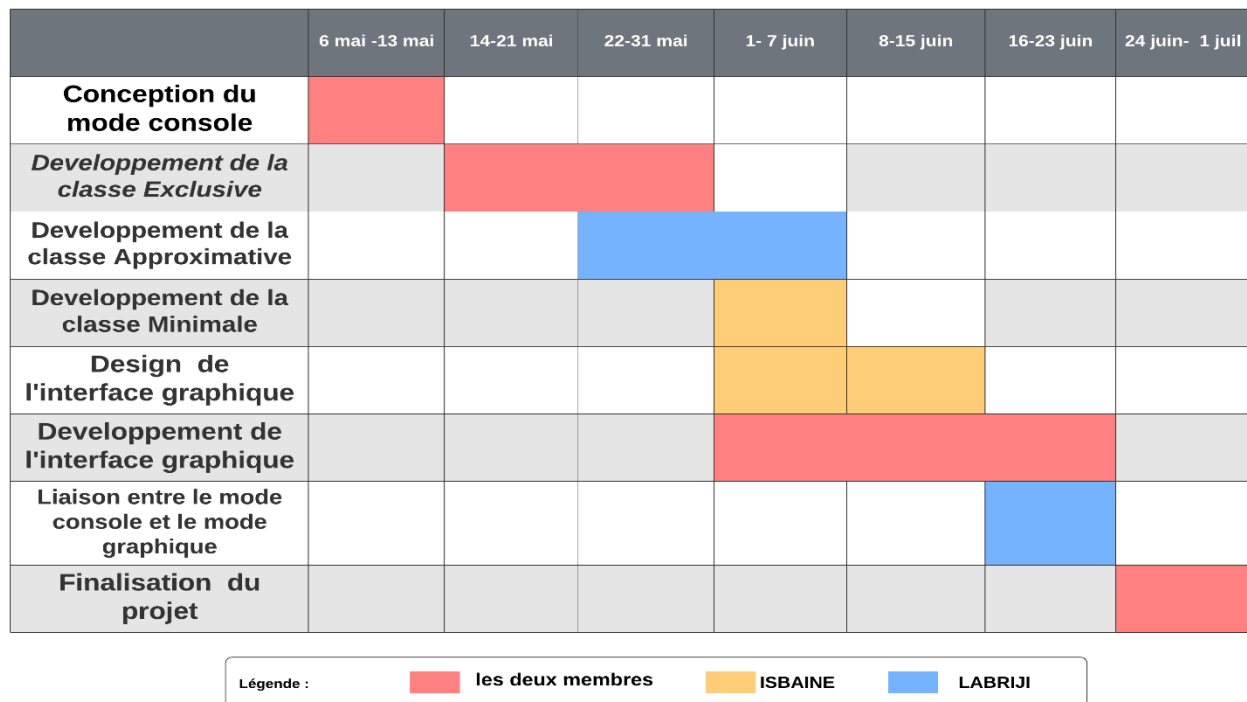


Figure 1 : Diagramme de Gantt de l'application **Dfinder**.

## I. Explication théorique du fonctionnement de l'application :

### Présentation :

Comme mentionné dans le contexte, l'application se base principalement sur 3 méthodes de recherches : la recherche Exclusive, la recherche Approximative et la recherche Minimale.

**La recherche Exclusive** : consiste à trouver les fichiers qui contiennent la phrase tapée par l'utilisateur telle quelle est (c'est à dire **tous** les mots **dans le même ordre**).

- Les fichiers supportés sont ceux d'extension : Txt, PDF, doc et docx
- La recherche ne prend pas en compte les lettres majuscules et minuscules.
- Les fichiers (de toute extension), dont le nom contient la phrase donnée telle quelle, seront affichés en premier.
- La recherche est effectuée dans l'arborescence du répertoire donné par l'utilisateur.
- Lorsque la recherche est terminée, un fichier (de la forme : `output_Exclusive[MM-dd-yyyy HH:mm:ss].txt` ) est créé par défaut dans un répertoire choisi par l'utilisateur.

**La recherche Approximative** : consiste à trouver l'ensemble des fichiers qui contiennent **tous** les mots de la phrase tapée par l'utilisateur mais **pas forcément dans le même ordre de saisie**.

- Les fichiers supportés sont ceux d'extension : Txt, PDF, doc et docx
- La recherche ne prend pas en compte les lettres majuscules et minuscules.
- Les fichiers (de toute extension) dont le nom contient la phrase donnée telle quelle seront affichés en premier.
- Les fichiers dont le contenu contient les mots recherchés sont triés et affichés selon quelques critères (qu'on expliquera dans la partie suivante).
- La recherche est effectuée dans l'arborescence du répertoire donné.
- Lorsque la recherche est terminée, un fichier (de la forme : **output\_Approximative[MM-dd-yyyy HH:mm:ss].txt** ) est créé par défaut dans un répertoire choisi par l'utilisateur.

**La recherche Minimale** : consiste à trouver l'ensemble des fichiers contenant, **au moins**, un mot de la phrase tapée.

- Les fichiers supportés sont ceux d'extension : Txt, PDF, doc et docx
- La recherche ne prend pas en compte les lettres majuscules et minuscules.
- Les fichiers (de toute extension) dont le nom contient la phrase donnée telle quelle seront affichés en premier.
- Les fichiers dont le contenu contient les mots recherchés sont triés et affichés selon quelques critères (qu'on expliquera dans la partie suivante).
- La recherche est effectuée dans l'arborescence du répertoire donné.
- Lorsque la recherche est terminée, un fichier (de la forme : **output\_Minimale[MM-dd-yyyy HH:mm:ss].txt** ) est créé par défaut dans un répertoire choisi par l'utilisateur.

**Remarque 1** : pour les trois méthodes de recherche, le nom du fichier est vérifié d'abord. En cas de correspondance avec la recherche, le fichier est gardé et son contenu n'est pas lu.

**Remarque 2** : l'utilisateur peut personnaliser son tri à l'affichage (tri selon la date de modification, tri selon la taille des fichiers ou bien le tri par ordre alphabétique des noms des fichiers). Par défaut les fichiers sont triés selon quelques critères qu'on détaillera dans la partie suivante.

## Comment sont triés les fichiers par défaut ?

### Pour la recherche Exclusive :

Les fichiers dont les noms correspondent à la recherche sont groupés et affichés en premier.

Le reste des fichiers (dont les noms ne correspondent pas à la recherche) seront classés par nombre d'occurrence de la phrase saisie par l'utilisateur. (C'est-à-dire le nombre de fois qu'on a trouvé la phrase saisie telle quelle est), puis par date de modification la plus récente, puis par taille la plus petite.

**Remarque :** l'utilisateur doit être absolument prudent lors de l'utilisation de la recherche exclusive : cette méthode recherche des fichiers qui contiennent exactement la séquence fournie par l'utilisateur. Donc un espace supplémentaire à gauche ou à droite de la séquence peut donner un résultat différent.

### Pour la recherche Approximative et Minimale :

Les fichiers dont les noms correspondent à la recherche sont groupés et affichés en premier. Ces fichiers sont tout d'abord triés et classés selon les critères suivants :

- ✓ **1<sup>er</sup> critère :** Le nombre d'occurrence pondérées des mots de la phrase ; dans chaque fichier trouvé, on calcule la somme des nombres d'occurrences des mots de la phrase selon deux méthodes au choix de l'utilisateur comme suit :

$$somme = \sum_{k=1}^n \text{nombreOccurrence}(\text{motk}) \quad : \text{METH 1}$$

$$somme = \sum_{k=1}^n \text{nombreOccurrence}(\text{motk}) * (n - k + 1) \quad : \text{METH 2}$$

*Avec motk : est le k – ième mot de la phrase.*

*et n : la taille de la phrase.*

**Pour la première méthode ( METH 1 ) :** L'ordre du mot dans la phrase n'est pas important ; la formule de calcul de la somme des occurrences suppose que tous les mots ont tous la même importance.

**Pour la deuxième méthode ( METH 2 ) :** L'ordre du mot dans la phrase est important ; la formule de calcul de la somme des occurrences suppose que le 1er mot de la phrase est le plus important, ... et le dernier est le mot est le moins important.

Et en se basant sur cette somme, on trie les fichiers trouvés par ordre décroissant.

- ✓ **2<sup>ème</sup> critère** : La date de dernière modification ; Lorsque on a des fichiers qui ont la même somme de nombre d'occurrence, on fait appel à la dernière date de modification (le fichier modifié le plus récemment sera prioritaire).
- ✓ **3<sup>ème</sup> critère** : La taille du fichier ; Si les deux premiers critères ne sont pas suffisants pour trier les résultats de la recherche. On fait appel à la taille du fichier. Le fichier le moins lourd sera priorisé.

Ces trois critères vont permettre de classer d'une manière pertinente les résultats obtenus.

Dès que les fichiers dont les noms correspondent à la recherche sont groupés et affichés en premier, le reste des fichiers (dont les noms ne correspondent pas à la recherche) sont lus et triés selon les mêmes critères ci-dessus, et puis affichés.

## Résumé général du fonctionnement de l'application ?

Après l'explication de ce que chacune des méthodes fait en théorie, cette partie a pour but d'introduire le code en expliquant brièvement le rôle de chaque classe et fichier de l'application. (Un diagramme de classe sera présenté ultérieurement pour détailler les choses encore plus).

**Remarque** : l'intégralité du code source du projet, ainsi que les bibliothèques externes utilisées dans ce projet se trouvent [ICI](#).

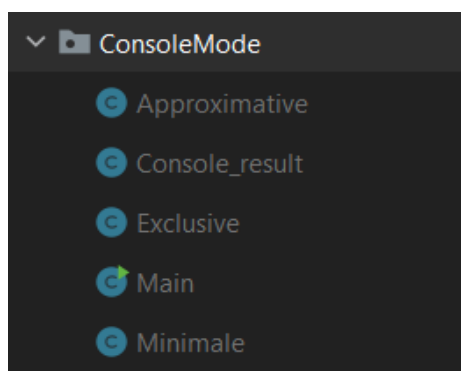


Figure 2 : contenu du package *ConsoleMode*.

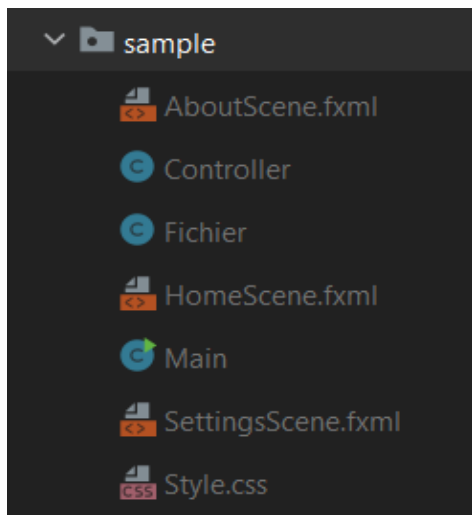
Le package **ConsoleMode** présent dans le projet **Dfinder** contient six classes : **Approximative.java**, **Console\_result.java**, **Exclusive.java**, **Main.java**, **Minimale.java** et **TreePrintStream.java**

**Exclusive.java**, **Approximative.java** et **Minimale.java** : 'classes de recherche' implémentent respectivement les méthodes de la recherche Exclusive, Approximative, et Minimale. Ces classes se contentent (à partir du lien du répertoire fourni par l'utilisateur, ainsi que sa recherche) de rechercher, lire et trier les fichiers. Après ce traitement, le résultat est affiché en console.

**TreePrintStream.java** : sauvegarde la sortie standard de l'application (la console) dans un fichier txt.

**Console\_result.java** : cette classe représente le pont qui lie le mode console avec l'interface graphique. Elle se contente de récupérer les choix de l'utilisateur par l'intermédiaire de l'interface graphique pour fournir les données nécessaires au fonctionnement des '**classes de recherche**', Après le traitement des données, elle récupère le résultat des '**classes de recherche**' et le fournit à la '**classe graphique**'.

**Main.java** : contient la méthode main qui lance le programme en mode console.



Le package **sample** présent dans le projet **Dfinder** contient trois classes : **Controller.java**, **Fichier.java** et **Main.java**

Et contient trois fichiers **fxml** : **HomeScene.fxml**, **AboutScene.fxml** et **SettingsScene.fxml**.

Et contient un fichier **css** : **Style.css**

Figure 3 : contenu du package *sample*.

**HomeScene.fxml**, **AboutScene.fxml** et **SettingsScene.fxml** : permettent respectivement de décrire les interfaces utilisateur (scènes) : **Home**, **About** et **settings**

**Style.css** : s'occupe du côté esthétique de l'application (animations et mise en forme des contenus des scènes).

**Main.java** : contient la méthode main qui lance le programme en mode interface graphique.

**Fichier.java** : adapte le résultat du mode console au type de donné compatible à l'affichage dans l'interface graphique.

**Controller.java** : '**classe graphique**' gère directement les scènes et leurs contenus. Elle s'occupe du traitement des demandes entrantes/sortantes entre le mode console et l'interface graphique.

## II. Conception :

Après avoir présenté le projet, il est temps de parler plus du côté technique. Dans cette section, on va présenter le diagramme des classes, qui va nous aider à mieux comprendre le fonctionnement de **Dfinder**.

## A. Diagramme de classes :

Un diagramme de classes fournit une vue globale d'un système en présentant ses classes, interfaces et collaborations, et les relations entre elles. Les diagrammes de classes sont statiques c'est-à-dire qu'ils affichent ce qui interagit mais pas ce qui se passe pendant l'interaction.

En notation UML, une classe est représentée sous la forme d'un rectangle divisé en plusieurs parties : le nom de la classe, les attributs (champs), les opérations (méthodes)... (voir figure ci-contre).

Dans la Modélisation, le rectangle de la classe est divisé en compartiments distincts pour les champs, les classes internes, les propriétés et les opérations. Les relations entre les classes sur les diagrammes de classes sont de type :

- Généralisation (implémentation) ;
- Dépendance ;
- Association : (association simple, agrégation, composition).
- Héritage.

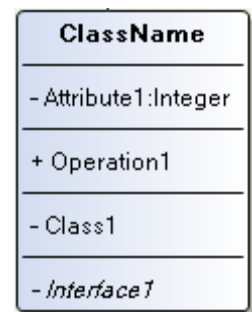


Figure 4 : Modélisation du rectangle de classe.

Dans notre cas, on a tracé le diagramme de classe pour le package ConsoleMode, qui contient l'ensemble des classes motrices de la recherche.

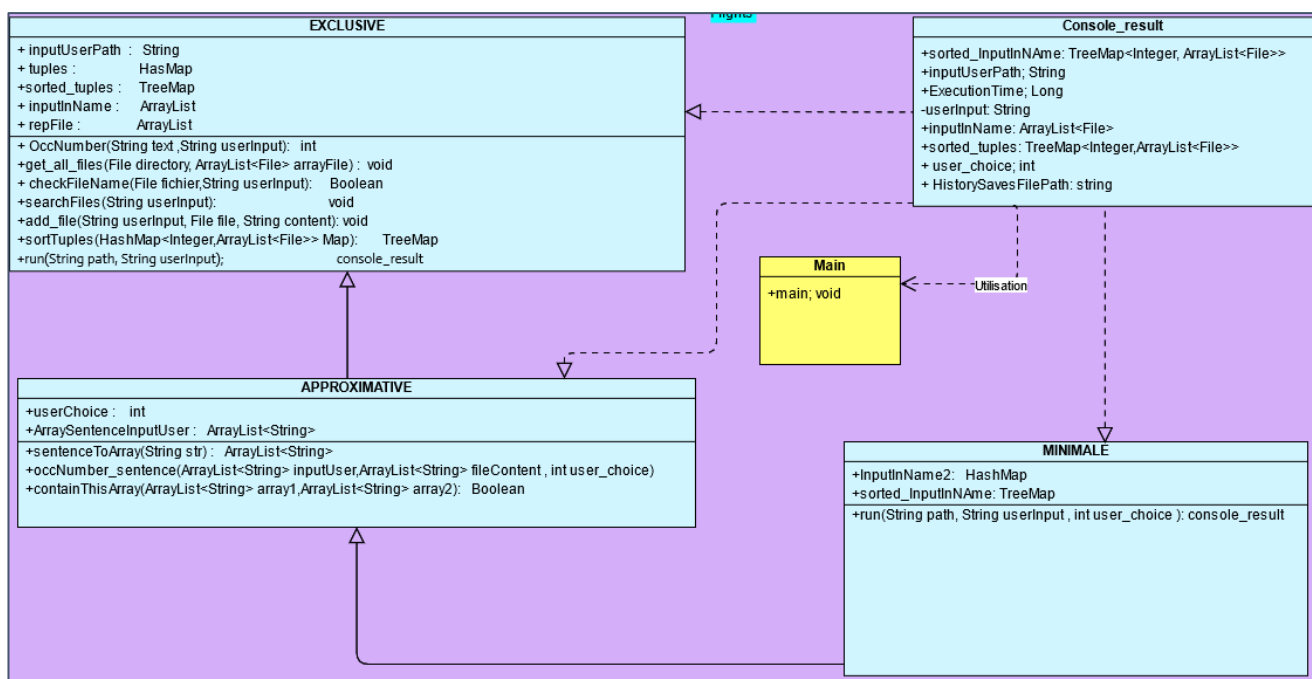


Figure 5 : Diagramme de classe du package **ConsoleMode**.

## 1. La classe Exclusive :

C'est la classe mère des deux classes Approximative et Minimale, puisqu'elles partagent plusieurs attributs et méthodes.

Cette classe comporte plusieurs attributs :

- ✓ **InputUserPath** : C'est une variable de type String qui est destinée à contenir le chemin (absolue ou bien relatif) du répertoire dans lequel l'utilisateur veut effectuer la recherche.
- ✓ **tuples** : C'est un attribut de type HashMap qui sert à stocker des objets de la classe File, qui modélise un répertoire ou bien un fichier, qui correspondent à la recherche. Cet attribut est sous forme de tuple, dont la valeur contient un entier qui représente un certain score (nombre d'occurrences). Et en valeur, on trouve un ArrayList de type File qui liste l'ensemble des fichiers qui répondent à la recherche et ayant la clé comme score.
- ✓ **Sorted\_tuples** : c'est un attribut de type TreeMap, qui contient les mêmes données de l'attribut '**tuples**', sauf qu'elles sont triées par nombre d'occurrence, puis la dernière date de modification et enfin la taille. Cet attribut est rempli à l'aide des méthodes qu'on détaillera dans les lignes qui suivent.
- ✓ **InputInName** : C'est un ArrayList de type File, qui va contenir l'ensemble des fichiers dont le nom correspond à la phrase recherchée.
- ✓ **RepFile** : ArrayList destiné à contenir l'ensemble des fichiers du répertoire de recherche.

Pour les méthodes :

- ✓ **OccNumber(String text, String userInput)** : cette méthode permet de calculer le nombre d'occurrence d'une chaîne de caractère 'userInput' dans la chaîne 'text'. Et retourne ce nombre.
- ✓ **get\_all\_files(File directory, ArrayList<File> arrayFile)** : permet de parcourir l'arborescence du répertoire donné en premier paramètre et en extraire tous les fichiers, et les mettre dans l'ArrayList 'arrayFile' donné en deuxième paramètre. Cette méthode sert à remplir l'attribut 'RepFile'.
- ✓ **checkFileName(File fichier, String userInput)** : permet de vérifier si la phrase recherchée (2ème paramètre) est contenue dans le titre du fichier (1er paramètre) insensiblement à la case.



- ✓ **searchFiles(String userInput)** : permet de chercher tous les fichiers dont le nom contient la phrase recherchée et les ajouter dans l'ArrayList 'InputName'. Ainsi, tous les fichiers de type Txt, doc, docx et PDF qui contiennent la phrase recherchée sont ajoutés au HashMap 'tuples'.
- ✓ **add\_file(String userInput, File file, String content)** : permet d'ajouter un fichier qui contient la phrase recherchée au tuple dont la clé est le nombre d'occurrence. Si la clé existe déjà, elle ajoute le fichier file au ArrayList correspondant à cette clé, sinon elle crée un nouveau couple (clé, valeur) et l'ajoute à 'tuples'.
- ✓ **sortTuples(HashMap<Integer, ArrayList<File>> Map)** : permet de trier le HashMap 'tuples' selon la clé (nombre d'occurrence), puis selon la date de dernière modification et ensuite la taille. Cette méthode retourne un TreeMap qui sera affecté à l'attribut 'sorted\_tuples' dont on a parlé.
- ✓ **run(String path, String userInput)** : Cette méthode permet d'instancier la classe Exclusive en appelant son constructeur avec le paramètre chemin du répertoire de recherche. Après elle sauvegarde la sortie dans un objet de type console\_result qui va permettre de communiquer ces résultats à l'interface graphique. Et finalement elle permet de créer le fichier de sortie aussi en appelant la classe TeePrintStream.

## 2. La classe Approximative :

Cette classe a beaucoup de points de ressemblance avec Exclusive, c'est pour cela qu'on a utilisé la notion d'héritage pour éviter la répétition et pour rendre le code plus souple et moins complexe.

Puisque cette classe permet de chercher tous les mots saisis mais pas forcément dans l'ordre, donc il est judicieux de faire quelques modifications sur quelques méthodes et ajouter certains attributs :

- ✓ **user\_choice** : cet attribut est de type entier, qui peut prendre les valeurs 1 ou bien 2. La valeur 1 désigne la méthode de recherche choisie par l'utilisateur est METH1, et la valeur 2 désigne la méthode METH2. Ces méthodes sont déjà détaillées avant.
- ✓ **ArraySentenceInputuser** : C'est un ArrayList de type String, qui est destiné à contenir les mots, de la phrase recherchée, séparés. Ceci va nous aider pour mieux rechercher dans chaque fichier et calculer les scores.

### Concernant les méthodes supplémentaires de la classe Approximative :

- ✓ **sentenceToArray(String str)** : cette méthode permet de découper une chaîne de caractère passée en paramètre en mots, et mettre ces derniers dans un ArrayList qui sera retourné. Cette méthode permet de remplir l'attribut 'ArraySentenceInputuser'.
- ✓ **occNumber\_sentence(ArrayList<String>inputUser, ArrayList<String>fileContent, int user\_choice)** : c'est une redéfinition de la méthode OccNumber déjà définie dans la classe Exclusive. Retourne la somme totale du nombre d'occurrence de chaque mot de \*inputUser\* dans \*fileContent\* en prenant en compte l'ordre ou pas selon la méthode choisie.
- ✓ **containThisArray(ArrayList<String>array1, ArrayList<String> array2)** : La nécessité de tester si un ArrayList est contenu dans un autre, nous a amené à définir cette méthode, qui retourne true si tous les éléments de array1 sont aussi dans array2. Et false sinon.

### 3. La classe Minimale :

Contrairement aux autres classes, la classe minimale permet de chercher au moins un mot de la phrase cherchée soit dans le nom du fichier ou bien dans son contenu. C'est pour cela qu'elle hérite de la classe Approximative, avec quelques attributs et méthodes de plus.

Les attributs qui sont spécifiques à cette classe sont :

- ✓ **InputInName2** : cet attribut est de type HashMap qui sert à contenir des tuples (score, ArrayList de fichiers), exactement comme l'attribut tuples.

Il est judicieux de se poser la question suivante : **pourquoi ne pas utiliser l'attribut tuples au lieu de créer un nouvel attribut ?**

Comme on a déjà déclaré, on veut afficher les éléments dont le titre correspond à la recherche en premier, et donc on ne peut pas utiliser tuples **car après le tri, ces fichiers vont se dissoudre et on ne pourra plus les distinguer.**

- ✓ **Sorted\_InputInName** : Contient les éléments de InputInName2 triés selon le score, dernière date de modification et la taille.

La classe Minimale possède aussi des méthodes spécifiques :

- ✓ **Run(String path, String userInput, int user\_choice)** : c'est une redéfinition de la méthode run de exclusive qui va prendre en considération les caractéristiques de la classe minimale.

## 4. La classe **Console\_result** :

Après avoir développé l'interface graphique, il était nécessaire de créer une liaison efficace entre le non graphique et le graphique. C'est pour cela qu'on a développé cette classe qui comporte plusieurs attributs :

- ✓ **Sorted\_InputInName** : Cet attribut renferme le contenu de 'sorted\_inputInName' de la classe Minimale en particulier, après instantiation.
- ✓ **InputUserPath** : chaîne de caractère pour contenir le chemin du répertoire de recherché.
- ✓ **userInput** : chaîne de caractère qui sert à contenir la phrase recherché.
- ✓ **ExecutionTime** : de type long qui sert à stocker la durée totale, en Milliseconde, du traitement pour donner à l'utilisateur une idée sur les performances de l'application.
- ✓ **InputInName** : ArrayList qui va héberger le contenu de l'attribut 'InputInName' des classes Exclusive et Approximative en particulier.
- ✓ **Sorted\_tuples** : de même pour 'sorted\_tuples' pour les trois classes.
- ✓ **user\_choice** : pour stocker le contenu de 'user\_choice'.
- ✓ **HistorySavesFilePath** : Chaîne de caractère qui sert à mémoriser l'emplacement choisi par l'utilisateur pour sauvegarder les fichiers de sortie de chaque recherche effectuée.

## 5. La classe **Main** :

Cette classe permet d'exécuter le programme en mode console, et donner la main à l'utilisateur pour saisir les données nécessaires. Elle appelle la méthode run de chaque classe et dépose ses résultats dans un objet console\_result.

## B. Mode Graphique :

Dans un monde de concurrence, il faut donner à l'utilisateur une meilleure expérience d'utilisation de l'application. C'est exactement ce qu'on a essayé de faire, en développant une interface graphique qui est facile à utiliser et efficace.

### 1. Maquettes graphiques :

Notre interface graphique s'étale sur trois scènes : Home, Settings et About.

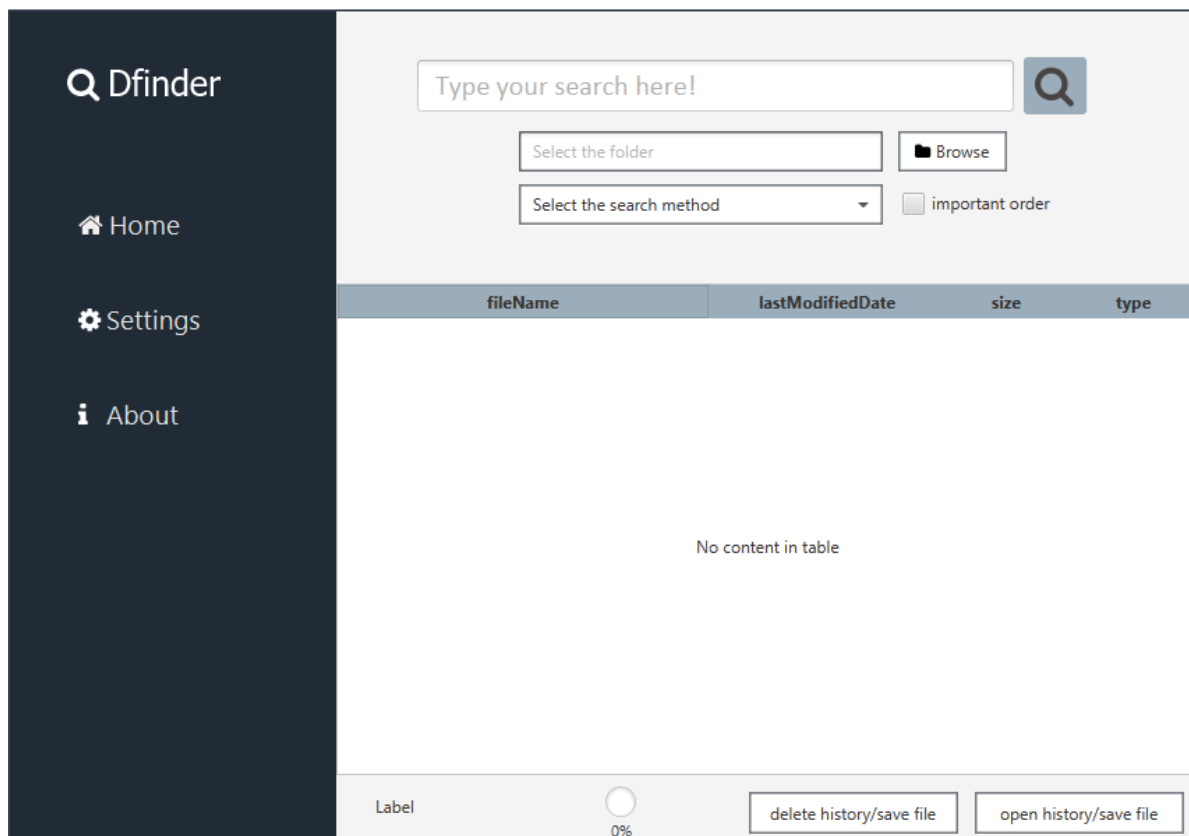


Figure 6 : la scène Home de l'application *Dfinder*.

Le design de page d'accueil est très simple et facile à utiliser. Il contient des champs de saisie pour taper la phrase à rechercher et un bouton pour valider la recherche. Ainsi qu'un bouton pour parcourir l'arborescence du système et sélectionner le dossier de recherche. Un tableau de quatre colonnes pour afficher les résultats de la recherche. Ensuite, des boutons pour supprimer l'historique de recherche ou bien ouvrir son emplacement. Et enfin, la barre de menu qui permet de changer la scène vers settings et about.

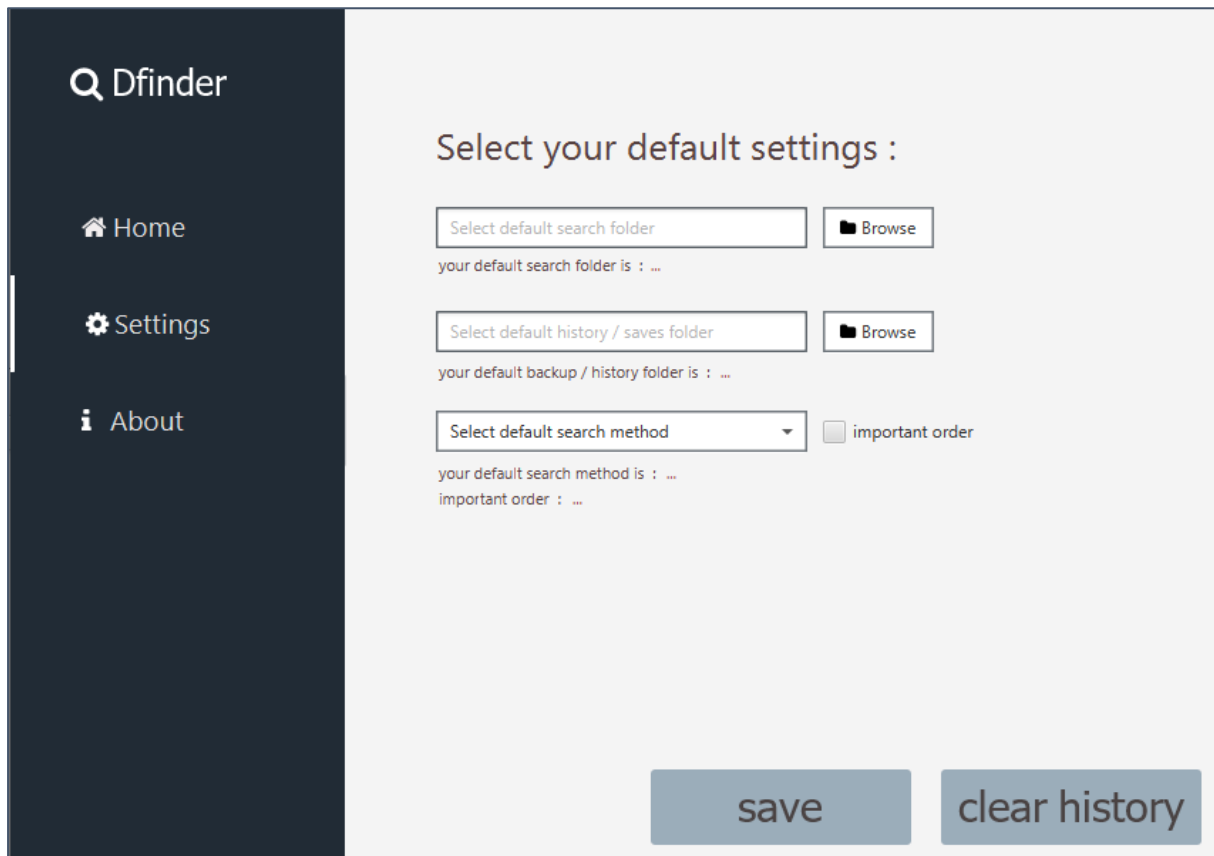


Figure 7 : la scène Settings de l'application **Dfinder**.

Cette interface donne à l'utilisateur la main pour sélectionner un répertoire de recherche par défaut, un dossier de sauvegarde par default et une méthode de recherche favorite qui sera sélectionnée automatiquement à chaque lancement de l'application.

De plus, un bouton pour supprimer l'historique et un autre pour sauvegarder les modifications faites.

L'interface About (**Figure 7**) donne à l'utilisateur des explications sur les méthodes (Approximative, Exclusive et Minimale).

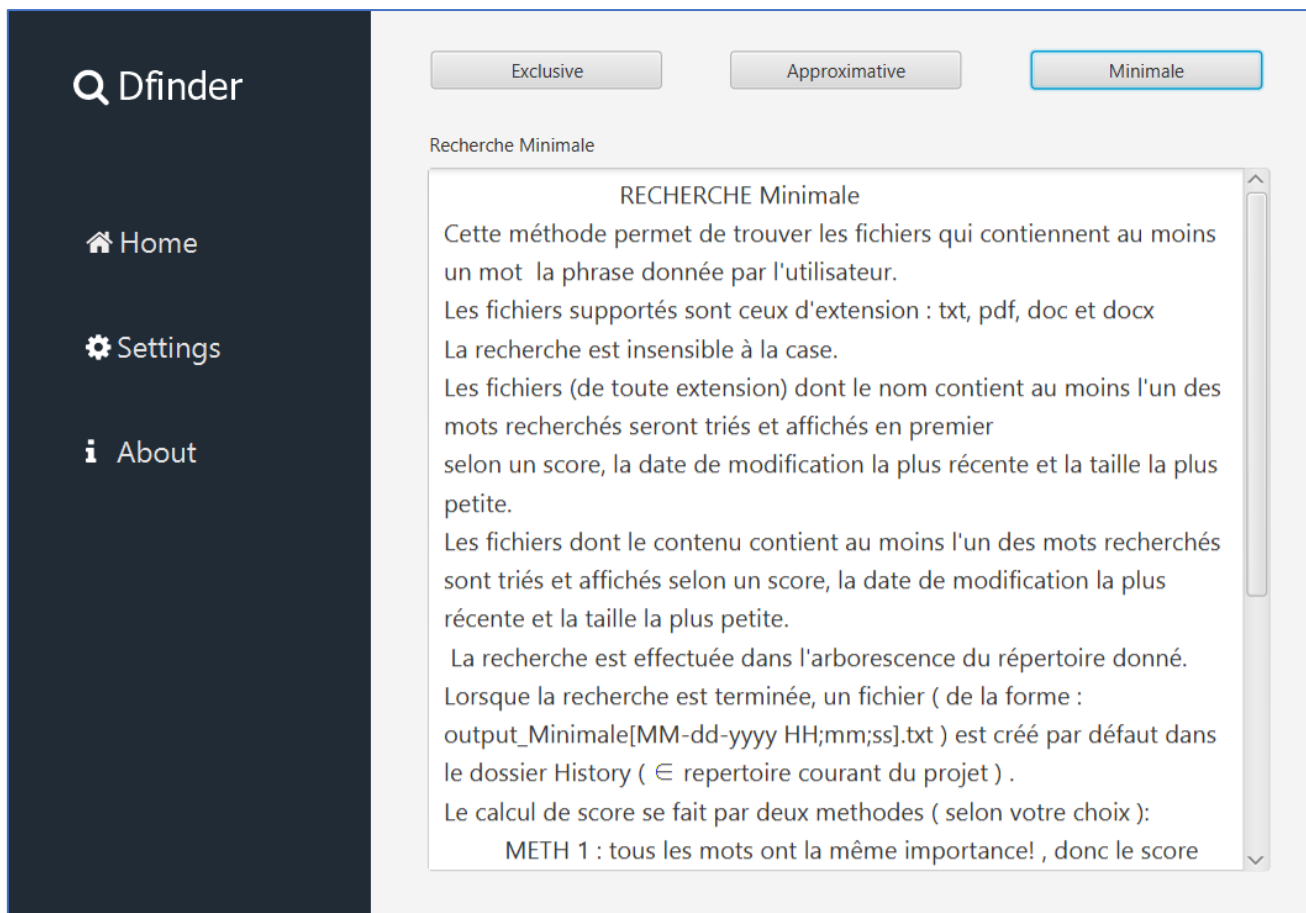


Figure 7 : la scène About de l'application **Dfinder**.

## 2. Choix du nom et du logo :

Le nom de l'application est '**Dfinder**', ce nom est composé de deux mots : Data et Finder :

- Data : qui est la traduction de 'données' en anglais.
- Finder : Qui veut dire chercheur.

Et donc le nom signifie qu'on est devant une application permettant la recherche de données. Ce qui rend le nom très significatif.

Concernant le logo, on a choisi d'être simple est droit au but.



Figure 8 : logo de l'application **Dfinder**.

Le logo est composé d'un papier écrit qui modélise un fichier et une loupe qui modélise la recherche. Ce qui représente bien la fonction principale et originale de notre application, qui est la recherche au sein des fichiers.

La couleur choisie correspond largement au thème de l'application.

## Tests et simulations du fonctionnement de l'application :

**Test 1 :** teste de la recherche réursive :

**Situation :** on va tester si l'application arrive à rechercher dans les dossiers, pour se faire on a créé deux fichiers dans des dossiers différents :

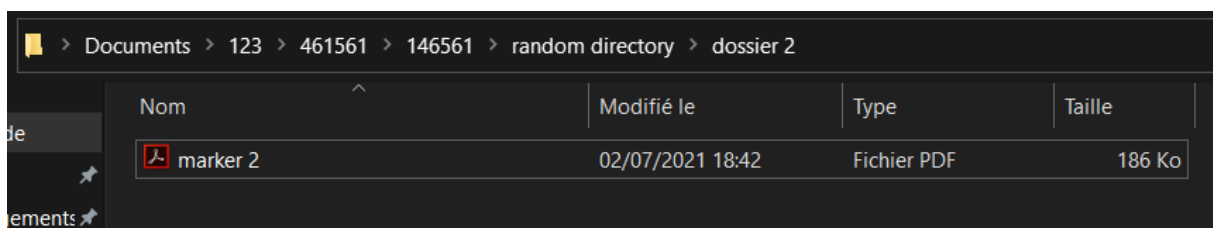


Figure 8 : Screenshot qui montre l'emplacement du fichier *marker 2.pdf*

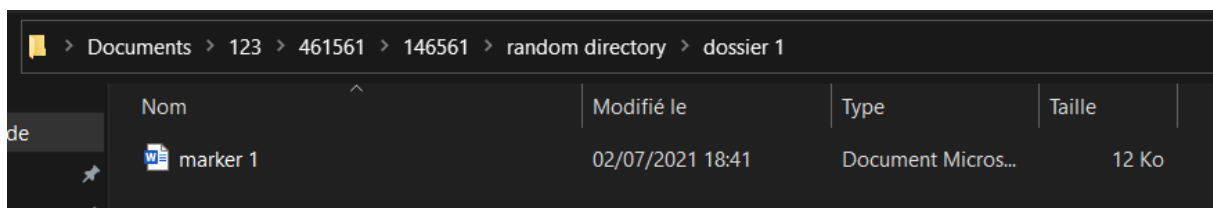


Figure 9 : Screenshot qui montre l'emplacement du fichier *marker 1.docx*

Les deux fichiers : **marker 1.pdf** et **marker 2.pdf** contiennent la phrase suivante :

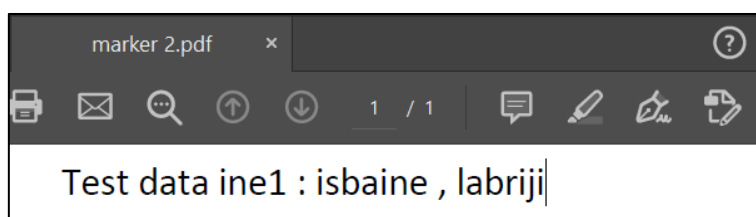


Figure10 : contenu des fichiers *marker 1.docx* et *marker 2.pdf*

L'arborescence du dossier **desktop** (dossier de recherche choisi) est la suivante :

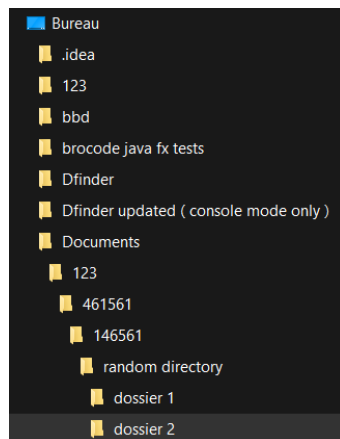


Figure 8 : arborescence qui montre l'emplacement des dossiers : **dossier 1** et **dossier 2**

## Résultat :

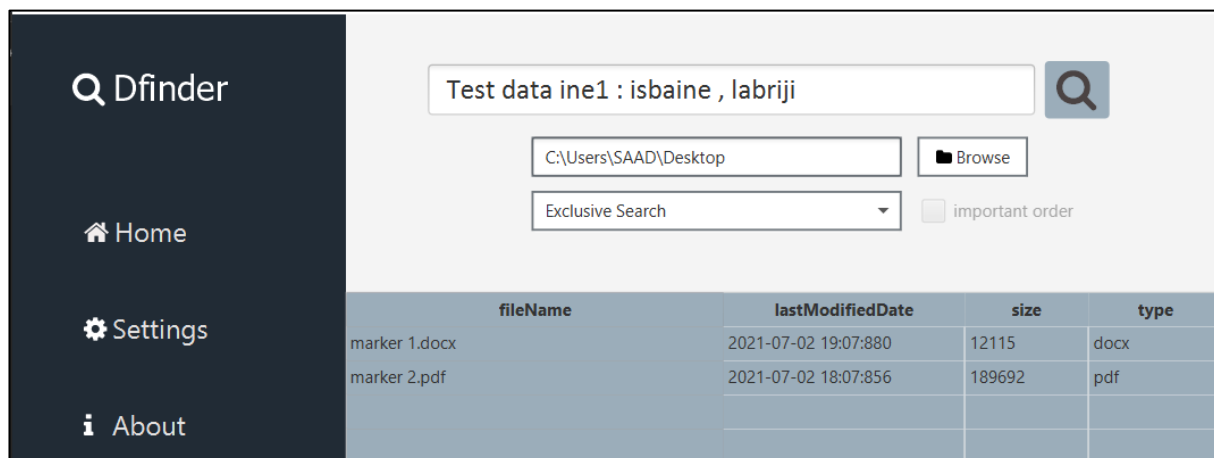


Figure 9 : résultat du test 1 (affiché dans l'interface graphique de l'application).

Lorsqu'on clique sur le bouton, [open history/save file](#) on obtient : (voir page suivante)



```

C:\Users\SAAD\Desktop\JavaFx Dfinder App\history-saves > output_Exclusive[07-02-2021 19:24:26].txt
1  ===== Récapitulatif =====
2  [SYSTEM] : la methode de recherche choisie : Recherche Exclusive
3  : le chemin du répertoire choisi : C:\Users\SAAD\Desktop
4  : la phrase à rechercher : Test data inel : isbaine , labriji
5  =====
6
7  [SYSTEM] : Aucun nom de fichier contient la phrase donnée telle quelle !
8  =====
9
10 [SYSTEM] : Voici les fichiers qui contiennent la phrase donnée telle quelle :
11 : ( les fichiers sont triés par nombre d'occurrence de la phrase donnée,
12 :   date de modification la plus récente la taille la plus petite. )
13 : ( Total = 2 )
14
15 [Occurence = 1] :
16 [file N°1] :   file name : marker 1.docx
17               last modification date : 02/07/2021 19:06:17
18               file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 1\marker 1.docx
19               file size : 12115 octets
20
21 [file N°2] :   file name : marker 2.pdf
22               last modification date : 02/07/2021 18:42:02
23               file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\marker 2.pdf
24               file size : 189692 octets
25
26 =====
27 [SYSTEM] * Temps d'exécution : 73824 Millisecondes.
28 [SYSTEM] * fin ! ( le fichier d'historique de recherche output_Exclusive[07-02-2021 19:24;26] .txt a été enregistré avec succès )
29 =====
30
31
32
33
34
35
36
37

```

Figure 10 : résultat du test 1 (affiché dans le fichier History/Save).

**Conclusion :** le teste montre bien que l'application arrive à rechercher de façon récursive dans les dossiers : les fichiers sont retrouvés. *Figure 11 et Figure 12*

**Remarque :** on a utilisé la recherche Exclusive car : c'est la méthode la plus appropriée puisque nous recherchons une phrase telle quelle. *Figure 11*

**Teste 2 :** teste de la recherche Exclusive :

**Situation :** on va tester si l'application arrive à rechercher et trier (selon les critères de tri par défaut vue dans la partie théorique) avec la recherche Exclusive, pour se faire : on a créé quelques fichiers, puis on a disperser un peu partout la séquence : **d\_a\_t ;a , 1** dans ces fichiers (soit dans leurs noms, soit dans leurs contenus).

Voici un Screenshot du dossier contenant ces fichiers :

Nom	Modifié le	Type	Taille
123d_a_t;a , 11234	02/07/2021 20:06	Microsoft Access ...	484 Ko
d_a_t;a , 111	02/07/2021 20:07	Archive WinRAR ZIP	1 Ko
file 1	02/07/2021 20:16	Document Micros...	13 Ko
file 2	02/07/2021 20:11	Fichier PDF	356 Ko
file 3	02/07/2021 20:12	Fichier TXT	1 Ko
file 4	02/07/2021 22:00	Document Micros...	13 Ko
file 4	02/07/2021 22:00	Fichier PDF	402 Ko
testd_a_t;a , 1	02/07/2021 20:07	Fichier TXT	0 Ko

Figure 11 : Screenshot qui montre les fichiers qu'on essaye de trouver avec la recherche **Exclusive**.

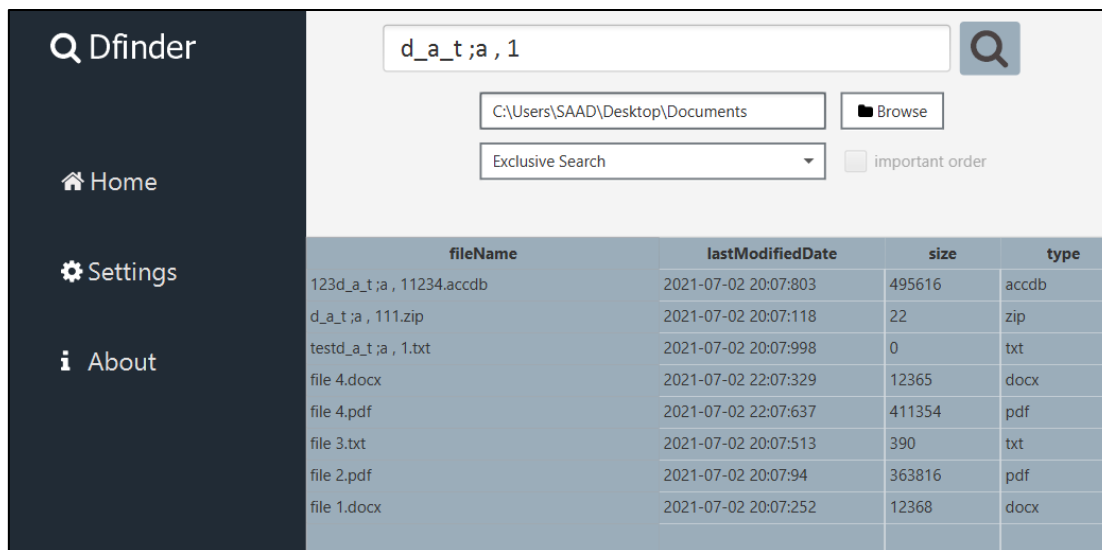
**Résultat :**

Figure 12 : Screenshot qui montre le résultat de la recherche Exclusive (affiché dans l'interface graphique de l'application).

**Interprétations des figures :**

- Le résultat montre bien que l'application arrive à rechercher de façon exclusive dans les noms et les contenus des fichiers. *Figure 14*
- Les trois premiers fichiers sont ceux dont leurs noms correspondent à la recherche, ils ne sont pas triés (affichées dans l'ordre de parcours de l'arborescence). *Figure 15*
- Les cinq derniers fichiers sont ceux dont leurs contenus correspondent à la recherche, ils sont triés : les fichiers sont triés selon du nombre d'occurrence, puis par date de modification la plus récente. *Figure 15*

```

C:\Users\SAAD\Desktop\JavaFx Dfinder App > history-saves > output_Exclusive[07-02-2021 22:01:12].txt
1  ===== Récapitulatif =====
2  [SYSTEM] : la methode de recherche choisie : Recherche Exclusive
3  : le chemin du répertoire choisi : C:\Users\SAAD\Desktop\Documents
4  : la phrase à rechercher : d_a_t ; a , 1
5  =====
6
7  [SYSTEM] : Voici les fichiers dont le nom contient la phrase donnée telle quelle:
8  : ( Total = 3 )
9
10 [file N°1] : file name : 123d_a_t ; a , 11234.accdb
11             last modification date : 02/07/2021 20:06:28
12             file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\123d_a_t ; a , 11234.
13             file size : 495616 octets
14
15 [file N°2] : file name : d_a_t ; a , 111.zip
16             last modification date : 02/07/2021 20:07:26
17             file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\d_a_t ; a , 111.zip
18             file size : 22 octets
19
20 [file N°3] : file name : testd_a_t ; a , 1.txt
21             last modification date : 02/07/2021 20:07:12
22             file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\testd_a_t ; a , 1.txt
23             file size : 0 octets
24
25 =====

```

```

25 =====
26 [SYSTEM] : Voici les fichiers qui contiennent la phrase donnée telle quelle :
27 : ( les fichiers sont triés par nombre d'occurrence de la phrase donnée,
28 :   date de modification la plus récente la taille la plus petite. )
29 : ( Total = 5 )
30
31 [Occurrence = 28] :
32 [file N°1] :      file name : file 4.docx
33                last modification date : 02/07/2021 22:00:26
34                file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\file 4.docx
35                file size : 12365 octets
36
37 [file N°2] :      file name : file 4.pdf
38                last modification date : 02/07/2021 22:00:08
39                file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\file 4.pdf
40                file size : 411354 octets
41
42 [file N°3] :      file name : file 3.txt
43                last modification date : 02/07/2021 20:12:09
44                file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\file 3.txt
45                file size : 390 octets
46
47 [Occurrence = 15] :
48 [file N°1] :      file name : file 2.pdf
49                last modification date : 02/07/2021 20:11:06
50                file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\file 2.pdf
51                file size : 363816 octets
52
53 [Occurrence = 11] :
54 [file N°1] :      file name : file 1.docx
55                last modification date : 02/07/2021 20:16:21
56                file path : C:\Users\SAAD\Desktop\Documents\123\461561\146561\random directory\dossier 2\dossier 3\file 1.docx
57                file size : 12368 octets
58
59 =====
60 [SYSTEM] * Temps d'exécution : 76720 Millisecondes.
61 [SYSTEM] * fin ! ( le fichier d'historique de recherche output_Exclusive[07-02-2021 22;01;12] .txt a été enregistré avec succès )
62 =====

```

Figure 13 : Screenshot qui montre le résultat de la recherche Exclusive (affiché dans le fichier History/Save).

**Conclusion :** le teste montre bien que l'application arrive à rechercher avec la méthode Exclusive dans les dossiers, trouver les fichiers correspondants à la recherche, puis les trier (selon les critères cités dans la partie théorique).

### Teste 3 : teste de la recherche Approximative :

**Situation :** on va tester si l'application arrive à rechercher et trier selon le nombre d'occurrence avec la recherche **Approximative**, pour se faire : on a créé quelques fichiers, puis on a dispersé un peu partout les mots : **labriji isbaine** dans ces fichiers (soit dans leurs noms, soit dans leurs contenus).

Voici un Screenshot du dossier contenant ces fichiers :

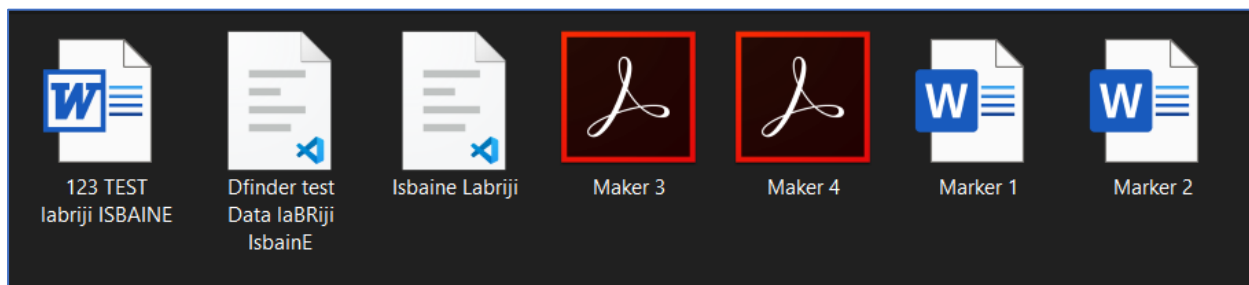


Figure 14 : Screenshot qui montre les fichiers qu'on essaye de trouver avec la recherche *Approximative*.

### Résultat :

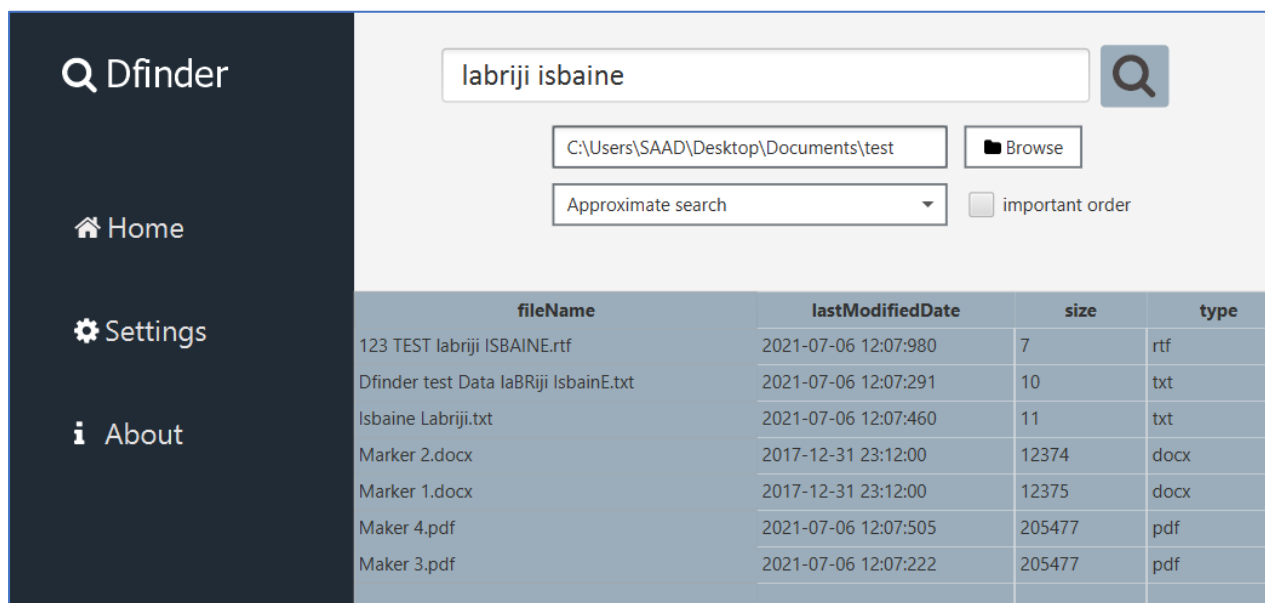


Figure 15 : Screenshot qui montre le résultat de la recherche *Approximative* (affiché dans l'interface graphique de l'application).

### Interprétations des figures : (suite dans la page suivante)

- Le résultat montre bien que l'application arrive à rechercher de façon Approximative dans les noms et les contenus des fichiers. *Figure 16 et Figure 17*
- Les trois premiers fichiers sont ceux dont leurs noms correspondent à la recherche : ils contiennent les mots recherchés **labriji isbaine**, ils sont triés :
  - Ils contiennent les mots **labriji isbaine** avec le même nombre d'occurrence, Donc ils ont un **score = 1 + 1 = 2**, (méthode de calcul 1 : **METH1**) *Figure 18*
  - Ils sont triés donc avec le 2<sup>ème</sup> critère : la taille du fichier (7 puis 10 puis 11).

```

C: > Users > SAAD > Desktop > 12365 > history-saves > output_Approximative[07-06-2021 13:00:37].txt
1  ===== Récapitulatif =====
2  [SYSTEM] : la methode de recherche choisie : Recherche Approximative
3            : la methode de calcul du score choisie : METH1
4            : le chemin du répertoire choisi : C:\Users\SAAD\Desktop\Documents\test
5            : le/les mot(s) à rechercher : labriji isbaine
6  =====
7
8  [SYSTEM] : Voici les fichiers dont le nom contient les mots donnés ( ordre non pris en considération )
9            : ( Total = 3 )
10
11 [file N°1] :   file name : 123 TEST labriji ISBAINE.rtf
12              last modification date : 06/07/2021 12:30:00
13              file path : C:\Users\SAAD\Desktop\Documents\test\123 TEST labriji ISBAINE.rtf
14              file size : 7 octets
15
16 [file N°2] :   file name : Dfinder test Data laBRiji IsbainE.txt
17              last modification date : 06/07/2021 12:51:19
18              file path : C:\Users\SAAD\Desktop\Documents\test\Dfinder test Data laBRiji IsbainE.txt
19              file size : 10 octets
20
21 [file N°3] :   file name : Isbaine Labriji.txt
22              last modification date : 06/07/2021 12:51:24
23              file path : C:\Users\SAAD\Desktop\Documents\test\Isbaine Labriji.txt
24              file size : 11 octets
25
26 =====

```

Figure 18 : Screenshot qui montre le résultat de la recherche Approximative (affiché dans le fichier History/Save).

```

=====
[SYSTEM] : Voici les fichiers qui contiennent les mots donnés :
          : ( les fichiers sont triés selon un score, date de modification
          :   la plus récente la taille la plus petite. )
          : ( Total = 4 )

[Score = 7] :
  [file N°1] :   file name : Marker 2.docx
                last modification date : 31/12/2017 23:00:00
                file path : C:\Users\SAAD\Desktop\Documents\test\Marker 2.docx
                file size : 12374 octets

  [file N°2] :   file name : Marker 1.docx
                last modification date : 31/12/2017 23:00:00
                file path : C:\Users\SAAD\Desktop\Documents\test\Marker 1.docx
                file size : 12375 octets

[Score = 6] :
  [file N°1] :   file name : Maker 4.pdf
                last modification date : 06/07/2021 12:47:05
                file path : C:\Users\SAAD\Desktop\Documents\test\Maker 4.pdf
                file size : 205477 octets

  [file N°2] :   file name : Maker 3.pdf
                last modification date : 06/07/2021 12:46:37
                file path : C:\Users\SAAD\Desktop\Documents\test\Maker 3.pdf
                file size : 205477 octets
=====

```

Figure 19 : Screenshot qui montre le résultat de la recherche Approximative (affiché dans le fichier History/Save). *Suite*

## Interprétations des figures : (suite)

- Les quatre derniers fichiers sont ceux dont leurs contenus correspondent à la recherche, ils sont triés : les fichiers sont triés :
  - Les fichiers **Marker 2.docx** et **Marker 1.docx** ont un score de 7, ils sont donc affichés en premier, ils ont la même date de modification, mais le fichier **Marker 2.docx** est plus léger donc affiché en premier. *Figure 17 et 18*
  - Les fichiers **Marker 4.pdf** et **Marker 3.pdf** ont un score de 6, ils sont donc affichés en dernier, ils ont la même taille, mais le fichier **Marker 4.pdf** a une date de modification plus récente donc affiché en premier. *Figure 17 et 18*

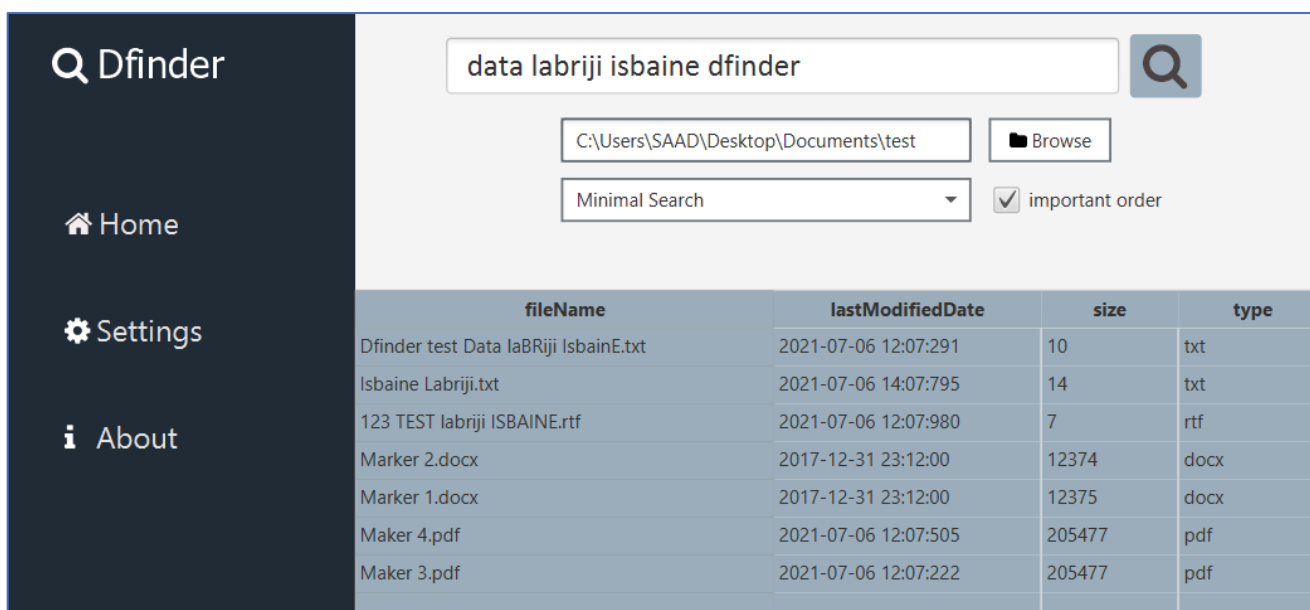
**Conclusion :** le teste montre bien que l'application arrive à rechercher avec la méthode Approximative dans les dossiers, trouver les fichiers correspondants à la recherche, puis les trier (selon les critères cités dans la partie théorique).

#### Teste 4 : teste de la recherche Minimale : (avec ordre important)

**Situation :** on va tester si l'application arrive à rechercher et trier selon le nombre d'occurrence avec la recherche **Minimale**, pour se faire : on a créé quelques fichiers, puis on a dispersé parmi les mots : **data labriji isbaine dfinder** un peu partout dans ces fichiers (soit dans leurs noms, soit dans leurs contenus).

Voici un Screenshot du dossier contenant ces fichiers : *Figure 16*

#### Résultat :



fileName	lastModifiedDate	size	type
Dfinder test Data laBRiji Isbaine.txt	2021-07-06 12:07:291	10	txt
Isbaine Labriji.txt	2021-07-06 14:07:795	14	txt
123 TEST labriji ISBAINE.rtf	2021-07-06 12:07:980	7	rtf
Marker 2.docx	2017-12-31 23:12:00	12374	docx
Marker 1.docx	2017-12-31 23:12:00	12375	docx
Maker 4.pdf	2021-07-06 12:07:505	205477	pdf
Maker 3.pdf	2021-07-06 12:07:222	205477	pdf

Figure 20 : Screenshot qui montre le résultat de la recherche Minimale (affiché dans l'interface graphique de l'application).

#### Interprétations des figures : (suite dans la page suivante)

- Le résultat montre bien que l'application arrive à rechercher de façon Minimale dans les noms et les contenus des fichiers. *Figure16 et Figure 20*
- Les trois premiers fichiers sont ceux dont leurs noms correspondent à la recherche : ils contiennent des mots parmi : **data labriji isbaine dfinder**, ils sont triés :
  - D'après la formule théorique de calcul :
    - Le fichier : **Dfinder test Data laBRiji Isbaine.txt** a un score  
 $= 1*1 + 1*4 + 1*2 + 1*1 = 8$
    - Le fichier : **Isbaine Labriji.txt** a un score  
 $= 1*2 + 1*3 = 5$
    - Le fichier : **123 TEST labriji ISBAINE.rtf** a un score  
 $= 1*2 + 1*3 = 5$
  - Le fichier **Isbaine Labriji.txt** a une date de modification plus récente, donc il est classé en premier. *Figure 20*

```

C:\Users\SAAD\Desktop\12365\history-saves > output_Minimale[07-06-2021 14:56:10].txt
1  ===== Récapitulatif =====
2  [SYSTEM] : la methode de recherche choisie : Recherche Minimale
3  : la methode de calcul du score choisie : METH2
4  : le chemin du répertoire choisi : C:\Users\SAAD\Desktop\Documents\test
5  : le/les mot(s) à rechercher : data labriji isbaine dfinder
6
7  =====
8  [SYSTEM] : Voici les fichiers dont le nom contient au moins un mot de la phrase saisie :
9  : ( les fichiers sont triés selon un score, date de modification
10 :   la plus récente la taille la plus petite. )
11 : ( Total = 3 )
12
13 [Score = 10] :
14 [file N°1] :   file name : Dfinder test Data laBRiji IsbainE.txt
15               last modification date : 06/07/2021 12:51:19
16               file path : C:\Users\SAAD\Desktop\Documents\test\Dfinder test Data laBRiji IsbainE.txt
17               file size : 10 octets
18
19 [Score = 5] :
20 [file N°1] :   file name : Isbaine Labriji.txt
21               last modification date : 06/07/2021 14:56:04
22               file path : C:\Users\SAAD\Desktop\Documents\test\Isbaine Labriji.txt
23               file size : 14 octets
24
25 [file N°2] :   file name : 123 TEST labriji ISBAINE.rtf
26               last modification date : 06/07/2021 12:30:00
27               file path : C:\Users\SAAD\Desktop\Documents\test\123 TEST labriji ISBAINE.rtf
28               file size : 7 octets
29
30 =====

```

Figure 21 : Screenshot qui montre le résultat de la recherche Minimale (affiché dans le fichier History/Save).

```

31  =====
32 [SYSTEM] : Voici les fichiers qui contiennent les mots donnés :
33 : ( les fichiers sont triés selon un score, date de modification
34 :   la plus récente la taille la plus petite. )
35 : ( Total = 4 )
36
37 [Score = 18] :
38 [file N°1] :   file name : Marker 2.docx
39               last modification date : 31/12/2017 23:00:00
40               file path : C:\Users\SAAD\Desktop\Documents\test\Marker 2.docx
41               file size : 12374 octets
42
43 [file N°2] :   file name : Marker 1.docx
44               last modification date : 31/12/2017 23:00:00
45               file path : C:\Users\SAAD\Desktop\Documents\test\Marker 1.docx
46               file size : 12375 octets
47
48 [Score = 15] :
49 [file N°1] :   file name : Maker 4.pdf
50               last modification date : 06/07/2021 12:47:05
51               file path : C:\Users\SAAD\Desktop\Documents\test\Maker 4.pdf
52               file size : 205477 octets
53
54 [file N°2] :   file name : Maker 3.pdf
55               last modification date : 06/07/2021 12:46:37
56               file path : C:\Users\SAAD\Desktop\Documents\test\Maker 3.pdf
57               file size : 205477 octets
58
59  =====

```

Figure 22 : Screenshot qui montre le résultat de la recherche Minimale (affiché dans le fichier History/Save). *Suite*

**Interprétations des figures : (suite)**

- Les quatre derniers fichiers sont ceux dont leurs contenus correspondent à la recherche, ils sont triés : les fichiers sont triés :
  - Les fichiers **Marker 2.docx** et **Marker 1.docx** ont un score de 18, ils sont donc affichés en premier, ils ont la même date de modification, mais le fichier **Marker 2.docx** est plus léger donc affiché en premier. *Figure 20 et 22*
  - Les fichiers **Marker 4.pdf** et **Marker 3.pdf** ont un score de 15, ils sont donc affichés en dernier, ils ont la même taille, mais le fichier **Marker 4.pdf** a une date de modification plus récente donc affiché en premier. *Figure 20 et 22*

**Conclusion :** le teste montre bien que l'application arrive à rechercher avec la méthode Minimale (avec ordre important) dans les dossiers, trouver les fichiers correspondants à la recherche, puis triés leurs contenus (selon les critères cités dans la partie théorique).

**Conclusion générale :**

Le développement de l'application Dfinder dans le cadre du projet de fin de la première année à l'INPT, et sous l'encadrement de notre professeur Benomar, Nous a permet de mettre en évidence nos connaissances techniques acquises lors des cours de programmation, ainsi que nos connaissances managériales. La réalisation de ce travail est passé tout d'abord par la détermination des méthodes de recherches, puis le développement en mode console, et ensuite le développement de l'interface graphique qui a donné de la simplicité et la facilité tout en assurant le coté esthétique. Et donc on a pu arriver à nos objectifs phares à savoir la recherche dans des arborescences en examinant les noms des fichiers, ainsi que leur contenu. Ce qui n'est pas fourni par d'autre outils de recherche. Cependant, la réalisation de ce projet a connu beaucoup de difficultés, à savoir le choix des technologies à utiliser, le choix des versions de jdk correspondantes. et bien aussi le développement de l'interface graphique et sa liaison avec le Backend. Il est à noter que le projet n'arrêtera pas à ce point, on ajoutera d'autres fonctionnalités comme la décompression des fichier zip et rar, et la lecture des fichiers de type pptx, xls. Ainsi, donner à l'utilisateur de chercher dans le web. Bref, nous somme très fiers de ce résultat et nous essayerons de le mettre en valeur.