

# PROJET S4 2025 - Rapport de soutenance

Groupe Outta Control Rebel - Seekly

24 février 2025



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Le projet</b>	<b>3</b>
2.1	L'équipe Outta Control Rebel . . . . .	3
2.2	Origine . . . . .	4
<b>3</b>	<b>État de l'art</b>	<b>5</b>
<b>4</b>	<b>Objectifs</b>	<b>6</b>
4.1	- Recherche de fichiers . . . . .	6
4.2	- Recherche avancée . . . . .	6
4.3	- Interface utilisateur . . . . .	6
4.4	- Indexation . . . . .	6
4.5	- Performances . . . . .	7
4.6	- Personnalisation / Bonus . . . . .	7
<b>5</b>	<b>Utilisation de l'application</b>	<b>8</b>
5.1	- Interface en ligne de commande (CLI) . . . . .	8
5.2	- Interface graphique (GUI) . . . . .	8
<b>6</b>	<b>Avancement du projet</b>	<b>10</b>
6.1	- Application CLI . . . . .	10
6.2	- Algorithmes . . . . .	12
6.3	- Site Web . . . . .	15
<b>7</b>	<b>Découpage du projet</b>	<b>17</b>
7.1	- Répartition des tâches . . . . .	17
7.2	- Prévisions du cahier des charges . . . . .	17
7.3	- Avancement . . . . .	18
<b>8</b>	<b>Moyens mis en oeuvre</b>	<b>18</b>
8.1	- Moyens humains . . . . .	18
8.2	- Moyens matériels . . . . .	18
8.3	- Moyens techniques . . . . .	18
8.4	- Documentation et communication . . . . .	19
<b>9</b>	<b>Conclusion</b>	<b>20</b>

# 1 Introduction

Nom du projet : Seekly

Type de projet : Application de recherche de fichiers et contenus, similaire à Spotlight, écrit en Rust pour un environnement Linux.

Objectif principal : Créer une application rapide, efficace et minimaliste permettant aux utilisateurs de rechercher facilement des fichiers, dossiers, et informations spécifiques. Elle vise à simplifier leur expérience en éliminant le besoin de naviguer manuellement dans le système de fichiers, tout en leur offrant des fonctionnalités avancées pour optimiser leur productivité

Notions clés : recherche avancée, interface intuitive, indexation performante, optimisation de la productivité, personnalisation des recherches, sécurité et rapidité grâce à Rust.

## 2 Le projet

### 2.1 L'équipe Outta Control Rebel

Le groupe Outta Control Rebel est composé de quatre jeunes passionnés par l'informatique, les jeux vidéos et l'intelligence artificielle :

#### - **KHAMCHANE Hamza**

Je m'appelle Hamza, j'ai 19 ans et je viens Drancy. Le monde de l'informatique est monde qui ma toujours fasciné que ce soit dans sa complexité ou son charme. Je suis attirée par ce monde depuis mes 13 ans. C'est surtout grace a l'influence des mes 3 oncles maternelle que j'ai decouvert ce monde. Depuis cette age j'ai essayer de decouvrir ce monde par plusieurs moyen, un stage , des recherche personnelles, programmation de mon cotée etc. Et de fil en aiguille je me suis retrouve aujourd'hui a Epita.

#### - **RICHARD Aubin**

Je suis passionné depuis plusieurs années par l'informatique. Le lycée est le moment où l'informatique est devenu encore plus importante pour moi. J'ai commencer à apprendre différents langage de programation et également à réaliser de mini-projets. J'ai ensuite décidé de rejoindre Epita pour continue de faire et d'apprendre l'informatique.

#### - **SUILLEROT Gaetan**

Mon intérêt pour l'informatique a commencé au collège avec Scratch. Au lycée, la création de projets en équipe est devenue une passion. J'ai choisi l'EPITA Paris pour sa réputation internationale. Mon rôle au sein de l'équipe me tient à cœur. Ce projet me fera découvrir de nouveaux domaines et enrichir mes compétences.

## **- VIDAL-LEFEBVRE Oscar**

Je m'appelle Oscar, ma passion pour l'informatique remonte à mes histoires du soir quand j'étais enfant. L'intérêt que j'y porte vient des possibilités qu'elle offre, particulièrement dans l'électronique qui est le domaine dans lequel j'aimerais m'orienter. J'ai choisi de m'investir dans ce groupe pour leurs idées originales, leur assiduité et régularité qui m'ont particulièrement séduit.

## **2.2 Origine**

L'idée de Seekly est née d'une analyse approfondie des besoins des utilisateurs Linux en matière de recherche de fichiers. Lors de nos recherches initiales, nous avons constaté que de nombreux outils de recherche existants manquaient de rapidité, d'ergonomie ou de flexibilité, surtout sur les systèmes ayant une grande quantité de fichiers. Inspirés par des solutions comme Spotlight sur macOS, nous avons cherché à combiner la simplicité d'utilisation avec les capacités avancées offertes par des technologies modernes.

Nous avons commencé par analyser les points faibles des outils actuels comme "locate" et "find" sous Linux, qui, bien qu'efficaces, ne permettent pas toujours une expérience utilisateur fluide. Cela nous a conduit à l'idée de développer une application reposant sur un moteur d'indexation puissant, tout en offrant une interface moderne et personnalisable. Rust a été choisi comme langage principal pour sa performance, sa gestion de mémoire sécurisée et sa communauté active, ce qui garantit la pérennité de notre projet.

### 3 État de l’art

Pour comprendre les besoins et lacunes actuelles dans le domaine des outils de recherche de fichiers, nous avons étudié plusieurs solutions existantes :

**- Locate/Find (Linux) :**

Ces outils sont largement utilisés dans l’écosystème Linux pour rechercher des fichiers. Cependant, ils souffrent d’un manque d’ergonomie et de fonctionnalités avancées. Par exemple, ils ne permettent pas une mise à jour en temps réel des modifications du système de fichiers et nécessitent souvent des commandes complexes peu accessibles aux utilisateurs non techniques.

**- Spotlight (macOS) :**

Spotlight est une référence en termes d’ergonomie et de rapidité. Il combine une interface utilisateur fluide avec des résultats de recherche rapides grâce à son système d’indexation performant. Toutefois, ce logiciel est exclusif à l’écosystème Apple, laissant un vide dans l’univers Linux.

**- Catfish (Linux) :**

Catfish est une solution graphique basée sur les commandes find et locate. Bien qu’intuitive, elle reste limitée en termes de performances pour des systèmes de fichiers contenant plusieurs millions de fichiers.

**- Recoll (Linux) :**

Cet outil offre une recherche full-text avancée mais peut être complexe à configurer pour les utilisateurs non expérimentés. Sa rapidité dépend également de la configuration matérielle et de la taille des index.

L’analyse des outils actuels nous a permis d’identifier les besoins suivants, une interface simple et moderne, accessible à tous les utilisateurs, un système d’indexation performant permettant une mise à jour en temps réel, une prise en charge des filtres avancés pour des recherches complexes et une solution open-source conçue spécifiquement pour l’écosystème Linux.

## 4 Objectifs

### 4.1 - Recherche de fichiers

- Offrir une recherche rapide et intuitive permettant de localiser des fichiers par leur nom, leur extension, ou des mots-clés spécifiques.
- Permettre la recherche dans des répertoires précis ou sur l'ensemble du système de fichiers pour maximiser la flexibilité.
- Inclure des options permettant de trier et filtrer les résultats en fonction de critères personnalisés comme l'ordre alphabétique ou la pertinence.

### 4.2 - Recherche avancée

- Proposer des filtres précis tels que la taille des fichiers, la date de dernière modification ou encore leur type.
- Intégrer la prise en charge des expressions régulières, offrant ainsi aux utilisateurs expérimentés la possibilité d'effectuer des recherches complexes et ciblées.
- Ajouter une fonction de recherche approximative pour corriger automatiquement les fautes de frappe dans les requêtes.

### 4.3 - Interface utilisateur

- Concevoir une interface en ligne de commande (CLI) qui soit conviviale, épurée et simple d'utilisation pour les utilisateurs avancés et débutants.
- Concevoir une interface graphique (GUI) pour une accessibilité accrue auprès des utilisateurs moins familiers avec les outils en ligne de commande.
- Intégrer un retour visuel clair, comme une barre de progression pour les recherches longues, ou un affichage des suggestions basées sur les recherches précédentes.

### 4.4 - Indexation

- Développer un moteur d'indexation robuste capable de traiter efficacement des systèmes de fichiers contenant plusieurs millions d'éléments.
- Mettre en place un mécanisme de mise à jour en temps réel de l'index pour refléter instantanément les changements effectués sur le système de fichiers.
- Prévoir une gestion optimisée des ressources pour minimiser l'impact de l'indexation sur les performances globales du système.

## 4.5 - Performances

- Assurer une recherche quasi-instantanée même dans des environnements où la densité de fichiers est élevée.
- Exploiter pleinement les capacités offertes par Rust pour garantir une exécution rapide et une gestion de mémoire sûre.
- Tester et optimiser l'application pour les environnements à faibles ressources (ordinateurs plus anciens ou serveurs modestes).

## 4.6 - Personnalisation / Bonus

- Permettre aux utilisateurs de configurer facilement des chemins de recherche exclus ou prioritaires pour affiner les résultats selon leurs besoins.
- Proposer des raccourcis personnalisables permettant d'accéder rapidement aux recherches fréquentes ou aux paramètres favoris.
- Inclure un mode « sombre » et des options de thèmes pour une meilleure expérience visuelle adaptée à tous les goûts.

## 5 Utilisation de l'application

L'application Seekly est conçue pour offrir une expérience utilisateur intuitive et efficace. Voici un aperçu de son fonctionnement, accompagné de croquis représentant l'interface utilisateur (CLI et GUI).

### 5.1 - Interface en ligne de commande (CLI)

L'interface CLI est conçue pour être rapide et accessible aux utilisateurs avancés. Voici un exemple de commande et son résultat :

```
seekly search --name "rapport.pdf" --directory /home/user/documents
```

Résultat attendu :

```
Recherche en cours...  
Fichier trouvé : /home/user/documents/rapport.pdf  
Taille : 2.3 Mo  
Dernière modification : 12/01/2025
```

Croquis de l'interface CLI :

```
+-----+  
| Seekly CLI - Recherche de fichiers |  
+-----+  
> Commande : seekly search --name "exemple.txt"  
  
[Résultat 1] /home/user/exemple.txt  
[Résultat 2] /home/user/documents/exemple_rapport.txt
```

### 5.2 - Interface graphique (GUI)


Une interface graphique est prévue pour les utilisateurs qui préfèrent une approche visuelle. Elle comportera les éléments suivants :

- Une barre de recherche en haut pour entrer des mots-clés.
- Des options de filtrage sur le côté gauche (taille, date de modification, type de fichier).
- Une liste des résultats affichés au centre avec des détails sur chaque fichier.



Croquis de l'interface GUI :

# Seekly



**fichier\_1.exe - /home/user/bin/**

**fichier\_2.png - /home/user/images**

**fichier\_3.txt - /local/tmp/**

**fichier\_4.wav - /usb/dev/media**

Cette interface sera développée en utilisant des bibliothèques comme egui ou tui pour garantir une intégration fluide et rapide sur les distributions Linux.

## 6 Avancement du projet

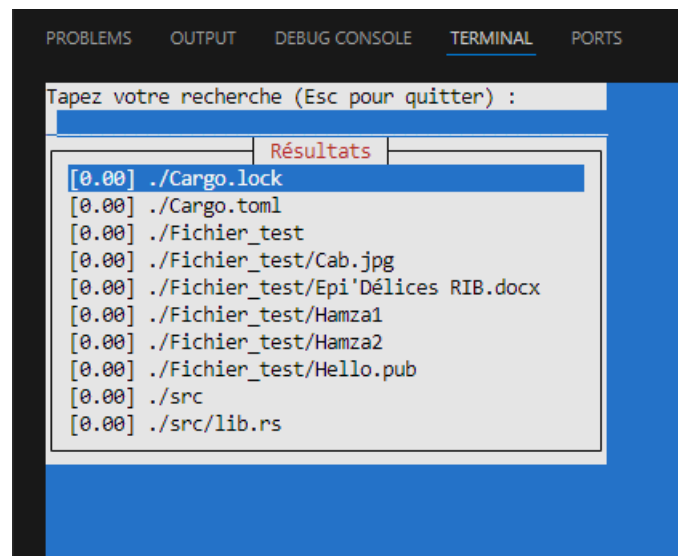
Pour cette soutenance nous avons réaliser plusieurs choses. Une application CLI capable de trouver et ouvrir un fichier en utilisant divers algorithmes. Egalement la création d'un site web récapitulatif de notre projet.

### 6.1 - Application CLI

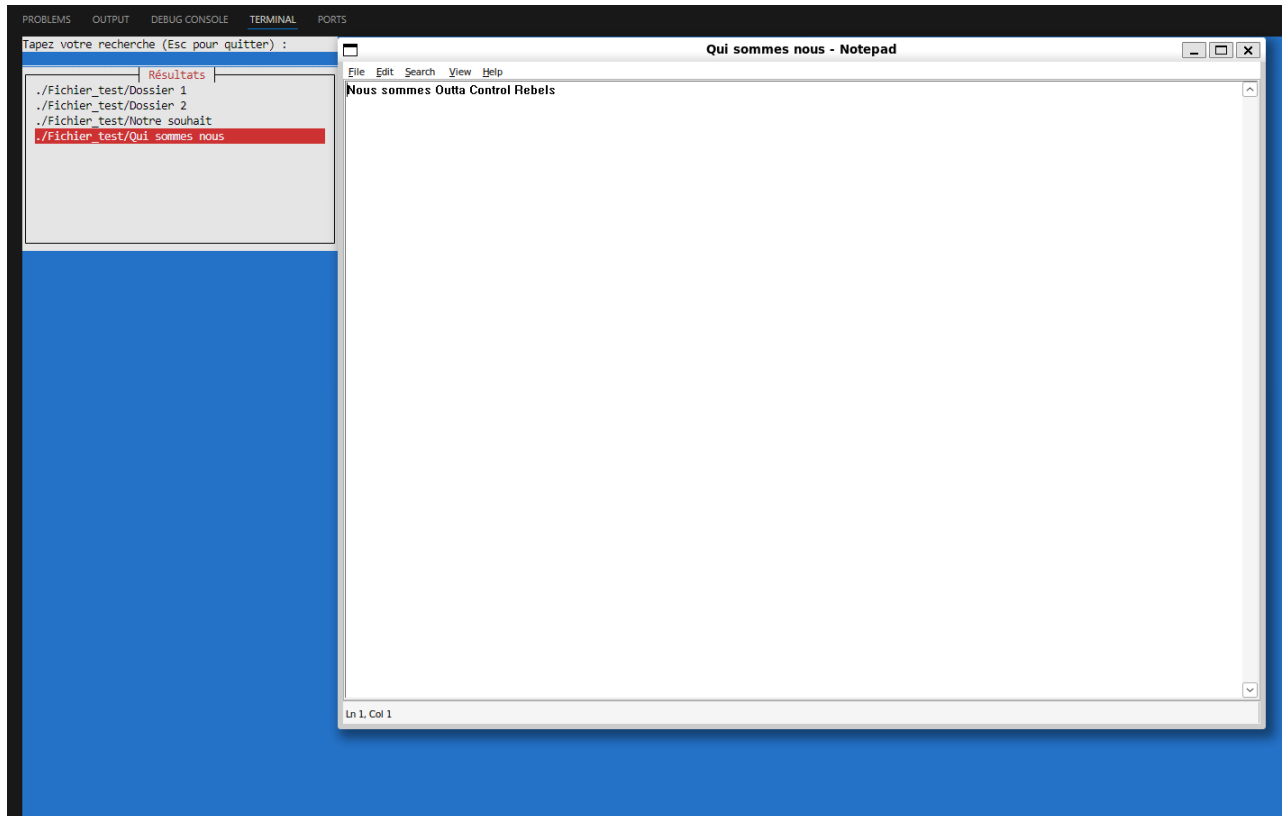
Dans le cadre de notre projet, nous avons choisi de développer une interface en ligne de commande (CLI). Ce choix repose sur plusieurs raisons essentielles. Une interface CLI consomme beaucoup moins de ressources qu'une interface graphique. Elle permet d'éviter les temps de chargement liés aux rendus visuels complexes. L'affichage en mode texte garantit une réactivité immédiate, ce qui est crucial pour une recherche en temps réel. Une CLI permet une navigation rapide avec seulement un clavier, sans nécessiter de souris. Elle convient parfaitement à un usage avancé, où l'utilisateur veut effectuer des actions rapidement sans distractions. L'affichage sous forme de liste rend les résultats accessibles d'un simple coup d'œil.

Notre CLI a été conçu pour offrir une expérience de recherche fluide, rapide et intuitive. Il permet de naviguer efficacement dans les fichiers et dossiers, tout en intégrant des fonctionnalités avancées pour répondre aux besoins des utilisateurs. Voici les principales fonctionnalités de notre interface en ligne de commande :

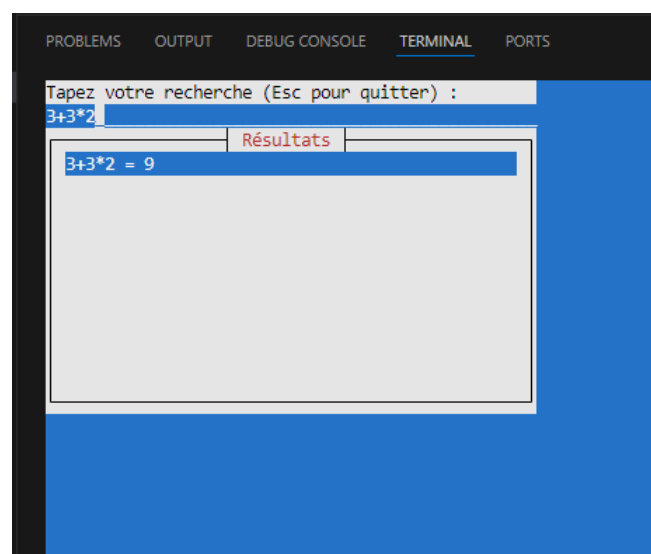
L'utilisateur peut taper un mot-clé et voir instantanément les fichiers et dossiers correspondants s'afficher. Les résultats sont triés intelligemment grâce à un système de notation qui évalue leur pertinence.



En sélectionnant un fichier, le programme tente de l'ouvrir avec l'application par défaut du système. Si un problème survient, un message d'erreur clair est affiché pour guider l'utilisateur.



Le moteur de recherche détecte si l'entrée est une expression mathématique. Il peut résoudre des calculs simples directement dans le terminal et afficher le résultat instantanément. Cette fonctionnalité permet une utilisation rapide du CLI comme calculatrice intégrée.



L'interface repose sur la bibliothèque Cursive, offrant un affichage clair et interactif. L'utilisateur peut naviguer dans les résultats à l'aide des flèches du clavier. Un simple appui sur Échap (Esc) permet de quitter proprement le programme.

Grâce à ces fonctionnalités, notre CLI est un outil rapide, pratique et puissant pour la gestion et la recherche de fichiers, tout en restant léger et optimisé pour une utilisation quotidienne.

## 6.2 - Algorithmes

Dans de nombreux systèmes informatiques, la recherche de fichiers est une fonctionnalité essentielle, permettant aux utilisateurs de localiser rapidement des documents, programmes ou tout autre contenu en fonction d'un critère spécifique. Que ce soit pour un gestionnaire de fichiers, un moteur de recherche local ou un outil d'indexation avancé, une implémentation efficace de la recherche doit être à la fois rapide et adaptable aux besoins des utilisateurs.

Dans ce projet, nous implémentons une fonction permettant de rechercher des fichiers dans un répertoire donné, avec la possibilité d'effectuer une recherche récursive dans les sous-répertoires. Cette approche est particulièrement utile lorsqu'il est nécessaire d'explorer de grandes arborescences de fichiers, comme celles présentes dans les systèmes d'exploitation modernes ou les bases de données de documents.

La fonction principale du fichier est `search-files`, qui permet de rechercher des fichiers dans un répertoire donné, avec ou sans recherche récursive. Elle prend trois paramètres :

- `base-path` : le répertoire de départ de la recherche.
- `query` : la chaîne à rechercher dans les fichiers.
- `recursive` : un booléen indiquant si la recherche doit être récursive dans les sous-répertoires.

La fonction retourne un vecteur de résultats sous forme de `SearchResult`, où chaque résultat représente un fichier correspondant à la requête.

```
pub fn search_files(base_path: &str, query: &str, recursive: bool) -> Vec<SearchResult> {  
    let mut results = Vec::new();  
    let path = Path::new(base_path);
```

#### Recherche non-réursive :

Si le paramètre `recursive` est défini sur `false`, la fonction utilise `std : :fs : :read-dir` pour lire les fichiers du répertoire spécifié dans `base-path`. Cela permet de ne pas parcourir les sous-répertoires. La fonction `read-dir` retourne un itérateur sur les entrées du répertoire, et pour chaque entrée, le chemin du fichier est récupéré et comparé à la requête via `matcher : :matches`.

```
if !recursive {
  if let Ok(entries) = std::fs::read_dir(path) {
    for entry in entries.flatten() {
      let path_buf = entry.path();
      if matcher::matches(&path_buf, query) {
        results.push(SearchResult { path: path_buf });
      }
    }
  }
}
```

`std : :fs : :read-dir(path)` permet de lire les fichiers dans le répertoire spécifié par `base-path`. Pour chaque entrée valide, la méthode `entry.path()` récupère le chemin du fichier. `matcher : :matches(path-buf, query)` est utilisé pour vérifier si le fichier correspond à la requête `query`. Si c'est le cas, le chemin est ajouté au vecteur `results` dans un objet `SearchResult`.

#### Recherche récursive :

Si le paramètre `recursive` est `true`, la recherche devient récursive grâce à `WalkDir`. Ce module parcourt le répertoire de manière récursive et permet de traiter également les sous-répertoires. Le filtrage avec `filter-map(|e| e.ok())` permet de s'assurer que seules les entrées valides sont traitées.

```
else {
  for entry in WalkDir::new(path).into_iter().filter_map(|e| e.ok()) {
    let path_buf = entry.path().to_path_buf();
    if matcher::matches(&path_buf, query) {
      results.push(SearchResult { path: path_buf });
    }
  }
}
```

#### Fonction score-results :

La fonction `score-results` permet d'attribuer un score de pertinence aux fichiers trouvés lors d'une recherche. Elle prend une liste de résultats (`SearchResult`), qui contiennent les chemins des fichiers trouvés, et une requête sous forme de texte. L'objectif est de comparer la requête avec le nom de chaque fichier et d'attribuer un score en fonction de la similarité. Plus la requête est proche du nom du fichier, plus le score est élevé. Ce mécanisme permet de trier les résultats en fonction de leur pertinence, facilitant ainsi la recherche des fichiers les plus appropriés.

Cette fonction prends deux paramètres :

**results** : Une référence à un tableau de SearchResult. Il s'agit des résultats de recherche qui contiennent des chemins de fichiers.

**query** : La chaîne de caractères que l'utilisateur recherche dans les noms de fichiers. La fonction retourne un vecteur de ScoredResult, qui contient les chemins des fichiers avec un score associé à chaque fichier, basé sur la correspondance avec la requête.

```
pub fn score_results(results: &[SearchResult], query: &str) -> Vec<ScoredResult>
```

La fonction utilise un `iter()` sur `results` pour parcourir chaque `SearchResult` et ensuite applique une fonction de transformation (`map`) pour chaque élément.

Pour chaque `SearchResult` :

Le nom du fichier est extrait avec `r.path.file-name()`. Si le nom est trouvé, il est converti en chaîne de caractères avec `to-string-lossy()` et ensuite transformé en minuscules avec `to-lowercase()`.

Si le nom du fichier contient la requête, le score est calculé en fonction du rapport entre la longueur de la requête et la longueur du nom du fichier. Le score est défini comme la longueur de la requête divisée par la longueur du nom du fichier.

Si le nom du fichier ne contient pas la requête ou est vide, le score est défini sur 0.

```
results.iter().map(|r|
{
    let file_name = r.path.file_name()
        .map(|s| s.to_string_lossy().to_lowercase())
        .unwrap_or_default();
    let score = if file_name.contains(&query_lc) && !file_name.is_empty()
    {
        query_lc.len() as f64 / file_name.len() as f64
    }
    else
    {
        0.0
    };
    ScoredResult { path: r.path.clone(), score }
}).collect()
```

`r.path.file-name()` : Extrait le nom du fichier à partir du chemin `r.path`.

`to-string-lossy()` : Convertit le nom en une chaîne de caractères, en gérant les erreurs de conversion si nécessaire.

`to-lowercase()` : Transforme la chaîne de caractères en minuscules pour une recherche insensible à la casse.

`contains(query_lc)` : Vérifie si le nom du fichier contient la requête (convertie elle aussi en minuscules).

Le calcul du score est effectué comme suit : la longueur de la requête est divisée par la longueur du nom du fichier, donnant ainsi une mesure de la correspondance relative.

## 6.3 - Site Web

Le site web de Seekly est conçu pour offrir une expérience utilisateur fluide et intuitive, permettant aux visiteurs de comprendre en quelques clics les fonctionnalités et avantages du logiciel. Il est organisé en plusieurs sections essentielles :

Dès l'arrivée sur le site, l'utilisateur est accueilli par une interface moderne et épurée. Cette page met en avant :

- Un slogan accrocheur qui résume l'essence de Seekly (ex. : "L'outil de recherche intelligent.").
- Une brève description du projet, expliquant pourquoi Seekly est une solution innovante.
- Un bouton d'action (CTA) bien visible pour télécharger le logiciel ou accéder à la démonstration.



Le but est de capter immédiatement l'intérêt du visiteur et de lui donner envie d'explorer davantage.

La section fonctionnalités détaille les atouts de Seekly en expliquant comment il révolutionne la recherche de fichiers. Elle est divisée en plusieurs points :

- L'outil CLI.

## L'outil CLI

L'interface CLI est conçue pour être rapide et accessible aux utilisateurs avancés. Voici un exemple de commande et son résultat :

```
seekly search --name "rapport.pdf" --directory /home/user/docume
```

Commande de recherche Seekly dans le terminal

Résultat attendu :

```
Recherche en cours...  
Fichier trouvé : /home/user/documents/rapport.pdf  
Taille : 2.3 Mo  
Dernière modification : 12/01/2025
```

Execution de Seekly et recherche du fichier spécifié

— L'interface graphique (GUI).

## L'interface graphique (GUI)

# Seekly



fichier\_1.exe - /home/user/bin/

fichier\_2.png - /home/user/images

La section équipe qui présente les différents membres du groupe. Enfin la section ressources où l'on trouve les ressources utilisées pour le projet.



Chaque page est conçue pour offrir une navigation intuitive et une expérience utilisateur agréable.

Nous avons utilisé HTML5 et CSS3 pour construire une interface responsive. Chaque fonctionnalité est isolée pour une meilleure maintenabilité.

Un design minimaliste mettant l'accent sur la clarté et l'ergonomie. Utilisation de styles CSS simples pour améliorer l'esthétique.

Le site web répond aux objectifs fixés, les visiteurs peuvent facilement comprendre les étapes nécessaires pour résoudre une grille grâce à une présentation claire et détaillée. Les membres de l'équipe sont valorisés via une page dédiée mettant en avant leurs contributions respectives. Egalement les rapports de soutenance sont accessibles via des liens de téléchargement.

## 7 Découpage du projet

### 7.1 - Répartition des tâches

	<b>Gaetan</b>	<b>Hamza</b>	<b>Oscar</b>	<b>Aubin</b>
<b>CLI</b>	Second	Main		
<b>GUI</b>			Main	Second
<b>Site Web</b>	Main		Second	
<b>Design</b>		Second		Main
<b>Algorithme</b>	Main		Second	

### 7.2 - Prévisions du cahier des charges

	<b>soutenance 1</b>	<b>soutenance 2</b>	<b>soutenance 3</b>
<b>CLI</b>	<b>65%</b>	<b>100%</b>	<b>100%</b>
<b>GUI</b>	<b>0%</b>	<b>60%</b>	<b>100%</b>
<b>Site Web</b>	<b>75%</b>	<b>95%</b>	<b>100%</b>
<b>Design</b>	<b>0%</b>	<b>60%</b>	<b>100%</b>
<b>Algorithme</b>	<b>45%</b>	<b>85%</b>	<b>100%</b>

## 7.3 - Avancement

	soutenance 1	soutenance 2	soutenance 3
CLI	85%	100%	100%
GUI	0%	60%	100%
Site Web	75%	95%	100%
Design	0%	60%	100%
Algorithme	55%	85%	100%

Par rapport aux prévisions de notre cahier des charges la partie CLI est plus avancées. Nous avons réussi à implémenter beaucoup de chose pour cette partie mais il reste encore des choses à implémenter notamment comme l'ouverture de tout type de fichier. La partie algorithme est bien avancée mais il reste de l'optimisation pour la recherche qui peut prendre du temps en fonction de où ce trouve le fichier ou dossier à trouver. Enfin le site web est sur la bonne voie il doit juste maintenant évoluer en même temps que le projet.

## 8 Moyens mis en oeuvre

Pour mener à bien le projet Seekly, l'équipe s'appuie sur une combinaison de technologies, d'outils et de compétences adaptés au développement d'une application performante et innovante. Voici un aperçu des moyens techniques, humains et matériels mobilisés :

### 8.1 - Moyens humains

- Équipe projet : Composée de quatre membres ayant des compétences complémentaires en programmation, gestion de projet.
- Réunions hebdomadaires : Organisation de points réguliers pour assurer l'avancement du projet et résoudre les éventuels problèmes.
- Formation continue : Chaque membre est encouragé à approfondir ses connaissances sur Rust, les moteurs d'indexation, et les outils associés.

### 8.2 - Moyens matériels

Ordinateurs personnels des membres de l'équipe équipés de distributions Linux.  
Utilisation de logiciels de gestion de versions (Git).

### 8.3 - Moyens techniques

Utilisation de Rust, choisi pour ses performances et sa gestion sécurisée de la mémoire.  
Bibliothèques et outils tiers : Nous allons utiliser différentes bibliothèques pour pouvoir réaliser une interface graphique propre. Egalement nous utiliserons cargo.

## 8.4 - Documentation et communication

Documentation technique rédigée avec LaTeX.

Documentation utilisateur incluant des guides pratiques pour l'installation et l'utilisation de l'application.

Outils de communication comme Discord pour coordonner l'équipe et gérer les tâches.

Ces moyens combinés assurent une gestion efficace du projet, tout en garantissant la qualité et la pérennité de l'application.

## 9 Conclusion

Le projet Seekly est la construction d’une application de recherche de fichiers dans l’écosystème Linux. Ce cahier des charges constitue le fondement du développement de cette application et met en avant une méthodologie rigoureuse et une vision claire des objectifs.

En combinant des technologies modernes comme Rust, des bibliothèques performantes, et une approche utilisateur centrée sur l’ergonomie, Seekly se démarquera par sa rapidité, sa précision, et sa flexibilité. Notre équipe a pris soin d’identifier les lacunes des outils existants pour concevoir une solution sur mesure adaptée aux besoins des utilisateurs, qu’ils soient novices ou expérimentés.

Au-delà de l’aspect technique, ce projet est aussi une opportunité pour les membres de l’équipe de renforcer leurs compétences en gestion de projet, programmation, et travail collaboratif. Chacun des membres apporte des compétences uniques, et la répartition claire des rôles garantit une progression efficace.

Grâce à une organisation bien définie, des moyens matériels adaptés, et une motivation commune. Nous visons à livrer une application robuste, performante, et extensible, qui pourra continuer à évoluer et à répondre aux futurs besoins de la communauté open-source.

Ce cahier des charges servira de guide et de référence tout au long du projet, assurant que chaque étape est accomplie avec succès et que le résultat final reflète notre engagement technique et utilisateur.