Fiche d’investigation de fonctionnalité

Fonctionnalité : Recherche principale de recettes	**Fonctionnalité #2**
Problématique : Permettre à l’utilisateur de rechercher efficacement une recette parmi une liste. Nous cherchons l’implémentation qui offre la meilleure performance tout en restant maintenable.	

Option 1 : Boucles natives (for, while)
Dans cette version, on utilise des boucles classiques (for, while) pour parcourir les recettes et leurs ingrédients. On interrompt les boucles dès qu'un résultat est trouvé pour améliorer la rapidité.

Avantages	**Inconvénients**
⊕ Contrôle précis sur l’optimisation (ex: continue, break) ⊕ Code simple pour ceux habitués aux boucles classiques	⊖ Code plus long et légèrement plus verbeux ⊖ Moins lisible pour certains développeurs modernes habitués aux méthodes fonctionnelles

Résultat de performance :
Lors des tests de performance, il s'est avéré que cette implémentation réalisait 4 207 opérations par seconde.

Option 2 : Programmation fonctionnelle (filter, some, etc.)
Dans cette version, on utilise des méthodes d’Array (filter, some) pour parcourir les recettes et chercher la correspondance.

Avantages	**Inconvénients**
⊕ Code plus concis et lisible ⊕ Expressivité naturelle avec filter et some pour la recherche ⊕ Plus conforme aux bonnes pratiques modernes JS	⊖ Moins de contrôle fin sur l'optimisation interne

Résultat de performance :
Lors des tests de performance, il s'est avéré que cette implémentation réalisait 4 127 opérations par seconde.

Solution retenue :
Nous avons retenu l'implémentation boucles natives.Même si la différence de performance est légère, cette version reste plus rapide et offre une optimisation plus fine, ce qui est important sur des volumes de données élevés. La maintenabilité reste acceptable car la logique est claire et bien structurée.

Annexes

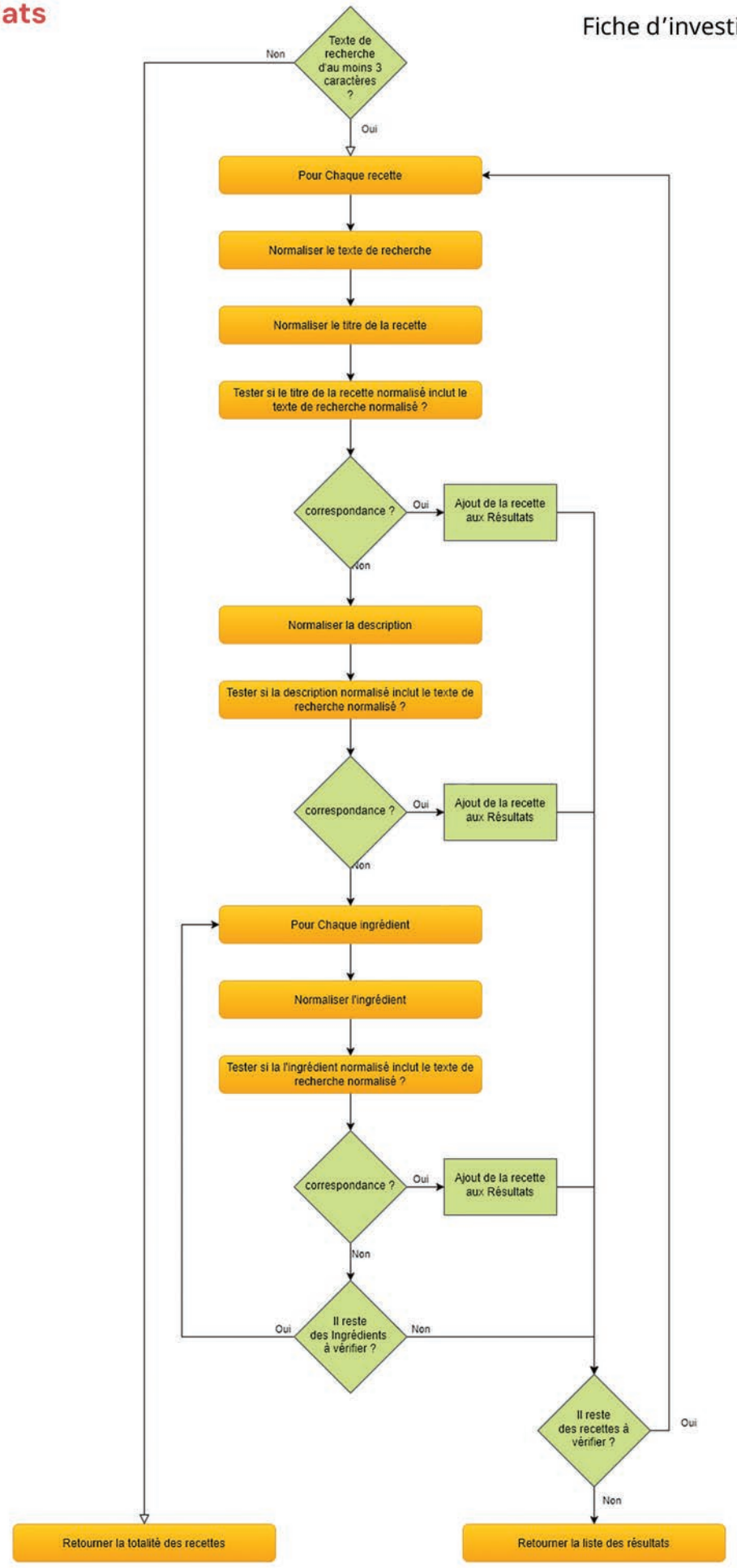

Figure 1 - Algorigramme recherche principale