

Fiche d'investigation de fonctionnalité

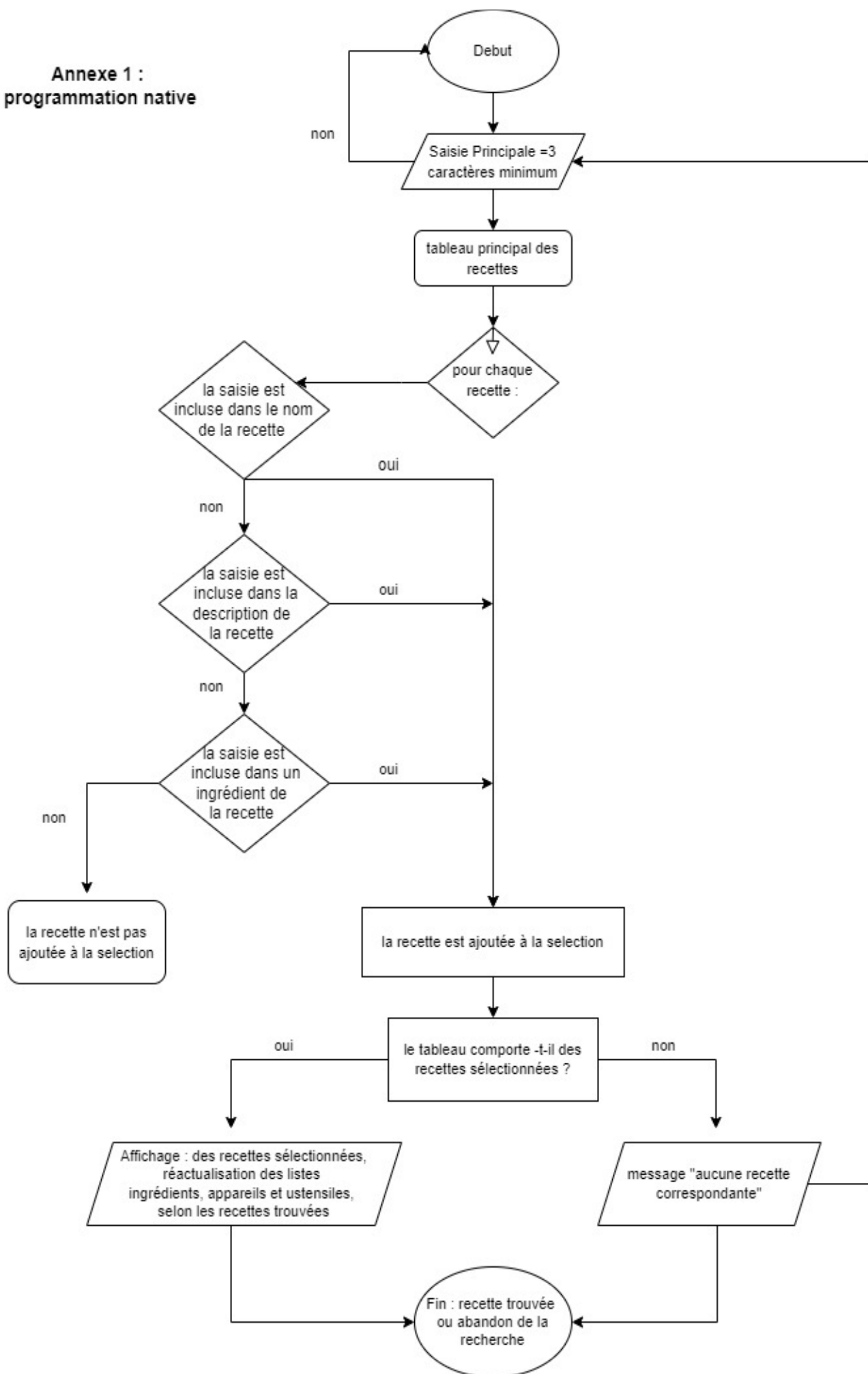
Fonctionnalité : Recherche de recettes	Fonctionnalité #1
Problématique : Filtrage des recettes dans l'interface utilisateur, l'utilisateur doit pouvoir accéder rapidement à la recette correspondant à sa recherche	

Option 1 : Programmation fonctionnelle (annexe 1) Utilisation des méthodes de l'objet Array (forEach, Filter...) Emploi ici de la méthode « filter » qui filtre les recettes suivant la saisie effectuée et les correspondances trouvées dans le nom ou la description ou les ingrédients de la recette. La recette trouvée est ajoutée à un tableau qui servira à l'affichage des recettes. De ce tableau, les différentes listes sont mises à jour.	
Avantages - code plus robuste et plus stable - code plus court - version plus rapide	Inconvénients - code moins lisible
Saisie de 3 caractères minimum dans le champ de recherche principal	


Option 2 : Programmation native (annexe 1) Utilisation des boucles (while, for ...). Ici utilisation de « for » qui itère sur le tableau des recettes et cherche s'il existe une correspondance entre la saisie, et le nom ou la description ou un des ingrédients de la recette. Si oui, la recette en question est ajoutée à un nouveau tableau qui servira à l'affichage des recettes trouvées. De ce tableau, les différentes listes sont mises à jour également	
Avantages - code plus lisible, plus facile à comprendre	Inconvénients - code moins stable, plus long - version plus lente
Saisie de 3 caractères minimum dans le champ de recherche principal	

Solution retenue
<u>Tests effectués sur 4000 recettes :</u>
<u>Test avec JsBench.me :</u> - la programmation fonctionnelle semble être la plus rapide d'après le test Jsbench (Annexe 2) - la programmation native avec « for » est près de 60% plus lente qu'avec un filter.
<u>Tests avec console.time :</u> - programmation fonctionnelle avec filter : 0,003441 ms (en moyenne) - programmation native avec for : 0,005028 ms (en moyenne)
Notre choix se porte donc sur l'option 1, la programmation fonctionnelle avec Filter

**Annexe 1 :
programmation native**



ANNEXE 2- TESTS JS BEN

 JSBench.Me

[Run](#) [Please login/register to save & publish tests](#) [Sponsor](#) [Settings](#) [Sign in](#)

test algorithme de recherche 4000 recettes v1

enter test suite description

Setup HTML	<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8" /> <meta http-equiv="X-UA-Compatible" content="IE=edge" /> <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" /> <!--Google Fonts--> <link rel="preconnect" href="https://fonts.googleapis.com/" /></pre>
Setup JavaScript	<pre>/* eslint-disable no-undef */ /**-----DOM-----*/ const recipesContainer = document.querySelector("#output"); const tags = document.querySelector(".selectedTag"); //inputs const ingredientFilter = document.querySelector("#ingredients-filter"); const applianceFilter = document.querySelector("#appliance-filter"); const ustensilFilter = document.querySelector("#ustensils-filter"); const principalSearch = document.querySelector("#recherche"); //chevrons</pre>
algo1 - filter	<pre>if (inputValue.length > 2) { let recipesChoice = []; recipesArray.filter((recipe) => { if (recipe.name.toLowerCase().replace(/\s/g, "").includes(inputValue) recipe.description .toLowerCase() .replace(/\s/g, "") .includes(inputValue) </pre>
finished	
1446.61 ops/s ± 98.96%	
Fastest	
algo2 - for	<pre>const inputValue = e.target.value.toLowerCase().replace(/\s/g, ""); //console.log(inputValue); if (inputValue.length > 2) { let recipesChoice = []; for (let recipe of recipesArray) { for (let i = 0; i < recipe.ingredients.length; i++) { const ingredientName = recipe.ingredients[i].ingredient .toLowerCase() .replace(/\s/g, "");</pre>
finished	
591.55 ops/s ± 3.53%	
59.11 % slower	