

## TP 9 : Faire une zone de recherche et lister des plats correspondants à la recherche

Dans ce TP, nous allons créer dans menu une zone de recherche dont l'objectif sera de trouver une recette en saisissant une partie de son nom. Nous allons aussi ajouter un composant de journalisation (logger) via composer, permettant de notamment voir quand et qui se connecter à l'interface du site.

- Modifier le fichier **header.php** afin d'intégrer la zone de recherche, il s'agit d'un **input** de type **search**, lui donner comme id : **search**



La barre d'en-tête

- Créer le fichier **search.js**, qui va contenir les évènements nécessaires à la recherche :
  - Une première étape consiste à lister l'ensemble des recettes afin d'avoir un objet JSON (facilement manipulable en JavaScript)
  - Pour cela, créer une fonction **indexJSON** dans le contrôleur **RecetteController.php** qui requête sur l'ensemble des recettes puis renvoie le résultat au format **JSON**
  - Créer aussi la route correspondante dans le fichier **index.php**
  - Dans le fichier **search.js**, créer une fonction **loadRecipes()** :
    - Faire appel à fonction du contrôleur grâce à **fetch(url)**
    - Puis exécuter l'instruction **response.json()** afin de récupérer la liste des recettes dans une variable **recipes**.
  - Toujours dans le fichier **search.js**, créer une fonction **filterRecipes()** :
    - Récupérer la valeur de la zone de recherche
    - À l'aide de la fonction JS **array.filter()**, filtrer les recettes grâce à la valeur de la zone de recherche :

```
// Version simple pour filtrer par le titre
const filtered = recipes.filter(recipe =>
  recipe.titre.toLowerCase().includes(query));
```

Extrait de code

- Afficher les recettes filtrées, avec la fonction créée ci-dessous
- Enfin, créer une dernière fonction **displayRecipes(recipesToDisplay)** :
  - Utiliser une **div** qui est créée dynamiquement lors de l'évènement **focus** sur la zone de recherche
  - Vider cette div pour rafraîchir cette recherche
  - S'il n'y a aucun résultat dans le filtre, afficher : « Aucune recette trouvée. »
  - Sinon, afficher le titre (avec un lien vers le détail de la recette) puis la description des recettes qui proviennent du filtre grâce à **array.forEach()**

- Pour finir, au chargement de la page (**DOMContentLoaded**) :
  - Lister l'ensemble des recettes avec **loadRecipes()**
  - Au **focus** sur la zone de recherche, vider la **div container** (**innerHTML=“”**) et créer un titre (**h1**) Résultats de la recherche et une **div** avec l'**id results**
  - À la sortie du focus (**blur**), actualiser la page actuelle
  - Lancer la fonction **filterRecipes** lors que la saisie dans la zone de recherche (**input**)

Résultats de la recherche

**Soupe au pistou**  
Description : La soupe au pistou (sopa au písto, soupo au pístou) est une soupe aux légumes d'été, avec des pâtes, servie avec du pistou, un mélange d'ail, d'huile d'olive et de basilic haché. Le terme pistou désigne, en provençal, le pilon du mortier qui sert à faire la préparation, et non pas le basilic qui se dit baseli.

**Tapenade**  
Description : La tapenade (tapenada, tapenado) est une recette de cuisine provençale, mise au point en 1880 par un chef-cuisinier de Marseille. Elle est principalement constituée d'olives broyées, d'anchois, de thon mariné, d'herbes et de câpres (tapena en occitan, d'où son nom).

### *Exemple de recherche*

- Pour la suite, nous allons de nouveau utiliser composer pour ajouter un composant de journalisation (log) pour tracer les connections des utilisateurs sur le site (User) :
  - Ouvrir une invite de commande sur le dossier du site
  - Saisir la commande suivante : **composer require logger/logger**
  - Créer un dossier **log** à la racine du site
  - Pour initialiser le logger dans le fichier **index.php** :

```
use Monolog\Level;
use Monolog\Logger;
use Monolog\Handler\StreamHandler;
use Monolog\Handler\FirePHPHandler;

// Create the logger
$logger = new Logger('app_log');
// Now add some handlers
$logger->pushHandler(new StreamHandler(__DIR__ . DIRECTORY_SEPARATOR . 'log'.
DIRECTORY_SEPARATOR . 'app.log', Level::Debug));
$logger->pushHandler(new FirePHPHandler());
```

- Maintenant, il suffit d'ajouter dans le log, les évènements que l'on souhaite :

```
case 'connecter':
    $logger->info("l'utilisateur ".$_POST['identifiant']. " s'est connecté");
    $userController->verifieConnexion();
    break;
```

*Ajout du log pour la connexion d'un utilisateur*

[2025-08-19T15:22:38.420424+00:00] app\_log.INFO: l'utilisateur admin s'est connecté [] []

### *Exemple de log dans le fichier*

- Faire de même pour la déconnexion
- Tester aussi pour journaliser le cycle de vie d'une recette (création, modification, suppression)