Table des matières

I- Problématique

L'objectif est d'enrichir l'expérience utilisateur en intégrant des éléments dynamiques tels que l'affichage en temps réel de la météo à Metz sur la page d'accueil, nécessitant l'utilisation d'une API REST. De plus, le sprint 4 prévoit également l'implémentation d'une fonction permettant d'afficher sur demande la liste des recettes associées à chaque produit, avec l'utilisation d'Ajax.

I- Solutions proposées

1) Mission 1

a) Opérations proposées

- La clé d'API

Dans le cas présent, nous nous rendons sur https://openweathermap.org/ pour récupérer une API qui fournit des données météorologiques.

Il faut ensuite ajouter la clé dans le fichier '.env'

```
###> API OpenWeatherMap ###
URL_API_OWM="http://api.openweathermap.org/data/2.5/weather?q=metz&lang=fr&units=metric&APPID=a5df47a3ea365c442c1048b66d90b77a"
###< API OpenWeatherMap ###
```

- Gérer l'API

Etant donné que nous affichons la météo dans l'accueil, la gestion se fait dans le contrôleur de celui-ci. (se référer aux commentaires)

```php
#[Route('/accueil', name: 'app_accueil')]
public function index(CategorieRepository $cRepository, EvenementRepository $eRepository): Response
{

    // récupérer la variable d'environnement désignant l'URl de l'API
    $urlAPI = $_SERVER['URL_API_OWM'];
    // Initialiser une session CURL
    $clientURL = curl_init();
    // Récupérer le contenu de la page
    curl_setopt($clientURL, CURLOPT_RETURNTRANSFER, 1);
    // Transmettre l'URL
    curl_setopt($clientURL, CURLOPT_URL, $urlAPI);
    // Exécutez la requête HTTP
    $reponse = curl_exec($clientURL);
    // Fermer la session
    curl_close($clientURL);
    // Récupérer les données au format JSON
    $donneesTemps = json_decode($reponse);
    $dateJour = new \DateTime('now', new \DateTimeZone('Europe/Paris'));

    $lesCategories = $cRepository->findAll();

    $lesEvenementsPasses= $eRepository->findByEvenementsPasses(3);
    $lesEvenementsFutures = $eRepository->findByEvenementsFutures(3);

    return $this->render('accueil/index.html.twig', [
        'lesCategories' => $lesCategories,
        'donneesTemps' => $donneesTemps,
        'dateJour' => $dateJour,
        'lesEvenementsFutures' => $lesEvenementsFutures,
        'lesEvenementsPasses' => $lesEvenementsPasses,
    ]);
}
```

- Gérer la vue

```twig
<div class="conteneur-meteo">
    <h2>Le temps à
        {{ donneesTemps.name }}</h2>
    <div>Le
        {{ dateJour | date("d/m/Y à H:i") }}</div>
    <div class="temps-previsions">{{donneesTemps.weather[0].description | capitalize }}</div>
    <div class="temps-previsions">
        <img src="http://openweathermap.org/img/w/{{ donneesTemps.weather[0].icon }}.png" class="temps-icone"/>
        {{ donneesTemps.main.temp_max | number_format(2, ',') | trim("0", "right") | trim(',') }}&deg;C
        <span class="temperature">{{ donneesTemps.main.temp_max | number_format(2, ',') | trim("0", "right") | trim(',') }}&deg;C</span>
    </div>
    <div>Humidité :
        {{ donneesTemps.main.humidity }}%</div>
    <div>Vent :
        {{ donneesTemps.wind.speed }}km/h</div>
</div>
```

Il faut penser à adapter le style et remplacer le code PHP par du twig.

2) Mission 2

a) Opérations proposées

- Ajouter la fenêtre modale

Nous choisissons l'emplacement de la fenêtre.

```twig
<td>
    <a data-toggle="modal" href="#" onclick="afficherMesRecettes({{ produit.id }});" >{{ produit.getRecettes() | length }}</a>
</td>
```

Puis l'apparence de la fenêtre

```
<!-- Modal -->
<div class="modal fade" id="mesRecettesModal" role="dialog">
    <div
        class="modal-dialog modal-dialog-centered">

        <!-- Modal content-->
        <div class="modal-content contenu">
            <div class="modal-body" id="mesRecettesContenu">
                <p>Du texte dans la fenêtre modale.</p>
            </div>
            <div class="modal-footer justify-content-center">
                <button type="button" class="btn btn-sm btn-primary" data-dismiss="modal">Fermer</button>
            </div>
        </div>

    </div>
</div>
```

- Ajouter la requête Ajax

Nous mettons le script à la fin de la page.

Ce dernier concatène dans une chaîne de caractère les recettes nécessitant le produit passé en paramètre.

```
{% block javascripts %}
    <script src="{{ asset('assets/jquery/jquery-3.6.0.min.js') }}"></script>
    <script src="{{ asset('assets/lib/bootstrap-4.4.1-dist/js/bootstrap.min.js') }}"></script>
    <script>
        function afficherMesRecettes(idProduit) {
            $(document).ready(function() {
                $.ajax({
                    url: '{{ path('ajax_recettes_produit') }}',
                    type: "POST",
                    dataType: "json",
                    data: {
                        "idProduit": idProduit
                    },
                    async: true,
                    success: function (lesRecettes) {
                        var chaineHtml = "";
                        for (var i = 0; i < lesRecettes.length; i++) {
                            chaineHtml = chaineHtml + "<div>" + lesRecettes[i].nom + "</div>";
                        }
                        $('#mesRecettesContenu').html(chaineHtml);
                        $('#mesRecettesModal').modal('show');
                    },
                    error: function(jqXHR){         // Fonction à appeler si la requête échoue
                        $("#mesRecettesContenu").html('<div class="text-danger">Erreur n°128</div>');
                    }
                })
            });
        }
    </script>
{% endblock %}
```

- Ajouter la méthode dans le contrôleur

```php
#[Route('/produit/ajaxrecettesproduit', name: 'ajax_recettes_produit')]
public function ajaxRecettesProduit(Request $request, RecetteRepository $repository): Response
{
    // récupérer la valeur de idProduit envoyée
    $idProduit = $request->request->get('idProduit');
    // chercher les recettes correspondantes
    $lesRecettes = $repository->findNameByProduit($idProduit);
    // retourner une réponse encodée en JSON
    $response = new Response(json_encode($lesRecettes));
    $response->headers->set('Content-Type', 'application/json');

    return $response;
}
```

- Ajouter la méthode dans le repository

C’est ici que nous récupérons les données de la BDD.

```php
/**
 * @return string[]
 */
public function findNameByProduit($idProduit): array
{
    $entityManager = $this->getEntityManager();

    // ce n'est pas du SQL mais du DQL : Doctrine Query Language
    // il s'agit en fait d'une requête classique mais qui référence l'objet au lieu de la table
    $query = $entityManager->createQuery(
        'SELECT r.nom
        FROM App\Entity\Recette r
        JOIN r.produits p
        WHERE p.id = :idProduit
        ORDER BY r.nom ASC'
    )->setParameter('idProduit', $idProduit);

    return $query->getResult();
}
```

II- Conclusion

En résumé, le quatrième sprint de Synthèse Maya a démontré la pertinence de la solution adoptée pour enrichir l'expérience utilisateur. L'intégration dynamique de la météo à Metz via une API REST, avec une gestion soignée de la clé d'API et une présentation esthétique dans le contrôleur, témoigne d'une approche innovante.

L'ajout d'une fonction Ajax pour afficher les recettes associées à chaque produit, avec une fenêtre modale bien intégrée, montre une réponse agile aux besoins des utilisateurs. La requête Ajax, placée stratégiquement en fin de page, assure une interaction fluide avec le serveur.

Les méthodes ajoutées dans le contrôleur et le repository ont permis une gestion efficace des données, soulignant la cohérence de l'architecture. En somme, ce sprint a réussi à créer une expérience utilisateur plus riche et interactive, mettant en évidence la pertinence de la solution adoptée pour les besoins spécifiques de Synthèse Maya.