Rapport d’activité hebdomadaire

Banque Populaire Val de France

GRIMAUD William

BTS SIO

21/05/2024 au 26/06/2024

Tuteur : Thomas DORVAL

Sommaire

Table des matières

Présentation du contexte et des missions

Présentation de l'entreprise

La Banque Populaire Val de France (BPVF) est une banque coopérative détenue par ses sociétaires. C'est une banque à taille humaine, implantée sur un vaste territoire comptant 10 départements, des Yvelines à la Vienne en passant par la Région Centre. La BPVF fait partie du groupe BPCE qui est le deuxième groupe bancaire de France. Elle est aussi actionnaire de ce même groupe.

Présentation du service Études et Développements Informatiques

Le service Études et Développements Informatiques qui appartient à la direction IT et Data est composé de 7 personnes. Le service est divisé en 2 équipes. L'équipe de production qui va assurer le suivi de la production, faire de la maintenance corrective et évolutive des outils existants et de la gestion de projet à faible charge. L'équipe de projet va s'occuper de la gestion de projet et de la réalisation de nouveaux développements. Les deux équipes s'occupent également de faire évoluer les processus et les outils internes au service. Le service utilise principalement la technologie de développement ASP .NET CORE 6 (C#). Le service utilise aussi la méthodologie Agile et chaque matin, une réunion « Daily » va permettre à chacun de faire le point sur les travaux réalisés la veille, les difficultés rencontrées ainsi que ce qu'il a planifié pour la journée.

Présentation de l'environnement matériel et logiciel

Le service dispose de plusieurs outils logiciels :

- Microsoft Office 365 avec Outlook, Word, Excel, PowerPoint et OneNote.
- Microsoft Viva Engine qui est un réseau social d'entreprise.
- Microsoft SharePoint qui sert d'intranet et de plateforme de gestion de contenu.
- Cisco Jabber pour la messagerie instantanée.
- Microsoft Teams pour les réunions et la communication collaborative.
- Microsoft SQL Server Management Studio pour la gestion de base de données.
- Microsoft Visual Studio pour le développement des applications et des traitements.
- Visual TOM pour l'ordonnance de tâches.
- Teradata SQL Assistant qui permet de requêter sur les entrepôts et tables informationnelles.

Pour ce qui est de l'environnement matériel, la BPVF dispose de nombreux serveurs pour :

- La production Web IIS et les bases de données SQL
- Le développement IIS et SQL
- Les applications hors développement informatique
- La production VTOM (ordonnanceur)
- Les échanges
- L'impression

Les serveurs sont virtualisés et infogérés dans les datacenters du groupe.

Présentation des missions confiées

Amélioration d'une application existante interne au service. L'application est divisée en deux parties : une qui permet de savoir le nombre de connexions à toutes les applications au sein de la banque selon un intervalle de temps que l'on peut choisir, l'autre partie va permettre de connaître tous les événements (informations, avertissements, erreurs) survenus dans chaque application dans un tableau. J'ai dû améliorer l'ergonomie de l'application et ajouter des nouvelles fonctionnalités. L'application utilise ASP .NET CORE 6 en C# comme langage et Microsoft SQL Server comme SGBD. On m'a donné plusieurs missions à effectuer concernant cette application.

D'abord, ajouter une page permettant d'afficher les messages d'erreurs affichés dans un tableau en entier. Les messages d'erreurs récupérés étant très longs pour la plupart, ils avaient donc été tronqués. Le seul moyen de les voir en entier était de passer la souris dessus.

Ensuite, il m'a été demandé de résoudre un problème existant lié à la pagination et au changement de page sur une des pages de l'application car elle n'était pas fonctionnelle.

On m'a aussi demandé de fixer les colonnes du tableau mentionné précédemment car quand la souris passe sur un message du tableau, une icône qui permet de copier le message d'erreur apparaît et décale toutes les colonnes du tableau.

Enfin, j'ai dû créer une nouvelle page qui permet de savoir le nombre total d'erreurs identiques survenues sous forme d'un tableau en affichant le nombre total, le type et le nom de l'erreur/événement selon l'application choisie avec un filtre permettant d'afficher le compte selon une période choisie.

Gestion de projet

Je n'ai pas reçu de date butoir pour les missions qui m'ont été confiées et elles ne sont pas divisées en sprints. L'entreprise utilisait auparavant l'outil de gestion de projet Sciforma mais il n'est plus utilisé et il n'y a pas de remplacement à l'heure actuelle. Je rends compte de mon travail quotidien lors des réunion quotidiennes « daily » et à mon tuteur. Lorsqu'un projet est terminé, une revue de code est effectuée par un membre de l'équipe. Cette phase est appelée la phase de recette.

Rapports hebdomadaires

Semaine du 21 mai

Découverte du service et des activités réalisées. Explication du fonctionnement du service ainsi que les outils utilisés. Prise en main des outils utilisés par le service (Teams, Outlook…).

Explication du manifeste Agile et des principes et sous-principes associés. Même si elle n'est pas appliquée à la lettre, le service s'en inspire et l'adapte à chaque projet réalisé.

Initiation à ASP .NET CORE, Transact-SQL, Entity Framework Core et tutoriels Microsoft Learn.

Création d'interfaces utilisateur web avec ASP .NET CORE et récupération de données relationnelles avec Entity Framework Core (ORM).

Quelques problèmes liés au proxy de la banque résolus grâce à l'équipe.

Participation aux réunions quotidiennes le matin et à une réunion redescente d'information avec le service.

La réunion redescente d'information est une réunion hebdomadaire effectuée tous les mercredis matin par le directeur de la section IT et Data. Pendant celle-ci, il transmet diverses information reçues la semaine passée à tous les membres de la direction.

Semaine du 27 mai

Tour du service et de chaque personne présente. Explication des tâches individuelles effectuées et explication du fonctionnement de certains outils utilisés. (VTOM, Azure DevOps, Serveurs SQL…)

Affectation de tâches à effectuer sur une application existante. Améliorer son ergonomie et corriger des problèmes existants. Afficher les messages d'événements en entier, corriger le problème lié à la pagination et fixer les colonnes du tableau.

J'ai pris en main l'application et son code.

Explication des méthodes utilisées pour l'affichage de certaines pages, notamment avec les vues partielles ainsi que les événements javascript utilisés pour récupérer les données nécessaires à l'affichage des pages.

J'ai réalisé une page pour pouvoir afficher les messages d'erreurs complets sous forme de tableau avec le nom de l'erreur et son message associé en utilisant des classes Bootstrap et les templates existantes de la banque.

NOM	MESSAGE
UsingInMemoryRepository	Using an in-memory repository. Keys will not be persisted to storage.

J'ai créé des classes et des méthodes nécessaires à l'affichage de la page ainsi que le script javascript pour récupérer les données en utilisant JQuery.

```
[HttpGet]
0 références | 0 modification | 0 auteur, 0 modification
public async Task<IActionResult> GetEventInfoFullAsync(int id)
{
    ApplicationDto model = await _eventService.GetApplicationDtoByIdAsync(id);
    return PartialView("_InfoMessage", model);
}
```

```
1 référence
function viewEventMessage(id) {
    showModal('bpce-toaster');
    $.ajax({
        url: '/BpvfLogger/LogApp/GetEventInfoFull',
        type: 'GET',
        data: { id: id },
        traditional: true,
        0 références
        success: function (data, status, xhr) {
            hideModal('bpce-toaster');
            $('#modalCommonContent').html(data);
            showModal('modalCommon');
        },
        0 références
        error: function (xhr, status, error) {
            alert($(xhr.responseText).last()[0].textContent);
            hideModal('bpce-toaster');
        }
    });
};
```

J'ai ajouté des icônes pour accéder à la page quand on passe la souris sur le message d'erreur/avertissement à l'aide d'un événement html *onclick* et d'une classe CSS.


```
.view-event-message:hover i {
    display: inline-block !important;
}
```

```
<td class="copy-to-clipboard view-event-message cursor-pointer" title="@app.Message">
    <i class="fal fa-eye mr-2" style="font-size: 18px; display:none;" onclick="viewEventMessage(@app.Id)"></i>
    @message
</td>
```

J'ai rencontré un problème avec Visual Studio car j'ai lancé le dossier du projet au lieu de lancer la solution.

Participation aux réunions quotidiennes le matin, à la réunion redescente d'information hebdomadaire ainsi qu'une réunion de partage.

La réunion de partage est une réunion qui a lieu toutes les deux semaines. Une ou plusieurs personnes peuvent s'inscrire pour faire une présentation sur divers sujets au personnel de la banque.

Semaine du 3 juin

Résolution du problème lié à la pagination. Ce problème était dû au fait que le compte total des pages ne marchait pas et donc qu'il y avait détection que d'une seule page ce qui désactivait le bouton pour passer à la page suivante. J'ai donc ajouté une ligne permettant ce compte.

Avant :

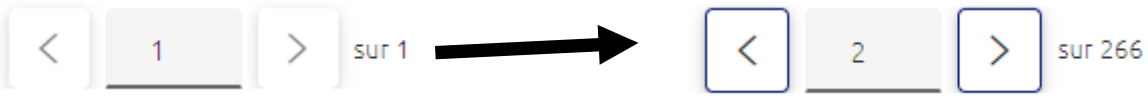
```
public async Task<PaginatedList<VEventDto>> GetCurrentsEventsAsync(int pageNumber, int pageSize, Filtre filtre)
{
    List<LogAppUnique> result = await _logAppUniqueRepository.ListAsync(new GetVEventSpecification(pageNumber, pageSize, filtre));

    List<VEventDto> listEvent = _mapper.Map<List<VEventDto>>(result);
    return listEvent.PaginatedListAsync(result.Count, pageNumber, pageSize);
```

Après :

```
public async Task<PaginatedList<VEventDto>> GetCurrentsEventsAsync(int pageNumber, int pageSize, Filtre filtre)
{
    List<LogAppUnique> result = await _logAppUniqueRepository.ListAsync(new GetVEventSpecification(pageNumber, pageSize, filtre));
    int count = await _logAppUniqueRepository.CountAsync(new GetVEventSpecification(pageNumber, pageSize, filtre));
    List<VEventDto> listEvent = _mapper.Map<List<VEventDto>>(result);
    return listEvent.PaginatedListAsync(count, pageNumber, pageSize);
```

Résultat sur la page :



J'ai également implémenté un événement JQuery qui permet de saisir un chiffre pour pouvoir directement accèder à la page voulue sans cliquer à répétition sur le bouton de page suivante.

```
0 références
$(document).on("change", ".page", function () {
    $("#PageNumber").val($(this).val());
    let _data = getFormData("formFilitre");
    search(_data, $("#formFilitre").data("type"));
});
```

Ensuite, j'ai fixé les lignes du tableau des erreurs pour qu'elles ne bougent plus quand les icônes pour afficher le message et le copier apparaissent à l'aide d'une classe CSS.

```
.fixed {
    table-layout: fixed;
}
```

```
<table class="table align-items-center mb-0 table-hover table-striped fixed" id="tableApplication">
```

J'ai eu quelques problèmes sur la page qui affiche les messages d'événements en entier car j'utilisais le nom de l'évènement pour pouvoir afficher le message correspondant mais je me suis rendu compte que certains messages ne possédaient pas de nom d'évènement et donc que la page ne s'affichait pas. Pour régler ce problème, un des membres de l'équipe m'a suggéré d'utiliser l'id de l'évènement plutôt que son nom. C'est donc ce que j'ai fait et cela a réglé ce problème (corrections visibles sur les captures d'écran de la semaine précédente).

Enfin, j'ai commencé à créer la page qui affiche le total d'évènements similaires ainsi que les classes et méthodes nécessaires.

Comme chaque semaine, participation aux réunions quotidiennes le matin et à la réunion redescente d'information hebdomadaire.

Semaine du 10 juin

J'ai créé la page qui permet d'afficher les messages d'évènement uniques. J'ai utilisé le layout de la banque déjà existant sur toutes les pages de l'application. C'est lui qui permet de changer de page entre les deux fonctionnalités de l'application ainsi que d'afficher l'utilisateur connecté.

J'ai ensuite ajouté le tableau d'affichage des erreurs.

J'ai créé une nouvelle spécification qui va requêter sur la base de données avec Entity Framework pour récupérer les informations nécessaires pour le tableau ainsi que pour l'affichage de la page.

```
2 références | 0 modification | 0 auteur, 0 modification
public class GetApplicationEventUniqueSpecification : Specification<Event, Event>
{
    1 référence | 0 modification | 0 auteur, 0 modification
    public GetApplicationEventUniqueSpecification(string app)
    {
        Query
            .Where(c => c.Application == app);

        Query
            .Select(c => new Event { Application = c.Application, LogLevel = c.LogLevel, EventName = c.EventName });
    }
}
```

Ces requêtes vont permettre de récupérer les messages d'événement de l'application sélectionnés dans la page précédente.

Ensuite, j'ai ajouté une méthode qui récupère ces événements sous forme de liste.

```
2 références | 0 modification | 0 auteur, 0 modification
public async Task<IEnumerable<ApplicationDto>> GetApplicationUniqueEventListAsync(string app)
{
    IEnumerable<Event> events = await _eventRepository.ListAsync(new GetApplicationEventUniqueSpecification(app));
    return _mapper.Map<IEnumerable<ApplicationDto>>(events);
}
```

Dans le contrôleur, j'ai également ajouté une méthode qui va permettre d'afficher la page avec les paramètres récupérés précédemment.

```
[HttpGet]
0 références | 0 modification | 0 auteur, 0 modification
public async Task<IActionResult> ApplicationLogsUniqueAsync(string app)
{
    return View(await _eventService.GetApplicationUniqueEventListAsync(app));
}
```

Enfin, j'ai implémenté un événement JQuery qui va afficher le nom de l'application sélectionnée dans l'url de la page quand on clique sur l'icône pour afficher la page.

https://localhost:44370/BpvfLogger/LogApp/ApplicationLogsUnique?app=AnimationCommerciale

```
0 références
$(document).on("click", ".logsUnique", function () {
    let appName = $(this).data("id");
    window.location.href = "/BpvfLogger/LogApp/ApplicationLogsUnique?app=" + appName;
});
```

J'ai ensuite créé un dictionnaire avec une liste dans la vue du tableau. J'ai effectué une requête GroupBy sur le nom de l'événement. C'est ce qui va me permettre de compter le nombre d'événements ayant le même nom. J'ai aussi ajouté un ToDictionnary pour mettre les clés dans une liste.

```
@{
    Dictionary
    <string, List<ApplicationDto>> dic = Model
    .GroupBy(c => c.EventName)
    .ToDictionary(g => g.Key, g => g.ToList())
    .OrderByDescending(x => x.Value.Count)
    .ToDictionary(x => x.Key, x => x.Value);
}
```

J'ai fait un foreach sur les valeurs et clés du dictionnaire pour pouvoir afficher les informations demandées dans le tableau ainsi que le compte des erreurs.

```
@foreach (KeyValuePair<string, List<ApplicationDto>> app in dic)
{
    <tr>
        <td>
            @if (app.Value.FirstOrDefault().LogLevel == EnumExtensions.GetDisplayName(LogLevels.Information))
            {
                <span class="badge rounded-pill bg-info">@app.Value.FirstOrDefault().LogLevel</span>
            }
            else if (app.Value.FirstOrDefault().LogLevel == EnumExtensions.GetDisplayName(LogLevels.Warning))
            {
                <span class="badge rounded-pill bg-warning">@app.Value.FirstOrDefault().LogLevel</span>
            }
            else if (app.Value.FirstOrDefault().LogLevel == EnumExtensions.GetDisplayName(LogLevels.Error))
            {
                <span class="badge rounded-pill bg-danger">@app.Value.FirstOrDefault().LogLevel</span>
            }
            else
            {
                <span>Événement introuvable </span>
            }
        </td>
        <td title="@app.Key">
            @if (app.Key == "")
            {
                <span>Aucun nom d'événement</span>
            }
            else
            {
                <span>@app.Key</span>
            }
        </td>
        <td>@app.Value.Count</td>
    </tr>
```

Après avoir terminé la page, on m'a demandé si c'était possible de trier le tableau sur le nombre d'événements pour avoir l'événement avec le plus d'occurrences en haut du tableau.

On m'a aussi demandé d'ajouter un message pour les événements qui ne possédaient pas de nom. Enfin, on m'a demandé d'ajouter un moyen d'afficher le nom de l'application choisie dans la page plutôt que dans l'url.

Pour le tri par occurrences, j'ai ajouté un OrderBy dans le dictionnaire (Voir précédente capture).

Pour les événements sans nom, j'ai ajouté une condition if pour afficher un message par défaut (Voir précédente capture).

Enfin, j'ai ajouté un div au-dessus du tableau pour pouvoir ajouter le nom de l'application sélectionnée. J'en ai profité pour également apporter cette modification sur la page du tableau qui affiche toutes les erreurs d'une application.

```
<div class="card h-100" style="font-size: 15px; font-weight: bold;">
    <div class="card-body pb-0 p-3">
        <div class="d-flex mb-2">
            <span class="me-2 text-sm text-capitalize">Application :</span>
            <span class="me-2 text-sm text-capitalize">@Model.FirstOrDefault().Application</span>
        </div>
    </div>
</div>
```

Application : AnimationCommerciale

J'ai eu un certain nombre de problèmes en réalisant cette page et le compte des événements.

D'abord, j'ai utilisé une méthode ListDistinct dans ma méthode (deuxième capture). Cette méthode récupérait bien les messages de manière unique, mais quand le compte était effectué, il n'y avait qu'une seule occurrence pour tous les messages d'événements à cause du distinct. J'ai donc changé ma méthode pour une méthode asynchrone ListAsync à la place de ListDistinct.

J'ai eu un autre problème car j'ai essayé de faire un compte dans le tableau, mais j'ai eu du mal et on m'a suggéré de créer un dictionnaire avec une liste puis d'effectuer une requête GroupBy pour pouvoir faire le compte.

Ensuite, vu que je ne faisais plus de boucle sur le Model de la page mais sur le dictionnaire, je n'arrivais plus à afficher les types d'événements. J'ai demandé de l'aide à un membre du service et il m'a conseillé d'ajouter la méthode FirstOrDefault ce qui m'a permis de pouvoir récupérer les types d'erreurs ainsi que les afficher. J'ai également utilisé cette méthode pour afficher le nom de l'application.

Enfin, j'ai eu un problème de conversion quand j'ai ajouté l'instruction OrderBy dans mon dictionnaire. J'ai effectué une recherche sur internet avant de tomber sur la solution utilisée dans la capture d'écran. J'ai dû trier sur un compte des valeurs du dictionnaire et ajouter une instruction ToDictionary après.

Comme chaque semaine, participation aux réunions quotidiennes le matin, à la réunion redescente d'information hebdomadaire et à la réunion de partage bimensuelle.

Semaine du 17 juin

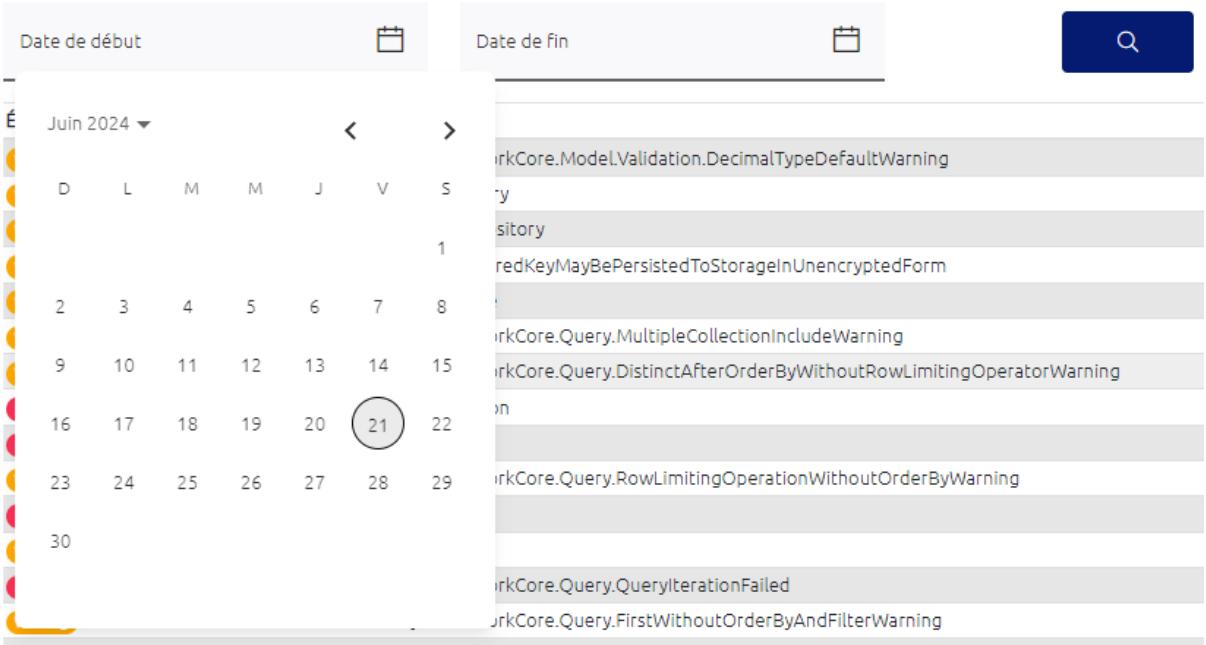
J'ai commencé par changer le message affiché lorsqu'un événement ne porte pas de nom. J'ai remplacé l'ancien message par le nom du namespace à qui il appartient pour améliorer la clarté.

```
<td>
    @if (app.Key == "")
    {
        <span title="@app.Value.FirstOrDefault().Name">@app.Value.FirstOrDefault().Name</span>
    }
    else
    {
        <span title="@app.Key">@app.Key</span>
    }
</td>
```

Microsoft.AspNetCore.DataProtection.Repositories.Ep...

Comme j'avais terminé de créer la page avec le compte des événements, on m'a demandé d'ajouter un filtre sur cette page pour pouvoir afficher le nombre total d'événements sur une période choisie.

J'ai d'abord ajouté les champs sous forme d'un calendrier. Ce filtre était déjà présent sur d'autres pages de l'application donc j'ai juste repris ce qui avait déjà été fait.



Puis, je me suis aperçu que le filtre avait besoin de la classe Filtre pour pouvoir fonctionner. C'est un problème car le modèle de la page était ApplicationDto or cette classe ne comporte pas de propriété Filtre. J'ai donc fait comme sur les autres pages de l'application. J'ai créé une

vue avec le modèle ApplicationViewModel qui comporte la propriété Filtre. J'y ai ajouté les champs de date.

Ensuite, pour le tableau des événements, je l'ai mis dans une vue partielle (avec le modèle ApplicationDto) que j'appelle dans la vue avec le filtre et le layout de la page.

```
<div id="tableLogUnique" class="mt-3">
    @await Html.PartialAsync("_TableLogUnique", Model.ListEvents)
</div>
```

J'ai rencontré un autre problème lors de cet ajout car ListEvents appartient à la classe PaginatedList or ma vue avec le tableau était de type IEnumerable. J'ai dû remplacer tous les types IEnumerable de mes méthodes avec PaginatedList et en y ajoutant le filtre pour pouvoir afficher mon tableau.

```
3 références | 0 modification | 0 auteur, 0 modification
public async Task<PaginatedList<ApplicationDto>> GetApplicationUniqueEventListAsync(Filtre filtre, string app)
{
    List<Event> events = await _eventRepository.ListAsync(new GetApplicationEventUniqueSpecification(app, filtre));
    List<ApplicationDto> listEventUnique = _mapper.Map<List<ApplicationDto>>(events);
    return listEventUnique.PaginatedListAsync(0, 0, 0);
}
```

```
[HttpGet]
0 références | 0 modification | 0 auteur, 0 modification
public async Task<IActionResult> SearchFilterByDateAsync(Filtre filtre)
{
    PaginatedList<ApplicationDto> result = await _eventService.GetApplicationUniqueEventListAsync(filtre, filtre.Application);

    return PartialView("_TableLogUnique", result);
}

[HttpGet]
```

Pour le contrôleur, j'ai dû changer ma méthode qui affiche la vue avec le filtre.

```
[HttpGet]
0 références | 0 modification | 0 auteur, 0 modification
public async Task<IActionResult> ApplicationLogsUniqueAsync(string app)
{
    ApplicationViewModel model = new()
    {
        Application = app
    };

    model.ListEvents = await _eventService.GetApplicationUniqueEventListAsync(model.Filtre, app);
    return View(model);
}
```

J'ai créé la méthode SearchFilterByDateAsync qui va afficher le tableau en fonction de la période choisie. Cette méthode est utilisée dans une fonction JQuery qui va être appelée dans un événement lorsqu'on valide le filtre.

```
1 référence
function searchDate(_data) {
    showModal('bpce-toaster');
    $.ajax({
        url: '/BpvfLogger/LogApp/SearchFilterByDate',
        type: 'GET',
        data: _data,
        traditional: true,
        0 références
        success: function (data, status, xhr) {
            hideModal('bpce-toaster');
            $('#tableLogUnique').html(data);
        },
        0 références
        error: function (xhr, status, error) {
            alert($(xhr.responseText).last()[0].textContent);
            hideModal('bpce-toaster');
        }
    })
}
```

```
0 références
$(document).on("submit", "#formLogUnique", function (e) {
    e.preventDefault();
    let _data = getFormData("formLogUnique");
    searchDate(_data, $("#formLogUnique").data("type"));
});
```

Enfin, j'ai implémenté les requêtes nécessaires pour récupérer la date dans la spécification qui me permet de récupérer les événements.

```
if (!string.IsNullOrEmpty(filtre.StartDate) && !string.IsNullOrEmpty(filtre.EndDate))
{
    DateTime du = Convert.ToDateTime(filtre.StartDate);
    DateTime au = Convert.ToDateTime(filtre.EndDate);
    Query
        .Where(c => c.Created >= du && c.Created <= au);
}

if (!string.IsNullOrEmpty(filtre.StartDate) && string.IsNullOrEmpty(filtre.EndDate))
{
    DateTime du = Convert.ToDateTime(filtre.StartDate);
    Query
       .Where(c => c.Created >= du);
}

if (string.IsNullOrEmpty(filtre.StartDate) && !string.IsNullOrEmpty(filtre.EndDate))
{
    DateTime au = Convert.ToDateTime(filtre.EndDate);
    Query
        .Where(c => c.Created <= au);
}
```

J'ai eu des problèmes lors du changement de IEnumerable vers PaginatedList car la méthode PaginatedListAsync (voir capture de la méthode GetApplicationUniqueEventListAsync) a besoin de numéros de page et d'une taille de page en paramètres. J'ai d'abord essayé d'ajouter une pagination sur mon tableau avec tous les paramètres nécessaires mais une ligne dans ma spécification rendait le compte des événements non fonctionnel.

```
.Take(pageSize);
```

C'est cette ligne qui faisait que mon compte n'affichait plus que, par exemple, 12 occurrences alors qu'il y en avait 7000 avant les changements. J'ai donc demandé de l'aide à un membre du service et nous n'avons trouvé qu'une seule solution. Nous avons supprimé cette ligne ainsi que tous les paramètres de pagination. Pour faire fonctionner la méthode PaginatedListAsync sans ces paramètres, nous avons remplacé les paramètres par des 0. Cette méthode n'est absolument pas idéale mais c'est la seule solution que nous ayons trouvée pour faire fonctionner cette méthode sans ces paramètres. Il fallait utiliser cette méthode car elle était nécessaire pour faire fonctionner le filtre sachant que toutes nos méthodes étaient de type PaginatedList. À la suite de ces changements, la page avec le filtre marche correctement sans problèmes.

Après avoir terminé la page, j'ai changé l'icône permettant d'y accéder à la demande d'un membre de l'équipe et j'ai ajouté un texte quand on passe la souris dessus pour savoir ce que fait l'icône.

Comme chaque semaine, participation aux réunions quotidiennes le matin et à la réunion redescente d'information hebdomadaire.

24 au 26 juin

Après avoir terminé d'ajouter le filtre, j'ai créé une branche avec mon code pour pouvoir la fusionner à la branche principale *master* et j'ai enregistré mes modifications dans Visual Studio, ce qui permettra d'afficher que ces modifications ont été réalisées par moi. J'ai ensuite créé une pull request (demande de tirage) sur Azure DevOps pour qu'un membre de l'équipe fasse une revue de mon code.

Je n'ai pas eu de problèmes pendant la revue, on m'a dit qu'il n'y avait pas de problème avec mon code.

Mes modifications ont ensuite été publiées sur l'application en production.

Enfin, j'ai participé aux réunions quotidiennes et à ma dernière redescente d'information.

Bilan de stage

Mesure de la réussite du projet

J'ai réussi toutes les tâches qui m'ont été données pendant ce stage de 5 semaines.

J'ai ajouté une page pour afficher les messages d'événements en entier, j'ai réglé le problème lié à la pagination, j'ai fixé les colonnes du tableau et j'ai ajouté une page pour afficher le total d'erreurs différentes avec un filtre pour pouvoir saisir une période.

Tout ce qui a été prévu a été réalisé, on m'a donné une nouvelle tâche dès que j'avais fini la tâche précédente qui m'avait été donnée.

Ce que m'a apporté le stage

Ce stage m'a permis de découvrir le Framework ASP .NET CORE et j'ai appris à faire de la programmation web orientée objet en C#. J'ai appris à requêter sur une base de données en utilisant l'ORM EntityFramework et j'ai formulé des requêtes dans mon code pour aller chercher directement des données dans la base de données. J'ai aussi consolidé mes connaissances en travaillant dans une architecture MVC et en utilisant JQuery et Bootstrap.

J'ai appris à utiliser Azure DevOps comme gestionnaire de code source pour pouvoir récupérer le code source de l'application à améliorer, créer des branches et pour formuler des demandes de tirages pour pouvoir faire revoir mon code et publier les modifications apportées.

Enfin, j'ai appris à m'intégrer et travailler au sein d'une équipe en participant à des réunions quotidiennes et en sollicitant des membres de l'équipe en cas de besoin. J'ai également appris à travailler dans un cadre professionnel au sein d'une entreprise et dans un cadre réglementé en étant vigilant à toute tentative de phishing et aux données sensibles.

Conclusion

Je suis satisfait de ce stage et j'ai appris de nombreuses choses pendant celui-ci. J'ai pu mettre en pratique ce que j'ai appris pendant l'année. J'ai pu acquérir de nouvelles compétences et consolider mes acquis. Enfin, j'ai pu davantage découvrir le travail en entreprise.

Page de remerciements

Je remercie le service Études et Développements Informatiques de m'avoir accueilli et accompagné pendant ces 5 semaines de stage.

Je remercie Thomas DORVAL de m'avoir accepté et accueilli en stage dans son service et de m'avoir accompagné durant mon stage.

Je remercie Thierry SALLIER de m'avoir accepté au sein de la direction IT&Data.

Enfin, je remercie la Banque Populaire Val de France de m'avoir ouvert ses portes pour mon stage.