Rapport de Stage
ARCNET NC

Module de Gestion de documents

Stagiaire :
BEROULE Luca
L3 MIAGE TREC 7,
Université de la Nouvelle-Calédonie

Tuteur de stage :
Erwan FAURE

Table des matières

1. Introduction

Ce rapport de stage décrit les activités effectués lors de mon stage chez ARCNET NC qui s'est déroulé du 02 octobre 2023 au 31 janvier 2024.

1.1 : Présentation de l'entreprise

ARCNET.NC est une société de services spécialisée dans le domaine de l'ingénierie informatique tel que le développement informatique, les bases de données, la création et le référencement de sites internet et les formations professionnelles. Elle dispose d'une solide expérience dans les technologies Microsoft et Oracle grâce à une équipe de spécialistes certifiés.

ARCNET est une TPE (Très petite entreprise) composé de 6 employés

Développeurs : Nicolas SAVENAY, Jonathan CAILLART, Bastien BOUCHETIERE
Chef d'équipe / Développeur : Erwan FAURE
Gérants : Davy GRIGRI et Sophie PEZERON

1.2 : Objectifs et Missions du stage

J'ai travaillé sur le projet CDE (Caisse des Ecoles)
L'objectif principal de ce stage était de créer un système de gestion de documents qui pourra gérer les documents du module Petit Matériel. Permettre la création d'un document, par exemple un devis ou une facture pour acheter ou faire louer du matériel via les articles présents dans les stocks.

Objectifs :

- Permettre de créer un document en renseignant les informations voulues
- Faire en sorte qu'un article soit sélectionnable dans un document (Ligne du document)
- Effectuer une gestion des stocks
- Respecter la structure et la charte graphique de l'application

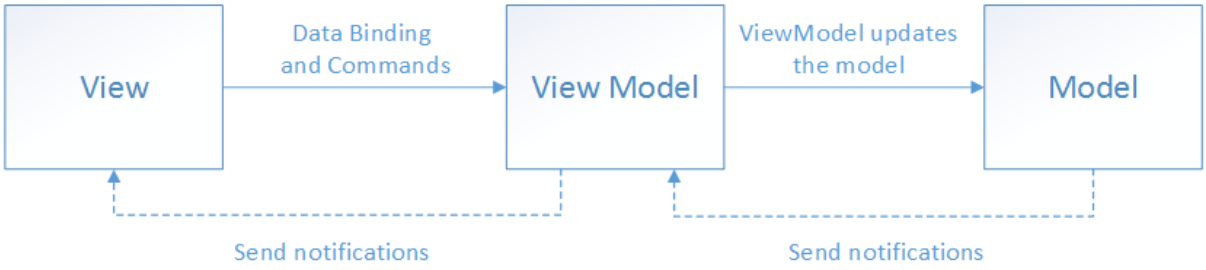
2. Présentation de l'application

2.1 : Architecture et technologies de l'application

2.1.1 : Architecture de l'application

L'architecture principale de cette application est en MVVM (Model-View-View-Model)
Le MVVM est une variante du MVC (Model View Controller) :
Model = Structure des données, classe C# transformé en table de BDD grâce à EntityFramework

View = Vue : chargée de définir la structure, la disposition et l'apparence de ce que l'utilisateur voit à l'écran
ViewModel (Variante MVVM) = Le ViewModel est chargé de transformer et organiser les Models afin d'exposer les données à afficher par la View.
Si un changement critique intervient au niveau du modèle métier, le ViewModel peut être adapté sans en impacter la View.
Controller = reçoit les interactions de l'utilisateur de la vue et les envoient au repository qui traitera les données et mettra à jour les Models. Le code à l'intérieur du contrôleur correspond essentiellement à du code de liaison (glue code) entre la vue et le modèle.
Repository : gère l'accès aux données, leur stockage et récupération.
Il existe trois composants principaux dans le modèle MVVM : le modèle, la vue et le modèle de vue. Chacun sert un objectif distinct. Le diagramme ci-dessous montre les relations entre les trois composants.



Le MVVM est adapté à de nombreux cas d'utilisation dans le développement d'applications web. Par exemple, il est idéal pour la création d'interfaces utilisateur réactives, la gestion de formulaires complexes avec des validations dynamiques, la mise en œuvre de flux de données en temps réel, ou encore la création d'applications à pages multiples avec une navigation fluide.

Comparé au modèle MVC traditionnel, le MVVM offre plusieurs avantages. Il améliore la séparation des préoccupations en isolant davantage la logique métier dans le modèle de vue. Le modèle de vue ne contient aucune référence directe à la vue. Au lieu de cela, il communique avec la vue via la liaison de données.
Ce qui permet aux concepteurs d'ajouter ou de remplacer facilement de nouveaux morceaux de code, de créer des tests unitaires plus faciles et donc qui facilite globalement la maintenance et l'évolutivité de l'application.

2.1.2 : FrontEnd

Le FrontEnd est développé en ASP.Net Razor Pages (HTML – CSS – JS + C# .NetCore) / Telerik UI for ASP.NET MVC (Basé sur MVC)

Razor a été conçu pour faciliter la conception des pages ASP.NET. Il introduit une syntaxe de programmation assez compréhensible, qui permet d'insérer du code serveur dans une page Web qui peut également contenir du HTML, du CSS et des scripts JavaScript.
Le moteur Razor est fluide, compact, expressif et s'appuie sur la syntaxe des langages .NET C# et Visual Basic .NET. Il offre la puissance d'ASP.NET pour la création rapide des applications Web fonctionnelles et sophistiquées.

Telerik UI for ASP.NET MVC est une suite de composants d'interface utilisateur (UI) destinée au développement web avec le Framework ASP.NET MVC. Telerik, une société spécialisée dans les outils de développement logiciel, propose cette bibliothèque pour simplifier la création d'interfaces utilisateur riches et interactives dans les applications web ASP.NET MVC.

La suite Telerik UI for ASP.NET MVC offre un ensemble de composants prêts à l'emploi, tels que des grilles de données, des graphiques, des calendriers, des formulaires, des boutons, etc.

Ces composants facilitent le développement en permettant aux développeurs de créer des interfaces utilisateurs attrayantes sans avoir à écrire tout le code HTML, CSS et JavaScript manuellement. Les composants sont hautement personnalisables, offrant une flexibilité pour répondre aux besoins spécifiques de l'application.

Exemple d'un composant Kendo : DropDownList

```
<div class="col-6 col-sm-6 form-group">
    <label asp-for="FournisseurId"></label>
    <span class="text-danger" asp-validation-for="FournisseurId"></span>
    @(Html.Kendo().DropDownListFor(m => m.FournisseurId)
        .OptionLabel("Sélectionnez un fournisseur")
        .HtmlAttributes(new { style = "width:100%;" })
        .DataTextField("Libelle")
        .DataValueField("Id")
        .Filter(FilterType.StartsWith)
        .DataSource(dataSource => dataSource
        .Custom()
        .Batch(true)
        .Transport(transport =>
        {
            transport.Read(read => read
            .Action("GetList", "Fournisseur")
            .DataType("json")
            .Type(HttpVerbs.Get)
            );
        })
        .AutoSync(true)
        )
        )
</div>
```

2.1.3 : BackEnd

Le BackEnd est développé en C# .NetCore.
C# est un langage de programmation moderne, orienté objet et de type sécurisé. C# permet aux développeurs de créer de nombreux types d'applications sécurisées et robustes qui s'exécutent dans .NET. C# prend sa source dans la famille de langages C et sera immédiatement reconnaissable aux programmeurs en C, C++, Java et JavaScript.

Plusieurs fonctionnalités C# permettent de créer des applications robustes et durables. Garbage Collection (Collector) récupère automatiquement la mémoire occupée par des objets inutilisés inaccessibles. Les types nullables protègent contre les variables qui ne font pas référence à des objets alloués.

La gestion des exceptions offre une approche structurée et extensible de la détection et de la récupération des erreurs. Les expressions lambda prennent en charge les techniques de programmation fonctionnelle.

La syntaxe LINQ (Language Integrated Query) crée un modèle commun pour utiliser les données de n'importe quelle source. La prise en charge du langage pour les opérations asynchrones fournit une syntaxe pour créer des systèmes distribués.

.NET est une plateforme de développement, gratuite, multiplateforme et open source permettant de créer de nombreux types d'applications.
.NET repose sur un runtime hautes performances utilisé en production par de nombreuses applications à grande échelle.
Le backend de l'application dispose aussi de sa propre structure. Deux parties pour les controllers / Une partie API qui va gérer les requêtes http envoyées ->

Exemple :

```
/// <summary>
/// Requête API permettant de récupérer la liste de tous les documents enregistrés en BDD.
/// Requête HTTP de type GET seulement.
/// Requête Ajax seulement (header XMLHttpRequest).
/// </summary>
/// <returns>Le code HTTP de la réponse avec d'éventuels objets en retour.</returns>
//// GET api/Documents
// Si l'on souhaite uniquement des requêtes Ajax, on met ça : [AjaxOnly]
[AllowAnonymous]
[HttpGet("GetDocuments")]
0 références | Davy GRIGRI | 1 auteur, 1 modification
public async Task<IActionResult> GetAll()
{
    try
    {
        var documents = await _documentRepository.GetDocumentsAsync<DocumentViewModel>();

        if( documents == null )
        {
            return NotFound();
        }

        // On retourne la liste des documents avec un code 200
        return Ok( documents.OrderBy( x => x.NoDocument ) );
    }
    catch
    {
        return StatusCode( 500, "Une erreur est survenue lors de la récupération de la liste des documents." );
    }
}
```

Et une autre partie qui va gérer la redirection vers les vues :

```
/// <summary>
/// Affiche le formulaire pour choisir un nouveau type et domaine de document.
/// </summary>
/// <returns>
/// Une vue partielle incluant le modèle DocumentEditViewModel.
/// </returns>
[HttpGet( "ajouter" )]
0 références | Luca BEROULE, il y a 42 jours | 2 auteurs, 2 modifications
public IActionResult FormCreate() => PartialView( new DocumentEditViewModel() );


/// <summary>
/// Affiche le formulaire pour supprimer un document.
/// </summary>
/// <returns>
/// Une vue partielle.
/// </returns>
[HttpGet( "FormDelete/{documentId}" )]
0 références | Luca BEROULE, il y a 42 jours | 1 auteur, 1 modification
public async Task<IActionResult> FormDelete(int documentId)
{
    if( documentId <= 0 )
    {
        return BadRequest( "L'ID fournit est hors limite" );
    }

    var document = await _documentRepository.GetDocumentAsync<DocumentDeleteViewModel>(documentId);

    if( document == null )
    {
        return NotFound();
    }

    // On retourne la vue partielle.
    return PartialView( document );
}
```

Des repositorys pour chaque modèle qui contiendront des méthodes de récupérations de données (Communique avec la base de données)

Chaque repository a son interface qui fait office de passerelle entre le repository et le controller :

Le repository est séparé du controller car le controller gère les interactions utilisateurs tandis que le repository lui gère l'accès aux données, leur stockage et récupération. De plus, séparer ses deux parties permet de faciliter la compréhension du code et sa maintenance sur le long terme.

```
public class DocumentController : Controller
{
    private readonly IDocumentRepository _documentRepository;
    private readonly IDocumentLigneRepository _documentLigneRepository;
    private readonly IReportingRepository _reportingRepository;

    /// <summary>
    /// Le constructeur du contrôleur.
    /// </summary>
    /// <param name="DocumentRepository"></param>
    0 références | 0 modifications | 0 auteurs, 0 modifications
    public DocumentController( IDocumentRepository documentRepository, IDocumentLigneRepository documentLigneRepository, IReportingRepository reportingRepository)
    {
        _documentRepository = documentRepository ?? throw new ArgumentNullException( nameof( documentRepository ) );
        _documentLigneRepository = documentLigneRepository ?? throw new ArgumentNullException( nameof( documentLigneRepository ));
        _reportingRepository = reportingRepository ?? throw new ArgumentNullException(nameof(reportingRepository));
    }
```

Si la référence de l'interface du repository n'est pas ajouté, alors les méthodes contenues dans celui-ci ne seront jamais accessibles.

Exemple si je veux rajouter une méthode qui supprime un document :

```
2 références
public async Task DeleteDocumentAsync( int id )
{
    var document = await _context.Documents
    .Include(d => d.DocumentLignes)
    .FirstOrDefaultAsync(d => d.DocumentId == id);

    if ( document == null )
    {
        throw new ArgumentNullException( nameof( document ) );
    }

    _context.DocumentsLignes.RemoveRange(document.DocumentLignes);
    _context.Documents.Remove( document );
    await _context.SaveChangesAsync();
}
```

```
public interface IDocumentRepository
{
    Get Document

    Get Documents

    #region Management

    2 références
    Task CreateDocumentAsync( DocumentEditViewModel Document );

    2 références
    Task UpdateDocumentAsync( DocumentEditViewModel Document );

    2 références
    Task DeleteDocumentAsync( int id );

    #endregion Management

    GetSouche
}
```

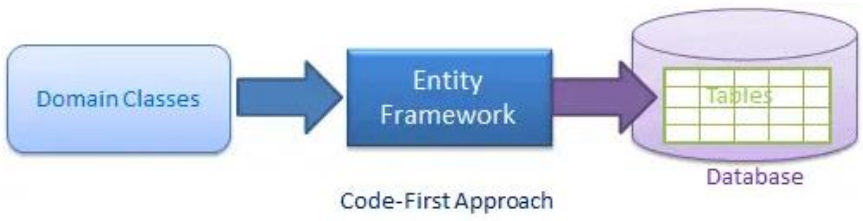
Il est important de rajouter une référence à la méthode DeleteDocumentAsync sans quoi la méthode ajoutée ne sera pas accessible dans le controller et entrainera une erreur.

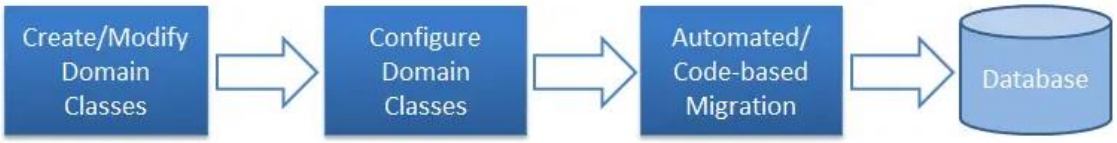
2.1.4 : Base de données

EntityFramework / SSMS (SQL SERVER MANAGEMENT STUDIO)

Entity Framework est un outil de mappage de relations objet moderne qui vous permet de créer une couche d'accès aux données propre, portable et de haut niveau avec . NET (C#) sur tout un éventail de bases de données, y compris SQL Database (local et Azure), SQLite, MySQL, PostgreSQL et Azure Cosmos DB.

Entity Framework adopte la méthode Code-First pour intégrer ses données :
On crée d'abord la classe en C# (le modèle) et Entity crée la table avec les données de ce modèle.





Microsoft SQL Server Management Studio est une application logicielle développée par Microsoft qui est utilisée pour configurer, gérer et administrer tous les composants de Microsoft SQL Server.

2.1.5 : Serveur de rapports et création de rapports

Microsoft Report Builder / SSRS (SQL Server Reporting Services)

Microsoft Report Builder est un outil de création de rapports disponible dans l'écosystème Microsoft SQL Server. Il permet aux utilisateurs de créer des rapports ad-hoc et des tableaux de bord interactifs en extrayant des données à partir de sources telles que des bases de données SQL Server, des sources de données OLE DB, des sources de données ODBC, etc.

SQL Server Reporting Services (SSRS) est une plateforme de génération de rapports développée par Microsoft. Elle est intégrée à l'écosystème Microsoft SQL Server et permet aux utilisateurs de créer, gérer et distribuer des rapports. SSRS offre des fonctionnalités avancées pour la création de rapports, la visualisation des données et la génération de tableaux de bord interactifs.

2.1.6 : Gestion de projets / Equipe / Environnements de travail

Azure DevOps prend en charge une culture collaborative et un ensemble de processus qui rassemblent les développeurs, les responsables de projets et les contributeurs pour développer des logiciels. Il permet aux organisations de créer et d'améliorer des produits à un rythme plus rapide qu'avec les approches traditionnelles de développement logiciel.

Microsoft Teams est une application de communication collaborative propriétaire en mode SaaS (Software as a Service)

L'IDE Visual Studio est un panneau de lancement créatif que vous pouvez utiliser pour modifier, déboguer et générer du code, puis publier une application. En plus de l'éditeur et du débogueur standard fournis par la plupart des IDE, Visual Studio inclut des compilateurs, des outils de complétion de code, des concepteurs graphiques et bien d'autres fonctionnalités pour améliorer le processus du développement de logiciels.

2.2 : A quoi sert l'application globalement ?

L'application s'appelle CDE (Caisse des Ecoles), elle permet de gérer la plupart des activités d'une école.
Elle contient les élèves inscrits dans un établissement ainsi que leurs responsables légaux, s'ils sont inscrits à la cantine et leurs cartes, les sorties scolaires annoncées, les transports, chauffeurs, et matériel disponible de l'école

3. Travail Effectué

3.1 : Problématique

Mettre en place un système de gestion des documents complète et cohérente et une gestion de stock intégré au module Petit Matériel du site CDE.

3.2 : Conception du module

3.2.1 : CRUD des documents

Afin de répondre aux attentes du modules. Il faut déjà pouvoir créer la base : C'est-à-dire un document. Ou l'utilisateur va renseigner le type de document, le domaine du document (Cantine, Etablissement, Cantine, Stock), Chaque domaine doit renseigner sa référence, par exemple si le domaine établissement est sélectionné, la dropdown qui contient tous les établissements est affiché et l'utilisateur doit sélectionner l'établissement et pareil pour les 2 autres domaines.

Donc étant donné que le modèle de document et son modèle de vue était déjà créé. J'ai simplement dû prendre connaissance de l'application qui a été ma principale source dans ce stage. J'ai donc pu reprendre et comprendre les méthodes CRUD déjà utilisées pour d'autres ViewModel et les recontextualiser pour qu'elles correspondent au contexte du document.

Exemple avec la méthode d'ajout d'un document :

```
/// <summary>
/// Requête API permettant de créer un nouveau document.
/// Requête HTTP de type POST seulement.
/// Requête Ajax seulement (header XMLHttpRequest).
/// </summary>
/// <param name="model">Le modèle incluant les données du nouveau document à créer.</param>
/// <returns>Le code HTTP de la réponse avec d'éventuels objets en retour.</returns>
//// POST api/Documents
// Si l'on souhaite uniquement des requêtes Ajax, on met ça : [AjaxOnly]

[HttpPost( "CreateDocument" )]
0 références | Luca BEROULE, il y a 42 jours | 2 auteurs, 2 modifications
public async Task<IActionResult> Post( DocumentEditViewModel model )
    {

    try
    {
        if( ModelState.IsValid )
        {

            //Vérification de la disponibilité du libellé choisi
            if ( await _documentRepository.NumeroExistsAsync( model.NoDocument ) )
            {
                //Méthode Max + 1 pour le numéro en fonction du domaine
                return StatusCode( 500, "Le document " + model.NoDocument + " existe déjà" );
            }

            await _documentRepository.CreateDocumentAsync( model );

            return Ok( new { id= model.DocumentId } );

        }

        // Conversion des erreurs du ModelState en JSON.
        string errors = JsonConvert.SerializeObject( ModelState.Values
            .SelectMany( state => state.Errors )
            .Select( error => error.ErrorMessage ) );

        return BadRequest( errors );
    }
    catch( Exception e )
    {
        var er = e.InnerException.Message;
        return StatusCode( 500, "Une erreur est survenue lors de la tentative d'ajout d'un nouveau Document. " );
    }
}
```

J'ai du aussi faire en sorte que le type de document soit cohérent avec le domaine sélectionné.
On ne peut pas avoir de type de document Entrée de Stock, si on a choisi un domaine Cantine.
Il faut choisir le domaine stock pour y accéder mais là aussi.
Si on choisit un domaine Stock alors, on ne peut sélectionner de type Devis, Facture ou encore Bon de Livraison.

Pour faire apparaitre les bonnes dropdowns correspondant au domaine choisi, j'ai dû faire appel à une fonction JavaScript. La logique était de masquer au préalable toutes les dropdowns liée au 3 domaines (Cantine, Etablissement et Fournisseur). Et en fonction de la valeur que l'on récupère dans la dropdown (donc le domaine que l'utilisateur choisi). On affiche la bonne dropdown.

- Fonction JavaScript onDomaineDocumentChange : Voir Figure 1
- Page d'ajout : Voir Figure 2
- Résultat dans l'index : Voir Figure 3

3.2.2 : CRUD des lignes des documents

Deuxième étape : Pouvoir renseigner des articles dans le document
Cette partie a été très compliquée car c'est ici que je suis rentré dans le vif du sujet.
J'ai donc crée une Kendo Grid dans le formulaire de modification d'un document qui prend

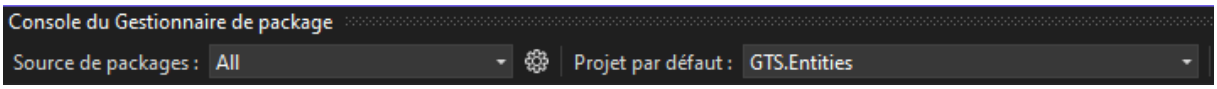
comme modèle DocumentsLignes. Un Nouveau modèle (table dans la base de données) que j'ai crée et ajouté (Avec les migrations de EF)

Les migrations de EntityFramework dans l'application fonctionnent de cette façon : GTS.Entities/Models est la partie qui gère les modèles, dans mon contexte ici avec le modèle DocumentLigne fraichement crée :

Extrait du modèle DocumentLigne

```
[Table("DocumentsLignes")]
23 références | Luca BEROULE, il y a 43 jours | 2 auteurs, 2 modifications | 2 modifications entrantes
public class DocumentLigne
{
    [Key]
    [Display(Name = "Code Ligne de Document :" )]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    6 références | Davy GRIGRI | 1 auteur, 1 modification | 2 modifications entrantes
    public int DocumentLigneId { get; set; }

    [Required]
    [Display( Name = "Code Document" )]
    4 références | Davy GRIGRI | 1 auteur, 1 modification | 2 modifications entrantes
    public int DocumentId { get; set; }

    [Display( Name = "Code Article" )]
    8 références | Luca BEROULE, il y a 43 jours | 2 auteurs, 2 modifications | 2 modifications entrantes
    public int? ArticleId { get; set; }

    [StringLength(255)]
    [Display(Name = "Désignation" )]
    10 références | Davy GRIGRI | 1 auteur, 1 modification | 2 modifications entrantes
    public string Designation { get; set; }

    [Display( Name = "Quantité" )]
    10 références | Luca BEROULE, il y a 43 jours | 2 auteurs, 2 modifications | 2 modifications entrantes
    public int? Quantite { get; set; }
```

Ensuite après avoir créé cette classe, je peux ouvrir la console du Gestionnaire de package, sélectionner le bon projet par défaut (ou Entity est paramétrée)



Et entrer la ligne suivante « Add-Migration ».

Il faut aussi rajouter une référence dans un fichier appelée ApplicationDbContext. C'est ici que la liaison entre le code C# et la base de données est faite.

Référence de la table DocumentsLignes :

```
13 références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
public DbSet<DocumentLigne> DocumentsLignes { get; set; }
```

Grace a ça, on pourra donc appeler la référence a la table que l'on vient de créer. On doit maintenant préparer ce que l'on veut afficher et pour ça, il faut créer un ViewModel et la vue, qui prendra comme paramètre, le ViewModel mapper à la table DocumentsLignes. Pour cela on utilise AutoMapper. Le modèle « Document » étant déjà mapper, il faut se rendre dans le DocumentProfile et y rajouter un mappage pour mon modèle :

Extrait de DocumentProfile :

```
CreateMap<DocumentLigne, DocumentLigneViewModel>()
    .ForMember(x => x.DocumentId, x => x.MapFrom(y => y.DocumentId))
    .ForMember(x => x.DocumentLigneId, x => x.MapFrom(y => y.DocumentLigneId))
    .ForMember(x => x.Designation, opt => opt.MapFrom(y => y.Designation))
    .ForMember(x => x.Quantite, x => x.MapFrom(y => y.Quantite))
    .ForMember(x => x.PrixUnitaire, x => x.MapFrom(y => y.PrixUnitaire))
    .ForMember(x => x.ArticleId, x => x.MapFrom(y => y.ArticleId))
```

Le mapping se fait dans deux sens : du modèle au modelé de vue et inversement (fonction .ReverseMap) Sans ça, l'application ne peut pas reconnaitre les propriétés dans le viewModel.

Et maintenant on peut attaquer la vue : J'ai donc décider de faire une Kendo Grid pour afficher les articles concernés par le document :

Extrait du composant Kendo : Voir Figure 4

Le bouton nouvelle ligne rajoute une ligne vide dans la grille. Une dropdown est affiché dès que le champ Nom de l'article est sélectionné :

Méthode du Controller :

```
[HttpPost("PostLigne")]
0 références | Luca BEROULE, il y a 57 jours | 1 auteur, 1 modification
public async Task<IActionResult> PostLigne(DocumentLigneEditViewModel model)
{
    try
    {
        if (ModelState.IsValid)
        {
            // Crée la ligne vide dans la base de données
            await _documentLigneRepository.CreateEmptyDocumentLigneAsync(model);

            // On retourne un code 201.
            return Ok("Les lignes ont été ajoutées");
        }

        // Conversion des erreurs du ModelState en JSON.
        string errors = JsonConvert.SerializeObject(ModelState.Values
            .SelectMany(state => state.Errors)
            .Select(error => error.ErrorMessage));

        return BadRequest(errors);
    }
    catch (Exception e)
    {
        var er = e.InnerException.Message;
        return StatusCode(500, "Une erreur est survenue lors de la tentative d'ajout d'une nouvelle ligne. ");
    }
}
```

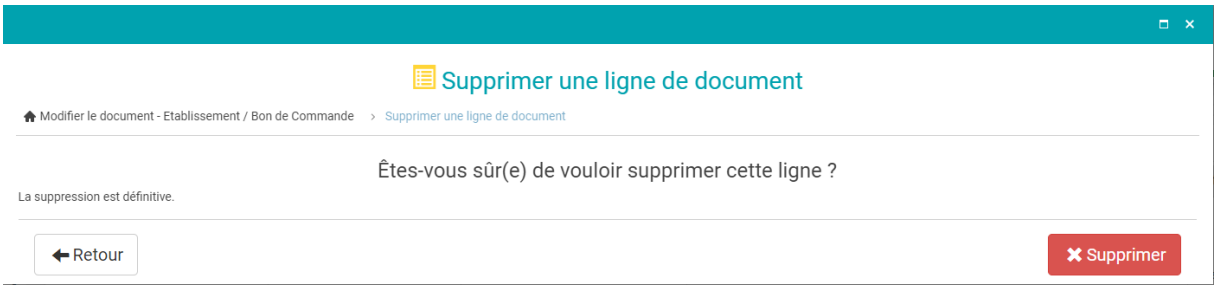
Méthode dans le repository :

```
2 références | Luca BEROULE, il y a 57 jours | 1 auteur, 1 modification
public async Task CreateEmptyDocumentLigneAsync(DocumentLigneEditViewModel model)
{

    var newDocumentLigne = mapper.Map<DocumentLigne>(model);

    try
    {
        newDocumentLigne.Designation = "";

        _context.DocumentsLignes.Add(newDocumentLigne);
        await _context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Les données peuvent être entré en mode édition, c'est-à-dire qu'un utilisateur peut modifier sa ligne quand il le souhaite. Mais Uniquement sur certains champs, par exemple, il ne peut pas modifier le prix d'un article qu'il a déjà sélectionné.
J'ai fait en sorte que la ligne soit mise à jour dès qu'une modification est détectée et que les montants soient calculés uniquement si la quantité est renseigné.

Articles Sélectionés

Le reste des fonctionnalités (Read et Delete) était assez facile bien que j'aie rencontrés quelques difficultés. Le bouton de suppression renvois sur une page de confirmation ->



La documentation de Kendo a été la ressource la plus consulté durant ce stage. Et c'est elle ainsi que mon tuteur de stage et collègue de travail qui m'ont aidé a terminé cette partie du module.

3.2.3 : Gestion des dépôts (Etat des stocks)

Ensuite, une étape tout aussi importante que les lignes du document : Les dépôts

En effet, on doit pouvoir avoir une gestion des dépôts car l'établissement doit pouvoir gérer et maitriser son inventaire. Les modifications du contenu des dépôts sont uniquement faites au travers des documents Stock (Domaine de document = Stock) et les types Entrée de stock / Sortie de stock et Transfert de stock.

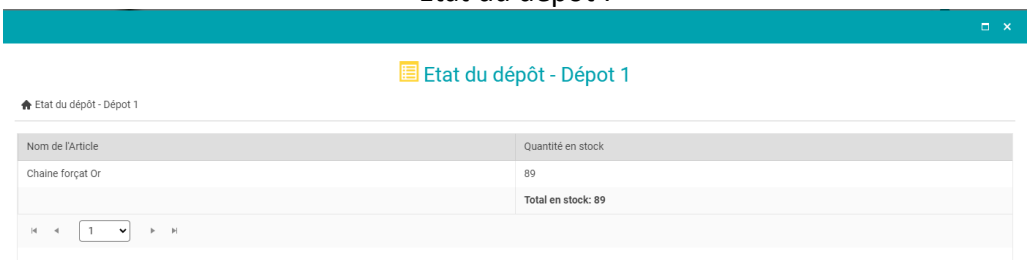
Chaque variante aura donc une fonction différente : L'entrée ajoute au dépôt, la sortie retire au dépôt et le transfert déplace d'un dépôt a un autre. Il y aura donc deux variantes de dépôt : un dépôt d'origine et un dépôt de destination. Autre critère important : Les conditions de suppression, car si un dépôt est mentionné : par exemple par une entrée ou un transfert, Il ne peut plus être supprimé tant que le document qui le mentionne n'est pas supprimé.

Entrée de Stock : Voir Figure 5

Etant donné que j'ai mis ceci en place, il me fallait un moyen de consulter l'inventaire du dépôt, alors j'ai rajouté un bouton dans l'onglet Dépôt qui a déjà été créé afin de visualiser le contenu du dépôt

Index des dépôts :

Etat du dépôt :



3.2.4 : Impression d'un document

L'impression du document doit permettre de faire sortir le document voulu en format PDF.

Mais avant de commencer à coder. Il me faut la base et en explorant du code déjà existant dans l'application. J'ai remarqué qu'ils utilisaient des « rapports » pour sortir leurs différents documents en PDF que ce soient les listes des cantines / transports (Module Les Rapports) etc.
Donc j'allais devoir faire un rapport pour mes documents et pour cela j'ai utilisé Microsoft Report Builder. J'ai du d'abord me connecter au Serveur de Rapport qui connait déjà le serveur de base de données sur lequel je travaille.

Voici le Rapport crée dans MRB : Figure 6

Les crochets désignent des éléments à afficher. Derrière chaque rapport, il y a une requête SQL. Par exemple dans mon cas. Avec un paramètre donné (DocumentId). Les données que je vais recevoir vont varier en fonction du document qui est demandé.

Le champ « Expr » correspond aux expressions de MRB : Les expressions sont couramment utilisées dans des rapports paginés pour récupérer, calculer, afficher, regrouper, trier, filtrer, paramétrer et mettre en forme les données.

Maintenant que nous avons notre rapport on peut construire notre bouton qui va récupérer ce rapport.

Il s'agit d'un simple bouton, qui une fois appelé, va aller chercher le bon « data-role » dans la fonction JS « OnCustomButtonClick » du fichier KendoGridAction.js

HTML du bouton :

```
<div class="clearfix row">
    <div class="col-3 col-sm-3 form-group">
        <button class="btn btn-primary" onclick="OnCustomButtonClick(this)" data-role="Print-DocumentMateriel">
            <span class="far fa-file-pdf" aria-hidden="true"></span>
            <span class="k-button-text">Exporter vers PDF</span>
        </button>
    </div>
</div>
```

Fonction JS - > (La fonction prend 2 paramètres. La nature (ici « DocumentMateriel ») et le type (ici « Print ») -> Voir Figure 7

Ensuite avec l'url donnée -> On exécute la méthode TelechargerDocument, qui prend en parametre l'ID du document que nous souhaitons exporter, dans le controller Document.

```
DocumentMateriel: {
    Create: "@Url.Action( "FormCreate", "Document" , new { Area = "Materiel", SubArea = "Document" })",
    Edit: "@Url.Action("FormEditDoc", "Document", new { Area = "Materiel", SubArea = "Document", documentId = "_documentId_", domaineDocumentId = "_domaineDocumentId_", documentType = "_documentType_" })",
    Delete: "@Url.Action( "FormDelete", "Document", new { Area = "Materiel", SubArea = "Document", documentId = "_documentId_" } )",
    CreateLigne: "@Url.Action( "PostLigne", "Document" , new {  Area = "Materiel", SubArea = "Document",  documentId = "_documentId_" } )",
    ModifLigne: "@Url.Action( "PutLignes", "Document" , new {  Area = "Materiel", SubArea = "Document",  documentLigneId = "_documentLigneId_" } )",
    DeleteLigne: "@Url.Action( "FormDeleteLigne", "Document", new { Area = "Materiel", SubArea = "Document", documentLigneId = "_documentLigneId_" } )",
    RefreshGrid: "@Url.Action("PostLignes", "Document", new { Area = "Materiel", SubArea = "Document" })",
    Print: "@Url.Action( "TelechargerDocument" , "Document" , new { Area = "Materiel", SubArea = "Document", documentId = "_documentId_" } )",
},
```

Voir Figure 8 : Méthode TelechargerDocument

Sortie du document en PDF : (peut varier en fonction du nombre d'articles)

Facture

Cantine : Cantine Etablissement 2

Numéro du document : CanFA000041

Référence :

Date du document : 16/01/2024 12:05:00

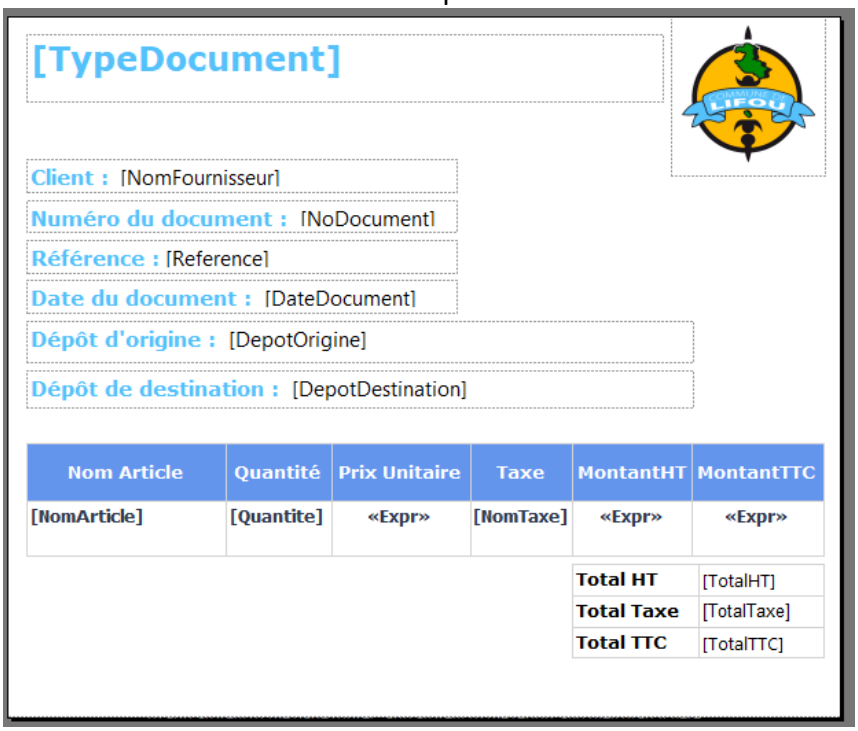
Nom Article	Quantité	Prix Unitaire	Taxe	MontantHT	MontantTTC
NewTest	5	560	TGC à 0%	2 800	2 800
test	10	500	TGC à 3%	5 000	5 150
Chaine forçat Or	10	80 000	TGC à 0%	800 000	800 000

Total HT	2 800
Total Taxe	0
Total TTC	2 800

Comme j'ai modifié les documents pour faire en sorte que le domaine sélectionné affiche la dropdown voulue : J'ai dû adapter la requête SQL du ReportBuilder pour l'adapter au document.

Finalement, j'ai recréé un rapport spécialement pour les documents de stocks. Mais en modifiant la requête (en rajoutant les dépôts origine et destination). Et aussi en ajoutant une condition d'affichage pour les champs dépôt origine et destination. Donc si c'est une entrée de stock, on affiche uniquement dépôt origine, si c'est une sortie, uniquement dépôt origine et si c'est un transfert on peut afficher les deux.

Rendu dans ReportBuilder :



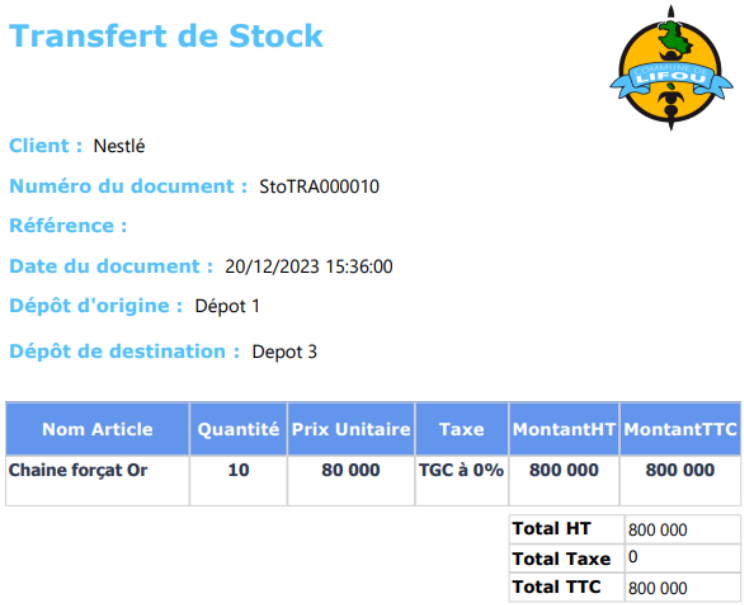
[TypeDocument]

Client : [NomFournisseur]
Numéro du document : [NoDocument]
Référence : [Reference]
Date du document : [DateDocument]
Dépôt d'origine : [DepotOrigine]
Dépôt de destination : [DepotDestination]

Nom Article	Quantité	Prix Unitaire	Taxe	MontantHT	MontantTTC
[NomArticle]	[Quantite]	«Expr»	[NomTaxe]	«Expr»	«Expr»

Total HT	[TotalHT]
Total Taxe	[TotalTaxe]
Total TTC	[TotalTTC]

Rendu PDF du document de Stock :



Transfert de Stock

Client : Nestlé
Numéro du document : StoTRA000010
Référence :
Date du document : 20/12/2023 15:36:00
Dépôt d'origine : Dépot 1
Dépôt de destination : Depot 3

Nom Article	Quantité	Prix Unitaire	Taxe	MontantHT	MontantTTC
Chaine forçat Or	10	80 000	TGC à 0%	800 000	800 000

Total HT	800 000
Total Taxe	0
Total TTC	800 000

3.2.5 : Améliorations du module

Quelques améliorations ont dû être apportés au module pour que son ensemble soit plus cohérent.
- Conditions de suppression : Les Taxes qui ont été mentionner par un article ne peuvent plus être supprimer
- Les articles mentionner par une ligne de document ne peuvent plus être supprimé
- Les familles mentionné par un article ne peuvent plus être supprimé

- Les dépôts mentionné par un document de stock ne peuvent plus être supprimé

j'ai pu mettre en place tout ça grâce à la classe abstraite DisplayViewModel :
Et plus particulièrement grâce à la variable CanDelete sur laquelle j'ai pu mettre une contrainte dans les profils de mappage.

DisplayViewModel (bool initialisé a true : VRAI)

```
46 références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
public abstract class DisplayViewModel
{
    [Display( Name = "Code" )]
    99+ références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
    public int Id { get; set; }

    2 références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
    public bool CanEdit { get; set; } = true;
    0 références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
    public bool CanView { get; set; } = true;
    26 références | Davy GRIGRI | 1 auteur, 1 modification | 1 modification entrante
    public bool CanDelete { get; set; } = true;
}
```

Et aussi grâce à la mise à jour des profile : Par Exemple :

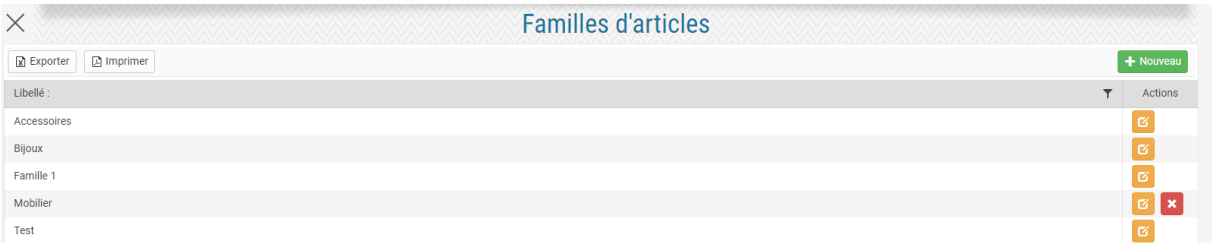
Profil de mappage pour Famille -> FamilleViewModel

```
CreateMap<Famille, FamilleViewModel>()
    .ForMember( x => x.Id, x => x.MapFrom( y => y.FamilleId ) )
    .ForMember( x => x.Libelle, x => x.MapFrom( y => y.Libelle ) )
    .ForMember( x => x.CanDelete, x => x.MapFrom( y => y.Articles.Count() == 0 ) );
```

Ici, on peut voir que le CanDelete est paramétré de façon a ce qu'il soit FAUX s'il y a un article qui fait référence à une famille. Donc une famille ne peut être supprimé si un article fait référence a la famille que l'on veut supprimer.

Ensuite, il faut pouvoir afficher / masquer les boutons en fonction de la booléenne. On fait cela avec du code JavaScript : Voir Figure 9

Index des articles (CanEdit sur le bouton jaune et CanDelete sur le bouton rouge)



Ensuite, j'ai dû revoir la façon de renseigner le numéro du document. Après avoir montré mon module, j'ai reçu des retours sur le numéro de document. En effet, le numéro de document était une simple chaine de caractères ou l'on pouvait y mettre absolument ce que l'on veut.
Le numéro doit être en readonly et attribué dès la création du document.

Pour pallier le problème, voici comment j'ai dû procéder :

Chaque document a un numéro attribué à la création. Ce numéro varie en fonction du domaine et du type de document choisi. Le numéro de document est stocké dans une table avec l'id du domaine, du type, une variable string « Souche » et un int « numéro ». Chaque fois qu'un document est créé, on lui attribue sa souche et son numéro (commence généralement a 0).

J'ai donc apporter une nouvelle interface dans l'onglet Administration, cet onglet gère la disponibilité des modules de l'application, les utilisateurs et leurs rôles sur l'application et une partie contrôle de données qui permet de vérifier certaines informations comme les élèves en doublon.

De mon côté, j'ai créé une interface afin de gérer les souches et leur numéro. Si un administrateur souhaite changer les souches et le numéro d'un cas de document particulier comme, par exemple : les devis d'un fournisseur :

Voir Figure 10 : Index des souches

Même processus que le canDelete, mais cette fois avec canEdit (Si un document est mentionné avec sa souche -> la souche ne pourra pas être modifiable)

Ensuite, il fallait que l'on puisse accéder à l'onglet « Les Souches » que je voulais ajouter ici :

Et j'ai pu découvrir l'aspect technique de la partie : SideMenu, donc le menu « principal » de l'application

Voir Figure 11 : Menu principal de l'application

Voir Figure 12 : Modification d'une souche

3.2.6 : Difficultés rencontrées

Ce stage aura été un véritable parcours du combattant entre les petits points sur lesquels je bloquais qui prenait des jours et des semaines. A certains points qui me paraissait impossible à finir en une semaine, qui finalement était terminés en une journée. Je peux dire que j'ai eu mon lot de difficultés et je vais énumérer lesquelles ont été les plus marquantes et les plus fréquentes.

- ➔ La gestion des url : Un « petit » fichier appelé KendoGridAction.js contenant une fonction très importante : OnCustomButtonClick. Cette fonction est indispensable pour être rediriger sur la bonne url. J'ai pris pas mal de temps à me familiariser avec elle. Mais elle m'aura causé bien des soucis. C'est notamment grâce a elle que j'ai pu m'habituer à travailler avec DevTools.
- ➔ La grille des lignes : Véritable point d'arrêt de mon stage, ou je suis passé par plusieurs version de celle-ci, avant d'en avoir une stable, logique et cohérente. De

plus, les composants kendo et la documentation sur celle-ci était assez contradictoire notamment du côté des dataSource ou j'ai eu le plus de mal à m'adapter.

➔ Créer une ligne vide : En effet la création d'une ligne vide bien que simple logiquement, a été un peu plus compliquée que ce que j'avais anticiper. Surtout car je pensais être sûr de devoir passer par un update au niveau du dataSource alors qu'au final j'ai du passer par une requête Ajax qui fonctionne parfaitement.

4. Conclusion

4.1 : Conclusion et remerciements

Durant ce stage, j'ai travaillé en tant que développeur full-stack sur le projet GTS / CDE

Un grand merci à Toute l'équipe d'ARCNET de m'avoir accueilli pour ce stage malgré les conditions…
Ce stage fut extrêmement enrichissant car j'ai pu augmenter mes connaissances en C# .NetCore, dans la logique en général. J'ai pu aussi apprendre beaucoup sur Javascript et JQuery. J'ai aussi pu m'autoformer sur de nouvelles méthodes de travail comme en utilisant des points d'arrêts dans Visual Studio ou sur DevTools pour debugger le JavaScript. J'ai pu aussi apprendre à maitriser des outils et frameworks tels que Telerik, AutoMapper et EntityFramework. J'ai pu aussi développer une certaine connaissance des erreurs et apprendre à comment les résoudre rapidement. En effet, savoir ou chercher et quelle partie du code est à modifier est essentiel dans la résolution d'un bug et au fur a mesure que mon stage se déroulait, je pouvais le corriger sans demander d'aide.

En parallèle de ce projet, j'ai pu apporter quelques modifications et fonctionnalités comme changer du style, corriger le breadcrumb (Fil d'Ariane) du module, automatiser l'attribution du numéro d'un document, rajouter un pied de page pour les montants totaux, ajouter une nouvelle interface pour gérer (afficher et modifier) les souches des documents. J'ai su aussi régler mes propres bugs car certains tests m'ont créé quelques soucis en plus. J'ai aussi pu réorganiser mon code et l'adapter a la logique métier de l'application, J'ai aussi pu apporter mon aide à mon collègue.
J'ai su trouver des réponses grâce aux pistes de mon tuteur et l'autonomie que ce stage a su m'incorporer.

4.2 : Sources bibliographiques

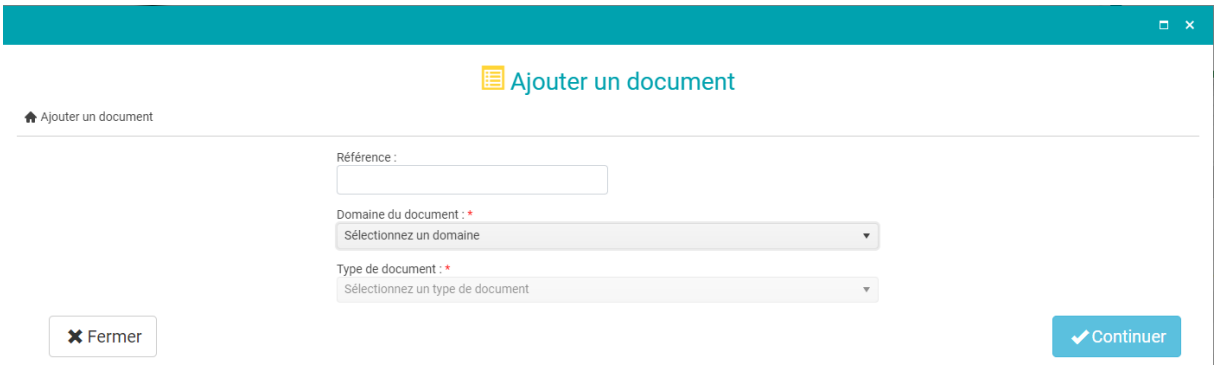
- Documentation Telerik MVC (Composants Kendo)
- Fonctions JQuery de Kendo (Dans API REFERENCE / JAVASCRIPT)
- W3Schools (HTML / CSS / JS / JQuery / C#)
- Blog - StackOverflow (Problèmes concrets et cas particuliers)
- Microsoft Learn pour la Syntaxe Entity Framework
- Documentation AutoMapper
- Chat GPT

4.3 : Annexes

Figure 1 : Fonction JS de traitement des dropdowns

```
function onDomaineDocumentChange() {
    var selectedDomaine = $("#DomaineDocumentId").data("kendoDropDownList").value();
    hideAllDropdowns();

    // Show the specific dropdown based on the selected value
    if (selectedDomaine === "3") {
        $(".cantine-dropdown").show();
        enableValidation("CantineId");
        setValidationMessage("CantineId", "La Cantine est obligatoire.");
    } else if (selectedDomaine === "2") {
        $(".etablissement-dropdown").show();
        setValidationMessage("EtablissementId", "L'etablissement est obligatoire.");
        enableValidation("EtablissementId");
    } else if (selectedDomaine === "4") {
        $(".fournisseur-dropdown").show();
        setValidationMessage("FournisseurId", "Le Fournisseur est obligatoire.");
        enableValidation("FournisseurId");
    } else if (selectedDomaine === "1") {
        hideAllDropdowns();
        disableValidation("CantineId");
        disableValidation("EtablissementId");
        disableValidation("FournisseurId");
    }
}

function hideAllDropdowns() {
    $(".cantine-dropdown, .etablissement-dropdown, .fournisseur-dropdown").hide();
}

function setValidationMessage(elementId, message) {
    // Set the validation message for the specified element
    $("#" + elementId).attr("data-val-required", message);
}

function enableValidation(elementId) {
    // Enable validation for the specified element
    $("#" + elementId).rules("add", { required: true });
}

function disableValidation(elementId) {
    // Disable validation for the specified element
    $("#" + elementId).rules("remove", "required");
}
```

Figure 2 : Formulaire d'ajout d'un document (Avec cas)



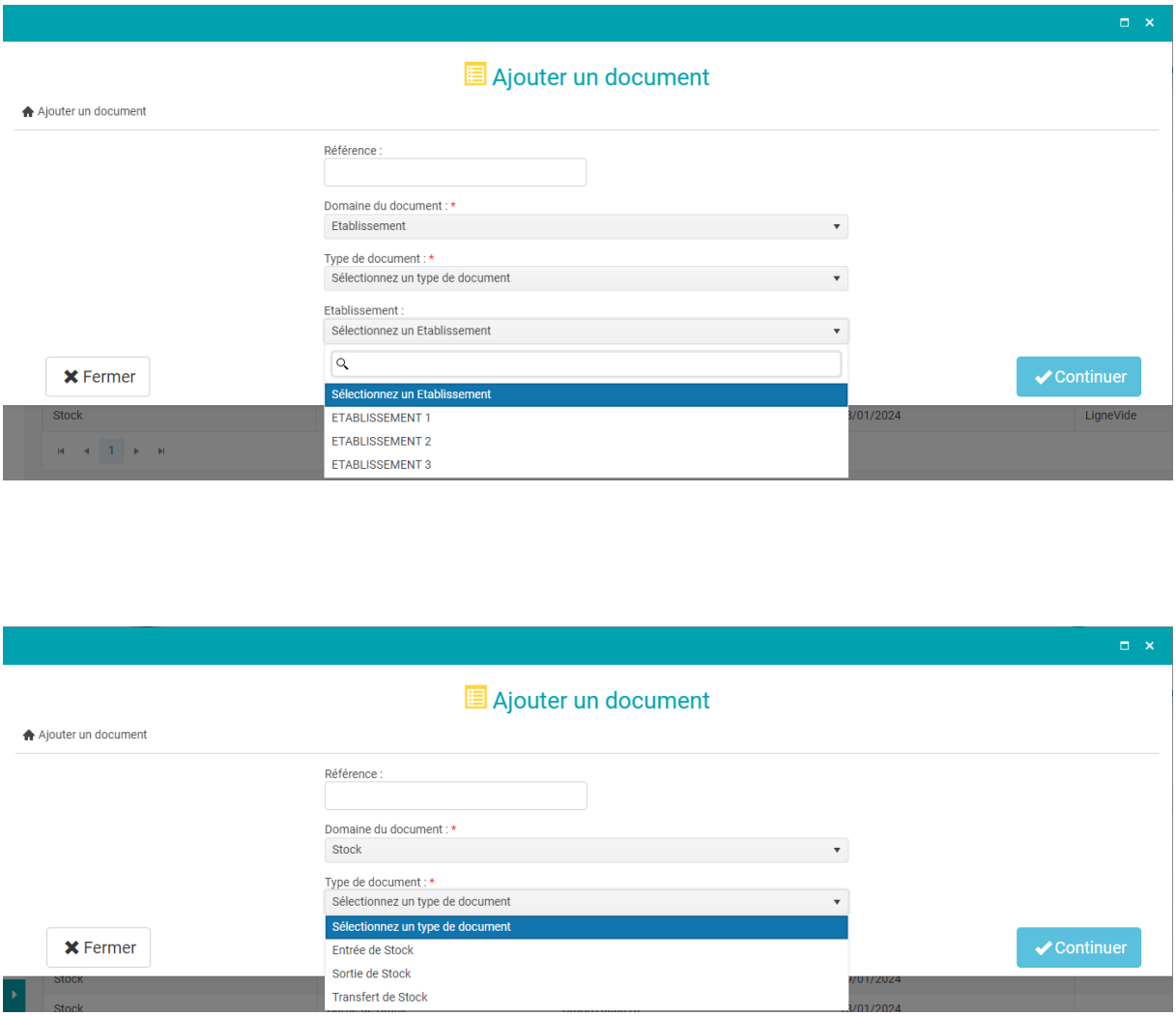


Figure 3 : Index des documents

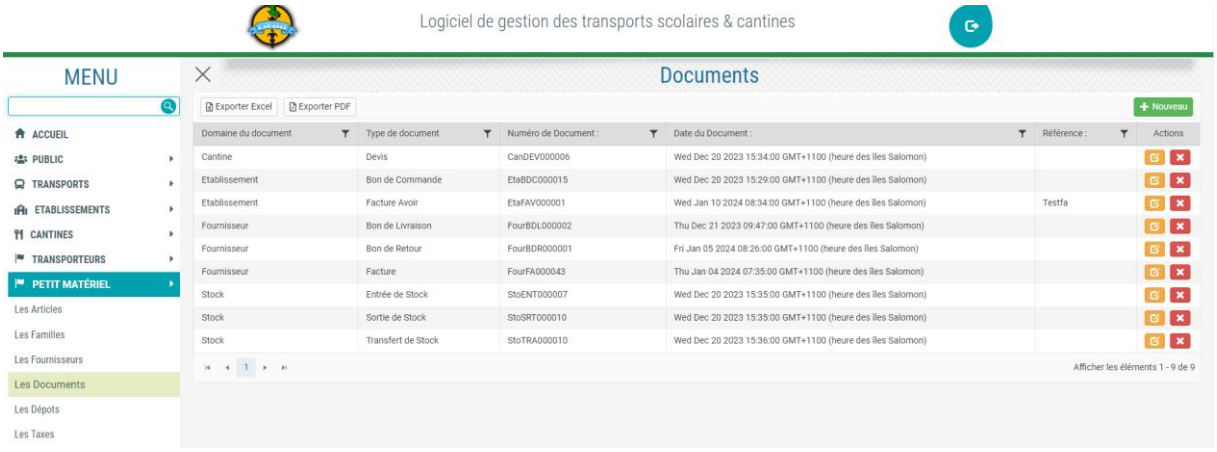


Figure 4 : Code des Lignes de documents

```
@(
Html.Kendo().Grid<DocumentLigneEditViewModel>()
        .Name("lignesgrid")
        .Columns(columns =>
        {
            columns.Bound(e => e.DocumentLigneId).Hidden();
            columns.Bound(e => e.DocumentId).Hidden(true);
            columns.Bound(e => e.ArticleId).Hidden(true);

            columns.Bound(e => e.Designation).Width("20");
            columns.Bound(e => e.Article).Width("20").ClientTemplate("#=Article?.Libelle ?? \"\"#").EditorTemplateName("_DropDownArticles").Title("Nom de l'Article");



            // Affichage du dépot de destination si le type est un Bon de Livraison ou une Facture
            if (Model.DocumentTypeId == 3 || Model.DocumentTypeId == 5)
            {
                columns.Bound(e => e.DepotDestination).EditorTemplateName("_DropDownListDepotDestination").ClientTemplate("#=DepotDestination?.Libelle ?? \"\"#").Title("Dépôt de Destination");
            }
            columns.Bound(e => e.Quantite).Width("20").Title("Quantité");

            columns.Bound(e => e.PrixUnitaire).Width("20").HtmlAttributes(new { disabled = "disabled" }).Title("Prix Unitaire");
            columns.Bound(e => e.MontantHT).Format("{0:N0}").Width("20");
            columns.Bound(e => e.MontantTaxe).Format("{0:N0}").Title("Montant de la Taxe");
            columns.Bound(e => e.MontantTTC).Format("{0:N0}").Width("20");
            columns.Command(command =>

            {
                command.Custom("DeleteLigne")
            .IconClass("glyphicon glyphicon-remove")
            .Text(" ")
            .Click("OnCustomButtonClick")
        .HtmlAttributes(new { @class = "btn btn-xs btn-danger center-btn", data_role = "DeleteLigne-DocumentMateriel", style = "text-align: center;" });
            })
            .Width("30px").Title("Actions")
            .HeaderHtmlAttributes(new { style = "text-align: center;" });
        })
        .HtmlAttributes(new { style = "text-align: center; font-size: 85%" })
        .Pageable()
        .Selectable(selectable => selectable
        .Mode(GridSelectionMode.Single))
        .Editable(editable => editable
        .Mode(GridEditMode.InCell)
        )
        .ToolBar(ToolBar =>
        {
            ToolBar.Custom().Text("Nouveau")
            .IconClass("glyphicon glyphicon-plus")
            .Text(" Nouvelle Ligne ")
            .HtmlAttributes(new
            {
                @class = "btn btn-xs btn-success pull-right",
                data_role = "CreateLigne-DocumentMateriel",
                id = "newLigne",
                onclick = "New(this)"
            });
        })
```

```
.DataSource(dataSource => dataSource
.Ajax()
.Aggregates(aggregates =>
{
    aggregates.Add(p => p.MontantHT).Sum();
    aggregates.Add(p => p.MontantTTC).Sum();
    aggregates.Add(p => p.MontantTaxe).Sum();
})
.Batch(true)
.AutoSync(true)
.PageSize(10)
.Model(model =>
{
    model.Id(p => p.DocumentLigneId);
    model.Field(p => p.DocumentId).Editable(false);
    model.Field(p => p.ArticleId).Editable(false);
    model.Field(p => p.PrixUnitaire).Editable(false);
    model.Field(p => p.DepotDestinationId).Editable(false);
    model.Field(p => p.Article).Editable(true);
    model.Field(p => p.DepotDestination).Editable(true);

})

.Events(events => events
.RequestEnd("onRequestEnd")
.Error("errorHandler")
)

.Read(read => read.Action("GetAllLignes", "Document", new { Area = "Materiel", SubArea = "Document" }).Data("RequestData").Type(HttpVerbs.Get))
.Update(update => update.Action("PutLignes", "Document", new { Area = "Materiel", SubArea = "Document" }).Data("RequestData").Type(HttpVerbs.Put))
)
)
```

Figure 5 : Formulaire de modification d'un document de stock

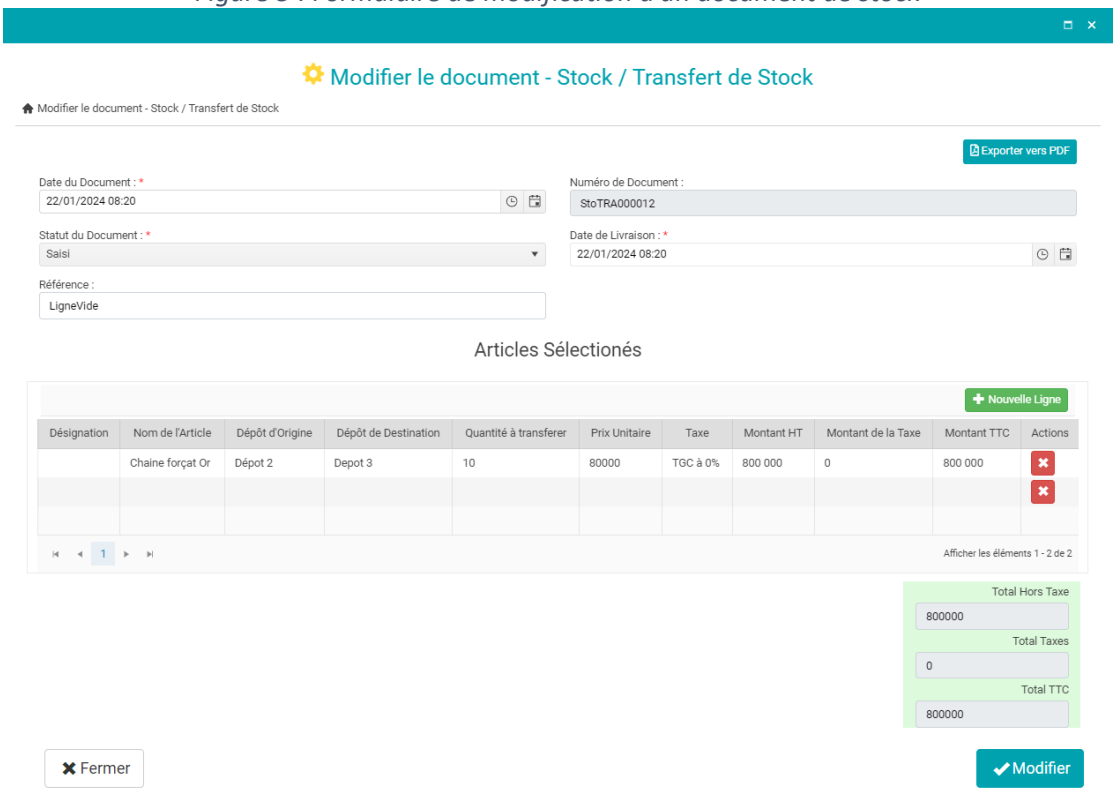


Figure 6 : Template d'un rapport

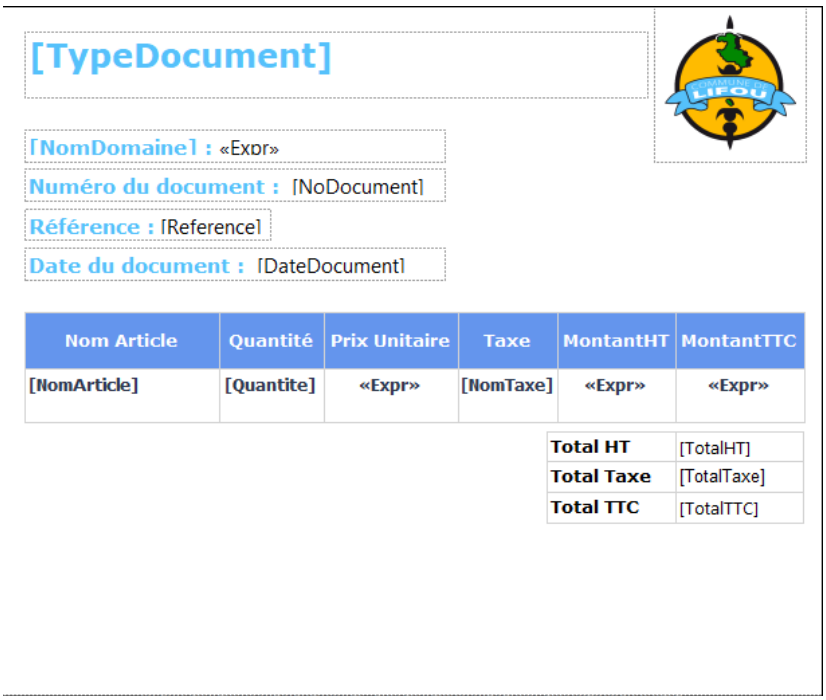


Figure 7 : Fonction JS qui gère la redirection d'url

```
function OnCustomButtonClick(e, source) {
    var action = "Standard";
    switch (source) {
        case "Planning":
            var role = e.role;
            break;
        default :
            var target = e.target ? $(e.target) : $(e);
            var role = target.data("role");
            break;
    }

    var dataItem = e.target ? this.dataItem(target.closest("tr")) : null;


    // Récupération de la nature ex: 'Alerte'
    var nature = role.split('-').splice(1, 1)[0];
    nature = nature[0].toUpperCase() + nature.substring(1);

    // Récupération du type ex: 'Edit'
    var type = role.split('-').splice(0, 1)[0];
    type = type[0].toUpperCase() + type.substring(1);

    var url;

    url = decodeHTMLString(config[nature][type]);

    url = url.replace("_id_", encodeURIComponent(dataItem?.Id));

    switch (nature) {
        case "CarteTransport":
            switch (type) {
                case "Create":
                    url = url.replace("_arretId_", $("#ArretId").val());
                    url = url.replace("_etablissementId_", $("#EtablissementId").val());
                    url = url.replace("_conditionId_", $("#ConditionId").val());
```

```
case "DocumentMateriel":
    switch (type) {
        case "View":
            url = url.replace("_documentId_", dataItem.Id);
            action = "View"
            break;
        case "Create":
            url = url.replace("_documentId_", $("#DocumentId").length == 0 ? 0 : $("#DocumentId").val() !== "" ? $("#DocumentId").val() : 0);
            url = url.replace("_domaineDocumentId_", $("#DomaineDocumentId").val());
            break;
        case "CreateLigne":
            url = url.replace("_documentId_", $("#DocumentId").val());
            break;
        case "Edit":
            let domaineId = dataItem.Domaine;
            url = url.replace("_domaineId_", domaineId);
            url = url.replace("_documentId_", dataItem.DocumentId);

            break;
        case "Delete":
            url = url.replace("_documentId_", dataItem.DocumentId);
            break;
        case "Print":
            console.log($("#DocumentId").val())
            var documentId = $("#DocumentId").val();
            url = url.replace("_documentId_", documentId);
            action = "Blank";
            break;
        case "DeleteLigne":
            if (dataItem.DocumentLigneId === 0) {
                var grid = $("#lignesgrid").data("kendoGrid");
                var documentId = $("#DocumentId").val();
                grid.removeRow(e.target.closest("tr"))
                action = "Cancel";

            }
```

Figure 8 : Code de la génération d'un rapport

```
[HttpGet("TelechargerDocument")]
//[Authorize(Policy = "Claim.Eleves.Access")]
0 références | 0 modifications | 0 auteurs, 0 modifications
public async Task<IActionResult> TelechargerDocument(int documentId)
{
    try
    {
        // Récupere le type de document du document selectionné
        var Document = await _documentRepository.GetDocumentAsync<DocumentEditViewModel>(documentId);

        // Genere le bon rapport en fonction du Type donné ( STOCK )
        if (Document.DocumentType.Rapport == "Document")
        {
            // On récupere le document
            var DocumentId = await _documentRepository.GetDocumentIdAsync(documentId);

            // le parametre a donnée (ici on a besoin de l'id du document pour accéder a ses informations)
            var parameter = new List<SSRService.ParameterValue>
            {
                new SSRService.ParameterValue() { Name = "DocumentId", Value = DocumentId.ToString() }
            };

            // On stocke dans cette variable, le rapport généré par le SSR
            var bytes = await _reportingRepository.RunReport("Document", parameter, OutputFormat.PDF, RenderFormat.A4);

            string fileName = "Document_" + DocumentId + ".pdf";

            return File(bytes, System.Net.Mime.MediaTypeNames.Application.Pdf, fileName);
        }
```

```
        // Genere le bon rapport en fonction du Type donné ( Tout sauf les documents de stock )
        else if (Document.DocumentType.Rapport == "DocumentStock")
        {
            var DocumentId = await _documentRepository.GetDocumentIdAsync(documentId);

            var parameter = new List<SSRService.ParameterValue>
            {
                new SSRService.ParameterValue() { Name = "DocumentId", Value = DocumentId.ToString() }
            };

            var bytes = await _reportingRepository.RunReport("DocumentStock", parameter, OutputFormat.PDF, RenderFormat.A4);

            string fileName = "Document_" + DocumentId + ".pdf";

            return File(bytes, System.Net.Mime.MediaTypeNames.Application.Pdf, fileName);
        }
        else
        {
            // Retourne NotFound si le rapport ne correspond pas
            return NotFound("Le document demandé n'existe pas ou n'est pas du type attendu.");
        }
    }
    catch (Exception)
    {
        // Retourne un StatusCode 500 en cas d'erreur inattendue
        return StatusCode(500, "Une erreur est survenue lors de la tentative de téléchargement du document.");
    }
}
```

Figure 9 : Fonction JavaScript / Affichage du bouton Supprimer

```
<script>
    $(document).ready(function () {
        var grid = $("#liste-items").data("kendoGrid");

        // Fonction pour masquer/afficher le bouton de suppression pour les familles
        function toggleDeleteButtonForFamille() {
            grid.tbody.find("tr").each(function () {
                var dataItem = grid.dataItem(this);

                // Vérifie la valeur de CanDelete pour décider d'afficher ou masquer le bouton
                var canDelete = dataItem.CanDelete;


                // Replace "FamilleDelete" par le nom correct du bouton de suppression de famille
                var deleteButton = $(this).find("[data-role='delete-famille']");

                if (canDelete) {
                    deleteButton.show();
                } else {
                    deleteButton.hide();
                }
            });
        }

        // Attache la fonction au événement DataBound de la grille
        grid.bind("dataBound", toggleDeleteButtonForFamille);

        // Attache la fonction au clic sur le bouton de rafraichissement de la barre d'outils
        grid.find(".k-grid-toolbar").on("click", ".k-pager-refresh", function (e) {
            e.preventDefault();
            grid.data("kendoGrid").dataSource.read();
            toggleDeleteButtonForFamille(); // Rafraichit également l'état du bouton de suppression
        });

        // Actualise la grille après une suppression réussie
        function onSuccessfulDelete() {
            grid.dataSource.read();
            toggleDeleteButtonForFamille(); // Rafraichit l'état du bouton de suppression
        }

        // Attache la fonction au clic sur le bouton de suppression lors du chargement initial
        grid.tbody.on("click", "[data-role='delete-famille']", onDeleteButtonClick);

        // Rafraichisse la grille au chargement initial pour masquer/afficher correctement les boutons
        toggleDeleteButtonForFamille();
    });
</script>
```

Figure 10 : Index des souches

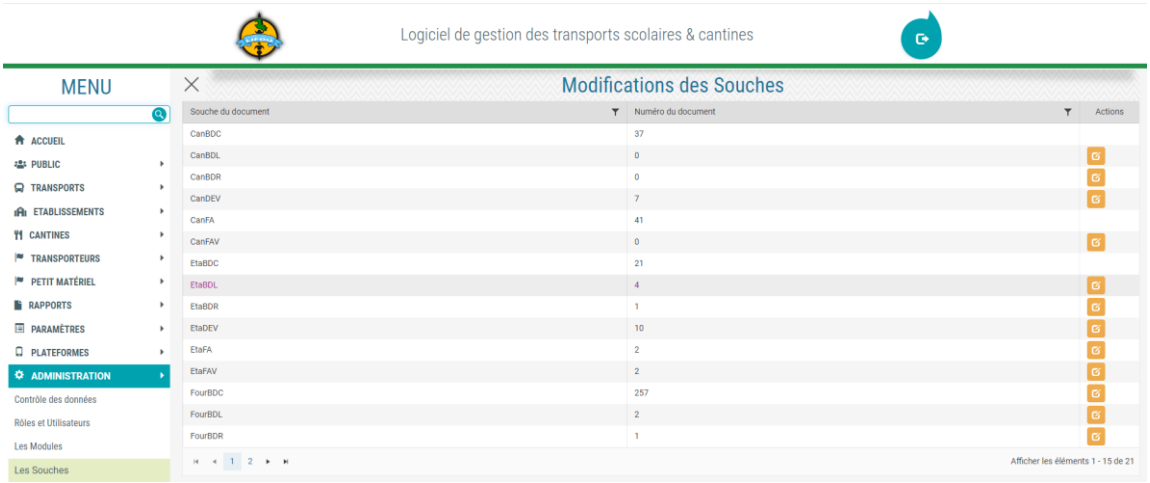


Figure 11 : Menu principal de l'application

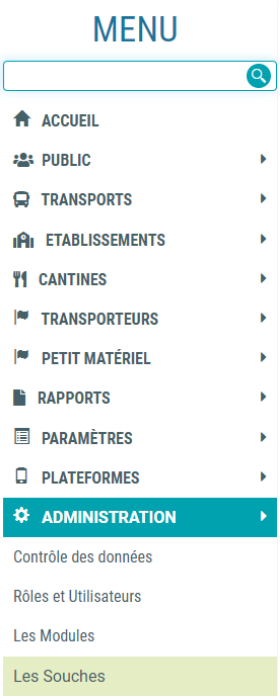
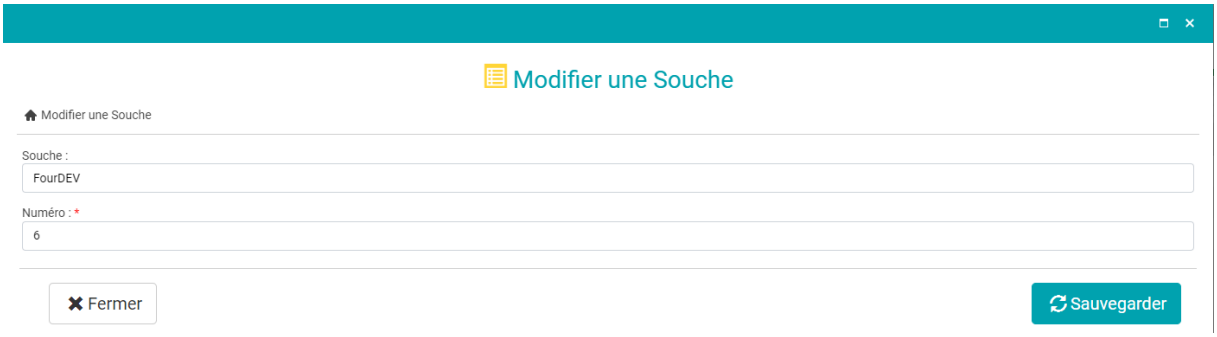


Figure 12 : Modification d'une souche



4.4 : Glossaire

- **Fil d'Ariane (Breadcrumb) :** Le fil d'ariane permet aux visiteurs de se repérer sur l'application, et ce, même s'ils sont situés dans les profondeurs du site. Ainsi, pour remonter en arrière, les internautes n'ont pas besoin d'utiliser le bouton précédent. Il leur suffit de cliquer sur les catégories du breadcrumb pour parcourir les niveaux supérieurs.

Exemple de breadcrumb dans l'application :

- **Dropdown** : Liste déroulante -> élément d'interface graphique qui permet à l'utilisateur de sélectionner une ou plusieurs options.

- **Framework** : Un framework propose une bibliothèque de fonctionnalités dans laquelle vos développeurs vont pouvoir piocher en fonction de vos besoins. En développement, l'utilisation d'un framework permet donc de gagner du temps. Aujourd'hui, il s'agit d'un standard dans la construction d'un projet web ou mobile.

- **Classe abstraite** : Une classe est abstraite si elle contient au moins une méthode abstraite ; elle ne peut pas être instanciée, mais ses sous-classes non abstraites le peuvent.

- **CRUD** : L'acronyme informatique anglais CRUD (pour Create, Read, Update, Delete)

- **DataSource** : est un composant clé qui agit comme une liaison de données entre les contrôles d'interface utilisateur (comme les grilles, les listes déroulantes, etc.) et les sources de données, telles que des tableaux JavaScript, des services web, ou même des bases de données.

- **SQL** : (Structured Query Language) langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

- **IDE** : Un environnement de développement intégré (IDE) est une application logicielle qui aide les programmeurs à développer efficacement le code logiciel.

- **URL** : Une URL (Uniform Resource Locator) couramment appelée adresse web, est une chaîne de caractères uniforme qui permet d'identifier une ressource du World Wide Web par son emplacement et de préciser le protocole internet pour la récupérer (par exemple http ou https). Elle peut localiser divers formats de données : document HTML, image, son.

- **Developpeur full-stack :** Un développeur full stack, est un développeur web capable de réaliser la programmation d'un site ou d'une application web à la fois en front-end et back-end. Il dispose ainsi de compétences variées lui permettant de travailler sur chaque étape d'un projet de création allant du développement à la production.

- **Ajax :** AJAX est une méthode utilisant différentes technologies ajoutées aux navigateurs web entre 1995 et 2005, et dont la particularité est de permettre d'effectuer des requêtes au serveur web et, en conséquence, de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète. Cette architecture informatique permet de construire des applications web et des sites web dynamiques interactifs.

- **DevTools** : DevTools est un ensemble d'outils de développement web qui apparaît à côté d'une page web rendue dans le navigateur. DevTools offre un moyen puissant d'inspecter et de déboguer des pages web et des applications web.

- **Front-End** : Le Front End ou développement web frontal correspond aux productions HTML, CSS et JavaScript d'une page internet ou d'une application qu'un utilisateur peut voir et avec lesquelles il peut interagir directement.

- **Back-End** : Le back end désigne les parties du code d'une application ou d'un logiciel permettant son fonctionnement et est inaccessibles à l'utilisateur. On le désigne aussi sous le nom de couche d'accès aux données d'un logiciel ou d'une machine. Il inclut toutes les fonctionnalités nécessitant un accès et une navigation par des moyens numériques.

- **SaaS** : Une solution dite SaaS (« Software as a Service » ou en français : « logiciel en tant que service ») est une solution logicielle applicative hébergée dans le cloud et exploitée en dehors de l'organisation ou de l'entreprise par un tiers, aussi appelé fournisseur de service. La solution SaaS est accessible à la demande via une connexion Internet.

- **Logique métier** : La logique métier est un élément très important dans la construction d'une application web ou mobile. Elle permet aux développeurs de décrire les règles et les processus qui régissent les opérations quotidiennes de l'application, telles que la gestion des utilisateurs, la sécurité et les transactions financières.

- **Types nullable** : Les types nullables sont une caractéristique de certains langages de programmation qui permettent de définir une valeur sur la valeur spéciale NULL au lieu des valeurs possibles habituelles du type de données.

- **Read-Only** : La lecture seule (Read-Only) est un état qui indique que quelque chose est inaltérable, capable d'être lu mais ne peut pas être modifié ou écrasé.

- **Débogueur** : Un débogueur ou débugueur est un logiciel qui aide un développeur à analyser les bugs d'un programme.