

Documentation LINA Web

Généralités

- [Organisation du Projet LINA Web](#)
- [Disposition des menus](#)

Conception de l'application

- [Ajout de menu / vue](#)
- [Création de ViewComponent](#)

Tableau de Bord

- [GridStack \(Tableau de bord\)](#)
- [Création de Widgets](#)

Compléments

- [DevExtreme](#)
- [Coder en Typescript](#)

Utilisation de l'application

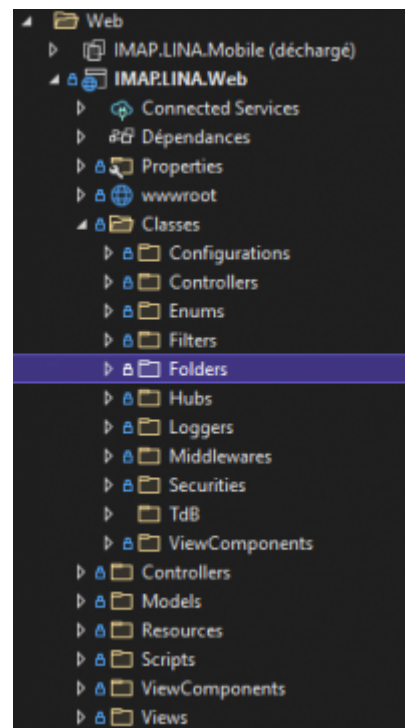
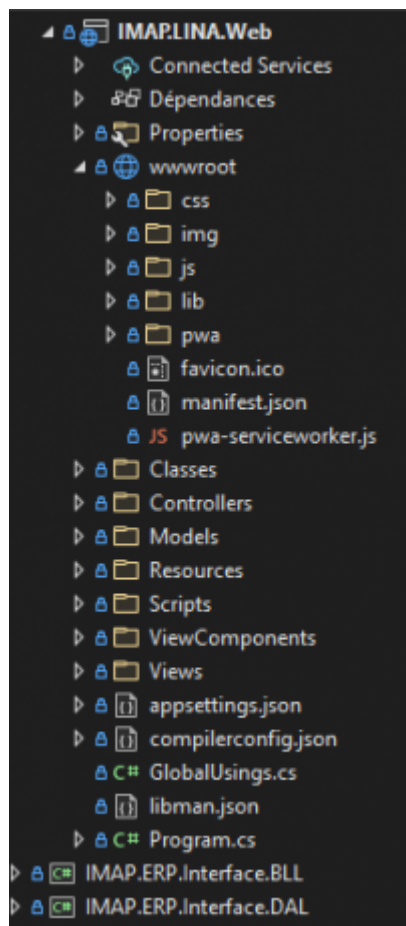
- [Tableau de Bord](#)
- [Alarmes](#)
- [Courbes](#)
- [Compteurs](#)
- [Evènements](#)
- [Planning OT](#)

Aides & ressources

- [Ressources du projet](#)

Organisation du Projet LINA Web

Arborescence globale du projet sous Visual Studio



Le dossier **wwwroot** contenant les dossiers *css*, *images*, *js* *libairies* et autres *fichiers statiques* du projet

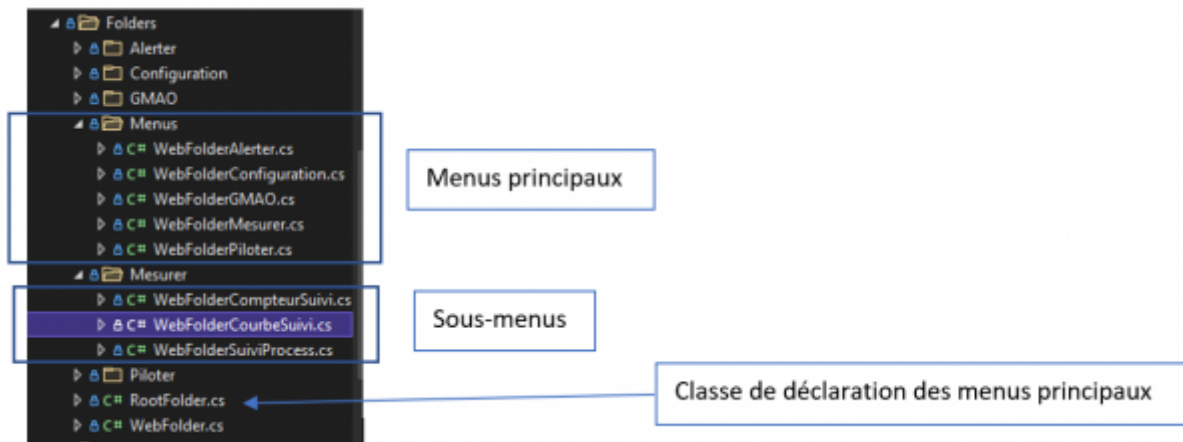
Le dossier **Classes** content les classes communes à l'application LINA Web

Le dossier **Folders** contenant les menus de la sidebar

Le dossier **Ressources** (Traduction des textes en diff. Languages)

Le dossier **Scripts** (Contient les scripts **Typescript**)

Arborescence des dossiers des menus



Fichiers de configuration

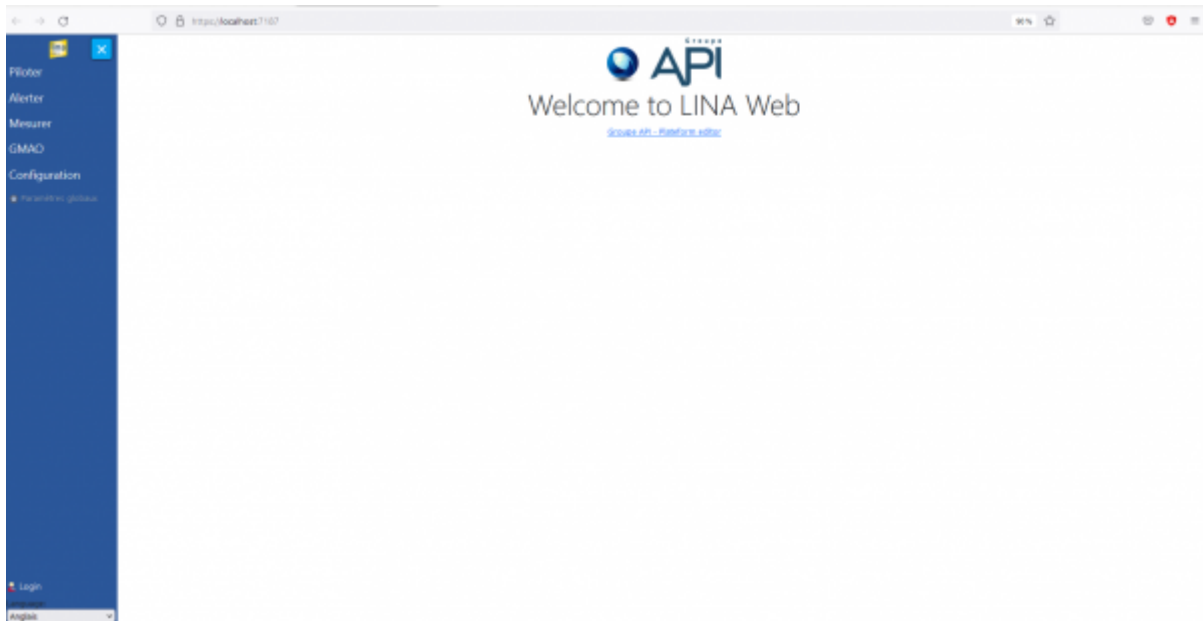
- IMAP.LINA.Web.csproj
- Program.cs
- appsettings.json
- compilerconfig.json

Disposition des menus

L'application exécute par défaut **_layout.cshtml** qui contient l'équivalent de la page de démarrage ci-dessous :

- A gauche, Sidebar avec menus de démarrage et Connexion/déconnexion
- Partie droite, Le « main » avec navbar et vue chargée (ici le home)

Nota : le home n'a pas de navbar



Ajout de menu / vue



Ex: Menu suivi des courbes

Comme dans LINA, on va créer la vue CourbeSuivi située dans le dossier Mesurer.

Création du WebFolder:

Dans le dossier `\Classes\Folders\Mesurer`, on crée **WebfolderCourbeSuivi.cs** (**Webfolder+NomVue.cs**)

```
namespace IMAP.LINA.Web
{
    public class WebFolderCourbeSuivi : WebFolder
    {
        public WebFolderCourbeSuivi()
        {
            this.FolderKey = LINADALFolderEnum.FOLDER_KEY_SUIVI_COURBE;
            this.FolderCaption =
LINADALFolderEnum.FOLDER_CAPTION_SUIVI_COURBE;
            this.FolderDescription =
LINADALFolderEnum.FOLDER_DESCRIPTION_SUIVI_COURBE;
            this.ImageKey = LINADALFolderEnum.FOLDER_IMAGE_SUIVI_COURBE;

            this.ControllerName =
GetControllerName(typeof(CourbeSuiviController));
        }
    }
}
```

On y renseigne les paramètres voulus (*FolderKey*, *FolderCaption*, *FolderDescription*, *ImageKey*) ainsi que son Controller : Ici, **CourbeSuiviController(NomVue+Controller.cs)** Que l'on crée. On prend soin de déclarer **WebfolderCourbeSuivi** dans la classe **WebFolderMesurer.cs**

```
public class WebFolderMesurer : WebFolder
{
    public WebFolderMesurer()
    {
        this.FolderKey = LINADALFolderEnum.FOLDER_KEY_MESURER;
        this.FolderCaption = LINADALFolderEnum.FOLDER_CAPTION_MESURER;
        this.FolderDescription =
LINADALFolderEnum.FOLDER_DESCRIPTION_MESURER;
        this.ImageKey = LINADALFolderEnum.FOLDER_IMAGE_MESURER;
```

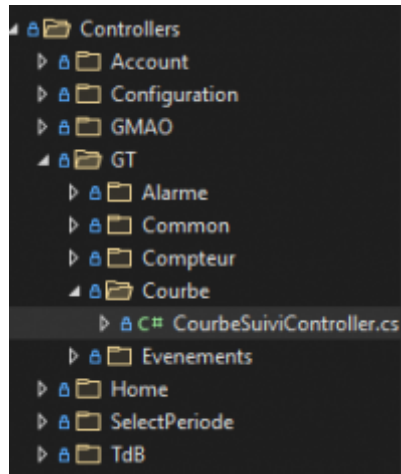
```

        this.ChildsFolder.Add(new WebFolderCourbeSuivi());
        this.ChildsFolder.Add(new WebFolderCompteurSuivi());
        this.ChildsFolder.Add(new WebFolderSuiviProcess());
    }
}

```

Création du controller:

On crée le Controller de la vue *CourbeSuivi* dans le dossier:



On y renseigne:

```

namespace IMAP.LINA.Web.Controllers
{
    public class CourbeSuiviController : LINAController
    {
        const string TEMPDATA_COURBE_SUIVI = "CourbeSuivi";

        public
        CourbeSuiviController(ArgumentControllerAgregator<CourbeSuiviController>
        pArgumentControllerAgregator) : base(pArgumentControllerAgregator)
        {

        }

        [LINAAuthorize(LINADALFolderEnum.FOLDER_KEY_SUIVI_COURBE)]
        public IActionResult Index()
        {
            SetFolderSelected();

            //Récupère le model stocké dans le tempdata
            ListArborescenceFonctSuiviModel model =
            GetTempData<ListArborescenceFonctSuiviModel>(TEMPDATA_COURBE_SUIVI);
            if (model == null) model = new
            ListArborescenceFonctSuiviModel(TypeFonction.COURBE);

            //Remplir modèle

```

```

        ArborescenceFonctSuiviDataController.InitialiserModele(ref
model);

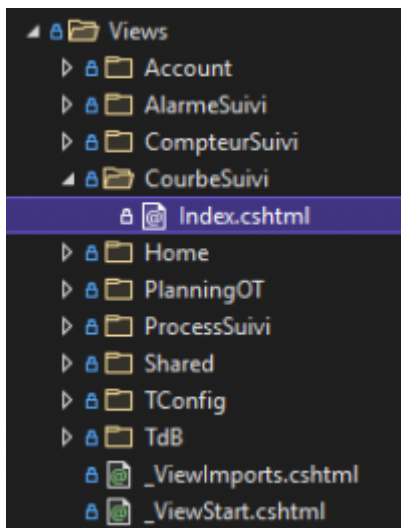
        //Afficher vue
        return View(model);
    }

    ***
}
}

```

Création de la Vue:

On crée le fichier **Index.cshtml** dans le dossier **CourbeSuivi**, sous-dossier de Views : Il se compose:



- *Modèle de données + Titre de la page (viewbag.title) + Variables globales*

```

@model ListArborescenceFonctSuiviModel

@{
    ViewBag.Title = "Suivi des courbes";
    string Chart_ID = "CourbeChart";
    string Arbo_ID = "TreeList_ArboCourbe";
    string DataGrid_ID = "CourbeDataGrid";
    string VisuTree_ID = "CourbeVisualisationTreeList";
}

```

- La section Scripts contenant les fichiers *.js (fonctions javascript utiles à la vue). **Attention ! Nous écrivons d'abord nos fonction en typescript (fichier *.ts dans le dossier Scripts\). Les fichiers .js sont automatiquement générés par VisualStudio**
- [Principe de base du Typescript](#)

```

<@section Scripts {

    @{

```

```

        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }

    <script src="~/js/Scripts/_lina_dx_generics/lina_dx_treelist.js" asp-
append-version="true"></script>
    <script
src="~/js/Scripts/TreeListArboFonctSuivi/TreeListArboFonctSuivi.js" asp-
append-version="true"></script>
    <script src="~/js/Scripts/_lina_dx_generics/lina_dx_chart.js" asp-
append-version="true"></script>
    <script src="~/js/Scripts/_lina_dx_generics/lina_dx_grid.js" asp-append-
version="true"></script>
    <script src="~/js/Scripts/CourbeSuivi/CourbeSuivi.js" asp-append-
version="true"></script>
}

```

- Code partiel de la vue

```

<div class="d-flex flex-column mx-0 h-100">
    <div class="groupbox mb-3">

        @*Sélection période*@
        <vc:Select-periode onselectedperiodechanged="DxReloadData(@Chart_ID,
@DataGrid_ID)" startaspfor="@nameof(PeriodeModel.PeriodeStart)"
stopaspfor="@nameof(PeriodeModel.PeriodeStop)"
periodeidaspfor="@nameof(PeriodeModel.PeriodeID)"
                periodestart="@Model.periode.PeriodeStart"
periodestop="@Model.periode.PeriodeStop">
            </vc:Select-periode>

        @*Sélection Arborescence thématique*@
        <vc:Select-arbo id="@Arbo_ID" idsassocies="@Chart_ID, @DataGrid_ID"
listarboaspfonct=@Model.listArborescenceFonct
arboaspid="@nameof(ArborescenceFonctSuiviModel.ArboID)"
arboparentaspid="@nameof(ArborescenceFonctSuiviModel.ArboParentID)"
        arboaspname="@nameof(ArborescenceFonctSuiviModel.ArboName)"
arboaspordpre="@nameof(ArborescenceFonctSuiviModel.ArboNumOrdre)"
arbofonctaspid="@nameof(ArborescenceFonctSuiviModel.ArboFonctionID)"
arbofonctasptype="@nameof(ArborescenceFonctSuiviModel.ArboFonctionType)">
            </vc:Select-arbo>

        @*Bouton Actualiser + Bouton Changement type de visualisation*@
        <vc:Select-visualisation
onclickactualiser="DxReloadData(@Chart_ID, @DataGrid_ID)"
id="@VisuTree_ID" idsassocies="@Chart_ID, @DataGrid_ID"
        ></vc:Select-visualisation>

        @*Bouton Sauvegarde url*@
        <vc:Qr-code></vc:Qr-code>

    </div>

```



```

<form asp-controller="ArborescenceFonctSuivi" asp-action="Selectionner">

    @*champs caché de retour des éléments sérialisés*@
    @Html.HiddenFor(x => x.TypeFonctionArbo)
    @Html.HiddenFor(x => x.SelectedSerializedArborescenceFonct)
    @Html.HiddenFor(x => x.SelectedSerializedArborescenceFonctTmp)
    @Html.HiddenFor(x => x.periode.PeriodeStart)
    @Html.HiddenFor(x => x.periode.PeriodeStop)
    @Html.HiddenFor(x => x.IntervalleUnite)
    @Html.HiddenFor(x => x.Intervalle)

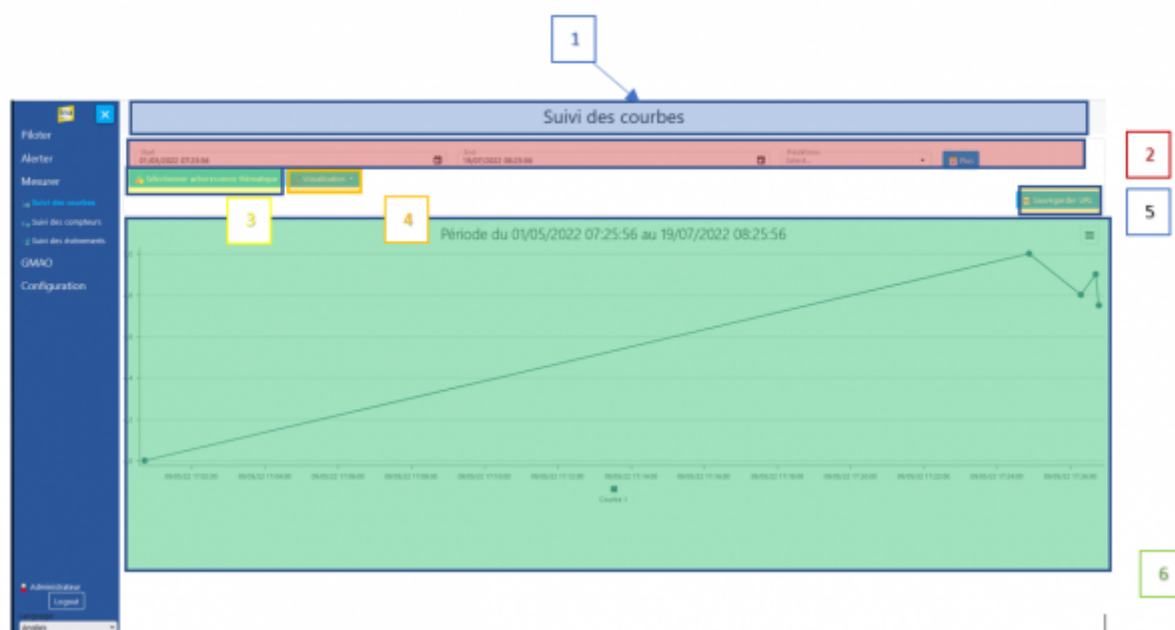
</form>

<div style="flex:1 1 0; min-height:0;">
    @*Graphique*@
    @(Html.DevExtreme().Chart()
        .....
    </div>
</div>

```

Le model de cette vue (*ListArborescenceFonctSuiviModel*) est situé dans le dossier **Models/common** car il est commun à plusieurs vues.

On obtient la vue:



1. **Navbar** (ViewBag.title)
2. ViewComponent **SelectPeriode**
3. ViewComponent **SelectArbo**
4. ViewComponent **SelectVisualisation**
5. ViewComponent **QrCode**
6. Composant DevExtreme (**DxChart** ici)

Créer un ViewComponent

Il est composé selon le modèle MVC: **Model** - **View** - Controller (**ViewComponent**)

Conventions de nommage:

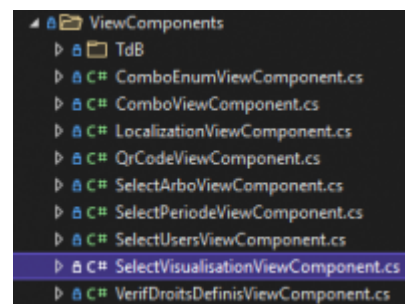
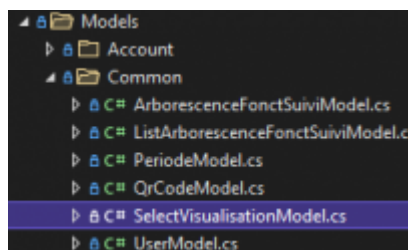
Supposons que vous vouliez créer un ViewComponent **VueX**:

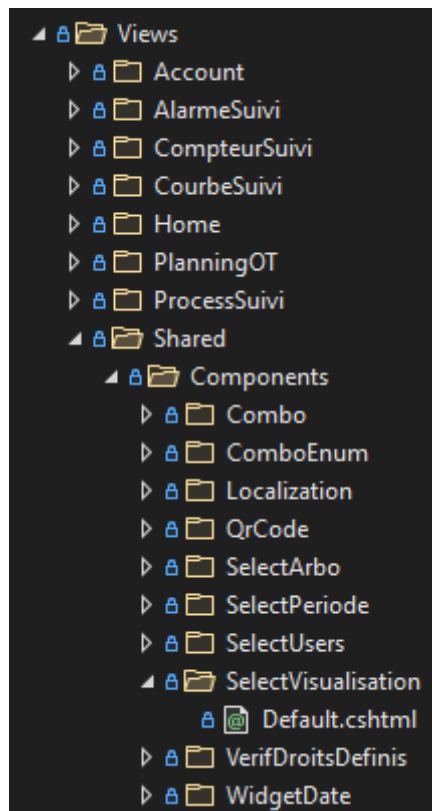
1. Le fichier de la vue devra être créé dans le dossier *Views\Shared\Components\VueX* Par défaut, on le nomme *Default.cshtml*
2. Son Fichier modèle (si nécessité) devra être dans *Models\Common* et se nommer **VueXModel.cs**
3. Le Controller de cette vue sera à créer dans *ViewComponents* et se nommer **VueXViewComponent.cs**

On appelle ce composant au moyen du tag helper

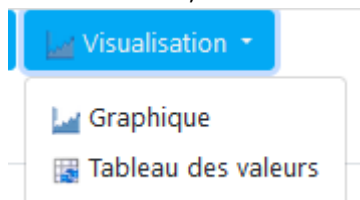
```
<vc:Vue-x (+params si nécessaire)></vc:Vue-x>
```

Exemple du Viewcomponent SelectVisualisation:





En mode web, il affiche :



- Fichier Vue **Default.cshtml**

csharp

```
@model SelectVisualisationModel

@{
    string Visualisation_ID = Model.ID;
}

<input id="@ (Visualisation_ID)_ids-associés"
name="@ (Visualisation_ID)_ids-associés" type="hidden"
value="@ (JsonConvert.SerializeObject (Model.IdsAssociés))" />

<button class="btn btn-lina dropdown-toggle" type="button"
id="dropdownMenuButton" data-bs-toggle="dropdown" aria-
expanded="true">
     Visualisation
</button>
```

```
<div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    @for (var i = 0; i < Model.IdsAssociés?.Length; i++)
    {
        string IdAssocie = Model.IdsAssociés[i].ToLower();
        string imgSrc = "";
        string text = "";
        if (IdAssocie.Contains("chart")) { imgSrc =
@IMAP.ERP.Interface.DAL.LINADALImageEnum.Graphique; text = "Graphique";
        }
        else if (IdAssocie.Contains("datagrid")) { imgSrc =
@IMAP.ERP.Interface.DAL.LINADALImageEnum.Tableau; text = "Tableau des
valeurs"; }
        else if (IdAssocie.Contains("treelist")) { imgSrc =
@IMAP.ERP.Interface.DAL.LINADALImageEnum.Arborescence; text =
"Arborescence"; }
        else if (IdAssocie.Contains("scheduler")) { imgSrc =
@IMAP.ERP.Interface.DAL.LINADALImageEnum.Calendrier; text =
"Calendrier"; }
        <a class="dropdown-item" href="#" id="action@(i)"
onclick="DxShowElement(@i, '@(Visualisation_ID)_ids-associés')">
             @text</a>
    }
</div>
```

- Fichier **SelectVisualisationModel.cs**

```
namespace IMAP.LINA.Web.Models
{
    public class SelectVisualisationModel
    {
        public string ID { get; set; } = string.Empty;
        public string OnClickActualiser { get; set; } = string.Empty;
        public string[]? IdsAssociés { get; set; } = null;
    }
}
```

- Controller **SelectVisualisationViewComponent.cs**

```
using IMAP.LINA.Web.Models;
using Microsoft.AspNetCore.Mvc;

namespace IMAP.LINA.Web.Controllers
{
    public class SelectVisualisationViewComponent : LINAViewComponent
    {
        public
        SelectVisualisationViewComponent(ArgumentControllerAgregator<SelectVisualisa
tionViewComponent> pArgumentControllerAgregator) :
        base(pArgumentControllerAgregator)
```

```
    {  
    }  
  
    public IViewComponentResult Invoke(string onclickactualiser, string  
id, string idsassocies = "")  
    {  
        SelectVisualisationModel model = new() {  
            OnClickActualiser = onclickactualiser,  
            ID = id,  
            IdsAssocies = idsassocies.Replace(" ", "").Split(","),  
        };  
        return this.View(model);  
    }  
}
```

Pour appeler le viewcomponent dans la vue souhaitée, on tape:

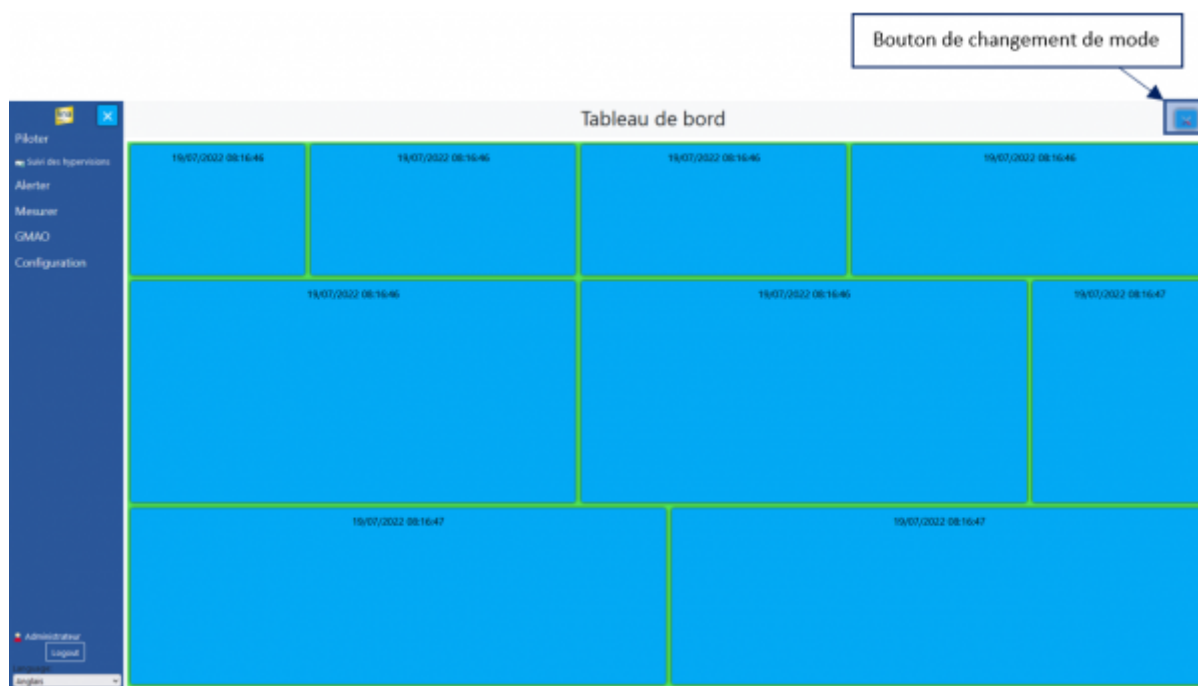
```
<vc:Select-visualisation  
    onclickactualiser="DxReloadData(@Chart_ID, @DataGrid_ID)"  
id="@VisuTree_ID" idsassocies="@Chart_ID, @DataGrid_ID"  
></vc:Select-visualisation>
```

Tableau de Bord

Mode Visualisation

(En cours de réalisation)

Par défaut, On a la visualisation du tableau de bord avec les widgets personnalisés défini par l'utilisateur:



Contenu partiel du fichier **_layout.cshtml** (Code du Bouton Changement de mode) :

```
@*Main*@
<div id="main" class="d-flex flex-column col px-0 max-vh-100
justify-content-between" style="overflow-y:auto;">
    @*Menu supérieur*@
    @if (ViewBag.Title=="Tableau de bord")
    {
        <nav class="d-block navbar navbar-expand navbar-light
bg-light d-block" style="flex:0 1 0;">
            <div class="container-fluid">
                <h1 class="container-fluid row justify-content-
center">@ViewBag.Title</h1>
                @*Définition du bouton de choix de type de
visualisation (Edition ou Consultation)*@
                @if (Model.Mode == WidgetDisplayMode.Display)
                {
                    <a class="btn btn-link m-2 visualisation-mode-
change data-bs-toggle="tooltip" data-bs-placement="left" title="Passer en
Mode Edition" asp-
controller="@WebFolder.GetControllerName(typeof(TdBController))" asp-
```

```

action="@nameof(TdBController.Design)">
    
    </a>
}
else
{
    <a class="btn btn-lina visualisation-mode-change
data-bs-toggle="tooltip" data-bs-placement="left" title="Passer en Mode
Visualisation" asp-
controller="@WebFolder.GetControllerName(typeof(TdBController))" asp-
action="@nameof(TdBController.Index)">
        
    </a>
}
</div>
</nav>
}
else{
@if (!string.IsNullOrEmpty(ViewBag.Title))
{
    <nav class="d-block navbar navbar-expand navbar-light
bg-light d-block" style="flex:0 1 0;">
        @*<button class="btn btn-lina m-2 float-right"
onclick="popupSidebar()">&#9776;</button>*&
        <h1 class="text-center">@ViewBag.Title</h1>
    </nav>
}
}
@*Body*@
<div class="d-block p-2" style="flex:1 1 0;min-height:0;">
    @*<hr />*&
    @RenderBody()
</div>
</div>

```

Mode Edition:

(En cours de réalisation)

En Design Mode, la sidebar Toolbox apparait à droite ainsi que les widgets paramétrables sur le tableau de bord:



Code partiel de **TdB\Index.cshtml** (Sidebar toolbox à droite):

```
<div>
.....
@*Définition des boutons composant la boite à outils*@
<div id="menu" class="sideDashboard row mx-0 px-2">
    <div class="col-12 px-0 pl-2">
        <div class="col-12">
            <button id="trash" class="btn btn-lina-danger-toolbox ui-
droppable mb-2 py-2 px-4"
                data-bs-toggle="tooltip" data-bs-placement="left"
                title="Déplacer un widget ici pour le supprimer">
                 Corbeille
            </button>
        </div>
        <div class="col-12">
            <button class="text-center btn btn-lina-danger-toolbox mb-2 mx-0
py-2 px-4" onclick="gridstackRemoveAll()">
             Tout effacer
            </button>
        </div>
        <div class="col-12">
            <div id="gridstack-widget-date" class="newWidget grid-stack-
item">
                <div class="grid-stack-item-content ui-draggable-handle btn
btn-lina-toolbox mb-2 py-2 px-4">
                    <div>
                        
                        <span class="pl-1"> <vc:Widget-date
mode="@Convert.ToInt32(WidgetDisplayMode.ToolBox)" config="" /></span>
```



```

        </div>
    </div>
</div>
<div id="gridstack-widget-courbe" class="newWidget grid-stack-
item">
    <div class="grid-stack-item-content ui-draggable-handle btn
btn-lina-toolbox mb-2 py-2 px-4">
        <div>
            
            <span class="pl-1"> <vc:Widget-courbe
mode="@Convert.ToInt32(WidgetDisplayMode.ToolBox)" config="" /></span>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
    @*Bouton de toggle:Cacher ou Afficher la toolbox*@
    <button class="btn btn-lina btn-lg m-2 sidebarDashboard-toggle"
onclick="popupSidebarDashBoard()" alt="sidebarButtonToggle">
        <span></span><span></span><span></span><span></span><span></span>
    </button>
</div>

```

Création de Widgets

Beaucoup de code a changé depuis que cette page a été écrite !

Mode Display, Design, ToolBox

Chaque Widget est un ViewComponent qui contient trois parties : Une pour l'affichage du tableau de bord, Une pour l'édition du tableau de bord, Une pour l'affichage du "bouton" à glisser dans le tableau de bord.

Par exemple :

```
@if (Model.Mode == WidgetDisplayMode.Design)
{
    @* Code pour afficher la partie Edition *@
}
```

WidgetDisplayMode vous permet de savoir dans quel mode vous êtes (Display, Design, Toolbox).

Dans tous les cas, un élément est requis pour avoir les informations du widget. Voici un exemple pour l'élément WidgetDate :

```
<input type="hidden" id="@strGuid"
    data-widget-mode="@Model.Mode"
    data-widget-type="widget-date"
    data-widget-guid="@strGuid"
    data-widget-freq="@Model.Frequence"
    data-widget-couleur-text="@Model.CouleurText"
    data-widget-couleur-fond="@Model.CouleurFond"/>
```

Un certain nombre d'informations de base sont passées par cet élément ;

- data-widget-type: Le type de widget, dans cet exemple, @WidgetModel.DataWidgetTypeDate renvoie "widget-date"
- data-widget-guid: L'ID unique du ViewComponent, renseigné en haut de page comme ceci :

```
@{ string strGuid = string.IsNullOrEmpty(Model.Guid) ?
    $"guid{Guid.NewGuid().ToString()}" : Model.Guid; }
```

Le Guid peut potentiellement commencer par un chiffre, ce qui posera problème dans le code, soyez donc sûr

- data-widget-freq: La fréquence de mise à jour du widget (quand l'action controler-name/action-name est appelée), en millisecondes
- data-widget-couleur-text: La couleur principale du texte du widget (chaque élément du

widget peut avoir sa propre couleur)

- `data-widget-couleur-fond`: La couleur de fond du widget

Chaque widget pourra avoir ses propres données stockées ici. Par exemple, le widget `WidgetCourbe` a une autre propriété, `data-widget-courbe-id="@Model.CourbeID"`, qui est nécessaire pour savoir d'où partir de quelle courbe les données seront récupérées.

Ajout d'un Widget dans la Toolbox

Dans la liste de widgets de votre Toolbox (div, ul > li, ...), insérez un nouveau widget comme suit :

```
<div class="newWidget grid-stack-item">
  <div class="grid-stack-item-content ui-draggable-handle btn btn-lina-
  toolbox mb-2 py-2 px-4">
    <div>
      
      <span class="pl-1"> <vc:Widget-date
      mode="@Convert.ToInt32(WidgetDisplayMode.ToolBox)" config="" /></span>
    </div>
  </div>
</div>
```

Evidemment, plusieurs choses peuvent être modifiées selon vos besoins, mais certaines choses restent indispensables. La première `<div>` nécessite les classes `newWidget` et `grid-stack-item`. La deuxième `<div>` nécessite la class `grid-stack-item-content`. A partir de là, l'intérieur du HTML dépend de vous ; mais un `ViewComponent` sera requis. Chaque élément devra avoir une image à gauche du texte et un titre clair.

Le ViewComponent

Voici un simple exemple de `ViewComponent` qui permettra d'afficher un widget de type `WidgetDate`:

```
@model WidgetDateModel
@{
    string strCaption = "Date/Heure";
    string strGuid = string.IsNullOrEmpty(Model.Guid) ?
    $"guid{Guid.NewGuid().ToString()}" : Model.Guid;
}

<input type="hidden" id="@strGuid"
  data-widget-mode="@Model.Mode"
  data-widget-type="widget-date"
  data-widget-guid="@strGuid"
  data-widget-freq="@Model.Frequence"
  data-widget-couleur-text="@Model.CouleurText"
  data-widget-couleur-fond="@Model.CouleurFond"/>
```

```

@if (Model.Mode == WidgetDisplayMode.ToolBox)
{
    <label class="mb-0">@strCaption</label>
}
@if (Model.Mode == WidgetDisplayMode.Design)
{
    @* Le contenu du widget en mode Design, avec son modal pour les
    paramètres... *@
}
@if (Model.Mode == WidgetDisplayMode.Display)
{
    @* Le contenu du widget en mode Display... *@
}

```

L'élément avec toutes les informations du widget (son type, guid, fréquence, ...) **doit avoir l'id "@strGuid"**. Tous les éléments générés ayant un id devraient avoir @strGuid aussi, de façon à ce que chaque id de chaque widget existant plusieurs fois sur le même tableau de bord aie un id différent.

Personnaliser un Widget

Si vous voulez créer un widget qui n'existe pas encore, plusieurs choses sont indispensables. Après avoir créé le ViewComponent et avoir rajouté le "bouton" dans la boîte à outils, il faudra faire certaines modifications et certains ajouts au Typescript.

Premièrement, du côté de TdB/TdB.ts :

Dans les enums en début de fichier, trouvez TdBWidgetType et ajoutez-y la valeur désirée. NouveauWidget = "widget-nouveau" sera utilisé pour cet exemple. Dans la fonction CreateWidget:

```

// Récupérer le nom de la classe à utiliser en fonction du type de widget
const widgetType: string = pWidgetElement.getAttribute(TdBWidgetData.Type);
let className!: string;
switch (widgetType) {
    case TdBWidgetType.Date: className = "WidgetDate"; break;
    case TdBWidgetType.Courbe: className = "WidgetCourbe"; break;
    default: className = "WidgetBase"; break;
}

```

A l'intérieur du switch, il faudra rajouter une ligne case TdBWidgetType.NouveauWidget: className = "WidgetNouveau"; break;.

Maintenant, il faudra créer une nouvelle classe WidgetNouveau dans un autre fichier (/TdB/WidgetNouveau.ts par exemple) :

```

class WidgetNouveau extends WidgetBase { }

```

Votre Widget est maintenant opérationnel, et hérite correctement de WidgetBase, qui a toutes les fonctions de base requises. Mais ce que l'on vient de faire ne sert à rien si vous ne personnalisez pas

votre widget un peu plus...

Ajout de paramètres personnalisés

Chaque élément de type `HTMLInputElement` (`<input />`) du `ViewComponent` avec la class `widget-param` sera lu dans la fonction `UpdateParams` de la classe `WidgetBase`. Ces éléments ne sont utilisés que lors de l'édition un widget, inutile donc de les placer dans la partie `ToolBox` ou `Display`. Voyons un exemple avec le widget `NouveauWidget` (Une partie du code a été retirée pour éviter perdre de la place sur cette page)...

```
@model WidgetNouveauModel
@{
    string strGuid = string.IsNullOrEmpty(Model.Guid) ?
    $"guid{Guid.NewGuid().ToString()}" : Model.Guid;
    string strInputFreqId = "txtFreq" + strGuid;
    string strInputCouleurTextId = "txtCouleurText" + strGuid;
    string strInputCouleurFondId = "txtCouleurFond" + strGuid;
}

<input type="hidden" id="@strGuid"
    data-widget-mode="@Model.Mode"
    data-widget-type="widget-date"
    data-widget-guid="@strGuid"
    data-widget-freq="@Model.Frequence"
    data-widget-couleur-text="@Model.CouleurText"
    data-widget-couleur-fond="@Model.CouleurFond"/>

@if (Model.Mode == WidgetDisplayMode.Design)
{
    /* Un champs input stockant la fréquence de rafraichissement du widget */
    <label asp-for="Frequence" class="control-label"></label>
    <input class="form-control widget-param widget-freq"
value="@Model.Frequence" valueold="@Model.Frequence" />

    /* Les champs input stockant la couleur du texte et du fond du widget,
    respectivement.
    Ceux-ci sont cachés (type="hidden") ; dans le code original, c'est
    grâce à un élément DxColorBox que l'utilisateur choisit ces valeurs */
    <label asp-for="CouleurText" class="control-label"></label>
    <input type="hidden" class="form-control widget-param widget-couleur-text"
value="@Model.CouleurText" valueold="@Model.CouleurText" />

    <label asp-for="CouleurFond" class="control-label"></label>
    <input type="hidden" class="form-control widget-param widget-couleur-fond"
value="@Model.CouleurFond" valueold="@Model.CouleurFond" />
}
```

A noter que dans ce code, les classes `widget-freq`, `widget-couleur-text` et `widget-couleur-fond` ont été ajoutés pour que le code Javascript derrière sache à quel paramètre chaque widget-

param correspond. Cela peut aussi être effectué par des IDs uniques, mais il faudra dans tous les cas le préciser dans la fonction UpdateParam de votre widget, comme ceci :

```
UpdateParam(elt: HTMLInputElement | HTMLSelectElement): void {
    super.UpdateParam(elt);
    if (elt.classList.contains(TdWidgetParams.Courbe)) // => Vérifier si
        l'élément a la classe "widget-courbe"
        this.DataElement.setAttribute(TdWidgetData.CourbeID, elt.value); //
        Attribuer la valeur saisie dans le paramètres "data-widget-courbe-id"
}
```

Comme vous pouvez le voir, le code ci-dessus utilise les enum TdWidgetParams et TdWidgetData. Ajoutez simplement la valeur voulue à ces enums avec leur nom de paramètres. Vous devriez aussi préciser un nouveau type dans l'enum TdWidgetType, et vérifier son utilisation dans le code Typescript pour être sûr que votre type de widget est bien reconnu par le code.

```
enum TdWidgetType {
    Date = "widget-date",
    Courbe = "widget-courbe",
    NouveauWidgetType = "widget-nouveau-type" // Ajout du nouveau type dans
    l'enum
}

// Utilisation de cet enum
CreateWidget(pWidgetElement: HTMLElement): WidgetBase {

    // Récupérer le nom de la classe à utiliser en fonction du type de widget
    const widgetType: string =
pWidgetElement.getAttribute(TdWidgetData.Type);
    let className!: string;
    switch (widgetType) {
        case TdWidgetType.Date: className = "WidgetDate"; break;
        case TdWidgetType.Courbe: className = "WidgetCourbe"; break;
        case TdWidgetType.NouveauWidgetType : className = "WidgetNouveauType";
break; // < Nouveau type avant "default"
        default: className = "WidgetBase"; break;
    }
    // Créer le widget en utilisant new window["nomDeLaClasse"], qui revient à
    faire new window["TdWidget"], équivalent à new TdWidget, par exemple
    const widget: WidgetBase = new window[className](this, pWidgetElement);
    return widget;
}
```

Le type permet de savoir quelle classe Javascript sera créée, grâce à cette précision le code créera une nouvelle instance de la classe WidgetNouveauType, qui héritera de toutes les propriétés de la classe WidgetBase. A partir de là, vous pouvez personnaliser toutes les fonctions de base et en rajouter de nouvelles.

```
class WidgetNouveauType extends WidgetBase {
    NomDeFonctionParente(): void {
        // Appel de la fonction parente
    }
}
```

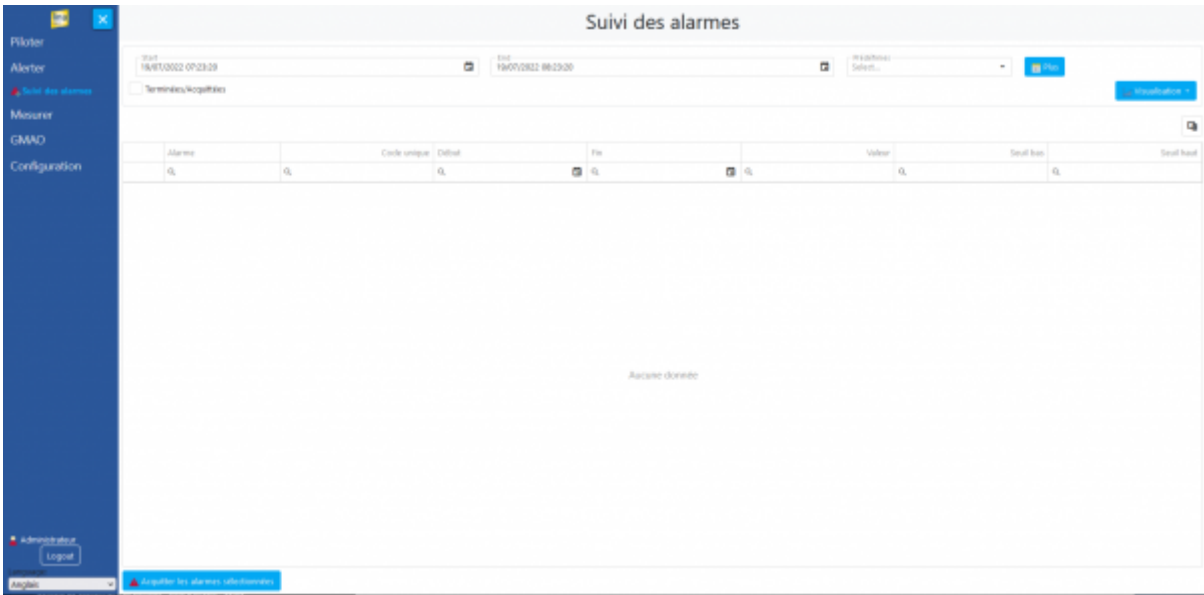
```
// super() suffit dans la fonction constructor()
super.NomDeFonctionParente();
// Appel d'une nouvelle fonction
this.NouvelleFonction();
}
NouvelleFonction(): void {
    // Code de la nouvelle fonction
}
}
```

Pour personnaliser un Widget à sa création, utilisez la fonction InitWidget :

```
InitWidget() {
    // Appel de la fonction originale !! REQUIS !!
    super.InitWidget();
    // Empêcher le widget d'avoir une hauteur et largeur inférieure à 2 cases
    (sur 12)
    this.GSAttrib.Set("MinW", 2, true);
    this.GSAttrib.Set("MinH", 2, true);
    // Changer les valeurs Frequence, CouleurText et CouleurFond par défaut
    this.Model.Frequence = 2;
    this.Model.CouleurText = "#000000";
    this.Model.CouleurFond = "#ffffff";
}
```

Alarmes

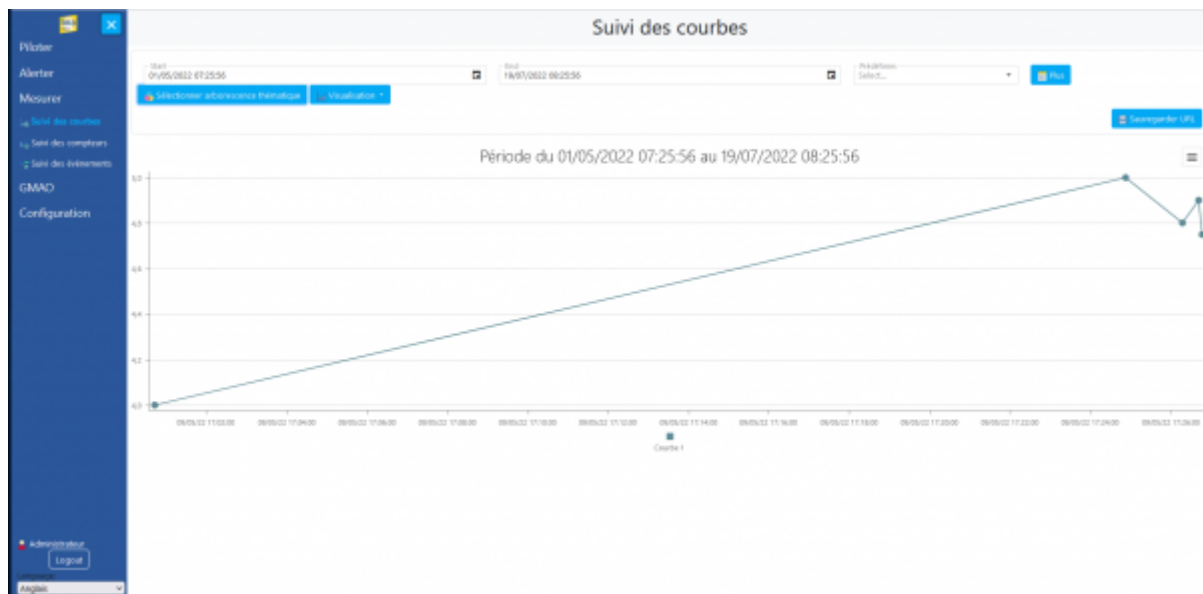
- Suivi des Alarmes



(En cours)

Courbes

- Suivi des Courbes



Traité sur la page "Conception de l'application"

Compteurs

- Suivi des Compteurs

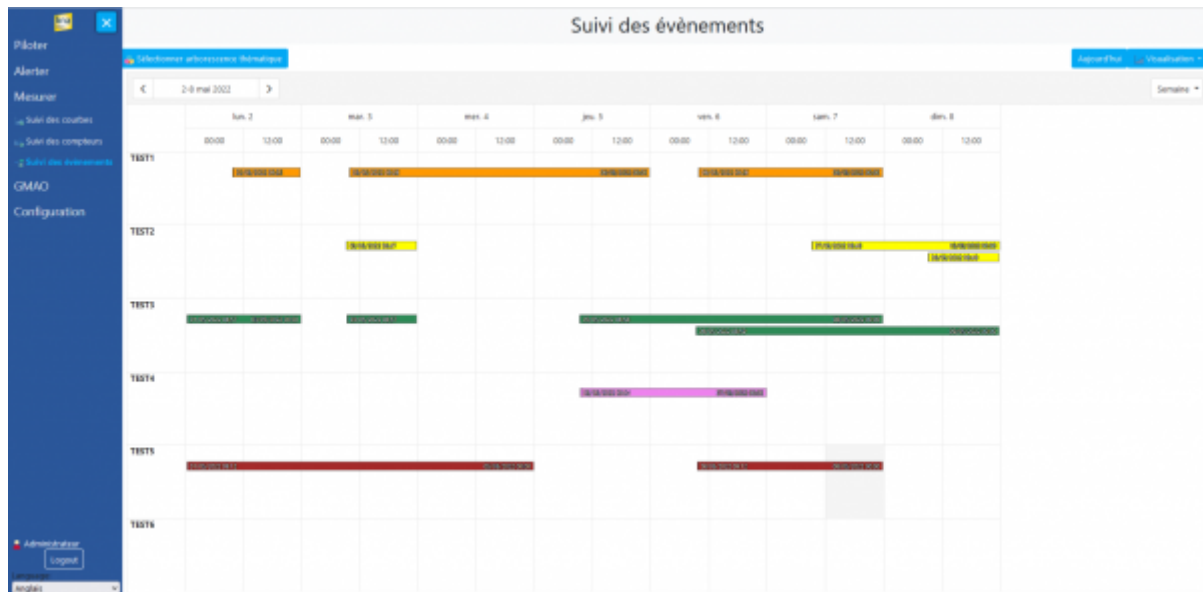


La page est similaire à celle "*Suivi des Courbes*" hormis que le composant DevExtreme appelé est DxChart en **bar** et non **line**

(En cours)

Evènements

- Suivi des Evènements



(En cours)

DevExtreme

Liens utiles

- [Coder en Typescript](#)
- [DevExtreme et le Typescript](#)

En cours...

Principe de base

Chaque fichier typescript (.ts) est converti en javascript (.js) lors de l'exécution ou la sauvegarde du projet Web dans `wwwroot/js/...`

Typescript permet de typer les variables, et a quelques facilités de syntaxe comparé au javascript dont il reste néanmoins assez proche.

Par exemple, Typescript permet l'utilisation de ? :

```
window[ID]?.OnInitialized();
```

qui, une fois transformé en javascript, devient :

```
var _a;  
(_a = window[ID]) === null || _a === void 0 ? void 0 : _a.OnInitialized();
```

C'est plus simple à lire et écrire.

Génération automatique de .cs.ts

Un simple `[TSCompilerInclude]` avant une classe en C# suffit à créer un fichier au format Typescript qui reprendra tous les attributs de celle-ci, une fois l'extension Typescript Definition Generator installée.

Par exemple, un fichier `ExempleModel.cs` comme suit :

```
[TSCompilerInclude]  
class ExempleModel  
{  
    public int? intExemple { get; set; } = null;  
    public string strExemple { get; set; } = string.Empty;  
}
```

générera un fichier `ExempleModel.cs.ts` qui ressemblera à :

```
class ExempleModel {  
    intExemple?: number;  
    strExemple: string;  
}
```

Il sera ensuite converti en `ExempleModel.cs.js` dans `wwwroot/js/Models/...`

Syntaxe

```
// extends remplace : de C#  
class UneClasse extends UneClasseParente {
```

```
// Cette variable ne pourra que contenir une Array de strings, mais ne
sera pas initialisée tout de suite.
// Quand une variable n'est pas initialisée, elle a la valeur undefined,
qui est différent de null.
// Par convention, les variables privées ont un _ avant leur nom.
private _variablePrivee!: Array<string>;
// Cette variable est initialisée en tant que boolean avec la valeur
false, et est publique.
public variablePublique: boolean = false;
public id: number;

constructor(id: number) {
    // super(arguments) appelle le constructeur de la classe parente, ici
    UneClasseParente.
    // Hors du constructeur, il faudra appeler
    super.nomDeFonction(arguments).
    super(id);
    this.id = id;
    // ...
}

// La variable _variablePrivee peut devenir accessible de l'extérieur
public get variablePrivee() {
    return this._variablePrivee ;
}
public set variablePrivee(value: Array<string>) {
    this._variablePrivee = value;
}

// Cette fonction ne renvoie rien, mais reçoit un argument arg de type
any, qui est un type "générique"
public nomDeFonction(arg: any): void {
    // ...
}

// La notion des fonctions static existe aussi en Typescript
public static nomDeFonctionStatique(arg: any): number {
    // On peut forcer le type d'une variable, mais aussi faire un cast,
    comme (int) en C# par exemple :
    const nombre: number = <number>arg;
    // ou
    const _nombre: number = arg as number;
    // Attention à ce que les types soient compatibles, forcer un objet à
    devenir un nombre risque de poser des problèmes...
    return typeof(nombre) === "number" ? nombre : 0;
}
}
```

Dans ce code, on peut voir que la ligne `private _variablePrivee!: Array<string>;` génère une variable avec comme valeur par défaut `undefined`. Il est important de bien faire la différence entre `null` et `undefined`.

```
var variable!: string;
console.log(variable, typeof(variable)); // => undefined, "undefined"
console.log(variable === null); // => false
console.log(variable == null); // => true
console.log(variable?.length); // => undefined
console.log(variable.length); // => Uncaught TypeError: variable is
undefined
```

Le javascript a tendance à être très généreux sur les types, attention donc à ne pas oublier de mettre `===` pour une comparaison de valeur et de type. Attention également aux additions de valeurs, `"1" + 1` ne renverra pas une erreur mais `"11"`, par exemple.

Références

Un fichier Typescript a parfois du mal à comprendre les fichiers référencés que l'on voudrait utiliser. Pour cela, nous pouvons utiliser cette commande :

```
/// <reference path="cheminVersFichier" />
```

Le chemin du fichier est relatif à l'emplacement du fichier sur lequel vous travaillez actuellement.

Il n'est pas possible d'utiliser `"~"` pour aller à la source du projet par défaut.

`path="../../fichier.ts"` cherchera `fichier.ts` dans le dossier parent du dossier parent de celui dans lequel se trouve votre fichier actuel. Une fois le bon fichier renseigné, Typescript comprendra le code qu'il référence, et pourra donc vous donner plus d'indications sur ce qu'il fait, et retirer des erreurs par la même occasion.

Parfois, malgré une référence correcte, Typescript n'arrive pas à trouver la référence qu'il nous faut. Si vous voulez malgré tout utiliser cette référence, vous ne pourrez pas profiter des suggestions de code de Visual Studio et si vous ne faites pas comme suit, vous aurez une erreur.

```
// @ts-ignore
var grid!: GridStack;
```

Nous utilisons actuellement la version ES5 de Javascript, mais si nous passons à la version ES6 ou supérieure, ces références ne devraient plus être nécessaires. Il faudrait utiliser `import` à la place.

DevExtreme et le Typescript

Liens utiles

- [DevExtreme](#)
- [Coder en Typescript](#)
- [Documentation DevExpress](#)

Scripts/_lina_dx.ts contient toutes les fonctions de base ainsi que la classe parente à tout élément DevExtreme compatible.

Le but de passer par Typescript pour les éléments DevExtreme est simple : utiliser une classe par élément qui héritera des fonctions et variables de ses parents. Ainsi, plutôt que d'avoir une page cshtml avec plein de fonctions différentes appelées, il suffira de faire quelque chose comme :

```
@(Html.DevExtreme().DataGrid()  
    .ID("ExempleDataGrid")  
    .OnInitialized("OnInitialized")  
)
```

Il faudra bien sûr rajouter les informations nécessaires pour charger des données, notamment grâce à .DataSource(ds => ds.Mvc()OnBeforeSend("ExempleDataGrid_BeforeSend")). Le code ci-dessus suffit cependant pour initialiser une nouvelle instance d'une classe de type DxDataGrid, héritant de Scripts/_lina_dx_generics/_lina_dx_grid.ts. Il vous faudra créer une classe dans un nouveau fichier Typescript, que l'on appellera ExempleDataGrid pour cet exemple.

```
/// <reference path="../../../lina_dx.ts"/>  
/// <reference path="../../../_lina_dx_generics/_lina_dx_grid.ts"/>  
  
/** Avant d'envoyer la requête Get au webservice */  
function ExempleDataGrid_BeforeSend(operation, ajaxSettings) {  
    DxGetVar("ExempleDataGrid").OnBeforeSend(operation, ajaxSettings);  
}  
  
class ExempleDataGrid extends DxDataGrid {  
  
    OnBeforeSend(operation: any, ajaxSettings: any): void {  
        // ajaxSettings.data. ...  
    }  
}
```

IMPORTANT : Le nom de la classe doit être EXACTEMENT le même que l'ID que vous avez donné à votre élément DevExtreme dans la partie HTML. .ID("ExempleDatagrid") cherchera la classe ExempleDatagrid, qui n'existe pas ; le G est en majuscule.

Une fois que l'ID et la classe sont créés, OnInitialized("OnInitialized") suffit à tout mettre en route ; il créera une instance de la classe ID pour vous et vous aurez accès à toutes les fonctions des classes parentes associées (DxClass et DxDataGrid, par exemple).

Plein de fonctions sont disponibles ([Voir liste non exhaustive](#)). Pour plus d'informations, rendez-vous sur [la documentation DevExpress](#) ; vous y trouverez toutes les options disponibles pour chaque

élément. Si vous souhaitez ajouter une fonction qui n'est pas présente dans la liste des options, il vous suffit d'ouvrir le fichier correspondant à votre type d'élément DevExtreme. Pour cette exemple, nous allons ajouter `onCellClick` aux options de `DxDataGrid`. [La documentation de cette option](#) vous donne certains détails sur son utilité et quand elle est appelée. Dans `Scripts/_lina_dx_generics/lina_dx_grid.ts`, comme pour toutes les classes génériques DevExtreme, vous trouverez le bloc suivant :

```
if (this.Instance !== undefined) {
  this.Instance.option({
    // options
  });
}
```

Modifions le afin d'ajouter l'option `onCellClick`, et ajoutons la fonction liée.

```
class DxDataGrid extends DxClass {
  constructor(ID: string) {
    super(ID, "datagrid");

    // Options par défaut pour toutes les classes héritant de DxDataGrid
    if (this.Instance !== undefined) {
      this.Instance.option({
        // nomOptions: Fonction, ...
        // Attention aux majuscules !
        onCellClick: OnCellClick
      });
    }
  }

  // Comme on peut le voir dans la documentation, onCellClick prend un
  // argument qui stockera plusieurs informations pour nous
  OnCellClick(arg: any): void {
    // On peut ensuite stocker et modifier les éléments stockés dans arg
    let cellElement: HTMLElement = arg.cellElement;
    // cellElement. ...
  }
}
```

Et c'est aussi simple que ça! Pas besoin de rajouter l'option `OnCellClick` dans la partie HTML de DevExtreme, la classe s'en chargera elle-même.

A noter

Il n'est pas possible de faire ceci (notez le `.`) :

```
@(Html.DevExtreme().DataGrid()
  .ID("ExempleDataGrid")
  .UneOption("ExempleDataGrid.UneOption")
)
```

Si vous avez besoin d'appeler l'instance de classe, faites comme suit (notez le `_`) :

```
@(Html.DevExtreme().DataGrid()  
  .ID("ExempleDataGrid")  
  .UneOption("ExempleDataGrid_UneOption")  
)
```

ExempleDataGrid_UneOption n'est qu'un exemple, la fonction peut s'appeler comme vous le désirez.

Il vous faudra ensuite créer la fonction associée, et la renvoyer vers la fonction de l'instance de classe désirée :

```
function ExempleDataGrid_UneOption(arguments) {  
  DxGetVar("ExempleDataGrid").UneOption(arguments);  
}
```

arguments dépend de l'option.

GridStack (Tableau de bord) et le Typescript

Liens utiles

- [Read Me GridStack](#)
- [Documentation GridStack](#)
- [Démonstrations GridStack](#)
- [Coder en Typescript](#)
- [Tableau de bord](#)
- [Création de Widgets](#)

Tout le code Typescript concernant le Tableau de bord se trouve dans Scripts/TdB.

Les Models générés automatiquement (voir [Génération automatique de .cs.ts](#)) se trouvent quant à eux dans Models/TdB.

**Les appels à la base de données se font via le controller
Controllers/TdB/TdBController.cs**

Tableau de bord

TdB.ts contient toutes les fonctions et variables nécessaires à la création et la mise à jour du Tableau de bord.

En cours...

Widgets

En cours...

[Création de Widgets](#)

Ressources du Projet

- **ASP.NET MVC**

<https://www.tutorialsteacher.com/mvc>

<https://docs.microsoft.com/fr-fr/aspnet/core/?view=aspnetcore-6.0>

- **DevExpress**

<https://demos.devexpress.com/ASPNetMvc/>

- **TypeScript**

<https://www.typescriptlang.org/docs/>

<https://geekflare.com/fr/typescript-vs-javascript/>

- **Bootstrap**

<https://getbootstrap.com/docs/5.2/getting-started/introduction/>

- **Gridstack**

<https://gridstackjs.com/>

- **Scss / css**

<https://developer.mozilla.org/fr/docs/Web/CSS>

Charte de codage

Règles de syntaxe

- Préfixe 'm' pour les membres privés globaux à une classe

```
BLLSyncSite mBLLSyncSite = null;  
DateTime mLastSync = DateTime.MinValue;
```

- Préfixe 'p' pour les paramètres des méthodes

```
public bool Synchronise(T_Sync_Site pRowSyncSite)  
{  
}
```

- Préfixe des variables selon son type

```
bool bBooleen = false;  
int intEntier;  
string strChaine;  
decimal decDecimal;  
double dblDouble;  
BLLArticle bllArt;  
T_Sync_Tampon rowSyncTampon = null;
```

Règles diverses

- Préférer new Datetime() au InterfaceDalCore.GetDate()

Benoit 8/6/2016 : Évitez d'initialiser une variable de type datetime dans un specific de la DAL avec InterfaceDalCore.GetDate() qui exécute 1 requête dans la BDD à chaque instantiation de la classe. C'était le cas dans T_SsChantier donc lorsqu'on listait les sous chantiers de GdP c'était très long car il y à 30000 lignes

Il vaut mieux utiliser new Datetime().

- Toujours se désabonner d'un évènement

Ajouter un « RemoveHandler » (généralement dans la méthode « Dispose ») pour chaque « AddHandler », il est même conseillé d'ajouter systématiquement un « RemoveHandler » avant chaque « AddHandler »

```
RemoveHandler txtValue.Click, AddressOf txtValue_Click  
AddHandler txtValue.Click, AddressOf txtValue_Click
```

Nettoyage du code

- Supprimer les “using” et “Import” non utilisés en haut des classes (apparaissent en gris)
- Supprimer les variables inutilisées
- Supprimer le code en commentaire

Charte graphique

- Ordre de tabulation : bien vérifier l'ordre des tabulations sur tous les écrans
- Radio et checkbox : le libellé doit toujours se trouver à gauche du champ
- Couleur de fond des champs, il a une signification
 - Blanc : champ simple, rien de particulier
 - Jaune pâle (LemonChiffon) : champ calculé [datacontrol & form]
 - Bleu clair : champ dont la valeur est obligatoire
 - Rouge : champs en erreur soit car non renseigné ou mal renseigné
 - Vert (LightGreen) : champ modifiable directement dans la grille
- Grilles, propriétés à initialiser dans le designer
 - ColumnAutoResize = True
 - KeepRowSettings = true
 - RowHeader = True
 - VisualStyleManager = Office2010
 - GroupByBoxVisible = False
 - FilterMode = Automatic
 - FilterRowUpdateMode = WhenValueChanged
 - SelectedFormatStyle.backColor et BackColorGradient = 167; 205; 240
 - SelectedInactiveFormatStyle.BackColor = 167; 205; 240
- DataControl
 - Préférer une présentation en formulaire, cad tous les champs sur une seule colonne (donc jamais plusieurs champs sur une même ligne)
 - Toujours vérifier que l'ordre des tabulations est logique (du haut vers le bas)
- Grilles dans un DataControl
 - Gérer les événements “DoubleClick” pour éditer et “KeyDown” (touche Del) pour supprimer
 - Définir un layout dans le designer de la grille
 - Associer un menu contextuel
 - Faire en sorte que l'on puisse éditer directement les enregistrements de la grille lorsque celle-ci est en preview (ReadOnly)

```
Dim m As New GridExContextMenu(Me.gridExArticle)
```

- Icônes :
 - Si nécessaire créer le fichier icône avec « Greenfish Icon editor » dans toutes les tailles.
 - Ajouter l'icône dans le dossier « Ressources\Icônes\Icône » du projet
 - Il doit porter le même nom que le texte qui a été mis à la propriété ImageKey du ApplicationFolder

- Résolutions d'écran
- Tous les écrans LINA doivent passer dans une résolution 1024 x 768
- Mobile : 240 x 320

DAL (Data Access Layer)

Utilise l'ORM Dapper, le code est en C#.

- [La DAL](#)

BLL (Business Logic Layer)

Implémente la couche métier de chaque objet (table), le code est en C#.

Remarque : Les méthodes qui ne peuvent pas être implémentées dans la BLL car besoin d'IHM sont à définir dans l'ApplicationFolder.

- [La BLL](#)
- [Test de la BLL](#)

UIL (User Interface Layer)

Correspond à la partie interface graphique, le code est en VB.NET

- [L'UIL](#)
- [Sélecteur manuel : grille de sélection sans passer par un textboxselect](#)
- [Application spé : personnaliser le ruban](#)
- [Les rapports DevExpress](#)
- [Editeur de valeur \(selon type\)](#)
- [Les combobox](#)

Divers

- [Astuces diverses](#)

Gestion de la sécurité

- [La gestion de la sécurité](#)

Les logs

- [La gestion des logs](#)

La localisation

- [La localisation de l'application](#)

GP/Hypervision

- [Container de supervision](#)
- [Fonction LINA](#)
- [Ajout d'une fonction GP \(pour type opération\)](#)

Règles pour l'Import/Export des déclarations LINA

- La table SQL doit avoir une colonne "xxxGuid [varchar](100) NULL"
- La BLL doit implémenter "ILINAExportImportBLL"
- Ajouter les méthodes "FillByNom" et "FillByGuid" (et les test qui vont avec)
- Dans "GetNewRow" initialiser un nouveau GUID "Guid.NewGuid().ToString()"
- Si une méthode dupliquer existe, régénérer un nouveau GUID
- Dans la méthode "DeleteRow" juste avant "Delete(Row)" faire appel à "BLLImportGuid.DeleteImportGuid(row.xxxGuid)"

Visual Studio

Liste des raccourcis utiles de Visual Studio :

Raccourcis	Description
Ctrl Espace	Auto complétion
Ctrl K + C	Commenter les lignes sélectionnées (fonctionne aussi dans SQL Server Management Studio)
Ctrl K + U	Décommenter les lignes sélectionnées (fonctionne aussi dans SQL Server Management Studio)
Ctrl K + D	Réorganiser les lignes indentations, les espaces et les retours à la ligne dans tous le code
Ctrl M + L	Réduit ou étend toutes les régions et le corps des fonctions/propriétés
Ctrl M + O	Réduit le corps des fonctions/propriétés
F5	Lancer le debug
F12	Atteindre la définition
Alt + F12	Aperçu de la définition
Shift + F12	Rechercher les références
F7	Afficher le code (si depuis le designer ou explorateur de solution)

Snippet et modèles intégrés à Visual studio

Les référentiels de modèles sont à récupérer sous :

"\SERVEURDATANAS\utilitaires\Informatique\Visual Studio AddOn\snippet"

Snippet

Pour ajouter un snippet dans Visual Studio, menu Outils > Gestionnaire des extraits de code > Importer

- **InsertionBLL** : permet l'utilisation rapide d'une BLL en VB en tapant “_bll” puis en appuyant sur Tab.

Modèles

Il s'agit de templates qui permettent de créer rapidement des DataControl, BLL, GridEx ... dans Visual Studio. Les fichiers .zip sont à mettre à l'emplacement suivant : “ C:\Users\Admin\Documents\Visual Studio ???\Templates\ItemTemplates\LINA “

Le dossier LINA sert à avoir les éléments regroupés dans une catégorie.

Il faut redémarrer Visual Studio pour pouvoir en profiter.

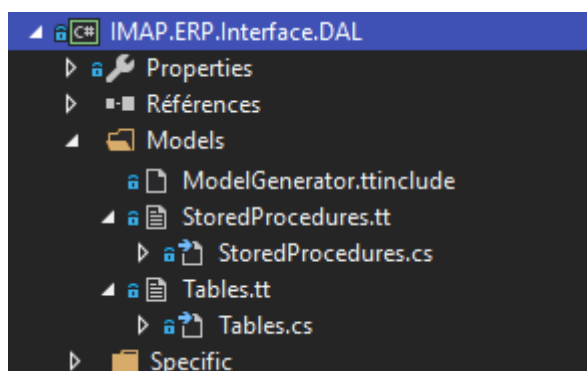
- **ApplicationFolder**
- **BllCommon**
- **DataControl**
- **DataForm**
- **GridEx**
- **GridExView**

La DAL

Le projet DAL (Data Acces Layer) correspond à la couche d'interface avec la base de données, il utilise l'ORM (Object Relational Mapping) « Dapper ».

Structure Dapper

Ce dernier permet de générer l'ensemble des classes C# correspondantes aux tables de la base de données et utilise le modèle ActiveRecord.



Les fichiers :

- ModelGenerator.ttinclude : Modèle T4 qui permet la génération des classes C# du modèle, ne pas le modifier
- StoredProcedures.tt : Modèle T4 pour la génération du code C# des procédures stockées
- StoredProcedures.cs : Code C# correspondant aux procédures stockées de la BDD
- Tables.tt : Modèle T4 pour la génération du code C# des tables
- Tables.cs : Code C# correspondant aux tables de la BDD
- Specific : Contient le code C# des classes partielles spécifiques

Lorsque la base de données change, il faut régénérer les classes C#. Pour cela, depuis Visual Studio : faire un clic droit sur le fichier Tables.tt et sélectionner "Exécuter un outil personnalisé"

Les classes C# sont alors actualisées selon le nouveau modèle de la base de données

```
public partial class T_RoleTable
{
    public static string _TableName = "T_Role";
    public static string IDRoleColumn = "IDRole";
    public static string _IDRoleColumn = "T_Role.IDRole";
    public static string RoleNameColumn = "RoleName";
    public static string _RoleNameColumn = "T_Role.RoleName";
    public static string DescriptionColumn = "Description";
    public static string _DescriptionColumn = "T_Role.Description";
    public static string StarterFolderKeyColumn = "StarterFolderKey";
    public static string _StarterFolderKeyColumn = "T_Role.StarterFolderKey";
    public static string RoleADColumn = "RoleAD";
    public static string _RoleADColumn = "T_Role.RoleAD";
    public static string RoleGuidColumn = "RoleGuid";
    public static string _RoleGuidColumn = "T_Role.RoleGuid";
}
```

```
/// <summary>
/// Represents the T_Role table.
/// </summary>
[Table("T_Role")]
70 références | GROUPEAPI\Xavier_Jafficher, il y a 330 jours | 1 auteur, 1 modification
public partial class T_Role : ActiveRecord
{
    private int iDRole;

    [Key]

    [Column]
    82 références | 0 modification | 0 auteur, 0 modification
    public int IDRole { get { return iDRole; } set { SetProperty(ref iDRole, value); } }

    private string roleName;

    [MaxLength(50)]

    [Column]
    5 références | 0 modification | 0 auteur, 0 modification
    public string RoleName { get { return roleName; } set { SetProperty(ref roleName, value); } }

    private string description;

    [MaxLength(100)]

    [Column]
    2 références | 0 modification | 0 auteur, 0 modification
    public string Description { get { return description; } set { SetProperty(ref description, value); } }
```

Nommages

Dapper génère également des propriétés de type « string » qui contiennent le nom des tables et des champs de la base, l'utilisation de ces propriétés permet de ne pas mettre en « dur » les noms dans le code, cela évite beaucoup d'erreur.

- Récupérer le nom d'une table : « T_ClientTable._TableName » renvoi « T_Client »
- Récupérer le nom court d'une colonne d'une table : « T_ClientTable.CltID » renvoi « CltID »
- Récupérer le nom long d'une colonne d'une table : « T_ClientTable._CltID » renvoi « T_Client.CltID »

Le dossier "Specific"

Dans certains cas il est nécessaire d'étendre les classes générées par Dapper. Pour cela on peut ajouter des classes partielles dans le dossier « Specific » du projet « DAL », et qui ne seront pas écrasées à chaque régénération de la DAL.

```
namespace IMAP.Common.DAL
{
    public partial class T_Member
    {
        public string RoleName { get; set; } = "";
    }
}
```

Dans l'exemple ci-dessus, on étend la classe « T_Member » (correspond à la table « T_Member » de la base de données) qui se trouve dans le namespace « IMAP.Common.DAL » en lui ajoutant une propriété « RoleName » de type « string ».

La BLL

Le projet « BLL » (Business Logic Layer) contient la logique métier de l'application, c'est dans ce projet que doivent être codé toute « l'intelligence » de l'application. Par convention au moins une classe « BLL » est générée par table de la base de données, celle-ci doit respecter certaines règles.

Le constructeur

- Doit être une classe « Public » qui hérite de `IMAP.Common.BLLCommon<T>`

```
public class BLLEntite : BLLCommon<T_Entite>
```

- Constructeur :

Doit définir sur quelle table on travaille `TableName`, quelle est la colonne de clé primaire `IDColumn`, les colonnes de tri `SortColumns`

```
this.TableName = T_EntiteTable._TableName;  
this.IDColumn = T_EntiteTable._EntIDColumn;  
this.SortColumns = T_EntiteTable._EntNomColumn;
```

Il existe aussi `FillColumns` pour définir quelles colonnes lister (par défaut "*"), et `InnerJoin` pour les jointures sur les autres tables.

```
this.FillColumns = string.Format("{0}.*, {1}.EntTypeNom, {2}.BankNom,  
{3}.JuridiqueNom ",  
    T_EntiteTable._TableName, T_Enum_Entite_TypeTable._TableName,  
    T_BanqueTable._TableName, T_Enum_FormeJuridiqueTable._TableName);  
  
this.InnerJoin = string.Format(" LEFT JOIN {1} ON {0}.EntTypeID =  
{1}.EntTypeID " +  
    " LEFT JOIN {2} ON {0}.BankID = {2}.BankID " +  
    " LEFT JOIN {3} ON {0}.JuridiqueID = {3}.JuridiqueID ",  
    T_EntiteTable._TableName, T_Enum_Entite_TypeTable._TableName,  
    T_BanqueTable._TableName, T_Enum_FormeJuridiqueTable._TableName);
```

Remarque : Dans le cas d'un tri sur plusieurs colonnes, les mettre toutes séparées par « , », si l'ordre de tri n'est pas spécifié, il est par défaut « ASC »

```
this.SortColumns = $"{T_EntiteTable._EntNomColumn},  
{T_EntiteTable._EntCodeColumn}";
```

Les méthodes standards

Méthodes disponibles

- Fill : Permet de lister toutes les lignes de la table (équivalent à « SELECT * FROM Table »)
- FillByID : Permet de lister 1 ligne de la table selon la valeur de la clé primaire (équivalent à « SELECT * FROM Table WHERE ID = 1 »)
- GetNewRowID : Permet de renvoyer la valeur suivante de la clé primaire, dans le cas d'une table en clé primaire automatique cette méthode doit renvoyer « -1 ».

Méthodes à redéfinir obligatoirement

- GetNewRow : Permet de renvoyer une instance « ActiveRecord » correspondante à la table avec des valeurs par défaut dans les champs non NULL.

```
public override T_Entite GetNewRow()
{
    return new T_Entite()
    {
        EntID = -1,
        EntNom = string.Empty,
        EntVille = string.Empty,
        EntPrincipale = false,
        EntInactif = false,
        EntGuid = Guid.NewGuid().ToString()
    };
}
```

Méthodes à redéfinir si besoin

- CheckRow : Permet d'effectuer les contrôles de cohérences sur un « ActiveRecord » avant que celui-ci soit enregistré dans la base. Peut ne pas être redéfinie s'il n'y a pas de contrôles à faire ou que tous rentrent dans des validations standards.

```
public override bool CheckRow(T_Entite pRow, bool pShowPopup)
{
    bool result = base.CheckRow(pRow, false);

    //Web
    if (!string.IsNullOrEmpty(pRow.EntWeb) &&
        !pRow.EntWeb.Contains("."))
    {
        mValidationList.Add(new
        CheckRowValidationCustom(T_EntiteTable.EntWebColumn,
        Ressources.Common.RessCommon.BLLEntite_CheckRow_4));
        result = false;
    }

    //OK (ou pas)
```

```
AfficheErreurs(pShowPopup);  
return result;  
}
```

- UpdateRow : Permet de personnaliser les traitements au moment de l'enregistrement d'un « ActiveRecord » dans la base de données.

```
public override bool UpdateRow(T_Entite prow)  
{  
    //Si Entité principale  
    if (prow.EntPrincipale)  
    {  
        //Si principale : Décoche les autres  
        BLLEntite bllEnt = new BLLEntite();  
        bllEnt.FillByPrincipale(true);  
        foreach (T_Entite row in bllEnt.DataSource)  
        {  
            if (row.EntID != prow.EntID && row.EntPrincipale)  
            {  
                row.EntPrincipale = false;  
                bllEnt.UpdateRow(row);  
            }  
        }  
        bllEnt.Dispose();  
    }  
  
    //Update  
    if (!base.UpdateRow(prow)) return false;  
  
    //Ok  
    return true;  
}
```

- DeleteRow : Permet de personnaliser les traitements au moment de la suppression d'un « ActiveRecord » dans la base de données (interrompre la suppression s'il y a des relations dans la base de données, suppression des enregistrements liés, ...), ces traitements doivent obligatoirement se faire dans une transaction.

```
public override bool DeleteRow(T_Entite prow)  
{  
    //DeleteChilds  
    if (!base.ExecuteDeleteChilds(prow)) return false;  
  
    //Démarre transaction  
    BeginTransaction();  
    try  
    {  
        //Link T_Client  
        BLLClient bllClt = new BLLClient();  
        bllClt.FillByEntID(prow.EntID);  
        if (bllClt.DataSource.Count > 0)  
        {
```

```
Logger.Message(string.Format(Ressources.Common.RessCommon.BLLEntite_DeleteRow_1, row.EntNom) +
GetLinkedRowInString(bllClt.DataSource, T_ClientTable.CltNomColumn),
LINADALCore.MessageCaption,
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Information);
        RollbackTransaction();
        return false;
    }
    bllClt.Dispose();

    //T_Entite_Compta
    BLLEntiteCompta bllEntCompta = new BLLEntiteCompta();
    bllEntCompta.FillByEntID(row.EntID);
    if (!bllEntCompta.DeleteAllRows())
    {
        RollbackTransaction();
        return false;
    }
    bllEntCompta.Dispose();

    //Delete row
    Delete(prow);

    //Valide
    CommitTransaction();
    return true;
}
catch (Exception ex)
{
    RollbackTransaction();
    Logger.Message(this.GetType().Name + ":DeleteRow() : " +
ex.Message, LINADALCore.MessageCaption,
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
    return false;
}
}
```

Ajout de méthodes « Fill »

En plus des deux méthodes de base Fill et FillByID, il très souvent nécessaire d'ajouter d'autres méthodes pour charger les enregistrements selon différents critères. Pour cela on ajoute autant de méthode FillBy... que nécessaire.

Par convention la méthode est nommé FillBy + Nom des colonnes de filtre, exemple :

- FillByCltNom : select des clients selon la colonne « CltNom »
- FillByCltNomAndCltCode : select des clients selon les colonnes « CltNom » et « CltCode »


```
public bool FillByEntEDINum(string pstrEntEDINum)
{
    return FillBy(T_EntiteTable._EntEDINumColumn,
GetStringValue(pstrEntEDINum));
}
```

Dans l'exemple ci-dessus on fait un « SELECT » avec un « WHERE » sur la colonne « EntEDINum » de la table « T_Entite. La méthode GetWhereRequest va renvoyer le texte complet de la requête selon la clause WHERE fournie en paramètre, il y a plusieurs appels possibles :

- GetWhereRequest(« Colonne 1 », « Valeur 1 ») = SELECT * FROM Table WHERE Colonne 1 = Valeur 1
- GetWhereRequest(« Colonne 1 », « Valeur 1 », « Colonne 2 », « Valeur 2 ») = SELECT * FROM Table WHERE Colonne 1 = Valeur 1 AND Colonne 2 = Valeur 2
- GetWhereRequest(« Colonne 1 > 0 OR Colonne 2 < 0 ») = SELECT * FROM Table WHERE Colonne 1 > 0 OR Colonne 2 < 0
- GetWhereRequestWithTop(5, « Colonne 1 », « Valeur 1 ») = SELECT TOP 5 * FROM Table WHERE Colonne 1 = Valeur 1

Remarque : Pour passer une valeur à « GetWhereRequest », **TOUJOURS** utiliser la méthode GetStringValue car elle permet de convertir la valeur de sorte à ce qu'elle soit correcte pour la requête SQL (ajout de cote pour les chaîne, remplacement de caractères invalides, prise en compte de la langue du serveur SQL, ...)

Remarque : Pour les noms des tables et des colonnes ont utilise **TOUJOURS** les constantes de la DAL et **JAMAIS** les noms en texte brut

Les validations génériques

L'intelligence étant localisée dans la BLL c'est elle qui est en charge de vérifier la cohérence des données, la méthode « CheckRow » est déjà là pour cela mais toutes une série de validations génériques permettent de simplifier le code. Toutes ces validations héritent de CheckRowValidationBase et sont déclarées dans le constructeur de la BLL.

Une validation consiste à vérifier une ou des conditions et en cas d'erreur à afficher un message à l'utilisateur et colorer en rouge le fond du champ correspondant dans le « DataControl » (uniquement s'il y a un binding sur le champ en question).

```
public BLLEntite()
{
    this.TableName = T_EntiteTable._TableName;
    this.IDColumn = T_EntiteTable._EntIDColumn;
    this.SortColumns = T_EntiteTable._EntNomColumn;

    //EntNom
    mValidationList.Add(new
CheckRowValidationRequired(T_EntiteTable.EntNomColumn,
Ressources.Common.RessCommon.BLLEntite_CheckRow_1));
    //Nom existe déjà
```

```

    mValidationList.Add(new
    CheckRowValidationUnique(T_EntiteTable.EntNomColumn,
    Ressources.Common.RessCommon.BLLEntite_CheckRow_5, typeof(BLLEntite),
    ((Func<string, bool>)FillByNom).Method.Name, T_EntiteTable.EntNomColumn));
    //BDD EBP
    mValidationList.Add(new
    CheckRowValidationUnique(T_EntiteTable.EntEBPDBNameColumn,
    Ressources.Common.RessCommon.BLLEntite_CheckRow_7, typeof(BLLEntite),
    ((Func<string, bool>)FillByEntEBPDBName).Method.Name,
    T_EntiteTable.EntEBPDBNameColumn));
    //Si Principale et inactive
    mValidationList.Add(new
    CheckRowValidationNand(T_EntiteTable.EntInactifColumn,
    Ressources.Common.RessCommon.BLLEntite_CheckRow_9,
    new List<CheckRowValidationBase>() {
        new
    CheckRowValidationCompareValue(T_EntiteTable.EntPrincipaleColumn, "",
    CompareOperator.Equal, true),
        new
    CheckRowValidationCompareValue(T_EntiteTable.EntInactifColumn, "",
    CompareOperator.Equal, true)}));
    //Caractères interdits
    mValidationList.Add(new
    CheckRowValidationForbiddenChars(T_EntiteTable.EntNomColumn,
    Ressources.Common.RessCommon.BLLEntite_CheckRow_1, new char[] { (char)64
    }));
}

```

La liste des validations à effectuer par une BLL est défini dans son constructeur.

a) Champ obligatoire

« **CheckRowValidationRequired** » : permet de vérifier qu'un champ n'est pas vide (NULL/Nothing), dans le cas d'une chaîne on vérifie en plus s'il elle ne contient pas que des espaces, dans le cas d'une IList/IBindingList on vérifie qu'il y a au moins 1 élément.

```

mValidationList.Add(new CheckRowValidationRequired(T_EntiteTable.EntNomColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_1));

```

Ci-dessus, permet de vérifier que la valeur du champ « EntNom » est renseigné.

b) Champ vide

« **CheckRowValidationEmpty** » : permet de vérifier qu'un champ est vide (NULL/Nothing), dans le cas d'une chaîne on vérifie en plus s'il elle contient que des espaces, dans le cas d'une IList/IBindingList on vérifie qu'il y a aucun élément.

```
mValidationList.Add(new CheckRowValidationEmpty(T_Client_Avoir_DetailTable.ArtIDColumn
, Ressources.Common.RessCommon.BLLEntite_CheckRow_1));
```

Ci-dessus, permet de vérifier que la valeur du champ « ArtID » est bien Null/Vide.

c) Mini et/ou maxi sur la valeur d'un champ

« **CheckRowValidationValue** » : permet de vérifier la valeur d'un champ par rapport à une valeur, il y a plusieurs opérateurs (égal, supérieur, inférieur, différent, supérieur ou égal, inférieur ou égal). Si la valeur est NULL/Nothing il y a erreur de validation.

```
mValidationList.Add(new CheckRowValidationValue(T_Client_BLTable.BLRemiseColumn, Ressources.CdeClient.RessCdeClient.BLLClientBL_CheckRow_22, 0, 100));
```

Ci-dessus, permet de contrôler que la valeur du champ « BLRemise » contient une valeur entre 0 et 100 inclus.

d) Mini et/ou maxi sur la longueur de la donnée d'un champ

« **CheckRowValidationLength** » : permet de vérifier la longueur d'une chaîne par rapport à un mini et/ou maxi, si la longueur est inférieure et/ou supérieure à la longueur définie il y a erreur.

```
mValidationList.Add(new CheckRowValidationLength(T_Client_BLTable.BLRefCltColumn, Ressources.CdeClient.RessCdeClient.BLLClientBL_CheckRow_22, 10, 20));
```

Ci-dessus, permet de vérifier que la valeur du champ « BLRefClt » a bien une longueur entre 10 et 20 caractères inclus.

e) Champ unique

« **CheckRowValidationUnique** » : permet de vérifier l'unicité d'un champ par rapport à d'autres enregistrements (c'est le champ de clé primaire de la table qui sert de clé). Si la valeur du champ est retrouvée sur un autre enregistrement il y a erreur de validation. Si la valeur du champ est NULL/Nothing la validation est conforme.

```
mValidationList.Add(new CheckRowValidationUnique(T_EntiteTable.EntNomColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_5, typeof(BLLEntite), ((Func<string, bool>)FillByNom).Method.Name, T_EntiteTable.EntNomColumn));
```

Ci-dessus, permet de vérifier que la valeur du champ « EntNom » est bien unique parmi toutes les autres entités. Cette validation fait appel à la méthode « FillByNom » de la BLL entité qui doit exister, et elle recevra en paramètre la valeur du champ « EntNom ».

f) Caractères interdits

« **CheckRowValidationForbiddenChars** » : permet de vérifier qu'un champ ne contient par une liste de caractères. Si le champ n'est pas une chaîne la validation est en erreur.

```
mValidationList.Add(new CheckRowValidationForbiddenChars(T_EntiteTable.EntNomColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_1, new char[] { (char)64}));
```

Ci-dessus, permet de vérifier que la donnée du champ « EntNom » ne contient pas « @ ».

g) Comparaison de champs

« **CheckRowValidationCompareFields** » : permet de comparer 2 colonnes entre elles, plusieurs opérateurs sont disponibles (égal, supérieur, inférieur, différent, supérieur ou égal, inférieur ou égal).

Ci-dessus, permet de vérifier que la valeur du champ « VehiculeKm » est supérieure ou égale à la valeur du champ « VehiculeKmAchat ».

h) Comparaison de valeurs

« **CheckRowValidationCompareValue** » : permet de comparer un champ à une valeur, plusieurs opérateurs sont disponibles (égal, supérieur, inférieur, différent, supérieur ou égal, inférieur ou égal), si le champ est NULL/Nothing il y a erreur.

```
mValidationList.Add(new CheckRowValidationCompareValue(T_VehiculeTable.VehiculeTypeIDColumn, Ressources.Salarie.RessSalarie.BLLVehicule_CheckRow_1, CompareOperator.Different, -1));
```

Ci-dessus, permet de vérifier que la valeur du champ « VehiculeTypeID » est différente de « -1 ».

i) Format adresse email

« **CheckRowValidationEmail** » : permet de vérifier que le champ contient bien une chaîne qui respecte les règles d'une adresse email, le champ peut contenir plusieurs adresses emails si elles sont séparées par « ; ». Si le champ est NULL/Nothing il n'y a pas d'erreur.

```
mValidationList.Add(new CheckRowValidationEmail(T_EntiteTable.EntMailColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_3));
```

Ci-dessus, permet de vérifier que la valeur du champ « EntMail » contient bien une adresse email

j) Validation multiple

« **CheckRowValidationNand** » : permet de combiner plusieurs validations entre elles, test les validations les unes à la suite des autres, dès qu'une validation est en erreur la validation multiple est conforme. C'est un « NOT (Cond 1 AND Cond 2 AND Cond 3) ». La liste des validations correspond donc à la combinaison que l'on ne souhaite pas avoir.

```
mValidationList.Add(new CheckRowValidationNand(T_EntiteTable.EntInactifColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_9,
    new List<CheckRowValidationBase>() {
        new CheckRowValidationCompareValue(T_EntiteTable.EntPrincipaleColumn, "", CompareOperator.Equal, true),
        new CheckRowValidationCompareValue(T_EntiteTable.EntInactifColumn, "", CompareOperator.Equal, true)}));
```

Ci-dessus, permet de vérifier que l'on n'a pas à la fois la valeur du champ « EntPrincipal » égale « True » ET la valeur du champ « EntInactif » égale « True ». Si les 2 conditions sont vérifiées le champ « EntInactif » est en erreur. Remarque : pour les « sous » validation, il n'est pas nécessaire de spécifier un message d'erreur car c'est celui de la validation « Nand » qui est affiché.

k) Validation personnalisée

« **CheckRowValidationCustom** » : permet dans le « CheckRow » d'ajouter une erreur de validation personnalisée en spécifiant le champ concerné et le message d'erreur associé.

```
public override bool CheckRow(SubSonic.Schema.IActiveRecord pRow, bool pShowPopup)
{
    bool result = base.CheckRow(pRow, false);
    T_Entite rowEnt = (T_Entite)pRow;

    //EntEBPDossier
    if (!string.IsNullOrEmpty(rowEnt.EntEBPDossier) && !rowEnt.EntEBPDossier.Trim().EndsWith(".ebp"))
    {
        if (pShowPopup) Log.Logger.Message(Ressources.Common.RessCommon.BLLEntite_CheckRow_8, InterfaceDALCore.MessageCaption, System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Information);

        mValidationList.Add(new CheckRowValidationCustom(T_EntiteTable.EntEBPDossierColumn, Ressources.Common.RessCommon.BLLEntite_CheckRow_8));
        result = false;
    }

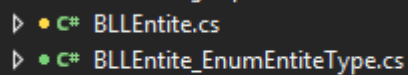
    //OK (ou pas)
    AfficheErreurs(pShowPopup);
    return result;
}
```

Ci-dessus, dans la méthode « CheckRow » le contrôle est fait manuellement et en cas d'erreur on ajoute une nouvelle instance « CheckRowValidationCustom » à la liste des validations.

Remarque : bien noté l'appel à « **base.CheckRow** » sans affichage de message au début de la méthode, et l'appel à « **AfficheErreurs** » en fin de méthode.

Surcharge d'une BLL

Dans certains cas il est nécessaire de surcharger une BLL existante, notamment pour modifier la liste des champs et tables à lister dans la base de données. Pour cela on crée une nouvelle classe dans le dossier "Specific" du projet "DAL" qui par convention va porter le même nom que la BLL d'origine avec le nom des tables jointes en plus.



```
▶ C# BLLEntite.cs
▶ C# BLLEntite_EnumEntiteType.cs
```

Dans l'exemple ci-dessus la classe `BLLEntite_EnumEntiteType_Banque_EnumFormeJuridique` hérite de `BLLEntite` et fait une jointure sur la table `T_Enum_Entite_Type`, `T_Banque` et `T_EnumFormeJuridique` de la base de données. Dans cette nouvelle classe il suffit de redéfinir le constructeur pour lister les jointures et les champs supplémentaires.

```
public class BLLEntite_EnumEntiteType_Banque_EnumFormeJuridique : BLLEntite
{
    /// <summary>
    /// Constructeur
    /// </summary>
    public BLLEntite_EnumEntiteType_Banque_EnumFormeJuridique()
    {
        this.FillColumns = string.Format("{0}.*, {1}.EntTypeNom, {2}.BankNom, {3}.JuridiqueNom ",
            T_EntiteTable._TableName, T_Enum_Entite_TypeTable._TableName,
            T_BanqueTable._TableName,
            T_Enum_FormeJuridiqueTable._TableName);

        this.InnerJoin = string.Format(" LEFT JOIN {1} ON {0}.EntTypeID = {1}.EntTypeID " +
            " LEFT JOIN {2} ON {0}.BankID = {2}.BankID " +
            " LEFT JOIN {3} ON {0}.JuridiqueID = {3}.JuridiqueID ",
            T_EntiteTable._TableName, T_Enum_Entite_TypeTable._TableName,
            T_BanqueTable._TableName,
            T_Enum_FormeJuridiqueTable._TableName);
    }
}
```

Remarque : Il faut bien prendre en compte que la surcharge d'une BLL doit s'accompagner d'un « Specific » dans la DAL. Les champs supplémentaires correspondent à une propriété portant le même nom dans le specific.

```
namespace IMAP.ERP.Interface.DAL
{
    public partial class T_Entite
    {
        public string EntTypeNom { get; set; }
        public string BankNom { get; set; }
        public string JuridiqueNom { get; set; }
    }
}
```

Libérer une BLL après utilisation

Il est important de bien vider et libérer une BLL lorsqu'elle ne sert plus, sinon ça consomme de la mémoire pour rien (le Garbage Collector ne fait pas bien son travail)

```
using (BLLSyncTampon bllTampon = new BLLSyncTampon())
{
    //Chargement des données
    bllTampon.Fill();

    //Traitements

    //Vide la source
    bllTampon.DataSource.Clear();
}
```

OU

```
BLLSyncTampon bllTampon = new BLLSyncTampon();

//Chargement des données
bllTampon.Fill();

//Traitements

//Vide la source et libère
bllTampon.DataSource.Clear();
bllTampon.Dispose();
```