

😊 Icône Ajouter

🖼️ Ajouter une couverture

Tutos MVC

Table des matières

Tuto 11/03

Base des contrôleurs

[Exemple de requêtes et leur résolution](#)

[Explication des types de retour courants](#)

Tuto 13/03

Rappel de prérequis

[Utiliser la traduction, _ViewImports.cshtml, _Layout.cshtml et Areas](#)

[Base des vues _ViewStart.cshtml](#)

[_Views\ _ViewImports.cshtml](#)

[Potentiel autre intérêt des Areas \(rappel : explication de l'intérêt, voyons plus tard pour les éventuels débats\)](#)

[_Views\Shared\ _Layout.cshtml](#)

[Exercice : Utiliser RenderSectionAsync pour afficher un "Dashboard" uniquement sur la page Index](#)

Tuto 18/03

ViewModel : Principe de ToModel et ToEntity

[Exercice : Utiliser un nouveau ViewModel dans HomeController, on l'appellera UserSafeViewModel du modèle Utilisateur](#)

Tuto 20/03

[Place au tuto du jour !](#)

[BaseController qu'il est bien, c'est le maître des contrôleurs](#)

[Résumé](#)

[Exercice : Créer un contrôleur qui hérite de BaseController BaseController<Model, IModelService, ModelViewModel>](#)

Tuto 25/03

[✅ Quel est le rôle de CosmosServices ?](#)

[🧱 Pourquoi un IGenericService ?](#)

[🔧 Et AddScoped dans ServiceRegistration ? \(capture plus haut avec tous les AddScoped<Services>\)](#)

[🌸 Résumé simple](#)

[Bootstrap et Bootstrap Icons](#)

[Exercice libre : Tester 1 élément layout et 2 éléments Components dans une vue](#)

Tuto 27/03

Les ViewComponent MVC

[✂ Exercice pratique : Créer un ViewComponent InfoBoxComponent](#)

Tuto 01/04

Révision Base du MVC

[🌸 1. Qu'est-ce que le MVC ?](#)

[🌸 Rappel du cycle d'une requête MVC](#)

[✅ Points clés à retenir](#)

Tuto 03/04

[Révision ViewImports.cshtml, _ViewStart.cshtml, _Layout.cshtml et Areas](#)

[_ViewImports.cshtml](#)

[_ViewStart.cshtml](#)

[Areas](#)

[_Layout.cshtml](#)

[Résumé](#)

Tuto 08/04

[🌸 Révision : ViewModel, ToModel et ToEntity](#)

[🌸 Pourquoi utiliser un ViewModel ?](#)

[📦 À quoi servent ToModel\(\) et ToEntity\(\) ?](#)

[🗑 Dans un contrôleur MVC :](#)

Tuto 10/04

[✅ Ce qui est commun](#)

[⚠ Les différences et problèmes potentiels](#)

[1. 🧱 Types SQL spécifiques](#)

[2. 🔄 Migrations différentes selon les bases](#)

[3. 🗝 Gestion des clés, index et contraintes](#)

[4. 📏 Longueurs de chaînes et unicodes](#)

[5. 🌿 Fonctions SQL et LINQ traduites](#)

[6. 🚫 Fonctionnalités non supportées \(Oracle surtout\)](#)

[💡 Stratégie recommandée](#)

[📁 En résumé](#)

[🌸 Révision : BaseController, le maître des contrôleurs](#)

[🔗 Pourquoi BaseController est génial \(même si un peu magique\) :](#)

[✅ Résumé clair](#)

[📁 Exemple : ActiviteController généré automatiquement](#)

[🌸 BaseController contient la "magie" générique](#)

[🗑 Rappel : Les services sont enregistrés dans CosmosServices/ServiceRegistration.cs](#)

 Bonus

Tuto 15/04

Tuto 17/04

Si les pages clients sont très spécifiques → Razor Class Library (RCL) par client

Tuto 11/03

Base des contrôleurs

Program.cs contient ceci

```
// From more specific to more general routes
app.MapControllerRoute(
    name: "Admin",
    pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");
// Default route
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

La règle du haut est pour les Areas (juste si on veut mettre certains contrôleurs dans des dossiers à part) dans le projet ça correspond à \CosmosWeb\Areas\Admin\Controllers\ mais on n'ira pas plus loin ici sur les Areas

La règle du bas est la règle par défaut des projets MVC, si contrôleur pas renseigné alors HomeController, si action pas renseigné alors Index et enfin id facultatif donc si je tape une URL en ne mettant que le contrôleur je tomberais toujours sur l'action/vue Index

L'action correspond à une vue (sauf POST) et respecte toujours Views\{controller}\NomDeLaction.cshtml, par exemple \Views\Alarme\Index.cshtml mais rien n'empêche une action Truc de faire un return View("Chose") (à condition d'avoir un Chose.cshtml)

Voici l'exemple des 2 actions Delete, l'une est la vue (donc implicitement un [HttpGet]) l'autre correspond à l'action POST Delete

```
public async Task<IActionResult> Delete(string id, CancellationToken cancellationToken)
{
    if (id == null)
        return NotFound();
    var entity = await Service.GetByIdAsync(id, cancellationToken);
    if (entity == null)
        return NotFound();
    return View(entity);
}
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 références | 0 modification | 0 auteur, 0 modification
public async Task<IActionResult> DeleteConfirmed(string id, CancellationToken cancellationToken)
{
    var entity = await Service.GetByIdAsync(id, cancellationToken);
    if (entity != null)
        await Service.DeleteAsync(entity, cancellationToken);
    return RedirectToAction(nameof(Index));
}
```

Petite convention : Un underscore sur un nom de fichier vue Razor indique une vue partielle (ex : _Edit.cshtml)

Exemple de requêtes et leur résolution

	≡ URL	≡ Contrôleur Résolu	≡ Action Résolue	≡ Paramètre id
1	/Home/Index	HomeController	Index	null
2	/Home/About	HomeController	About	null
	/Admin/Dashboard	DashboardController	Index	null

3		(dans Admin)		
4	/Products/Details/5	ProductsController	Details	5

Exemple le bouton de création dans chaque Index.cshtml de chaque modèle, on est dans View/Alarm donc contrôleur AlarmController

```
@model IEnumerable<AlarmeViewModel>

<h2>Alarme List</h2>
<p>
    <a asp-action="Create" class="btn btn-primary">@Lang["New"]</a>
</p>
```

Autres exemples

```
<li> <a href="@Url.Action("About", "Home")">Voir la page À propos</a> </li>
```

```
<li> <a asp-action="About" asp-controller="Home">Voir la page À propos</a> </li>
```

Exemple formulaire

```
<h2>Formulaire avec @Url.Action</h2>
<form method="post" action="@Url.Action("SubmitForm", "Home")">
    <input type="text" name="name" placeholder="Entrez votre nom" />
    <button type="submit">Envoyer</button>
</form>
```

Avec un argument, les deux génèrent l'URL /Products/Details/5

```
<a href="@Url.Action("Details", "Products", new { id = 5 })">Voir produit 5</a>
```

```
<a asp-action="Details" asp-controller="Products" asp-route-id="5">Voir produit 5</a>
```

Explication des types de retour courants

- `View()` → Retourne une page HTML (vue associée).
- `Content("texte")` → Retourne une simple réponse en texte brut.
- `Json(data)` → Retourne des données JSON.
- `RedirectToAction("NomAction")` → Redirige vers une autre action.

Exemple la vue Index de HomeController (voir à nouveau capture des 2 Delete au-dessus)

```
public IActionResult Index()
{
    return View();
}
```

<https://localhost:7039/Categorie/Edit?categorie1=1> return View("_Edit", CategorieViewModel.ToModel(categorie));

<https://localhost:7039/Categorie/Details/1> pour exemple de renvoi Json

Tuto 13/03

Pour répondre à une question du tuto précédent, 1er exemple trouvé sur le net

```
[Route("")]
[Route("/")]
[Route("Index")]
public string Index()
{
    .
```

```
{
    return "Index() Action Method of HomeController";
}
```

Rappel de prérequis

Visual Studio 17.12 minimum pour NET 9.0 et installation dernier SDK 9.0.101

Extension Visual Studio EF Core Power Tools de ErikEJ (futur tuto EF Power Tools)

Adapter la chaîne de connexion dans CosmosMVC\CosmosWeb\appsettings.json

HomeController (celui par défaut) et DevController sur le projet sont des contrôleurs standards utilisable pour les exercices

Utiliser la traduction, _ViewImports.cshtml, _Layout.cshtml et Areas

Pour les modèles et ViewModels, il existe un attribut `CosmosModels/Localization/LocalizedDisplayNameAttribute.cs`

```
[LocalizedDisplayName("CompanyName")]
9 références
public required string RaisonSociale { get; set; }
```

Affichera par exemple dans le formulaire d'édition "Raison sociale" du modèle Société

```
<label asp-for="RaisonSociale" class="form-label"></label>
<input asp-for="RaisonSociale" class="form-control" />
<span asp-validation-for="RaisonSociale" class="text-danger"></span>
```

En haut de la vue est déclaré le modèle utilisé, donc la vue sait que l'on fait référence à RaisonSociale (donc autocomplétions possible) de cette instance de Société (futur tuto ViewModel)

```
@model SocietyViewModel
```

Exemple :

- AdresseRue2 n'a pas de LocalizedDisplayName, il s'affiche tel quel.
- RaisonSociale a un LocalizedDisplayName, il affiche "Raison sociale".

Raison sociale

Société principale

AdresseRue2

Dans les vues il suffit d'utiliser `@Lang["CledeTrad"]` car la ligne d'injection est dans `_ViewImports.cshtml` donc dans toutes les vues (explication après)

Même principe dans les contrôleurs, s'ils dépendent de BaseController ils ont Lang aussi, pour les cas particuliers, il existe un helper pour ça, `LocalizationHelper`

```
public IActionResult TestLang()
{
    var localizer = LocalizationHelper.GetLocalizer(HttpContext);
    var translation = localizer["Hello"];
    return Content($"Translation: {translation}");
}
```

Toutes les traductions se font dans `CosmosWeb\Localization\SharedResources.resx` (avec un fr et en)

Je propose que toute nos traductions soient préfixées, exemple `Common.Accept`, `Society.CompanyName`, etc (raison tout est en un fichier)

Il existe une classe `LoggingStringLocalizer` dans `CosmosServices` dont le but est de faire apparaître les traductions manquantes, actuellement en warning dans la console sous cette forme

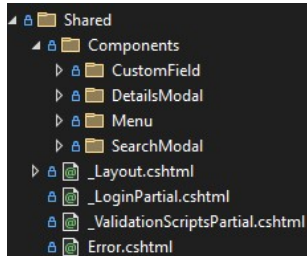
```
warn: CosmosServices.Localization.LocalizationService[0]
      => SpanId:9341ce5776999948, TraceId:43a31aa660ac9b69758c20e97c804654, ParentId:0000000000000000
      LANG Missing translation : TradBidon
```

Base des vues _ViewStart.cshtml

\Views_ViewStart.cshtml contient simplement un appel à _Layout mais par le code on pourrait très bien dire "Si mobile alors _LayoutMobile"

```
@{
    Layout = "_Layout";
}
```

Noter que _Layout est dans \Views\Shared_Layout.cshtml c'est là que toutes les vues partagées sont, ainsi que les ViewComponent (futur tuto ViewComponent)



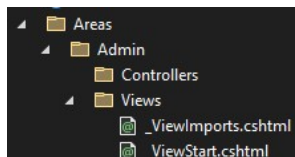
\Views_ViewImports.cshtml

```
@using CosmosWeb.ViewModels
@using CosmosModels.Models
@inject CosmosWeb.Localization.SharedLocalizer Lang
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

En gros, ce sont les imports, les usings pour toutes les vues de \Views, on y voit l'injection de Lang pour qu'il soit disponible partout

Potentiel autre intérêt des Areas (rappel : explication de l'intérêt, voyons plus tard pour les éventuels débats)

Concernant les Areas, certes ça compartimente certains aspects de l'application mais chaque Area à son propre _ViewImports et _ViewStart (qui par défaut pointe vers le même _Layout dans \Views\Shared)



Là encore, par défaut cette Area Admin pointe vers le même _Layout

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
}
```

L'intérêt est que l'on peut avoir une structure différente, des imports autres dans les vues, ou même css et js différents dans l'area, voir même une charte graphique différente par Areas, à nous de voir ce que l'on veut en faire.

Les exemples connus ce sont les applications web avec une interface d'administration différente (Wordpress, Shopify, Prestashop), un Area pour "cloisonner" le mobile, la GTA, etc

\Views\Shared_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
```

```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>@ViewData["Title"] - CosmosWeb</title>
<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
<link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
<link rel="stylesheet" href="~/CosmosWeb.styles.css" asp-append-version="true" />
<link href="~/lib/bootstrap-icons/font/bootstrap-icons.min.css" rel="stylesheet" />
</head>
<body>
  <header>
    @await Component.InvokeAsync("Menu")
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2024 - CosmosWeb - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  <script src="~/js/jquery.dirtyforms/jquery.dirtyforms.min.js"></script>
  @await RenderSectionAsync("Scripts", required: false)
  @await Component.InvokeAsync("SearchModal")
  @await Component.InvokeAsync("DetailsModal")
</body>
</html>

```

Les await Component seront pour les futurs tutos ViewComponent, ils sont là pour être déclarés sur l'ensemble de l'application

RenderBody est l'insertion des vues

RenderSectionAsync c'est pour l'insertion des scripts spécifiques à une vue (ex : une librairie calendar.js pour une vue de planning)

```

@section Scripts {
    <script>
        console.log("Ce script ne s'exécute que sur cette page.");
    </script>
}

```

Petit rappel sur tout ce qui est async, une notion qu'on rencontrera plusieurs fois dans d'autres tutos

Pourquoi utiliser RenderSectionAsync plutôt que RenderSection ?

1. Optimisation des performances : RenderSectionAsync est asynchrone et ne bloque pas le rendu de la page.
2. Compatibilité avec .NET Core et plus récent : Les bonnes pratiques recommandent l'usage de méthodes asynchrones.
3. Gestion plus propre des sections optionnelles : Permet d'attendre le chargement sans provoquer d'erreur si la section est absente.

Exercice : Utiliser RenderSectionAsync pour afficher un "Dashboard" uniquement sur la page Index

Ajouter une section Dashboard qui ne s'affichera que sur la page Index du HomeController. On utilisera RenderSectionAsync pour gérer cette section et tester son comportement avec required:true et IsSectionDefined. Vous avez aussi DevController pour tester si la section apparaît

Utiliser ViewData["Title"] côté vue ET côté contrôleur avec la trad "TestLang" (= "fr Teste Langue")

Modifier _Layout.cshtml en ajoutant juste avant </body>

```

@if (IsSectionDefined("Dashboard"))
{
    @await RenderSectionAsync("Dashboard", required: true)
}

```


Tuto 18/03

ViewModel : Principe de ToModel et ToEntity

Alors tout d'abord l'intérêt d'utiliser des ViewModels :

- Les champs persos, on crée un SocieteViewModel qui contient Societe et les champs persos donc tout dans un seul objet
- Regrouper plusieurs modèles dans une même vue (comme les champs persos)
- Séparer la logique métier (gérée par les **Models**) de la logique d'affichage (liée aux **Views**)
- Garder que les données utiles. Exemple : Un UserModel contient des champs liés à l'authentification (PasswordHash, Sel, etc), mais la vue n'a besoin que de Username et Email
- Possibilités de règles de validation unique pour la vue, exemple un format Email pour la vue mais juste un string sans contrôle en base
- Transformer les données pour qu'elles soient directement exploitables par la vue (exemple convertir les dates ou les decimales)

Voici le ToModel de Société, il est à refaire vu les modifs depuis mais le principe est là

```
public SocieteViewModel ToModel(Societe societe)
{
    var typeInformations = Context.TypeInformations.Where(x => x.IdTable == "SOCIETE").ToList();
    var informationGeneric = Context.InformationGeneriques.Where(x => x.IdAppartenance == societe.Societe1 && typeInformations.Select(t => t.NoTypeInfo).Contains(x.NoTypeInfo));
    var typeInformationListes = Context.TypeInformationListes.Where(x => typeInformations.Select(t => t.NoTypeInfo).Contains(x.NoTypeInfo)).ToList();

    var societeModel = new SocieteViewModel
    {
        Societe1 = societe.Societe1,
        AdresseRue1 = societe.AdresseRue1,
        AdresseRue2 = societe.AdresseRue2,
        CodePostal = societe.CodePostal,
        DJOuvres2 = societe.DJOuvres2,
        DtMajUsr = societe.DtMajUsr,
        EtatExterne = (int?)societe.EtatExterne,
        Nationalite = societe.Nationalite,
        NbJouvres1 = (int?)societe.NbJouvres1,
        NbJouvres2 = (int?)societe.NbJouvres2,
        NoExterne = societe.NoExterne,
        NoTypeCalendrier = (int?)societe.NoTypeCalendrier,
        NoTypeCompteurParking = (int?)societe.NoTypeCompteurParking,
        RaisonSociale = societe.RaisonSociale,
        Representant = societe.Representant,
        UniteGestion = societe.UniteGestion,
        Ville = societe.Ville,
        TypeInfo = typeInformations ?? null,
        InformationGeneriques = informationGeneric ?? null,
    };
    return societeModel;
}
```

Exemple ci-dessous fourni par Generator, les ViewModels avec des champs persos seront à modifier Post-Generator

```
public static ActiviteViewModel ToModel(Activite entity)
{
    return new ActiviteViewModel
    {
        NoActivite = (int)entity.NoActivite,
        NoTypeActivite = (int?)entity.NoTypeActivite,
        NoPersonnel = (int?)entity.NoPersonnel,
        NoRessource = (int?)entity.NoRessource,
        NoPlageActivite = (int?)entity.NoPlageActivite,
        DtDebut = entity.DtDebut,
        DtFin = entity.DtFin,
        Complement = entity.Complement,
        DtMajUsr = entity.DtMajUsr,
    };
}

2 références | Laury Poyer, Il y a 1 jour | 1 auteur, 1 modification
public static Activite ToEntity(ActiviteViewModel from, Activite to)
{
    to.NoActivite = from.NoActivite;
    to.NoTypeActivite = from.NoTypeActivite;
    to.NoPersonnel = from.NoPersonnel;
    to.NoRessource = from.NoRessource;
    to.NoPlageActivite = from.NoPlageActivite;
    to.DtDebut = from.DtDebut;
    to.DtFin = from.DtFin;
    to.Complement = from.Complement;
    to.DtMajUsr = from.DtMajUsr;
    return to;
}
```


Donc dans un contrôleur pour la vue Edit, on va utiliser ToModel pour correspondre au ViewModel déclaré en haut de la vue

```
@model ActiviteViewModel
```

```
return View(ActiviteViewModel.ToModel(activite));
```

Pour l'action POST Edit, on va utiliser ToEntity pour fournir au service l'entité modifiée

```
await Service.UpdateAsync(ActiviteViewModel.ToEntity(model, activite), cancellationToken);
```

Tuto cours donc aperçu comment c'est généré côté Generator

"Model" comme disent les anglais, mais Entité (de la base de données) pour nous de Activite

```
[Index("NoRessource", Name = "ACTIVITE_FK3")]
[Index("NoPlageActivite", Name = "ACTIVITE_FK4")]
10 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
public partial class Activite
{
    [Key]
    [Column("NO_ACTIVITE", TypeName = "numeric(7, 0)")]
    3 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public decimal NoActivite { get; set; }

    [Column("NO_TYPE_ACTIVITE", TypeName = "numeric(7, 0)")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public decimal? NoTypeActivite { get; set; }

    [Column("NO_PERSONNEL", TypeName = "numeric(7, 0)")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public decimal? NoPersonnel { get; set; }

    [Column("NO_RESSOURCE", TypeName = "numeric(7, 0)")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public decimal? NoRessource { get; set; }

    [Column("NO_PLAGE_ACTIVITE", TypeName = "numeric(7, 0)")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public decimal? NoPlageActivite { get; set; }

    [Column("DT_DEBUT", TypeName = "datetime")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtDebut { get; set; }

    [Column("DT_FIN", TypeName = "datetime")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtFin { get; set; }

    [Column("COMPLEMENT")]
    [StringLength(250)]
    [Unicode(false)]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public string Complement { get; set; }

    [Column("DT_MAJ_USR", TypeName = "datetime")]
    2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtMajUsr { get; set; }
}
```

Le ViewModel utilisé (c'est le même fichier que plus haut avec ToModel et ToEntity)

```
22 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
public class ActiviteViewModel
{
    6 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public int NoActivite { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public int? NoTypeActivite { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public int? NoPersonnel { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public int? NoRessource { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public int? NoPlageActivite { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtDebut { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtFin { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public string? Complement { get; set; }
    4 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
    public DateTime? DtMajUsr { get; set; }
}
```

2 références | Laury Poyer, il y a 2 jours | 1 auteur, 1 modification
 public static `ActiviteViewModel` ToModel(`Activite` entity)

Après plein de tests et de variables, c'est ainsi généré dans Generator (aperçu du code qui est avant à la demande)

```

    string viewModelContent = @"
using {modelNameNamespace};

namespace {viewModelNamespace}
{
    public class {modelName}ViewModel
    {
        {string.Join("\n", properties)}

        public static {modelName}ViewModel ToModel({modelType} entity)
        {
            return new {modelName}ViewModel
            {
                {string.Join("\n", toModelProperties)}
            };
        }

        public static {modelType} ToEntity({modelName}ViewModel from, {modelType} to)
        {
            {string.Join("\n", toEntityProperties)}
            return to;
        }
    }
};

File.WriteAllText(viewModelFilePath, viewModelContent);
Console.WriteLine($"Généré : {modelName}ViewModel.cs");

```

Exercice : Utiliser un nouveau ViewModel dans HomeController, on l'appellera UserSafeViewModel du modèle Utilisateur

Créer le ViewModel et l'utiliser dans la vue Index du HomeController, le but est de manipuler du Razor et voir l'autocomplétion dans l'affichage

Tuto 20/03

Réponses tuto précédent :

Plusieurs modèles : On ne peut pas utiliser plusieurs modèles dans une vue, 2 solutions, soit un ViewModel soit

```
@model (UserModel User, List<OrderModel> Orders)
```

Y'a une 3ème solution avec ViewData mais on n'est pas des bêtes 😊

Pagination version manuelle

```

public IActionResult Index(int page = 1, int pageSize = 5)
{
    var articles = _articleService.GetAll(); // Récupère tous les articles
    int totalItems = articles.Count();

    var paginatedArticles = articles
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .ToList();

    var viewModel = new PaginationViewModel<Article>
    {
        Items = paginatedArticles,
        CurrentPage = page,
        TotalPages = (int)Math.Ceiling((double)totalItems / pageSize)
    };
}

```

```

    return View(viewModel);
}

```

```

public class PaginationViewModel<T>
{
    public List<T> Items { get; set; } = new();
    public int CurrentPage { get; set; }
    public int TotalPages { get; set; }

    public bool HasPreviousPage => CurrentPage > 1;
    public bool HasNextPage => CurrentPage < TotalPages;
}

```

```

@model PaginationViewModel<Article>

<h2>Liste des articles</h2>

<ul>
    @foreach (var article in Model.Items)
    {
        <li>@article.Title</li>
    }
</ul>

<!-- Navigation de la pagination -->
<nav>
    <ul class="pagination">
        @if (Model.HasPreviousPage)
        {
            <li class="page-item">
                <a class="page-link" href="?page=@(Model.CurrentPage - 1)">Précédent</a>
            </li>
        }

        @for (int i = 1; i <= Model.TotalPages; i++)
        {
            <li class="page-item @(Model.CurrentPage == i ? "active" : "")">
                <a class="page-link" href="?page=@i">@i</a>
            </li>
        }

        @if (Model.HasNextPage)
        {
            <li class="page-item">
                <a class="page-link" href="?page=@(Model.CurrentPage + 1)">Suivant</a>
            </li>
        }
    </ul>

```

Pagination avec PageList (déjà dans le projet)

```

public IActionResult Index(int page = 1)
{
    int pageSize = 5;
    var articles = _context.Articles.ToPagedList(page, pageSize);
    return View(articles);
}

```

```

@model IPagedList<Article>

<h2>Liste des articles</h2>

<ul>
    @foreach (var article in Model)
    {
        <li>@article.Title</li>
    }
</ul>

<!-- Affichage automatique de La pagination -->
@Html.PagedListPager(Model, page => Url.Action("Index", new { page }))

```

Validation modèle/formulaire

```

public class UserController : Controller
{
    [HttpPost]
    public IActionResult Register(UserViewModel model)
    {
        if (!ModelState.IsValid)
        {
            return View(model); // On retourne la vue avec les erreurs
        }

        // Si tout est valide, on peut traiter les données
        return RedirectToAction("Success");
    }
}

```

Souvenez-vous le **Raison Social** manquant ou le asp-validation-for de jeudi dernier

```

<form asp-action="Register" method="post">
    <div>
        <label>Nom :</label>
        <input asp-for="Name" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>

    <div>
        <label>Email :</label>
        <input asp-for="Email" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>

    <button type="submit">S'inscrire</button>
</form>

```

Ben c'est les messages de ViewModel qui apparaîtront en rouge

```

public class UserViewModel
{
    [Required(ErrorMessage = "Le nom est obligatoire.")]
    public string Name { get; set; }

    [EmailAddress(ErrorMessage = "L'email est invalide.")]
    public string Email { get; set; }
}

```


Place au tuto du jour !

BaseController qu'il est bien, c'est le maître des contrôleurs

L'intérêt est lié à l'intérêt du CosmosServices et des ViewModels, **Regrouper** les fonctions et **Uniformiser** un maximum le code (+ Test Unitaire)

C'est le dernier truc un peu indigeste, la suite n'aura que 5% de matières grasses, promis

Résumé

- ✓ Évite la répétition de code dans les contrôleurs.
- ✓ Gère automatiquement l'accès aux services `Service`, `Logger`, `Lang`.
- ✓ Convertit les entités en ViewModels si `ToModel()` existe.
- ✓ Fournit des actions de base `Index()`, `Delete()`, `Details()` prêtes à l'emploi.

On peut redéfinir bien sûr si besoin comme ci-dessous

```
public new async Task<ActionResult> Index(CancellationToken cancellationToken)
```

Le BaseController est fait ainsi (je ne rentrerais pas dans le détail, même sous la torture, c'est de la magie)

```
public class BaseController<T, TService, TViewModel> : Controller where TService : class, IGenericService<T> where T : class
```

Exemple ActiviteController par Generator

```
public class ActiviteController : BaseController<Activite, IActiviteService, ActiviteViewModel>
```

Ainsi ils utilisent tous Service dépendant de leur entité reliée

```
var entity = await Service.GetByIdAsync(id, cancellationToken);
```

Ce code renvoi une entité T dans BaseController (la say nul) mais enverra une entité Activite dans ActiviteController (la say po nul)

BaseController donne accès à Service de son entité, ainsi que Logger et Lang (tuto de jeudi dernier) + fonctions Index, Details, Delete et DeleteConfirmed qui sont assez standards

```
private IStringLocalizer? _lang;
private TService? _service;
private ILogger<T>? _logger;

99+ références | Laury Poyer, il y a 6 jours | 1 auteur, 1 modification
protected IStringLocalizer Lang => _lang ??= GetLocalizer();
99+ références | Laury Poyer, il y a 6 jours | 1 auteur, 1 modification
protected TService Service => _service ??= HttpContext.RequestServices.GetRequiredService<TService>();
1 référence | Laury Poyer, il y a 6 jours | 1 auteur, 1 modification
protected ILogger<T> Logger => _logger ??= HttpContext.RequestServices.GetRequiredService<ILogger<T>>();
```

Les services sont déclarés au démarrage dans CosmosServices/ServiceRegistration.cs (les lignes sont fournies par le Generator si besoin)

```
99+ références | Laury Poyer, il y a 6 jours | 1 auteur, 2 modifications
public static void AddApplicationServices(this IServiceCollection services, Type sharedResourceType)
{
    // Add all IGenericService and GenericService
    services.AddScoped<ISocietyService, SocietyService>();
    services.AddScoped<ISiteService, SiteService>();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IAbsAutoService, AbsAutoService>();
    services.AddScoped<IAbsAutoTheoriqueService, AbsAutoTheoriqueService>();
    services.AddScoped<IAbsDemandeService, AbsDemandeService>();
    services.AddScoped<IAbsDemandeCommentaireService, AbsDemandeCommentaireService>();
    services.AddScoped<IAbsDemandeValidationService, AbsDemandeValidationService>();
    services.AddScoped<IAbsDroitService, AbsDroitService>();
    services.AddScoped<IAbsDroitAjoutService, AbsDroitAjoutService>();
    services.AddScoped<IAbsImportService, AbsImportService>();
    services.AddScoped<IAbsPriseService, AbsPriseService>();
    services.AddScoped<IAccueilService, AccueilService>();
    services.AddScoped<IAcquittementService, AcquittementService>();
    services.AddScoped<IAcquittementEtatService, AcquittementEtatService>();
    services.AddScoped<IActionService, ActionService>();
}
```

```

services.AddScoped<IActionAutomatiqueService, ActionAutomatiqueService>();
services.AddScoped<IActionKtService, ActionKtService>();
services.AddScoped<IActionObjetEvtService, ActionObjetEvtService>();
services.AddScoped<IActionOperateurService, ActionOperateurService>();
services.AddScoped<IActionPointService, ActionPointService>();
services.AddScoped<IActiviteService, ActiviteService>();
services.AddScoped<IAffectationProfilService, AffectationProfilService>();
services.AddScoped<IAffectationProfilKtService, AffectationProfilKtService>();
services.AddScoped<IAffectationProfilParentService, AffectationProfilParentService>();
services.AddScoped<IAlarmeService, AlarmeService>();
services.AddScoped<IAlerteService, AlerteService>();

```

Ainsi tous les services ont accès à ces fonctions

```

public interface IGenericService<T>
{
    Task<IEnumerable<T>> GetAllAsync(CancellationToken cancellationToken);
    Task<IEnumerable<TViewModel>> GetAllAsViewModelsAsync<TViewModel>(CancellationToken cancellationToken) where TViewModel : class;
    IQueryable<T> GetAllQuery();
    Task<T?> GetByIdAsync(object id, CancellationToken cancellationToken);
    Task AddAsync(T entity, CancellationToken cancellationToken);
    Task AddRangeAsync(IEnumerable<T> entities, CancellationToken cancellationToken);
    Task UpdateAsync(T entity, CancellationToken cancellationToken);
    Task UpdateRangeAsync(IEnumerable<T> entities, CancellationToken cancellationToken);
    Task DeleteAsync(T entity, CancellationToken cancellationToken);
    Task DeleteRangeAsync(IEnumerable<T> entities, CancellationToken cancellationToken);
}

```

Et chaque service peut avoir des fonctions définies liées à son service

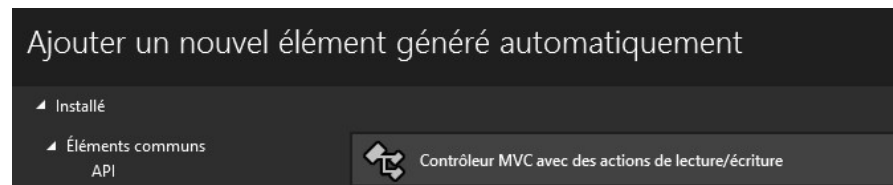
```

3 références | Laury Poyer, il y a 7 jours | 1 auteur, 1 modification
public interface ISocietyService : IGenericService<Societe>
{
    1 référence | Laury Poyer, il y a 7 jours | 1 auteur, 1 modification
    Task<Societe?> GetBySiret(string siret);
}

```

Exercice : Créer un contrôleur qui hérite de BaseController `BaseController<Model, IModelService, ModelViewModel>`

Clic droit sur le dossier Controllers puis ajouter -> Contrôleur -> Contrôleur MVC avec des action de lecture/écriture



Faites hériter BaseController à ce contrôleur avec le modèle que vous avez choisi, puis créer une vue Index et afficher le modèle choisi. Objectif : faire du Razor et utiliser Service de ce contrôleur

Tuto 25/03

Réponses au tuto précédent :

CosmosModels → Contient les classes métiers (ex: Utilisateur, Société, etc.)

CosmosServices → Contient la logique métier et les accès aux données

CosmosWeb → Projet MVC (utilise Models + Services)

CosmosAPI → Projet API (utilise Models + Services aussi)

✅ Quel est le rôle de CosmosServices ?

CosmosServices est la couche métier / logique fonctionnelle entre les contrôleurs (MVC ou API) et les modèles (CosmosModels).

On y met :

- Les traitements ou accès aux données (ex : récupérer une entité, en créer une, etc.)
- La logique commune réutilisable (via les `IGenericService`)
- Des validations, des règles métier, ou des adaptations avant/au lieu de parler directement à la BDD

C'est là qu'on code "le cerveau" de l'application.

🧱 Pourquoi un IGenericService ?

Un `IGenericService<T>` sert à :

- Mutualiser les méthodes basiques (CRUD) pour tous les modèles
- Éviter de réécrire 10 fois les méthodes `GetAll()`, `GetById()`, `Delete()`, etc.

Ensuite chaque modèle (ex : `SocieteService`, `UtilisateurService`, etc.) peut hériter d'un service générique, et ajouter sa logique spécifique seulement si besoin.

🔧 Et AddScoped dans ServiceRegistration ? (capture plus haut avec tous les AddScoped<Services>)

C'est ce qui permet à ASP.NET Core d'injecter automatiquement les services au moment où un contrôleur en a besoin.

🧠 Résumé simple

- CosmosServices = cerveau / logique métier.
- IGenericService = gain de temps en regroupant les méthodes communes.
- AddScoped = liaison automatique entre les services et leur interface (via injection de dépendances).
- MVC/API peuvent utiliser les mêmes services sans se marcher dessus.

Idee en passant : Il est possible d'avoir plusieurs bases sur un projet MVC, on pourrait très bien stocker certaines choses sur des LiteDB, MongoDB, SQLite

Bootstrap et Bootstrap Icons

- Grille responsive → Permet de structurer les pages avec un système de colonnes (`col`, `row`).
- Composants prêts à l'emploi → Boutons, cartes, formulaires, modals, etc.
- Facile à personnaliser → On peut modifier les couleurs, espacements, et autres styles via des classes CSS.

C'est déclaré dans `_Layout.cshtml` et comme d'autres frameworks CSS, on divise la grille responsive en 12 blocs

Bootstrap Icons est une bibliothèque d'icônes compatibles avec Bootstrap, intégrée dans notre projet.

Sur l'`Index.cshtml` de Dev il y a ce test

```
Test icone Bootstrap
<i class="bi bi-heart-fill"></i>
<i class="bi bi-toggles"></i>
<i class="bi bi-shop"></i>
```

Test icone Bootstrap   

Moteur de recherche d'icône sur le site officiel <https://icons.getbootstrap.com/>

- Aucune image requise, tout est en SVG (léger et rapide).

- Facile à intégrer et à personnaliser avec des classes CSS (`bi` , `bi-house-door`).
- Support natif de Bootstrap, parfait pour accompagner les composants.



Exemple de grille responsive simple

✦ Exemple de structure responsive

```
html

<div class="container">
  <div class="row">
    <div class="col-md-6">
      <h2>Colonne 1</h2>
      <p>Texte dans la première colonne.</p>
    </div>
    <div class="col-md-6">
      <h2>Colonne 2</h2>
      <p>Texte dans la deuxième colonne.</p>
    </div>
  </div>
</div>
```

👉 Ce code crée deux colonnes égales sur écran moyen et plus grand.

col-md-6 veut dire Column, Medium (écran moyen et plus) et 6 car grille de 12 donc la moitié

✦ Exemple d'un bouton stylisé

```
html

<button class="btn btn-primary">Bouton Bootstrap</button>
```

👉 Un bouton bleu avec les styles Bootstrap.

Quelques exemples pour le plaisir

Active

Much longer nav link

Link

Disabled

```

<nav class="nav nav-pills nav-fill">
  <a class="nav-link active" aria-current="page" href="#">Active</a>
  <a class="nav-link" href="#">Much longer nav link</a>
  <a class="nav-link" href="#">Link</a>
  <a class="nav-link disabled">Disabled</a>
</nav>
```

Copy

Primary

Secondary

Success

Danger

Warning

Info

Light

Dark

Link

```

<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>
<a href="#" class="btn btn-link">Link</a>
```

Copy

```
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>
```

Four directions

Four options are available: top, right, bottom, and left aligned. Directions are mirrored when using Bootstrap in RTL.

Top popover

Popover on top

Popover on right

Right popover

Bottom

Popover on left

```
<button type="button" class="btn btn-secondary" data-bs-container="body" data-bs-toggle="popover" data-bs-placement="top">
  Popover on top
</button>
<button type="button" class="btn btn-secondary" data-bs-container="body" data-bs-toggle="popover" data-bs-placement="right">
  Popover on right
</button>
<button type="button" class="btn btn-secondary" data-bs-container="body" data-bs-toggle="popover" data-bs-placement="bottom">
  Popover on bottom
</button>
<button type="button" class="btn btn-secondary" data-bs-container="body" data-bs-toggle="popover" data-bs-placement="left">
  Popover on left
</button>
```

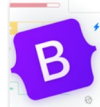
Copy

Exercice libre : Tester 1 élément layout et 2 éléments Components dans une vue

B Introduction

Get started with Bootstrap, the world's most popular framework for building responsive, mobile-first sites, with jsDelivr and a template starter page.

<https://getbootstrap.com/docs/5.1/getting-started/introduction/>



Bootstrap v5

Quickly design and customize responsive mobile-first sites with the world's most popular front-end open source toolkit.

Le but est simplement de voir les principes de Bootstrap, en plus c'est des principes similaires sur tous les autres frameworks CSS à ma connaissance (Foundation, Bulma, Tailwind, Material UI, etc)

Tuto 27/03

Les ViewComponent MVC

(À ne pas confondre avec les Components Razor qui sont des fichiers .razor utilisé en Blazor)

Alors il est tout d'abord déclaré en bas du _Layout SearchModal et DetailsModal ici

```
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - CosmosWeb</title>
  <link rel="stylesheet" href="~/Lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  <link rel="stylesheet" href="~/CosmosWeb.styles.css" asp-append-version="true" />
```

```

<Link href="~/lib/bootstrap-icons/font/bootstrap-icons.min.css" rel="stylesheet" />
</head>
<body>
  <header>
    @await Component.InvokeAsync("Menu")
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

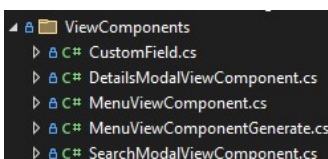
  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2024 - CosmosWeb - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  <script src="~/js/jquery.dirtyforms/jquery.dirtyforms.min.js"></script>
  @await RenderSectionAsync("Scripts", required: false)
  @await Component.InvokeAsync("SearchModal")
  @await Component.InvokeAsync("DetailsModal")
</body>
</html>

```

On peut voir aussi le Component Menu dans le header

Un ViewComponent MVC est déclaré dans \ViewComponents pour le C# et dans

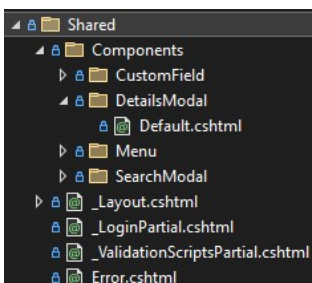
\Views\Shared\Components\NomDuComponent\Default.cshtml pour la vue (convention par défaut)



```

ViewComponents
├── CustomField.cs
├── DetailsModalViewComponent.cs
├── MenuViewComponent.cs
├── MenuViewComponentGenerate.cs
└── SearchModalViewComponent.cs

```



```

Shared
├── Components
│   ├── CustomField
│   ├── DetailsModal
│   │   └── Default.cshtml
│   ├── Menu
│   └── SearchModal
├── _Layout.cshtml
├── _LoginPartial.cshtml
├── _ValidationScriptsPartial.cshtml
└── Error.cshtml

```

Le component peut être simple, exemple DetailsModal et SearchModal n'affiche que la vue (ici est appelé Default.cshtml par défaut si jamais on avait besoin de "briser" la convention citée au-dessus)

```

public class DetailsModalViewComponent : ViewComponent
{
    0 références | Laury Poyer, il y a 9 jours | 1 auteur, 1 modification
    public IViewComponentResult Invoke()
    {
        return View();
    }
}

```

Alors que le MenuComponent renvoi une liste de MenuItemViewModel

```

public class MenuViewComponent : ViewComponent
{
    0 références | Laury Poyer, il y a 9 jours | 1 auteur, 1 modification
    public IViewComponentResult Invoke()
    {
        var currentUrl = HttpContext.Request.Path;
        var menuItems = new List<MenuItemViewModel>
        {
            new() { Name = "Society", Url = "/Society", Icon = "building" },
            new() { Name = "User", Url = "/User", Icon = "person" },
            new() { Name = "Dev", Url = "/Dev", Icon = "braces" },
        }
    }
}

```

```

new() {
    Name = "Paramètres",
    Url = "#",
    Icon = "database",
    SubMenuItems = new List<MenuItemViewModel>
    {
        new() { Name = "Sites", Url = "/Site" },
        new() { Name = "Status", Url = "/Status" },
    },
    new() {
        Name = "Generate",
        Url = "#",
        Icon = "database",
        SubMenuItems = MenuGeneratedItems.GetGeneratedItems()
    }
};

foreach (var item in menuItems)
{
    item.IsActive = item.Url.Equals(currentUrl, StringComparison.OrdinalIgnoreCase);
}

return View(menuItems);
}
}

```

Voici le code de Default.cshtml de DetailsModal, le point à retenir est que l'on remplit le tbody details-modal-body avec le retour de la fonction Details, \${type} étant le contrôleur concerné

```

<div class="modal fade" id="detailsModal" tabindex="-1" aria-labelledby="detailsModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="detailsModalLabel">Détails</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Fermer"></button>
      </div>
      <div class="modal-body">
        <table class="table">
          <tbody id="details-modal-body">
            <!-- Modal load here -->
          </tbody>
        </table>
      </div>
    </div>
  </div>
</div>

<script>
  window.loadDetailsModal = function(type, id) {
    fetch(`/ ${type}/Details?id=${id}`) // Warning backtick
    .then(response => response.json())
    .then(data => {
      let detailsTable = document.getElementById('details-modal-body');
      detailsTable.innerHTML = "";

      const ignoredFields = ["id", "password", "createdat", "sel", "motdepasse"];

      for (const key in data) {
        if (data.hasOwnProperty(key) && !ignoredFields.includes(key.toLowerCase())) {
          let row = `<tr>
            <td><strong>${key}</strong></td>
            <td>${data[key]}</td>
          </tr>`; // Warning backtick
          detailsTable.innerHTML += row;
        }
      }

      var modal = new bootstrap.Modal(document.getElementById('detailsModal'));
      modal.show();
    })
    .catch(error => console.error("Erreur lors du chargement des détails :", error));
  };
</script>

```

BaseController envoi un Json de son entité pour tous les contrôleurs

```

public async Task<ActionResult> Details(string id, CancellationToken cancellationToken)
{
    if (id == null)
        return NotFound();
    var entity = await Service.GetByIdAsync(id, cancellationToken);
    if (entity == null)
        return NotFound();
    return Json(entity);
}

```





```
}  
return @Html.ActionView();  
}
```

Appel de la fonction js dans ma vue Utilisateur vu que c'est dans _Layout je l'appelle n'importe où

```
<a asp-action="Edit" asp-route-id="@item.NoUtilisateur">Edit</a> |  
<button class="btn btn-info btn-sm" onclick="loadDetailsModal('User', @item.NoUtilisateur)">  
  <i class="bi bi-eye"></i>  
</button>
```

Ce qui donne

DtMajUsr	DureeValidite	ChangerMdp	Actions
26/09/2024 15:29:25			<div>Edit </div> <div></div> <div>Delete</div>
17/02/2025 15:25:58	0		<div>Edit </div> <div></div> <div>Delete</div>

Voici ce que renvoi Site/Details/1 dans le navigateur, directement le Json

```
Impression élégante  
[{"noSite":1,"codeSite":"SITE","libelle":"Site principal","responsable":null,"noCalendrierPeriodePaie":null,"uniteGestion":null,"noTypeCompteurParking":null,"noTypeCalendrier":null,"dTravClose":null,"dtMajUsr":"2024-09-02T09:35:15","modeVeilleNonResident":1,"noOsoConfig":null}]
```

Et version dans la modal


Détails ×

noSite	1
codeSite	SITE
libelle	Site principal
responsable	null
noCalendrierPeriodePaie	null
uniteGestion	null
noTypeCompteurParking	null
noTypeCalendrier	null
dTravClose	null
dtMajUsr	2024-09-02T09:35:15
modeVeilleNonResident	1
noOsoConfig	null

Le SearchModal a le même mode de fonctionnement si on regarde ce code

```
<!-- SearchModal -->  
<div class="modal fade" id="searchModal" tabindex="-1" aria-labelledby="searchModalLabel" aria-hidden="true">  
  <div class="modal-dialog modal-lg">  
    <div class="modal-content">  
      <div class="modal-header">  
        <h5 class="modal-title" id="searchModalLabel">Recherche</h5>  
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>  
      </div>  
      <div class="modal-body">  
        <!-- Search input -->  
        <div class="mb-3">
```

Clic sur UniteGestion d'une société

	Nationalite
	NoTypeCompteurParking
	

Recherche

×

Rechercher...

Code	Libelle	Action
1	Default	<div>✓</div>

été

Nationalite

NoTypeCompteurParking

Le code dans ModalController

```
[HttpGet]
0 références | Laury Poyer, il y a 13 jours | 1 auteur, 1 modification
public IActionResult GetListUniteGestion()
{
    var results = new List<object>();

    results =[.. _context.UniteGestions.Select(n => new
    {
        Code = n.UniteGestion1,
        n.Libelle,
        Organigramme = n.NoOrganigramme
    })];

    return Json(results);
}

[HttpGet]
0 références | Laury Poyer, il y a 13 jours | 1 auteur, 1 modification
public IActionResult GetListTypeCompteurParking()
{
    var results = new List<object>();

    results =[.. _context.TypeCompteurParkings.Select(n => new
    {
        Code = n.NoTypeCompteurParking,
        n.Libelle,
    })];

    return Json(results);
}
```

Si on revient au MenuComponent et sa vue c'est surtout un for et du css

```
@model List<CosmosWeb.ViewModels.MenuItemViewModel>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#menuNavbar" aria-controls="menuNavbar" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="menuNavbar">
            <ul class="navbar-nav me-auto">
                @foreach (var item in Model)
                {
                    if (item.SubMenuItems != null && item.SubMenuItems.Any())
                    {
                        <li class="nav-item dropdown">
                            <a class="nav-link dropdown-toggle @(item.IsActive ? "active" : "")" href="#" id="dropdown-@item.Name.Replace(" ", "")" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                                @if (!string.IsNullOrEmpty(item.Icon))
                                {
                                    <i class="bi bi-@item.Icon"></i>
                                }
                                @item.Name
                            </a>
                            <ul class="dropdown-menu" aria-labelledby="dropdown-@item.Name.Replace(" ", "")">
                                @foreach (var menuItem in item.SubMenuItems)
                                {
                                    <li>
                                        <a class="dropdown-item @(menuItem.IsActive ? "active" : "")" href="@menuItem.Url">@menuItem.Name</a>
                                    </li>
                                }
                            </ul>
                        </li>
                    }
                    else
                    {
                        <li class="nav-item">
                            <a class="nav-link @(item.IsActive ? "active" : "")" href="@item.Url">
                                @if (!string.IsNullOrEmpty(item.Icon))
                                {
                                    <i class="bi bi-@item.Icon"></i>
                                }
                                @item.Name
                            </a>
                        </li>
                    }
                }
            </ul>
        </div>
    </div>
</nav>
```


Exemple de à quoi les champs persos apparaissent dans une fiche Société

Édition de la Société

Nom	Nationalite
<input type="text" value="1"/>	<input type="text"/>
UniteGestion	NoTypeCompteurParking
<input type="text" value="1"/>	<input type="text"/>
NoTypeCalendrier	Raison sociale
<input type="text" value="1"/>	<input type="text" value="Namee"/>
TradBidon	Raison sociale
<input type="text" value="street1"/>	<input type="text"/>
CodePostal	Ville
<input type="text" value="76000"/>	<input type="text" value="Villee"/>
Representant	
<input type="text" value="moa"/>	
1	
<input type="text" value="1 - Champ12"/>	
2	
<input type="text" value="1,1"/>	
5	
<input type="text" value="12/01/1982 13:00"/>	
4	
<input type="checkbox"/> Oui	
1	
<input type="text" value="Test1"/>	

Enregistrer

Exercice pratique : Créer un ViewComponent InfoBoxComponent

1 Objectif

Créer un **ViewComponent InfoBoxComponent** qui affiche un encadré contenant une information passée en paramètre. Il sera appelé depuis _Layout.cshtml et pourra être réutilisé n'importe où.

ViewComponents/InfoBoxComponent.cs : Doit hériter de ViewComponent et donc une fonction IViewComponentResult Invoke qui renvoi View

Créer le fichier Views/Shared/Components/InfoBox/Default.cshtml et mettre du code dedans (une div Hello World suffit)

Il suffirait d'un petit truc pour appeler ce component et pouvoir le tester, mais quoi ?

Tuto 01/04

Révision Base du MVC

1. Qu'est-ce que le MVC ?

MVC = Model - View - Controller

C'est un **patron d'architecture** qui sépare clairement :

	≡ Composant	≡ Rôle
1	Model	Représente les données et la logique métier (ex : Utilisateur, Société)
2	View	Affiche l' interface utilisateur (fichiers <code>.cshtml</code>)
3	Controller	Gère les actions de l'utilisateur , appelle les services, choisit la vue à afficher

Rappel du cycle d'une requête MVC

1. L'utilisateur tape une URL (ex : `/Utilisateur/Index`)
2. Le **routeur** cherche le bon **contrôleur** (`UtilisateurController`)
3. Il appelle l'**action** `Index()`
4. Le contrôleur appelle éventuellement un **service** (ex : `UtilisateurService`) pour récupérer les données
5. Il **retourne une vue** (`Views/Utilisateur/Index.cshtml`) avec ou sans modèle

Exemple dans un `UtilisateurController.cs`

```
public async Task<IActionResult> Index()
{
    var utilisateurs = await _service.GetAllAsync();
    return View(utilisateurs); // => Views/Utilisateur/Index.cshtml
}
```

La vue `Index.cshtml`

```
<!-- View : Views/Utilisateur/Index.cshtml -->
@model List<UtilisateurViewModel>

<h2>Liste des utilisateurs</h2>

<ul>
@foreach (var user in Model)
{
    <li>@user.Nom</li>
}
</ul>
```

Points clés à retenir

- Chaque **contrôleur** hérite de `Controller` (ou `BaseController` dans notre projet)
- Le nom du fichier `.cs` est toujours **NomController**, et les vues sont dans `Views/Nom/`
- Le retour d'une action est généralement un `IActionResult` ou `Task<IActionResult>`
- On peut retourner :
 - `View(model)` → affiche une page Razor
 - `Json(objet)` → retourne du JSON
 - `RedirectToAction()` → redirige vers une autre action
 - `PartialView()` → retourne une vue partielle (ex: modals)

- `NotFound()`, `BadRequest()` → pour gérer les erreurs

1. Dans `HomeController.cs`, ajoute :

```
csharp
public IActionResult Bonjour(string nom)
{
    ViewBag.Nom = nom;
    return View(); // Views/Home/Bonjour.cshtml
}
```

2. Créer la vue `Views/Home/Bonjour.cshtml` :

```
html
<h2>Bonjour @ViewBag.Nom !</h2>
<p>Bienvenue dans cette application MVC.</p>
```

3. Tester dans l'URL :

```
/Home/Bonjour?nom=Laurent
```

Tuto 03/04

Révision `ViewImports.cshtml`, `_ViewStart.cshtml`, `_Layout.cshtml` et `Areas`

`_ViewImports.cshtml`

Rôle :

Fichier partage pour toutes les vues d'un dossier (et ses sous-dossiers). Il permet d'éviter de répéter certaines directives Razor.

Contenu courant :

- `@using` pour importer des namespaces
- `@addTagHelper` pour activer les tag helpers HTML
- `@inject` pour injecter un service dans toutes les vues (ex. `@inject IStringLocalizer Lang`)

Où le placer :

- `Views/_ViewImports.cshtml` pour tout le projet MVC
- `Areas/XXX/Views/_ViewImports.cshtml` pour une Area spécifique

`_ViewStart.cshtml`

Rôle :

Définit le layout par défaut utilise par toutes les vues du dossier concerne.

Exemple basique :

```
@{
    Layout = "_Layout";
}
```

Exemple dynamique :

```
@{
    Layout = Request.Headers["User-Agent"].ToString().Contains("Mobile") ? "_LayoutMobile" : "_Layout";
}
```

Où le placer :

- Views/_ViewStart.cshtml (global)
- Areas/XXX/Views/_ViewStart.cshtml (spécifique à une Area)

Areas

Rôle :

Permettent de structurer l'application en sous-modules indépendants (ex. Admin, Client, API).

Structure classique :

```
Areas/
├─ Admin/
│   ├── Controllers/
│   ├── Views/
│   │   └─ Home/Index.cshtml
│   ├── _ViewImports.cshtml
│   └─ _ViewStart.cshtml
```

Utilisation :

- Chaque Area a ses propres vues et contrôleurs
- L'URL prend la forme /Admin/Home/Index par exemple

Idée potentielle pour les spécifiques (client)

Exemple avec deux clients : **ClientA** et **ClientB**.

```
Areas/
├─ ClientA/
│   ├── Controllers/
│   │   └─ DashboardController.cs
│   ├── Views/
│   │   └─ Dashboard/
│   │       └─ Index.cshtml
│   ├── _ViewImports.cshtml
│   └─ _ViewStart.cshtml
├─ ClientB/
│   └─ ...
```

Exemple : Autoriser uniquement les utilisateurs du client A

```
[Area("ClientA")]
```

```
[Authorize(Policy = "ClientAOnly")]
public class DashboardController : Controller
{
    public IActionResult Index() => View();
}
```

Dans `Program.cs` :

```
services.AddAuthorization(options =>
{
    options.AddPolicy("ClientAOnly", policy =>
        policy.RequireClaim("Client", "ClientA"));
});
```

Dans les vues ou `layouts`, ne montre que ce que l'utilisateur peut voir :

```
@if (User.HasClaim("Client", "ClientA"))
{
    <a href="/ClientA/Dashboard">Accès Client A</a>
}
```

_Layout.cshtml

Role :

Définit la structure HTML globale utilisée par toutes les vues qui héritent de ce layout.

Contenu typique :

- (liens CSS, meta, etc.)
- Navigation, header/footer
- `@RenderBody()` pour afficher le contenu spécifique à chaque page
- `@RenderSection()` pour ajouter des blocs facultatifs (scripts, sidebar...)

Emplacement recommande :

Views/Shared/_Layout.cshtml

Résumé

	≡ Élément	≡ Rôle principal	≡ Emplacement
1	_ViewImports.cshtml	Usings, inject Razor partages	Views/ ou Areas/.../Views/
2	_ViewStart.cshtml	Choix du layout Razor	Views/ ou Areas/.../Views/
3	_Layout.cshtml	Template HTML principal	Views/Shared/
4	Areas	Modules MVC organisés en sous-espaces indépendants	Areas/...

Tuto 08/04

🧠 Révision : ViewModel, ToModel et ToEntity

📌 Pourquoi utiliser un ViewModel ?

Un ViewModel est un objet intermédiaire entre le Model (entité métier ou base de données) et la View (ça a créé le MVVM). Il permet :

1. D'ajouter des champs spécifiques à la vue, comme des champs persos (ex : `SocieteViewModel` qui contient `Societe` + des métadonnées).
2. De regrouper plusieurs modèles dans un seul objet (utile dans des vues complexes).
3. De séparer la logique d'affichage de la logique métier (le ViewModel ne contient que ce qui est nécessaire à l'IHM).
4. De filtrer les données sensibles ou inutiles à la vue (ex : pas besoin du `PasswordHash` dans un formulaire de profil).
5. D'ajouter des règles de validation spécifiques à la vue (`[EmailAddress]` , `[Required]` , etc.).
6. De transformer les données directement dans le ViewModel (formater une date, convertir une devise, etc.).

📄 À quoi servent ToModel() et ToEntity() ?

	☰ Méthode	☰ Sert à...
1	<code>ToModel()</code>	Convertir une entité en ViewModel pour l'afficher dans une vue
2	<code>ToEntity()</code>	Convertir un ViewModel en entité pour l'enregistrer ou le modifier

🕒 Dans un contrôleur MVC :

- GET Edit(id) → on charge l'entité, on fait un `ToModel()` pour l'envoyer à la vue
- POST Edit(ViewModel) → on fait un `ToEntity()` pour envoyer l'entité au service
-

```
// GET
public async Task<IActionResult> Edit(string id)
{
    var entite = await _service.GetByIdAsync(id);
    return View(entite.ToModel());
}

// POST
[HttpPost]
public async Task<IActionResult> Edit(UserSafeViewModel vm)
{
    var entite = vm.ToEntity();
    await _service.UpdateAsync(entite);
    return RedirectToAction("Index");
}
```

```
public class UserSafeViewModel
{
    public string Id { get; set; }
    public string Nom { get; set; }
    public string Email { get; set; }

    public static UserSafeViewModel FromEntity(Utilisateur user) => new()
    {
        Id = user.Id,
        Nom = user.Nom,
        Email = user.Email
    };

    public Utilisateur ToEntity() => new()
    {
        Id = this.Id,
        Nom = this.Nom,
        Email = this.Email
    };
}
```

Tuto 10/04

Réponses aux questions précédentes

Environnement	Commande	Commentaire
Dev/local	Update-Database	OK en dev, rapide.
Prod/client	Script-Migration → SQL	Plus sûr, versionnable, validable.

Points SQL Server, PostgreSQL, Oracle sous Entity Framework

✓ Ce qui est commun

Entity Framework Core est conçu pour être relativement **abstrait** du SGBD, donc **beaucoup de choses marchent sans changement** :

- Modèles C# (DbSet , Entity) avec propriétés simples)
- Requêtes LINQ
- Ajout/suppression/modification
- Lazy loading
- Migrations (avec adaptations)
- Navigation / relations entre entités

⚠ Les différences et problèmes potentiels

1. 🗄 Types SQL spécifiques

≡ Type C#	≡ SQL Server	≡ PostgreSQL	≡ Oracle
-----------	--------------	--------------	----------

1	decimal	numeric(18, 2)	numeric	NUMBER(18, 2)
2	string	nvarchar, varchar	text, varchar	VARCHAR2
3	DateTime	datetime	timestamp	DATE (sans time zone)

👉 **Solution** : Ne PAS forcer les `TypeName` avec `[Column(...)]` si tu veux du multi-DB. Utilise le **Fluent API** avec des `if (Database.IsNpgsql())` ou autres conditions dynamiques pour adapter.

2. 🛠 Migrations différentes selon les bases

- Les scripts de migration sont différents.
- Tu ne peux pas utiliser la même migration `code-first` pour toutes les bases à l'identique.

👉 **Solution** : Créer une migration spécifique par base (par exemple dans un dossier `Migrations.SqlServer`, `Migrations.PostgreSQL`, etc.) ou utiliser des fichiers de scripts SQL générés (`Script-Migration`) et les adapter.

3. 🗝 Gestion des clés, index et contraintes

- `IDENTITY` vs `SERIAL` vs `SEQUENCE`
- `GUID` vs `UUID`
- Index partiels, constraints spécifiques

Élément	SQL Server	PostgreSQL	Oracle
Auto-incrément (<code>[Key][Identity]</code>)	IDENTITY	SERIAL ou GENERATED	SEQUENCE + TRIGGER
Noms d'index longs	OK	OK	⚠ max 30 caractères
<code>HasColumnType()</code> manuel	OK	⚠ à adapter	⚠ très sensible
Contraintes uniques/indexes complexes	OK	OK	⚠ limité

4. 📏 Longueurs de chaînes et unicodes

- Oracle est strict sur la longueur des champs.
- PostgreSQL ne se soucie pas vraiment de la limite de `varchar`.
- SQL Server adore ses `nvarchar(450)`.

👉 Évite de faire des `StringLength` arbitraires si ce n'est pas utile, ou adapte-les en Fluent API selon la base.

5. 🧰 Fonctions SQL et LINQ traduites

Certaines expressions LINQ ne sont pas traduites de la même façon :

- `DateTime.Now`, `Contains`, `StartsWith`, `SqlFunctions` → Attention, tous ne sont pas traduits en SQL identique.
- Utiliser `EF.Functions.Like()` est plus portable.

6. 🚫 Fonctionnalités non supportées (Oracle surtout)

Oracle est le plus capricieux :

- Pas de `LIMIT` → il faut du `ROWNUM`
- Moins de support pour certains types de jointures
- Les noms sont souvent limités à 30 caractères

💡 Stratégie recommandée

Si tu veux une même appli EF Code First pour plusieurs bases, tu peux faire :

C#

```
1 protected override void OnModelCreating(ModelBuilder modelBuilder)
2 {
3     if (Database.IsSqlServer())
4     {
5         modelBuilder.Entity<Activite>().Property(p => p.NoActivite).HasColumnType("numeric(7,
6         0)");
7     }
8     else if (Database.IsNpgsql())
9     {
10        modelBuilder.Entity<Activite>().Property(p => p.NoActivite).HasColumnType("integer");
11    }
12    else if (Database.IsOracle())
13    {
14        modelBuilder.Entity<Activite>().Property(p =>
15        p.NoActivite).HasColumnType("NUMBER(7,0)");
16    }
17 }
```

Et maintenir des dossiers de migration distincts si besoin :

Shell

```
1 Add-Migration InitMigration.SqlServer -Context AppDbContext -OutputDir Migrations.SqlServer
2 Add-Migration InitMigration.PostgreSQL -Context AppDbContext -OutputDir Migrations.PostgreSQL
3
```

📁 En résumé

	≡ Point clé	≡ SQL Server	≡ PostgreSQL	≡ Oracle
1	Types numériques	numeric ou int	numeric , integer	NUMBER
2	Migrations	Update-Database	Update-Database ou script	script SQL manuel recommandé
3	Identifiants auto	IDENTITY	SERIAL , GENERATED	SEQUENCE
4	Date/Heure	datetime	timestamp	DATE
5	Particularités	Très compatible EF	Bonne compatibilité	Moins bon support EF Core

SGBD	Auto-incrément	Décimal (decimal)	Index [Index]	Migrations Update-Database
SQL Server	IDENTITY	numeric	✓	✓
Azure SQL	IDENTITY	numeric	✓	✓ (via script conseillé)

PostgreSQL	SERIAL / IDENTITY	numeric	✓	✓
SQLite	INTEGER PRIMARY KEY	⚠ REAL (double)	✗	⚠ partiel
MySQL	AUTO_INCREMENT	decimal	⚠ limites index	✓ avec Pomelo
Oracle	SEQUENCE + TRIGGER	NUMBER	⚠ noms limités	⚠ script manuel recommandé

🧠 Révision : BaseController, le maître des contrôleurs

🔗 Pourquoi BaseController est génial (même si un peu magique) :

Il est directement lié à toute la logique qu'on a vue côté `CosmosServices` et `ViewModel`. Le but :

👉 Factoriser, centraliser, uniformiser le code des contrôleurs.

👉 Rendre tout ça testable et maintenable dans la durée.

✓ Résumé clair

- Évite la répétition de code dans tous les contrôleurs.
- Gère automatiquement l'accès à :
 - `Service` (générique selon l'entité),
 - `Logger` (centralisé),
 - `Lang` (pour la localisation).
- Convertit les entités en `ViewModels` si `ToModel()` existe.
- Fournit les actions standards déjà prêtes :
 - `Index()`
 - `Details()`
 - `Delete()`
 - `DeleteConfirmed()`

📁 Exemple : ActiviteController généré automatiquement

C#

```

1 public class ActiviteController : BaseController<Activite, IActiviteService, ActiviteViewModel>
2 {
3 }
4
```

➡ Pas besoin de refaire tout le code standard : il hérite des méthodes de base, et peut ajouter ou redéfinir ce qu'il veut.

🔗 BaseController contient la "magie" générique

C#

```

1 public class BaseController<T, TService, TViewModel> : Controller
2     where T : class
3     where TService : IGenericService<T>
4 {
5     protected readonly TService Service;
6     protected readonly ILogger Logger;
7     protected readonly IStringLocalizer Lang;
8
9     public virtual async Task<IActionResult> Index() { ... }
10    public virtual async Task<IActionResult> Details(string id) { ... }
11    public virtual async Task<IActionResult> Delete(string id) { ... }
```

```

12     public virtual async Task<IActionResult> DeleteConfirmed(string id) { ... }
13 }
14

```

Tu peux redéfinir n'importe laquelle de ces méthodes dans un contrôleur concret si besoin.

🔥 Rappel : Les services sont enregistrés dans CosmosServices/ServiceRegistration.cs

```

1 services.AddScoped<IActiviteService, ActiviteService>();
2 services.AddScoped(typeof(IGenericService<>), typeof(GenericService<>));
3

```

➡ Ce registre permet à l'injection de dépendances de faire le lien automatiquement dans les contrôleurs.

💡 Bonus

- Le `Service` injecté est celui de l'entité concernée
- Le `BaseController` est pensé pour fonctionner avec les ViewModels générés automatiquement
- C'est cette base qui rend le CRUD ultra rapide à mettre en place dans CosmosWeb

Tuto 15/04

Révision 01/04 et 08/04

Tuto 17/04

Révision 03/04 et 10/04

Réponse du mardi 15

Si les pages clients sont très spécifiques → Razor Class Library (RCL) par client

→ Chaque client = un projet type `CosmosWeb.Client.MonClient` → Ça te sort une DLL autonome que tu peux plug en prod → Les vues Razor sont compilées dedans donc pas de fichiers dispersés.

Ton Startup.cs :

```

1 services.AddMvc()
2     .AddApplicationPart(typeof(MonClientController).Assembly)
3     .AddRazorRuntimeCompilation();
4

```

Solution.sln

```

|
|─ CosmosWeb                # Projet principal MVC (NET 9)
|   |─ Controllers/
|   |─ Views/
|   |─ Program.cs / Startup.cs
|
|─ CosmosWeb.Client.MonClient # Projet Razor Class Library (RCL)
|   |─ Views/

```

```
| |   └─ MonClient/  
| |       └─ Index.cshtml  
| └─ MonClientController.cs
```

1. Création du projet Razor Class Library

```
bash  
  
dotnet new razorclasslib -n CosmosWeb.Client.MonClient
```

⚠ Choisir l'option `--support-pages-and-views` si tu es en CLI.

2. Exemple de contrôleur dans le RCL

MonClientController.cs dans CosmosWeb.Client.MonClient :

```
csharp  
  
using Microsoft.AspNetCore.Mvc;  
  
namespace CosmosWeb.Client.MonClient.Controllers  
{  
    public class MonClientController : Controller  
    {  
        public IActionResult Index()  
        {  
            return View("/Views/MonClient/Index.cshtml");  
        }  
    }  
}
```

4. Dans CosmosWeb : enregistrer la DLL

A. Ajouter une référence à la DLL dans CosmosWeb.csproj :

```
xml  
  
<ItemGroup>  
  <ProjectReference Include="..\CosmosWeb.Client.MonClient\CosmosWeb.Client.MonClient.csproj" />  
</ItemGroup>
```

B. Dans Program.cs ou Startup.cs, activer la détection :

```
csharp  
  
builder.Services.AddControllersWithViews()  
    .AddApplicationPart(typeof(MonClientController).Assembly)  
    .AddRazorRuntimeCompilation(); // utile pour voir les changements sans rebuild
```

🚧 3. En prod :

- Tu déploies `CosmosWeb.dll` + `CosmosWeb.Client.MonClient.dll` uniquement
- Tu n'as pas besoin de coller les vues sur disque : elles sont compilées dans la DLL

ModalController celui qui n'est lié à aucun modèle (+component SearchModal)

CosmosServices : `IGenericService` et `GenericService`, intérêt du `CosmosServices`

Generator c'est lui le plus fort, explication du générateur de fichiers *.cs

EF Power Tools, recréer les modèles en reverse engineering + le pourquoi de `CosmosBaseContext`
