

TP N° 11

Authentification & Autorisation

Utiliser les éléments créés en TP N°10

Vous partez du projet créé :

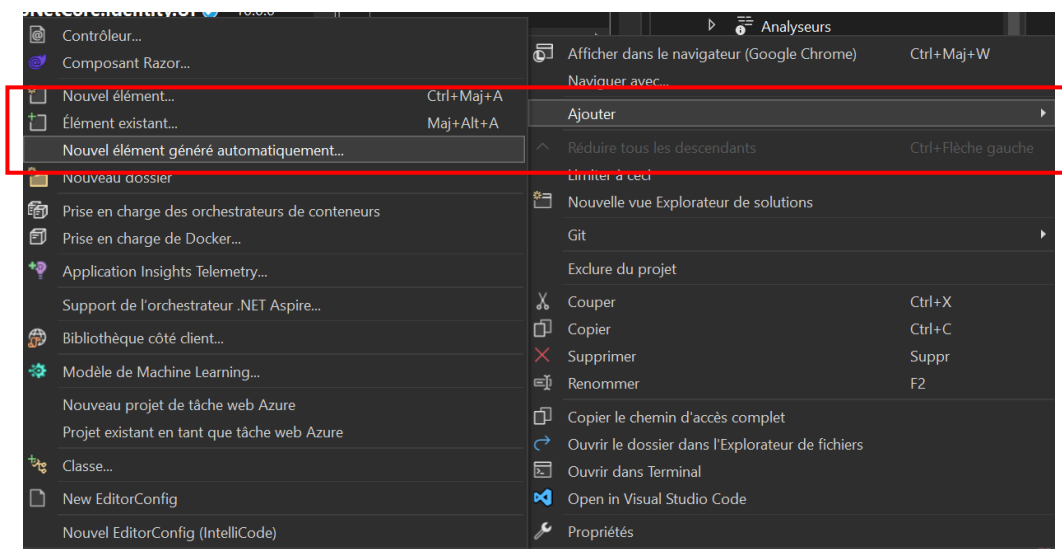
- Application **ASP.NET Core MVC (.NET 10)**
- Base de données **BD_VENTE_MIG** déjà créée par migration
- Contexte métier **VenteContext**

Partie 1 : Ajouter ASP.NET Core Identity

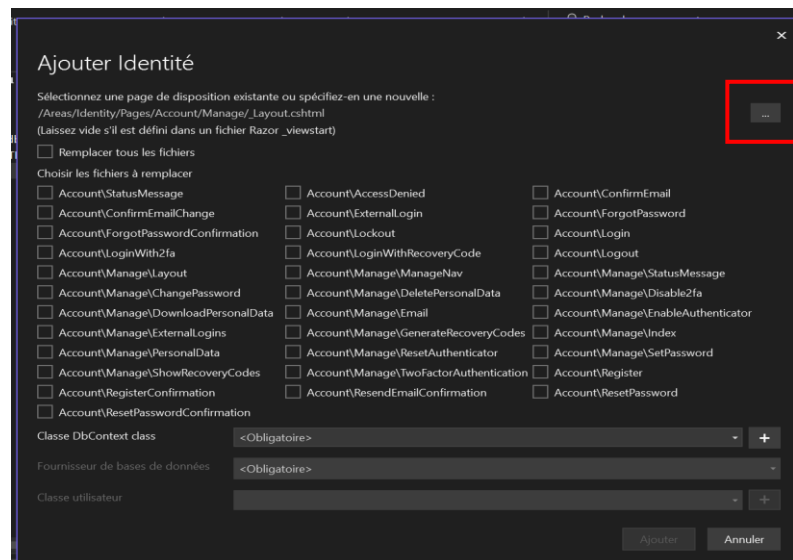
Vérifier que le package est installé : **Microsoft.AspNetCore.Identity.UI (version 10.0.0)**

1. Scaffolding de l'authentification :

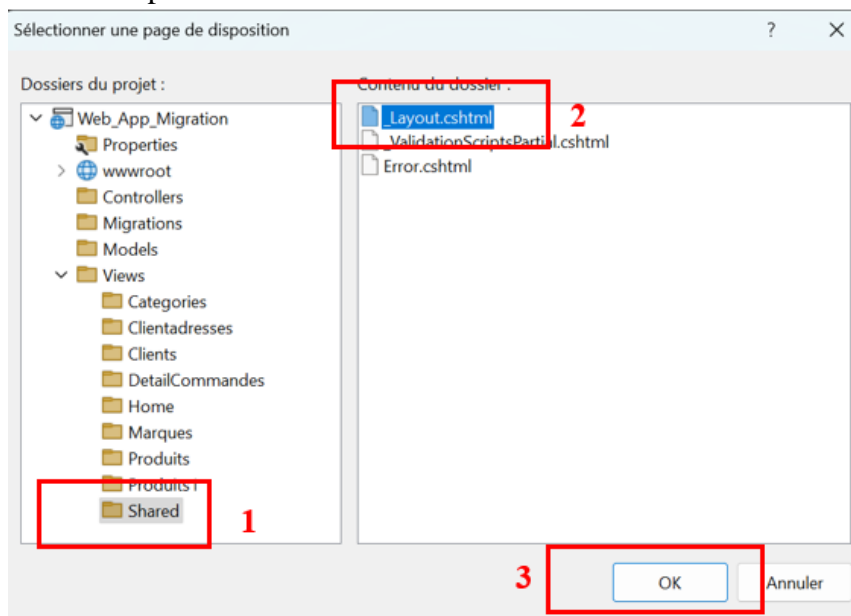
- Clic droit sur le **Projet**
- Ajouter → Nouvel élément généré automatiquement (New Scaffold item)
- Sélectionner **Identity**
- Cliquer sur **Ajouter**



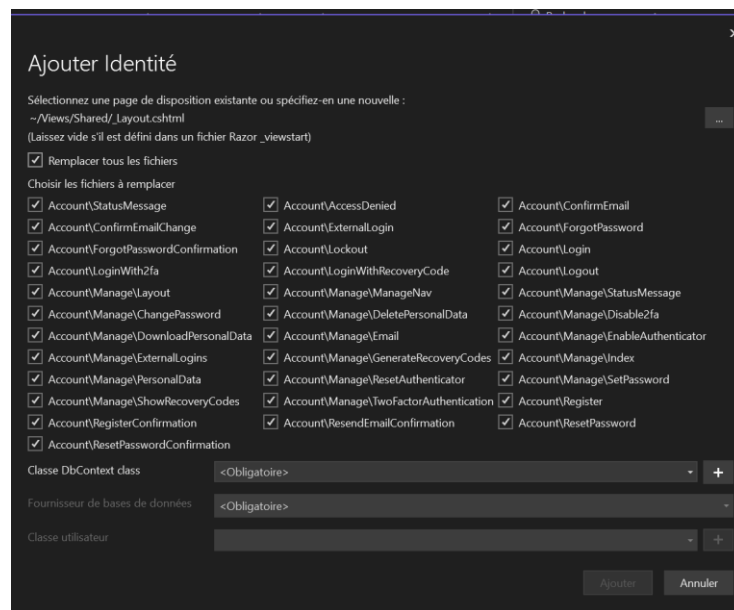
- Puis suivre les étapes suivantes :



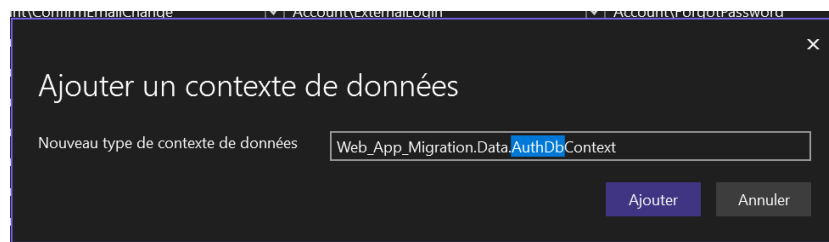
- Choisir les options suivantes :



- Laisser ces éléments cochés :



- Choisir le nom du nouveau **Context file**, exemple : *AuthDbContext*



2. Vérifier les chaînes de connexion :

- Dans *appsettings.json*, ajouter / vérifier :

```
"ConnectionStrings": {
  "VenteDb": "Data Source=YOUR_SERVER;Initial Catalog=DB_VENTE_MIG;Integrated Security=True;TrustServerCertificate=True",
  "AuthDb": "Data Source=YOUR_SERVER;Initial Catalog=BD_VENTE_AUTH;Integrated Security=True;TrustServerCertificate=True"
}
```

⇒ **VenteDb** : pour les données métier

⇒ **AuthDb** : pour les utilisateurs & rôles (nouvelle base)

Partie 2 : Configurer Identity dans Program.cs

1. Ouvrir Program.cs et ajouter (avant `var app = builder.Build();`):

```
// DB Métier
builder.Services.AddDbContext<VenteContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("VenteDb")));

// DB Auth
builder.Services.AddDbContext<AuthDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("AuthDb")));
```

2. Activer les Razor Pages (nécessaires pour Identity):

```
builder.Services.AddRazorPages();
```

3. Ajouter Identity :

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
    options.SignIn.RequireConfirmedAccount = false;
})
.AddRoles<IdentityRole>()
.AddEntityFrameworkStores<AuthDbContext>()
.AddDefaultTokenProviders();
```

4. Ajouter après var build ajouter :

```
app.MapRazorPages();
app.UseAuthentication();
app.UseAuthorization();
```

Partie 3 : Migration de la base Auth

1. Générer la migration

Dans la console PM>

```
Add-Migration IdentityInitial -Context AuthDbContext
```

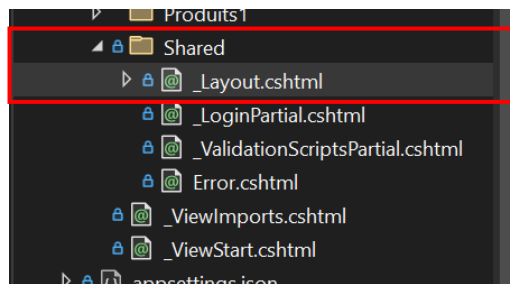
2. Appliquer la migration

```
Update-Database -Context AuthDbContext
```

- ⇒ Génère le script SQL de création des tables Identity
- ⇒ Crée un dossier Migrations pour l'authentification

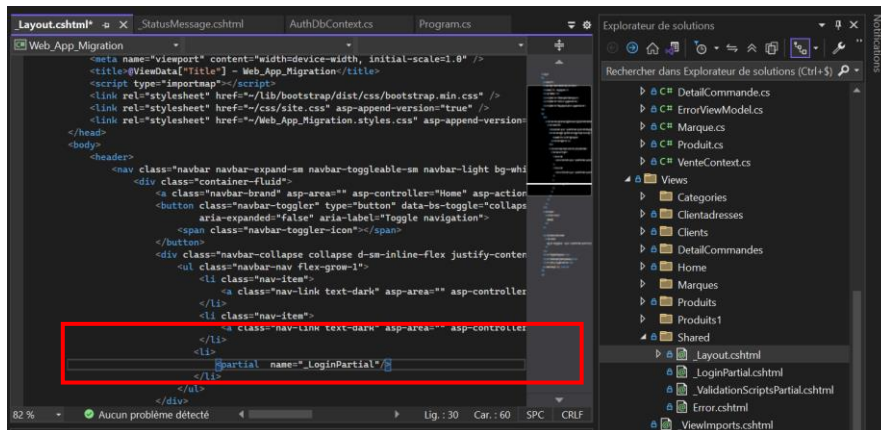
Partie 4 : Intégration du Login dans le Layout

1. Modifier dans le fichier _Layout.cshtml :



Ajouter dans la barre de navigation (nav) :

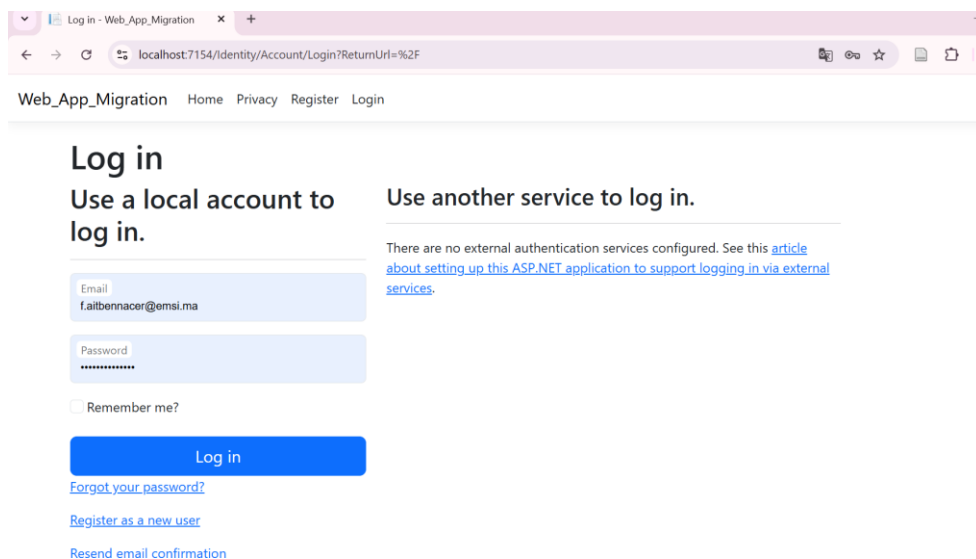
```
<li>
    <partial name="_LoginPartial"/>
</li>
```



Cela ajoute automatiquement :

- Bonjour + nom utilisateur
- Bouton Logout
- Lien Register / Login si non connecté

Partie 5 : Lancer et tester l'application



Partie 6 : Gestion des autorisations

1. Créer des rôles et Attribuer un rôle

- Créer automatiquement des rôles « Admin » et « User » **Program.cs** (après `var app = builder.Build();`)

```
// Création automatique des rôles + Admin
using (var scope = app.Services.CreateScope())
{
    var roleManager =
scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    var userManager =
scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

    string[] roles = { "Admin", "User" };
```

```

foreach (var role in roles)
{
    if (!await roleManager.RoleExistsAsync(role))
        await roleManager.CreateAsync(new IdentityRole(role));
}

// Compte admin par défaut
var adminEmail = "f.aitbennacer@emsi.ma";
var adminUser = await userManager.FindByEmailAsync(adminEmail);

if (adminUser != null && !await userManager.IsInRoleAsync(adminUser, "Admin"))
    await userManager.AddToRoleAsync(adminUser, "Admin");
}

```

2. Restriction d'accès en fonction des rôles :

- ASP.NET Core propose l'attribut **[Authorize]** avec un paramètre Rôles :

Exemple du contrôleur Produits :

```

[Authorize(Roles = "Admin")]
public class ProduitsController : Controller
{
    /*
    */
}

```

- ⇒ Cette annotation restreint **toutes** les actions du contrôleur aux utilisateurs ayant le rôle « Admin ». Les autres utilisateurs seront bloqués ou redirigés vers une page d'accès refusé.

3. Autorisation dans les vues (Razor) :

- Dans Index.cshtml du Produits (views), ajouter :

```

<p>
    @if (User.IsInRole("Admin"))
    {
        <a asp-action="Create" class="btn btn-danger">Create New (Admin)</a>
    }
</p>

```

- Exemple de résultat pour **admin**:

Web_App_Migration Home Privacy Hello f.aitbennacer@emsi.ma! Logout

Index

Create New (Admin)

| Nom | Description | Prix | Quantite | DateAjout | Disponible | Categorie | Marque |
|---------------------|---------------------------|---------|----------|------------|-------------------------------------|-----------|--------|
| Ordinateur Portable | PC 15 pouces, i5, 8Go RAM | 7990,00 | 1 | 04/12/2025 | <input checked="" type="checkbox"/> | 1 | 2 |

[Edit](#) | [Details](#) | [Delete](#)

- Exemple pour un autre rôle :

Access denied

You do not have access to this resource.

NB : si vous avez un problème lors de la création d'un nouveau compte (Register) : Enlever:

```
// Identity + Rôles
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<AuthDbContext>()
    .AddDefaultTokenProviders();
```

Et remplacer le code par :

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
    options.SignIn.RequireConfirmedAccount = false;
})
.AddRoles<IdentityRole>() // ← pour les rôles
.AddEntityFrameworkStores<AuthDbContext>()
.AddDefaultTokenProviders();
```

4. Ajout d'un rôle Manager

- Admin : tous les droits
- Manager : **modifier (Edit)** uniquement
- User : consultation (Index)

➤ Dans Program.cs, remplacer l'attribut rôle par :

```
string[] roles = { "Admin", "Manager", "User" };
```

➤ Ajouter un compte manager pour test :

| Rôle | Email | Mot de passe |
|---------|-----------------|-----------------|
| Admin | Your email | (déjà existant) |
| Manager | manager@emsi.ma | Manager@123 |
| User | via Register | selon saisie |

```
var managerEmail = "manager@emsi.ma";
var managerUser = await userManager.FindByEmailAsync(managerEmail);

if (managerUser == null)
{
    managerUser = new IdentityUser
    {
        UserName = managerEmail,
        Email = managerEmail,
        EmailConfirmed = true
    };
}
```

```

};

await userManager.CreateAsync(managerUser, "Manager@123");
await userManager.AddToRoleAsync(managerUser, "Manager");
}

```

```

// Création automatique des rôles + comptes par défaut
using (var scope = app.Services.CreateScope())
{
    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

    // Création des rôles
    string[] roles = { "Admin", "Manager", "User" };

    foreach (var role in roles)
    {
        if (!await roleManager.RoleExistsAsync(role))
            await roleManager.CreateAsync(new IdentityRole(role));
    }

    // Compte Admin par défaut
    var adminEmail = "f.aitbennacer@emsi.ma";
    var adminUser = await userManager.FindByEmailAsync(adminEmail);

    if (adminUser != null && !await userManager.IsInRoleAsync(adminUser, "Admin"))
    {
        await userManager.AddToRoleAsync(adminUser, "Admin");
    }

    // Compte Manager pour test
    var managerEmail = "manager@emsi.ma";
    var managerUser = await userManager.FindByEmailAsync(managerEmail);

    if (managerUser == null)
    {
        managerUser = new IdentityUser
        {
            UserName = managerEmail,
            Email = managerEmail,
            EmailConfirmed = true
        };

        await userManager.CreateAsync(managerUser, "Manager@123");
        await userManager.AddToRoleAsync(managerUser, "Manager");
    }
}

```

➤ Modifier les autorisations des méthodes sur ProduitsController

[Authorize] → authentification obligatoire
 [Authorize(Roles="Admin")] → Admin uniquement
 [Authorize(Roles="Admin,Manager")] → plusieurs rôles
 [AllowAnonymous] → accès libre

5. Relancer l'application