Laboratoire: Authentification

# Objectif(s)

* MS Identity
* Authentification

# Créer la branche de la fonctionnalité

Clonage du repository

1. Dans Gitkraken, clonez le repository
   * Soit via URL
   * Soit via GitHub.com
2. Ouvrez le repository (OK)

Création d’une branche pour la fonctionnalité

1. Créez une branche appelée **FCT \_Authentification**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

Dans ce laboratoire, vous aurez à décider vous-mêmes quand faire les *Comment, commit, push.*

# Configurer MS Identity

## Installer les packages

1. Microsoft.AspNetCore.Identity.EntityFrameworkCore 6.21
2. Microsoft.AspNetCore.Identity.UI 6.21

## Changer l’héritage du DbContext

1. Ouvrez le **ZombiePartyDbContext**

public class ZombiePartyDbContext : IdentityDbContext

{

public ZombiePartyDbContext(DbContextOptions<ZombiePartyDbContext> options) : base(options)

{

}

1. Ajoutez le code suivant dans le **DbContext** afin d’éviter les bogues à la génération des pages **Razor**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

//Générer des données de départ

modelBuilder.GenerateData();

}

## Générer les pages Razor MS Identity

1. Cliquez-droit sur le projet **ZombieParty**
2. Choisissez **Ajout – Nouvel élément généré automatiquement..**
3. Sélectionnez **Identité**

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

## Vérifier l’injection de dépendance et ajouter les éléments pour les Razor pages

1. Ouvrez le fichier **Program**
2. Ajoutez les injections de dépendances et les références aux pages Razor et enlevez l’option de la vérification obligatoire par courriel (elle se met automatiquement selon la version)

builder.Services.AddDefaultIdentity<IdentityUser>()

.AddEntityFrameworkStores<ZombiePartyDbContext>();

builder.Services.AddRazorPages();

1. Ajoutez les éléments dans le pipeline

app.UseAuthentication();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.MapRazorPages();

app.Run();

## Ajouter la vue partielle du login

1. Ouvrez **\_Layout**
2. Ajoutez la vue partielle **\_LoginPartial**

</ul>

<div class="col-md-2 text-right">

<**partial** **name**="\_LoginPartial" />

</div>

<div class="col-md-2 text-right">

<**partial** **name**="\_SelectLanguage" />

</div>

</div>

## Ajouter les tables aspnet

1. Si vous ne l’avez pas encore fait : update-database
2. Ajoutez une migration  **MsIdentity**
3. Update-database
4. Vérifiez que les tables ont été générées
5. Testez le ***Register***en créant un user

Vous devriez obtenir ce message qui vous indique qu’il faudrait configurer votre système pour envoyer des courriels de confirmation. On ne couvre pas cette partie dans ce cours, alors pour l’instant on clique simplement sur l’option qui nous est offerte : « Click here to confirm your account »

Une image contenant texte, capture d’écran, Police, information

Description générée automatiquement

Une fois que c’est fait, vous devriez avoir le message suivant et le nouvel utilisateur a maintenant un courriel confirmé.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

1. Vérifiez que les informations ont été enregistrées dans la table **AspNetUsers**. Vous allez peut-être remarquer qu’il y a plusieurs autres tables de Identity qui ont été ajoutées. Nous allons couvrir les rôles dans un autre laboratoire.

## Modifier l’utilisateur par défaut

1. Créez la classe **ApplicationUser** qui hérite de **IdentityUser** dans **/Models**
2. Ajoutez un champ string **NickName** qui n’existe pas dans la table **AspNetUsers**

public class ApplicationUser: IdentityUser

{

public string NickName { get; set; }

}

1. Ouvrez le fichier **ZombiePartyDbContext**
2. Ajoutez le **DbSet** pour la classe

public DbSet<Weapon> Weapons { get; set; }

public DbSet<ApplicationUser> ApplicationUsers { get; set; }

1. Ajoutez une migration **AjoutApplicationUser**

ATTENTION : La migration a également créé un champ **Discriminator**. Il existe maintenant que l’on utilise notre propre classe pour IdentityUser. Comme Discriminator un champ obligatoire (on peut le voir dans la migration), il rend les utilisateurs que l’on a déjà créé invalides. Gardons ça simple pour l’instant et **EFFACEZ** votre BD avant de faire update-database!

1. Update-database
2. Vérifiez que le champ **NickName** a été ajouté dans la table

# Modifier la page Register

Les champs *NickName* et *PhoneNumber* existent dans la table. Nous voulons que l’utilisateur puisse remplir ces champs en s’enregistrant.

## Modifier la page Register

1. Ouvrez le fichier *Areas/Identity/Pages/Account/***Register.cshtml.cs (**Regardez bien l’extension, on parle bien d’un fichier **cshtml.cs** qui est directement lié à **Register.cshtml)**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Ajoutez les deux champs dans la section *InputModel*

public class InputModel

{

/// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used

/// directly from your code. This API may change or be removed in future releases.

/// </summary>

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

public string NickName { get; set; }

public string PhoneNumber { get; set; }

1. Ouvrez le fichier **Register.chtml**
2. Sélectionnez tout le contenu de la div contenant Input.Email et copiez-le deux fois

<div class="form-floating">

<**input** **asp-for**="Input.Email" class="form-control" autocomplete="username" aria-required="true" />

<**label** **asp-for**="Input.Email"></**label**>

<**span** **asp-validation-for**="Input.Email" class="text-danger"></**span**>

</div>

1. Modifiez les tagHelper pour les deux champs

<div class="form-floating">

<**input** **asp-for**="Input.NickName" class="form-control" autocomplete="username" aria-required="true" />

<**label** **asp-for**="Input.NickName"></**label**>

<**span** **asp-validation-for**="Input.NickName" class="text-danger"></**span**>

</div>

<div class="form-floating">

<**input** **asp-for**="Input.PhoneNumber" class="form-control" autocomplete="username" aria-required="true" />

<**label** **asp-for**="Input.PhoneNumber"></**label**>

<**span** **asp-validation-for**="Input.PhoneNumber" class="text-danger"></**span**>

</div>

1. Testez la création d’un User (Dans notre exemple, PhoneNumber est simplement une string sans validation spéciale et ce n’est pas très important pour le moment)

Attention : Pour l’instant les données NickName et PhoneNumber ne sont pas sauvegardées dans la BD, nous allons voir comment le faire correctement au prochain laboratoire!

## Limiter l’accès à la vue Weapon/Index

Nous voulons limiter l’accès à la gestion des **Weapon** à un utilisateur connecté sauf pour **Index**.

1. Ouvrez le contrôleur **Weapon**
2. Ajoutez l’annotation ***Authorize***

[Authorize]

public class WeaponController : Controller

{

private ZombiePartyDbContext \_baseDonnees { get; set; }

private readonly IStringLocalizer<WeaponController> \_localizer;

1. Ajoutez l’annotation [**AllowAnonymous**] à la méthode Index

[AllowAnonymous]

public async Task<IActionResult> Index()

1. Vous pouvez maintenant vérifier qu’un utilisateur qui est connecté normalement peut utiliser Weapon sans problème. Mais que si vous n’êtes pas connecté, il est impossible d’accéder aux autres pages Edit ou Create sans d’abord se connecter.

## Commentaires et validation (Commit) des changements dans le code

1. Faites un dernier **Commit**
2. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 
3. Merge la branche **FCT \_ Authentification** dans la branche **Main** et faites un **dernier push.**

**Fin du laboratoire!**