Laboratoire: semaine 9

# Objectif(s)

* Authentification
* Autorisations
* Rôles et accès
* Seed DbInitializer.cs (authentification, autorization)
* Seed GenerateData (données autres tables)

# Créer le repo pour les laboratoires individuels

## Création de la branche de la séance

1. Dans Gitkraken, clonez votre repository de laboratoire
2. Créez la branche de la séance **S09\_Autorization**
3. *Comment-commit-push*

# Charger des Admin et roles au départ

## Créer les constantes pour les rôles

1. Dans le projet Utility, dans la classe des constantes du projet, ajoutez les Constantes pour les rôles afin d'éviter les chaines magiques harcodées
2. Ajoutez les rôles nécessaires (voir l’étude de cas de début de projet)

// Roles et Administrateur

public const string AdminRole = "Admin";

public const string CustomerRole = "Customer";

//Autres Rôles

public const string EmailAdmin = "valerie.turgeon@cegepmontpetit.ca";

## Créer le Initializer

1. Dans le projet DataAccess, au même niveau que JungleDbContext, créez un dossier **Initializer**
2. Dans ce dossier, créez un Interface publique **IDbInitializer**

public interface IDbInitializer

{

void Initialize();

}

1. Dans le même dossier, créez une classe publique **DbInitalizer** qui hérite de l'interface
2. Implémentez l'interface, la méthode **Initialize** est vide pour l'instant
3. Créez les propriétés pour le DbContext, le UserManager et RoleManager de Identity et le constructeur pour les passer en paramètre.

private readonly JungleDbContext \_db;

private readonly UserManager<IdentityUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public DbInitializer(JungleDbContext db, UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager)

{

\_db = db;

\_roleManager = roleManager;

\_userManager = userManager;

}

1. Ouvrez le fichier TXT **Jungle** **DbInitalizer** dans S09\_FichiersComplementaires
2. Allez à **\_roleManager.CreateAsync** …
3. Vérifiez le nom du rôle, en rose (nom de votre classe de constantes du projet Utility)
4. Créez une ligne pour chaque rôle (prendre Admin qui est déjà là comme exemple)

if (\_db.Roles.Any(r => r.Name == AppConst.AdminRole)) return;

     \_roleManager.CreateAsync(new IdentityRole(AppConst.AdminRole)).GetAwaiter().GetResult();

// Autres roles

1. Allez à la fin du fichier et changez le Email, pour le vôtre, trois endroits
2. Vérifiez le nom du rôle, en rose (nom de votre classe de constantes du projet Utility)

NOTE ne changez pas le mot de passe

\_userManager.CreateAsync(new ApplicationUser

{

UserName = "9999999@cegepmontpetit.ca",

Email = "9999999@cegepmontpetit.ca",

EmailConfirmed = true,

NickName= "AdminJungle",

PhoneNumber = "111111111111"

}, "Jungle1234\*").GetAwaiter().GetResult();

ApplicationUser user = \_db.ApplicationUser.FirstOrDefault(u => u.Email == "9999999@cegepmontpetit.ca");

\_userManager.AddToRoleAsync(user, AppConst.AdminRole).GetAwaiter().GetResult();

1. Dans la méthode **Initialize**, ajoutez le code du TXT
2. Injectez le Initializer dans le Startup

services.AddScoped<IUnitOfWork, UnitOfWork>();

     services.AddScoped<IDbInitializer, DbInitializer>();

# Charger des données de départ

## Créer de ModelBuilderDataGenerator

1. Dans le projet DataAccess, au même niveau que JungleDbContext, créez une classe **ModelBuilderDataGenerator**
2. Créez la méthode static **GenerateData** passant le ModelBuilder en paramètre

public static class ModelBuilderDataGenerator

{

public static void GenerateData(this ModelBuilder builder)

{

}

}

## préparer le ModelBuilderDataGenerator

1. Ouvrez le fichier TXT **Jungle Model BuilderDataGenerator** (S09\_FichiersComplementaires)
2. Vérifiez que le nom des tables et des champs correspondent à vos données, faites des modifications si nécessaire (rechercher-remplacer)
3. Copiez le contenu du fichier TXT dans la méthode **GenerateData**

## Appeler le ModelBuilderDataGenerator

1. Dans JungleDbContext, dans **OnModelCreating**, après le code FluentAPI, ajoutez l'appel du générateur de données (créé précédemment)

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Configuration fluent API

….

//Générer des données de départ

modelBuilder.GenerateData();

base.OnModelCreating(modelBuilder);

}

NOTE: Vos tables ne doivent pas contenir de données entrées via SQL Management Studio, si oui supprimez-les. Les clés primaires se réinisialisent à 1. Si vous ajoutez des données dans la méthode plus tard, elles seront AJOUTÉES à celles déjà dans la BD

1. Ajoutez une migration **SeedData**
2. Faites un **update-database**
3. *Comment-commit-push*