# 🚀 Procédure Complète pour Ajouter un Repository dans MyConcierge

## 🏗 1. Créer le Modèle de Données

Dans `MyConcierge.Domain/Models/`, ajoute une nouvelle classe représentant l'entité.

📌 Exemple : Ajout d’un Utilisateur

namespace MyConcierge.Domain.Models  
{  
 public class Utilisateur  
 {  
 public int Id { get; set; }  
 public string Nom { get; set; } = string.Empty;  
 public string Email { get; set; } = string.Empty;  
 public string Telephone { get; set; } = string.Empty;  
  
 // Clé étrangère vers ReferenceTypes (ex: Locataire, Propriétaire, Concierge)  
 public int ReferenceTypeId { get; set; }  
 public ReferenceType? ReferenceType { get; set; }  
 }  
}

## 🏗 2. Créer l'Interface du Repository

Dans `MyConcierge.Domain/Interfaces/`, ajoute une interface pour définir les méthodes d’accès aux données.

public interface IUtilisateurRepository  
{  
 Task<List<Utilisateur>> GetAllAsync();  
 Task<Utilisateur?> GetByIdAsync(int id);  
 Task AjouterAsync(Utilisateur utilisateur);  
 Task SupprimerAsync(int id);  
}

## 🏗 3. Ajouter l'Implémentation du Repository

Dans `MyConcierge.Infrastructure/Repositories/`, ajoute une classe qui implémente `IUtilisateurRepository` avec EF Core.

public class UtilisateurRepository : IUtilisateurRepository  
{  
 private readonly AppDbContext \_context;  
  
 public UtilisateurRepository(AppDbContext context)  
 {  
 \_context = context;  
 }  
  
 public async Task<List<Utilisateur>> GetAllAsync()  
 {  
 return await \_context.Utilisateurs.Include(u => u.ReferenceType).ToListAsync();  
 }  
  
 public async Task<Utilisateur?> GetByIdAsync(int id)  
 {  
 return await \_context.Utilisateurs  
 .Include(u => u.ReferenceType)  
 .FirstOrDefaultAsync(u => u.Id == id);  
 }  
  
 public async Task AjouterAsync(Utilisateur utilisateur)  
 {  
 \_context.Utilisateurs.Add(utilisateur);  
 await \_context.SaveChangesAsync();  
 }  
  
 public async Task SupprimerAsync(int id)  
 {  
 var utilisateur = await \_context.Utilisateurs.FindAsync(id);  
 if (utilisateur != null)  
 {  
 \_context.Utilisateurs.Remove(utilisateur);  
 await \_context.SaveChangesAsync();  
 }  
 }  
}

## 🏗 4. Ajouter `Utilisateurs` dans `AppDbContext`

Dans `MyConcierge.Infrastructure/AppDbContext.cs`, ajoute cette ligne pour que EF Core gère la table `Utilisateurs`.

public DbSet<Utilisateur> Utilisateurs { get; set; } // Ajout ici ✅

## 🏗 5. Enregistrer le Repository dans `Program.cs`

Dans `MyConcierge.Presentation/Program.cs`, ajoute :

builder.Services.AddScoped<IUtilisateurRepository, UtilisateurRepository>();

## 🏗 6. Générer une Migration pour Ajouter la Table

dotnet ef migrations add AddUtilisateurTable --project MyConcierge.Infrastructure --startup-project MyConcierge.Presentation

## 🏗 7. Appliquer la Migration et Mettre à Jour la Base

dotnet ef database update --project MyConcierge.Infrastructure --startup-project MyConcierge.Presentation

## 🎯 Résumé des Étapes

| Étape | Action |  
| ------ | -------------------------------------------------------------------------------------- |  
| \*\*1.\*\* | Créer `Utilisateur.cs` (modèle) |  
| \*\*2.\*\* | Créer `IUtilisateurRepository.cs` (interface du repository) |  
| \*\*3.\*\* | Créer `UtilisateurRepository.cs` (implémentation) |  
| \*\*4.\*\* | Ajouter `Utilisateurs` dans `AppDbContext.cs` |  
| \*\*5.\*\* | Ajouter `AddScoped<IUtilisateurRepository, UtilisateurRepository>()` dans `Program.cs` |  
| \*\*6.\*\* | Générer la migration : `dotnet ef migrations add AddUtilisateurTable` |  
| \*\*7.\*\* | Appliquer la migration : `dotnet ef database update` |