

Annotations

La propriété Id doit être:

- Key (Id is already a primary key By Convention)

```
[Key]
public int Id{ get; set; }
```

```
[ForeignKey("CategoryId")]
public Category myCategory { get; set; }
```

La propriété Password doit être:

- Password (hidden characters in the input) `[DataType(DataType.Password)]`
- Minimum length 8 characters `[MinLength(8)]`
- Required `[Required(ErrorMessage ="Mot de
passe Obligatoire")]`
- Not mapped in the database `< [NotMapped]`
- Password
- Same value as "Password" property `[Compare("Password")]`

La propriété Email doit être:

- Email `[Required]`
- Required `[DataType(DataType.EmailAddress)]`
`public string email{ get; set; }`
- Positive integer `[Range(0,float.MaxValue)]`
- Valid Date `[Range(0,int.MaxValue)]`
- Displayed as "Production Date" `[DataType(DataType.DateTime)]`
`[Display(Name ="Production Date")]`
- Multiligne. `[DataType(DataType.MultilineText)]`

```
[DataType(DataType.Currency)]
1 référence
public double Recompense { get; set; }
```

entier positif qui ne dépasse pas la valeur 24.

[Range(0,24]

Classe associative

la clé primaire composée de deux de la classe **Participation**.

Classe Participation

```
public int ParticipantFk { get; set; }
public int CagnotteFk { get; set; }

//prop de navig
[ForeignKey("ParticipantFk")]
public virtual Participant Participant { get; set; }
[ForeignKey("CagnotteFk")]
public virtual Cagnotte Cagnotte { get; set; }
```

Classe cagnotte

```
//prop navigation (1 cagnotte -> 1 Entreprise)
public int EntrepriseId { get; set; }
public virtual Entreprise Entreprise { get; set; }
(* participations)
public virtual ICollection<Participation> Participations { get; set; }
```

Classe Entreprise

```
//prop navigation(1 Entreprise -> * Cagnottes)
public virtual ICollection<Cagnotte> Cagnottes { get; set; }
```

Classe participant (* participation)

```
public virtual ICollection<Participation> Participations { get; set; }
```

Mapper les propriétés CodePostal, Ville et Region de l'entité Entreprise à un type complexe « Adresse » et faites les modifications nécessaires (1pt)

{[Owned]}

```
public class Address
{
    public string StreetAddress { get; set; }
    public string City { get; set; }
}
}
```

Classe entreprise

```
public Address myaddress { get; set; }
```

ENUM (dehors de la classe)

```
public enum Type { CadeauCommun, DépenseàPlusieurs, ProjetSolidaire,
Autres }
```

FLUENT API

Clé primaire composé

```
public class FactureConfiguration :
IEntityTypeConfiguration<Facture>
{
    public void Configure(EntityTypeBuilder<Facture>
builder)
    {
        builder.HasKey(f => new
        {
            f.DateAchat,
            f.ClientFk,
            f.ProductFk
        });
//Table associative
        builder.HasOne(f => f.Client)
            .WithMany(c => c.Factures)
            .HasForeignKey(f => f.ClientFk);

        builder.HasOne(f => f.Product)
            .WithMany(p => p.Factures)
            .HasForeignKey(f => f.ProductFk);
    }
}
```

En utilisant la fluent API, configurer la relation many-to-many entre **Postulant** et **Offre** en spécifiant « **Candidature** » comme nom de la table d'association. (1.5pt)

Data => Configuration => creer la classe
PostulantConfiguration

Many to Many

```
public class PostulantConfiguration : IEntityTypeConfiguration<Postulant>
{
    //Many to Many
    builder.HasMany(p => p.Offres)
```

```

        .WithMany(v => v.Postulants)
        .UsingEntity(j => j.ToTable("Candidature")); //Table d'association
    }
}

```

One to Many

```

One Cat with many prod
builder.HasOne(prod => prod.myCategory)
    .WithMany(cat => cat.products)
    .HasForeignKey(prod=>prod.CategoryId)
    .OnDelete(DeleteBehavior.ClientSetNull);

```

■ Le nom de la table correspondante à l'entité Categorie dans la base de données doit être "MyCategories"

```

builder.ToTable("MyCategories");

```

■ CategoryId est la clé primaire de la table

```

builder.HasKey(c => c.CategoryId);

```

La propriété Name est obligatoire et a une longueur maximale de 50

```

builder.Property(c =>
c.Name).IsRequired().HasMaxLength(50);

```

Configurer la relation many-to-many entre products et providers

```

builder.HasMany(prod => prod.Providers)
    .WithMany(prov => prov.Products)
    .UsingEntity(e=>e.ToTable("Providing"));

```

■ Configurer la relation one-to-many entre la class Product et Category

```

builder.HasOne(prod => prod.MyCategory)
    .WithMany(cat => cat.Products)
    .HasForeignKey(prod =>prod.CategoryId)
    .OnDelete(DeleteBehavior.ClientSetNull);

```

- Configurer le type d'entité détenu Address

■ La propriété StreetAddress a une longueur maximale de 50 et le nom de la colonne correspondante à cette propriété dans la base de données doit être "MyAddress"

■ La propriété City est obligatoire et le nom de la colonne correspondante à cette propriété dans la base de données doit être "MyCity"

```
builder.OwnsOne(c => c.MyAddress, //type d'entite detenu
adresse
    addr =>
    {
        addr.Property(a => a.StreetAddress).
HasMaxLength(50).HasColumnName("MyAddress");
        addr.Property(a => a.City)
            .IsRequired().HasColumnName("MyCity");
    });
}
```

6. Dans la classe PSContext, configurer toute les propriétés de type string et dont le nom commence par "Name"

■ Le nom des colonnes correspondantes à ces propriétés dans la base de données doit être "MyName"

```
foreach (var pb in modelBuilder.Model
    .GetEntityTypes()
    .SelectMany(t => t.GetProperties())
    .Where(p => p.ClrType == typeof(string) &&
p.Name.StartsWith("Name")))
{
    pb.SetColumnName("MyName");
}
```

configurer toute propriété qui commence par Id comme clé primaire

```
foreach (var property in
modelBuilder.Model.GetEntityTypes().SelectMany(t =>
t.GetProperties())
    .Where(p => p.Name.StartsWith("Id")))
{
    property.IsPrimaryKey();
}
```

Configurer la relation d'héritage des classes **Entraîneur** et **Joueur** par rapport à la classe **Membre** en précisant que la colonne Type est la colonne de discrimination et qu'elle aura la valeur « E » pour les enregistrements de type entraîneur et « J » pour les joueurs.

Dans context

```
modelBuilder.Entity<Membre>()  
    .HasDiscriminator<string>("Type")  
    .HasValue<Entraîneur>("E")  
    .HasValue<Joueur>("J")  
    .HasValue<Membre>("M");
```

Faites le nécessaire pour générer la base de données **FederationBD**.

```
public Context()  
{  
    Database.EnsureCreated();  
}  
  
public DbSet<Equipe> Equipe { get; set; }  
public DbSet<Joueur> Joueur { get; set; }.....  
++++  
  
static void Main(string[] args)  
{  
    System.Console.WriteLine("Hello World!");  
    Context c = new Context();  
    c.SaveChanges();  
}  
  
//Faire appel aux classes de configuration que nous venons de  
créer(1ère méthode)  
New ArticpationConfiguration().Configure(modelBuilder.Entity<Participation>());
```

```
PM> add-migration  
applet de commande Add-Migration à la position 1 du pipeline de la commande  
Fournissez des valeurs pour les paramètres suivants :  
Name: FistMigrtion  
Build started...  
Build succeeded.  
To undo this action, use Remove-Migration.  
PM> Update-database  
Build started...  
Build succeeded.  
Applying migration '20211021092226_FistMigrtion'.  
Done.  
PM> |
```

Créer une convention personnalisée qui permet de configurer toute propriété qui commence par Mail comme propriété obligatoire (**1.5pt**)

context

```
foreach(var property in modelBuilder.Model.GetEntityTypes()  
    .SelectMany(t => t.GetProperties())  
    .Where(p => p.Name.StartsWith("Mail")))  
{  
    property.IsRequired();  
}
```

Partie 2: Les Services ;

Service => IService<classe>: les entetes des methodes

=> Service<classe> : implementer les methodes

```
1 public interface IServiceEntreprise : IService<Entreprise>
2     public class ServiceEntreprise : Service<Entreprise>, IServiceEntreprise
        public ServiceEntreprise(IUnitOfWork utwk) : base(utwk)
        {
        }
```

- Retourner la liste des offres qui sont publiées **pendant le mois en cours.**

serviceOffre

```
public IEnumerable<Offre> OffresMoisEnCours()
{
    return GetMany(o => (o.DatePublication.Month == DateTime.Now.Month)
        && (o.DatePublication.Year == DateTime.Now.Year));
}
```

- **Calculer** pour chaque postulant le nombre des offres auxquelles il a postulé.

ServicePostulant

```
public int NbrOffres(int id)
{
    return GetMany().Where(p => p.IdPostulant == id).Select(p =>
        p.Offres).Count();
}
```

Retourner les 2 premières entreprises qui ont publié le plus grand nombre d'offres pour un type de contrat passé en paramètre ayant le plus grand nombre de postulants.

```
public IEnumerable<Entreprise> Top2Entreprise(string type)
{
    IDatabaseFactory factory = new DataBaseFactory();
    IUnitOfWork utwk = new UnitOfWork(factory);

    var linq = (from i in utwk.getRepository<Offre>().GetMany()
        join j in GetMany() on i.Entreprise.IdEntreprise equals
            j.IdEntreprise
        where i.TypeContrat == type
        orderby i.Postulants.Count()
        select j).Take(2);

    return linq;
}
```


Calculer le nombre d'entreprises qui appartiennent à la catégorie PME (Petite et Moyenne Entreprise). Une PME est une entreprise comprenant entre 10 et 250 salariés et dont le chiffre d'affaires annuel est inférieur à 50 millions d'euros.

```
public int NbrPME()
{
    return GetMany().Where(p => (p.Effectif > 10) && (p.Effectif <
250) && (p.ChiffreAffaire < 50000000)).Count();
}
```

Retourner la liste des entreprises groupées par secteur pour une ville passée en paramètre.

```
public IEnumerable<IGrouping<Secteur,Entreprise>> ParSecteur(string ville)
{
    var query = from entreprise in GetMany(p =>
                (p.AdresseEntreprise.Ville.Equals(ville)))
                group entreprise by entreprise.Secteur into newGroup
                orderby newGroup.Key
                select newGroup;
    return query;
}
```

- Calculer la somme des Récompenses qu'a récolté une équipe eq à travers ses trophées.

```
public double Recompense(Equipe eq)
{
    var req = GetMany().Select(eq => eq.Trophees);
    double somme = 0;
    foreach (Trophee t in req)
        somme = somme + t.Recompense;
    return somme;
}
```

- Retourner la liste des joueurs ayant participé à un trophée passé en paramètre, c'est-à-dire les joueurs qui ont fait partie de l'équipe qui a remporté le trophée pendant la date du trophée.

```
public IEnumerable<Joueur> JoueursTrophee(Trophee t)
{
    IDatabaseFactory factory = new DataBaseFactory();
    IUnitOfWork utwk = new UnitOfWork(factory);

    var req = utwk.getRepository<Contrat>().
        GetMany(c => c.EquipeFk == t.EquipeFK && (t.DateTrophee -
            c.DateContrat).Days < c.DureeMois * 30)
        .Select(c => c.Membre).OfType<Joueur>();
    return req;
}
```

- Retourner tous les entraineurs d'une équipe

```
public IEnumerable<Entraîneur> EntraîneurEquipe(Equipe e)
{
    IDatabaseFactory factory = new DataBaseFactory();
    IUnitOfWork utwk = new UnitOfWork(factory);

    return utwk.getRepository<Contrat>()
        .GetMany(c => c.EquipeFk == e.EquipeId)
        .Select(c => c.Membre).OfType<Entraîneur>();
}
```

```

public List<Provider> GetProviderByName(String Name)
{
    var query = from p in ProviderList where p.Username.Contains(Name)
                select p;
    return query.ToList();
}
0 références
public IEnumerable<Provider> GetProviderByName2(String LINQName)
{
    var query = from p in ProviderList where p.Username.Contains(LINQName)
                select p;
    return query;
}

0 références
public Provider GetFirstProviderByName(String LINQName)
{
    var query = from p in ProviderList where p.Username.Contains(LINQName)
                select p;
    return query.FirstOrDefault();
}

0 références
public Provider GetProviderById(int id)
{
    var query = from p in ProviderList where p.Id == id select p;
    return query.SingleOrDefault();
}

```

```

public IEnumerable<Chemical> Get5Chemical(double price)
{
    var query = from p in ProductList
                where p.Price > price
                select p;
    return query.Take(5).OfType<Chemical>();
}

```

```

0 références
public IEnumerable<Product> GetProductPrice(double price)
{
    var query = from p in ProductList
                where p.Price > price
                select p;
    return query.Skip(2);
}

```

```

0 références
public double GetAveragePrice()
{
    var query = from p in ProductList select p.Price;
    return query.Average();
}

```

```

0 références
public Product GetMaxPrice()
{
    var query = from p in ProductList select p.Price;
    var query2 = from p in ProductList where p.Price == query.Max() select p;
    return query2.FirstOrDefault();
}

```

Retourner la liste des cagnottes qui sont en cours (n'ont pas dépassé leurs dates limites).

```
public IEnumerable<Cagnotte> Encours()
{
    return GetMany().Where(c => c.DateLimite.CompareTo(DateTime.Now) > 0);
}
```

Retourner l'entreprise qui a créé la cagnotte ayant le plus grand nombre de participants.

```
public Entreprise PlusParticipants()
{
    var x = utwk.getRepository<Cagnotte>().GetMany().OrderBy((e => e.Participations.Count())).First();

    return x.Entreprise;
}
```

Retourner les 2 premières entreprises qui ont créé le plus grand nombre de cagnotte pour un type passé en paramètre.

```
public IEnumerable<Entreprise> Top2Entreprise(string Type)
{
    IDatabaseFactory factory = new DataBaseFactory();
    IUnitOfWork utwk = new UnitOfWork(factory);
    var linq = (from i in utwk.getRepository<Entreprise>().GetMany()
                join j in GetMany() on i.EntrepriseId equals
                    j.Entreprise.EntrepriseId
                where j.Type.ToString() == Type
                orderby i.Cagnottes.Count()
                select i).Take(2);
    return linq;
}
```

Partie III: ASP MVC

```
Vue => Dossier <classe>
// POST: EquipeController/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 références
public ActionResult Create(Equipe e, IFormFile fi
{
    e.logo = file.FileName;
```

Controllers (ajouter , 1er choix) => <classe>Controller : Cliquez droite ;
ajouter une vue (2ème choix)

STARTUP

```
services.AddScoped<IServiceClient, ServiceClient>()
        .AddScoped<IServiceConseiller, ServiceConseiller>()
        .AddScoped<IUnitOfWork, UnitOfWork>()
        .AddScoped<IDatabaseFactory, DatabaseFactory>();
```

```
public class CagnotteController : Controller
{
    private readonly IServiceCagnotte cagnotteService;
    private readonly IServiceEntreprise entrepriseService;

    public CagnotteController(IServiceCagnotte sc, IServiceEntreprise se)
    {
        cagnotteService = sc;
        entrepriseService = se;
    }
}
```

CREATE

Liste deroulante

une autre classe (1 cagnotte -> 1 entreprise)

```
// GET: CagnotteController/Create
public ActionResult Create()
{
    var entreprises = entrepriseService.GetMany();
    ViewBag.EntrepriseId = new SelectList(entreprises, "EntrepriseId",
                                           "NomEntreprise");
    return View();
}

// POST: CagnotteController/Create
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(Cagnotte c, IFormFile file)
{
    c.Photo = file.FileName;
    if (file != null)
    {
        var path = Path.Combine(Directory.GetCurrentDirectory(),
                                "wwwroot", "uploads",
                                file.FileName);
        using (System.IO.Stream stream = new FileStream(path,
                                                         FileMode.Create))
        {
            file.CopyTo(stream);
        }
    }
    try
    {
        cagnotteService.Add(c);
        cagnotteService.Commit();
        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}

- Enum      :
<div class="form-group">
    <label asp-for="Type" class="control-label"></label>
    <select asp-for="Type" class="form-control"
            asp-items="Html.GetEnumSelectList<Domain.Type>()">
    </select>
```

```

        <span asp-validation-for="Type" class="text-danger"></span>
    </div>

```

- Classe :

```

        <div class="form-group">
            <label asp-for="CategoryId" class="control-label"></label>
            <select asp-for="CategoryId" class="form-control" asp-
items="ViewBag.CategoryId"></select>
        </div>

```

Importer une image

Créer fichier uploads sous wwwroot

```

<form asp-action="Create" enctype = "multipart/form-data" >

<div class="form-group">
<label asp-for="Image" class="control-label"></label>
<input type="file" name="file" class="form-control" />
<span asp-validation-for="Image" class="text-danger"></span>
</div>

```

Rediriger une vue vers une vue Index,(dans controller)

```

return RedirectToAction(nameof(Index));

```

INDEX

```
public class EquipeController : Controller
{
    IEquipeService iequipe;
    public EquipeController(IEquipeService ie)
    {
        iequipe = ie;
    }
    // GET: EquipeController
    public ActionResult Index()
    {
        return View(iequipe.GetMany());
    }
}
```

Recherche :

un champ de recherche par nom du club,

-vue

```
<fieldset>
    <legend></legend>
    <form asp-action="index">
        <label for="filter">Nom du Club:</label>
        <input type="text" name="filter" />
        <input type="submit" value="Search" />
    </form>
</fieldset>
```

- Controller

```
// POST: Product/Index
[HttpPost]
public ActionResult Index(string filtre)
{
    var list = iprod.GetMany();
    if (!String.IsNullOrEmpty(filtre))
    {
        list = list.Where(p => p.Name.ToString().Equals(filtre)).ToList();
    }
    return View(list);
}
```

Afficher l'image

```
<td>
    
</td>
```


Ajouter un lien Entreprise à la vue Index, qui doit diriger vers une vue détails

Vue index

```
<td>  
    @Html.ActionLink(linkText: "Entreprise", actionName: "Details",  
controllerName: "Entreprise", routeValues: new { id = item.EntrepriseId },  
htmlAttributes: null)  
</td>
```

INDEX By ID (détail)

```
// GET: EntrepriseController/Details/5  
public ActionResult Details(int id)  
{  
    return View(entrepriseService.GetById(id));  
}
```