TP N° 12
LINQ to Entities

Utiliser les éléments créés en TP N°11

Exercice 1 : Filtrer les produits disponibles

1- Dans le contrôleur **ProduitsController**, modifiez l'action **Index()** pour permettre trois modes d'affichage :
   - Afficher tous les produits
   - Afficher uniquement les produits disponibles
   - Afficher uniquement les produits indisponibles
   - Ajouter un paramètre filter dans les paramètres de la méthode Index, pour applique le filtre sur Disponibilité

Exemple :

```
        var produits = _context.Produits
                               .Include(p => p.Categorie)
                               .Include(p => p.Marque)
                               .AsQueryable();
    // Filtre disponibilité
    if (filter == "disponible")
        produits = produits.Where(p => p.Disponible == true);
    /*

    */
}
```

NB : *.AsQueryable()* permet de construire la requête étape par étape

2- Modifier dans la page razor (index.cshtml de produit), pour afficher la disponibilité :

```
            @foreach (var item in Model) {
            /*

            */

           @if (item.Disponible)
           {
               <span class="badge bg-success">Disponible</span>
           }
           else
           {
               <span class="badge bg-danger">Indisponible</span>
           }
            /*

            */
        }
```

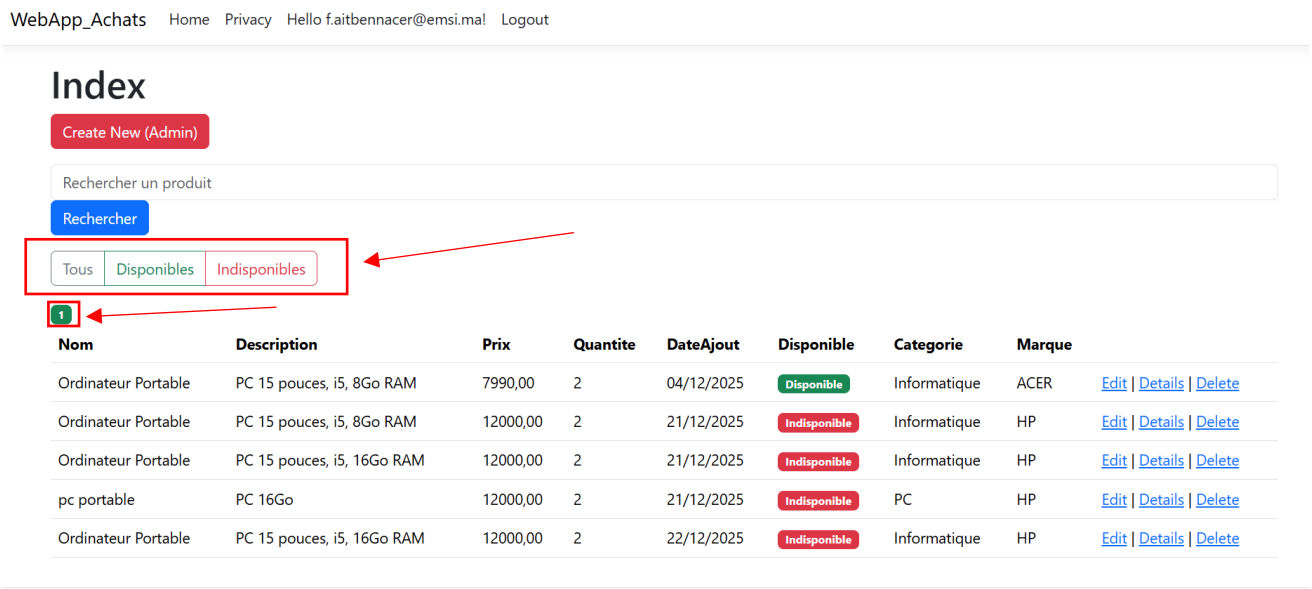
3- Ajouter les boutons de filtrage :

```
<div class="btn-group mb-3">

    <a asp-action="Index" class="btn btn-outline-secondary">Tous</a>
```

```
<a asp-action="Index" asp-route-filter="disponible" class="btn btn-outline-success">
    Disponibles
</a>

<a asp-action="Index" asp-route-filter="indisponible" class="btn btn-outline-danger">
    Indisponibles
</a>
```

4- Ajouter un compteur dans Index.cshtml :
   - Dans la vue **Index.cshtml** (au-dessus du tableau des produits), afficher un badge indiquant combien de produits sont disponibles.
     Utiliser **Count**

Résultat :



Exercice 2 : Regrouper les produits par catégorie

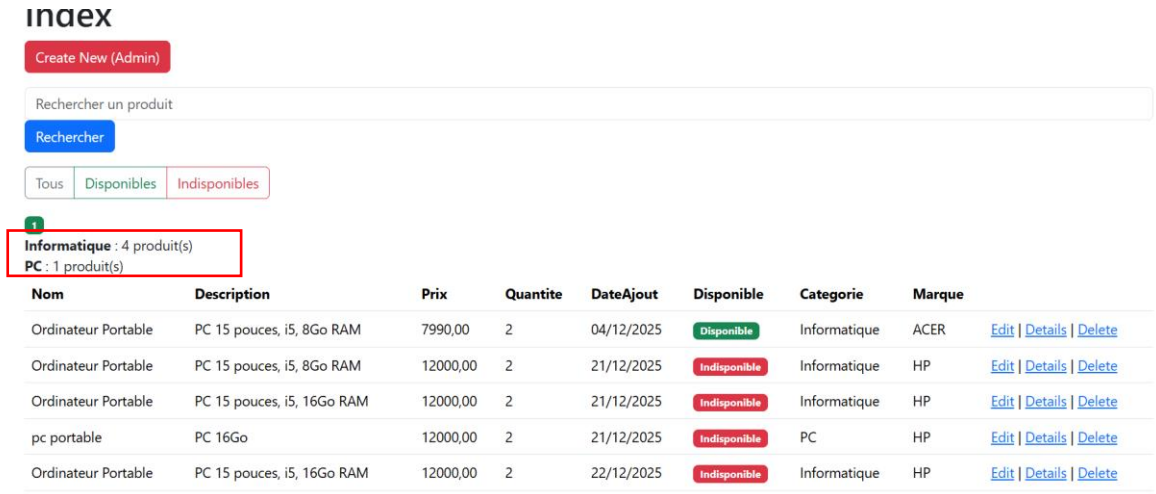
1. Dans le contrôleur **ProduitsController**, modifiez l'action **Index()** pour permettre afficher le nombre de produits par catégorie :

```
ViewBag.StatsParCategorie = _context.Produits
.Include(p => p.Categorie)
.GroupBy(p => p.Categorie.Nom)
.Select(g => new
{
    Categorie = g.Key,
    Total = g.Count()
})
.ToList();
```

2. Afficher le nombre total des produits dans la vue razor (index.cshtml), exemple :

```
@if (ViewBag.StatsParCategorie != null)
{
    foreach (var item in ViewBag.StatsParCategorie)
    {
        <div>
            <strong>@item.Categorie</strong> :
            @item.Total produit(s)
        </div>
    }
}
```

Résultat :



Exercice 3 : Tri ascendant et descendant des produits selon le prix

1. Dans le contrôleur **ProduitsController**, modifiez l'action **Index()** pour permettre trier les produits par prix :
   - Ajouter un troisième paramètre dans l'action index **sortOrder** de type string
   - Ajouter le code permettant de faire le tri ascendant et descendant en utilisant **OrderBy** et **OrderByDescending** :
   - Exemple :

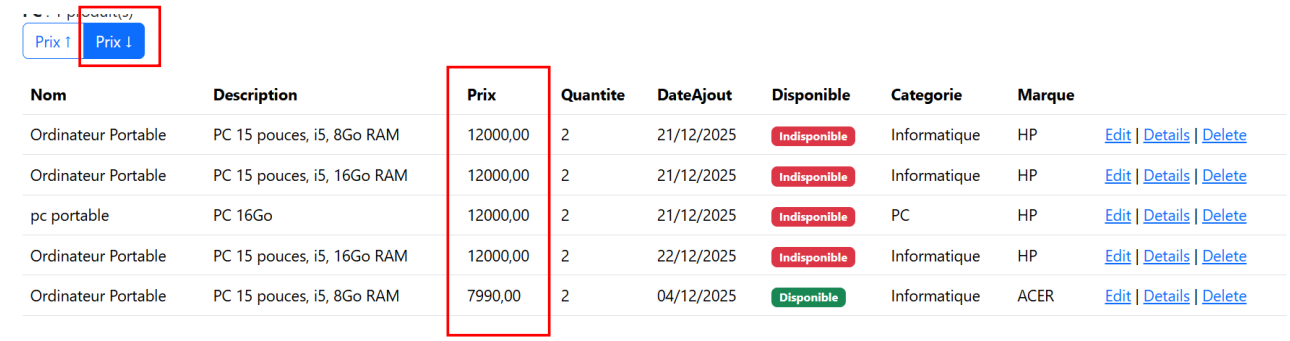
```
if (sortOrder == "prix_asc")
     produits = produits.OrderBy(p => p.Prix);
```

2. Ajouter le code dans la vue index.cshtml pour visualiser le groupage par prix :

```
<div class="btn-group mb-3">
    <a asp-action="Index" asp-route-sortOrder="prix_asc" class="btn btn-outline-primary">
        Prix ↑
    </a>

    <a asp-action="Index" asp-route-sortOrder="prix_desc" class="btn btn-outline-primary">
        Prix ↓
    </a>
   </div>
```

NB: asp-route-… sert à passer un paramètre à une action, ici l'URL est : `/Produits?sortOrder=prix_asc`

Résultat:



Exercice 4 : Afficher les produits avec catégorie :

1. Dans le contrôleur **ProduitsController**, modifiez l'action **Index()** pour permettre afficher les produits et les catégories en utilisant **Join … on … equals**
   - Ajouter le code permettant de faire la jointure :

   Exemple :

```
ViewBag.JoinProduitsCategories =
     (from p in _context.Produits
      join c in _context.Categories
      on p.CategorieId equals c.CategorieId
      select new
      {
          Produit = p.Nom,
          Prix = p.Prix,
          Categorie = c.Nom
      }).ToList();
```

2. Ajouter le code dans la vue index.cshtml pour visualiser le tableau:

   Exemple :

```
<h4>Résultats LINQ JOIN catégories</h4>

<table class="table table-bordered">
    <thead>
        <tr>
            <th>Produit</th>
            <th>Prix</th>
            <th>Catégorie</th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in ViewBag.JoinProduitsCategories)
{
        <tr>
            <td>@item.Produit</td>
            <td>@item.Prix</td>
            <td>@item.Categorie</td>
        </tr>
}
    </tbody>
</table>
```

Tableau de synthèse : Opérateurs LINQ (BD_VENTE_MIG) :

Opérateur LINQ	Syntaxe (Lambda / Query)	Rôle / Fonction	Exemple
Select	`Select(p => p.Nom)`	Projection : extraire des colonnes précises	`Select(p => new { p.Nom, p.Prix })`
Where	`Where(p => p.Disponible)`	Filtrage conditionnel	`Where(p => p.Prix > 1000)`
OrderBy	`OrderBy(p => p.Prix)`	Tri croissant	`OrderBy(p => p.Nom)`
OrderByDescending	`OrderByDescending(p => p.Prix)`	Tri décroissant	`OrderByDescending(p => p.Prix)`
ThenBy	`OrderBy(...).ThenBy(...)`	Tri secondaire	`OrderBy(p => p.Prix).ThenBy(p => p.Nom)`
GroupBy	`GroupBy(p => p.CategorieId)`	Regroupement	`GroupBy(p => p.Categorie.Nom)`

Join	`Join(Categories, ...)`	Jointure entre tables	`Join(Categories, p => p.CategorieId, c => c.Id...)`
Include	`Include(p => p.Categorie)`	Chargement relations EF	`Include(p => p.Marque)`
Any	`Any(p => p.Disponible)`	Test d'existence	`Any(p => p.Prix < 100)`
All	`All(p => p.Prix > 0)`	Test global	`All(p => p.Disponible)`
Count	`Count()`	Nombre d'éléments	`Count(p => p.Disponible)`
Sum	`Sum(p => p.Prix)`	Total numérique	`Sum(p => p.Prix)`
Max	`Max(p => p.Prix)`	Valeur maximale	`Max(p => p.Prix)`
Min	`Min(p => p.Prix)`	Valeur minimale	`Min(p => p.Prix)`
Average	`Average(p => p.Prix)`	Moyenne	`Average(p => p.Prix)`
First / FirstOrDefault	`First() / FirstOrDefault()`	Retourne le premier élément	`First(p => p.Disponible)`
Single / SingleOrDefault	`Single()`	Retourne un seul élément unique	`Single(p => p.Id == 3)`
Skip / Take	`Skip().Take()`	Pagination	`Skip(10).Take(10)`