Laboratoire: CRUD Linq et Seed

# Objectif(s)

* Seed DB
  + CRUD dans Controller
* Linq
* Validations supplémentaires :
  + Conventions et annotations de validation
  + TagHelpers

# 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\_Zombie\_CRUD**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

# Ajouter des données de départ

## Ajouter un seed

1. Faites **update-database** afin d’avoir la BD de départ de ce laboratoire
2. Démarrez l’application : elle devrait fonctionner mais sans données
3. Ouvrez le fichier **ZombiePartyDbContext**
4. Ajoutez la méthode **OnModelCreating** qui gère les configurations au démarrage de la BD
   * Appelez la méthode static de génération de données **GenerateData**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Générer des données de départ

modelBuilder.GenerateData();

}

1. Ajoutez une migration **SeedZombieZombieTypeWeapon**
2. Démarrez l’application : vous devriez voir les données pour *ZombieType, Zombie* et *Weapon*
   * Non? Avez-vous oublié **update-database?**
3. Ouvrez le fichier Data/**ModelBuilderDataGenerator**
4. Consultez le contenu de la classe static
5. Dans la Region Weapon, ajoutez un *Weapon* de votre cru
6. Démarrez l’application : Testez *Weapon*
   * Voyez-vous votre *Weapon?* Non…pourquoi?

Parce que cette techique passe par *Add-migration, update-database*

1. Ajoutez une migration **SeedMonWeapon**
2. N’oubliez pas *update-database!*
3. Démarrez l’application : Testez *Weapon*
   * Voyez-vous votre *Weapon?* Oui, yé!

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

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary**: FCT Seed Zombie ZombieType et Weapon
   * **Description**: Création données de départ pour Zombie ZombieType et Weapon
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Ajout de fonctionnalités sur Zombie

1. Allez consultez les classes du modèle : Zombie, ZombieType, HuntingLog
2. Remarquez la décoration, l’annotation [ValidateNever] au-dessus des propriétés de navigation :

Pourquoi cette annotation? Pour éviter les erreurs de validation du modèle.

## Ajout du TypeName dans l’index

1. Ouvrez le Contrôleur **ZombieController**
2. Mettez un point d’arrêt sur la dernière ligne de la fonction Index.

Une image contenant texte, Logiciel multimédia, logiciel, Police

Description générée automatiquement

1. Démarrez l’application pour tester Zombie/Index
2. Allez vérifier le contenu d’un zombie de la liste. Remarquez que ZombieType est null.

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

Description générée automatiquement

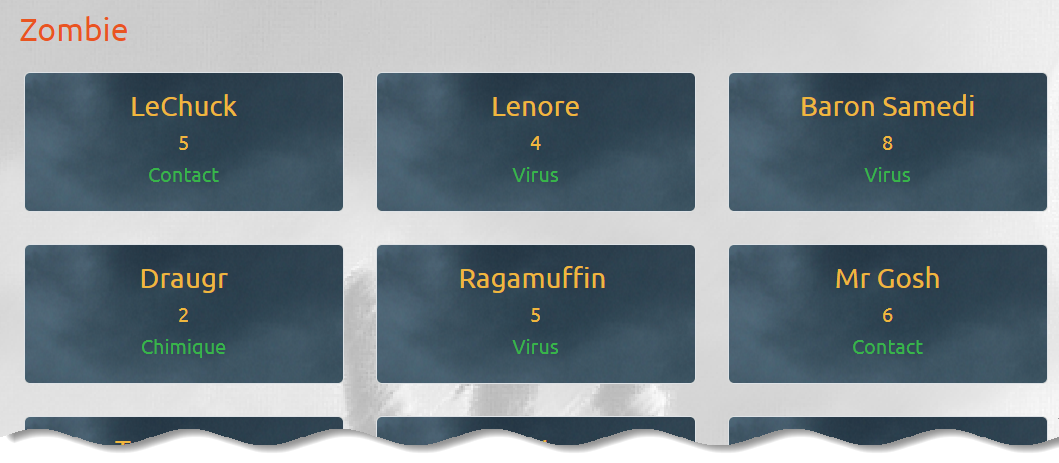
1. Arrêtez l’application
2. Modifiez le code afin d’afficher le **TypeName** du ZombieType dans l’index

Essayez de vous rappeler des notions **Linq** de Web Serveur. Si ça ne vient pas, découvrez la réponse en mettant le texte ci-dessous en noir.

List<Zombie> zombiesList = \_baseDonnees.Zombies.Include(z => z.ZombieType).ToList();

1. Démarrez l’application pour tester Zombie/Index
2. Allez vérifier le contenu d’un zombie de la liste avec votre breakpoint. Remarquez que ZombieType n’est plus null.
3. Arrêtez l’application
4. Enlevez le point d’arrêt
5. Ouvrez la View **Zombie/Index**
6. Ajoutez le **TypeName** écrit en vert (text-success), mettez-le dans un H5 après le **Point**

Ça devrait donner :



Pas certain, ça ne fonctionne pas? découvrez la réponse en mettant le texte ci-dessous en noir.

<h5> @item.Point </h5>

<h5 class ="text-success"> @item.ZombieType.TypeName </h5>

## Modification du tri dans l’index

1. Ouvrez le Contrôleur **ZombieController**
2. Modifiez le code afin de trier les Zombies par leur **Name**

Essayez de vous rappeler des notions Linq de Web Serveur. Si ça ne vient pas, découvrez la réponse en mettant le texte ci-dessous en noir.

List<Zombie> zombiesList = \_baseDonnees.Zombies.OrderBy(z => z.Name).Include(z => z.ZombieType).ToList();

1. Démarrez l’application pour tester Zombie/Index
2. Allez vérifier que les zombies sont en ordre alphabétique selon leur Name.
3. Arrêtez l’application

## Ajout de la liste des meilleurs zombies

1. Ouvrez le Contrôleur **ZombieController**
2. Copiez l’action Index et changez le nom pour **StrongestZombies**
3. Ajoutez au code existant ce qu’il faut afin de n’afficher que les zombies ayant **8 points et plus**

Essayez de vous rappeler des notions Linq de Web Serveur. Si ça ne vient pas, découvrez la réponse en mettant le texte ci-dessous en noir.

List<Zombie> zombiesList = \_baseDonnees.Zombies.Where(z => z.Point >= 8).OrderBy(z => z.Name).Include(z => z.ZombieType).ToList();

1. Mettez un point d’arrêt sur la ligne de **Return View**
2. Démarrez l’application pour tester via l’URL **Zombie/StrongestZombies** (Pour l’instant on regarde seulement dans Visual Studio, on va ajouter la View après)
3. Vérifiez que ZombiesList ne contient que les trois zombies ayant 8 points ou plus.
4. Arrêtez l’application
5. Copiez la View Zombie/Index et modifiez le nom pour **StrongestZombies**
6. Ouvrez votre nouvelle vue et supprimez le div qui contient le bouton Create Zombie (ça n’a pas sa place ici)
7. Dans le \_Layout, remplacez l’élément « something else here » par un lien vers votre nouvelle page

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

Description générée automatiquement

1. Vérifiez que votre nouvelle page fonctionne

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

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary :** FCT Zombie Index
   * **Description**: Ajout du TypeName dans la view Zombie Index
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

## Ajout de la méthode Edit

**GET**

1. Ouvrez le Contrôleur **ZombieController**
2. Faites une copie de la méthode GET **Create** et appelez-la **Edit**
3. Modifiez la méthode afin qu’elle trouve et retourne le zombie dont le Id est passé en paramètre :
   1. Ajoutez un paramètre int id
   2. Ajoutez au ZombieVM le zombie correspondant au id

public IActionResult Edit(int id)

{

ZombieVM zombieVM = new ZombieVM();

zombieVM.Zombie = \_baseDonnees.Zombies.Find(id);

zombieVM.ZombieTypeSelectList = \_baseDonnees.ZombieTypes.Select(t => new SelectListItem

{

Text = t.TypeName,

Value = t.Id.ToString()

}).OrderBy(t => t.Text);

return View(zombieVM);

}

1. Mettez un point d’arrêt sur la ligne de **Return View** (Encore une fois, on test seulement dans VisualStudio il n’y a pas encore de vue)
2. Démarrez l’application pour tester avec le Zombie ayant le Id 2 via l’URL **/Zombie/Edit/2**
3. Vérifiez que les informations retournées sont bien celles de **Lenore**
4. Arrêtez l’application
5. Désactivez le Point d’arrêt sans le supprimer (bouton droit de la souris)
6. Copiez la View Zombie **Create** et appelez-la **Edit**
7. Ouvrez la view Zombie **Edit**
8. Faites les modifications afin d’adapter la View à la modification
   1. Modifiez le titre dans le ViewData et le h1
   2. L’attribut *value* du bouton *submit* du formulaire
   3. Le *tagHelper* doit appeler l’action **Edit**

<div class="col-md-4">

<div class="row p-3 ">

<**form** **asp-action**="Edit">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

1. Démarrez l’application pour tester avec le Zombie ayant le Id 2 via l’URL **/Zombie/Edit/2**
2. Vérifiez que les informations retournées sont bien celles de **Lenore** et que c’est la viewEdit
3. Arrêtez l’application (Il faut encore ajouter le post pour que ça soit fonctionnel)

**POST**

1. Faites une copie de la méthode POST **Create** et appelez-la **Edit**
2. Ajoutez la décoration pour éviter est d'empêcher les attaques de falsification de requêtes. Il serait sage de l’ajouter également dans le Create

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Edit(ZombieVM zombieVM)

1. Modifiez la méthode afin qu’elle modifie le zombie du ZombieVM passé en paramètre

//Si le modèle est valide le zombie est modifié et nous sommes redirigé vers index.

if (ModelState.IsValid)

{

\_baseDonnees.Zombies.Update(zombieVM.Zombie);

\_baseDonnees.SaveChanges();

TempData["Success"] = $"Zombie {zombieVM.Zombie.Name} has been modified";

1. Mettez un point d’arrêt sur la ligne if (ModelState.IsValid)
2. Démarrez l’application pour tester avec le Zombie ayant le Id 2 via l’URL **/zombie/Edit/2**
3. Modifiez la description
4. Faites F10
5. Allez vérifier votre *ModelState :* Il est valide MAIS si on regarde de plus près, il y a tout de même un problème!
6. Allez vérifier le contenu du ViewModel passé en paramètre :

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

Description générée automatiquement

Le id est vide. Pourquoi? Il ne figure pas dans la View.

1. Arrêtez l’application. (Si vous n’arrêtez pas l’application, il va rouler le update avec un Id de zéro. Le comportement par défaut dans ce cas, c’est d’ajouter une nouvelle entrée et de générer un nouvel Id. Vous allez donc multiplier les zombies!).
2. Ajoutez le dans le formulaire, mais caché, il n’y a pas d’avantage à montrer la clé primaire

<**form** **asp-action**="Edit">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group row">

<**input** **asp-for**="@Model.Zombie.Id" class="form-control" hidden/>

<div class="col-4">

<**label** **asp-for**="@Model.Zombie.Name" class="control-label"></**label**>

</div>

1. Démarrez l’application pour tester avec le Zombie ayant le Id 2 via l’URL **/zombie/Edit/2**
2. Modifiez la description
3. Faites F10
4. Allez vérifier votre *ModelState :* Il est valide!
5. Allez vérifier le contenu du ViewModel passé en paramètre : Tout est là? Yé!
6. Arrêtez l’application
7. Dans Zombie Index, ajoutez un bouton afin de pouvoir modifier le Zombie, ajoutez les taghelpers nécéssaires

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

Description générée automatiquement

<h5 class ="text-success"> @item.ZombieType.TypeName </h5>

<**a** **asp-action**="Edit" **asp-route-Id**="@item.Id" class="btn btn-info">

Edit

</**a**>

</div>

1. Testez le passage de la view Index à Edit

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

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary :** FCT Zombie Edit
   * **Description**: Ajout de la View Edit et des actions dans ZombieController
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

## Ajout de la méthode Delete

1. Ouvrez le Contrôleur **ZombieController**
2. Copiez la méthode GET Edit et appelez-la **Delete**
3. Nettoyez le code afin d’enlever les éléments non nécessaires pour que l’utilisateur puisse confirmer l’opération (Delete) sur le bon Zombie.
4. Créez une vue **Delete** correspondante. Avez-vous besoin d’afficher toutes les informations du Zombie?
5. N’oubliez pas de tester que votre code fonctionne tel que prévu
6. Ajoutez une méthode POST DeletePost qui prend id en paramètre (Pourquoi DeletePost et pas simplement Delete? C# ne permet pas d’avoir 2 méthodes qui portent le même nom et qui ont les même paramètres)

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult DeletePost(int id)

{

Zombie? zombie = \_baseDonnees.Zombies.Find(id);

if (zombie == null)

{

return NotFound();

}

\_baseDonnees.Zombies.Remove(zombie);

\_baseDonnees.SaveChanges();

TempData["Success"] = $"Zombie {zombie.Name} terminated";

return RedirectToAction("Index");

}

1. Mettez un point d’arrêt à la ligne \_baseDonnees.SaveChanges();
2. Démarrez l’application pour tester avec le Zombie ayant le Id 2 via l’URL **/Zombie/Delete/2**
3. Arrêtez l’application
4. Dans Zombie Index, ajoutez un bouton afin de pouvoir supprimer le Zombie, ajoutez les taghelpers nécéssaires

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

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modification: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary :** FCT Zombie Delete
   * **Description**: Ajout de l’action Delete dans ZombieController
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 
7. Merge la branche **FCT\_Zombie\_CRUD** dans la branche **Main** et faites un **autre push.**

# Ajout de fonctionnalités sur ZombieType

1. Créez une branche appelée **FCT\_ZombieType\_CRUD**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL
3. Ajoutez les fonctionnalités suivantes dans ZombieType :

* Edit
* Delete

1. N’oubliez pas d’ajouter les boutons dans la View Index
2. Testez par étapes, utilisez les points d’arrêt.
3. Faites des commits fréquents et pertinents.

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

1. Une fois que tout est terminé, n’oubliez pas de faire un push.
2. Merge la branche **FCT\_ZombieType\_CRUD** dans la branche **Main** et faites un **dernier push.**

**Fin du laboratoire!**