Laboratoire: révision MVC

# Objectif(s)

**GitHub**

* Clonage
* Création de branches
* Commit/Push/Merge

**Révision 2W5**

* Views
* ViewModels
* Annotations
* ModelState

**NOTE :** Dans les laboratoires, vous ferez beaucoup plus de branches et de *merge* que nécessaire : c’est pour la pratique. Il est plutôt rare de faire trois ou quatre branches en deux heures! Faites-les quand même, parfois il y a des surprises!

# Repository GitHub

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

# Création de la base de données en listes

## Associer un ZombieType à aux Zombies

1. Ouvrez la solution **ZombieParty**
2. Ouvrez la classe **Zombie**
3. Remplacez la propriété Type par la clé étrangère liée à **ZombieType**

public class Zombie

{

public string Name { get; set; }

public int Point { get; set; }

~~public string Type { get; set; }~~

public int ZombieTypeId { get; set; }

public ZombieType ZombieType { get; set; }

}

## Création de la classe de l’entrepôt de données

**NOTE et RAPPEL** : Ce n’est pas la meilleure façon d’enregistrer des données de façon permanente, nous le verrons dans les prochaines séances. Cette technique permet cependant de partager la même instance de **BaseDonnees** dans toute l’application (tant qu’elle roule! Les modifications seront perdues dès la fermeture), appelé un *Singleton*.

1. Créez une classe public appelée **BaseDonnees** dans le dossier **Models**
2. Ajoutez le code ci-dessous afin de créer des listes de données, tel que vu dans le cours **2W5**

public class BaseDonnees

{

public List<ZombieType> ZombieTypes { get; set; }

public List<Zombie> Zombies { get; set; }

public BaseDonnees()

{

//Initialiser des listes vides

Zombies = new List<Zombie>();

//Remplir la liste des ZombieTypes

ZombieTypes = new List<ZombieType>()

{

new ZombieType() { TypeName = "Virus", Id = 1 },

new ZombieType() { TypeName = "Contact", Id = 2 },

new ZombieType() { TypeName = "Radioactif", Id = 3 }

};

//Remplir la liste des Zombies

Zombies.Add(new Zombie() { Name = "LeChuck", Point = 5, ZombieTypeId = 2 });

Zombies.Add(new Zombie() { Name = "Lenore", Point = 4, ZombieTypeId = 2 });

Zombies.Add(new Zombie() { Name = "Draugr", Point = 2, ZombieTypeId = 3 });

Zombies.Add(new Zombie() { Name = "Ragamuffin", Point = 5, ZombieTypeId = 1 });

Zombies.Add(new Zombie() { Name = "Taxidermy", Point = 1, ZombieTypeId = 2 });

Zombies.Add(new Zombie() { Name = "Chien", Point = 7, ZombieTypeId = 1 });

Zombies.Add(new Zombie() { Name = "Avogadro", Point = 9, ZombieTypeId = 3 });

}

}

## Injection de la dépendance à la BD dans program

1. Ouvrez le fichier **Program.cs**
2. Injectez la dépendance à **BaseDonnees** en Singleton  
   avant le builder.Buid()

// Add services to the container.

builder.Services.AddControllersWithViews();

// Injection des dépendances

builder.Services.AddSingleton<BaseDonnees>();

var app = builder.Build();

## 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 Base de données en listes et liaisons
   * **Description**: Ajout de la classe contenant la base de données en listes et ajout de la liaison entre Zombie et ZombieType. Injection du Singleton
5. Cliquez sur **Commit**

NOTE: les changements sont enregistrés dans le repo LOCAL SEULEMENT

1. "Poussez" (**Push)** les modifications sur le repo en ligne (**Remote).** Cliquez sur 
2. C’est la fin de la fonction Base de Données (BD)
3. Merge la branche **FCT\_BD** dans la branche **Main.**
4. "Poussez" (**Push) MAIN** les modifications sur le repo en ligne (**Remote).** Cliquez sur 

# Repository GitHub

## Création d’une branche pour la fonctionnalité

1. Créez une branche appelée **FCT\_Zombie**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

# Modifier ZombieTYpe

## Modifier le controller ZombieType

Modification du Controller afin d’utiliser le Singleton de BaseDonnees tel que vu en 2W5.

1. Ouvrez le ZombieTypeController
2. Ajoutez la variable privée et injectez la dépendance dans le contructeur du Controller

private BaseDonnees \_baseDonnees { get; set; }

public ZombieTypeController(BaseDonnees baseDonnees)

{

\_baseDonnees = baseDonnees;

}

1. Remplacez l’action Index afin d’utiliser le Singleton de BaseDonnees

public IActionResult Index()

{

this.ViewBag.MaListe = \_baseDonnees.ZombieTypes.ToList();

return View();

}

1. Modifiez l’action Create POST afin d’utiliser le Singleton, de valider le ModelState et d’afficher les erreurs

//POST

[HttpPost]

public IActionResult Create(Models.ZombieType zombieType)

{

if (ModelState.IsValid)

{

// Ajouter à la BD

\_baseDonnees.ZombieTypes.Add(zombieType);

return this.RedirectToAction("Index");

}

return this.View(zombieType);

}

1. Testez la création d’un type de zombie

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

Description générée automatiquement

1. Modifiez le texte pour obtenir le displayName directement avec les paramètres du message d’erreur.

public class ZombieType

{

public int Id { get; set; }

[DisplayName("Type Name")]

[Required(AllowEmptyStrings = false, ErrorMessage = "{0}Type Name has to be filled.")]

public string TypeName { get; set; }

}

## 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 ZombieType utilization du singleton
   * **Description**: Ajout de l’utilisation du singleton BaseDonnees pour ZombieType.
5. Cliquez sur **Commit**

# Modifier Zombie

## Modifier le controller Zombie l’action index

1. Ouvrez le **ZombieController**
2. Ajoutez la variable privée et injectez la dépendance dans le constructeur

private BaseDonnees \_baseDonnees { get; set; }

public ZombieController(BaseDonnees baseDonnees)

{

\_baseDonnees = baseDonnees;

}

1. Modifiez l’action Index

public IActionResult Index()

{

this.ViewBag.MaListe = new List<Zombie>();

this.ViewBag.MaListe = \_baseDonnees.Zombies.ToList();

return View();

}

## Modifier le controller Zombie l’action create

1. Ajoutez l’action Create GET

public IActionResult Create()

{

ViewBag.ZombieTypes = new SelectList(\_baseDonnees.ZombieTypes.ToList(), "Id", "TypeName", null);

return View();

}

1. Testez la méthode Create Get
   * Ajoutez une point d’arrêt sur la première ligne de code de la méthode, vous allez tester la méthode (même si la *View* n’existe pas)
   * Entez l’URL : <https://localhost:NNNN/zombie/create> (ou NNNN est votre port de localhost)
   * Le débogueur arrête sur la ligne du ViewBag : celui-ci est vide et BaseDonnees.ZombieTypes contient 3 types
   * Une image contenant texte, capture d’écran, logiciel, Page web

     Description générée automatiquement
   * Faites F10 pour passer à la ligne suivante
   * Le ViewBag.ZombieTypes contient maintenant les 3 types.
2. Arrêtez le débogueur
3. Ajoutez l’action Create POST

Si le ModelState est valide : Le zombie est ajouté et nous sommes redirigés vers la page Index, si le ModelState est invalide, le formulaire reste affiché.

[HttpPost]

public IActionResult Create(Zombie zombie)

{

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

if (ModelState.IsValid)

{

\_baseDonnees.Zombies.Add(zombie);

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

return this.RedirectToAction("Index");

}

//Il faut repopuler le zombieType dans le ViewBag

//Aller chercher le ZombieType sélectionné, rappel 2W5 Linq

ZombieType selectedZombieType = \_baseDonnees.ZombieTypes.Where(zt => zt.Id == zombie.ZombieTypeId).SingleOrDefault();

zombie.ZombieType = selectedZombieType;

ViewBag.ZombieTypes = new SelectList(\_baseDonnees.ZombieTypes.ToList(), "Id", "TypeName", selectedZombieType);

return View(zombie);

}

1. Ajoutez la View **Create**

@model ZombieParty.Models.Zombie

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Zombie</h4>

<hr />

<div class="row">

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

<div class="row p-3 ">

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

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

<div class="form-group row">

<div class="col-4">

<**label** **asp-for**="Name" class="control-label"></**label**>

</div>

<div class="col-8">

<**input** **asp-for**="Name" class="form-control" />

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

</div>

<div class="form-group row">

<div class="col-4">

<**label** **asp-for**="Point" class="control-label"></**label**>

</div>

<div class="col-8">

<**input** **asp-for**="Point" class="form-control" />

<**span** **asp-validation-for**="Point" class="text-danger"></**span**>

</div>

</div>

<div class="form-group row">

<div class="col-4">

<**label** **asp-for**="ZombieTypeId" class="control-label"></**label**>

</div>

<div class="col-8">

<**select** class="form-control form-select" **asp-for**="ZombieTypeId" **asp-items**="@ViewBag.ZombieTypes" ></**select**>

<**span** **asp-validation-for**="ZombieTypeId" class="text-danger"></**span**>

</div>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

1. Ajoutez le bouton **Create** dans la **View Zombie-Index**

<div class="container p-3">

<div class="row pt-4">

<div class="col-10">

<h2 class="text-primary">Zombie</h2>

</div>

<div class="col-2 text-right">

<**a** **asp-action**="Create" class="btn btn-success">

Create New Zombie

</**a**>

</div>

</div>

<div class="row">

1. Testez la méthode Create Post
   * Ajoutez une point d’arrêt sur if (ModelState.IsValid)
   * Entrez des informations valides, exemple :

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

Description générée automatiquement

* + Cliquez sur Create
  + Le débogueur arrête sur la ligne sur if (ModelState.IsValid), vérifiez l’état du **ModelState**.

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

Description générée automatiquement

* + Pourquoi est-il a false?

Allez voir la classe Zombie : Parce que **ZombieType** est **null** et il n’est pas explicitement permis de le mettre null

1. Arrêtez le débogueur
2. Dans la classe **Zombie**, ajoutez un ? à la propriété afin de permettre les lignes, nous gérerons ce problème plus tard.
3. Testez à nouveau la méthode Create Post
   * Le débogueur arrête sur la ligne sur if (ModelState.IsValid), vérifiez l’état du ModelState.
4. Arrêtez le débogueur

## 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 et Create et Validation avec messages d’erreurs
   * **Description**: Maintenant possible de voir la liste de zombies et d’en ajouter avec gestion d’erreurs.
5. Cliquez sur **Commit**
6. "Poussez" (**Push)** les modifications sur le repo en ligne (**Remote).** Cliquez sur 
7. C’est la fin de la fonction Zombie

# tester et améliorer Zombie

Nous allons maintenant faire certains changements afin de bien maîtriser le fonctionnement et améliorer Zombie.

## Tester et Modifier le controller Zombie l’action Create

1. Démarrez l’application
2. Essayez de créer un Zombie avec un nom « vide », un message d’erreur devrait apparaître
3. Le chargement des messages d’erreur est-il rapide?
4. Sur la vue Zombie/Create, mettez en commentaires la section de script pour valider ce que cette section permet de faire :

@~~section Scripts {~~

~~@{~~

~~await Html.RenderPartialAsync("\_ValidationScriptsPartial");~~

~~}~~

~~}~~

1. Refaites le même test : Vous devriez voir, lorsque vous soumettez le formulaire qu’il y a un chargement de la page de plus qu’avant lorsque vous faites Create au lieu d’afficher les erreurs immédiatement. (C’est quand même subtile!)
2. Pourquoi?
3. Sur la page de création de Zombie. Mettez le nom Croco Chanel. Sélectionner le type contact. Laissez Point vide.
4. Cliquer sur Create. Évidement, sans Point, ça ne s’ajoute pas en BD.
5. Les données restent-elles affichées (oui/non)?
6. Dans l’action POST Create du Controller, mettez la ligne du ViewBag en commentaire
7. Refaites le même test
8. Les données restent-elles affichées (oui/non)? , la liste des ZombieTypes est-elle encore disponible? Pourquoi?
9. Décommentez la ligne du ViewBag
10. Dans l’action POST Create du controller, on met une valeur dans TempData lors du succès de l’enregistrement mais on ne voit jamais ce message. Pourquoi?
11. Dans \_Layout.cshtml, ajoutez le code pour lire TempData au succès d`un ajout.

<div class="container">

@if (!string.IsNullOrWhiteSpace(TempData["Success"] as string))

{

<div class="alert alert-success">@TempData["Success"]</div>

}

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

1. Ajoutez un Point d’arrêt à la ligne if (ModelState.IsValid)
2. Essayez de créer un Zombie valide
3. Avancez dans l’exécution du code avec F10 afin de bien comprendre ce qui se passe.

* Est-ce que ModelState est valide?
* Où sont les informations saisies?
* Le message de confirmation s’affiche-t-il?

1. Essayez de créer un Zombie avec des informations manquantes
2. Avancez dans l’exécution du code avec F10 afin de bien comprendre ce qui se passe.

* Est-ce que ModelState est valide?
* Où sont les informations saisies?

1. Enlevez les commentaires sur le Script de validation pour réactiver la validation côté client (défaire l’étape 4)

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

## 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 Create ajout de message de confirmation
   * **Description**: Ajout d’un message de confirmation lors de la création d’un Zombie.
5. Cliquez sur **Commit**

# améliorer ZombieType

## Tester et Modifier le controller ZombieType l’action Create

1. Ajoutez un message de confirmation pour la création d’un ZombieType.

if (ModelState.IsValid)

{

\_baseDonnees.ZombieTypes.Add(zombieType);

TempData["Success"] = $"{zombieType.TypeName} zombie type added";

return this.RedirectToAction("Index");

}

1. Testez la création d’un Zombie Type valide.

## 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 ZombieType Create ajout de message de confirmation
   * **Description**: Ajout d’un message de confirmation lors de la création d’un type de zombie.
5. Cliquez sur **Commit**
6. "Poussez" (**Push)** les modifications sur le repo en ligne (**Remote).** Cliquez sur 
7. C’est la fin de la fonction Zombie
8. Merge la branche **FCT\_Zombie** dans la branche **Main.**
9. "Poussez" (**Push) MAIN** les modifications sur le repo en ligne (**Remote).** Cliquez sur 

# Utiliser les ViewModels et les annotations

Nous allons maintenant créer une View présentant les informations de plusieurs modèles à la fois et un calcul.

## Création d’une branche pour la fonctionnalité

1. Créez une branche appelée **FCT\_ZombieType\_VM\_Annotations**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

## Améliorer la View Index

1. Ouvrez le **ZombieTypeController**
2. Modifiez le code de l’action **Index** afin de ne pas utiliser le ViewBag

public IActionResult Index()

{

this.ViewBag.MaListeList<ZombieType> zombieTypesList = \_baseDonnees.ZombieTypes.ToList();

return View(zombieTypesList);

}

1. Modifiez la View **Index** afin d’utiliser le model envoyé en paramètre, soit **ZombieTypeList**

@model IEnumerable<ZombieType>

﻿<div class="container p-3">

<div class="row pt-4">

<div class="col-10">

<h2 class="text-primary">Zombie Type</h2>

</div>

……

<tbody>

@foreach (var item in Model)

{

<tr>

<td width="80%">@item.TypeName</td>

<td class="text-center">

<div class="w-75 btn-group" role="group">

1. Testez la View **Index,** vous devriez avoir ceci :

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

Description générée automatiquement

## View affichant le type de zombie et la liste des zombies assocés

1. Ouvrez le ZombieTypeController
2. Ajoutez une action GET pour **Details**

public IActionResult Details()

{

return View();

}

1. Générez la View **Details,** remplacez le contenu par le départ suivant :
   * Note : Pas besoin de rien changer à cette vue pour le moment, vous allez utiliser un ViewModel un peu plus tard pour afficher les bonnes valeurs.

@{

ViewData["Title"] = "Details";

}

<h1>Type: NOM DU ZOMBIETYPE</h1>

<div class="row">

@\* POUR CHAQUE ZOMBIE:\*@

<div href="#" class="card border m-3 text-center bg-warning" style="width: 20rem; background-image: url('/images/imagesLayout/Fond.png'); ">

<div class="card-body text-warning">

<h3 class="card-title"> NOM DU ZOMBIE </h3>

<h5> POINT </h5>

</div>

</div>

</div>

@\* SI AUCUN ZOMBIE ASSOCIÉ\*@

<div class="alert alert-danger">No associated zombie for this type</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

1. Ajoutez le code nécessaire dans la view **Index** afin que la View **Détails** s’affiche en cliquant sur un ZombieType Name (même si la vue est très statique pour l’instant). Puisque la View doit afficher un ZombieType en particulier, le Id du ZombieType doit être envoyé à l’action du Controller

@foreach (var item in Model)

{

<tr>

<td width="80%">

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

@item.TypeName

</**a**>

</td>

1. Testez, vous devriez avoir ceci :

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

Description générée automatiquement

1. Puisque l’on a besoin d’informations provenant de plus d’un modèle, il faut un **ViewModel**.
2. Ajoutez un dossier **ViewModels** au projet
3. Ajoutez une classe appelée **ZombieTypeVM** pour concevoir un ViewModel qui présente un type de zombie et la liste des zombies de ce type.

public class ZombieTypeVM

{

public ZombieType ZombieType { get; set; }

public List<Zombie> ZombiesList { get; set; } = new List<Zombie>();

}

La View présente un ZombieType en particulier : ça prend donc le Id du zombieType afin de trouver le bon. Puis la liste des zombies qui ont ce ZombieType Id.

1. Ajoutez dans l’action **Details** du controller ZombieType le code permettant d’envoyer le ViewModel avec les informations requises à la View. Il faut ressortir vos notions sur les ViewModels et Linq de 2W5!

public IActionResult Details(int id)

{

ZombieTypeVM zombieTypeVM = new()

{

ZombieType = new(),

ZombiesList = \_baseDonnees.Zombies.Where(z => z.ZombieTypeId == id).ToList()

};

zombieTypeVM.ZombieType = \_baseDonnees.ZombieTypes.FirstOrDefault(zt => zt.Id == id);

return View(zombieTypeVM);

}

1. Ajoutez un Point d’arrêt au début de la méthode et testez-là
2. Modifiez la vue **Details** afin qu’elle utilise les données du ViewModel **ZombieTypeVM**
3. Au début de la View, ajoutez le modèle

@model ZombieParty.ViewModels.ZombieTypeVM

@{

ViewData["Title"] = "Details";

}

1. Comment éviter d’avoir à écrire le nom du projet et le dossier, **ZombieParty.ViewModels**, dans chaque View?
2. Modifiez le \_**ViewImports** afin d’ajouter le **using**

@using ZombieParty

@using ZombieParty.Models

@using ZombieParty.ViewModels

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

1. Au début de la View, modifiez l’appel du modèle en enlevant la partie non nécessaire

@model ZombieParty.ViewModels.ZombieTypeVM

@{

ViewData["Title"] = "Details";

}

1. Complétez la View afin de présenter les informations souhaitées : Le nom du type et la liste des zombies de ce type.

<h1>Type: @Model.ZombieType.TypeName</h1>

<div class="row">

@\* POUR CHAQUE ZOMBIE:\*@

@foreach (var item in Model.ZombiesList)

{

<div href="#" class="card border m-3 text-center bg-warning" style="width: 20rem; background-image: url('/images/imagesLayout/Fond.png'); ">

<div class="card-body text-warning">

<h3 class="card-title"> @item.Name </h3>

<h5> @item.Point </h5>

</div>

</div>

}

</div>

@\* SI AUCUN ZOMBIE ASSOCIÉ\*@

@if(Model.ZombiesList.Count == 0){

<div class="alert alert-danger">No associated zombie for this type</div>

}

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

1. Ajoutez un type de zombie (qui n’aura pas de zombies associés)
2. Testez la View

## 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 Création View ZombieType Details
   * **Description**: Ajout de la view Details affichant le ZombieType et la liste des zombies associés.
5. Cliquez sur **Commit**

## Ajouter le nombre de zombies dans la View Details

Que devez-vous modifier afin de présenter le nombre de zombies et la moyenne des points de la liste de zombies pour chaque ZombieType?

1. Modifiez le ViewModel **ZombieTypeVM**

public class ZombieTypeVM

{

public ZombieType ZombieType { get; set; }

public List<Zombie> ZombiesList { get; set; } = new List<Zombie>();

public int ZombiesCount { get; set; }

public double PointsAverage { get; set; }

}

1. Modifiez l’action Details du controller

{

ZombieType = new(),

ZombiesList = \_baseDonnees.Zombies.Where(z => z.ZombieTypeId == id).ToList(),

ZombiesCount = \_baseDonnees.Zombies.Count(),

PointsAverage = \_baseDonnees.Zombies.Average(p => p.Point)

};

1. Modifiez la View Details

@model ZombieTypeVM

@{

ViewData["Title"] = "Details";

}

<h1>Type: @Model.ZombieType.TypeName</h1>

<h3>Nombre de zombies: @Model.ZombiesCount</h3>

<h3>Moyenne des points: @Model.PointsAverage</h3>

<div class="row">

@\* POUR CHAQUE ZOMBIE:\*@

1. ATTENTION : Mais ça n’affiche pas le bon nombre de zombies, ni la bonne moyenne d’ailleurs… Trouvez une façon de corriger le problème!

## 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 View Details ZombieType Nombre Zombie et Moyenne Points
   * **Description**: Ajout du nombre de zombies et la moyenne des points pour le Type de zombie dans la View Details..
5. Cliquez sur **Commit**
6. "Poussez" (**Push)** les modifications sur le repo en ligne (**Remote).** Cliquez sur 
7. C’est la fin de la fonction Zombie
8. Merge la branche **FCT\_ZombieType\_VM\_Annotations** dans la branche **Main.**

# Ajouter une armoirerie

## Création d’une branche pour la fonctionnalité

1. Créez une branche appelée **FCT\_Armory**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

## Ajouter le menu déroulant

1. Ouvrez le \_Layout
2. Ajoutez le menu déroulant **Armory** et l’action **Manage Weapons**

<li class="nav-item dropdown">

<a class="nav-link text-dark dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">

Armory

</a>

<ul class="dropdown-menu" aria-labelledby="navbarDropdown">

<li><**a** class="dropdown-item" **asp-controller**="Weapon" **asp-action**="Index">Manage weapons</**a**></li>

</ul>

</li>

## Ajouter la classe Weapon

1. Ajoutez la classe **Weapon** ci-dessous.

public class Weapon

{

public string Name { get; set; }

public string? Description { get; set; }

public decimal Force { get; set; }

public decimal Price { get; set; }

public DateTime CreatedDate { get; set; } = DateTime.Now;

public string? Image { get; set; }

public int Qty { get; set; }

public int QtyBought { get; set; }

}

Voici les contraintes pour propriétés de la classe Weapon:

* Name (string, max 250, min 2, required)
* Description (string, max 2500, min 0, NOT required, multiline)
* Force (decimal, min 0, max 500)
* Price : decimal (min 0, max 100000, Currency) :
  + Message d’erreur doit être : The <PRICE> has to be between <MINIMUM> and <MAXIMUM> (Changez les **<lettres majuscules>** par les paramètres du message d’erreur)
* CreatedDate : DateTime
* Image : ImageUrl
* Qty :int
* QtyBought :int
  + Un Display name qui contient l’espace

1. Ajoutez les attributs appropriés pour valider les champs : StringLength, Maxlength, Range
2. Mettez les Display Name sur (name, description et Image) pour avoir le préfixe Weapon’s
   1. Ex. Weapon’s Name
3. Mettez-les aussi sur les propriétés qui ont deux mots pour avoir l’espace. (ex. Qty Bought)
4. Ajoutez une validation personnalisée à Weapon qui vérifie que la description est de plus de 3 mots (séparé par des espaces) s’il y en a une :

public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)

{

var item = validationContext.ObjectInstance as Weapon;

if(item == null) yield break;

if(string.IsNullOrWhiteSpace(item.Description)) yield break;

if(item.Description.Split(" ").Length <= 3)

yield return new ValidationResult("Description needs to have more than 3 words please.", new[] { "Description" });

}

Notez que yield veut dire qu’il y a une énumération. Au lieu d’envoyer la liste au complet, on le renvoie morceau par morceau.   
yield break arrête l’énumération

1. Une fois que vous avez ajoutez la méthode Validate à la classe Weapon, il faut également que cette classe implémente l’interface IValidatableObject pour que le framework sache que Weapon doit être validé.
2. Ajoutez le DataType DateTime, ImageUrl, Multiline et Currency pour les champs correspondants.

## Ajouter le controller Weapon

1. Ajouter le contrôleur pour les Weapons avec les actions Index et Create, ne retournant que la View pour l’instant.

public class WeaponController : Controller

{

public IActionResult Index()

{

return View();

}

public IActionResult Create() {

return View();

}

[HttpPost]

public IActionResult Create(Weapon weapon)

{

return View();

}

}

## Ajouter les Views controller Weapon

1. Ajouter les vues générées de liste et de création pour Weapon.
   1. Sur la méthode du contrôleur, faites : clic droit > Ajoutez une vue > Vue Razor >

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

Description générée automatiquement

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

Description générée automatiquement

## Ajouter des Seeds de données pour Weapons

1. Ajoutez 2 Weapon de base dans l’entrepôt de données.

public class BaseDonnees

{

public List<ZombieType> ZombieTypes { get; set; }

public List<Zombie> Zombies { get; set; }

public List<Weapon> Weapons { get; set; }

public BaseDonnees()

{

//Initialiser des listes vides

Zombies = new List<Zombie>();

ZombieTypes = new List<ZombieType>();

Weapons = new List<Weapon>();

…

//Remplir la liste des Weapons

Weapons.Add(new Weapon() { Name = "Leopold", Description = "This is to cut your friendly zombies", Force = 200, Image = "https://i.ibb.co/y5nCnXT/dagger-gfce31e967-1920.png", Price = 421, Qty = 21, QtyBought = 3, CreatedDate = new DateTime(1999, 01, 1) });

Weapons.Add(new Weapon() { Name = "Master Shield", Description = "An awesome Stainless Steel Shield", Force = 10, Image = "https://publicdomainvectors.org/photos/Shield-ClassicMedieval1.png", Price = 500, Qty = 30, QtyBought = 1, CreatedDate = new DateTime(1999, 01, 1) });

}

## Compléter le WeaponController et les views

1. Le tout n’est pas encore fonctionnel. Complétez les actions du contrôleur et les Views en suivant les informations ci-dessous.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

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

Description générée automatiquement

**NOTE** : Le blanc a été fait en ajoutant la classe bg-white et shadow

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

Description générée automatiquement

Une image contenant texte, Police, ligne, Tracé

Description générée automatiquement

1. Assurez-vous de pouvoir afficher les **Weapon** et en créer de nouveaux.

## 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 Weapon Index et Create
   * **Description**: Maintenant possible de voir la liste de Weapons et d’en ajouter.
5. Cliquez sur **Commit**
6. "Poussez" (**Push)** les modifications sur le repo en ligne (**Remote).** Cliquez sur 
7. Merge la branche **FCT\_Armory** dans la branche **Main.**

**Fin du laboratoire!**