Laboratoire: introduction à Entity Framework

# Objectif(s)

* Models: Entity Framework
  + Créer la connexion DB
  + Créer le DbContext
  + Ajouter les Data Annotations de base
  + Effectuer des Migration et *update database*
* Validation
  + Ajouter des annotations de validation
  + Utiliser les TagHelpers liés à la validation et aux données

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

# Configuer le projet pour entity framework

## Installer les packages requis

1. Ouvrez le projet Visual Studio qui se trouve dans votre dossier cloné
2. Dans Visual Studio, assurez-vous d'être dans la bonne branche (en bas à droite)
3. Allez dans Outils-Gestionnaire de packages NuGet-Gérer les packages de la solution
4. Installer les packages suivants, acceptez les licences et confirmez l'installation (messages):

* Microsoft.EntityFrameworkCore 6.0.16
* Microsoft.EntityFrameworkCore.SqlServer 6.0.16
* Microsoft.EntityFrameworkCore.tools 6.0.16
* Microsoft.EntityFrameworkCore.Design 6.0.16 (va s’installer avec le précédent, vérifiez)

# Configuer la bd du projet

## Ajout de la Connexion à la DB

1. Ouvrez **SQL Management Studio**
2. Connectez-vous et copiez le nom du serveur
3. Ouvrez le fichier appSettings.json
4. Ajouter la connexion soit le texte en vert ATTENTION n'oubliez pas la virgule!

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*" ,

"ConnectionStrings": {

"DefaultConnection": "Server=localhost\SQLEXPRESS;Database=ZombieParty;Trusted\_Connection=True;MultipleActiveResultSets=True"

}

}

## 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 Configuration EntityFramwork Packages et connexion
   * **Description**: Installation des packages minimum pour EntityFramework et SQL Server et de la connexion à la BD.
5. Cliquez sur **Commit**

## Création de l'application DbContext

1. Créez un dossier **Data** (facultatif) dans **Models**
2. Créez la classe publique **ZombiePartyDbContext** qui implémente **DbContext**

Ajoutez la référence (using) à **Microsoft.EntityFrameworkCore** (proposé par Visual Studio dans la bulle)

1. Créer un constructeur (ctor)

Passer le BdContext options en paramètres dont la connectionString

public ZombiePartyDbContext(DbContextOptions<ZombiePartyDbContext>options):base(options)

{

}

# configurer le service SqlServeur

## Injection de dépendance (ajout de la Dependency Injection)

1. Ouvrez le fichier **Program**
2. Dans la méthode ConfigureServices, injectez la dépendance au service SQL serveur avec le nom de la connexion (figurant dans appsettings.json)

Le code en vert

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<ZombiePartyDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

var app = builder.Build();

1. Ajoutez les références (Using) requis proposés par Visual Studio

## 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 Configuration EntityFramwork DbContext DI
   * **Description :** Création du ApplicationDbContext et ajout de DI du service SQL Server et de la connexion à la BD.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Modèle de données

## Création du diagramme de classes de départ

1. Dans Visio, complété le diagramme des classes en fonction de la portion de l’étude de cas ci-dessous, le fichier est dans le repo, pour accélérer le processus, les classes et certaines propriétés et annotations sont présentes mais parfois incomplètes.
2. Identifiez les propriétés (type, longueur, obligatoire, etc) manquantes
3. Identifiez les relations entre les classes

Les prochaines étapes vont permettrent de modifier le modèle de données existant et de générer la base de données en fonction du modèle de classes UML de l’étude de cas ci-dessous.

**ZombieParty :**

Les zombies appartiennent à un type de zombie (*ZombieType)*, ils ont un nom (*Name*) entre 5 et 20 caractères obligatoire, un pointage (*Point)* de 1 à 20 et une courte description (*ShortDesc*) jusqu’à 255 caractères. Le *ZombieType* doit avoir un nom obligatoire entre 5 et 10 caractères et un pointage (*Point*) entre 2 et 5.

L’armurerie, comprendant seulement la classe *Weapon* pour l’instant, n’est lié a rien pour le moment. Les critères pour les annotations ont été donnés dans un laboratoire précédent et quelques propriétés ont été ajoutées.

Les chasseurs complètent un journal de chasse (*HuntingLog*) qui relate leurs mésaventures avec les zombies. Chaque entrée du *HuntingLog* comporte un titre de 5 à 25 caractères et une description de la mésaventure. Celle-ci peut impliquer un ou plusieurs zombies et un zobie peut être impliqué dans plusieurs aventure *HuntingLog*.

**Puisque l’on change les modèles de données, le chargement de données dans BaseDonnees a été mis en commentaire sinon il y aura des erreurs et le projet ne compilera pas.**

public BaseDonnees()

{

idInstance = Guid.NewGuid();

//Initialiser des listes vides

Zombies = new List<Zombie>();

ZombieTypes = new List<ZombieType>();

Weapons = new List<Weapon>();

//Remplir la liste des ZombieTypes

//{

// ZombieTypes.Add(new ZombieType() { TypeName = "Virus", Id = 1 });

// ZombieTypes.Add(new ZombieType() { TypeName = "Contact", Id = 2 });

// ZombieTypes.Add(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 de l'enfer", Point = 7, ZombieTypeId = 1 });

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

//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", Lifespan = 50, Price = 421, Qty = 21, QtyBought = 3, Resistance = 10, UpdatedDate = new DateTime(2012, 1, 1), 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", Lifespan = 10, Price = 500, Qty = 30, QtyBought = 1, Resistance = 800, UpdatedDate = new DateTime(2012, 1, 1), CreatedDate = new DateTime(1999, 01, 1) });

}

## Modification des classes FCT Zombie et ZombieType

1. Ouvrez la classe *ZombieType*
2. Notez que la propriété Id (int) permet de créer une clé primaire automatique (no auto-incrémenté)
3. Ajoutez l'annotation pour en faire une clé primaire explicite (Key)

[Key]

public int Id { get; set; }

* 1. Ajoutez les références requises (intellisence, *Using*)

1. Ouvrez la classe *Zombie*
2. Ajoutez l’annotation *ForeighKey* pour la clé étrangère ZombieTypeId

// FACULTATIF on peut formellement identifier le champ lien

// sinon le champ de foreignKey sera auto généré dans la BD

[Display(Name = "Zombie Type")]

[ForeignKey("ZombieType")]

public int ZombieTypeId { get; set; }

//Propriété de navigationpour la relation 1 à plusieurs avec ZombieType

public ZombieType ZombieType { get; set; }

## Ajout des classes à l'ApplicationDbContext

1. Ouvrez **Data/ZombiePartyDbContext**
2. Ajoutez après le constructeur, les **DbSets**, 1 pour les classes du modèle de données **Zombie** et **ZombieType** soient utilisées pour générer des tables, selon la syntaxe ci-dessous

public DbSet<NOM\_CLASSE> NOM\_ENTITÉ\_DANS\_BD { get; set; }

# Migration et update de la DB FCT Zombie\_ZombieType

## création initiale de la BD

1. Ouvrez la console *Outils-Gestionnaire de packages Nuget-Console du gestionnaire de package*
2. Créez la 1ère migration appelée **CreationBD\_Zombie\_ZombieType** qui va créer la BD avec 2 tables correspondant aux 2 classes, tapez:

add-Migration CreationBD\_Zombie\_ZombieType

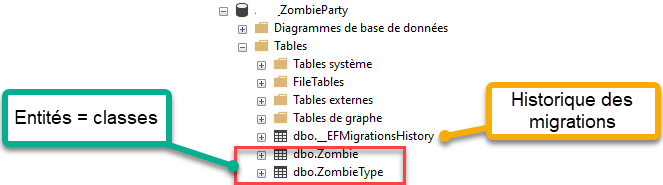
1. Lisez (oui oui pour vrai!) le message d’erreur affiché dans la console en rouge : Vous ne pouvez pas créer des tables sans clés primaires (à moins de le configurer explicitement)

Comment corriger cette erreur?

1. Ajoutez une clé primaire Id pour la classe du modèle **Zombie**
2. Refaite la commande de migration (🡹rappelle l’instruction précédente)
3. Consulter le script généré: notez les tables, attributs, les clés primaires.
4. Allez voir dans SQL Management Studio : la BD n’est pas encore là : le script n’a pas encore été exécuté
5. Effectuez la commande suivante dans la console afin d’exécutez le script qui va créer la BD et les tables dans le SQL Server

update-database

1. Ouvrez Management Studio afin de vérifier si votre DB a bien été créée.



## Modification des classes du modèle de données

**Classe Zombie**

1. Ajoutez les propriétés et les annotations manquantes ou modifiez les afin qu’elles correspondent au modèle de classe, avec les messages d’erreur génériques   
   Exemple : [Range(2, 5, ErrorMessage = "{0} required a value between {1} and {2}.")]
2. Mettez à jour la BD :
   * Add-migration AjoutZombiePK\_ShortDesc\_annotations
   * Vérifiez le script
   * Update-database
3. Vérifiez le résultat
4. Notez que tous les champs sont NULL interdit

**Classe ZombieType**

1. Ajoutez les propriétés et les annotations manquantes ou modifiez les afin qu’elles correspondent au modèle de classe
2. Mettez à jour la BD :
   * Add-migration ModifierZombieType\_Annotations
   * Vérifiez le script
   * Update-database
3. Vérifiez le résultat

## Corriger les contrôleurs et les Vues

1. Compilez la solution

Dans les étapes suivantes, vous allez corriger les contrôleurs et les Views Zombie et ZombieType :

* + Utiliser les DbSet du DbContext liés à la vrai base de données au lieu des listes du singleton BaseDonnees
  + utiliser les bons champs (clés étrangères et clés primaires)
  + Enlever les appels aux ViewBags
  + Visualiser tous les champs requis dans les Views

**ZombieType :**

1. Ouvrez le contrôleur
2. Remplacez l’injection de **BaseDonnees** (le singleton en listes) par le **ZombiePartyDbContext**

private ZombiePartyDbContext \_baseDonnees { get; set; }

public ZombieTypeController(ZombiePartyDbContext baseDonnees)

{

\_baseDonnees = baseDonnees;

}

1. Modifiez le code des actions du controlleur afin d’utiliser les DbSet du DbContext.

**NOTE** : Si vous avez mis le nom des tables au pluriel, peu de modification, si vous les avez mis au singulier, plus de modifications. Modifiez le nom des tables dans le ZombiePartyDbContext afin de vous assurez qu’ils sont au pluriel.

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

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

1. Mettez à jour la DB (add-migration update-database).
2. Vérifiez le résultat dans la DB.

**ZombieType Views Create et Index**

1. Ouvrez la View Create
2. Ajoutez le input pour Point

<div class="form-group row">

<div class="col-2">

<**label** **asp-for**="Point" class="col-form-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>

1. Ajoutez un Point d’arrêt sur Post Create sur la 1ère ligne de code de l’Action
2. Testez Index : la liste est vide
3. Testez la création avec un ZombieType valide, par exemple :
   * Name : Rampant
   * Point**:** 5
4. Vérifiez l’état du modèle : IsValid est a false; pourquoi?

Parce que le modèle de données demande une Liste de zombies qui ne peut pas être null.

1. Arrêtez l’application
2. Ajoutez un ? à la propriété dans le modèle de classe.
3. Refaites le test avec les mêmes données
4. Vérifiez l’état du modèle : IsValid est a true, yé!
5. Faites F10, vérifiez chaque étape.
   * Vous avez reçu un message de confirmation de la création mais l’index n’affiche pas le nouveau type, pourquoi?
6. Allez dans la BD : le nouveau type n’est pas enregistré, la table est vide.

Une étape très importante doit être ajoutée : Enregistrer les modifications! Pour la création, modification et suppression.

1. Dans l’action Create, après l’ajout ajoutez l’enregistrement des modifications

if (ModelState.IsValid)

{

\_baseDonnees.ZombieTypes.Add(zombieType);

\_baseDonnees.SaveChanges();

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

1. Vérifiez l’état du modèle : IsValid est a true, yé!
2. C’est dans l’Index et dans la BD, yé!
3. Enlevez le Point d’arrêt

**Zombie :**

1. Ouvrez le contrôleur
2. Remplacez l’injection de **BaseDonnees** (le singleton en listes) par le **ZombiePartyDbContext**
3. Remplacez l’utilisation du ViewBag par une liste.

public IActionResult Index()

{

List<Zombie> ZombiesList = \_baseDonnees.Zombies.ToList();

return View(ZombiesList);

}

**Zombie View Index**

1. Ajoutez le model en haut de la vue
2. Modifiez le foreach pour utiliser le model retourné à la vue

En cas de doute, référez-vous au ZombieType ou au repo de Démo.

**Zombie Create**

La vue Create pour Zombie présente l’information de plus d’une classe de modèle : Zombie et liste des ZombieType. Afin de respecter les bonnes pratiques, nous allons créer un ViewModel.

1. Créez un ViewModel **ZombieVM**

public class ZombieVM

{

// Pour Upsert 1 zombie à la fois

public Zombie Zombie { get; set; }

// Pour créer les deux listes déroulantes

public IEnumerable<SelectListItem> ZombieTypeSelectList { get; set; }

}

1. Modifiez le code du controleur des actions Create afin d’utiliser le ViewModel

public IActionResult Create()

{

ZombieVM zombieVM = new ZombieVM();

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

{

Text = t.TypeName,

Value = t.Id.ToString()

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

return View(zombieVM);

}

[HttpPost]

public IActionResult Create(ZombieVM zombieVM)

{

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

if (ModelState.IsValid)

{

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

\_baseDonnees.SaveChanges();

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

return this.RedirectToAction("Index");

}

zombieVM.ZombieTypeSelectList = (IEnumerable<SelectListItem>) \_baseDonnees.ZombieTypes.ToList();

return View(zombieVM); //retourne l'objet pour avoir les données

}

**Zombie View Create**

1. Modifiez le model en haut de la vue et les *inputs* pour utiliser le ViewModel
2. Ajoutez un *input* pour Point et un *textArea* pour ShortDesc (height 300 px),

<div class="form-group row">

<div class="col-4">

<**label** **asp-for**="@Model.Zombie.ShortDesc" class="col-form-label"></**label**>

</div>

<div class="col-8">

<**textarea** **asp-for**="@Model.Zombie.ShortDesc" class="form-control" style="height:300px;"></**textarea**>

<**span** **asp-validation-for**="@Model.Zombie.ShortDesc" class="text-danger"></**span**>

</div>

</div>

1. Remplacez l’utilisation du ViewBag par la SelectList du ViewModel

<div class="form-group row">

<div class="col-4">

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

</div>

<div class="col-8">

<**select** **asp-for**="@Model.Zombie.ZombieTypeId" **asp-items**="@Model.ZombieTypeSelectList"

class="form-select">

<**option** disabled selected>--Select Zombie Type--</**option**>

</**select**>

<**span** **asp-validation-for**="@Model.Zombie.ZombieTypeId" class="text-danger"></**span**>

</div>

## 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**: REFACTOR Zombie et ZombieType
   * **Description**: Création initiale BD, remplacement des listes par des tables en BD pour Zombie et ZombieType
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Migration et update de la DB FCT HuntingLog

## Modification des classes du modèle de données

**Classe HuntingLog**

1. Dans Models, créez la classe HuntingLog avec les propriétés du modèle de classe:
   * N’oubliez pas la clé primaire : Id (int)
   * Les propriétés de navigation des deux côtés!
2. Créez une migration AjoutHuntingLog
3. Vérifiez le script généré: il est vide! Oups….

Qu'avez-vous oublié? Et oui, le DbSet dans ZombiePartyDbContext!

1. Dans la console, supprimez la migration

Remove-Migration

1. Dans **ZombiePartyDbContext**, ajoutez un dbSet pour HuntingLog
2. Créez une migration AjoutHuntingLog
3. Vérifiez le script généré: Yé! HuntingLog est là!
4. Mettez à jour la BD
5. Vérifiez que la BD correspond à vos modifications

Zombie existait déjà, HuntingLog a été ajoutée plus la table de résolution du plusieurs-plusieurs entre HuntingLog et Zombie

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

Description générée automatiquement

## 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 HuntingLog
   * **Description**: Création de la table HuntingLog, liaison avec Zombie plusieurs à plusieurs
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

**NOTE** : pas de Controller et Views pour l’instant pour HuntingLog

# Migration et update de la DB FCT Amory

## Modification des classes du modèle de données

**Classe Weapon**

1. Dans Models, la classe existe déjà
2. Créez une migration AjoutWeapon
3. Vérifiez le script généré: il est vide! Oups….

Qu'avez-vous oublié? Et oui, le DbSet dans ZombiePartyDbContext!

1. Dans la console, supprimez la migration

Remove-Migration

1. Dans **ZombiePartyDbContext**, ajoutez un dbSet pour Weapon
2. Créez une migration AjoutWeapon! Oups…pas de clé primaire!
3. Dans la console, supprimez la migration…encore!
4. Ajoutez la clé primaire à la classe weapon
5. Créez une migration AjoutWeapon
6. Vérifiez le script généré: Yé! Weapon est là!
7. Mettez à jour la BD
8. Vérifiez que la table a été ajoutée

**NOTE** : pas de Views de plus pour l’instant pour Weapon

## 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 Weapon
   * **Description**: Création de la table Weapon
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 