Laboratoire: GitHub et révision MVC

# 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 séance

## Création du Repository et de la branche de la séance

1. Dans Gitkraken, clonez votre repository GitHub **A21\_3W6\_Labos\_NOM\_PRENOM**
2. Cliquez sur la branche de la séance précédente: **S01\_Github\_Gitkraken\_MVC**
3. Créez la branche de la séance: **S02\_EntityFramework\_Intro**

NOTE: Les branches S01 et S02 sont identiques pour le moment. La branche S02 n'existe que sur votre git LOCAL.

# Configuer le projet pour entity framework

## Installer les packages requis via powershell

1. Dans Gitkraken, assurez-vous d'être dans la branche de la séance: **S02\_EntityFramework\_Intro**
2. Ouvrez votre projet qui se trouve dans votre dossier cloné
3. Dans Visual Studio, assurez-vous d'être dans la branche de la séance: **S02\_EntityFramework\_Intro**
4. Allez dans Outils-Gestionnaire de packages NuGet-Gérer les packages de la solution
5. Installer les packages suivants, acceptez les licences et confirmez l'installation (messages):

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.tools
* Microsoft.EntityFrameworkCore.Design

# Configuer la bd du projet

## Ajout de la Connexion à la DB

1. Ouvrez Management Studio
2. Connectez-vous et copiez le nom du serveur

ATTENTION: Dans les laboratoires, le nom du serveur par défaut n'est pas le bon!

1. Ouvrez le fichier appSettings.json
2. 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=(localdb)\\MSSQLLocalDB;Database=ZombieParty;Trusted\_Connection=True;MultipleActiveResultSets=True"

}

}

## 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**

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 **Startup**
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

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ZombiePartyDbContext>(options =>

options.UseSqlServer(

Configuration.GetConnectionString("DefaultConnection")));

services.AddControllersWithViews().AddRazorRuntimeCompilation();

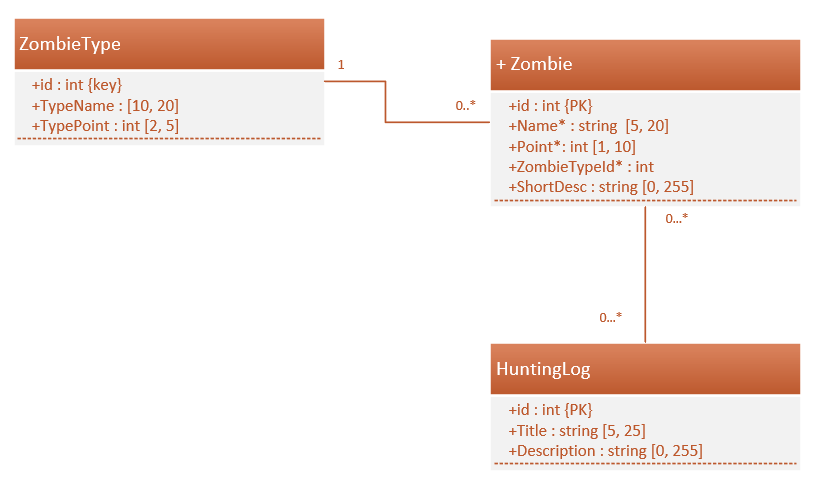
}

## 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**: CONF Configuration EntityFramwork
   * **Description**: Installation des packages minimum pour EntityFramework et SQL Server, 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

**Pour l'instant vous n'avez que les classes Zombie et ZombieType**



## Modification des classes

1. Ouvrez la classe Models/ZombieType
2. Notez que la propriété Id (int) permet de créer une clé primaire automatique (no auto-incrémenté)
3. Ouvrez la classe Models/Zombie
4. Ajoutez la propriété Id (int)
5. Ajoutez l'annotation pour en faire une clé primaire explicite (Key)
   1. Ajoutez les références requises (intellisence)

NOTE: aucune liaison n'a été configurée pour l'instant

## Ajout des classes à l'ApplicationDbContext

1. Ouvrez Data/**ZombiePartyDbContext**
2. Ajoutez après le constructeur, les **DbSets**, 1 pour chaque classe du Model, selon la syntaxe ci-dessous

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

# Migration et update de la DB

## création initiale de la BD

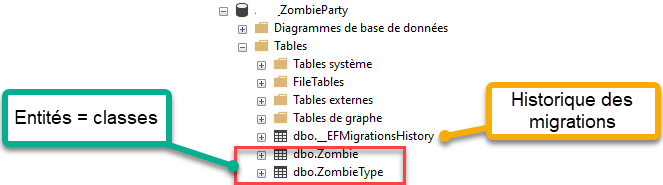
1. Ouvrez la console PowerShell
2. Créez la 1ère migration appelée **CreationInitiale** qui va créer la BD avec 2 tables correspondant aux 2 classes, tapez:

add-Migration CreationInitiale

1. Consulter le script généré: notez les tables, attributs, les clés primaires.
2. 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 une propriété ShortDesc (string)
2. Créez une migration AddZombieTypeShortDesc
3. Sur la propriete Point, ajoutez une validation de range de 1 à 10, message: The point must be 1 to 10.
4. Vérifiez le script généré
5. Mettez à jour la BD
6. Vérifiez que le champ a été ajouté

**Classe ZombieType**

1. Ajoutez une propriété Point (int) avec une validation de plage de 2 à 5 (pas de message pour le moment)
2. Créez une migration AddZombiePoint
3. Vérifiez le script généré
4. Mettez à jour la BD
5. Vérifiez que le champ a été ajouté

## Ajout des liaisons entre les classes

**Classe Zombie**

[Display(Name="Zombie Type")]

// FACULTATIF on peut formellement identifier le champ lien

public int ZombieTypeId { get; set; }

[ForeignKey("ZombieTypeId")]

//OBLIGATOIRE Pour la relation 1 à plusieurs avec ZombieType

public virtual ZombieType ZombieType { get; set; }

1. Créez une migration Add\_FK\_ZombieType\_Zombie
2. Vérifiez le script généré
3. Mettez à jour la BD
4. Vérifiez que la liaison a été ajoutée

## 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
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

## Modification des classes du modèle de données

**Classe HuntingLog**

1. Dans Models, créez la classe HuntingLog avec les propriétés suivantes:
   * Id (int)
   * Title (string)
   * Description (string)
2. Créez une migration AddHuntingLog
3. Vérifiez le script généré: il est vide! Oups….

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

1. Dans la console, supprimez la migration

Remove-Migration

1. Dans **ZombiePartyDbContext**, ajoutez un **dbSet** pour **HuntingLog**
2. Créez une migration AddHuntingLog
3. Vérifiez le script généré: Yé! HuntingLog est là!
4. Mettez à jour la BD
5. Vérifiez que la table a été ajoutée

## 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
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 