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: **S03\_EntityFramework\_Suite**
3. Créez la branche de la séance: **S04\_EF\_Fluent\_API**

# Configuer la bd du projet

## Modification 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. Modifier le nom du serveur SEULEMENT!

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*" ,

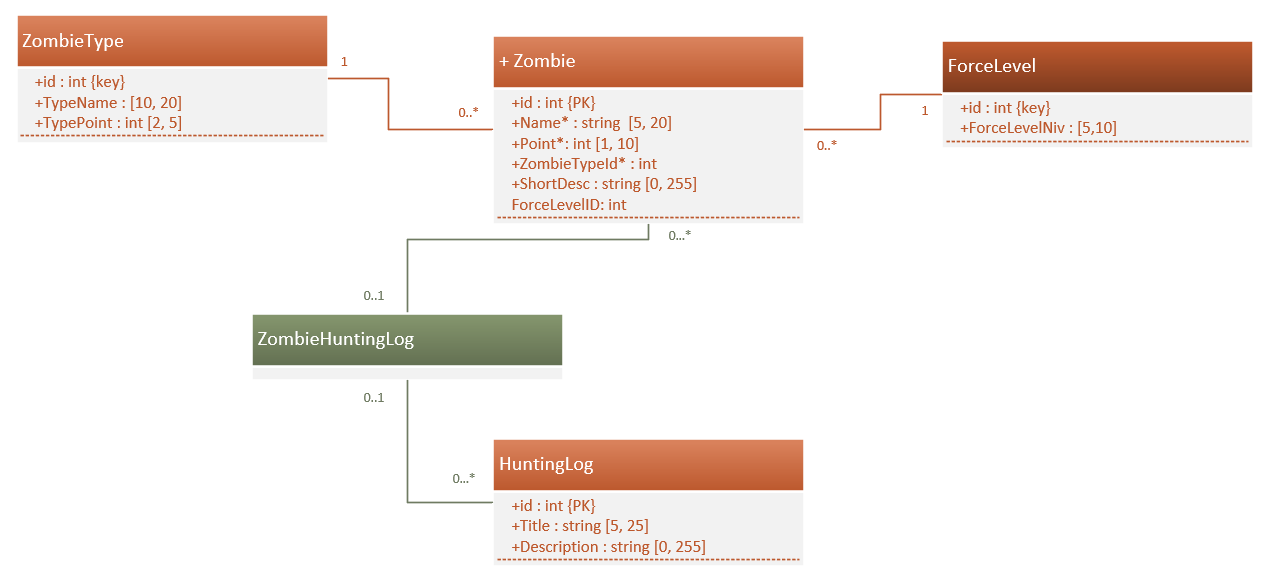
"ConnectionStrings": {

"**DefaultConnection":** "**Server**=(localdb)\\MSSQLLocalDB;Database=ZombieParty;Trusted\_Connection=True;MultipleActiveResultSets=True"

}

}

# Configurer la relation Zombie et HuntingLog



## Création de la classe Zombie\_HuntingLog

Créer la classe qui permet de résoudre la relation plusieurs à plusieurs entre **Zombie** et **HuntingLog**

1. Dans Models, créez la classe public **ZombieHuntingLog**
2. Ajoutez les clés étrangères explicites (Foreigh key)
3. Ajoutez les propriétés de navigation vers les deux côtés un des deux relations

//[Key]

[ForeignKey("Zombie")]

public int Zombie\_Id { get; set; }

//[Key]

[ForeignKey("HuntingLog")]

public int HuntingLog\_Id { get; set; }

// Propriété de navigation vers zombie

//OBLIGATOIRE Pour la relation 1 à plusieurs avec Zombie

public Zombie Zombie { get; set; }

// Propriété de navigation vers HuntingLog

//OBLIGATOIRE Pour la relation 1 à plusieurs avec HuntingLog

public HuntingLog HuntingLog { get; set; }

1. Ajoutez le DbSet pour la classe **ZombieHuntingLog** dans le ZombiePartyDbContext

public DbSet<ZombieHuntingLog> ZombieHuntingLog { get; set; }

## Modification des classes Zombie et HuntingLog

Ajouter la propriété de navigation entre **Zombie** et **ZombieHuntingLog**

1. Dans Models, ouvrez la classe **Zombie**
2. Ajoutez la propriété de navigation de Zombie vers **ZombieHuntingLog**

// Propriété de navigation vers zombieHuntingLog

//OBLIGATOIRE Pour la relation 1 à plusieurs avec zombieHuntingLog

public ICollection<ZombieHuntingLog> zombieHuntingLogs { get; set; }

1. Dans Models, ouvrez la classe **HuntingLog**
2. Ajoutez la propriété de navigation de Zombie vers **ZombieHuntingLog**

// Propriété de navigation vers zombieHuntingLog

//OBLIGATOIRE Pour la relation 1 à plusieurs avec zombieHuntingLog

public ICollection<ZombieHuntingLog> zombieHuntingLogs { get; set; }

## Création la migration et Mise à à jour la BD

1. Allez dans la console PowerShell (Outils-Gestionnaire de Package-NutGet- Console du gestionnaire de paclage)

Add-migration **AddZombieHuntingLog**

1. Message d’erreur : ZombieHuntingLog n’a pas de clé primaire

## Ajout de la clé composée pour Zombiehuntinglog

1. Ouvrez le fichier **ZombiePartyDbContext**
2. Ajoutez la méthode override **OnModelCreating** prenant le ModelBuilder en paramètre
3. Ajoutez le code qui permet la création de la cllé primaire composée pour la table **ZombieHuntingLog** comprenant la combinaison des deux clés étrangères

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// ZombieHuntingLog: clé composée (composite key)

modelBuilder.Entity<ZombieHuntingLog>().HasKey(zh => new { zh.Zombie\_Id, zh.HuntingLog\_Id });

}

## faire la migration et mettre à jour la BD

1. Allez dans la console PowerShell (Outils-Gestionnaire de Package-NutGet- Console du gestionnaire de paclage)

Add-migration **AddZombieHuntingLog**

1. Consultez le script généré
2. Notez la section de la création de PrimaryKey

table.PrimaryKey("PK\_ZombieHuntingLog", x => new { x.Zombie\_Id, x.HuntingLog\_Id });

1. Notez la section de la création des foreigh keys

"FK\_ZombieHuntingLog\_HuntingLog\_HuntingLog\_Id",

column: x => x.HuntingLog\_Id,

principalTable: "HuntingLog",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_ZombieHuntingLog\_Zombie\_Zombie\_Id",

column: x => x.Zombie\_Id,

principalTable: "Zombie",

principalColumn: "Id",

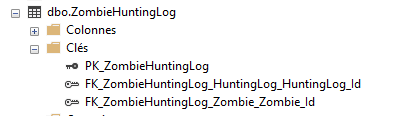
onDelete: ReferentialAction.Cascade);

1. Appliquez les modifications dans la BD

Update-database

## Valider la BD dans SQL Management

1. Ouvrez management Studio afin de valider les modifications



## Générer le schéma de bd

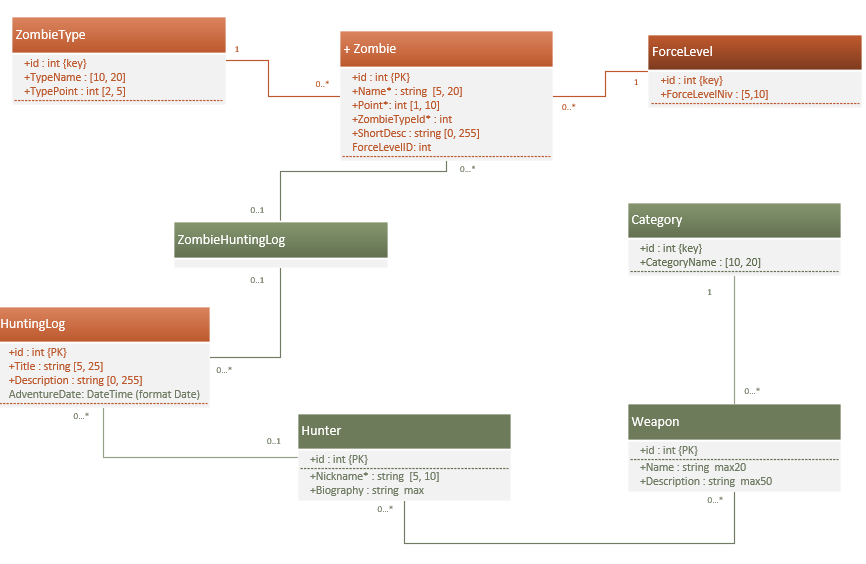
1. Dans Management Studio développez le contenu de la BD ZombieParty
2. Faaites un clic-droit sur le dossier **Diagramme de la base de données, nouveau schéma de la base de données**
3. Installez les packages demandés
4. Sélectionnez toutes les tables

IMAGE BD

## 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 ZombieHuntingLog
   * **Description**: Ajout des clés étrangères entre Zombie et HuntingLog et des propriétés de navigation.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Vue d’ensemble des modifications à venir



# ajout de la fonctionnalité Hunter

## Ajout de la fonctionnalité Hunter

1. Ajoutez la classe **Hunter** dans Models
2. Ajoutez les propriétés nécessaires à la création de la liaison entre **HuntingLog** et **Hunter** dans les deux classes tel que un Hunter (chasseur) peut avoir une ou plusieurs HuntingLog
3. Assurez-vous d’avoir les annotations correspondantes aux schéma pour toutes les propriétés
4. Ajoutez le DbSet pour la classe **Hunter** dans le ZombiePartyDbContext
5. Ajoutez une migration **AddHunter**
6. Mettez à jour la BD

## 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 Hunter et liaison HuntingLog
   * **Description**: Ajouter la classe Hunter et les propriétés de navigation entre Hunter et HuntingLog.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# ajout de la fonctionnalité Weapon

## Ajout des fonctionnalités Category et weapon

1. Ajoutez la classe **Category** dans Models
2. Ajoutez la classe **Weapon** dans Models
3. Ajoutez les propriétés nécessaires à la création de la liaison entre **Weapon** et **Category** dans les deux classes tel qu’un **Weapon** appartient à une seule **Category**
4. Assurez-vous d’avoir les annotations correspondantes aux schéma pour toutes les propriétés
5. Ajoutez les DbSet pour les deuxdans le ZombiePartyDbContext
6. Ajoutez une migration **AddWeaponAndCategory**
7. Mettez à jour la BD

## 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 et Category
   * **Description**: Ajout des classes Weapon et Category et les propriétés de navigation entre deux classes.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# ajout de la fonctionnalité hunter possède un ou plusieurs weapon(s)

## Ajout de la fonctionnalité hunter pouvant avoir plusieurs weapon(s)

1. Ajoutez et/ou modifiez les classes et propriétés nécessaire afin qu’un **Hunter** possède un ou plusieurs **Weapon** et un **Weapon** peut être possédé par un ou plusieurs **Hunter.**
2. Ajoutez la propriété **Quantity,** qui varie selon le **Weapon** et le **Hunter**, dont la valeur par défaut est 1.
3. Assurez-vous d’avoir les annotations correspondantes aux schéma pour toutes les propriétés
4. Ajoutez si nécessaire de(s) DbSet
5. Ajoutez une migration **AddRelationWeaponHunter**
6. Mettez à jour la BD

## 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 et Hunter
   * **Description**: ÉCRIRE UN COMMENTAIRE PERTINENT.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# ajout de propriétés et d’annotations diverses

## Classe HuntingLog

1. Ajoutez la propriété **AdventureDate** qui détermine la Date (dataType)
2. Ajoutez une migration **AddHuntingLogAdventureDate**
3. Mettez à jour la BD

## 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 add AdventureDate
   * **Description**: ÉCRIRE UN COMMENTAIRE PERTINENT.
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 