Laboratoire: semaine 1

# Objectif(s)

* Révision :
  + Fractionner le backend et le frontend en plusieurs Areas
  + Fractionner une solution en plusieurs projets (DataLayers)
  + Mise en place des Repository Patterns
  + Propriétés de navigation (relations)

Projet Jungle

# Vue d’ensemble

L’application **Jungle** est conçue pour les agences de voyage spécialisées dans les safaris. Chaque agence de voyage qui s’abonnent à l’application ont leurs propres conseillers et guides. Les conseillers en voyages *Travel* s’occupent de l’organisation des destinations, forfaits voyages et les options disponibles auxquels les clients peuvent s’inscrire en ligne. L’administrateur s’occupe de la maintenance des données de base (**Country** seulement pour ce laboratoire) et des droits d’accès (laboratoires futurs). Les conseillers affectent également un guide à chaque forfait voyage. Les guides peuvent consulter leurs affectations et modifier leur profil. Les clients peuvent comparer les forfaits voyages et choisir celui (ceux) auxquels ils souhaitent s’inscrire. Une fois inscrits, ils peuvent choisir les options parmi celles offertes par toutes les agences abonnées à l’application.

## Étapes principales du laboratoire :

* Créer la structure multiprojets Net Core 6
* Ajouter les classes de *Model* selon le diagramme fourni
  + Le nombre entre parenthèse après les propriétés texte correspond au nombre de caractère maximum
* Fractionner le projet Web en *areas*
* Mettre en place le *repository* *Patterns* et *Unit of work*
* Créer le modèle de départ
  + Avec les propriétés de navigation
  + Les *PrimaryKey* et *ForeightKey* explicites

**NOTES :**

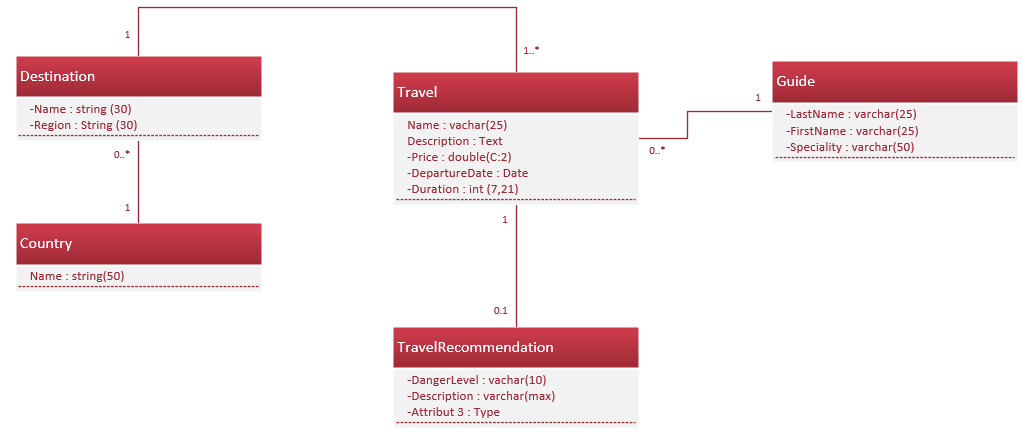
Ce laboratoire ne couvre que certains éléments présentés dans l’étude de cas.

Il est en synchrone, les prochains seront en asynchrone.

Les seules *views* créées seront celles des **Index** de chaque **area**, qui présentent un tableau de chaque classe de base et une *view* **Upsert** et une *view* **Delete**.

Seules les liaisons un à un et un à plusieurs seront définies.

Les classes n’ont que le minimum des champs requis pour accélérer la conception pour ce laboratoire.



# Configurer la solution en *multi-Layer*

## Créer le projet Web MVC

1. Créez un projet de type MVC Net Core appelé **Jungle\_Web** dans la SOLUTION **Jungle**
2. Installez les *packages* : (6.0.22)
   * Microsoft.EntityFrameworkCore
   * Microsoft.EntityFrameworkCore.SqlServer
   * Microsoft.EntityFrameworkCore.Tools
   * Microsoft.VisualStudio.Web.CodeGeneration.Design
3. Testez si l’application démarre bien
4. Ajoutez la connexion dans *appsettings.json*

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)[\\MSSQLLocalDB;Database=](file:///\\MSSQLLocalDB;Database=)Jungle\_01;Trusted\_Connection=True;MultipleActiveResultSets=True"

}

1. Ajoutez un dossier **ViewModels** (et oui, les VM vont avec les Vues!)
2. *Comment-Commit-Push*

## Créer le projet pour les accès aux données

1. Créez un projet de type librairie de classe Net Core appelé **Jungle\_DataAccess**
2. Installez les *packages* : (6.0.22)
   * Microsoft.EntityFrameworkCore
   * Microsoft.EntityFrameworkCore.SqlServer
   * Microsoft.EntityFrameworkCore.Tools
3. Créez une classe **JungleDbContext** qui hérite de **DbContext**
   * Créez le constructeur qui prend en paramètre **DbContextOptions** et exécute le constructeur de base

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

{

}

1. Ajoutez les références requises entre les projets
2. *Comment-Commit-Push*

### Configurer le repository générique

1. Créez un dossier **Repository** et un sous-dossier **IRepository**
2. Créez un interface publique appelé **IRepository** dans le dossier **IRepository**

// Ce doit être une interface générique <T> (T pour s'adapter au type d'objet classe ) publique

public interface IRepository<T> where T : class

{

// Les méthodes devant être implantées dans les repositories

T Get(int id);

IEnumerable<T> GetAll(

Expression<Func<T, bool>> filter = null,

Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,

string includeProperties = null,

bool isTracking = true

);

// Retourne le 1er seulement

T FirstOrDefault(

Expression<Func<T, bool>> filter = null,

string includeProperties = null,

bool isTracking = true

);

void Add(T entity);

void Remove(T entity);

void RemoveRange(IEnumerable<T> entity);

void Save();

}

1. Créez une classe publique appelée **Repository** dans le dossier **Repository** qui implémente IRepository
2. Implémentez l’interface (au besoin référez-vous au projet démo)

### Configurer le UnitOfWork

1. Créez un interface publique appelé **IUnitOfWork** qui implémente l’interface **IDiposable** dans le dossier **IRepository**
2. Ajoutez la signature pour Save()
3. Créez une classe publique appelée **UnitOfWork** dans le dossier **Repository** qui implémente **IunitOfWork**
4. Liez le **UnitOfWork** au **DbContext**
5. Ajoutez les références requises entre les projets
6. *Comment-Commit-Push*

## Créer le projet pour le Modèle de données

1. Créez un projet de type librairie de classe Net Core appelé **Jungle\_Models**
2. Installez les *packages* : 6.0.22
   * Microsoft.EntityFrameworkCore
   * Microsoft.AspNetCore.Mvc.ViewFeatures
3. Créez un dossier:
   * Models

## Créer les *areas*

1. Au niveau du projet MVC, créez la hiérarchie de dossiers pour les *areas* en fonction du contexte.
2. Dans le **area** du client (nom que vous lui avez donné), déplacez le HomeController et les views Home Index et Privacy
3. Modifiez le *rooting path* par défaut dans le Setup pour qu’il renvoit au **Home Index**
4. Ajoutez les références requises entre les projets

# Ajouter les fonctionnalités **une par une**

Une par une ça veut dire : Model(s), DbSet(s), controller et views pour une fonctionnalité. Nous c’est petit, mais généralement plusieurs tables par fonctionnalité. Voici les fonctionnalités pour ce labo :

|  |  |  |
| --- | --- | --- |
| Fonctionnalités | Model(s) impliqués | AREAS |
| Destinations | Destination  Country | Admin |
| Travels (voyages) | Travel  TravelRecommendation | Conseillers |
| Guides | Guide | Guides |

## Ajouter une fonctionnalité

**Pour chaque fonctionnalité :**

**Models :**

1. Créez la/les classe/s dans le projet **Jungle\_Models**/**Models**
2. Ajoutez les annotations requises
3. Ajoutez le/les DbSet/s dans le projet **Jungle\_DataAccess**/**JungleDbContext**
4. Ajoutez une *migration-update database* Attention au projet par défaut dans le haut de la console!
5. Vérifiez que votre BD contient vos modifications
6. *Comment-Commit-Push*

**Repository Pattern :**

1. Dans le projet **Jungle\_DataAccess** , ajoutez un interface public spécifique pour l’entité héritant de l’interface générique
2. Ajoutez dans l’interface la signature pour le **Update**
3. Définir une classe de **repository** spécifique pour la classe, qui hérite de la classe générique et implémente l’interface spécifique
4. Ajoutez dans l’interface **IUnitOfWork** la propriété du type de l’interface spécifique
5. Ajoutez le code nécessaire à la création du **repository** spécifique

**Views Controllers :**

1. Dans le projet **Jungle\_Web** Ajoutez les *views* et le *Controller* dans le AREA requis qui présente le contenu avec le(s) principaux champs des tables liées
   * **Index** (toutes les fonctionnalités)
   * **Upsert** (seulement pour *Country* et *Destination)* avec une liste déroulante des *Country* dans la *view* *Destination* en ordre alphabétique
   * **Delete** (seulement pour *Country* et *Destination)*
2. *Comment-Commit-Push*

NOTE : Ne perdez pas de temps sur la mise en page des *Views*,