Laboratoire: semaine 2

# 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

# Créer le repo pour les laboratoires individuels

## Télécharger les fichiers de départ du laboratoire

1. Dans GitHub, allez sur le repo GitHub [**A21\_5W5\_Laboratoires**](https://github.com/valerieturgeon/A21_5W5_Laboratoires.git)
2. Sélectionnez la branche S02\_DataLayers\_RepoPatterns
3. Téléchargez et décompressez (*dézipez*) les fichiers de départ

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

1. Si ce n’est pas déjà fait, dans Github créez un repository GitHub **A21\_5W5\_Labos\_NOM\_PRENOM**
   * **ReadMe**
   * **Gitignore (Visual Studio)**

N’ajoutez rien d’autre dans la branche MAIN

1. Dans Gitkraken, clonez votre repository
2. Créez la branche de la séance **S02\_DataLayers\_RepoPatterns**
3. Copiez la solution de départ (pas README et GITIGNORE) dans le dossier de votre clone
4. *Comment-commit-push*

Projet Jungle

# Vue d’ensemble

L’application est pour une agence de voyage spécialisée dans les safaris. Les conseillers en voyages s’occupe de l’organisation des destinations, forfaits voyages et les optionsdisponibles auxquels les clients peuvent s’inscrire en ligne. L’administrateur s’occupe de la maintenance des données de base et des droits d’accès. Les conseillers affectent également un guide à chaque forfait voyage. Les guides peuvent consulter leur affections et modifier leur profil. Les clients peuvent comprer les forfaits voyages et choisir celui (ceux) auxquels ils souhaites s’inscrirent. Un fois inscris, ils peuvent choisir les options parmis celles offertes.

## Objectifs du laboratoire :

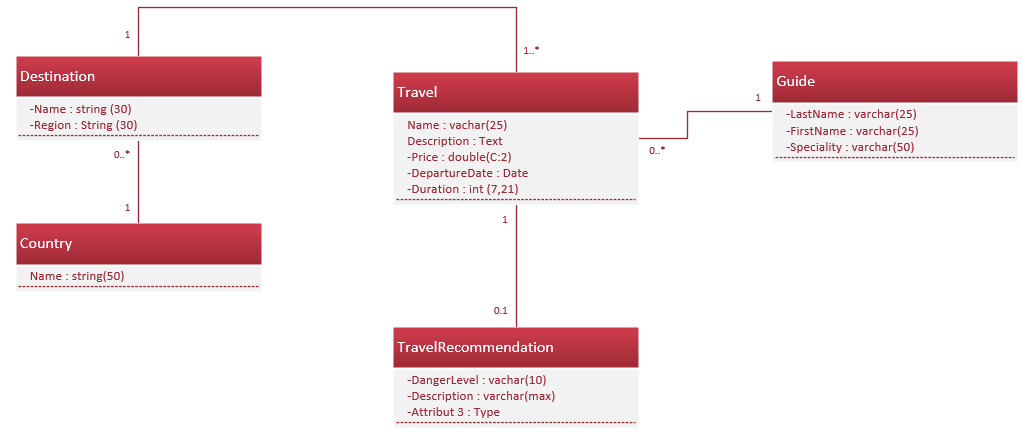
* Créer la structure multi-projets
* Fractionner le projet en areas
* Mettre en place le repository Patterns et Unit of work
* Créer le modèle de départ
  + Avec les propriétés de navigation
  + Les PrimaryKey et ForeightKey explicites

**NOTE :**

Les seules Views créées seront celles des Index de chaque area, qui présentent un tableau de chaque classe de base.

Seules les liaisons un à un et un à plusieurs seront définies (les plusieurs à plusieurs seront ajoutées dans le prochain lab)

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



# Configurer la solution en multi-Layers

## Configurer la BD du projet

1. Modifiez la connexion dans Appsettings.json
2. Consultez les classes modèles fournies
3. Effectuez une migration initiale et update database

## Créer le projet pour les accès aux données

1. Créez un projet de type librairie de classe Net Core
2. Insallez les packages :
   * 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

### Configurer le repository générique

1. Créez un dossier **Repository** et un sous dossier **IRepository**
2. Créez un interface public 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 public 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 public appelé **IUnitOfWork** qui implémente l’interface **IDiposable** dans le dossier **IRepository**
2. Ajoutez la signature pour Save()
3. Créez une classe public appelée **UnitOfWork** dans le dossier **Repository** qui implémente **IunitOfWork**
4. Liez le UnitOfWork au DbContext

## Créer le projet pour le Modèle de données

1. Créez un projet de type librairie de classe Net Core
2. Insallez les packages :
   * Microsoft.EntityFrameworkCore
   * Microsoft.AspNetCore.Mvc.ViewFeatures

## 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

## Ajouter une fonctionnalité

**Pour chaque entité :**

**Modèle :**

1. Créez la classe dans le projet Models
2. Copiez les propriétés de départ (fichiers complémentaires)
3. Ajoutez les annotations requises
4. Ajoutez une migration-update database

**Repository Pattern :**

1. 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