

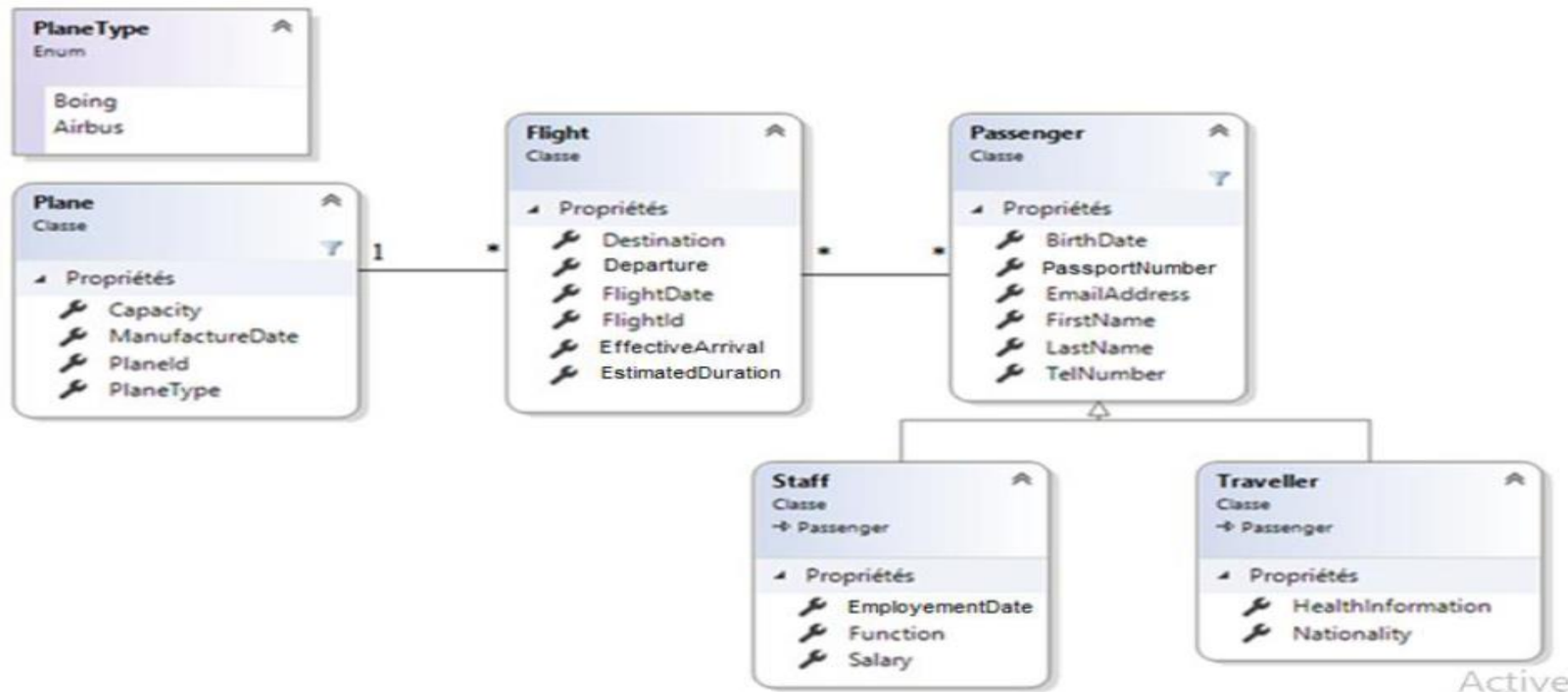


TP AIRPORT MANAGEMENT

Tarek Ayari

MISE EN PLACE DE LA SOLUTION:

Notre objectif principal est de développer une application pour la gestion des activités d'un aéroport, utilisant le framework .NET 8. Soit le diagramme de classe suivant:



PARTIE 1 : DIAGRAMME DE CLASSES

- Créer une solution vide nommé “AirportManagement” et y ajouter les projets suivants
 1. **AM.UI.Console** : Projet de type application Console (.NET 8.0)
 2. **AM.ApplicationCore** : Projet de type Bibliothèque de classe (.NET 8.0)
 3. Sous le projet AM.ApplicationCore, créer le dossier « **Domain** » et y implémenter les différentes classes du diagramme de classes ci-dessus
 4. Ajouter la référence de projet Core dans le projet Console

PARTIE 1 : DIAGRAMME DE CLASSES

4. Représenter l'héritage entre la classe **Passenger** et les deux classes **Staff** et **Traveller**
5. Implémenter les propriétés qui représentent les différents attributs.
6. Représenter les relations au biais des objets de navigation
 - a. Par exemple, la relation 1-* entre **Plane** et **Flight** sera représentée par les objets de navigation suivants :
 - i. — Une propriété de type **ICollection<Flight>** dans la classe **Plane**
 - ii. — Une propriété de type **Plane** dans la classe **Flight**

PARTIE 1 : DIAGRAMME DE CLASSES

7. Implémenter le diagramme de classes suivant :
 - a. Créer les relations **entre les différentes classes** et décorer les propriétés de navigation par le mot clé Virtual
 - i. **One to many** entre les entités « Plane » et « Flight»
 - ii. **Many to many** entre les entités « Flight» et « Passenger»
8. Réimplémenter **la méthode ToString()** pour toutes les classes qui retourne les propriétés de chaque classe

PARTIE 2 : INSTANCIATION DES OBJETS

6. Créer un objet non initialisé de type Plane en utilisant le constructeur non paramétré de la classe, puis initialiser ses attributs à travers leurs propriétés.
7. Créer le constructeur suivant pour la classe Plane.
 - `public Plane (PlaneType pt, int capacity, DateTime date)`
 - Puis créer un autre avion en utilisant ce constructeur.
 - Supprimer le constructeur crée précédemment et instancier un autre avion en utilisant les initialiseurs d'objet.
 - → C'est plus simple et intuitif d'utiliser les initialiseurs d'objets.

PARTIE 3 : POLYMORPHISME DE SURCHARGE

8. Dans l'entité **Passenger**, créer les deux méthodes **bool CheckProfile(...)**

suivantes :

- a. Une méthode pour vérifier le profile en utilisant **deux paramètres: nom** du passager et **prénom** du passager.
- b. Une méthode pour vérifier le profile en utilisant **trois paramètres: nom** du passager, **prénom** du passager et **email** du passager.
- c. Tester les deux méthodes dans le program *Main* en créant un objet *Passenger*.

PARTIE 4 : POLYMORPHISME D'HERITAGE

9. Dans la classe `Passenger`, implémenter la méthode **`PassengerType`** qui affiche « I am a passenger ».

- «I am a passenger » si l'objet est de type «`Passenger`».
 - «I'm a traveller"» si l'objet est de type «`Traveller`».
 - «I'm a staff member» si l'objet est de type «`Staff`».
- Tester la méthode `PassengerType` dans le projet console pour 3 instances de types **`Passenger`, `Staff` et `Traveller`**.

PARTIE 4 : INTERFACE ET SERVICE

10. Sous le projet « AM.ApplicationCore », créer les deux dossiers **Interfaces** et **Services**.
11. Créer l'interface **IServiceFlight** dans le dossier Interfaces et la classe **ServiceFlight** dans le dossier Services.
12. Dans la classe ServiceFlight, créer la propriété `public List<Flight> Flights {get ; set ;}` et l'initialiser à une liste vide.
 - Nous formulerons les requêtes de ce TP en se basant sur cette liste comme source de données.
- N'oubliez pas de mettre à chaque fois la signature de chaque méthode dans l'interface

PARTIE 5 : PRÉPARATION DES DONNÉES DE TEST

13. Ajouter dans le projet « AM.Domain » la classe statique **TestData** qui contient les données statiques de test du tableau suivant.

- **Planes:**

Planes		
PlaneType	Capacity	ManufactureDate
Boing	150	03/02/2015
Airbus	250	11/11/2020

PARTIE 5 : PRÉPARATION DES DONNÉES DE TEST

- **Staff:**

Staff					
FirstName	LastName	EmailAddress	BirthDate	EmploymentDate	Salary
captain	captain	Captain.captain@gmail.com	01/01/1965	01/01/1999	99999
hostess1	hostess1	hostess1.hostess1@gmail.com	01/01/1995	01/01/2020	999
hostess2	hostess2	hostess2.hostess2@gmail.com	01/01/1996	01/01/2020	999

PARTIE 5 : PRÉPARATION DES DONNÉES DE TEST

- Travellers:

Travellers					
FirstName	LastName	EmailAddress	BirthDate	HealthInformation	Nationality
Traveller1	Traveller1	Traveller1. Traveller1@gmail.com	01/01/1980	No troubles	American
Traveller2	Traveller2	Traveller2. Traveller2@gmail.com	01/01/1981	Some troubles	French
Traveller3	Traveller3	Traveller3. Traveller3@gmail.com	01/01/1982	No troubles	Tunisian
Traveller4	Traveller4	Traveller4. Traveller4@gmail.com	01/01/1983	Some troubles	American
Traveller5	Traveller5	Traveller5. Traveller5@gmail.com	01/01/1984	Some troubles	Spanish

PARTIE 5 : PRÉPARATION DES DONNÉES DE TEST

- Flights:

Flights					
FlightDate	Destination	EffectiveArrival	Plane	EstimatedDuration	Passengers
01/01/2022 15:10:10	Paris	01/01/2022 17:10:10	Airbus	110	All created travellers
01/02/2022 21:10:10	Paris	01/02/2022 23:10:10	Boing	105	
01/03/2022 5:10:10	Paris	01/03/2022 6:40:10	Boing	100	
01/04/2022 6:10:10	Madrid	01/04/2022 8:10:10	Boing	130	
01/05/2022 17:10:10	Madrid	01/05/2022 18:50:10	Boing	105	
01/06/2022 20:10:10	Lisbonne	01/06/2022 22:30:10	Airbus	200	

PARTIE 5 : PRÉPARATION DES DONNÉES DE TEST

- Dans la même classe, **créer la liste statique `List<Flight> listFlights`** et l'initialiser avec tous les vols créés précédemment.
- Dans le projet console, **créer une instance de la classe `ServiceFlights`** puis affecter `listFlights` à la **propriété `Flights`** de cette classe service.
- Tester toutes les méthodes qui suivent en se basant sur ces données de test.

PARTIE 5.1 : ITÉRATIONS / STRUCTURES CONDITIONNELLES

11. En utilisant la boucle For, implémenter la méthode **GetFlightDates (string destination)** dans la classe **ServiceFlight** qui retourne la liste des dates de vols d'une destination passée en paramètre

- Tester la méthode **GetFlightDates ()** avec plusieurs paramètres
- Reformuler la fonction en utilisant `foreach`, Re-Tester la méthode **GetFlightDates ()**

PARTIE 5.2 : ITÉRATIONS / STRUCTURES CONDITIONNELLES

11. Implémenter la méthode **GetFlights(string filterType, string filterValue)** qui affiche les vols en fonction de type de filtre et sa valeur. Le type de filtre représente un attribut de la classe Flight.

- Par exemple `GetFlights("Destination", "Paris")` permettra d'afficher les vols dont la valeur de Destination est égale Paris.
- Tester la méthode `GetFlights ()` avec plusieurs paramètres

LINQ



Présentation
Microsoft PowerPoint

PARTIE 6 : LINQ

12. Implémenter les méthodes suivantes et les tester à chaque fois dans le projet

Console:

- Reformuler la méthode **GetFlightDates(string destination)** en utilisant une requête LINQ.
- **ShowFlightDetails(Plane plane):** Afficher les dates et les destinations des vols d'un avion passé en paramètre
- **ProgrammedFlightNumber(DateTime startDate):** Retourner le nombre de vols programmés pour une semaine (7jours) à partir d'une date donnée

PARTIE 6 : LINQ

12. Implémenter les méthodes suivantes et les tester à chaque fois dans le projet

Console:

- **DurationAverage(string destination):** Retourner la moyenne de durée estimées des vols d'une destination donnée
- **OrderedDurationFlights():** Retourner les Vols ordonnés par EstimatedDuration du plus long au plus court

PARTIE 6 : LINQ

- **SeniorTravellers(Flight flight):** Retourner les 3 passagers, de type traveller, les plus âgés d'un vol
- **DestinationGroupedFlights():** Retourner les vols groupés par destination et les afficher sous ce format

Destination Paris

Décollage : 03/05/2022 12 : 10 :00

Décollage : 05/05/2022 23 : 00 :00

Décollage : 10/05/2022 21 : 15 :00

Destination Madrid

Décollage : 01/05/2022 10 : 10 :00

Décollage : 02/05/2022 13 : 10 :00

PARTIE 7 : LES MÉTHODES D'EXTENSION

14. Créer la classe **PassengerExtension** dans le projet « AM.ApplicationCore » qui étend la classe **Passenger** et qui contient la méthode d'extension suivante.

- **UpperFullName(Passenger p)** : qui met en majuscule la première lettre du nom et du prénom d'un passager.
- Tester la méthode d'extension dans l'application Console

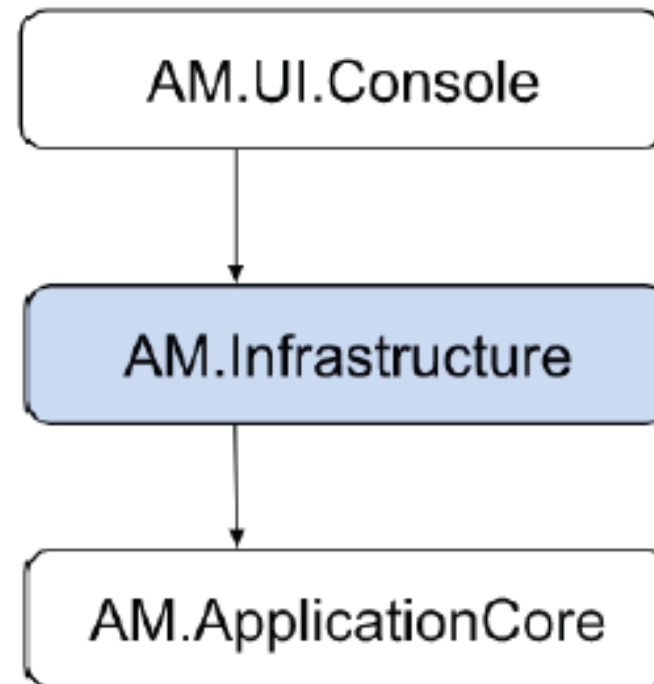
Entity Framework Core



Présentation
Microsoft PowerPoint

PART 9 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

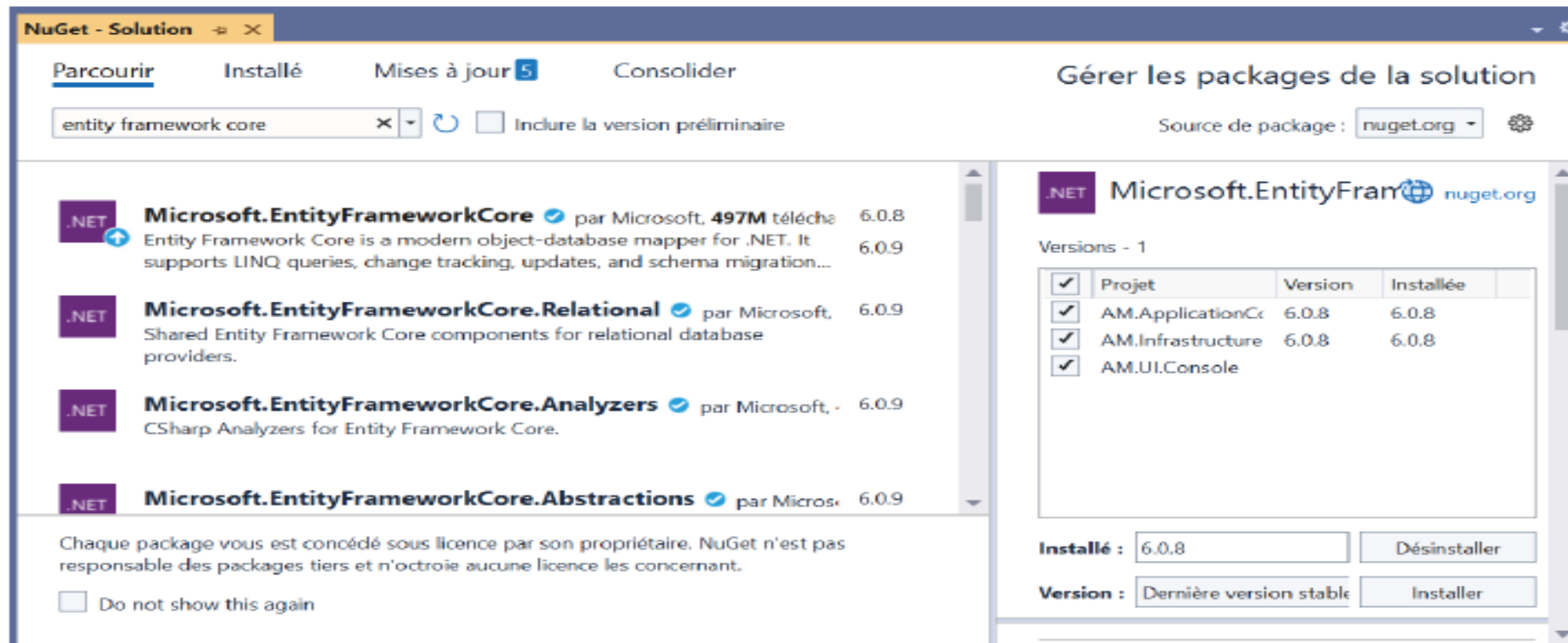
- Dans la solution nommée “AirPortManagement”, ajouter le projet suivant:
- AM.Infrastructure : Projet de type Bibliothèque de classe
- Ajouter les références entre les projets: Le projet AM.UI.Console doit référencer le projet AM.Infrastructure. Le projet AM.Infrastructure doit référencer le projet AM.ApplicationCore.



PART 9 – IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

- Installer les packages nécessaires d'Entity Framework Core dans 'Infrastructure', 'ApplicationCore', 'UI.Console' à travers 'NuGet Package'.

1/ Microsoft.EntityFrameworkCore V8.0.8



PART 9 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

- Installer EntityFrameworkCore.Design dans le projet AM.UI.Console'.
- Installer EntityFrameworkCore.Tools dans le projet AM.Infrastructure
- Installer EntityFrameworkCore.SqlServer dans le projet AM.Infrastructure

2/ Microsoft.EntityFrameworkCore.SqlServer V8.0.8

3/ Microsoft.EntityFrameworkCore.Design V8.0.8

4/ Microsoft.EntityFrameworkCore.Tools V8.0.8

Entity Framework Core: Chargement des entités associées



Présentation
Microsoft PowerPoint

Entity Framework Core DbContext : Introduction



Microsoft
PowerPoint Presentation

Entity Framework Core - Héritage : Introduction



Microsoft
PowerPoint Presentation

Entity Framework CORE - migration: Introduction



Microsoft
PowerPoint Presentation

PART 10 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

Etape 2 : Implémentation du Contexte

1. Créer une classe AMContext dans le projet « AM.Infrastructure »
2. Ajouter le namespace suivant: `using Microsoft.EntityFrameworkCore;`
3. Ajouter l'héritage de DbContext
4. Redéfinir la méthode OnConfiguring
5. Ajouter la chaîne de connexion:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer(@"Data Source=(localdb)\mssqllocaldb;
        Initial Catalog=AirportManagementDB;Integrated Security=true");
    base.OnConfiguring(optionsBuilder);
}
```

PART 10 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

Etape 2 : Implémentation du Contexte

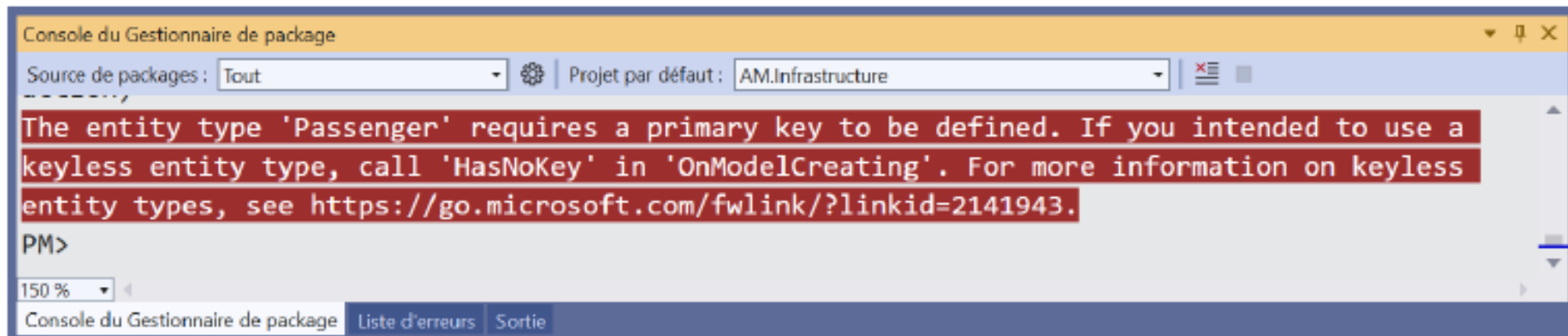
1. Installer le package `Microsoft.EntityFrameworkCore.Proxies` dans le Projet Infrastructure et `ApplicationCore`
2. Activer l'utilisation de la fonction `UseLazyLodingProxies()` dans la fonction `OnConfiguring()`
3. Ajouter les DBSet nécessaires.

PART 10 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

Etape 3 : Migration

1. A partir de la console du gestionnaire de package et en ciblant le projet AM.Infrastructure:
 - Exécuter la commande qui permet d'ajouter une migration

La console nous renvoie ce message:



The screenshot shows the 'Console du Gestionnaire de package' window. At the top, there are dropdowns for 'Source de packages' (set to 'Tout') and 'Projet par défaut' (set to 'AM.Infrastructure'). The main area displays a red error message: 'The entity type 'Passenger' requires a primary key to be defined. If you intended to use a keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless entity types, see <https://go.microsoft.com/fwlink/?linkid=2141943>.' Below the message, the prompt 'PM>' is visible. At the bottom, there is a zoom level dropdown set to '150 %' and a tab bar with 'Console du Gestionnaire de package', 'Liste d'erreurs', and 'Sortie'.

```
Console du Gestionnaire de package
Source de packages : Tout
Projet par défaut : AM.Infrastructure

-----
The entity type 'Passenger' requires a primary key to be defined. If you intended to use a
keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless
entity types, see https://go.microsoft.com/fwlink/?linkid=2141943.
PM>
150 %
Console du Gestionnaire de package  Liste d'erreurs  Sortie
```


PART 10 — IMPLÉMENTATION DU CONTEXTE ET DES ENTITÉS:

Etape 4 : Migration

1. Dans la classe Passenger, ajouter la propriété **public int Id { get; set; }**
2. A partir de la console du gestionnaire de package et en ciblant le projet AM.Infrasructure:
 - Exécuter la commande qui permet d'ajouter une migration
 - Exécuter la commande qui permet de mettre à jour la base de données
 - Allez vers Affichage → Explorateur d'objets SQL Server sous visual studio

PART 11 —MISE À JOUR DU MODÈLE:

Etape 5 : Mise a jour Modele

1. Ajouter la propriété public string Airline { get; set; } à l'entité Flight.
2. Ajouter une migration nommée NewPropertyAirline.
3. Exécuter la commande qui permet de mettre à jour la base de données **Update-Database**
4. Changer le nom de la propriété public string Airline {get; set;} à public string AirlineLogo {get; set;}.
5. Ajouter une migration nommée ModifyAirlineProperty et mettre à jour la base de données.
6. Restaurer le schéma de la base de données à l'état précédent.

Entity Framework - Data Annotations: Introduction



Microsoft
PowerPoint Presentation

PARTIE 12 — LES ANNOTATIONS:

Etape 1 : Ajouter using “System.ComponentModel.DataAnnotations” au différentes classes de dossier “Domain”

Etape 2 : Ajouter les différentes annotations qui nous permettent de configurer les entités comme suit :

Dans la class Passenger

- La propriété **PasseportNumber** doit être:
 - un champ de 7 caractères cote Interface
- La propriété **FirstName** doit être :
 - de longueur minimale 3 caractères
 - de longueur maximale 25 caractères
 - un message d'erreur doit être affiché si les règles ne sont pas respectées

PARTIE 12 — LES ANNOTATIONS:

- La propriété **BirthDate** doit être:
 - affichée “Date of Birth”
 - une date valide
- La propriété **EmailAddress** doit être:
 - une adresse mail valide
- La propriété **TelNumber** doit être :
 - un numéro de téléphone contenant 8 chiffres

Dans la class Plane:

- La propriété **Capacity** doit être:
 - un entier positif

PARTIE 12 — LES ANNOTATIONS:

Dans la class Flight:

- La propriété **PlaneId** doit être
 - La clé étrangère correspondante à l'entité Plane

Dans la class Staff:

- La propriété **Salary** doit être:
 - Une valeur monétaire

- Mettez à jour la base de données en utilisant la migration (Nom: **EnableAnnotation**)

Entity Framework - Fluent API: Introduction



Microsoft
PowerPoint Presentation

PARTIE 13 —FLUENT API:

NB : les annotations et la configuration utilisant FluentApi peuvent cohabiter dans un même projet

Etape 1 :

- Ajouter un nouveau dossier “Configurations” dans le projet “AM.Infrastructure”

Etape 2 :

- Ajouter la class `PlaneConfiguration` dans le dossier “Configurations”
- Inclure `Microsoft.EntityFrameworkCore.Metadata.Builders`
 - Le nom de la table correspondante à l’entité Plane dans la base de données doit être **“MyPlanes”**
 - Le nom de la colonne correspondante à la propriété **Capacity** dans la base de données doit être **PlaneCapacity**

PARTIE 13 —FLUENT API:

- Ajouter la class `FlightConfiguration` dans le dossier “Configurations”
- Inclure `Microsoft.EntityFrameworkCore.Metadata.Builders`
 - Configurer la relation **many-to-many** entre la classe `Flight` et la classe `Passenger`
 - Renommer la table d’association « **Reservation** »
 - Configurer la relation **one-to-many** entre la classe `Flight` et la classe `Plane` avec la clé étrangère « `PlaneId` » et mettre a nulle les instances filles dans l’action de suppression
- Faire appel aux classes de configuration que nous venons de créer dans la classe `AMContext` dans la redéfinition de la méthode `OnModelCreating`

PARTIE 13 —FLUENT API:

Heritage:

- Configurer l'héritage schématisé dans le diagramme de classe en changeant le champ discriminator vers `IsTraveller` et configure les valeurs suivantes:
 - " 1 " si le type de de Passenger est Traveller,
 - "2" si le type de Passenger est Staff
 - la valeur "0" sinon
- Mettre à jour la base de données en utilisant la migration

PARTIE 13 —FLUENT API:

Pré-Conventions:

- Dans la classe `AMContext`, redéfinir la méthode `ConfigureConvention`
- Configurer toute les propriétés de type `DateTime`
 - Le type des colonnes correspondantes à ces propriétés dans la base de données doit être “datetime2” à la place de type par défaut “datetime”
- Mettre à jour la base de données en utilisant la migration.

PARTIE 13 —FLUENT API:

Types d'entité détenus:

- Ajouter la classe FullName dans le dossier Domain du projet AM.ApplicationCore.

```
namespace AM.ApplicationCore.Domain
{
    0 références
    public class FullName
    {
        0 références
        public string FirstName { get; set; }
        0 références
        public string LastName { get; set; }
    }
}
```

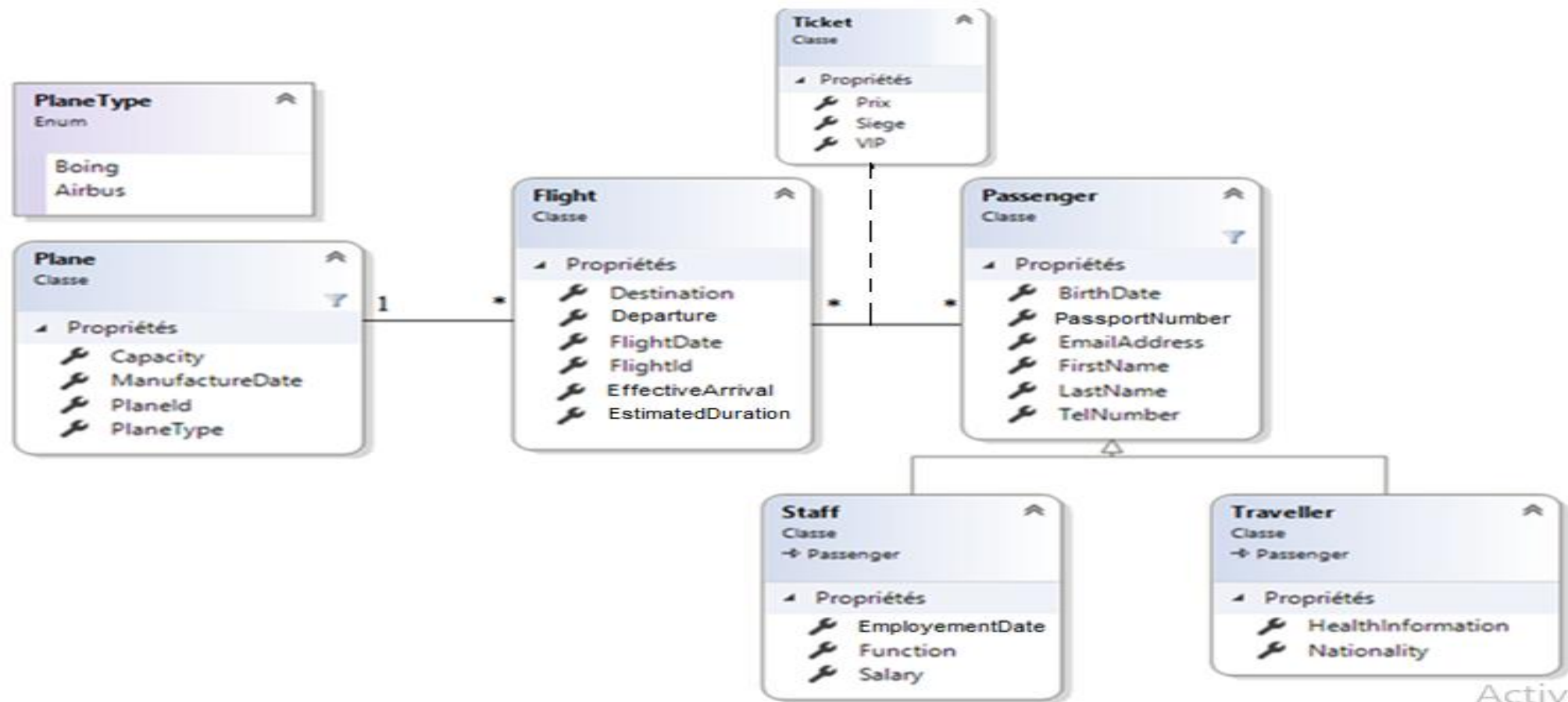
PARTIE 13 —FLUENT API:

Types d'entité détenus:

- Mettre à jour l'entité **Passenger** par le type détenu que nous venons de créer.
- Ajouter la class **PassengerConfiguration** dans le dossier Configurations:
 - Configurer le type d'entité détenu FullName
 - La propriété FirstName a une **longueur maximale de 30** et le nom de la colonne correspondante à cette propriété dans la base de données doit être **PassFirstName**
 - La propriété LastName est **obligatoire** et le nom de la colonne correspondante à cette propriété dans la base de données doit être **PassLastName**
- Faire appel à la classe de PassengerConfiguration dans la classe AMContext.
- Mettre à jour la base de données en utilisant la migration.

PART 14 — TABLE PORTEUSE DE DONNÉES :

Etape 1 : 3. Dans le projet AM.ApplicationCore ajouter la classe Ticket schématisés dans le diagramme suivant avec tous les champs:



PART 15 — TABLE PORTEUSE DE DONNÉES :

- Ajouter la propriété de navigation entre les classes Ticket, Passenger et Flight.
- Ajouter les clés étrangères PassengerFk et FlightFk dans la classe Ticket
- Ajouter la propriété de navigation de la classe Ticket dans la classe Passenger
- Ajouter la propriété de navigation de la classe Ticket dans la classe Flight

PART 15 — TABLE PORTEUSE DE DONNÉES :

Etape 1 :

- Ajouter la class “TicketConfiguration” dans le dossier “Configurations” :
- Configurer la clé primaire composée suivante (PassengerFk , FlightFk , NumTicket)
- Configurer la relation One to Many avec la classe Passenger dans le premier sens
- Configurer la relation One to Many avec la classe Flight dans le deuxième sens

Etape 2 : Ajouter le DbSet de l’entité Ticket dans la classe AMContext.

Etape 3 : Mettre à jour le Context pour [faire appel a cette nouvelle class configuration](#) que nous venons de créer

Etape 4 : Faire appel à la classe de TicketConfiguration dans la classe AMContext.

Etape 5 : Mettez à jour la base de données en utilisant la migration (AddTable)

PART 15 — INSERTION DES DONNÉES:

- Faite une opération d'ajout d'un Flight en lui attribuant un Plane et une opération d'affichage de ce Flight puis lancer l'application.

```
AMContext context = new AMContext();  
context.Flights.Add(TestData.flight2);  
context.SaveChanges();  
Console.WriteLine(context.Flights.First());
```

- Vérifier cote Base de données si l'insertion est réalisé

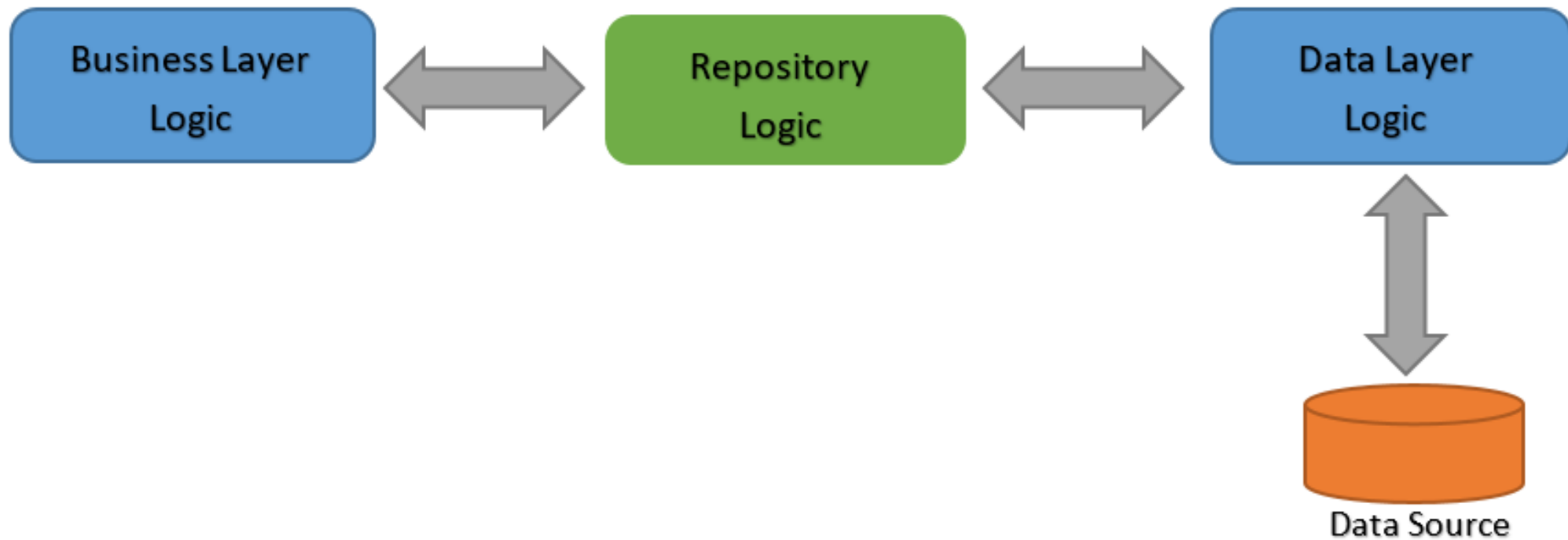
Design Pattern: Introduction



Microsoft
PowerPoint Presentation

PART 16 — DESIGN PATTERNS :

Etape 1: Implémentation du patron Repository:



PART 16 — DESIGN PATTERNS :

Etape 1:

Implémentation du patron Repository:

- Sous le dossier Interfaces du projet « AM.ApplicationCore », , ajouter une interface générique nommée **IGenericRepository**. Cette dernière contient toutes les méthodes CRUD.
- Sous le projet **AM.Infrastructure**, créer une classe nommée **GenericRepository**. Cette dernière implémente l'interface **IGenericRepository**

PART 16 — DESIGN PATTERNS :

Etape 1 :

- 1. Sous le dossier Interfaces du projet AM.ApplicationCore:
 - Créer l'interface de services IServicePlane contenant les trois méthodes suivantes :
 - Une méthode Add() qui permet d'ajouter dans la BD un avion.
 - Une méthode Save() qui permet de sauvegarder dans la BD un avion.
 - Une méthode GetAll() qui permet de retourner tous les avions enregistrés dans la BD.

```
1 référence
public interface IServicePlane
{
    2 références
    public void Add(Plane p);
    0 références
    public void Save();
    0 références
    public IList<Plane> GetAll();
}
```

PART 16 — DESIGN PATTERNS :

1. Sous le dossier Services,

- Créer la classe ServicePlane qui implémente l'interface IServicePlane.
- Créer un constructeur qui prend en paramètre du constructeur un objet de type IRepositoryGeneric.
- Implementer les methodes Add, Save, GetAll en passant par l'instance Repository.

```
private IRepositoryGeneric<Plane> genericRepository;
```

0 références

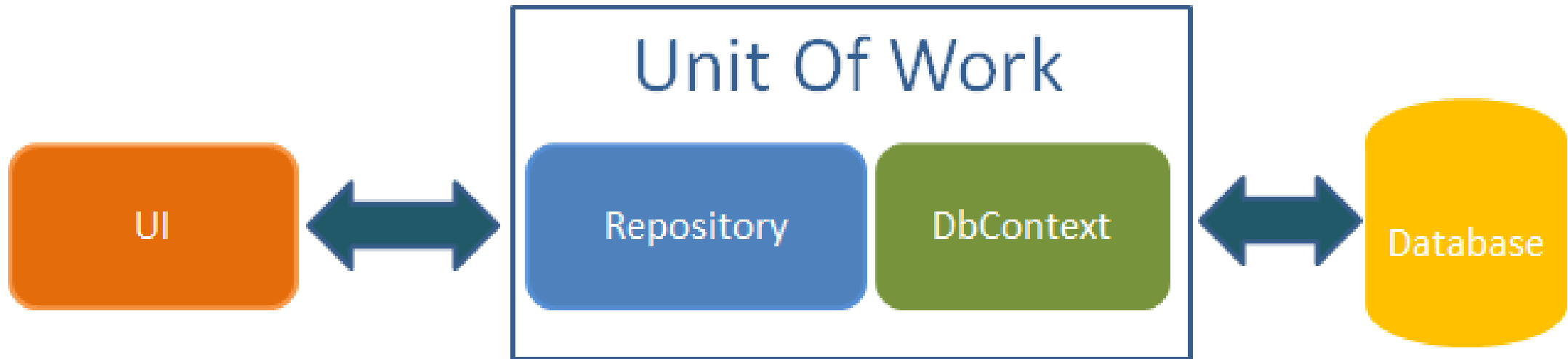
```
public ServicePlane(IRepositoryGeneric<Plane> genericRepository)  
{  
    this.genericRepository = genericRepository;  
}
```

PART 16 — DESIGN PATTERNS :

- Tester l'ajout d'un avion cote Console
- Que Constater vous?

PART 16 — DESIGN PATTERNS :

Etape 2 : Implémentation du patron UnitOfWork:



PART 16 — DESIGN PATTERNS :

Etape 2 :

Implémentation du patron UnitOfWork:

- Sous le dossier Interfaces du projet « AM.ApplicationCore », ajouter une interface ajouter une interface nommée IUnitOfWork contenant une propriété de type DbContext et une méthode Save().
- Sous le projet **AM.Infrastructure**, créer une classe nommée **UnitOfWork** qui implémente l'interface **IUnitOfWork**

PART 16 — DESIGN PATTERNS :

1. Sous le dossier Services,

- Modifier la classes ServicePlane afin d'utiliser un objet de type IUnitOfWork
- Créer un constructeur qui prend en paramètre du constructeur un objet de type IUnitOfWork.

```
private IUnitOfWork unitOfWork;  
0 références  
public ServicePlane(IUnitOfWork unitOfWork)  
{  
    this.unitOfWork = unitOfWork;  
}
```

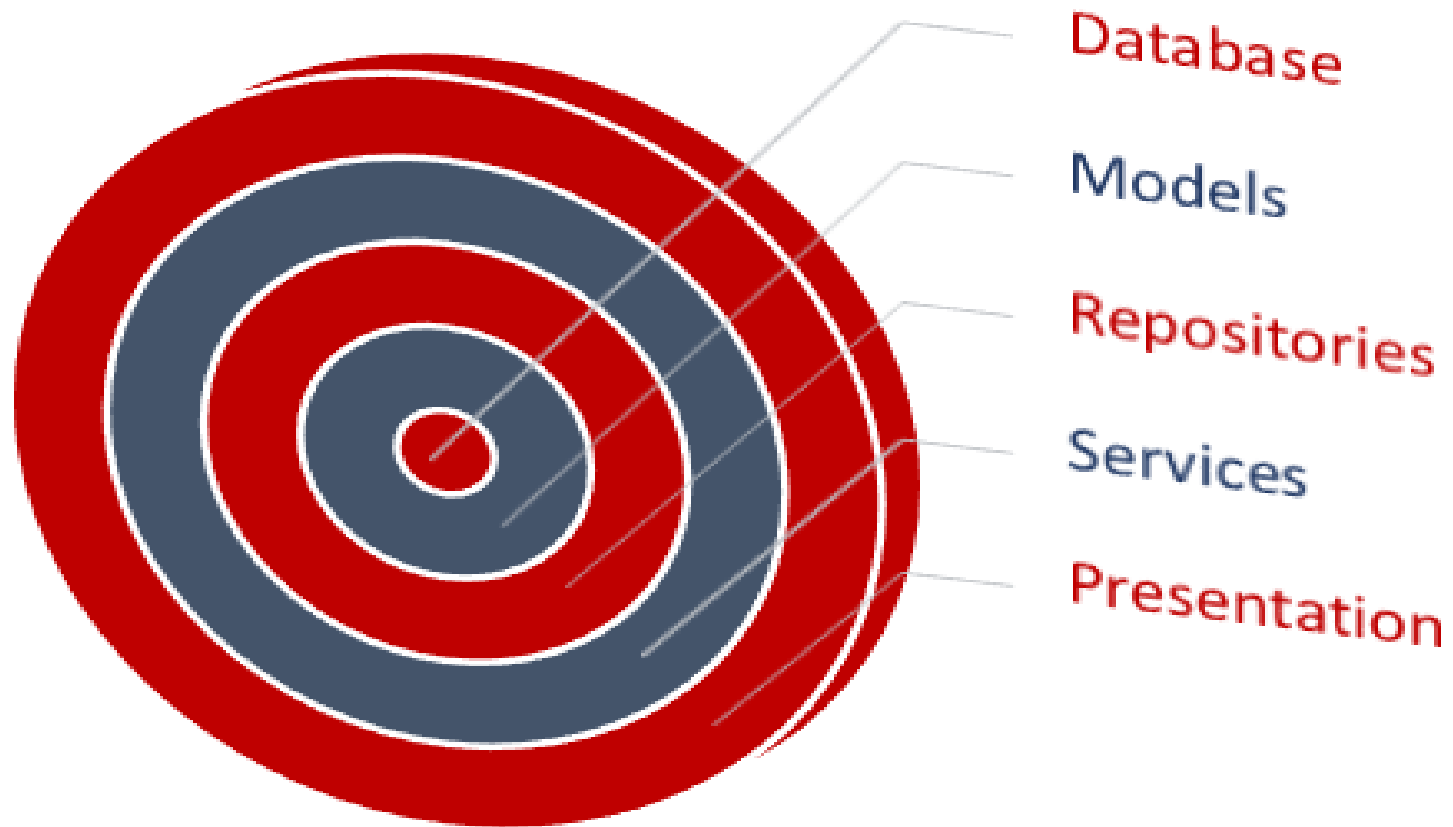
PART 16 — DESIGN PATTERNS :

- Tester l'ajout d'un avion cote Console
- Que Constater vous?

```
ServiceFlight service = new ServiceFlight(new UnitOfWork(ctx, typeof(GenericRepository<>)));  
service.Add(f1);
```

PART 17 —SERVICE GENERIQUE:

Etape1: Service Generique



PART 17 —SERVICE GENERIQUE:

Etape1: Service Generique

- Sous le dossier Interfaces du projet AM.ApplicationCore, créer l'interface générique « IService ».
- Sous le dossier Services du projet AM.ApplicationCore, créer la classe Service générique qui implémente l'interface IService.

PART 17 —SERVICE SPÉCIFIQUE:

Etape2: Service Spécifique

- Modifier toutes les interfaces de Services Spécifiques, **IServicePlane** afin qu'ils héritent de l'interface du service générique « **IService** ».
- Modifier toutes les classes de Services Spécifiques, **ServicePlane** et **ServiceFlight**, afin qu'ils héritent de la classe du service générique « **Service** ».

```
namespace AM.ApplicationCore.Services
{
    public class ServicePlane : Service<Plane>, IServicePlane
    {
        private readonly IUnitOfWork _unitOfWork ;
        public ServicePlane(IUnitOfWork unitOfWork) : base(unitOfWork)
        {
            _unitOfWork = unitOfWork ;
        }
    }
}
```

PARTIE 18 : UTILISATION DE LA COUCHE DE SERVICE SPECIFIQUE

Dans la classe `ServicePlane`, implémenter les services suivants :

- Le service **GetPassenger** qui retourne les voyageurs d'un avion passé en paramètre.
 - Tester et Afficher le résultat de cette fonction dans le programme Console
- Le service **GetFlights** qui retourne les vols ordonnés par date de départ des n derniers avions, groupe par `IdPlane`.
 - Tester et Afficher le résultat de cette fonction dans le programme Console

PARTIE 18 : UTILISATION DE LA COUCHE DE SERVICE SPECIFIQUE

Dans la classe `ServicePlane`, implémenter les services suivants :

- Le service **IsAvailablePlane** qui retourne true si on peut réserver n place à un vol passé en paramètre.
 - Tester et Afficher le résultat de cette fonction dans le programme Console
- Le service **DeletePlanes** qui supprime tous les avions dont la date de fabrication a dépassé 10 ans
 - Tester et Afficher le résultat de cette fonction dans le programme Console

PART 18 — SERVICE GÉNÉRIQUE ET SERVICE SPÉCIFIQUE:

- Ajouter les interfaces et les classe de Service spécifique pour tous les entités de l'application
- Créer la fonction InsertData () qui permet de :
 - Insérer des « Planes » et des Flights dans la base de donnée a partir de l'application Console
 - Insérer 3 Passengers a partir de l'application Console
 - Insérer 3 Tickets qui sont lie aux Passenger et Flight