**TP AirportManagement**

**Partie 6: Couche services – Patrons de conceptions**

**6.1. Implémentation du patron Repository**

* 1. Sous le dossier **Interfaces**, ajouter une interface générique nommée **IGenericRepository**. Cette dernière contient toutes les méthodes CRUD.

public interface IGenericRepository<TEntity> where TEntity : class

{

void Add(TEntity entity);

void Update(TEntity entity);

void Delete(TEntity entity);

void Delete(Expression<Func<TEntity, bool>> where);

TEntity GetById(params object[] keyValues);

TEntity Get(Expression<Func<TEntity, bool>> where);

IEnumerable<TEntity> GetAll();

IEnumerable<TEntity> GetMany(Expression<Func<TEntity, bool>> where);

}

* 1. Sous le projet **AM.Infrastructure**, créer une classe nommée **GenericRepository.** Cette dernière implémente l’interface **IGenericRepository**.

public class GenericRepository<T> :IGenericRepository<T> where T : class

{

private DbContext context;

private DbSet<T> dbSet;

public GenericRepository(DbContext ctx)

{

context = ctx;

dbSet = context.Set<T>();

}

public void Add(T entity)

{

dbSet.Add(entity);

}

public void Delete(T entity)

{

dbSet.Remove(entity);

}

public void Delete(Expression<Func<T,bool>> where)

{

dbSet.RemoveRange(dbSet.Where(where));

}

public T Get(Expression<Func<T, bool>> where)

{

return

dbSet.Where(where).FirstOrDefault();

}

public IEnumerable<T> GetAll()

{

return dbSet.AsEnumerable();

}

public T GetById(params object[] keyValues)

{

return dbSet.Find(keyValues);

}

public IEnumerable<T> GetMany(Expression<Func<T, bool>> where)

{

if (where != null)

return dbSet.Where(where);

else

return dbSet.AsEnumerable(); }

public void Update(T entity)

{

dbSet.Update(entity);

}}

* 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 **Remove()** qui permet de supprimer de la BD un avion.
* Une méthode **GetAll()** qui permet de retourner tous les avions enregistrés dans la BD.

public interface IServicePlane

{

public void Add(Plane plane);

public void Remove(Plane plane);

public IEnumerable<Plane> GetAll();

}

* + - Ajouter à l’interface **IServicePassenger** les trois méthodes précédentes.

public interface IServicePassenger

{

public void Add(Passenger passenger);

public void Remove(Passenger passenger);

public IEnumerable<Passenger> GetAll();

}

* 1. Sous le dossier **Services**,
     + Créer la classe **ServicePlane** qui implémente l’interface **IServicePlane** et qui prend en paramètre du constructeur un objet de type **IRepositoryGeneric**.

public class ServicePlane : IServicePlane

{

private IGenericRepository<Plane> repo;

public ServicePlane(IGenericRepository<Plane> repository)

{

repo = repository;

}

public void Add(Plane plane)

{

repo.Add(plane);

}

public IEnumerable<Plane> GetAll()

{

return repo.GetAll();

}

public void Remove(Plane plane)

{

repo.Delete(plane);

}

}

* + - Modifier la classe **ServicePassenger** afin d’implémenter l’interface **IServicePassenger en** prenant en constructeur un objet de type **IRepositoryGeneric**.

public class ServicePassenger : IServicePassenger

{

private IGenericRepository<Passenger> repo;

public ServicePassenger(IGenericRepository<Passenger> repository)

{

repo = repository;

}

public void Add(Passenger passenger)

{

repo.Add(passenger);

}

public IEnumerable<Passenger> GetAll()

{

return repo.GetAll();

}

public void Remove(Passenger passenger)

{

repo.Delete(passenger);

}

}

* 1. Que constatez-vous ?

Pas de persistance de données.

Impossible d’utiliser la classe du contexte dans le projet **AM.ApplicationCore.**

**Solution** : Utilisation du parton UnitOfWork

**6.2. Implémentation du patron UnitOfWork**

* 1. Sous le dossier **Interfaces**, ajouter une interface nommée **IUnitOfWork** contenant une **propriété** de type DbContext et une méthode **Save()**.

public interface IUnitOfWork

{

int Save();

IGenericRepository<TEntity> Repository<TEntity>() where TEntity : class;

}

* 1. Dans le projet **AM.Infrastructure**, créer une classe nommée **UnitOfWork** qui implémente l’interface **IUnitOfWork**.

public class UnitOfWork : IUnitOfWork

{

private DbContext context;

public UnitOfWork(DbContext ctx)

{

context = ctx;

}

public IGenericRepository<T> Repository<T>() where T : class

{

return new GenericRepository<T>(context);

}

public int Save()

{

return context.SaveChanges();

}

}

* 1. Modifier les deux classes **ServicePlane** et **ServiceFlight** afin d’utiliser un objet de type **IUnitOfWork**.

public class ServicePlane : IServicePlane

{

private IUnitOfWork uow;

public ServicePlane(IUnitOfWork unitOfWork)

{

uow = unitOfWork;

}

public void Add(Plane plane)

{

uow.Repository<Plane>().Add(plane);

}

public IEnumerable<Plane> GetAll()

{

return uow.Repository<Plane>().GetAll();

}

public void Remove(Plane plane)

{

uow.Repository<Plane>().Delete(plane);

}

}

public class ServicePassenger : IServicePassenger

{

private IUnitOfWork uow;

public ServicePassenger(IUnitOfWork unitOfWork)

{

uow = unitOfWork;

}

public void Add(Passenger passenger)

{

uow.Repository<Passenger>().Add(passenger);

}

public IEnumerable<Passenger> GetAll()

{

return uow.Repository<Passenger>().GetAll();

}

public void Remove(Passenger passenger)

{

uow.Repository<Passenger>().Delete(passenger);

}

}

* 1. Que constatez-vous ?

Le garbage collector (GC) ne libère pas les objets de type DbContext. Ce dernier est un objet non géré par GC.

**Solution** : Utilisation de l’interface **IDisposable**

* 1. Modifier l’interface **IUnitOfWork** afin qu’elle hérite de l'interface **IDisposable**.

public interface IUnitOfWork:IDisposable

{

int Save();

IGenericRepository<TEntity> Repository<TEntity>() where TEntity : class;

}

* 1. Modifier la classe **UnitOfWork** en implémenter la méthode Dispose (bool disposing). Cette méthode permet de libérer l’objet du **contexte**.

private bool disposedValue;

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (disposing)

{

context.Dispose();

}

disposedValue = true;

}

}

public void Dispose()

{

Dispose(disposing: true);

GC.SuppressFinalize(this);

}