Voir le code complet de tous les Tps réalisés sur : https://github.com/wailhadad/tp_csharp

Table des matières

Rapport sur la solution "Ex_1 – Gestion Des Absences"

Ce projet C# est organisé en plusieurs packages distincts, suivant une architecture bien structurée et favorisant un **couplage faible**. Il semble s'agir d'une application de gestion des absences, avec une séparation claire des responsabilités.

Structure Du Projet :

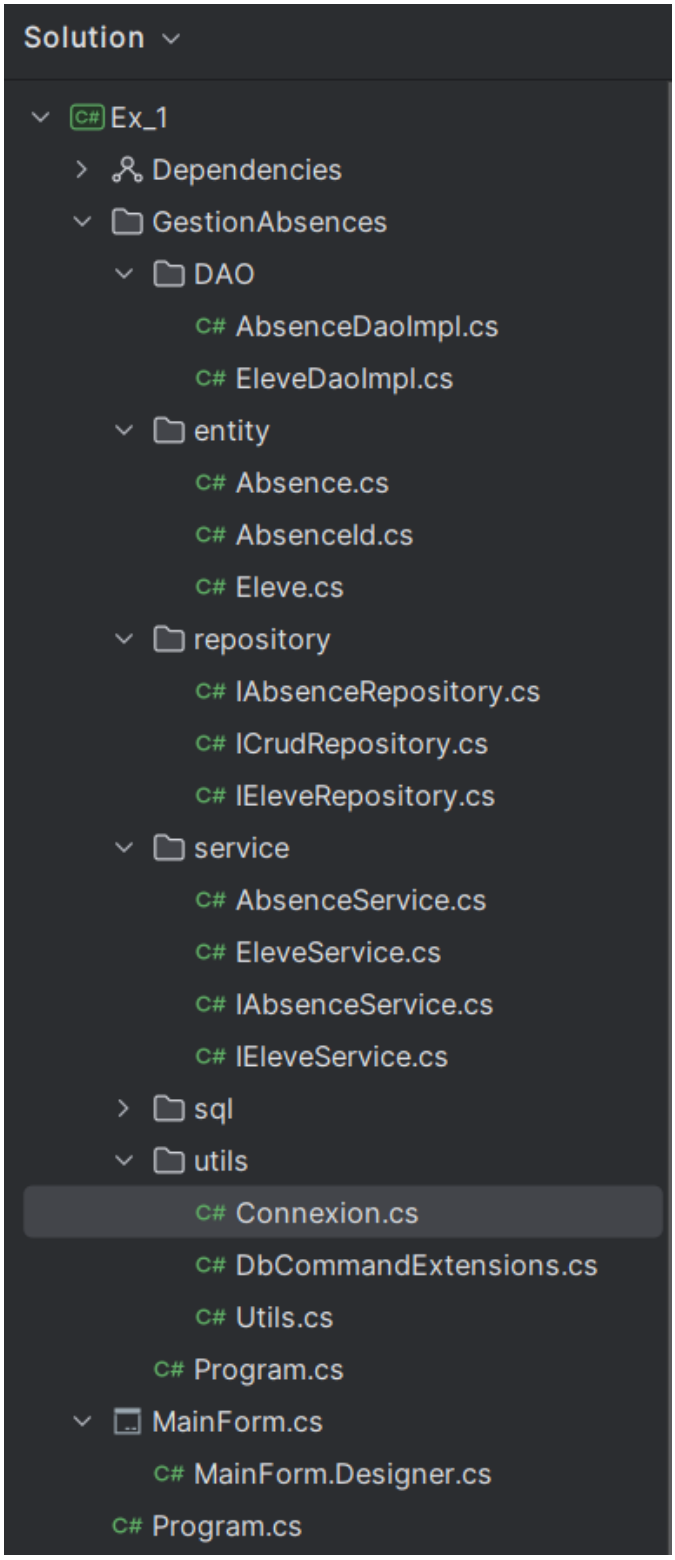
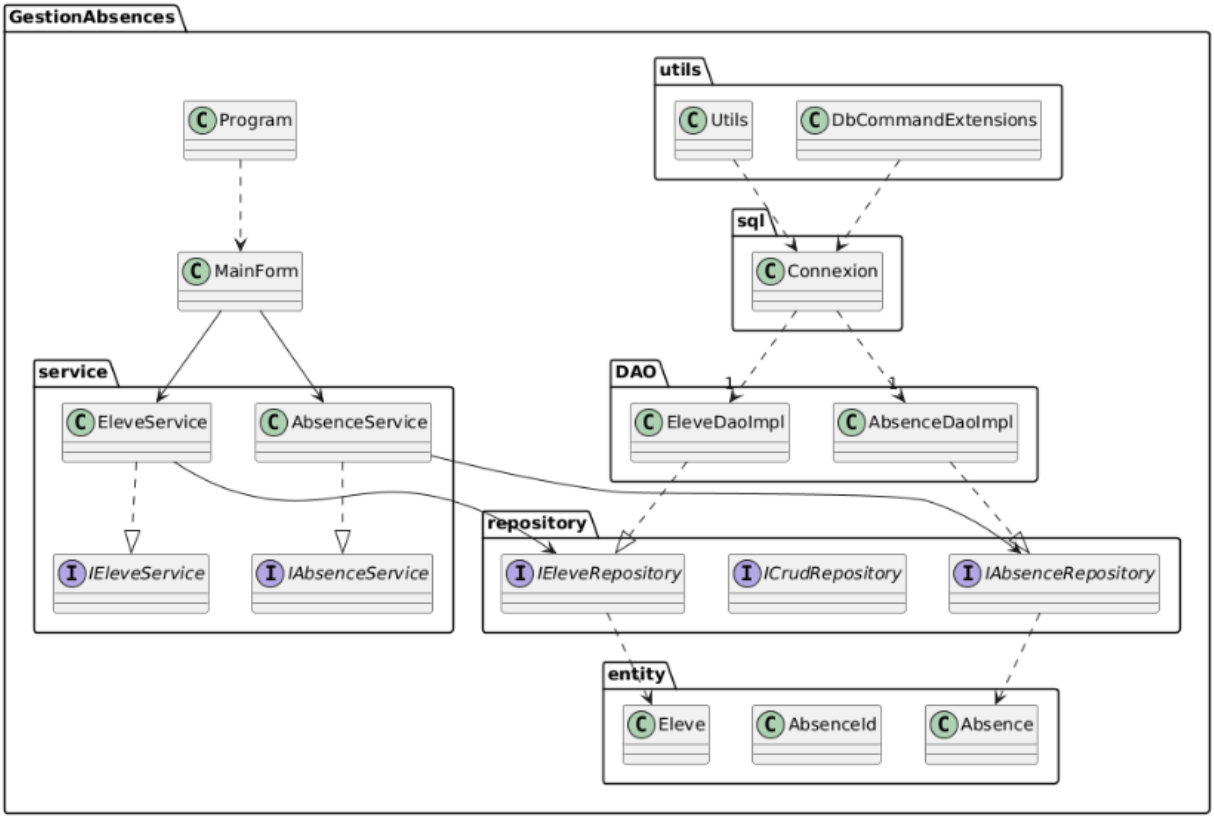


Diagramme De package :



1. Analyse des packages et fichiers

DAO (Data Access Object)

- **AbsenceDaoImpl.cs & EleveDaoImpl.cs** :
    - Contiennent probablement l'implémentation de la logique d'accès aux bases de données pour les absences et les élèves.
    - Ils permettent d'effectuer des requêtes comme l'ajout, la suppression, la modification ou la récupération des données.

Entity

- **Absence.cs, AbsenceId.cs, Eleve.cs** :
    - Représentent les modèles de données.
    - **AbsenceId.cs** pourrait être une classe servant d'identifiant composite pour les absences.

Repository

- **IAbsenceRepository.cs, ICrudRepository.cs, IEleveRepository.cs** :
    - Interfaces définissant les méthodes à implémenter pour gérer les entités de manière générique.
    - **ICrudRepository.cs** suggère une interface commune pour les opérations CRUD (Create, Read, Update, Delete).
    - Cela permet de changer facilement la base de données sans impacter les services qui consomment ces repositories.

Service

- **AbsenceService.cs, EleveService.cs, IAbsenceService.cs, IEleveService.cs** :
    - Contiennent la logique métier.
    - Ils utilisent les repositories pour accéder aux données et appliquer des règles métier.

SQL

- Ce package pourrait contenir des scripts SQL (non affichés ici), utilisés pour la création des tables ou des requêtes spécifiques.

Utils (Utilitaires)

- **Connexion.cs** : Gère la connexion à la base de données.
- **DbCommandExtensions.cs** : Permet d'ajouter facilement des paramètres aux commandes SQL et supporte différentes bases de données.
- **Utils.cs** : Contient une méthode d'extension pour convertir un objet en dictionnaire.

MainForm & Program

- **MainForm.Designer.cs** : Contient la définition de l'interface utilisateur en WinForms.
- **Program.cs** : Point d'entrée principal de l'application.

---

2. Explication des fichiers ambigus

Connexion.cs

Ce fichier est censé gérer la connexion à la base de données. Il permet de centraliser la gestion des connexions, évitant ainsi la duplication du code.

DbCommandExtensions.cs (Fichier récupéré)

- Ce fichier étend la classe **IDbCommand** pour ajouter une méthode **AddParameter** qui facilite l'ajout de paramètres SQL.

- Il gère les différences entre les bases de données :
    - Si la connexion est Oracle, les paramètres sont préfixés par :
    - Sinon, ils restent sous leur forme standard.
- **Avantage** : Permet d'avoir un code générique supportant plusieurs bases de données sans modifications majeures.

Utils.cs (Fichier récupéré)

- Contient une méthode **ToDictionary** qui convertit un objet en dictionnaire clé/valeur.
- Cela peut être utilisé pour sérialiser un objet ou l'afficher dynamiquement dans une interface graphique.

---

3. Couplage faible et avantages

L'organisation en packages apporte un **couplage faible**, car :

1. **Séparation des responsabilités** :
    - La **couche DAO** s'occupe uniquement des accès à la base de données.
    - La **couche Repository** abstrait ces accès avec des interfaces.
    - La **couche Service** contient la logique métier.
    - La **couche UI** (MainForm) est totalement séparée du reste.
2. **Flexibilité** :
    - Si demain, on veut changer de base de données (passer de MySQL à Oracle ou PostgreSQL), on n'a qu'à modifier **Connexion.cs** et **DbCommandExtensions.cs**.
3. **Testabilité** :
    - Grâce aux interfaces (IAbsenceRepository, ICrudRepository, etc.), il est facile de **mock** les bases de données pour les tests unitaires.

---

4. Support multi-SGBD

Le projet est conçu pour être compatible avec plusieurs SGBD :

- **DbCommandExtensions.cs** gère la syntaxe des paramètres entre **Oracle** et d'autres bases.
- **Connexion.cs** (non affiché ici) pourrait être configuré pour s'adapter à différentes bases (MySQL, SQL Server, etc.).
- **ICrudRepository** offre une abstraction qui évite d'écrire du code spécifique à chaque base.

Conclusion

Cette solution est bien architecturée et favorise un **code maintenable, modulaire et évolutif**. Elle pourrait être améliorée en ajoutant des logs et des tests unitaires pour renforcer la robustesse du système.

Vidéo Démonstrative de L'application Windows Forms

« Gestions Des Elèves Et des Absences ».

Voir la vidéo ci-jointe pour la démonstration de l'app :

app_demo.mp4