Application Mediateq
Eléments de qualité logicielle

1. La gestion des exceptions

Dans le cadre de la séparation des tâches, le traitement des exceptions ne doit pas être effectué dans les classe métier ou DAO, mais dans le formulaire (qui nous sert ici de contrôleur). Pour cela, les exceptions captées, particulièrement dans les accès à la base de données, seront « remontées » jusqu'au contrôleur.

1.1 Nouvelle classe pour les exceptions

- Créer une classe *ExceptionSIO*, qui hérite de *Exception*, qui nous servira à instancier les exceptions à gérer. Cette classe contient 2 propriétés : un libellé propre et un niveau de priorité.

```
4 references
class ExceptionSIO : Exception
{
    private int niveauExc;
    private string libelleExc;

    2 références
    public ExceptionSIO(int pNiveau, string pLibelle, string pMessage):base(pMessage)
    {
        niveauExc = pNiveau;
        libelleExc = pLibelle;
    }

    1 référence
    public int NiveauExc { get => niveauExc; set => niveauExc = value; }
    1 référence
    public string LibelleExc { get => libelleExc; set => libelleExc = value; }
}
```

1.2 La capture des exceptions

- Placer tous les accès à la base de données dans des sections *try*. Dans les sections *catch*, capter les exceptions éventuelles (connexion impossible, problème de foreign key, etc…) et remonter une exception *ExceptionSIO*, à instancier avec un libellé clair et un niveau de priorité.

```
public static void connecter()
{
    try
    {
        connexion.Open();
    }
    catch (Exception e)
    {
        throw new ExceptionSIO(2, "problème ouverture connexion BDD", e.Message);
    }
}
```

1.3 La remontée des exceptions

- Les fonctions appelantes seront aussi placées dans un *try*, afin de capter les éventuelle exceptions lancées.
- Dans les couches DAO, faire remonter de la même manière ces exceptions :

```
public static List<Descripteur> getAllGenres()
{
    List<Descripteur> lesGenres = new List<Descripteur>();

    try
    {
        string req = "Select * from descripteur";

        DAOFactory.connecter();

        MySqlDataReader reader = DAOFactory.execSQLRead(req);

        while (reader.Read())
        {
            Descripteur genre = new Descripteur(reader[0].ToString(), reader[1].ToString());
            lesGenres.Add(genre);
        }
        DAOFactory.deconnecter();
    }


    catch (Exception exc)
    {
        throw exc;
    }
    return lesGenres;
}
```

- Dans le contrôleur (ici : le formulaire), prévoir un traitement clair et approprié de l'exception. Dans cet exemple, on se contente d'afficher un pop-up, mais il faut ensuite décider de la suite selon le niveau de l'exception captée.

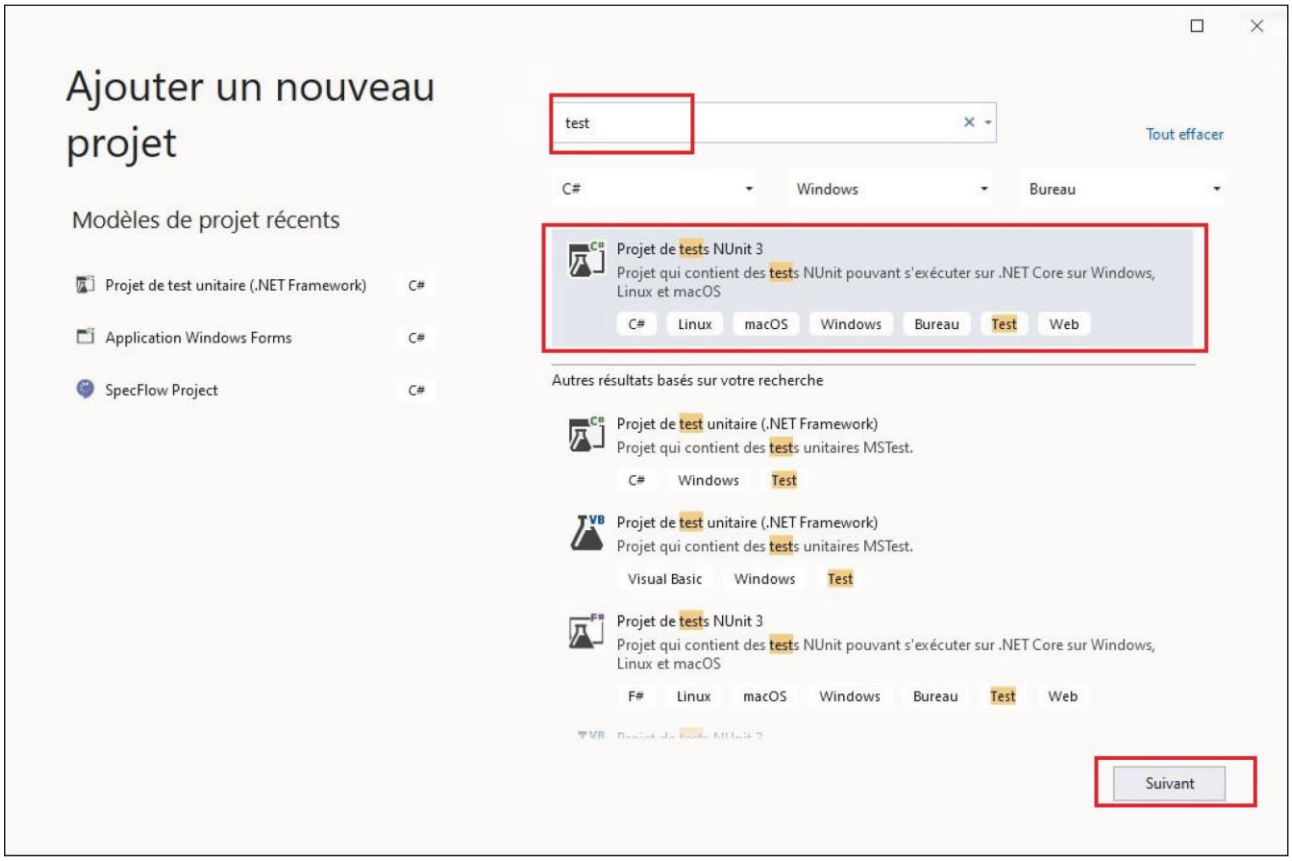
```
private void FrmMediateq_Load(object sender, EventArgs e)
{
    try
    {
        // Création de la connexion avec la base de données
        DAOFactory.creerConnection();

        // Chargement des objets en mémoire
        lesDescripteurs = DAODocuments.getAllGenres();
        lesTitres = DAOPresse.getAllTitre();
    }
    catch (ExceptionSIO exc)
    {
        MessageBox.Show(exc.NiveauExc + " - " + exc.LibelleExc + " - " + exc.Message);
    }

}
```

2. Les tests unitaires

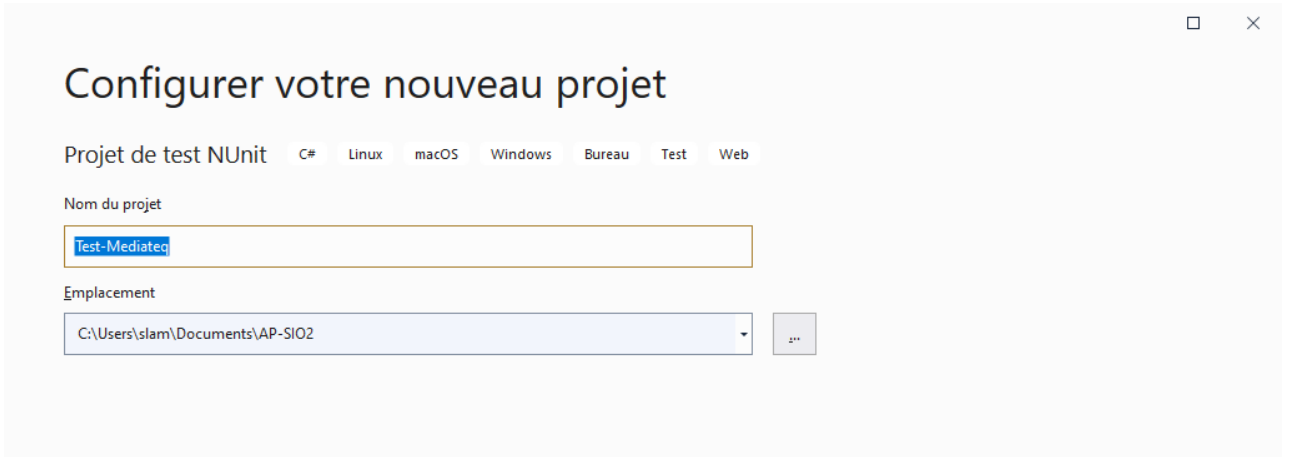
2.1 Créer un projet de test unitaire

Pour tester un projet, il faut ajouter, dans la même solution, un second projet qui est un projet de test, et faire en sorte de lier les deux projets.

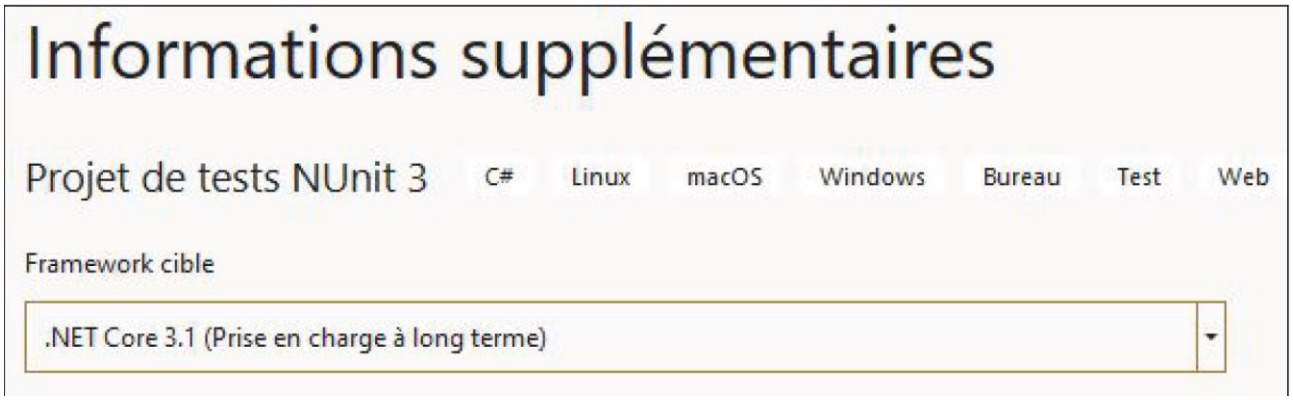
- Ouvrir le projet à tester, ou plutôt la solution (fichier *.sln*) qui contient le projet à tester.
- Dans "explorateur de solutions", faire un clic droit sur la solution, "Ajouter > nouveau projet".
- Dans la boite de dialogue "Ajouter un nouveau projet", dans la zone de recherche, taper "test" et sélectionner "Projet de tests NUnit" :



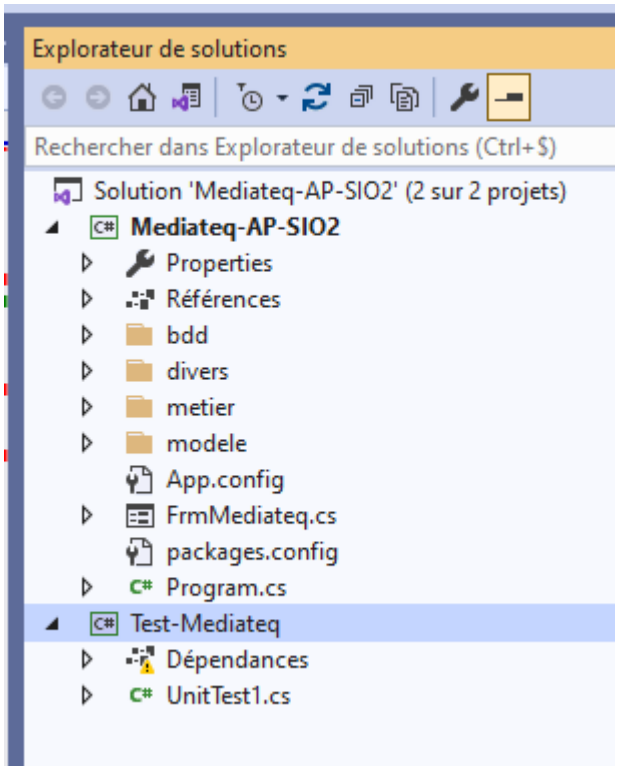
- Cliquer sur "Suivant". À l'étape suivante, donner un nom (par exemple "Test" suivi du nom du projet à tester), garder l'emplacement proposé (qui est celui de la solution du projet à tester) :



- Cliquer sur "Suivant". À l'étape "Informations supplémentaires", gardez le Framework par défaut qui est celui de la solution :



- Cliquer sur "Créer". Dans l'explorateur de solutions, un second projet s'est ajouté, en plus du projet à tester :



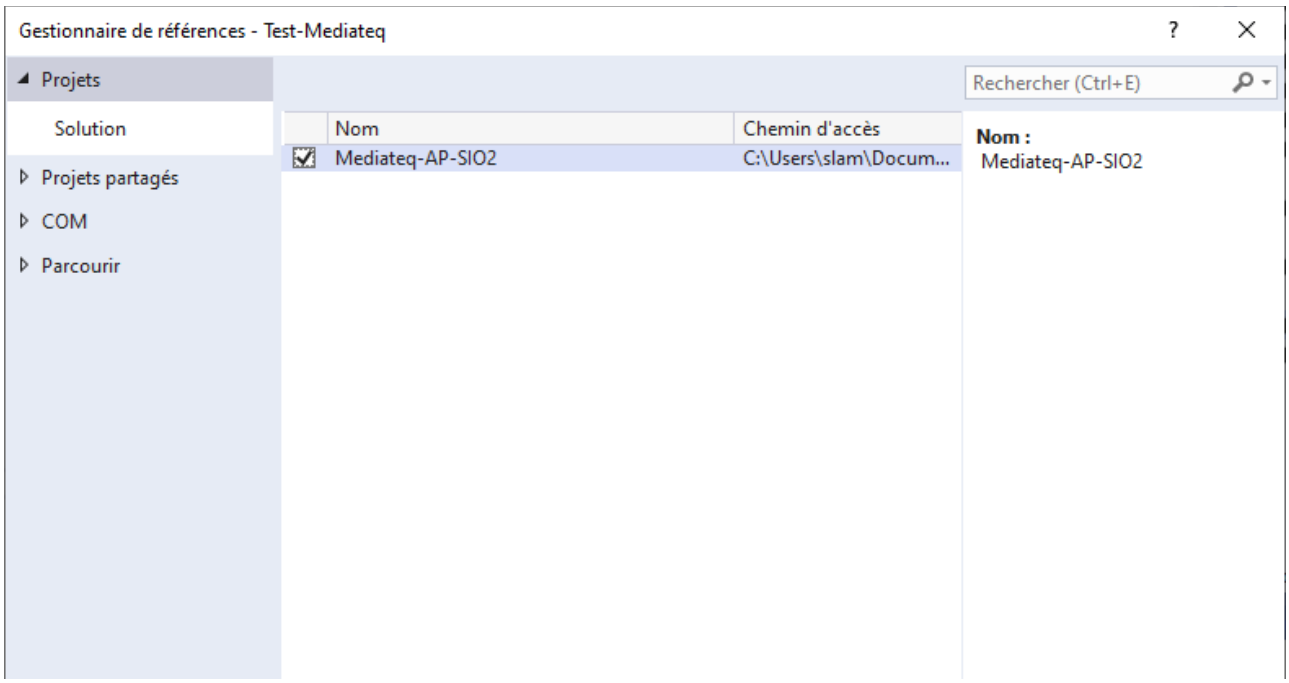
Ce projet contient pour le moment, entre autres, le fichier "UnitTest1.cs", qui a été automatiquement ouvert et qui contient le code suivant :

```
using NUnit.Framework;
namespace TestFormes
{
    public class Tests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void Test1()
        {
            Assert.Pass();
        }
    }
}
```

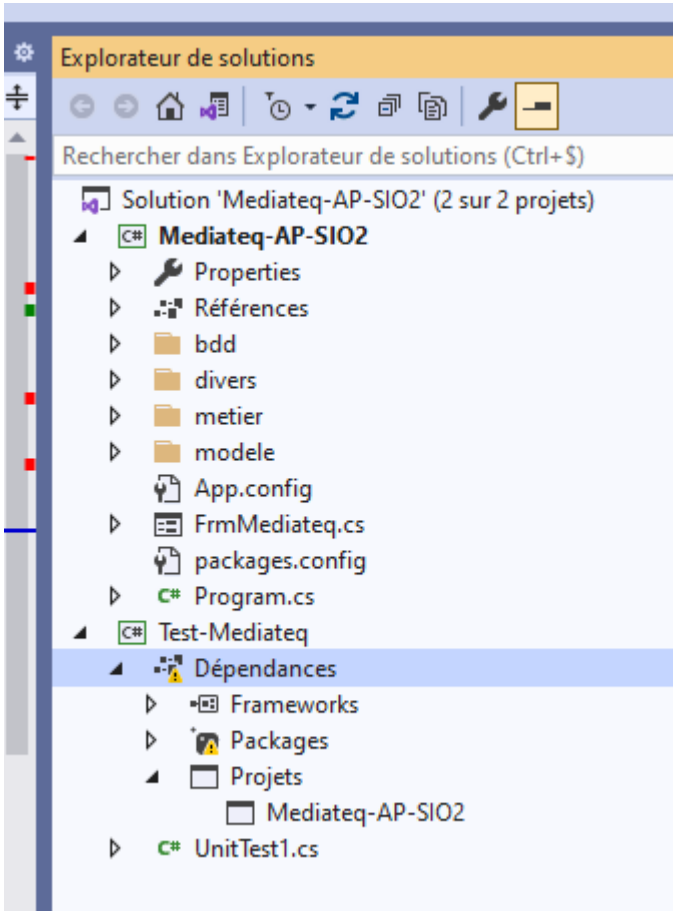
2.2 Lier le projet à tester avec le projet de test unitaire

Pour que le projet à tester soit lié au nouveau projet de test, il faut ajouter le premier comme référence du second.

- Faire un clic droit sur "Dépendances" du nouveau projet "Test" et choisir "Ajouter une référence".
- Dans la fenêtre "Gestionnaire de référence", cocher le projet à tester puis "OK" :



- Si on déroule 'Dépendances', puis 'Projets', on voit que le projet à tester s'est ajouté :



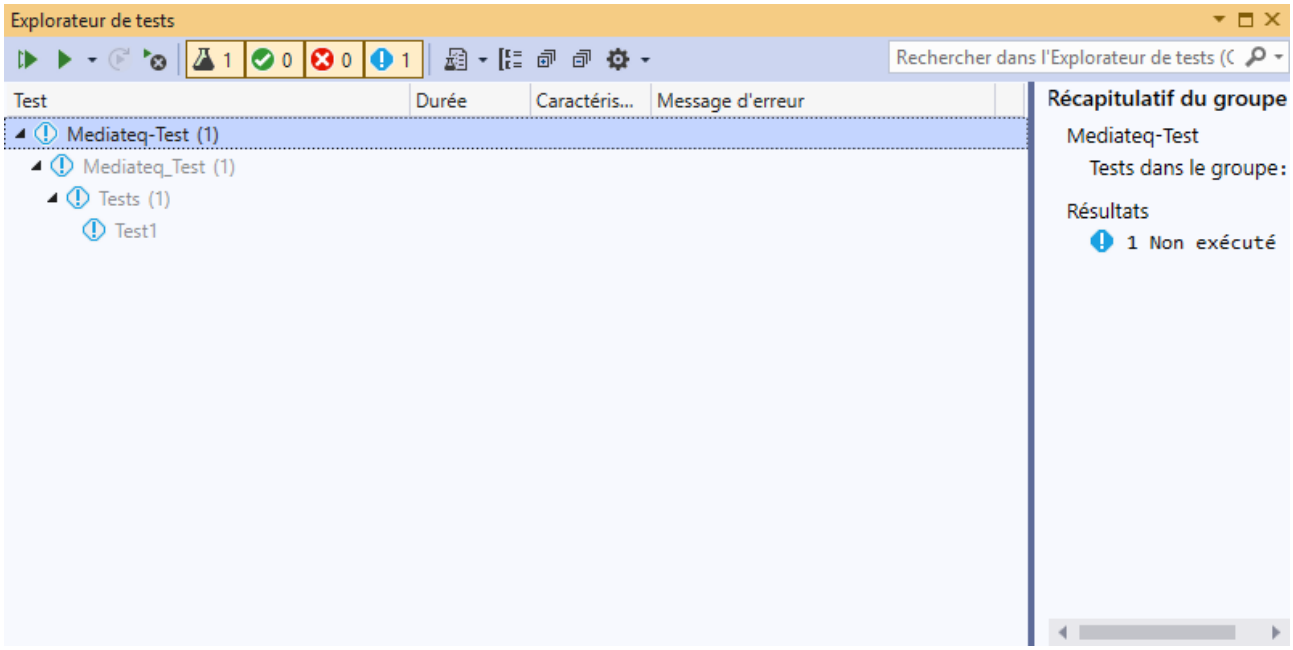
2.3 Créer un test unitaire

- Dans la méthode 'Test1', il est possible d'écrire un premier test unitaire. Par exemple, on va simplement instancier un objet et vérifier qu'un attribut s'est bien valorisé :

```
public void Test1()
{
     //Assert.Pass();
     Revue revue = new Revue("123", "GEO", 'o', "mensuel", new DateTime(2022,
10, 6), 3, "00002");
     Assert.AreEqual(3, revue.DelaiMiseADispo);
}
```

- Pour exécuter le test, commencer par le générer la solution (menu "Générer > générer la solution").

- Puis aller dans le menu "Test > Explorateur de tests". Une fenêtre s'ouvre qui va permettre de gérer les tests. En déroulant le test, on voit la seule méthode actuellement écrite : 'Test1'.

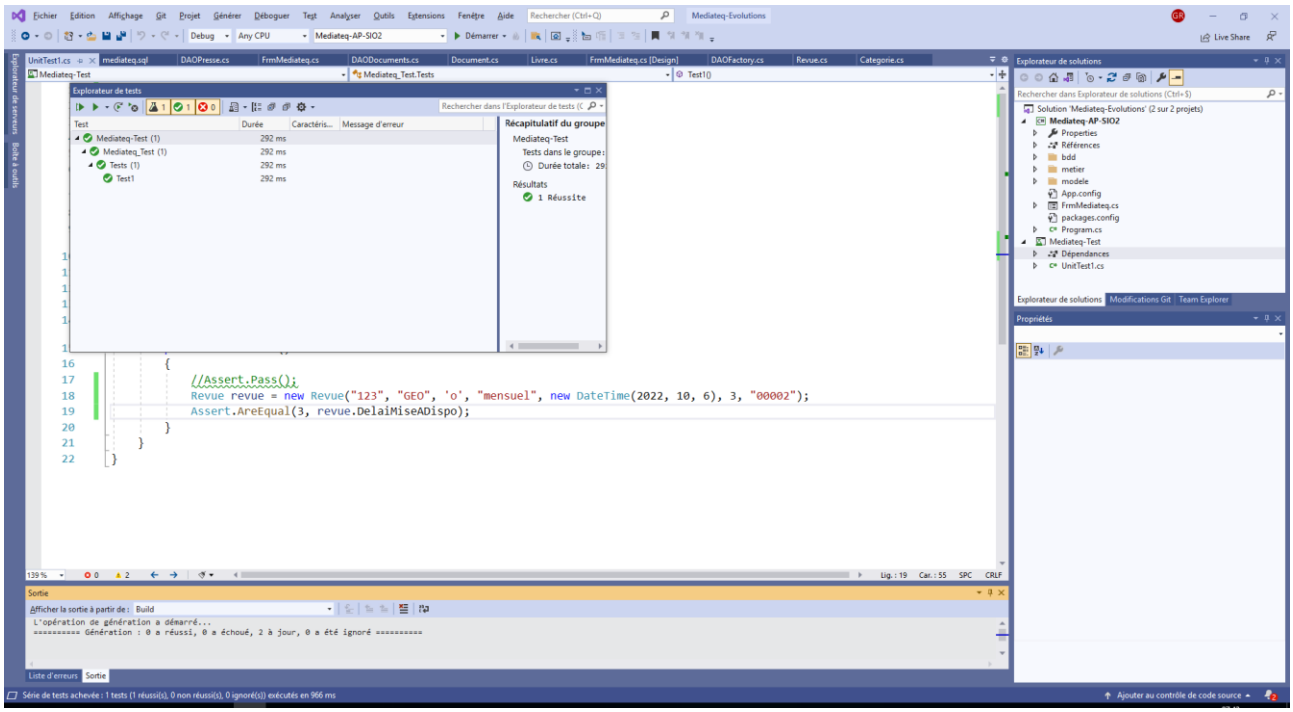


En haut à gauche de cette fenêtre se trouvent 2 flèches vertes :

— celle totalement à gauche, porte la mention "Exécuter tous les tests dans la vue" ;

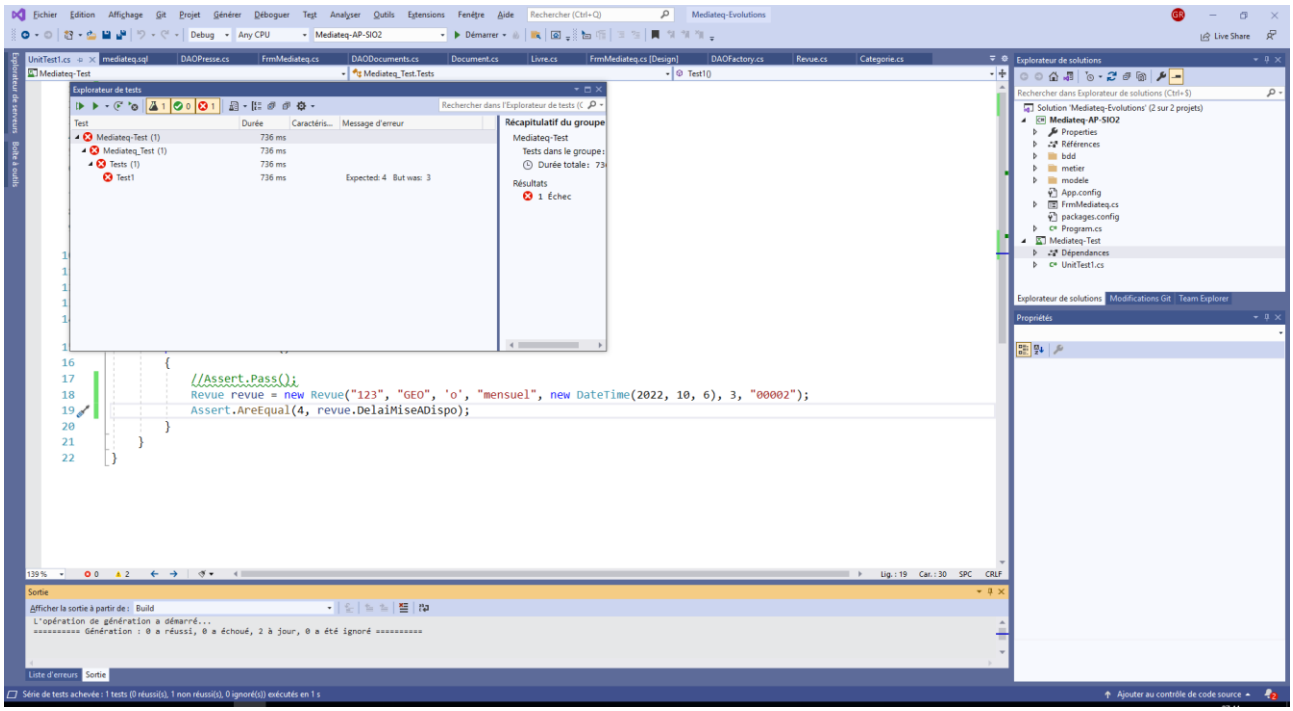
— celle qui est en second, porte la mention "Exécuter" qui permet d'exécuter un seul test, mais qui contient aussi un sous menu avec plusieurs possibilités.

➢ Vu qu'ici il n'y a pour le moment qu'un seul test, l'un ou l'autre donnera le même résultat. Cliquer sur "Exécuter". Au bout de quelques instants, on obtient le résultat. Si le test a réussi, les coches vertes apparaissent :



Le test peut échouer dans 2 cas : le code de la méthode à tester n'est pas correct, ou le test lui-même utilise une valeur incorrecte. Évidemment, dans la réalité, c'est normalement le code de la méthode à tester qui pose problème. Ici, on va juste modifier la valeur dans le test, en mettant 4 à la place de 3.

- Faire cette modification et relancer le test. Le résultat suivant est obtenu :



Cette fois, des croix rouges signifient que le test a échoué.

Le message d'erreur est très explicite : "Expected: 4 But was: 3".
Avec ce genre de test, il est donc facile de trouver l'origine de l'erreur dans le code.

3. Un outil de qualité de code

- Installer l'outil SonarLint dans l'IDE Visual Studio.

Cet outil donne en continu des informations, conseils, préconisations, etc.. permettant d'améliorer la qualité du code. La fenêtre de sortie permet de visualiser ces informations :