Compte-rendu

TÂCHE 1 : CRÉATION DE LA CLASSE D'ACCES AUX DONNÉES

Dans la nouvelle application, une classe d'accès aux données doit être créée, permettant les fonctionnalités classiques d'accès aux données. Pour connexion à la base MySQL, j'ai décidé de créer un singleton dont le but est de créer qu'une seule connexion à la fois pour exécuter nos requêtes.

Grâce à l'import de la bibliothèque MySql.Data.MySqlClient, j'ai créé une méthode de connexion à la base de données et une autre qui demande la connexion. S'il n'y a pas encore de connexions créée, j'appelle ma première méthode mais si une connexion est déjà ouverte, alors je retourne cette dernière.

```
/**
 * Fonction statique qui crée l'unique instance de la classe
 *
 * @return l'unique objet de la classe Connection
 */
2 references
public static Connection getPdoGsb()
{
    if (Connection.monPdoGsb == null)
    {
        Connection.monPdoGsb = new Connection();
    }
    return Connection.monPdoGsb;
}
```

Pour les requêtes, la classe MySqlDataReader me permet d'en exécuter avec une chaine de caractères mais le problème est que pour pouvoir lire les réponses retournées par la base de données, je dois ouvrir et absolument fermer le lecteur de réponse. Je ne peux donc pas retourner directement la réponse car après le return rien ne peut s'exécuter et avec le lecteur fermé, je n'ai plus accès à mes réponses, alors j'ai créé un tableau 2D qui se compose de String dans une Liste qui est elle-même dans une Liste.

```
List<List<string>> tableau = new List<List<string>>();
```

Les String sont bien sûr les éléments du SELECT alors je fais une boucle ligne par ligne, colonne par colonne pour remplir mon tableau que je retourne à la fin, sans oublier de fermer mon Reader.

```
while (reader.Read())
{
    List<string> ligne = new List<string>();
    for (int i = 0; i < reader.FieldCount; i++)
    {
        ligne.Add(reader[i].ToString());
    }
    tableau.Add(ligne);
}
```

Pour la lecture de ces données, je n'ai juste qu'à boucler sur les Listes pour afficher le résultat dans le terminal :

```
PS C:\Users\LENOVO\Documents\Mission2> dotnet run
78 202002 CL
a17 201912 CL
a55 202001 CL
78 202003 CR
a17 202001 MP
78 202001 VA
```

Comme à aucun moment dans l'application, un SELECT n'est effectué, cette méthode restera inutile mais pour les requêtes d'update, une 2ème méthode est créée car celle-ci ne retournera pas de réponse.

```
public void sendNonQuery(String req)
{
    monPdo.Open();
    var cmd = new MySqlCommand(req, monPdo);
    cmd.ExecuteNonQuery();

    monPdo.Close();
}
```

Nous pouvons donc maintenant utiliser une (et une seule) connexion pour interroger la base de données.

TÂCHE 1 : CRÉATION D'UNE CLASSE DE GESTION DE DATES

Pour pouvoir savoir quelle est la date du mois précédant ou suivant, je créé une classe dédiée aux calculs de date. Chaque méthode a une surcharge car s'il n'y a pas de paramètre date. La méthode s'exécutera avec la date courante. Comme demandé, j'utilise la classe DateTime qui me permettra d'avoir accès à la fonction addMonth() qui même me permet d'avoir le mois suivant mais aussi le mois précédant si le paramètre est négatif.

Pour la méthode entre(), il y a des simples conditions, avec la vérification que la date minimum ne soit pas la même que la date maximum car sinon mes conditions sont biaisées. D'ailleurs c'est grâce aux tests unitaires que j'ai pu découvrir cet oubli.

```
9 references
public static Boolean entre(int date1, int date2,DateTime date){
    DateTime actu = date;
    if(date1==date2){return actu.Day == date1;}
    else if(date1<date2){return actu.Day>=date1 && actu.Day<=date2;}
    else{return actu.Day>=date2 && actu.Day<=date1;}
}
```

Les tests sont réalisés avec Xunit et sont regroupés par nom de méthode (donc en 3 fonctions de tests).
Pour les tests de méthode qui utilisent la date courante (DateTime.Now), il faudra revenir modifier quelques lignes de code car le test est adapté au jour courant du test.

TÂCHE 3 : CRÉATION DE L'APPLICATION

Pour cette dernière partie de mission, je vais donc réutiliser les deux classes précédemment créées. Le premier jour du mois je clôture toutes les fiches de frais du mois précédant de la base de données. Et le 20, je mets les fiches en état de remboursement.

```
if (today.ToString("dd").Equals("01"))
```

Une simple condition pour que je puisse effectuer ma requête SQL qu'une seule fois par mois.

```
var requete = "UPDATE fichefrais SET idetat = 'CL' WHERE mois = '" + today.ToS
tring("yyyy") + gestionDate.getMoisPrecedent() + "' AND idetat = 'CR' ";
```

La concaténation de ma requête avec l'aide de ma classe GestionDate pour pouvoir avoir le mois précédant et ensuite l'exécution avec mon singleton de connexion.

```
maConnection.sendNonQuery(requete);
```

J'effectue le même procédé pour le 20 du mois et la mission est finie.