***Cette partie 2 est pour les étudiants plus rapides…***

## 1- Utilisation d'une procédure stockée

A chaque fois qu'un virement est effectué, il sera enregistré dans la table `Virement` grâce à l'exécution d'une procédure stockée.

***BONNE PRATIQUE :*** *L'utilisation de procédures ou fonctions stockées empêche l'injection SQL. En outre, elles peuvent intégrer une gestion d'erreur. On peut aussi utiliser des requêtes paramétrées à la place des procédures/fonctions stockées : [https://webman.developpez.com/articles/aspnet/sqlparameter/csharp/](https://webman.developpez.com/articles/aspnet/sqlparameter/csharp/)*

Etapes à suivre :

- Exécuter le code de la procédure stockée du fichier `Procedure stockee Virement PostgreSQL.sql` dans la base de données.
  Regarder le code. Cette fonction réalise la mise à jour du solde de chaque compte bancaire puis insère une ligne dans la table `virement`. Ce code est du PL/PgSQL que vous verrez en R3.07.
- Mettre en commentaire dans la méthode `Virement` de `ServiceCompte` le code d'appel à la méthode `SetData` ainsi qu'éventuellement la chaine de code SQL passée à cette méthode.
- Code de l'appel de la fonction stockée (à ajouter dans le fichier `DataAccess.cs`) :

```
/// <summary>
/// Exécution d'une PS
/// </summary>
/// <param name="idCompteDebit">ID du compte à débiter</param>
/// <param name="idCompteCredit">ID du compte à créditer</param>
/// <param name="montant">Montant du virement</param>
/// <exception cref="DataBaseException">Exception levée si PS en erreur</exception>
public void VirementBancaire(int idCompteDebit, int idCompteCredit, double montant)
{
   try
   {
      OpenConnection();

      //Création d'une commande de type procédure stockée
      NpgsqlCommand npgsqlCommand = new NpgsqlCommand("sp_virement_append", NpgSQLConnect)
      {
         CommandType = CommandType.StoredProcedure
      };

      //Création des paramètres de la procédure stockée
      NpgsqlParameter paramIdCompteDebit = npgsqlCommand.Parameters.Add("pidcomptedebit",
NpgsqlTypes.NpgsqlDbType.Integer);
      paramIdCompteDebit.Direction = ParameterDirection.Input;
      NpgsqlParameter paramIdCompteCredit = npgsqlCommand.Parameters.Add("pidcomptecredit",
NpgsqlTypes.NpgsqlDbType.Integer);
      paramIdCompteCredit.Direction = ParameterDirection.Input;
      NpgsqlParameter paramMontant = npgsqlCommand.Parameters.Add("pmontant", NpgsqlTypes.NpgsqlDbType.Numeric);
      paramMontant.Direction = ParameterDirection.Input;

      //Initialisation de la valeur des paramètres aux arguments de la méthode
      paramIdCompteDebit.Value = idCompteDebit;
      paramIdCompteCredit.Value = idCompteCredit;
      paramMontant.Value = montant;

      //Exécution de la procédure stockée
      npgsqlCommand.ExecuteNonQuery();

      CloseConnection();
   }
   catch (Exception ex)
```

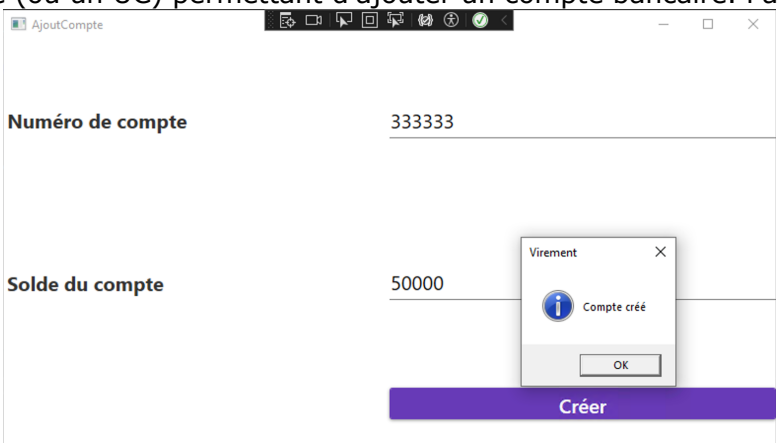
```
        {
            CloseConnection();
            string message = "Erreur de base de données. Impossible de créer le virement.";
#if DEBUG
            MethodBase? m = MethodBase.GetCurrentMethod();
            throw new DataBaseException(message + " L'erreur provient de la classe " + this.GetType().Name + " / Méthode : " + m!.Name +
".\n"
                + ex.Message);
#else
            throw new DataBaseException(message);
#endif
        }
    }
```

*Remarque : On est obligé de créer les paramètres (NpgsqlParameter) qui seront passés à la procédure stockée. Il n'est donc pas possible de réaliser un code générique dans le cas de l'appel de procédures stockées.*

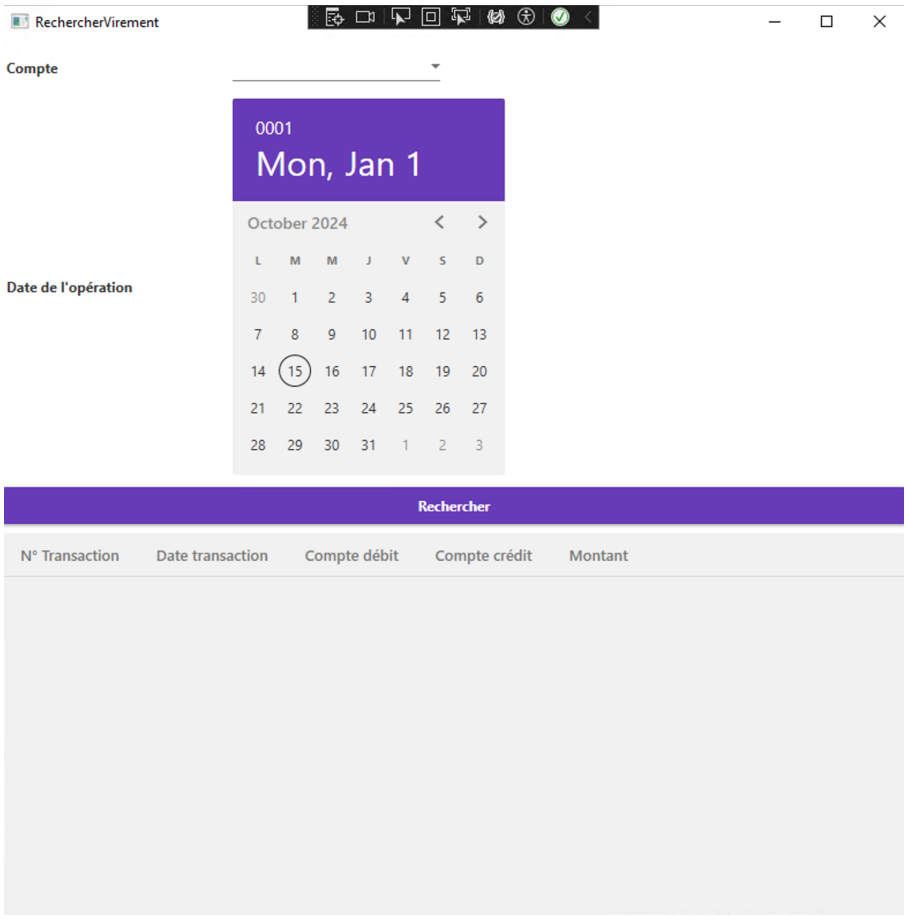
- Créer le code dans la méthode `Virement` de `ServiceCompte` appelant la méthode `VirementBancaire` de `DataAccess.cs`.

## 2- Ajouts fonctionnels

1. Ajouter une fenêtre (ou un UC) permettant d'ajouter un compte bancaire. Par exemple :

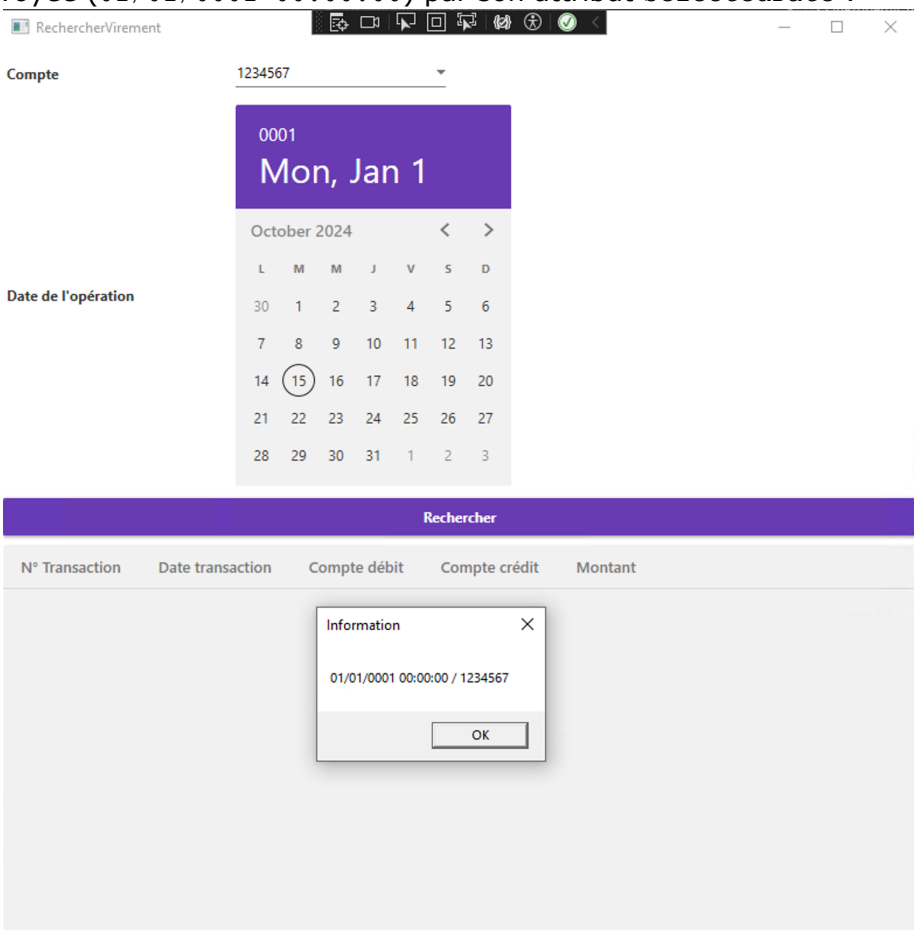



2. Ajouter une fenêtre (ou un UC) permettant de rechercher les virements (utiliser un *DataGrid* pour afficher les opérations) en fonction d'une date et/ou d'un compte.




Indications :

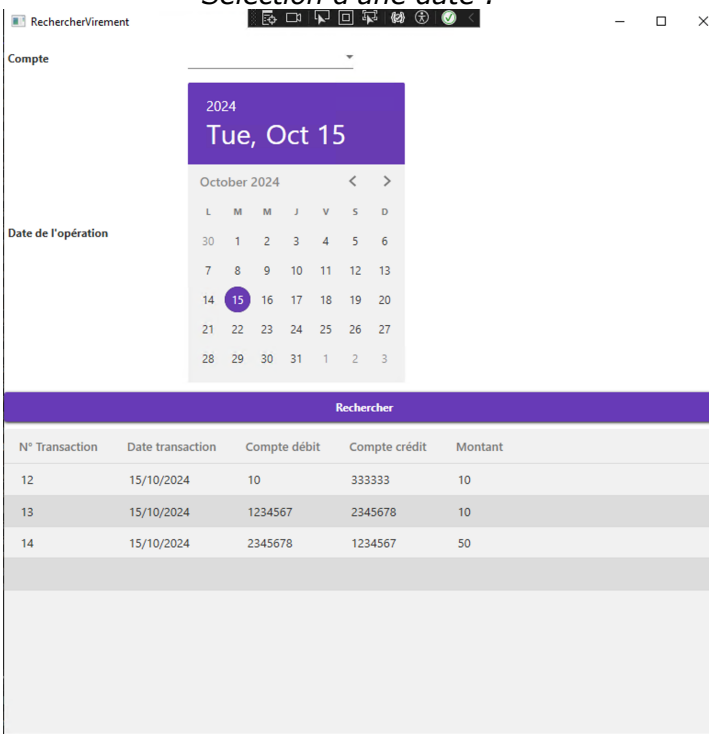
- Utiliser le binding de données.
- Ajouter une méthode de recherche des opérations dans `ServiceCompte` avec comme paramètres le compte recherché et la date (de type `DateTime`).
- Si vous utilisez un contrôle `Calendar` et ne sélectionnez pas de date, une date sera tout de même renvoyée (`01/01/0001 00:00:00`) par son attribut `SelectedDate` :



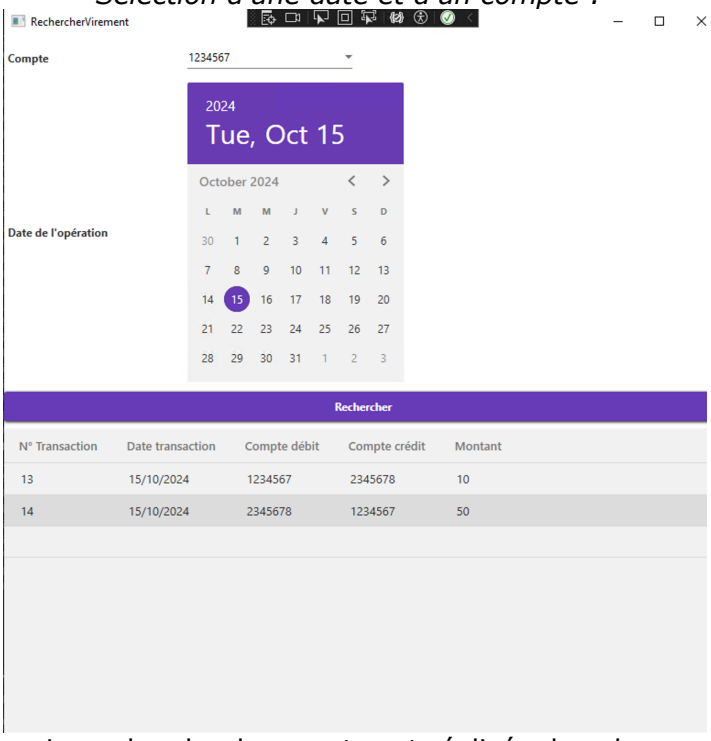
Vous pouvez aussi fixer cette date à la date du jour par défaut.

- Pour ne récupérer que la date d'un `DateTime`, vous pouvez utiliser la méthode `ToShortDateString()`.
- Exemples d'affichage :

*Sélection d'une date :*

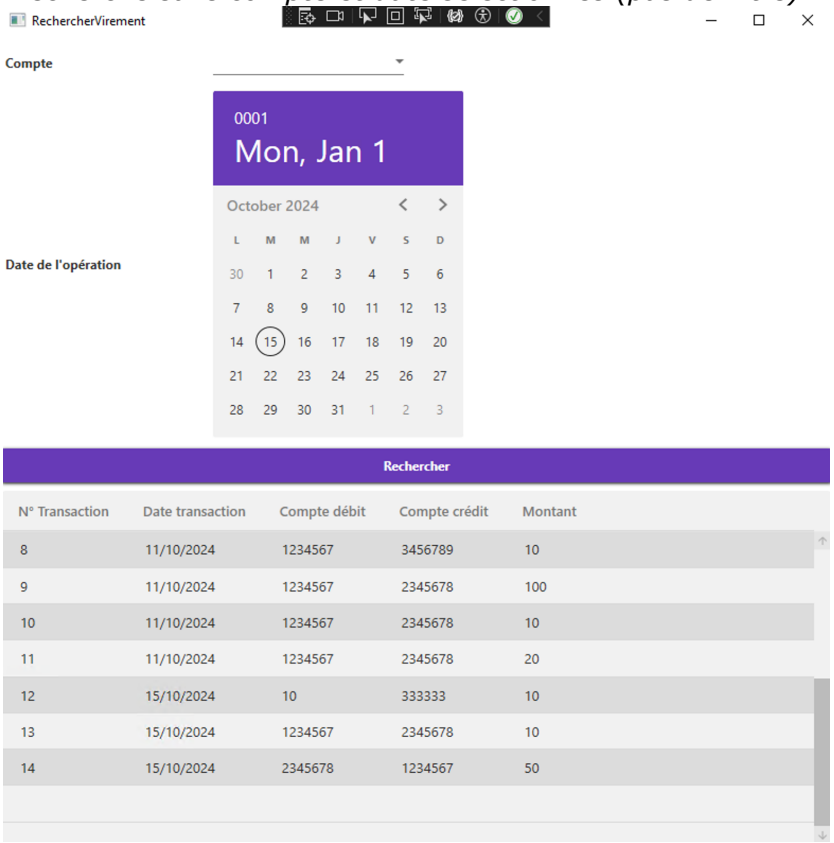



*Sélection d'une date et d'un compte :*




La recherche du compte est réalisée dans les comptes de crédit et de débit

*Recherche sans compte et date sélectionnés (pas de filtre) :*




- Dans les affichages précédents nous avons spécifié les colonnes du DataGrid afin de fixer le format d'affichage de la date. Nous ne sommes cependant pas obligés…

```
<DataGrid x:Name="DataGridOperations" Grid.Row="3" CanUserReorderColumns="True" CanUserResizeColumns="True"
 CanUserResizeRows="False" CanUserSortColumns="True"
      ItemsSource="{Binding Virements}" AutoGenerateColumns="False"
      AlternatingRowBackground="Gainsboro" AlternationCount="2">
  <DataGrid.Columns>
```

```
    …
    <DataGridTextColumn Header="Date transaction" IsReadOnly="True" Binding="{Binding DateTransaction,
StringFormat=\{0:dd/MM/yyyy\}}"/>
    …
  </DataGrid.Columns>
</DataGrid>
```

- Amélioration possible : ajouter une liste déroulante possédant 3 valeurs (crédit, débit, tous) afin de limiter la recherche du compte aux comptes de crédit ou aux comptes de débit ou aux 2.