Intégration et utilisation de bibliothèques externes

1. Le gestionnaire de paquets NuGet

NuGet (nuget.org/) est un gestionnaire de packages pour la plateforme .NET visant à faciliter le processus d'intégration de bibliothèques externes à une application développée dans .NET.

NuGet est développé principalement par Microsoft et est distribué sous forme d'extensions des environnements de développement Visual Studio.

Un package NuGet est un fichier compressé avec l'extension « .nupkg », qui contient du code compilé (sous forme de DLL) ainsi que le contenu nécessaire aux projets qui les utilisent. C'est du code réutilisable que des développeurs mettent à la disposition d'autres développeurs pour l'utiliser dans leurs projets.

Le mécanisme de partage de code, sous forme de packages NuGet, définit la façon dont les packages .NET sont créés, hébergés et utilisés et fournit des outils pour chacun de ces rôles.

En effet, les développeurs qui ont du code à partager créent des packages et les publient sur un hôte public ou privé. Les consommateurs récupèrent ces packages, les ajoutent à leurs projets, puis appellent leurs fonctionnalités dans le code de leur projet.

Les packages NuGet peuvent être installés dans n'importe quel projet .NET, à condition qu'ils prennent en charge la même version cible de .NET Framework que le projet.

2. Utilisation du Framework « Newtonsoft.Json »

Un DataGrid (grille de données) est un contrôle qui affiche les données dans une grille personnalisable. Il fournit un moyen flexible d'afficher une collection de données dans des lignes et des colonnes.

Un DataGrid peut être rempli manuellement ou avec du code-behind. On peut aussi y charger le contenu d'un fichier.

- Reprenez la solution Visual Studio de l'exemple, Gestion des participants, vu dans le chapitre précédent. Renommez le projet « 1-Newtonsoft.json ».

- On veut que les données des participants soient chargées à partir d'un fichier d'extension « .JSON » (JavaScript Object Notation). En plus, les données de chaque nouveau participant seront sauvegardées dans le même fichier. En cas de suppression, les enregistrements en question doivent être supprimées du fichier.
- JSON est un format d'échange de données léger permettant de stocker des données selon un formatage qui rend facile pour le code de l'analyser et de le gérer.
- Les données des participants se trouvent dans le fichier « fParticipants.json ». Placez ce fichier dans le dossier « bin/Debug » de votre projet. Voici un extrait de ce fichier.
- Chaque enregistrement d'un fichier JSON est formé d'une collection de paires "nom":"valeur".

```
[{"id":1,"Prenom":"Damita","Nom":"Garton","Gender":"F","Niveau":"Professionnel","Email"
:"dgarton0@aol.com"},
{"id":2,"Prenom":"Letizia","Nom":"Muldowney","Gender":"M","Niveau":"Professionnel","Ema
il":"lmuldowney1@google.com"},
{"id":3,"Prenom":"Leupold","Nom":"Temlett","Gender":"M","Niveau":"Débutant","Email":"lt
emlett2@github.io"}]
```

NB : Il ne faut pas avoir des sauts de lignes dans le fichier JSON.

1.1. Intégration du package « Newtonsoft.Json » dans le projet

- « Newtonsoft.Json » (newtonsoft.com/json) est le package NuGet est le Framework le plus téléchargé et le plus utilisé dans la communauté. Il permet de gérer la sérialisation et désérialisation de fichiers respectant le format « JSON ».
- Dans l'explorateur de solutions, cliquez avec le bouton droit sur « Références » et choisissez « Gérer les packages NuGet ».
- Assurez-vous que la source des packages est « nuget.org ».
- Sélectionnez l'onglet « Parcourir » et dans la zone de texte de recherche inscrivez « Newtonsoft.Json ». Sélectionnez-le dans la zone de résultat, puis cliquez sur « installer ».
- Remarquez que dans les « références » du projet, l'espace de nommage de « Newtonsoft.Json » a été ajouté.
- Actuellement, les informations des participants sont chargées dans une « ObservableCollection » nommée « participants » dans le constructeur de la classe « UCGestionPart ».

  Mettez en commentaire les instructions permettant de remplir la collection, exceptée l'instruction : `dgParticipant.ItemsSource = participants;` qui précise la source des données du DataGrid.

- C'est dans la classe « Participant » que nous allons ajouter le code qui nous permettra de lire les données se trouvant dans le fichier de données « fParticipants.json » :

- Ajoutez l'instruction permettant de demander l'utilisation du namespace « Newtonsoft.Json » dans la classe « Participant ».
- Déclarez une constante de type chaîne de caractères contenant le chemin relatif du fichier de données « fParticipants.json ».

1.2. Désérialisation du contenu du fichier dans une liste

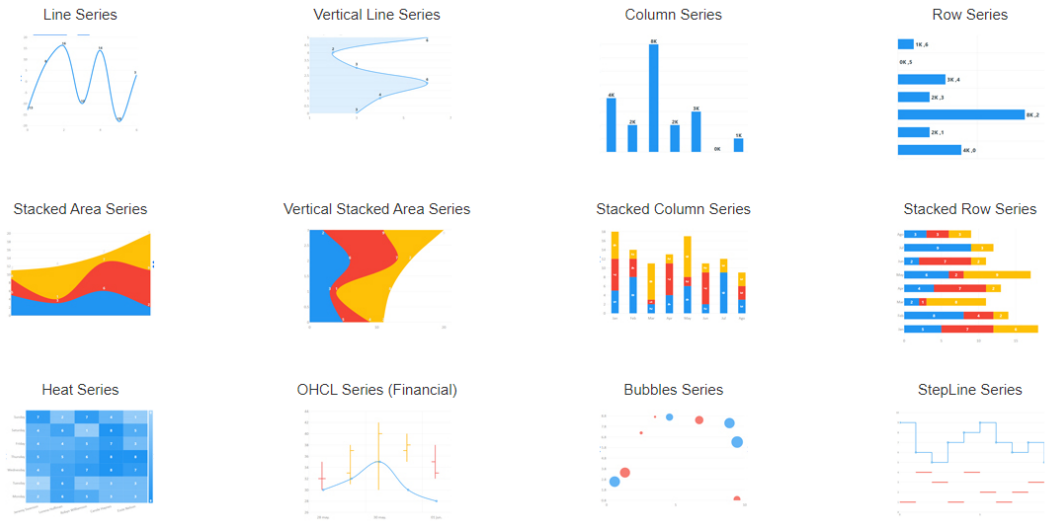
- Définissez la méthode « ChargerFichier » dans la classe « Participant ». Cette méthode lit le fichier « fParticipants.json » et le désérialise dans la collection observable des participants.
- **NB : Quand vous utilisez la méthode de désérialisation du contenu du fichier, les noms des entêtes dans le fichier « .json » doivent être exactement les mêmes que les propriétés dans la classe. En plus, la classe devrait contenir <u>un seul constructeur</u>.**
- Dans le constructeur de la classe « UCGestionPart » et juste avant l'instruction qui précise la source des données du DataGrid « dgParticipant », appelez la méthode « ChargerFichier ».

1.3. Ajout, modification et suppression de données d'un fichier « .json »

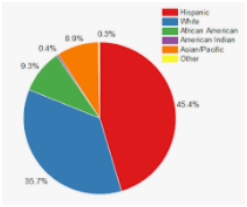
- Il est possible d'ajouter de nouveaux participants dans le fichier « fParticipants.json ». Dans la classe « MainWindow », ajoutez l'instruction d'utilisation du namespace « Newtonsoft.json ».
- Dans la classe « MainWindow », créez la méthode statique « ReecrireFichier » permettant de réécrire le fichier « fParticipant.json » à partir du contenu de la collection de participants.
- Lors de l'ajout d'un nouveau participant, on veut que ses données soient sauvegardées dans le fichier « fParticipants.json ». Appelez la méthode « ReecrireFichier » à la fin de la méthode « Click_Valider » de la classe « UCAjoutPart ».
- Ajoutez aussi l'appel de « ReecrireFichier », dans la classe « UCGestionPart », à la fin de la méthode « BtnModifPart_Click », pour MAJ le fichier après une modification et à la fin de la méthode « BtnSupprPart_Click », pour MAJ le fichier après la suppression d'un participant.

3. Utilisation du Framework « LiveCharts.Wpf »

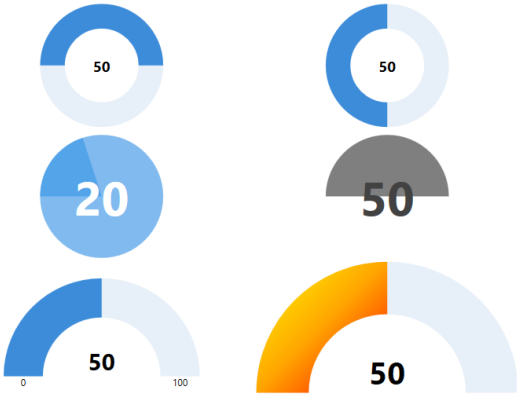
- « LiveCharts » (https://lvcharts.net/) est un Framework de visualisation de données sous forme de graphiques.
- Parmi les types de graphiques principaux que propose LiveCharts, on cite :
  - Graphique cartésien (CartesianChart) : permet de tracer n'importe quelle série qui utilise un système de coordonnées cartésien. Chaque point est une paire de valeurs (X, Y).



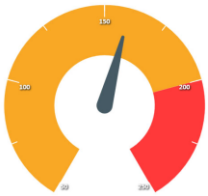
- Diagramme circulaire (PieChart) :



- Jauge (Gauge) : permet d'afficher la progression ou l'achèvement. Dispose de 2 modes : 180 et 360 degrés.

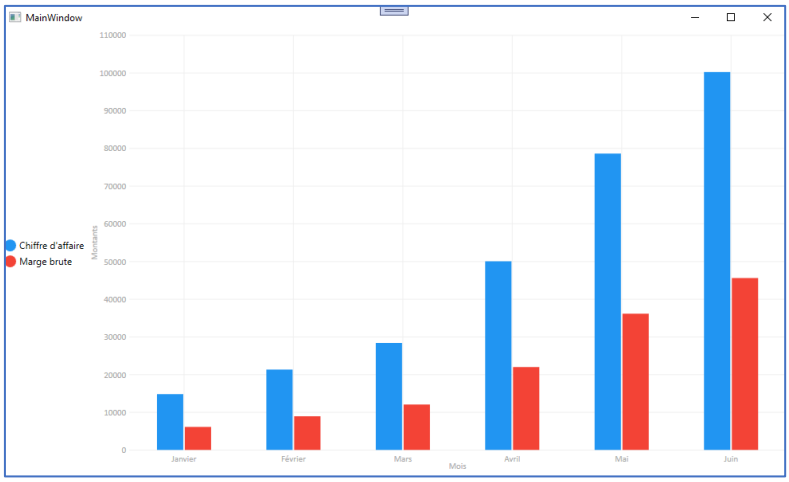


- Jauge angulaire (AngularGauge) : Permet d'afficher une valeur dans une plage de valeurs possibles.



Exemple :

- Créez un nouveau projet « 2-LiveCharts ».
- Nous voulons une interface utilisateur contenant un « graphique cartésien » sous forme de « série de colonnes » pour afficher l'évolution du bilan d'activités d'une entreprise durant le premier semestre de l'année (de Janvier à Juin).



- Nous utiliserons le Framework « LiveCharts » pour la présentation du graphique. Dans l'explorateur de solution, bouton droite sur le nom du projet, « Gérer les packages Nuget… ». Dans l'onglet « Parcourir », rechercher « LiveCharts.Wpf » puis l'installer.
- Nous commençons par le code-behind de cette application. Dans le fichier « MainWindows.xaml.cs », faites appel aux espaces de nommage de LiveCharts.

```
using LiveCharts;
using LiveCharts.Wpf;
```

- Dans la classe « MainWindow », ajoutez les instructions permettant la création de la collection des séries du graphique et des labels sur l'axe des abscisses.
- Dans le constructeur de la classe « MainWindow », on crée la première série de données qui prend la forme d'un graphique en barres appelée « ColumnSeries » ayant les deux propriétés « Title » pour le titre de la colonne et « Values » pour stocker les chiffres d'affaires des mois de Janvier à Mai.
- Il n'est pas trop tard si on veut ajouter le chiffre d'affaires du mois de juin. Étant donné que les chiffres d'affaires sont ceux de la première collection de séries, ces valeurs se trouvent à l'indice 0 de « SC ». Ajoutez le chiffre d'affaires du mois de juin.
- Les titres des colonnes représentent les mois et on peut les ajouter en les insérant dans le tableau « Labels ».
- Toujours dans le même graphique, on veut afficher aussi une autre collection de valeurs qui représentent cette fois les marges brutes que l'entreprise a réalisé par mois.
- On peut aussi ajouter la marge brute du mois de juin. Cette collection se trouve à l'indice 1 de « SC ».

- À la fin du constructeur de la classe « MainWindow », définissez le contexte de données du graphique.
- Dans le code XAML de l'application, nous utiliserons du binding pour relier les contrôles de l'interface utilisateur aux propriétés dans le code-behind. Dans la balise racine « Window » du fichier XAML, ajoutez la déclaration qui mappe l'espace de nommage de « LiveCharts.Wpf ».

```
xmlns:lvc="clr-namespace:LiveCharts.Wpf;assembly=LiveCharts.Wpf"
```

- Modifiez les propriétés Title, Height et Width :

```
Title="Bilan d'activités_premier semestre" Height="600" Width="1000"
```

- Dans le panel « Grid », ajoutez les contrôles permettant de définir les colonnes du graphique cartésien, l'axe des abscisses, l'axe des ordonnées et la légende.
- Supposons que nous voulons que les marges brutes soient affichées sous forme d'un graphique en ligne. Dans le code-behind, remplacez « ColumnSeries » par « LineSeries ».