Documentation

ChartJSPilotageQMR - 2023

Informations

Nom du projet	ChartJSPilotageQMR
Type de document	Documentation
Date	14/03/2023
Version	1.0
Mots-clés	Graphique – Chart JS – Base de données
Auteur	Arthur Manceau

Rédaction et modifications

Version	Date	Nom	Description
1.0	14/03/2023	Arthur Manceau	• Première version
1.1	21/03/2023	Arthur Manceau	• Modifications + ajout créer un projet

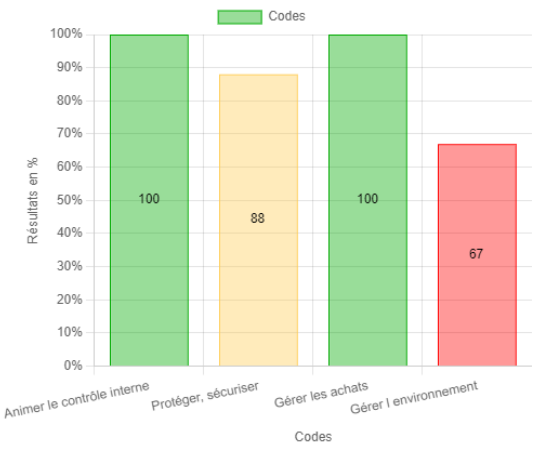
Tables des matières

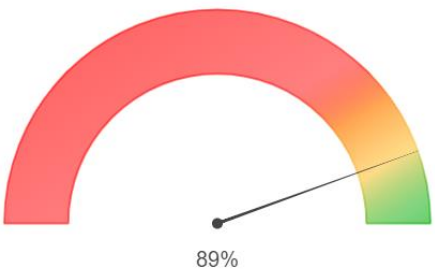
..... 1
1. Résumé du document 3
2. Créer un projet dans Visual Studio 4
2.1. Créer le projet dans Visual Studio 4
2.2. Créer une page web dans Visual Studio 5
3. Appel de la librairie ChartJS 9
4. Création d'une zone de traçage | interprétation du code JavaScript 9
5. Connaissances de base sur les variables 10
6. Création d'un histogramme à l'aide de ChartJS 12
5.1. Ajout du graphique à la page web 15
5.2. Pour aller plus loin 15
7. Création d'une jauge à l'aide de ChartJS 16
6.1. Création d'un dégradé de couleur intelligent 17
6.2. Création de la constante data 18
6.3. Création de l'aiguille intelligente 21

1. Résumé du document

Ce document est la documentation du projet web ChartJSPilotageQMR, il retrace toute la manière à employer pour réaliser un histogramme est une jauge. Il sera donc divisé en deux parties :

- Création d'un histogramme à partir de ChartJS ;
- Création d'une jauge à partir de ChartJS ;



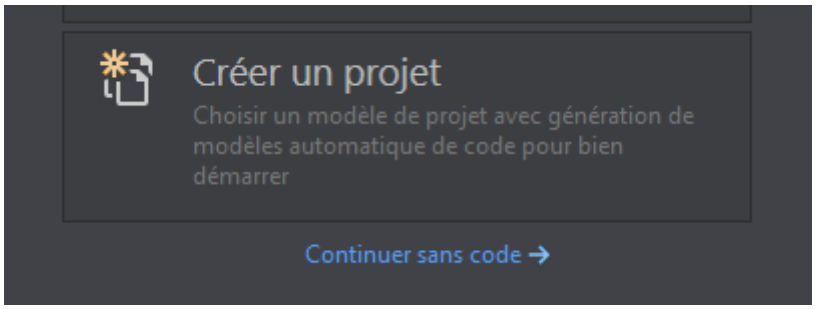


2. Créer un projet dans Visual Studio

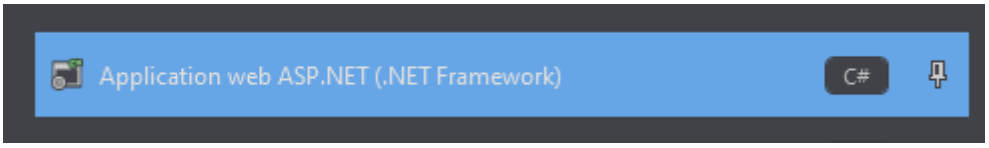
2.1. Créer le projet dans Visual Studio

Tout d'abord nous allons créer un projet dans Visual studio pour cela assurez vous d'avoir l'application Visual Studio.

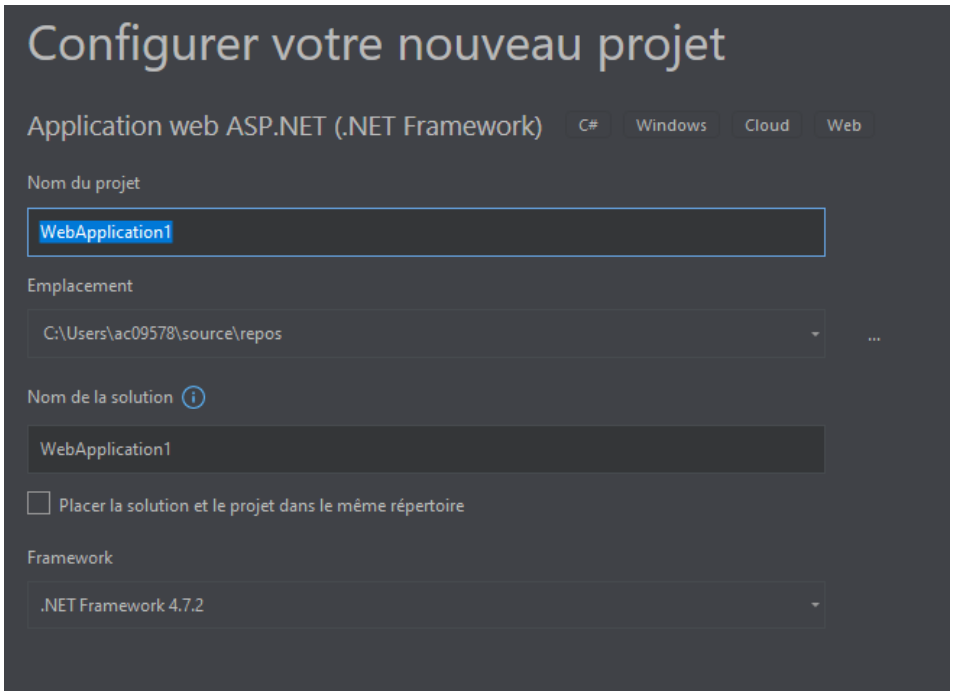
- Créer un projet :



- Choisir le modèle :

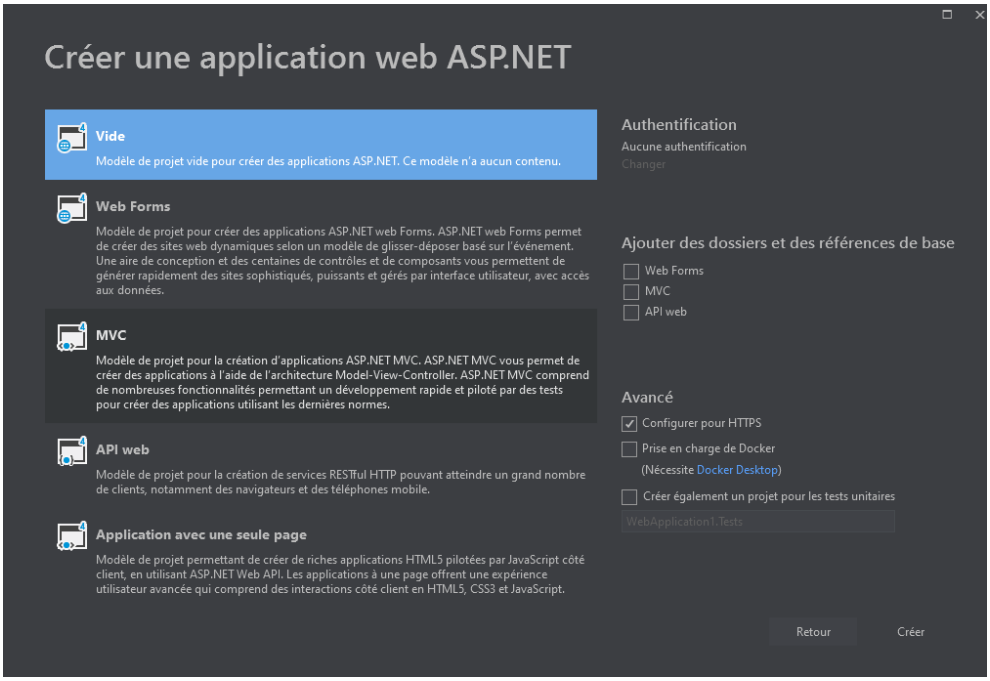


- Cliquer sur « Suivant » en bas à droite de l'écran
- Modifier le nom de votre projet si vous le souhaiter :



- Cliquer sur « Créer » en bas à droite de l'écran

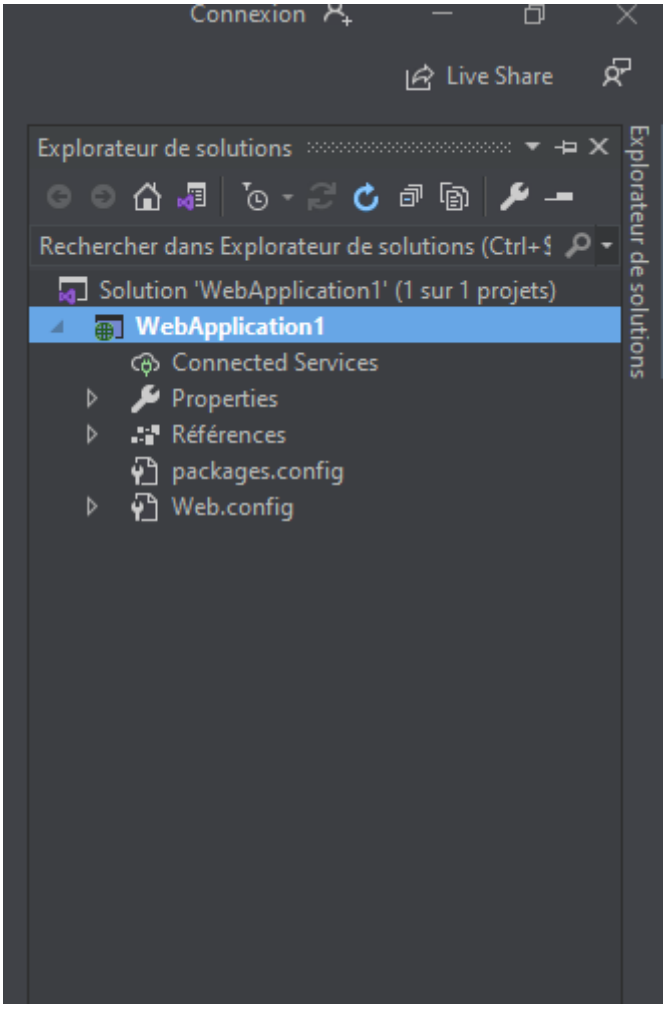
- Sélectionner « Vide », puis « Créer » :



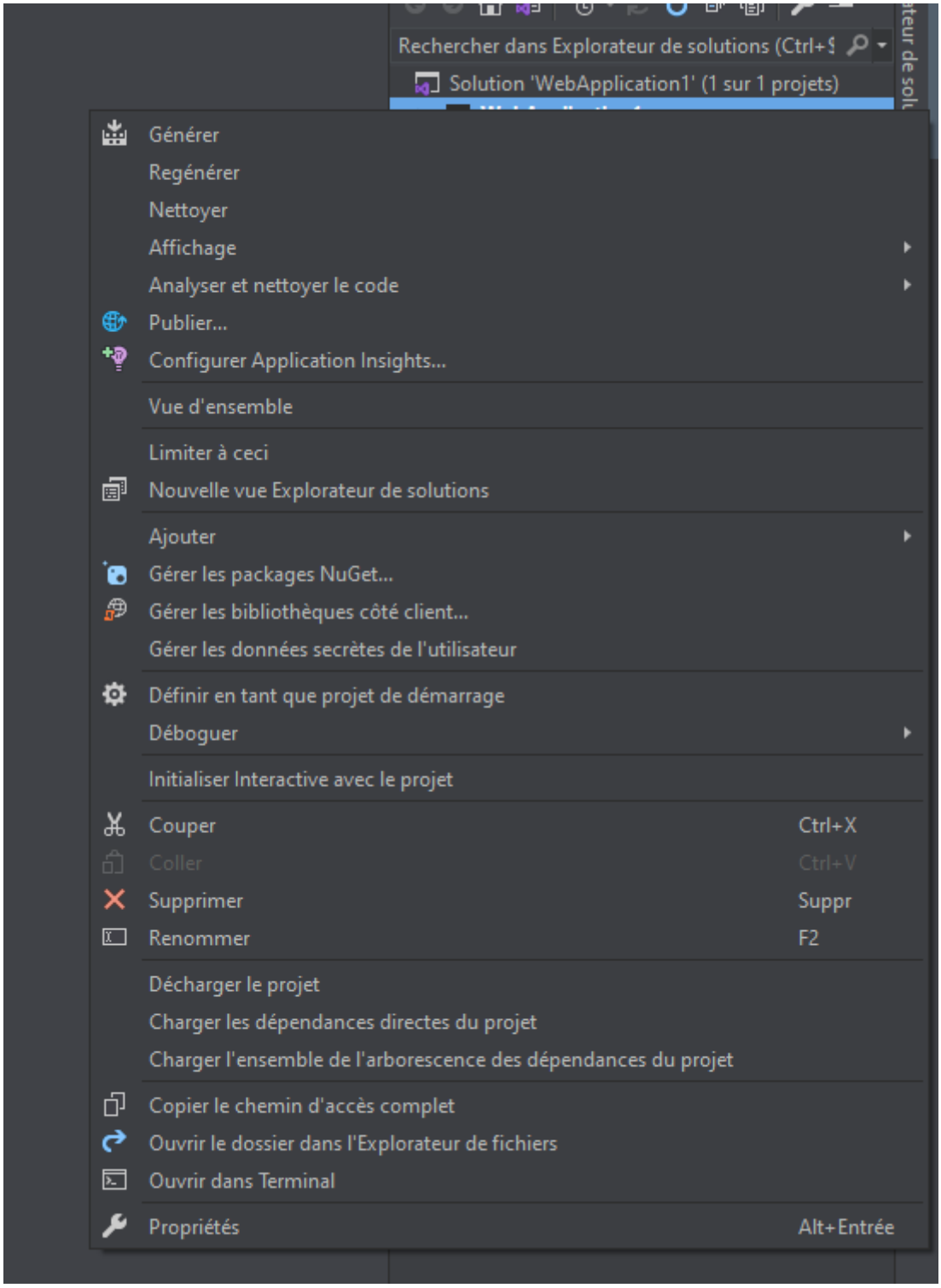
2.2. Créer une page web dans Visual Studio

Après avoir ouvert votre nouveau projet dans Visual Studio, il est nécessaire de créer une page web pour pouvoir réaliser les graphiques.

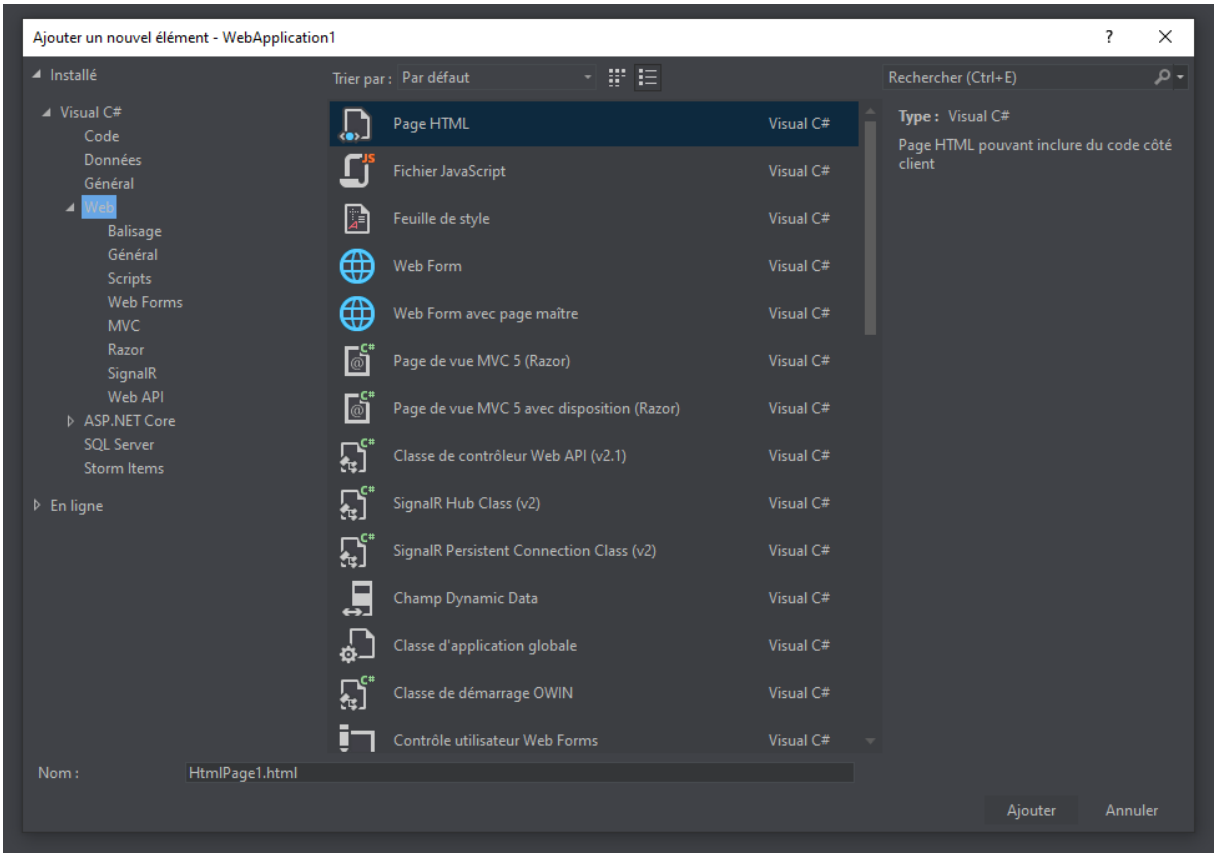
- Aller dans « Explorateur de solution » présent sur la droite de l'écran :



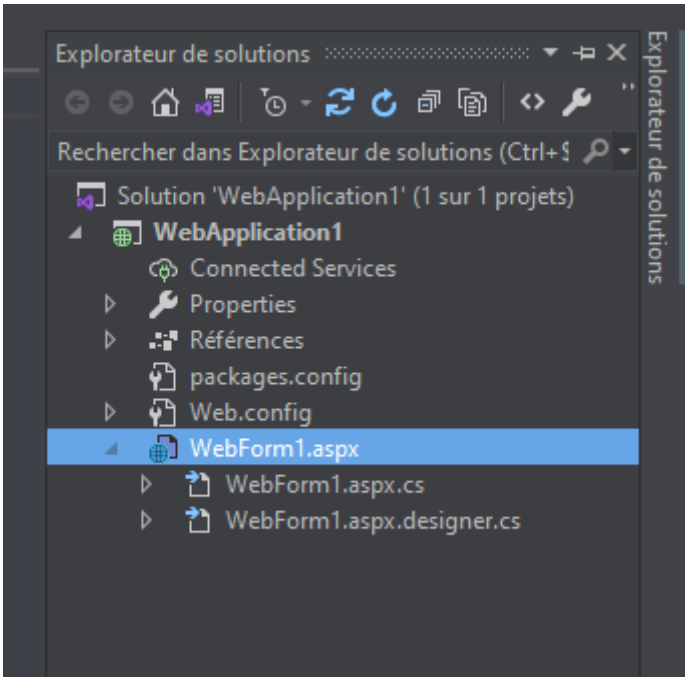
- Faire un clic droit sur le nom de votre solution :



- Cliquer sur « Ajouter », « nouvel élément » :



- Sélectionner « Web Form » (vous pouvez changer le nom de la page en bas de la fenêtre dans Nom) puis appuyer sur « Ajouter »
- Pour afficher le code C# de cette page, déplier le triangle présent sur la gauche du nom de votre page dans l'explorateur de solutions



- Cliquer en suite sur NomdeMaPage.aspx.cs, vous y trouverez le code behind de la page web.

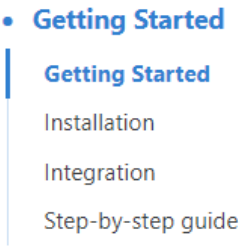
3. Appel de la librairie ChartJS

Tout d'abord, la création de ces graphiques nécessite l'appel de la librairie ChartJS. Pour faire appel à cette librairie il suffit de rajouter un lien JavaScript dans le header de la page web.

Pour récupérer ce lien, vous devez aller sur la documentation de ChartJS : https://www.chartjs.org/

La page suivante s'affiche, cliquer sur « Get Started » pour commencer :

Puis dans l'onglet "Getting Started" :



Vous pouvez y retrouver le lien du script suivant : https://cdn.jsdelivr.net/npm/chart.js

Allons maintenant dans le code, après avoir créé un nouveau projet, dans la partie .NET de votre projet (mapage.aspx) il faut ajouter le lien dans le header (entre les balises <head></head>) de la manière suivante :

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

La librairie est maintenant importée.

4. Création d'une zone de traçage | interprétation du code JavaScript

Il est impossible d'écrire du code JavaScript directement dans la page C# de votre projet (mapage.aspx.cs), nous allons donc contourner ce problème de la manière suivante :

Page aspx	Page aspx.cs
- Création d'un panel avec un id	- Création d'une fonction qui va créer le graphique - Déclaration d'une variable string où l'on va insérer le code JavaScript - Alimenter la variable avec le code JavaScript - Déclaration d'un Literal - Ajout de la variable au Literal - Ajout du Literal au panel créé dans la page aspx

Il est nécessaire de réaliser les actions dans l'ordre.

Explications : le navigateur va lire le panel déclaré qui lui, contient le Literal, dans ce Literal une variable est présente. Donc le navigateur va lire la variable déclarée comportant le script. Cette variable est une chaîne de caractère que le navigateur va interpréter, alors il interprétera le code JavaScript présent à l'intérieur de cette variable.

- Insérer un Panel dans votre page web :

```
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Panel ID="myPanel1" runat="server"></asp:Panel>
        </div>
    </form>
</body>
```

5. Connaissances de base sur les variables

Nous allons maintenant travailler uniquement dans la page C# (mapage.aspx.cs) à l'intérieur de la variable de type chaîne de caractères. Il est important de vous rappeler que pour mettre dans une variable de type string plusieurs lignes il faut ajouter un « @ » devant la première « " ».

Exemple 1 :

String maVariable1 = "";

maVariable1 = "Les singes sont des mammifères de l'ordre des primates,

généralement arboricoles, à la face souvent glabre et

caractérisés par un encéphale développé et de longs

membres terminés par des doigts.

";

Une erreur va survenir car le navigateur ne verra uniquement la première ligne, et cherchera sur cette ligne la dernière « " » qui permet de fermer la chaîne. Mais il ne la trouvera pas.

Exemple 2 :

String maVariable2 = "";

maVariable2 = @"Les singes sont des mammifères de l'ordre des primates,

généralement arboricoles, à la face souvent glabre et

caractérisés par un encéphale développé et de longs

membres terminés par des doigts.

";

Le navigateur ressortira la phrase suivante : Les singes sont des mammifères de l'ordre des primates, généralement arboricoles, à la face souvent glabre et caractérisés par un encéphale développé et de longs membres terminés par des doigts.

De plus, il est nécessaire de savoir que pour ajouter à une variable de type string du texte en plus il faut ajouter un « + » devant l'égale.

Exemple 3 :

String maVariable1 = "";

maVariable1 = "Les singes sont des mammifères de l'ordre des primates,";

maVariable1 = "généralement arboricoles, à la face souvent glabre et";

maVariable1 = "caractérisés par un encéphale développé et de longs";

maVariable1 = "membres terminés par des doigts";

Le navigateur affichera : membres terminés par des doigts. Car le contenu de la variable est écrasé à chaque ligne, donc il prendra uniquement le résultat du dernier ajout de texte à la variable.

Exemple 4 :

String maVariable2 = "";

maVariable2 += "Les singes sont des mammifères de l'ordre des primates,";

maVariable2 += "généralement arboricoles, à la face souvent glabre et";

maVariable2 += "caractérisés par un encéphale développé et de longs";

maVariable2 += "membres terminés par des doigts";

Le navigateur affichera : Les singes sont des mammifères de l'ordre des primates, généralement arboricoles, à la face souvent glabre et caractérisés par un encéphale développé et de longs membres terminés par des doigts.

Dans le code ci-dessus on met un « + » devant le « = » pour ajouter le contenu de la variable au précédent contenu ajouté.

Maintenant que vous êtes capable de déclarer une variable et que vous connaissez les particularités des variables de type chaîne de caractères nous allons pouvoir ajouter le code JavaScript à la variable.

6. Création d'un histogramme à l'aide de ChartJS

Commençons par déclarer une variable, dans les exemples suivants je lui donnerais le nom de « chart » mais vous pouvez l'appeler de n'importe quelle manière. Notez qu'il est nécessaire qu'elle possède un nom explicite pour la compréhension du code.

Nous allons tout d'abord déclarer un canevas qui permet d'effectuer des rendus dynamiques d'images en html.

```
//Déclaration chaîne de caractère qui va posséder tout le script du graphique
string chart = @"
       <div>
              <canvas id='myChart' width='300' height='250'>
              </canvas>
       </div>";
```

La balise « <canvas></canvas> » représente donc le canevas qui comportera le graphique. Il est nécessaire de lui donner un identifiant pour pouvoir ajouter au canevas le du graphique, pour ma part il se nommera « myChart ».

Nous allons ensuite pouvoir commencer à écrire du code JavaScript. Pour que le navigateur comprenne que l'on écrit du JavaScript il va falloir ajouter la balise suivante « <script></script> » :

```
chart += @"<script>

              //Le code JavaScript se trouvera entre ces balises

      "<script> ;
```

Toutes les balises sont maintenant déclarées.

Tout le code que nous allons écrire est à placer entre ces balises « <script></script> ».

Premièrement nous allons déclarer une constante permettant de trouver le canevas afin de lui ajouter le graphique. En JavaScript la déclaration d'une constante se fait de la manière suivante :

const maConstante = … ; donc nous allons déclaré une constante qui trouvera le canevas dans le document en lui passant en paramètre l'identifiant du canevas.

```
const ctx = document.getElementById('myChart');
```

Après avoir trouvé le canevas, nous allons créer le graphique. Pour cela nous allons créer un nouveau graphique qui prendra en compte :

- ctx → constante qui trouve le canevas ;
- type → le type de graphique ;
- data → les données du graphique ;
- labels → les données de l'axe des abscisses ;
- datasets → l'ensemble des données ;
- data → les données de l'axe des ordonnées ;
- label → le titre de la série de données ;

Notez que l'on peut apercevoir deux « data » le premier data est considéré comme les données générales du graphique c'est-à-dire les données en x et en y. Le second « data » est celui des données de l'axe des abscisses. Ils se nomment pareils mais sont complètement différents.

```
new Chart(ctx, {
    type: 'bar',
    data: {
       //Les données en x et y se trouveront ici
    }
  });
```

Ici on exprime un graphique de type « bar » qui signifie un histogramme. On lui ajoute la section des données générales.

On va maintenant alimenter la section « data ». Dans cette section plusieurs sous-sections sont présentes :

- labels → les données de l'axe X ;
- datasets → l'ensemble des données ;

```
new Chart(ctx, {
       type: 'bar',
       data: {
              labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
              datasets: [{
                     //les données de y et le titre se trouveront ici
              ]}
       }
  });
```

Les données de l'axe x sont maintenant définis, il ne reste plus qu'à insérer le titre de la série de données et les données de l'axe y.

```
new Chart(ctx, {
       type: 'bar',
       data: {
              labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
              datasets: [{
                     label: '# of Votes',
                     data: [12, 19, 3, 5, 2, 3],
                     borderWidth: 1
              ]}
       }
  });
```

Notez que j'ai jouté le paramètre « borderWidth » qui est l'épaisseur de la bordure de chaque barre du graphique pour un effet esthétique, il n'est pas nécessaire de l'ajouter.

Remarque : pour séparer un paramètre/section d'un autre il faut utiliser la virgule.

Exemple 1 :

label: '# of Votes'

data: [12, 19, 3, 5, 2, 3]

Le navigateur comprendra : label: '# of Votes'data: [12, 19, 3, 5, 2, 3], en effet si l'on ne sépare pas les lignes avec des virgules le navigateur considère que les deux lignes forment une seule ligne.

Exemple 2 :

label: '# of Votes',

data: [12, 19, 3, 5, 2, 3],

Le navigateur comprendra :

label: '# of Votes',

data: [12, 19, 3, 5, 2, 3],

Le graphique est maintenant créé.

5.1. Ajout du graphique à la page web

Il ne reste plus qu'à ajouter le graphique à la page en ajoutant la variable à un Literal qui lui est ajouté au panel déclaré dans la page aspx comme vu précédemment. Notez que le code suivant est du code C#, donc il est écrit en dehors des balises « <script></script> ». On l'ajoute après celles-ci.

```
//Déclaration d'un literal LTD
Literal LTD = new Literal();

//Ajout de la variable chart qui contient le script du graphique
LTD.Text = chart;

//Ajout du literal au panel déclaré dans la page aspx
myPanel1.Controls.Add(LTD);
```

5.2. Pour aller plus loin

Pour des questions d'esthétique, nous pouvons rajouter des couleurs pour chaque barre, elles peuvent être différentes ou non. J'ai choisi d'avoir des barres de couleur différentes, pour cela nous allons ajouter à la section « datasets » deux paramètres :

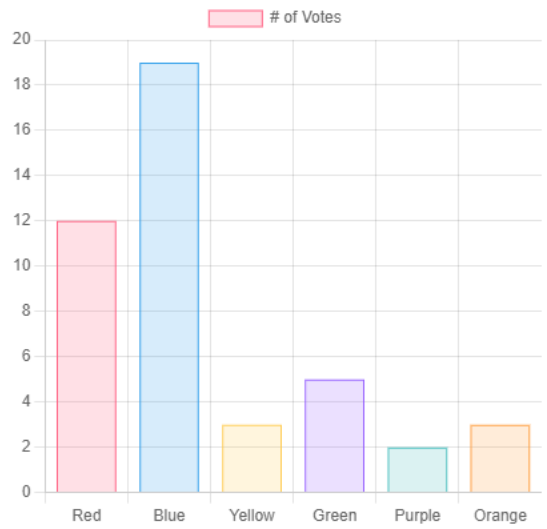
- backgroundColor → couleur de la barre ;

- borderColor → couleur de la bordure de la barre ;

Comme j'ai sur mon graphique six barres, je vais ajouter six couleurs.

```
new Chart(ctx, {
	type: 'bar',
	data: {
		labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
		datasets: [{
			label: '# of Votes',
			data: [12, 19, 3, 5, 2, 3],
			borderWidth: 1,
			backgroundColor: [
				'rgba(255, 99, 132, 0.2)',
				'rgba(54, 162, 235, 0.2)',
				'rgba(255, 205, 86, 0.2)',
				'rgba(153, 102, 255, 0.2)',
				'rgba(75, 192, 192, 0.2)',
				'rgba(255, 159, 64, 0.2)'
			],
			borderColor: [
				'rgb(255, 99, 132)',
				'rgb(54, 162, 235)',
				'rgb(255, 205, 86)',
				'rgb(153, 102, 255)',
				'rgb(75, 192, 192)',
				'rgb(255, 159, 64)'
			],
		]}
	}
  });
```

Voici le rendu avec les couleurs :



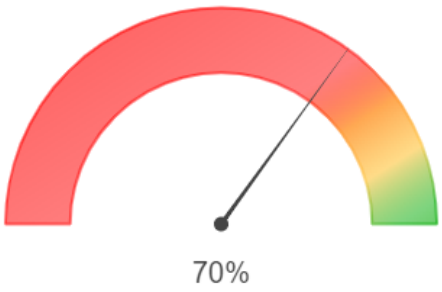
7. Création d'une jauge à l'aide de ChartJS

Premièrement, Nous allons redémarrer au niveau des balises scripts. Il est n'est pas nécessaire de réaliser un nouveau panel, nous pouvons très bien mettre les deux graphiques sur dans le même panel. Notez qu'il faut obligatoirement changer l'identifiant du canevas car un identifiant est unique. De plus une nouvelle chaîne de caractère doit être déclarée comme :

String gauge = "";

Tout d'abord, reprenons les bases : la jauge est un graphique de type doughnut que l'on coupe en deux. Cette jauge n'affiche qu'une seule valeur et représente cette valeur par rapport à un chiffre général.

Voici un aperçu de la jauge que nous allons réaliser :



6.1. Création d'un dégradé de couleur intelligent

Nous allons donc commencer par réaliser un dégradé intelligent, c'est-à-dire un dégradé qui s'adapte automatiquement à la taille de la jauge, qui n'est pas fixe.

Afin de réaliser ce dégradé nous allons créer une fonction nommée « getGradient » pour ma part mais il est toujours possible de la renommer.

```
let width, height, gradient;
function getGradient(ctx, chartArea) {

  //Déclaration des constantes chartWidth et chartHeight
  const chartWidth = chartArea.right - chartArea.left;
  const chartHeight = chartArea.bottom - chartArea.top;

  //Pour créer le dégradé avant le graphique
  if (!gradient || width !== chartWidth || height !== chartHeight) {

    //Calcul automatique des valeurs à obtenir pour le dégradé selon la taille de
la jauge
    width = (chartWidth / 1.5);
    height = chartHeight + (chartHeight / 20);

    //Le dégradé est définit sur une échelle de 1, 1 = 100%

    //Le dégradé peut prendre en paramètre (x, y, width, height)
    //Création d'un angle précis avec width et height pour que l'angle des couleurs
soit coordonné avec l'arrondi de la jauge
    gradient = ctx.createLinearGradient(0, 0, width, height);

    //Ajout de la couleur rouge avec 0.7 d'opacité au point 0
    gradient.addColorStop(0, 'rgba(255, 0, 0, 0.7)');

    //Ajout de la couleur rouge avec 0.5 d'opacité au point 0.6 --> obliger pour
bien définir les couleurs du dégradé au bon endroit
    gradient.addColorStop(0.6, 'rgba(255, 0, 0, 0.5)');

    //Ajout de la couleur orange avec 0.9 d'opacité au point 0.8
    gradient.addColorStop(0.7, 'rgba(255, 159, 64, 0.9)');

    //Ajout de la couleur jaune avec 0.7 d'opacité au point 0.8
    gradient.addColorStop(0.8, 'rgba(255, 205, 86, 0.7)');

    //Ajout de la couleur vert avec 0.5 d'opacité au point 0.9
    gradient.addColorStop(0.9, 'rgba(0, 170, 0, 0.5)');

    //Ajout de la couleur vert avec 0.7 d'opacité au point 1 --> obliger pour bien
définir les couleurs du dégradé au bon endroit
    gradient.addColorStop(1, 'rgba(0, 170, 0, 0.7)');
  }

  //On retourne la valeur composée de toutes les couleurs du dégradé
  return gradient;
}
```

Dans cette fonction on a donc ajouté 6 couleurs dont deux sont les même avec une opacité différente. Elles peuvent aussi être changées tout comme leur opacité, vous pouvez jouer avec leur positionnement pour avoir le dégradé voulu. De plus, le nombre de couleur n'importe pas, il est tout à fait possible d'en mettre deux comme dix.

6.2. Création de la constante data

Nous allons maintenant réaliser la constante « data ». En effet la constante data sera la valeur que l'on mettra dans la section « data », il est possible d'ajouter ces valeurs dans la section directement mais pour la réalisation d'une jauge il est préférable de travailler uniquement avec des constantes puis ajouter chaque constante à la fin. C'est dans cette constante que nous allons réaliser la forme de notre jauge, ainsi que la valeur de l'aiguille.

```
//Déclaration de la constante qui comportera la donnée en y donc la valeur totale
de la jauge (100)
const data = {

  datasets: [{

    //Ajout de la valeur 100 insérée au début pour mettre la jauge sur 100
    data: [100],

    //Ajout à la couleur de la jauge la fonction qui a créé le dégradé
    backgroundColor: function(context) {
      const chart = context.chart;
      const {ctx, chartArea} = chart;

      if (!chartArea)
      {
        return;
      }
      return getGradient(ctx, chartArea);
    },

    //Ajout à la couleur de la jauge la fonction qui a créé le dégradé
    borderColor: function(context) {
      const chart = context.chart;
      const {ctx, chartArea} = chart;

      if (!chartArea)
      {
        // This case happens on initial chart load
        return;
      }
      return getGradient(ctx, chartArea);
    },

    //Définition de la valeur de l'aiguille --> seule donnée du graphique
    needleValue: '70',

    //Pour que le graphique soit responsive = lorsque diminution de la fenêtre,
graphique diminue aussi correctement
    responsive: true,

    //On coupe le doughnut en 2 (180 degrés)
    circumference: 180,

    //On fait une rotation de 270 degrés pour le que demi doughnut soit à
l'horizontal
    rotation: 270,

    //Définition de l'épaisseur de la jauge
    cutout: '70%',
    aspectRatio: 2,
  }]
};
```

6.3. Création de l’aiguille intelligente

Pour une question d’esthétique nous allons ajouter une aiguille qui adapte sa position selon sa valeur. Il est tout à fait possible de faire autre chose. L’aiguille se compose de deux parties :

- Le point à la base de l’aiguille ;
- La ligne qui réalise l’aiguille ;

Nous ajouterons en dessous de celle-ci un texte comprenant la valeur de l’aiguille pour une meilleure lisibilité de la jauge.

```
const gaugeNeedle = {

	//Définition de l'id de l'aiguille
	id: 'gaugeNeedle',

	//On réalise le dessin de l'aiguille
	afterDatasetDraw(chart, args, options) {
		const {ctx, config, data, chartArea: { top, bottom, left, right,
width, height } } = myGauge;
	ctx.save();

	const needleValue = data.datasets[0].needleValue;
	const dataTotal = data.datasets[0].data.reduce((a,b) => a+ b, 0);

	//Calcul de l'angle de l'aiguille
	const angle = Math.PI + (1 / dataTotal * needleValue * Math.PI);

	//Calcul de l'emplacement de l'aiguille
	const cx = width / 2;
	const cy = chart._metasets[0].data[0].y;

	//Calcul de la hauteur de l'aiguille selon la hauteur de la jauge
	const needleHeight = height / 2;

	//Ajouter ici la mise en forme de l’aiguille
	//Ajouter ici la mise en forme du point sous l’aiguille
	//Ajouter ici la mise en forme de la valeur sous le point de l’aiguille

}
```

On peut apercevoir que le taille de l’aiguille varie selon la hauteur et la largeur de la jauge.

- La mise en forme de l’aiguille :

```
//Mise en forme de l'aiguille :

    //Définition du placement de l'aiguille sur le graphique
    ctx.translate(cx, cy);

    //Définition de l'angle de l'aiguille
    ctx.rotate(angle);
    ctx.beginPath();

    //Définition de l'épaisseur de l'aiguille
    ctx.moveTo(0, -1.5);
    ctx.lineTo(needleHeight, 0);
    ctx.lineTo(0, 1.5);

    //Définition de la couleur de l'aiguille
    ctx.fillStyle = '#444';
    ctx.fill();
```

- La mise en forme du point :

```
//Mise en forme du point sous l'aiguille :

    //Définition du placement du point sur le graphique
    ctx.translate(-cx, -cy);
    ctx.beginPath();

    //Définition de la forme de point (donc rond ici)
    ctx.arc(cx ,cy, 5, 0, 10);
    ctx.fill();
    ctx.restore();
```

- La mise en forme du texte sous le point :

```
//Mise en forme de la valeur y sous l'aiguille :

    //Définition du style du texte
    ctx.font = '20px Helvetica';

    //Définition de la couleur du texte
    ctx.fillStyle = '#444';

    //Définition de l'emplacement du texte
    ctx.fillText(needleValue + '%', cx, cy + 40);
    ctx.textAlign = 'center';
    ctx.restore();
```

Nous allons maintenant déclarer la constante config pour la mise en forme de toute la jauge ainsi que ses données et le dégradé. On ajoute aussi le plugin « gaugeNeedle » pour pouvoir créer la jauge et l'aiguille.

```
//Déclaration de la constante config
const config = {

  //Définition du type de graphique
  type: 'doughnut',

  //Définition des données de la jauge
  data,
  options: {
    plugins: {

      //On cache la légende de la jauge
      legend: {
        display: false
      },

      //On enlève la possibilité d'afficher la donnée au survol de la jauge
      tooltip: {
        enabled: false
      }
    }
  },

  //On importe le plugin gaugeNeedle pour pouvoir réaliser des jauges
  plugins: [gaugeNeedle]

};
```

Il ne reste plus qu'a créer un graphique et ajouter le tout dedans de la même manière que la création de l'histogramme :

```
//On déclare un nouveau graphique et on y ajoute tout le reste
const myGauge = new Chart(
  document.getElementById('myGauge'),
  config
);
```

Vous avez maintenant une jauge ainsi qu'un histogramme réalisé à l'aide ChartJS.