Validation technologique : affichage en temps réel de graphs dans un navigateur HTML5

Sommaire

I/ Choix de la technologie utilisé

1) Différentes possibilités

Il existe différentes méthodes et librairies capables de créer, gérer et afficher des graphiques. Le client souhaite avoir trois graphiques représentant chacun un axe de mesure du sismomètre.

Les solutions possibles sont :

- **HighCharts** : basé sur la technologie HTML 5, il est compatible avec la pluparts des navigateurs et utilise des technologies de rendu différentes selon ce dernier (Canvas, SVG ou VML). Son principal défaut son ses performances sur tablette (ralentissement remarquable), mais cela reste une librairie abordable et gratuite.
- **xCharts** : librairie JavaScript utilisant un combiné du HTML, CSS et SVG (Scalable Vector Graphics), elle se veut dynamique, fluide et intègre des outils d'intégration et de customisation. C'est une librairie libre.
- **CanvasJS** : librairie choisie pour répondre au cahier des charges et dont la présentation est faite ci-dessous.

2) Méthode choisie

La méthode choisie pour intégrer les graphiques du sismomètre est **CanvasJS**, une librairie JavaScript basé sur le canvas (feuille de dessin HTML5).

Les avantages sont nombreux :

- Le rendu graphique utilise une technologie **JavaScript** et **HTML 5**, il est donc parfaitement compatible avec les navigateurs présents sur les PC, les téléphones et les tablettes.
- Elle propose une **API** simple d'utilisation et intuitive.
- **Performante** : CanvasJS peut gérer l'affichage de milliers de points simultanément sans utiliser beaucoup de ressources CPU. Il est jusqu'à 10 fois plus rapide qu'un affichage Flash. Pour cela, le canvas précédent est copié à chaque mise à jour et on n'écrit donc que les nouvelles valeurs (ce qui assure un gain de temps et une consommation du processeur réduite à moins de 5%).

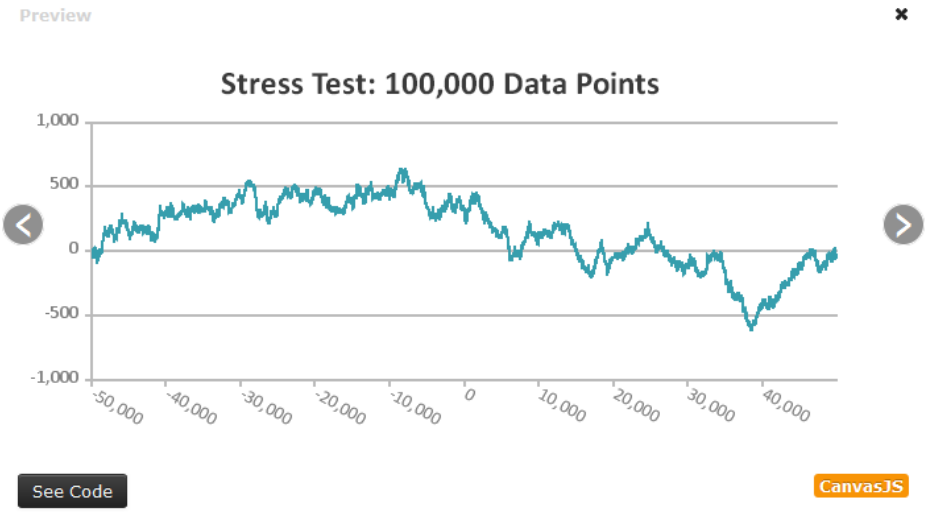

Figure 1 : exemple de graphique avec 100 000 points

II/ Tests et validation des graphiques

Dans notre cas, on souhaite pourvoir afficher trois graphiques simultanément, avec un affichage dynamique : les valeurs rentrent dans le canvas selon un intervalle donné et défilent (les plus vieilles valeurs défilent et sortent du graphique par la gauche et les nouvelles valeurs entrent par la droite du graphique).

On doit donc tester la création de trois graphiques individuels, ainsi qu'un graphique sur lequel il y'aura trois courbes différentes (chaque courbe représentant un axe X, Y ou Z).

1) Création d'un graphique aux valeurs statiques

Pour cette première validation, on va créer un graphique statique et l'afficher sur une page HTML.

a) Code JavaScript

```
1  /*
2   * Création d'un graphique statique avec canvasJS
3   */
4  window.onload = function () {
5      var dpsX = [];      // les points sur X
6      var dpsY = [];      // les points sur Y
7      var dpsZ = [];      // les points sur Z
8      var nbPoints = 0;           // nbr de points dans le graphe
9      var yValX = 0;              // valeurs initiales
10     var yValY = 0;
11     var yValZ = 0;
12     var nbVal = 3000;             // nb de points ajoutés à chaque tick
13     var dataLength = 3000;      // nombre de points visibles dans le graphe
14
15     var chart = new CanvasJS.Chart("divChart", {
16         includeZero: true,
17         axisX:{ valueFormatString: "hh:mm:ss"},
18         axisY: {
19             valueFormatString: "#0mg",
20             includeZero: true
21         },
22         zoomEnabled: true,
23         data: [
24             {
25                 type: "line",
26                 color: "orange",
27                 dataPoints: dpsX
28             },
29             {
30                 type: "line",
31                 color: "green",
32                 dataPoints: dpsY
33             },
34             {
35                 type: "line",
36                 color: "darkBlue",
37                 dataPoints: dpsZ
38             }
39         ]
40     });
```

L'écriture d'un graphique se fait dans du code JavaScript qui s'exécute une fois la page HTML chargée. On utilise donc *window.onload* pour lancer le script une fois la page chargé (ligne 4).

Les points sont stockés dans des tableaux (ligne 5,6 et 7). Le graphique est créé dans la variable *chart* (ligne 15) : on instancie un nouveau CanvasJS et on indique l'ID dans lequel il devra être rendu (*divChart* dans notre cas).

b) Code HTML

```
1   <!DOCTYPE HTML>
2   <html>
3       <head>
4           <title>Images</title>
5           <meta charset="utf-8">
6           <meta name="viewport" content="width=device-width, initial-scale=1">
7           <script type="text/javascript" src="../js/canvasjs.min.js"></script>
8           <script type="text/javascript" src="LGraphes1.js"></script>
9       </head>
10
11      <body>
12          <div>
13              <div id="divChart" style="height: 450px; width:100%;"> </div>
14          </div>
15      </body>
16  </html>
```

On importe la librairie CanvasJS (ligne 7) ainsi que le code JavaScript du graphique (ligne 8). On appelle le graphique dans le corps de la page HTML (*body*) par son ID, et on peut définir son style (sa hauteur en pixel et sa largeur en pourcentage).

c) Exécution de la page HTML

Figure 2 : le graphique est bien généré

2) Création de trois graphiques mise à jour en temps réel

a) Code JavaScript

```
1  /*
2   * Création de 3 graphes X, Y, Z avec canvasJS
3   */
4  window.onload = function () {
5      var dpsX = [];      // les points sur X
6      var dpsY = [];      // les points sur Y
7      var dpsZ = [];      // les points sur Z
8
9      var nbPoints = 0;           // nbr de points dans le graphe (inutile)
10     var yValX = 0;              // valeur initiale de la courbe X
11     var yValY = 0;
12     var yValZ = 0;
13     var nbVal = 20;             // nb de points ajoutés à chaque intervalle de temps
14     var updateInterval = 200;   // durée d'un intervalle de temps en ms
15     var dataLength = 3000;      // nombre de points visibles dans le graphe
16                                 // 30s à 100Hz
17     var chartX = new CanvasJS.Chart("divChartX", {  // Instanciation du graphs par son ID
18         axisX:{ valueFormatString: "hh:mm:ss"},     // Choix de l'échelle pour l'axe des X
19         axisY: { valueFormatString: "#0mg" },       // Choix de l'échelle pour l'axe des Y
20         includeZero: true,      // Permet d'avoir toujours le 0 afficher sur le graphique
21         data: [{
22             type: "line",       // Choix du type de graph
23             color: "orange",    // Choix de la couleur
24             dataPoints: dpsX    // Choix du tableau dans lequel les points seront stockés
25         }]
26     });
27     var chartY = new CanvasJS.Chart("divChartY", {
28         axisX:{ valueFormatString: "hh:mm:ss"},
29         axisY: {valueFormatString: "#0mg"},
30         includeZero: true,
31         data: [{
32             type: "line",
33             color: "green",
34             dataPoints: dpsY
35         }]
36     });
37     var chartZ = new CanvasJS.Chart("divChartZ", {
38         axisX:{ valueFormatString: "hh:mm:ss"},
39         axisY: {valueFormatString: "#0mg"},
40         includeZero: true,
41         data: [{
42             type: "line",
43             color: "darkBlue",
44             dataPoints: dpsZ
45         }]
46     });
```

On créé des variables pour définir l'intervalle de rafraichissement que l'on initialise à 200ms.

Le nombre de points visibles dans le graphe initialisé à 3000 (ligne 5 à 15). Il faut trouver un bon équilibre de point affiché pour une bonne compréhension et une bonne lecture des graphiques, et les tests ont montré que 3000 était un nombre correct.

On instancie les graphs avec « new » et ils sont de type CanvasJS.

Une fois un graph instancié, on indique toutes ses caractéristiques : le nom des axes X et Y, le type de graphique (ligne, colonne, circulaire, etc) et sa couleur. Les variables *dpsX*, *dpsY* et *dpsZ* sont des tableaux (array) et elles vont stocker les points qui seront rendus sur le graphique.

```
48      var updateChart = function () {
49          var date = new Date();
50          var time = date.getTime();
51
52          for (var j = 0; j < nbVal; j++) {
53              yValX +=  Math.round(5 -10*Math.random());
54              yValY +=  Math.round(5 -10*Math.random());
55              yValZ +=  Math.round(5 -10*Math.random());
56              dpsX.push({ x: date, y: yValX });
57              dpsY.push({ x: date, y: yValY });
58              dpsZ.push({ x: date, y: yValZ });
59              nbPoints++;
60              time += 10;                 // ajout de 10ms, acquisition à 100 Hz
61              date = new Date(time);
62          };
63          if (dpsX.length > dataLength) {
64              nbEnTrop = dpsX.length - dataLength;
65              for (var i = 0; i < nbEnTrop; i++) {
66                  dpsX.shift();
67                  dpsY.shift();
68                  dpsZ.shift();
69              }
70          }
71          chartX.render();
72          chartY.render();
73          chartZ.render();
74      };
75      // pour des graphes fixes (test)
76      // updateChart();
77      // mise à jour cyclique (temps réel) des datas
78      setInterval(function(){updateChart();}, updateInterval);
79  };
```

On créé ensuite une fonction qui va permettre de mettre à jour le graphique. Dans notre cas, les nouvelles données qui rentrent dans les graphs sont générées aléatoirement grâce à *Math.random* et sont arrondi grâce à *Math.round* qui retourne la valeur d'un nombre arrondi à l'entier le plus proche (ligne 53 à 55).

On insère les nouveaux points grâce à *push* (on lui donne comme paramètre les coordonnées sur l'axe X et Y). On incrémente donc le nombre total de point de un et on met à jour la date en incrémentant le temps de 10ms (ligne 56 à 61).

On place dans la boucle *for* une condition Si le nombre de point est supérieur au nombre de point visible sur le graphique, on utilise la méthode *shift* pour retirer les points à gauche du graphique (les premiers rentrés).

Après être passé par toutes ces étapes, on fait le rendu de chaque graphique avec *render*.

Actuellement en période de test, on ne reçoit aucune valeur pour mettre à jour les graphs, on fait donc ligne 78 une méthode qui va répéter la fonction d'update des graphs pour l'intervalle donné (dans notre cas défini à 200ms).

b) Code HTML

```
1   <!DOCTYPE HTML>
2   <html>
3       <head>
4           <title>Images</title>
5           <meta charset="utf-8">
6           <meta name="viewport" content="width=device-width, initial-scale=1">
7           <script type="text/javascript" src="../js/canvasjs.min.js"></script>
8           <script type="text/javascript" src="LGraphes.js"></script>
9       </head>
10
11      <body>
12          <div>
13              <div id="divChartX" style="height: 150px; width:100%;"> </div>
14              <div id="divChartY" style="height: 150px; width:100%;"> </div>
15              <div id="divChartZ" style="height: 150px; width:100%;"> </div>
16          </div>
17      </body>
18  </html>
```

Figure 3 : code de la page HTML

Pour tester ces graphs, on les implémente dans un page HTML grâce au code ci-dessus.

On importe la librairie CanvasJS ligne 7 et notre fichier JavaScript sur lequel ce trouve nos graphs ligne 8.

Dans le corps de la page, on appelle les graphs par leur id, et on définit leur style (hauteur en pixel et largeur en %).

c) Exécution de la page HTML

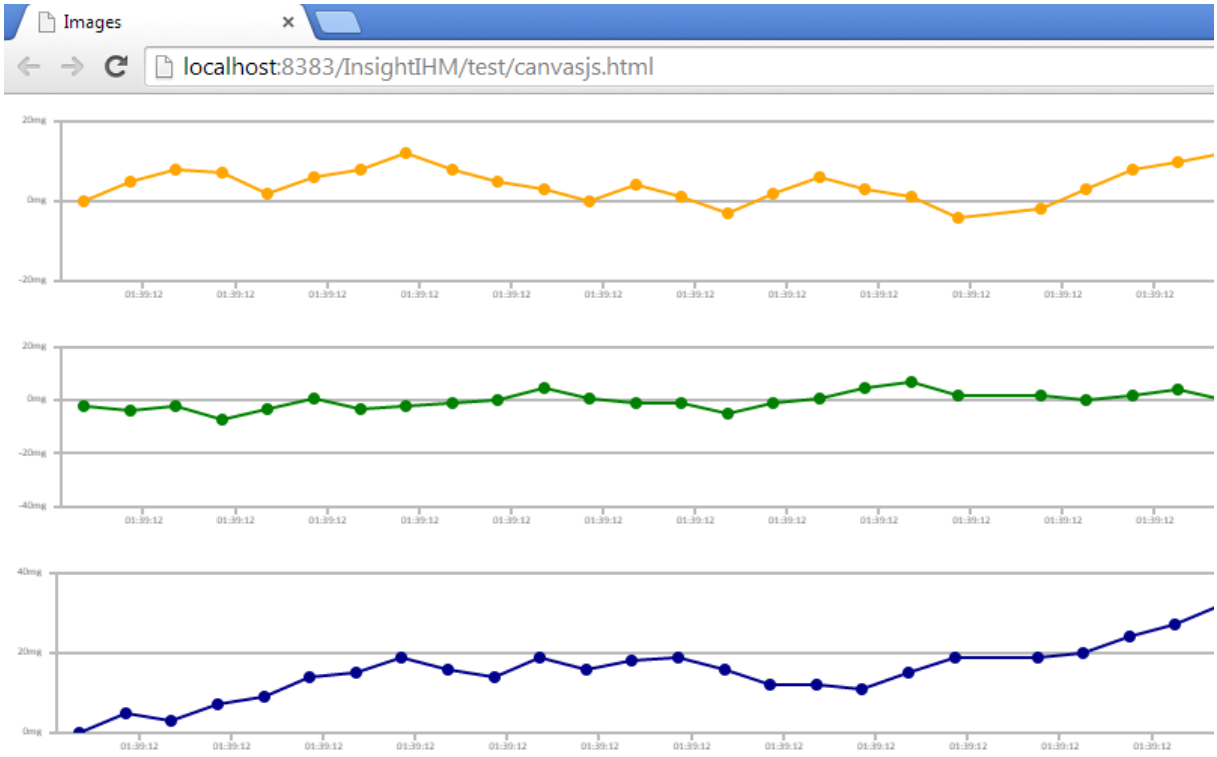

Figure 4 : affichage de la page HTML

Les trois graphiques se sont bien générés, chacun avec des données aléatoires. Si on laisse la page ouverte on voit bien que les graphs se mettent à jour et insère constamment de nouvelles données.

La lecture du graphique et d'un point particulier peut être compliquée lorsque beaucoup de valeur sont affichées. Pour remédier à ce problème, une solution est fournie dans la librairie CanvasJS : la tooltip.

La tooltip est incluse par défaut dans les graphs, mais elle peut être modifiée si nécessaire.

Figure 5 : tooltip

III. Conclusion

Bien qu'il existe une multitude de solution pour la création de graphique, il a fallu trouver un moyen qui allie simplicité, efficacité et performance. Comme nous le démontre cette validation, CanvasJS répond bien à ces trois critères grâce à sa librairie JavaScript.