Utiliser Chart.js pour afficher des graphiques avec Vue.js

Exemple simple

Si vous n'êtes pas familier avec Chart.js , cela vaut la peine de l'examiner. C'est un moyen puissant et simple de créer des graphiques propres avec l'élément `<canvas>` HTML5. Ne vous inquiétez pas, vous n'avez pas besoin de savoir quoi que ce soit sur le `<canvas>`pour utiliser Chart.js. Avec l'objet Vue `data()`, il est facile de stocker nos données et de les manipuler pour changer notre graphique en cas de besoin.

Installez Chart.js

La première chose à faire est de créer une application Vue.js à l'aide du modèle webpack`-simple` et d'installer Chart.js.

```
$ vue init webpack-simple <project-name>
$ cd <project-name>
$ npm install
$ npm install chart.js --save
```

Accédez à votre fichier `App.vue` et supprimez tout le code généré. Le graphique `<canvas>` sera dans l'élément `#app` (à la racine). Ensuite, importez Chart.js en utilisant ES6 dans votre composant Vue.

App.vue

```
import Chart from 'chart.js';
```

Créer le graphique

Ce graphique va avoir deux ensembles de données :

1) Le nombre de lunes par planète dans notre système solaire
2) La masse globale de chaque planète.

Avec ces deux ensembles de données, nous pouvons avoir différents types de graphiques pour afficher les corrélations dans les données.

Chaque graphique Chart.js doit avoir un `<canvas>` dans le balisage HTML. Le `id` du graphique est utilisé comme sélecteur pour lier le JavaScript à celui-ci.

App.vue

```
<template>
  <div id="app">
    <canvas id="planet-chart"></canvas>
  </div>
</template>
```

Structure du graphique

Dans sa forme la plus simple, chaque graphique à la même structure de base:

```
const ctx = document.getElementById('planet-chart');
const myChart = new Chart(ctx, {
  type: '',
  data: [],
  options: {},
});
```

Vous pouvez commencer par ajouter vos données à cet objet `Chart` et continuer à répéter ce processus pour chaque nouveau graphique que vous souhaitez créer. Cependant, ce processus peut être beaucoup plus facile si nous avions une fonction dans laquelle passer des arguments.

Commencez par créer une nouvelle fonction dans l'objet `methods` de votre composant et donnez-lui deux paramètres, `chartId` et `chartData`.

App.vue

```
methods: {
  createChart(chartId, chartData) {
    const ctx = document.getElementById(chartId);
    const myChart = new Chart(ctx, {
      type: chartData.type,
      data: chartData.data,
      options: chartData.options,
    });
  }
}
```

Chart.js a tendance à avoir beaucoup de code. Ce graphique simple de « planètes », par exemple, contient au moins 50 lignes de code. Imaginez avoir plusieurs graphiques avec des données complexes.
Votre composant Vue à fichier unique peut devenir rapidement volumineux et déroutant. Utilisons donc ES6 pour l' `import` des données de notre graphique, afin de garder notre composant Vue léger et concentré.

Création des données du graphique

Créez un nouveau fichier `.js` dans le répertoire `src` (à la racine). Nommez-le `chart-data.js`. Cependant, vous pouvez le nommer comme vous le souhaitez. Créez une `const` et nommez-la `planetChartData`.

Gardez à l'esprit que vous voudrez lui donner un nom unique et descriptif basé sur les données. Vous pouvez avoir plusieurs objets de données dans ce fichier pour différents graphiques.

chart-data.js

```
export const planetChartData = {
  type: 'line',
  data: {
    labels: ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus',
'Neptune'],
    datasets: [
      { // one line graph
        label: 'Number of Moons',
        data: [0, 0, 1, 2, 67, 62, 27, 14],
        backgroundColor: [
          'rgba(54,73,93,.5)', // Blue
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)',
          'rgba(54,73,93,.5)'
        ],
        borderColor: [
          '#36495d',
          '#36495d',
          '#36495d',
          '#36495d',
```

```
          '#36495d',
          '#36495d',
          '#36495d',
          '#36495d',
        ],
        borderWidth: 3
      },
      { // another line graph
        label: 'Planet Mass (x1,000 km)',
        data: [4.8, 12.1, 12.7, 6.7, 139.8, 116.4, 50.7, 49.2],
        backgroundColor: [
          'rgba(71, 183,132,.5)', // Green
        ],
        borderColor: [
          '#47b784',
        ],
        borderWidth: 3
      }
    ]
  },
  options: {
    responsive: true,
    lineTension: 1,
    scales: {
      yAxes: [{
        ticks: {
          beginAtZero: true,
          padding: 25,
        }
      }]
    }
  }
}

export default planetChartData;
```

Remarque : Vous pouvez consulter la documentation de Chart.js pour plus d'informations sur les graphiques en ligne, ainsi que d' autres comme `bar`, `polarArea`, `radar`, `pie` et `doughnut`.

En exportant `planetChartData`, vous autorisez l'importation de la `const` dans un autre fichier JavaScript. Plus important encore, vous séparez les données du composant. Cela le rend beaucoup plus facile à gérer et pour créer un nouveau graphique avec de nouvelles données à l'avenir.
Importez les données de votre graphique dans votre composant `App.vue`.

App.vue

```
import planetChartData from './chart-data.js';
```

Ensuite, stockez les données du graphique unique dans la fonction `data()` de Vue .

App.vue

```
data() {
  return {
    planetChartData: planetChartData,
  }
}
```

Remarque: vous pouvez également utiliser le raccourci ES6. Étant donné que la propriété de données et la valeur ont le même nom, vous pouvez simplement utiliser `planetChartData` à la place de `planetChartData: planetChartData`.

Initialisation du graphique

À ce stade, Chart.js doit être installé et les données du graphique doivent être importées dans le composant `App.vue`. Dans l'objet `methods`, vous avez également ajouté une fonction qui crée l'objet de graphique avec les données du fichier `chart-data.js`.

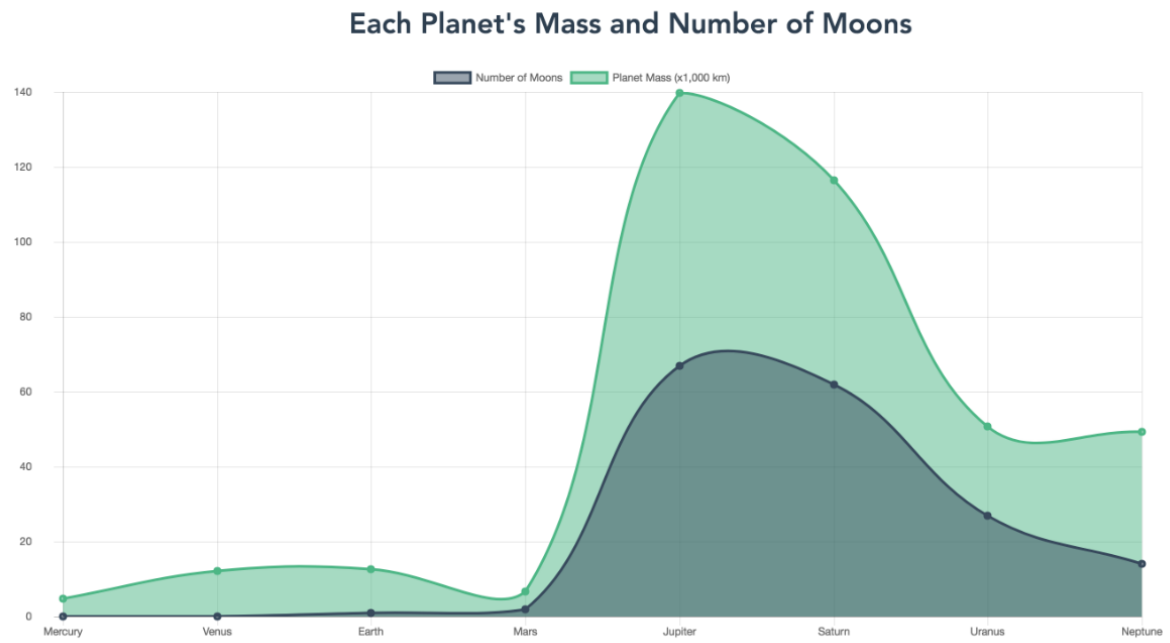
Vous devriez déjà avoir un élément `<canvas>` créé dans le modèle du composant. À ce stade, il est temps d'initialiser le graphique et d'écrire dans le fichier `<canvas>`.

Pour ce faire, vous devez exécuter la `createChart()`fonction dans la méthode `mounted()` du cycle de vie du composant. Cette fonction prend deux arguments; la `chartId`(chaîne) et `chartData`(objet dans nos données à partir de `chart-data.js`).

App.vue

```
mounted() {
  this.createChart('planet-chart', this.planetChartData);
}
```

Le graphique devrait être rendu maintenant lorsque le composant est monté!



Comme vous pouvez le voir, nous pouvons nous concentrer sur nos données et laisser **Chart.js** faire le travail.

Graphiques mixtes

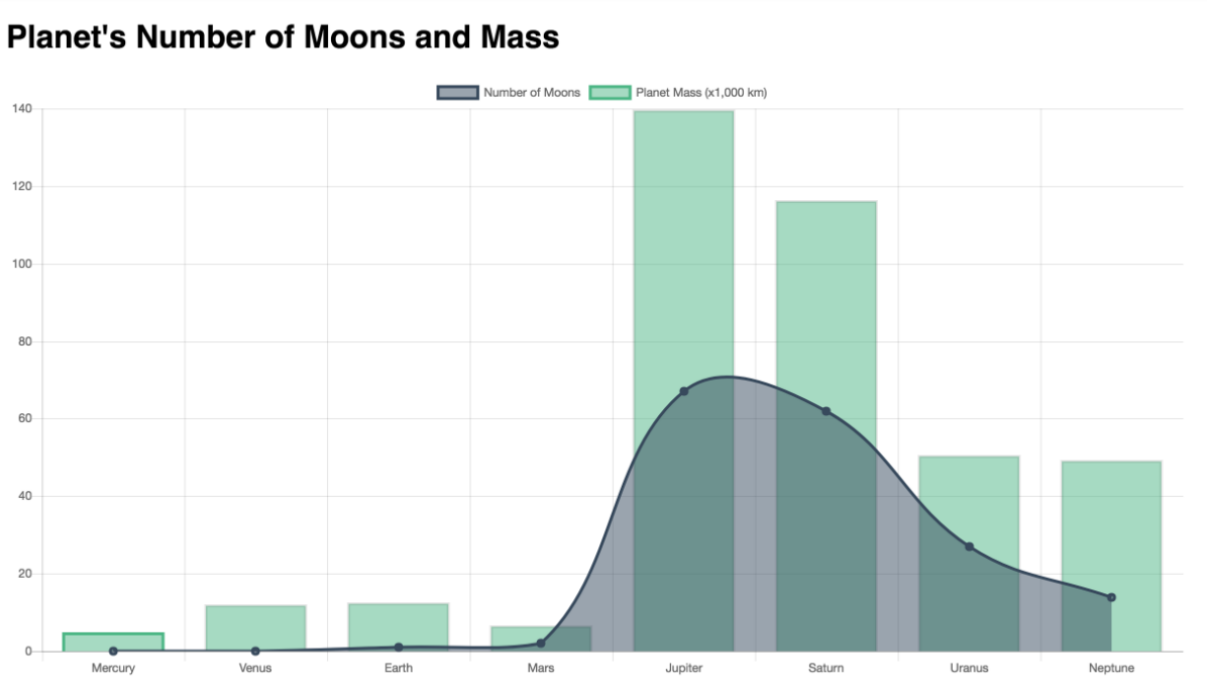
Chart.js prend également en charge les graphiques mixtes. Poursuivons avec le graphique planètes que vous avez créé ci-dessus, montrons ces mêmes données avec deux types de graphiques.
Ouvrez votre fichier `chart-data.js` et modifions la propriété `types` de notre graphique et `datasets`. Recherchez la propriété `type` des données de votre graphique et modifiez-la en `bar`. À ce stade, les deux graphiques seront en bar. Cependant, nous voulons que le graphique montre une barre et un graphique linéaire.
Pour changer cela, dans chaque objet `dataset`, ajoutez une propriété `type` en dessous de la propriété `label`. Pour le premier objet `dataset`, donnez-lui une propriété `type` avec une valeur de `line` et pour le second, donnez-lui une propriété `type` avec une valeur de `bar`.

chart-data.js

```
data: {
  type: 'bar', // was "line"
  labels: ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus',
'Neptune'],
  datasets: [
    {
      label: 'Number of Moons',
      type: 'line', // Add this
```

```
      data: [...],
      backgroundColor: [...],
      borderColor: [...],
      borderWidth: 3
    },
    {
      label: 'Planet Mass (x1,000 km)',
      type: 'bar', // Add this
      data: [...],
      backgroundColor: [...],
      borderColor: [...],
      borderWidth: 3
    }
  ]
}
```

Une fois vos composants montés, vous devriez voir quelque chose comme ceci:



Ce message ne fait qu'effleurer la surface de ce que vous pouvez faire avec **Chart.js**. Vous devez avoir une compréhension de base sur la façon d'utiliser **Chart.js** avec **Vue.js** en séparant vos données avec les importations ES6 .