Utiliser Axios pour consommer des API avec Vue.js

Exemple simple

Lors de la création d'une application Web, il est fréquent que vous souhaitiez utiliser et afficher les données provenant d'une API. Il existe plusieurs manières de le faire, mais une approche très populaire consiste à utiliser **axios**, un client HTTP basé sur les Promesses.

Créez un nouveau projet nommé « demoaxios » avec une structure simple : une page **index.html**, une page **script.js** et une page **style.css**.
Ajouter le CDN de vue :

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

Dans cet exemple, nous allons utiliser l'**API CoinDesk** pour afficher les prix du Bitcoin qui sont mis à jour toutes les minutes. Premièrement, nous devons installer axios avec npm/yarn ou à partir d'un lien CDN :

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

Il existe plusieurs manières d'interroger une API, mais il est préférable de d'abord connaitre la structure des données qu'elle renvoie afin de savoir ce qu'elle va afficher. Pour ce faire, nous allons appeler le point de terminaison de l'API et afficher le résultat afin que nous puissions connaître sa structure et son contenu. Nous pouvons voir dans la documentation de l'API de CoinDesk que l'appel doit être effectué à l'adresse :

```
https://api.coindesk.com/v1/bpi/currentprice.json
```

Nous allons donc commencer par créer une donnée qui gardera nos informations, puis nous récupérerons les données et les attribuerons à l'aide de l'étape `mounted` du cycle de vie :

script.js

```
new Vue({
  el: "#app",
  data() {
    return {
      info: null
    };
  },
  mounted() {
    axios
      .get("https://api.coindesk.com/v1/bpi/currentprice.json")
      .then(response => (this.info = response))
  },
});
```

index.html

```
<div id="app">
   {{info}}
</div>
```

Nous obtenons ceci :


```
{ "data": { "time": { "updated": "Dec 31, 2020 10:41:00 UTC", "updatedISO": "2020-12-31T10:41:00+00:00", "updateduk": "Dec 31, 2020 at 10:41 GMT" }, "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org", "chartName": "Bitcoin", "bpi": { "USD": { "code": "USD", "symbol": "&#36;", "rate": "29,161.6550", "description": "United States Dollar", "rate_float": 29161.655 }, "GBP": { "code": "GBP", "symbol": "&pound;", "rate": "21,350.3849", "description": "British Pound Sterling", "rate_float": 21350.3849 }, "EUR": { "code": "EUR", "symbol": "&euro;", "rate": "23,748.7852", "description": "Euro", "rate_float": 23748.7852 } } }, "status": 200, "statusText": "", "headers": { "cache-control": "max-age=15", "content-length": "678", "content-type": "application/javascript", "expires": "Thu, 31 Dec 2020 10:42:07 UTC" }, "config": { "transformRequest": {}, "transformResponse": {}, "timeout": 0, "xsrfCookieName": "XSRF-TOKEN", "xsrfHeaderName": "X-XSRF-TOKEN", "maxContentLength": -1, "headers": { "Accept": "application/json, text/plain, */*" }, "method": "get", "url": "https://api.coindesk.com/v1/bpi/currentprice.json" }, "request": {} }
```


Parfait ! Nous avons des données. Mais cela semble assez désordonné pour le moment, alors affichons-les correctement et ajoutons un traitement d'erreur si les choses ne fonctionnent pas comme prévu, ou s'il faut plus de temps que nécessaire pour obtenir les informations.

Exemple concret : l'utilisation des données

Affichage des données d'une API

Il est assez courant que les informations dont nous avons besoin se trouvent dans la réponse. Nous devons parcourir ce que nous venons de stocker pour y accéder correctement. Dans notre cas, nous pouvons voir que les informations de prix dont nous avons besoin sont stockées dans `response.data.bpi`. Si nous l'utilisons, notre sortie sera :

script.js

```
axios
   .get("https://api.coindesk.com/v1/bpi/currentprice.json")
   .then(response => (this.info = response.data.bpi))
```

Résultat :


```
{ "USD": { "code": "USD", "symbol": "&#36;", "rate": "29,161.6550", "description":
"United States Dollar", "rate_float": 29161.655 }, "GBP": { "code": "GBP",
"symbol": "&pound;", "rate": "21,350.3849", "description": "British Pound
Sterling", "rate_float": 21350.3849 }, "EUR": { "code": "EUR", "symbol": "&euro;",
"rate": "23,748.7852", "description": "Euro", "rate_float": 23748.7852 } }
```


C'est beaucoup plus facile à afficher ; nous pouvons donc mettre à jour notre code HTML pour n'afficher que les informations dont nous avons besoin à partir des données reçues. Pour ce faire, nous allons créer un **filtre** pour nous assurer que la décimale se trouve également à la place appropriée.

Index.html

```
<div id="app">
    <div v-for="currency in info">
       {{ currency.description }} :
      <span> {{ currency.rate_float | currencydecimal }}
         <span v-html="currency.symbol"></span>
      </span>
    </div>
</div>
```

script.js

```
filters: {
    currencydecimal(value) {
      return value.toFixed(2);
    },
  },
```

Travailler avec des erreurs

Parfois, nous ne pouvons pas recevoir de données de l'API. Il peut y avoir de nombreuses raisons pour qu'un appel puisse échouer. Par exemple :

- L'API est hors-service.
- La requête a mal été réalisée.
- L'API ne nous donne pas les informations dans le format attendu.

Quand nous créons cette requête, nous devrions vérifier si de tels cas se produisent et nous informer pour traiter ce problème. Dans un appel axios, nous pouvons le faire en utilisant `catch`.

Cela nous permettra de savoir si quelque chose a échoué lors de la requête à l'API, mais que se passerait-il si les données sont endommagées ou si l'API est en panne ? Pour l'instant, l'utilisateur ne verra rien. Nous devrions peut-être créer un loader pour ce cas, puis informer l'utilisateur si nous ne pouvons pas obtenir les données.

script.js

```
new Vue({
  el: "#app",
  data() {
    return {
      info: null,
      loading: true,
      errored: false
    };
  },
  filters: {
    currencydecimal(value) {
      return value.toFixed(2);
    },
  },
  mounted() {
    axios
      .get("https://api.coindesk.com/v1/bpi/currentprice.json")
      .then(response => { this.info = response.data.bpi })
      .catch(error => { console.log(error); this.errored = true })
      .finally(() => this.loading = false)
  },
});
```

Index.html

```
<div id="app">
        <div class="vignette">
            <h1>Index des prix du Bitcoin</h1>
            <section v-if="errored">
                <p>Nous sommes désolés, nous ne sommes pas en mesure de récupé
rer ces informations pour le moment.
                    Veuillez réessayer ultérieurement. </p>
            </section>
            <section v-else>
                <div v-if="loading">Chargement...</div>
                <div v-else v-
for="currency in info" class="monnaie"> {{ currency.description }} : <span
                        class="valeur">{{
                        currency.rate_float | currencydecimal }} <span v-
html="currency.symbol"></span></span> </div>
            </section>
        </div>
    </div>
```

Style.css

```
h1{
    margin-left: auto;
    margin-right: auto;
}
.vignette{
    width: 380px;
    background-color: rgb(39, 1, 1);
    color:lightgray;
    padding:5px 0px 20px 20px;
    border-radius: 20px;
    margin-left: auto;
    margin-right: auto;
}
.monnaie{
    color:grey;
    font-size: 20px;
}
.valeur{
    color:beige
}
```

Vous pouvez appuyer sur le bouton de réexécution de cet exemple pour connaitre brièvement l'état du chargement pendant la collecte des données à partir de l'API :

Cela peut encore être amélioré avec l'utilisation de composants pour différentes sections et un rapport d'erreurs plus distinct, en fonction de l'API utilisée et de la complexité de votre application.