Consumiendo APIs

Ing. Luis Guillermo Molero Suárez

Usando Axios para consumir APIs

Ejemplo Base

En muchas ocasiones, al construir una aplicación web, se quiere consumir y mostrar datos de un API. Existen diversas formas de hacerlo, pero un enfoque muy popular es utilizar **axios**, un client HTTP basado en promesas.

En este ejercicio, se utilizará la **API CoinDesk** para mostrar los precios de Bitcoin, que se actualizan minuto a minuto. Primero, se instalará axios mediante npm/yarn o a través de un enlace CDN.
Hay distintas formas de solicitar información del API, pero es conveniente conocer primero la forma de los datos en pos de entender que mostrar. Para ello, se realizará una llamada al endpoint en la API y se mostrará la respuesta, con tal de analizarla. Podrá comprobar en la documentación del API de CoinDesk, que esta llamada se realizará a `https://api.coindesk.com/v1/bpi/currentprice.json`.

Comience creando una propiedad data que eventualmente almacenará la información deseada. A continuación, se realiza el pedido y asigna la respuesta utilizando el lifecycle hook `mounted`:

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

Se obtiene lo siguiente:

¡Excelente! Tiene algunos datos, pero lucen bien desordenados, así que a continuación los mostrará apropiadamente. Además, incluirá manejo de errores en caso de que las cosas no funcionen o demoren en obtener la información más de lo esperado.

Ejemplo del mundo real: Trabajando con los datos

Mostrando datos de un API

Es bastante común que la información buscada se encuentre dentro de la respuesta y sea necesario recorrer el objeto recién almacenado, con tal de acceder a ella apropiadamente. En este caso, note que la información de precios se encuentra en `response.data.bpi`. Si utiliza esta propiedad, en su lugar, obtiene lo siguiente:

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

Esto es mucho más sencillo de utilizar, por tanto, ahora podrá actualizar el HTML para mostrar solo la información necesaria de los datos recibidos. Creará, además, un **filtro** para asegurar que el decimal se encuentre en el lugar apropiado.

```
<div id="app">
  <h1>Listado de Precio de Bitcoin</h1>
  <div
    v-for="currency in info"
    class="currency"
  >
    {{ currency.description }}:
    <span class="lighten">
      <span v-html="currency.symbol"></span>{{ currency.rate_float | currencydecimal }}
    </span>
  </div>
</div>
filters: {
  currencydecimal (value) {
    return value.toFixed(2)
  }
},
```

Manejando errores

Hay ocasiones en las que no obtendrá los datos necesarios del API. Existen múltiples razones por las cuales la llamada axios puede fallar, incluyendo, pero no limitadas a:

- El API esta fuera de servicio.
- La llamada se realizó incorrectamente.
- El API no devuelve la información en el formato anticipado.

Cuando realiza esta llamada debe, comprobar la ocurrencia de tales circunstancias y, proporcionar información en cada caso, de forma tal que sepa cómo manejar el problema. En una llamada axios, ello se logra utilizando `catch`.

```
axios
  .get('https://api.coindesk.com/v1/bpi/currentprice.json')
  .then(response => (this.info = response.data.bpi))
  .catch(error => console.log(error))
```

¿Esto permitirá saber si algo fallo durante la llamada a la API, pero que ocurre si los datos están estropeados o la API está fuera de servicio? Por ahora, el usuario simplemente no verá nada. Por ello se quisiera incluir un cargador y notificar al usuario si no es factible obtener datos en lo absoluto.

```
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)
  }
})
```

```
<div id="app">
  <h1>Listado de Precio de Bitcoin</h1>

  <section v-if="errored">
    <p>Lo sentimos, no es posible obtener la información en este momento, por favor intente nuevamente más tarde</p>
  </section>

  <section v-else>
    <div v-if="loading">Cargando...</div>

    <div
      v-else
      v-for="currency in info"
      class="currency"
    >
      {{ currency.description }}:
      <span class="lighten">
        <span v-html="currency.symbol"></span>{{ currency.rate_float | currencydecimal }}
      </span>
    </div>

  </section>
</div>
```

Puede accionar el botón rerun en esta nota para ver el estado de cargando brevemente, mientras se solicitan los datos al API:

Esto puede ser mejorado, utilizando componentes para las diferentes secciones y más errores distintos, en dependencia del API empleada y la complejidad de la aplicación.

Patrones alternativos

API Fetch

La **API Fetch** es una potente API nativa para este tipo de llamadas. ¡Puede que haya escuchado que uno de los beneficios del API Fetch es que no necesita cargar un recurso externo para utilizarla, lo cual es cierto! Excepto que… no es totalmente soportada aún, por tanto, todavía necesita utilizar un polyfill. Existen además otros gotchas al trabajar con esta API, por lo cual muchos prefieren usar axios por ahora. Esto bien que podría cambiar en el futuro.