Vue3

Ibrahim ALAME

27 décembre 2023

0.1 Introduction et requête AJAX

0.1.1 Rappels sur AJAX

L'AJAX est une pratique de permettant de mettre à jour simplement des parties du DOM d'une page HTML au lieu de devoir recharger la page entière.

AJAX permet également d'exécuter du code de manière asynchrone, c'est-à-dire que votre code continue à s'exécuter pendant qu'une ou plusieurs requêtes sont en cours.

N'hésitez pas à revoir le chapitre réseau de la formation JavaScript si tout cela n'est pas parfaitement clair pour vous.

0.1.2 Rappels sur fetch

fetch est une Web API disponible dans tous les navigateurs qui permet d'envoyer des requêtes HTTP. La syntaxe de l'API est très simple :

```
const promesse = fetch(url, [options]);
```

1. Le premier paramètre est l'URL cible de la requête.
2. Le second paramètre est un objet d'options que nous étudierons en détails.
3. La Web API retourne une promesse qui sera tenue si le serveur répond.
4. La promesse est résolue avec un objet Response.

A ce stade, vous pouvez accéder aux propriétés suivantes :

- url : url de la requête.
- redirected : booléen indiquant si la requête a été redirigée par le serveur.
- status : code du statut de la requête.
- ok : booléen pour savoir si la requête s'est bien déroulée (true si le code du statut est compris entre 200 et 299).
- type : type de la requête cors ou basic (nous y reviendrons).
- statusText : message du statut de la requête.
- headers : headers de la réponse. Il faut utiliser la méthode get() et passer le nom du header à récupérer.

0.1.3 Le service Dyma restapi.fr

Nous allons utiliser un service de Dyma pour le backend qui permet notamment de sauvegarder des données facilement en utilisant un véritable service REST.

Ce service est www.restapi.fr et il permet de sauvegarder et de manipuler des données facilement. Il permet également de générer des données suivant un modèle que vous définissez.

0.2 Requêtes POST

0.2.1 Effectuer une requête POST avec fetch

Pour effectuer une requête de type POST il faut utiliser deux arguments. Voici un exemple de requête :

```
const utilisateur = {
  name: "Paul Dupont"
};
const reponse = await fetch("https:/restapi.fr/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(utilisateur)
});
```

Nous devons préciser dans l'objet options les propriétés suivantes :

- method : La méthode de la requête, par exemple `DELETE` ou `POST`.
- headers : Les entêtes à ajouter à la requête. Il faut préciser nécessairement le type des données envoyées. Ici nous envoyons des données au format JSON nous spécifions donc `"Content-Type": "application/json"`.
- body : Le contenu de la requête `POST`. Il doit être au format que nous précisons dans l'entête `Content-Type`, donc en JSON. Nous devons donc utiliser la méthode `JSON.stringify()` pour transformer notre objet JavaScript en `JSON`.

0.2.2 Code de la vidéo

Voici le code de la vidéo :

```
<template>
  <div class="container">
    <div class="p-20">
      <h3 class="mb-10">Formulaire</h3>
      <form @submit="mySubmit">
        <input
          v-model="nameValue"
          class="mr-10"
          type="text"
          placeholder="Prénom"
        />
        <input
          v-model="emailValue"
          class="mr-10"
          type="text"
          placeholder="Email"
        />
        <button class="btn btn-primary">Sauvegarder</button>
      </form>
    </div>
    <div class="p-20">
      <h3>Liste des utilisateurs</h3>
      <ul>
        <li v-for="user in state.users">
          <p>{{ user.name }} - {{ user.email }}</p>
        </li>
      </ul>
```

```
    </div>
  </div>
</template>

<script setup lang="ts">
import { useForm, useField } from 'vee-validate';
import { reactive } from 'vue';

interface User {
  name: string;
  email: string;
  createdAt?: string;
  _id?: string;
}

const state = reactive<{ users: User[] }>({
  users: [],
});

const { handleSubmit, resetForm } = useForm();

const mySubmit = handleSubmit(async (value) => {
  try {
    const response = await fetch('https://restapi.fr/api/vueuser', {
      method: 'POST',
      body: JSON.stringify(value),
      headers: {
        'Content-Type': 'application/json',
      },
    });
    const user: User = await response.json();
    state.users.push(user);
    resetForm();
  } catch (err) {
    console.error(err);
  }
});

const { value: emailValue } = useField('email');
const { value: nameValue } = useField('name');
</script>

<style lang="scss">
@import './assets/scss/base.scss';
</style>
```

0.3 Requêtes GET et DELETE

0.3.1 Envoyer une requête GET

Sans passer d'option, fetch va effectuer une requête HTTP avec la méthode `GET` :

```
const reponse = await fetch("https://restapi.fr/users");
```

Pour lire le contenu de la réponse, il faut la parser, c'est-à-dire attendre que tout le body soit reçu et le lire dans un format particulier. Dans une application, le plus courant sera de parser la réponse au format `JSON` avec la méthode `json()` :

```
const reponse = await fetch("https://jsonplaceholder.typicode.com/users");
const donnees = await reponse.json();
```

0.3.2 Envoyer une requête DELETE

Pour envoyer une requête HTTP de type `DELETE`, il suffit de préciser cette méthode dans l'objet d'options passé à `fetch()` :

```
await fetch(`https://restapi.fr/api/users?id=${userId}`, {
  method: 'DELETE',
});
```

0.3.3 Code de la vidéo

Voici le code de la vidéo :

```
<template>
  <div class="container">
    <div class="p-20">
      <h3 class="mb-10">Formulaire</h3>
      <form @submit="mySubmit">
        <input
          v-model="nameValue"
          class="mr-10"
          type="text"
          placeholder="Prénom"
        />
        <input
          v-model="emailValue"
          class="mr-10"
          type="text"
          placeholder="Email"
        />
        <button class="btn btn-primary">Sauvegarder</button>
      </form>
    </div>
    <div class="p-20">
      <h3>Liste des utilisateurs</h3>
      <ul>
```

```
        <li v-for="user in state.users">
          <p class="mr-10">{{ user.name }} - {{ user.email }}</p>
          <button
            @click="deleteUser(user._id)"
            type="button"
            class="btn btn-danger"
          >
            Supprimer
          </button>
        </li>
      </ul>
    </div>
  </div>
</template>

<script setup lang="ts">
import { useForm, useField } from 'vee-validate';
import { reactive } from 'vue';

interface User {
  name: string;
  email: string;
  createdAt?: string;
  _id?: string;
}

const state = reactive<{ users: User[] }>({
  users: [],
});

const { handleSubmit, resetForm } = useForm();

const mySubmit = handleSubmit(async (value) => {
  try {
    const response = await fetch('https://restapi.fr/api/vueusers', {
      method: 'POST',
      body: JSON.stringify(value),
      headers: {
        'Content-Type': 'application/json',
      },
    });
    const user: User = await response.json();
    state.users.push(user);
    resetForm();
  } catch (err) {
    console.error(err);
  }
});

const { value: emailValue } = useField('email');
const { value: nameValue } = useField('name');
```

```
async function fetchUsers() {
  try {
    const response = await fetch('https://restapi.fr/api/vueusers');
    const users: User | User[] = await response.json();
    if (users) {
      state.users = Array.isArray(users) ? users : [users];
    }
  } catch (err) {
    console.error(err);
  }
}

async function deleteUser(userId?: string) {
  try {
    if (userId) {
      await fetch(`https://restapi.fr/api/vueusers?id=${userId}`, {
        method: 'DELETE',
      });
      state.users = state.users.filter((user) => user._id !== userId);
    }
  } catch (err) {
    console.error(err);
  }
}

fetchUsers();
</script>

<style lang="scss">
@import './assets/scss/base.scss';
</style>
```

0.4 Requêtes PATCH

0.4.1 Requête PATCH

La requête PATCH est très similaire à une requête POST. Il faut bien sûr passer au serveur l'_id de la ressource à modifier et utiliser la méthode PATCH.

```
const response = await fetch(
  `https://restapi.fr/api/vueusers?id=${state.selectedUser._id}`,
  {
    method: 'PATCH',
    body: JSON.stringify(value),
    headers: {
      'Content-Type': 'application/json',
    },
  }
);
```

```
const user: User = await response.json();
```

0.4.2 Code de la vidéo

Voici le code de la vidéo :

```
<template>
  <div class="container">
    <div class="p-20">
      <h3 class="mb-10">Formulaire</h3>
      <form @submit="mySubmit">
        <input
          ref="name"
          v-model="nameValue"
          class="mr-10"
          type="text"
          placeholder="Prénom"
        />
        <input
          v-model="emailValue"
          class="mr-10"
          type="text"
          placeholder="Email"
        />
        <button class="btn btn-primary">Sauvegarder</button>
      </form>
    </div>
    <div class="p-20">
      <h3>Liste des utilisateurs</h3>
      <ul>
        <li
          @click="state.selectedUser = user"
          class="mb-10 d-flex"
          v-for="user in state.users"
        >
          <span class="mr-10 flex-fill"
            >{{ user.name }} - {{ user.email }}</span
          >
          <button
            @click.stop="deleteUser(user._id)"
            type="button"
            class="btn btn-danger"
          >
            Supprimer
          </button>
        </li>
      </ul>
    </div>
  </div>
</template>
```

```
<script setup lang="ts">
import { useForm, useField } from 'vee-validate';
import { reactive, ref, watch, onMounted } from 'vue';

interface User {
  name: string;
  email: string;
  createdAt?: string;
  _id?: string;
}

const state = reactive<{
  users: User[];
  selectedUser: User | null;
}>({
  users: [],
  selectedUser: null,
});
const name = ref<HTMLInputElement | null>(null);

onMounted(() => name.value?.focus());

const { handleSubmit, resetForm } = useForm();

const mySubmit = handleSubmit(async (value) => {
  try {
    if (state.selectedUser) {
      const response = await fetch(
        `https://restapi.fr/api/vueusers?id=${state.selectedUser._id}`,
        {
          method: 'PATCH',
          body: JSON.stringify(value),
          headers: {
            'Content-Type': 'application/json',
          },
        }
      );
      const user: User = await response.json();
      state.users = state.users.map((u) => (u._id === user._id ? user : u));
      state.selectedUser = null;
    } else {
      const response = await fetch('https://restapi.fr/api/vueusers', {
        method: 'POST',
        body: JSON.stringify(value),
        headers: {
          'Content-Type': 'application/json',
        },
      });
      const user: User = await response.json();
      state.users.push(user);
    }
```

```
    resetForm();
    name.value?.focus();
  } catch (err) {
    console.error(err);
  }
});

const { value: emailValue } = useField('email');
const { value: nameValue } = useField('name');

async function fetchUsers() {
  try {
    const response = await fetch('https://restapi.fr/api/vueusers');
    const users: User | User[] = await response.json();
    if (users) {
      state.users = Array.isArray(users) ? users : [users];
    }
  } catch (err) {
    console.error(err);
  }
}

fetchUsers();

async function deleteUser(userId?: string) {
  try {
    if (userId) {
      await fetch(`https://restapi.fr/api/vueusers?id=${userId}`, {
        method: 'DELETE',
      });
      state.users = state.users.filter((user) => user._id !== userId);
    }
  } catch (err) {
    console.error(err);
  }
}

watch(
  () => state.selectedUser,
  (user: User | null) => {
    if (user) {
      nameValue.value = user.name;
      emailValue.value = user.email;
    } else {
      nameValue.value = '';
      emailValue.value = '';
    }
  }
);
</script>
```

```
<style lang="scss">
@import './assets/scss/base.scss';
</style>
```