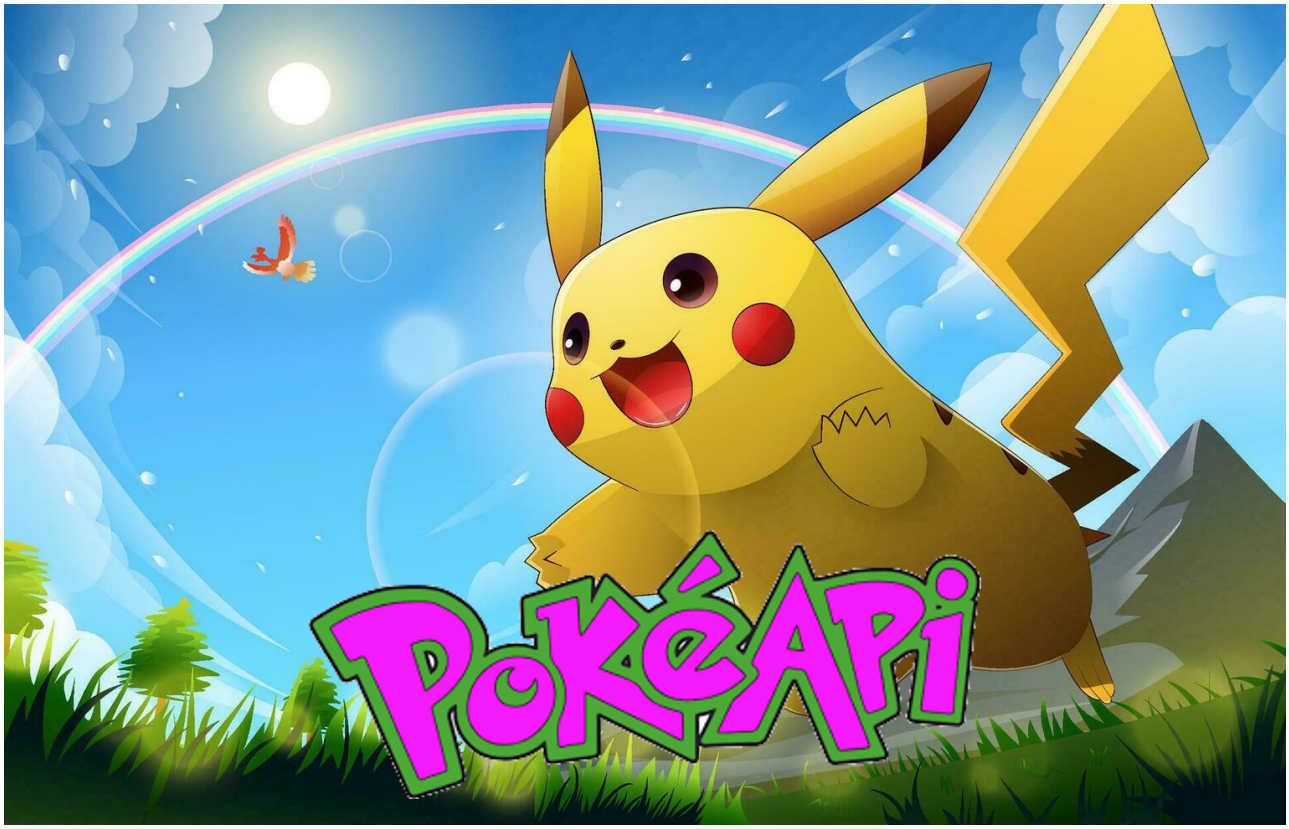

Mise en place d'un système de notifications

Pour une fois ont va prendre les devants et commencer par insérer la balise du composant de notification. Il sera dans `App.vue` comme ceci:

```
<template>
  <NotifBar/>
  <header>
    <h1>Pokedex</h1>
  </header>
  <RouterView />
</template>
```

Passons à la création du composant, rien de bien compliqué ici, ce sont que des concepts que nous avons déjà vu. La nouveauté concerne un objet étrange appelé `useNotifStore`, mais qu'est ce donc et pourquoi avons nous besoin de lui ? La réponse tiens dans la conception de VueJS3. Vue est un framework léger et surtout optimiser. Il ne vérifie pas tout de façon continue. Pour adapté l'affichage il vérifie uniquement ses variables d'états, et il y a deux moyens de s'en procurer:

1. La fonction `ref`.
2. La fonction `reactive`.

Bien, revenons à notre cas d'utilisation. Nous souhaitons avoir une variable réactive mais pas seulement dans un composant. Nous la voulons pour notre application entière. Ca tombe bien, les **stores** sont exactement fait pour ça ! Cela corresponds à notre cas d'utilisation car les notifications sont partout dans notre application (ou possiblement partout). Passons maintenant à la création du composant de notifications:

```
<script setup lang="ts">
import { useNotifStore } from '@/stores/Notif';
const notifStore = useNotifStore();

function close() {
  notifStore.setActive(false);
}
</script>

<template>
  <div id="notif_ctn" v-if="notifStore.isActive">
    <span id="notif_close" @click="close">X</span>
    <p id="notif_text">{{ notifStore.notif.message }}</p>
  </div>
</template>

<style>
#notif_ctn {
  width: 230px;
  height: 60px;
  background-color: red;
  position: absolute;
  top: 40px;
  right: 15px;
  color: #ffffff;
}

#notif_close {
  position: absolute;
  top: 5px;
  right: 5px;
  font-size: 8px;
  height: 10px;
  width: 10px;
  border: solid 2px white;
  border-radius: 10px;
  line-height: 10px;
  text-align: center;
  cursor: pointer;
}
```

```
#notif_text {
  text-align: left;
  padding-left: 10px;
}
</style>
```

VueJS3 intègre un module de gestion du **store** mais celui-ci n'est pas très pratique à utiliser. C'est pour cette raison que nous allons utiliser **Pinia**, une alternative qui fait l'unanimité dans la communauté de VueJS3 et qui à réussis à s'imposer face à la solution officielle. Ainsi nous créons le fameux **store**:

```
import { ref, computed } from 'vue';
import { defineStore } from 'pinia';

export const useNotifStore = defineStore('notif', () => {
  const notif = ref({
    isActive: false,
    message: ''
  });

  const isActive = computed(() => {
    return notif.value.isActive;
  });

  function setMessage(message: string) {
    notif.value.message = message;
    notif.value.isActive = true;
  }

  function setActive(active: boolean) {
    notif.value.isActive = false;
  }

  return {
    notif,
    isActive,
    setMessage,
    setActive
  };
});
```

Exercice: Faire disparaitre le message de notif après quelques secondes.
Exercice: Ajout de couleurs différentes si c'est un message de succès ou d'échec.

Créer une créature

La route:

```
{ path: 'creatures/create', name: 'creatures-create', component:
Public.CreatureCreate },
```

Le lien vers le formulaire de création:

```
<router-link :to="{ name: 'creatures-create' }">Ajouter une créature</router-link>
```

Le composant/vue `CreatureCreate`:

```
<script setup lang="ts">
import { ref } from 'vue';
import { CreatureType, CreatureRace } from '@/_models/Enums';
import type { Creature } from '@/_models/Creature';
import * as CreatureService from '@/_services/CreatureService';
import { useNotifStore } from '@/stores/Notif';

const notifStore = useNotifStore();
const types = Object.values(CreatureType);
const races = Object.values(CreatureRace);

const creature = ref<Creature>({
  id: -1,
  name: '',
  pv: 0,
  atk: 0,
  def: 0,
  speed: 0,
  capture_rate: 0,
  type: types[0],
  race: races[0],
  user_id: 0
});

const errors = ref<any>({});

function handleFileUpload (event: any) {
  creature.value.avatar_blob = event.target.files[0];
}

function create() {
  errors.value = {};
```

```
  CreatureService.createCreature(creature.value).then(() => {
    notifStore.setMessage('Créature crée avec succès');
  }).catch(error => {
    notifStore.setMessage('Données invalides');
    errors.value = error.response.data.errors;
  });
}
</script>

<template>
  <div>
    <form @submit.prevent="create">
      <h2 class="form-title">{{ $t("CREATURE.CREATE") }}</h2>
      <div class="form-group">
        <label for="creature_name">Name</label>
        <input type="text" id="creature_name" v-model="creature.name" />
      </div>
      <div class="form-error">{{ errors?.name }}</div>
      <div class="form-group">
        <label for="creature_pv">PV</label>
        <input type="number" id="creature_pv" v-model="creature.pv" />
      </div>
      <div class="form-error">{{ errors?.pv }}</div>
      <div class="form-group">
        <label for="creature_atk">ATK</label>
        <input type="number" id="creature_atk" v-model="creature.atk" />
      </div>
      <div class="form-error">{{ errors?.atk }}</div>
      <div class="form-group">
        <label for="creature_def">DEF</label>
        <input type="number" id="creature_def" v-model="creature.def" />
      </div>
      <div class="form-error">{{ errors?.def }}</div>
      <div class="form-group">
        <label for="creature_speed">Speed</label>
        <input type="number" id="creature_speed" v-model="creature.speed" />
      </div>
      <div class="form-error">{{ errors?.speed }}</div>
      <div class="form-group">
        <label for="creature_capture_rate">Capture Rate</label>
        <input type="number" id="creature_capture_rate" v-
model="creature.capture_rate" />
      </div>
      <div class="form-error">{{ errors?.capture_rate }}</div>
      <div class="form-group">
```

```
        <label for="creature_type">Type</label>
        <select id="creature_type" v-model="creature.type">
            <option v-for="type in types" :value="type">{{ $t("ENUMS.CREATURE_TYPE."
+ type) }}</option>
        </select>
      </div>
      <div class="form-error">{{ errors?.type }}</div>
      <div class="form-group">
        <label for="creature_race">Race</label>
        <select id="creature_race" v-model="creature.race">
            <option v-for="race in races" :value="race">{{ $t("ENUMS.CREATURE_RACE."
+ race) }}</option>
        </select>
      </div>
      <div class="form-group">
        <label>File
          <input type="file" id="file" ref="inputFile"
@change="handleFileUpload($event)"/>
        </label>
      </div>
      <div class="form-error">{{ errors?.avatar }}</div>
      <div class="form-group">
        <button type="submit" class="button">Créer</button>
      </div>
    </form>
  </div>
</template>

<style>
form {
  max-width: 300px;
  margin: 0 auto;
}
.form-title {
  text-align: center;
  border-bottom: 1px black solid;
  margin-bottom:10px;
}
.form-group {
  display: flex;
  justify-content: space-between;
  margin-bottom: 8px;
}

.form-error {
  color: red;
```

```
  font-size:12px;
  margin-bottom:10px;
}
</style>
```

Focus sur la gestion des erreurs champs par champs

```
const errors = ref<any>({});
```

La fonction `create`:

```
function create() {
  errors.value = {};
  CreatureService.createCreature(creature.value).then(() => {
    notifStore.setMessage('Créature crée avec succès');
  }).catch((error) => {
    notifStore.setMessage('Données invalides');
    errors.value = error.response.data.errors;
  });
}
```

Et enfin pour l'affichage:

```
<div class="form-error">{{ errors?.name }}</div>
```

Remarquez l'utilisation de la syntaxe `?.` pour inclure la condition d'affichage. Cet opérateur signifie que l'accès à la propriété se feras uniquement si `errors` est non nulle. C'est une double protection, d'une part cela évite une erreur de propriété non définie et de l'autre une façon élégante de gérer l'affichage. Contrairement à `errors['name']` qui produirai une erreur dans le cas ou `name` n'existe pas. **Exercice:** Ajoutez la gestion des erreurs partout ou il y a besoin.

Editer une créature

La route:

```
{ path: 'creatures/edit/:id(\\d+)', name: 'creatures-edit', component:
Public.CreatureEdit, props: true },
```

Le lien vers le formulaire d'édition:

```
<router-link :to="{ name: 'creatures-edit', params: { id: creature.id }}">{{
creature.name }}</router-link>
```

Le composant `CreatureEdit`:

```
<script setup lang="ts">
import { ref } from 'vue';
import { onMounted, defineProps } from 'vue';
import { CreatureType, CreatureRace } from '@/_models/Enums';
import type { Creature } from '@/_models/Creature';
import * as CreatureService from '@/_services/CreatureService';
import { useNotifStore } from '@/stores/Notif';

const props = defineProps(['id']);
const notifStore = useNotifStore();
const types = Object.values(CreatureType);
const races = Object.values(CreatureRace);

const creature = ref<Creature>({
  name: '',
  pv: 0,
  atk: 0,
  def: 0,
  speed: 0,
  capture_rate: 0,
  type: types[0],
  race: races[0],
  user_id: 0
});

const errors = ref<any>({});

onMounted(async () => {
  creature.value = await CreatureService.getCreature(props.id);
});

function handleFileUpload (event: any) {
  creature.value.avatar_blob = event.target.files[0];
}

function update() {
  errors.value = {};
  CreatureService.updateCreature(creature.value).then((res) => {
    notifStore.setMessage('Créature éditée avec succès');
  }).catch(error => {
    notifStore.setMessage('Données invalides');
```

```
    errors.value = error.response.data.errors;
  });
}
</script>

<template>
  <div>
    <form @submit.prevent="update">
      <h2 class="form-title">{{ $t("CREATURE.EDIT") }}</h2>
      <div class="form-group">
        <label for="creature_name">Name</label>
        <input type="text" id="creature_name" v-model="creature.name" />
      </div>
      <div class="form-error">{{ errors?.name }}</div>
      <div class="form-group">
        <label for="creature_pv">PV</label>
        <input type="number" id="creature_pv" v-model="creature.pv" />
      </div>
      <div class="form-error">{{ errors?.pv }}</div>
      <div class="form-group">
        <label for="creature_atk">ATK</label>
        <input type="number" id="creature_atk" v-model="creature.atk" />
      </div>
      <div class="form-error">{{ errors?.atk }}</div>
      <div class="form-group">
        <label for="creature_def">DEF</label>
        <input type="number" id="creature_def" v-model="creature.def" />
      </div>
      <div class="form-error">{{ errors?.def }}</div>
      <div class="form-group">
        <label for="creature_speed">Speed</label>
        <input type="number" id="creature_speed" v-model="creature.speed" />
      </div>
      <div class="form-error">{{ errors?.speed }}</div>
      <div class="form-group">
        <label for="creature_capture_rate">Capture Rate</label>
        <input type="number" id="creature_capture_rate" v-
model="creature.capture_rate" />
      </div>
      <div class="form-error">{{ errors?.capture_rate }}</div>
      <div class="form-group">
        <label for="creature_type">Type</label>
        <select id="creature_type" v-model="creature.type">
            <option v-for="type in types" :value="type">{{ $t("ENUMS.CREATURE_TYPE."
+ type) }}</option>
        </select>
```

```
      </div>
      <div class="form-error">{{ errors?.type }}</div>
      <div class="form-group">
        <label for="creature_race">Race</label>
        <select id="creature_race" v-model="creature.race">
            <option v-for="race in races" :value="race">{{ $t("ENUMS.CREATURE_RACE."
+ race) }}</option>
        </select>
      </div>
      <div class="form-error">{{ errors?.race }}</div>
      <div class="form-group">
        <label>File
          <input type="file" id="file" ref="inputFile"
@change="handleFileUpload($event)"/>
        </label>
      </div>
      <div class="form-error">{{ errors?.avatar }}</div>
      <div class="form-group">
        <button type="submit" class="button">Mettre à jour</button>
      </div>
    </form>
  </div>
</template>

<style>
form {
  max-width: 300px;
  margin: 0 auto;
}
.form-title {
  text-align: center;
  border-bottom: 1px black solid;
  margin-bottom:10px;
}
.form-group {
  display: flex;
  justify-content: space-between;
  margin-bottom: 8px;
}

.form-error {
  color: red;
  font-size:12px;
  margin-bottom:10px;
}
</style>
```

Exercice expérimentale: Regrouper l'édition et l'ajout en un seul composant pour éviter les duplications de codes.

Supprimer une créature

La suppression est très souvent gérer directement au niveau des listing. Ce qui implique d'envoyer une requête de suppression à l'API, puis à supprimer manuellement l'item concerné de notre tableau `creatures`:

```
function handleDelete(index: number) {
  const creature = creatures.value[index];
  CreatureService.deleteCreature(creature.id).then(() => {
    creatures.value.splice(index, 1);
  });
}
```

```
<div v-for="(creature, index) in creatures" :key="creature.id">
  <span @click="handleDelete(index)">X</span>
  <router-link :to="{ name: 'creatures-edit', params: { id: creature.id }}">
    {{ creature.name }}
  </router-link>
</div>
```