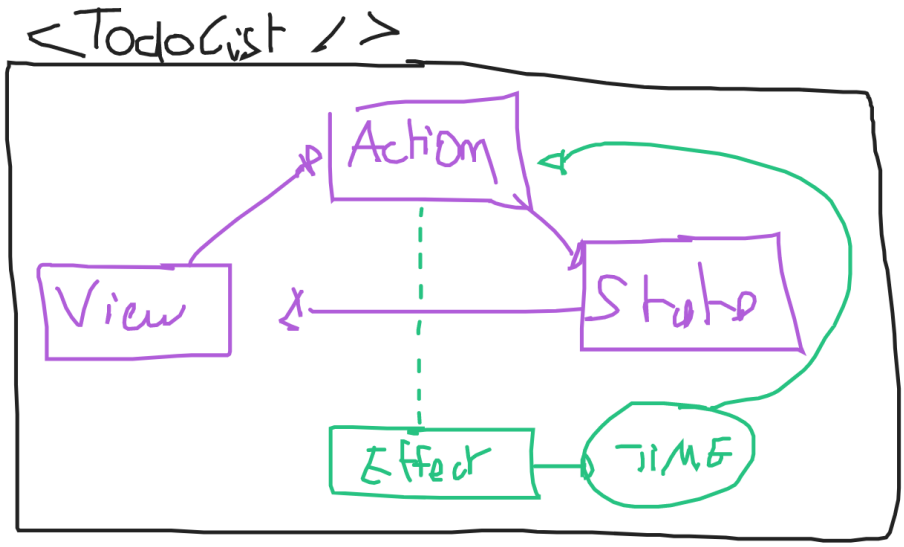
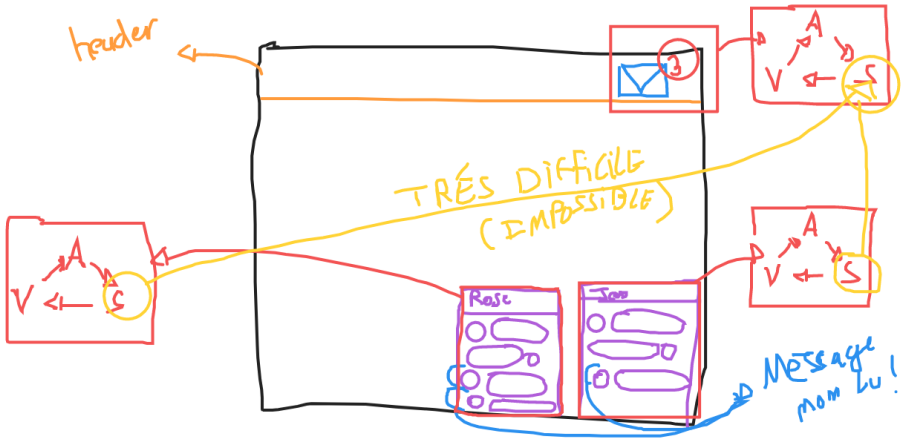
Centraliser l'état avec nanostores

En react, chaque composant est dit "reactive", c'est à dire qu'il posséde ses propres actions, états, vue et effets.



Le problème

Dans une application il est souvent nescessaire de partager les actions, les états et les effets entre nos différents composants. De base, react ne permet pas ce genre de comportement ou très difficilement.



Centraliser l'état

Afin de règler ce problème, nous pouvons mettre en place de multiples solutions :

Utiliser des contextes React :

Cette solution fonctionne, mais elle est très coûteuse en ligne de codes. On se retrouve rapidement submergé par de très nombreux composants. Elle est conseillé lorsque nous devons développer une application de petite à moyenne taille.

Utiliser une librairie : Redux

Cette solution est très puissantes. Le principe est d'enlever l'état et les actions des composants, de les extraires pour pouvoir les utiliser partout.

C'est la librairie a plus compléte, mais la courbe d'apprentissage est très longue.

Utiliser nanostores

Cette solution, est une micro librairie permettant de séparer les actions et l'état des composants.

Elle est très simple utilisé, par contre elle peut devenir difficile à maintenir dsur de très gros projets.

Il en existe d'autres ...

- PubSub
- Remix
- ...

Présentation de nanostores

L'idée de nanostore est de séparer nos états (ne plus utiliser `useState`) mais plutôt créer ce que l'on nomme des "stores" contenant notre état.

Généralement les "stores" sont rangé dans leurs propre dossier (`Store`). Nous possèdons un store par composant.

Exemple :

- Pour un composant : `src/Composant/Calculator` vous aurez un store `src/Store/Calculator`

Pour cela, nous avons le choix entre 2 types de store :

- `Atom` : Convient pour un scalaire (number, string, boolean, etc ...)
- `map` : Convient pour les objets et les tableaux (c'est le plus utilisé).

Installation

Pour installer nanostore :

```
npm i nanostores
npm i @nanostores/react
```

Utilisation d'un atom et la map

```
// Pour utiliser un atom il faut importer la fonction atom
import { atom } from 'nanostores'

// On créer un "état" globale à toute notre application
// contenant une adresse email
export const emailStore = atom('')

// Ici email store est Sotre nanostore :
// pour obtenir le contenu du store
const email = emailStore.get()

// Pour changer l'email du store :
emailStore.set('john@mail.com')
```

Il est aussi possible de créer des stores contenant des objets. Ce sont les store les plus commun !

```
// On importe map, permettant de créer un store sous forme d'objet
import { map } from 'nanostores'

// On peut maintenant créer un état sous forme d'objet. Par
// l'état d'un formulaire de login
export const loginStore = map({
  email: '',
  emailError: '',
  password: '',
  passwordError: '',
  sending: false,
})

// Nous pouvons récupérer le contenue du store :
const email = loginStore.get().email
const password = loginStore.get().password
const loginState = loginStore.get() // Je récupére tout l'objet

// Pour modifier un élément de mon objet :
loginStore.setKey('email', 'john@mail.com')
loginStore.setKey('password', 'johny1234')
```

Utilisation d'un état dans un composant

Immaginons un composant `<Login />` qui a besoin de l'état du login. Pour utiliser l"état il faut utiliser la fonction hook `useStore`

```
import { useStore } from '@nanostores/react'
import { loginStore } from '../Store/Login'

export default function Login() {
  // Pour récupérer tout l'état du login :
  const state = useStore(loginStore)

  // Maintenant je peux utiliser l'état :
```

```
  return (
    <>
      <h1>Login</h1>
      <input type="email" value={state.email}>
      <input type="password" value={state.password}>
    </>
  )
}
```

Utiliser des action avec nanostores

Il est possible de créer avec nanostores des actions. Ce sont des fonctions (qui peuvent être asynchrone) qui ont pour rôle de modifier l'état.

Pour créer une action, il faut utiliser la fonction nanostores `action`. Généralement il est conseillé de mettre les actions dans le meme fichier que le store.

Exemple avec le store de login :

```
// On importe map, permettant de créer un store sous forme d'objet
import { map, action } from 'nanostores'

// On peut maintenant créer un état sous forme d'objet. Par
// l'état d'un formulaire de login
export const loginStore = map({
  email: '',
  emailError: '',
  password: '',
  passwordError: '',
  sending: false,
})

/**
 * Création d'une action permettant de changer l'email
 */
export const setEmail = action(
  loginStore,
  'setEmail',
  (store, newEmail: string) => {
    // On change l'email par l'email envoyé en paramètre
    store.setKey('email', newEmail)
  },
)

// Pour utiliser l'action :
setEmail('marie@mail.com')
```