



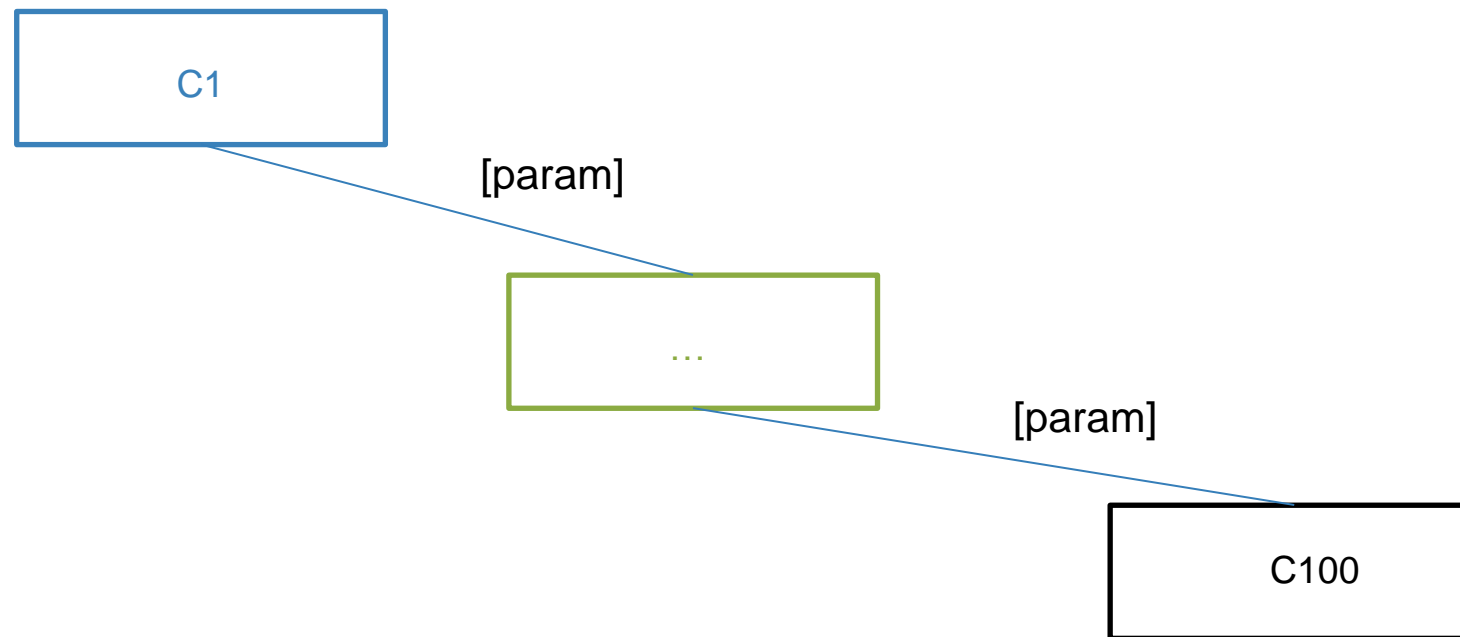
VueX

Une nouvelle manière de gérer vos données

Animé par Mazen Gharbi

Comment faire?

- Vous avez une application Vue complexe, il faut passer une information du composant n°1 au composant n°100..



Introduction à Vuex

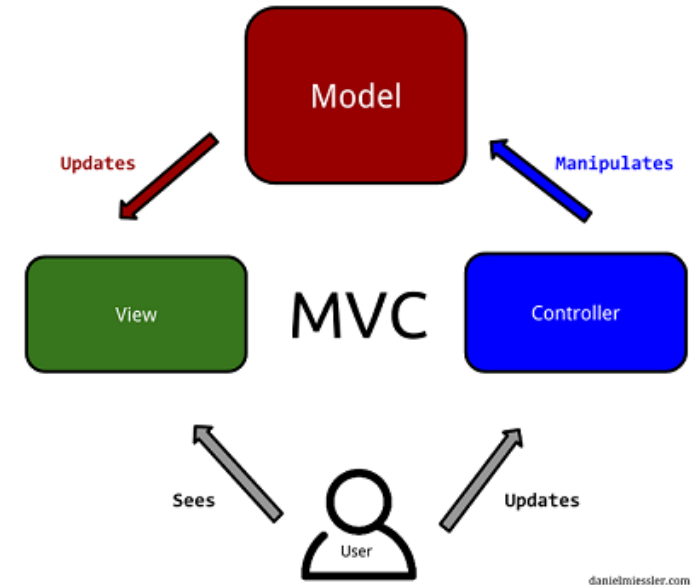
- ▷ Vuex est une librairie officielle
 - › Non présente par défaut, il faut l'installer
- ▷ Permet la gestion d'états
 - › Et surtout, applique le principe de « single source of truth »
- ▷ Boilerplate conséquent
 - › Mais bien plus léger qu'en Angular (NgRx) ou React (Redux)
- ▷ Librairie externalisée (mais **OFFICIEL**)

Application State vs UI State

- ▷ Il existe deux types de « state » dans une app. :
 - › Application State : état général d'une application. Peut être stocké dans une base de données ou ailleurs
 - › UI State : état propre à une partie de l'application (*ex: formulaire*), éphémère et qui peut être effacé
- ▷ Une bonne pratique est de gérer l'Application State par VueX et le UI State par les composants
- ▷ *NB: Cette règle n'est pas inscrite dans le marbre.*

Uni-directional Data Flow

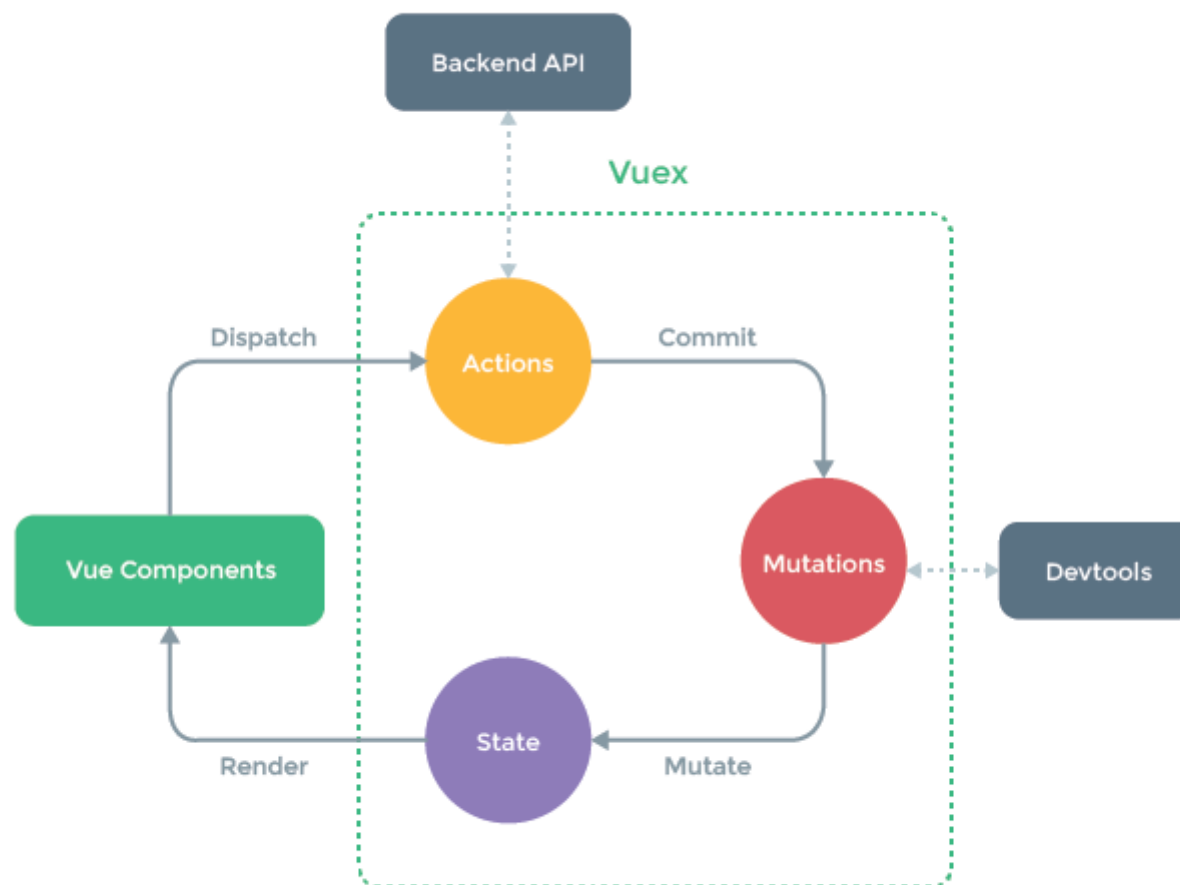
▷ Flux – Design pattern créé par Facebook



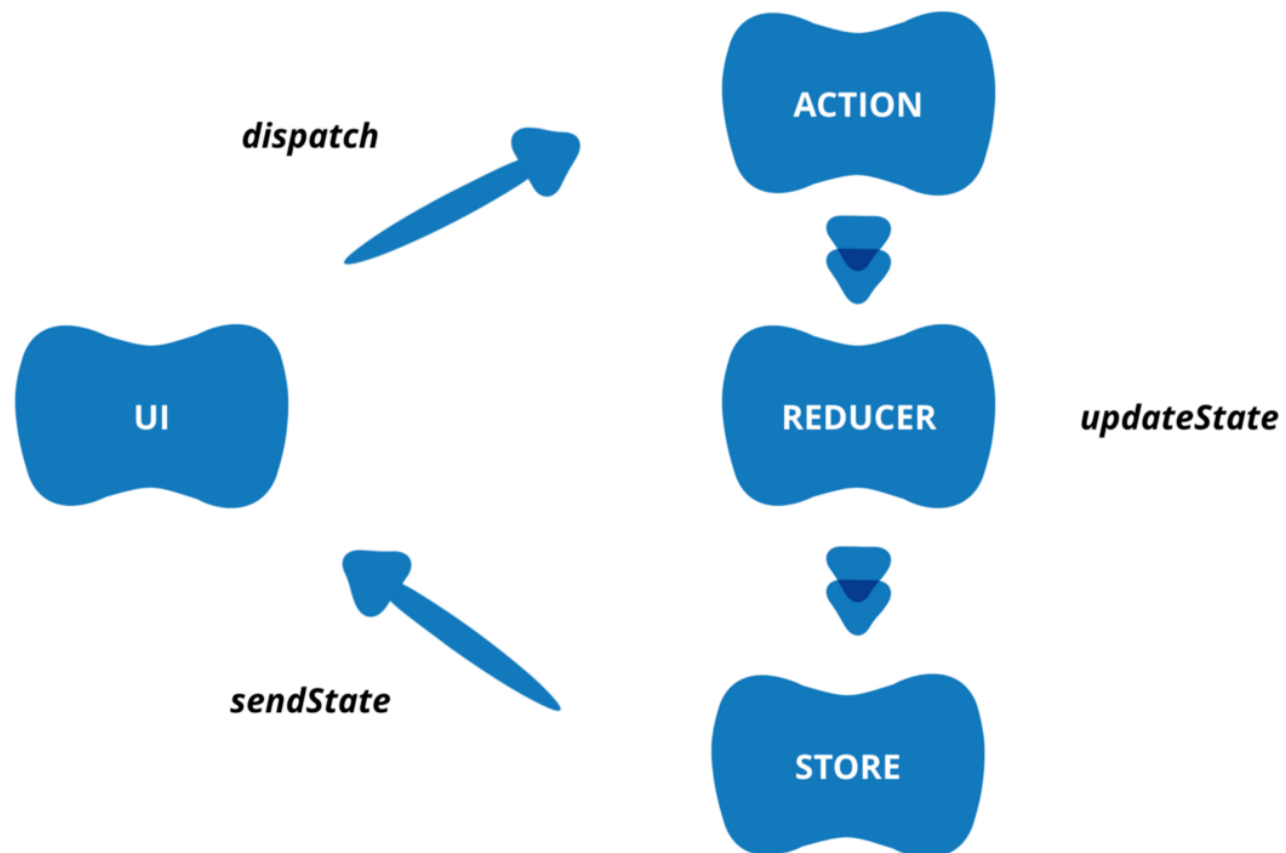
▷ L'architecture Flux repose sur l'idée d'un flux de données unidirectionnel strict. On ne peut affecter les données qui y transitent qu'en suivant un sens précis (on ne le court-circuite pas).

▷ VueX implémente cette architecture avec un vocabulaire qui lui est propre mais le principe est bien le même.

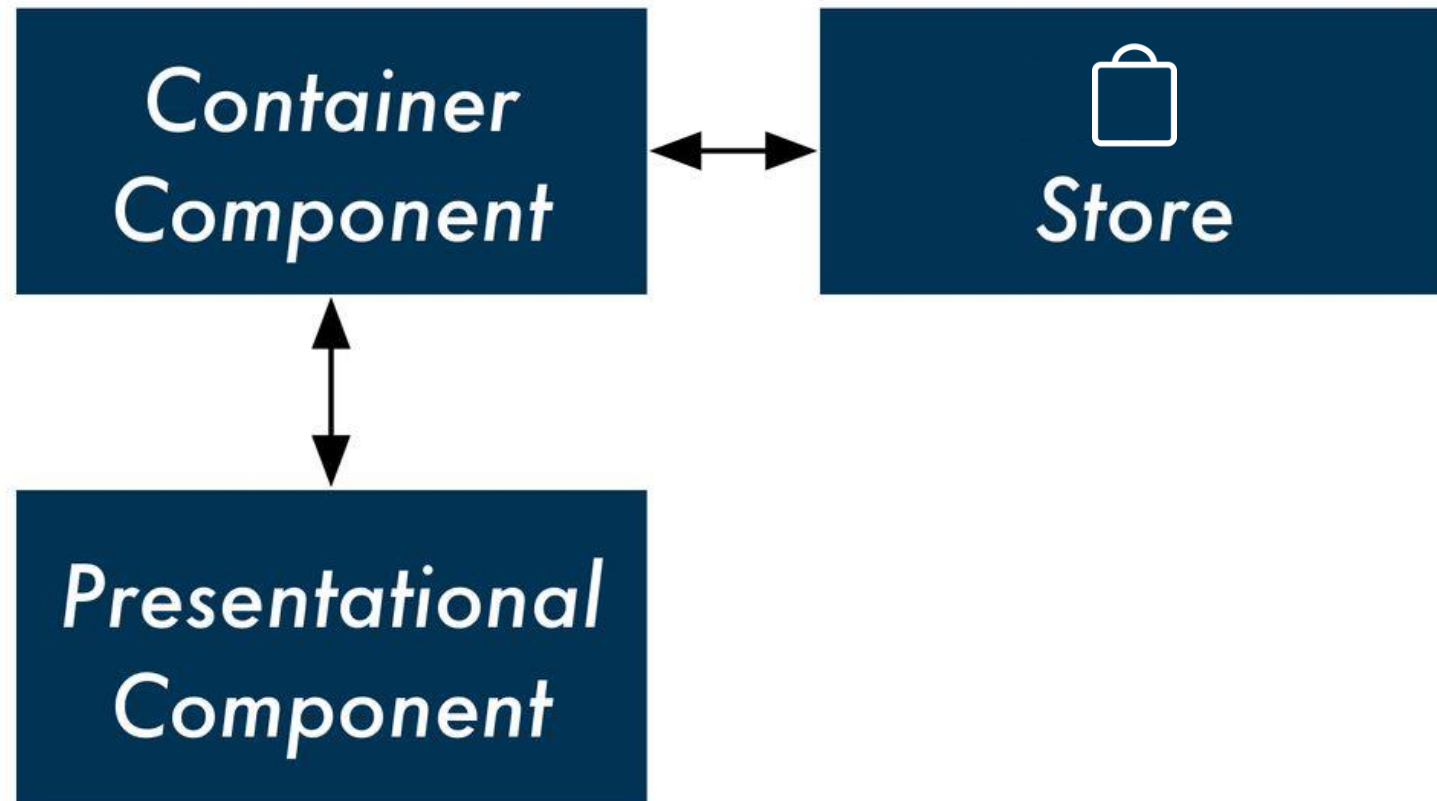
Principe



Principe



Responsabilité des composants



Vocabulaire

- ▷ **Store** : Il constitue la partie centrale de Vuex. Permet de cristalliser l'ensemble des mutations / actions et getters pour les rendre disponible auprès des « Smart Components » *élus*. Stocke le/les state(s) de l'application ;
- ▷ **Mutations** : Fonction permettant de modifier le State actuelle. Prend en premier paramètre le state puis les arguments. **Il doit être prédictible !**
- ▷ **Actions** : Fonctions permettant d'appeler les Mutations. Utilisé par les Smart components pour interagir avec le State
- ▷ **Getters** : Equivalent des « computed properties » mais au niveau Global. Permet de relier les composants au state afin de mettre à jour la vue au changement du State
- ▷ **State** : Objet réactif représentant l'état global de votre application. Il peut être composé de plusieurs sous-objets

Ticketterie

▷ On va créer une billetterie en ligne

Billetterie en ligne

Il y a 50 tickets disponibles

Ajouter un ticket

[Supprimer](#)

Pokémon



Il reste 4 tickets

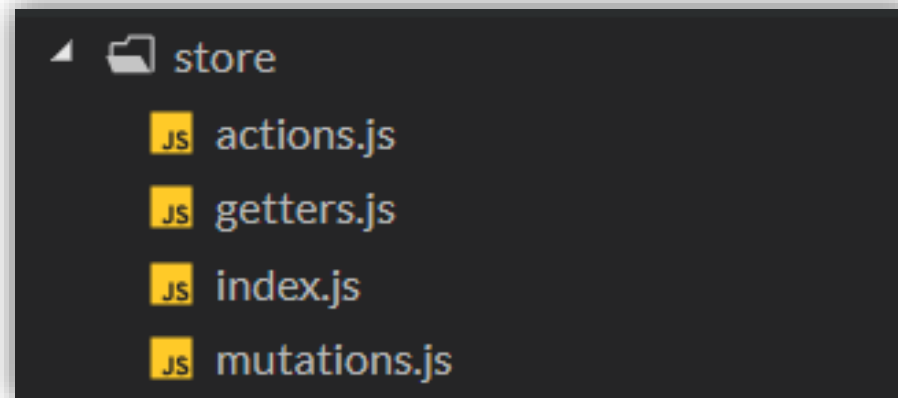
Pokémon

Initialisation

- ▷ Installer au préalable :

```
> npm install vuex --save
```

- ▷ On va essayer d'appliquer une architecture de dossiers / fichiers lisible :



Création du store

- ▷ Comme toute librairie, la première étape va être d'intégrer VueX à Vue :

```
import VuexStore from './store/';  
  
APP.use(VuXStore);
```

store/index.js

- ▷ Puis, déterminer l'ensemble des states du store :

```
import { createStore } from 'vuex'  
export default createStore({  
  state: {  
    tickets: initialTicketsState  
  },  
  actions,  
  mutations,  
  getters,  
})
```

store/index.js

Externalisés dans
d'autres fichiers

Dénomination du state

La valeur peut prendre
tous les types

Création des mutations

- ▷ Les mutations ont la responsabilité de mettre à jour le state auquel ils sont affiliés
- ▷ Ils prennent 1 paramètre obligatoire qui est le **state global** !

```
export default {  
  ADD_TICKETS: function (state, ticket) {  
    state.tickets.unshift(ticket);  
  },  
  REMOVE_TICKET: function (state, ticketIndex) {  
    state.tickets.splice(ticketIndex, 1);  
  }  
};
```

store/mutations.js

On peut également interagir sur d'autres states

- ▷ Il est recommandé d'avoir un mutation dédié par state

Mutations

- ▷ Les mutations **doivent être PREDICTIBLES**
 - › Pas de random
- ▷ **Les mutations sont SYNCHRONES !**
- ▷ Nous ne devons pas effectuer des appels asynchrones à l'intérieur
 - › Pas de timer, ni d'appel serveur
- ▷ Pour les appels asynchrones, il sera nécessaire de les réaliser dans les actions

Actions

- ▷ Les actions permettent d'appeler les mutateurs ;
 - › NE MODIFIENT PAS LE STATE DIRECTEMENT !
- ▷ Ils permettent par exemple d'appeler le serveur puis de modifier le state si nécessaire au retour ;
- ▷ Ils contiennent généralement l'algorithmie ;
- ▷ Prenne en premier paramètre un objet (*context*) contenant les fonctions pour communiquer avec le store. Il contient :
 - › **commit** : Fonction permettant d'appeler un mutateur ;
 - › **state** : Permet d'accéder aux states globaux ;
 - › **getters** : Accéder aux getters ;
 - › **dispatch** : Fonction pour appeler une autre action

Mise en place de nos actions

store/actions.js

```
export default {
  addTicket({ commit }, ticket) {
    commit('ADD_TICKETS', ticket);
  },
  removeTicket({ commit, state }, idTicket) {
    let indexToFind = -1;
    const goodTicket = state.tickets.forEach((tick, index) => {
      if (tick.id === idTicket) {
        indexToFind = index;
        return;
      }
    });

    if (indexToFind !== -1) {
      commit('REMOVE_TICKET', indexToFind);
    }
  },
};
```

Envoyer l'information
au mutation pour modifier
le state

Getters

- ▷ Créons un getter qui permettra aux composants de connaître le nombre total de tickets

```
export default {  
  listTicketsSize(state) {  
    return state.tickets.reduce((nb, ticket) => nb + ticket.nbLefts, 0);  
  }  
};
```

store/getters.js

- ▷ Voir fonctionnement de « reduce » [ici](#)

Relier composant & store

- ▷ La dernière étape va être d'appeler les actions suite aux actions utilisateur
- ▷ Pour ce faire, VueX fournit plusieurs fonctions pour « binder » nos Actions / Getters avec nos composants ;
 - › Il est également possible d'interagir avec le store grâce à « `this.$store` »
- ▷ `mapGetters` permet de relier certains getters au composant
 - › Il existe le même principe avec `mapActions`
- ▷ Renvoient un objet de fonctions

```
▼ {removeTicket: f}  
  ► removeTicket: f mappedAction()  
  __proto__: Object
```

mapGetters & mapActions

- ▷ Ces fonctions doivent d'abord être importées de Vuex

```
import { mapGetters, mapActions } from 'vuex';
```

- ▷ Puis on les relie à notre composant :

pages/home.js

```
computed: {  
  ...mapGetters(['listTicketsSize']),  
},  
methods: {  
  ...mapActions(['removeTicket']),  
}
```

SPREAD

On indique le nom des fonctions que l'on veut « mapper »


Dispatcher les actions

[Testez ce code !](#)

- ▷ Grâce au mapping, il suffit simplement d'appeler l'action « bindée » afin de communiquer avec le store :

pages/home.js

```
<div class="bloc-delete">  
  <span class="delete" @click="removeTicket(ticket.id)">Supprimer</span>  
</div>
```



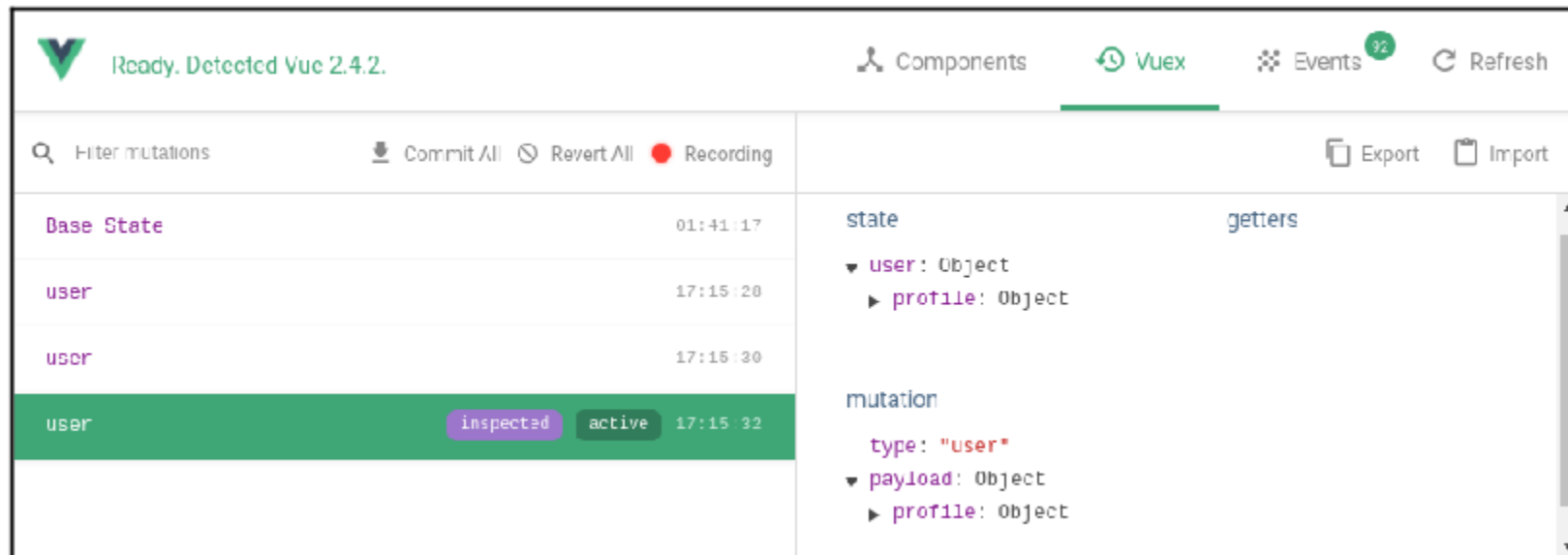
```
mapActions(['removeTicket'])
```

- ▷ Mais il est également possible de passer par l'objet \$store :
 - › La fonction « dispatch » appelle l'action passée en premier paramètre

```
this.$store.dispatch('addTicket', ticketToAdd);
```

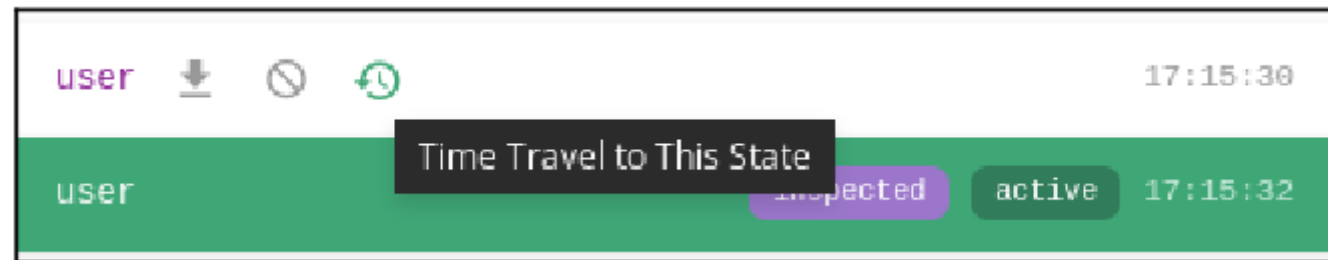
VueX & DevTools

- ▶ La programmation réactive rend le debug parfois difficile
 - › Il arrive que l'on ait du mal à comprendre comment le state est modifiée
- ▶ Le Vue DevTools est inclut avec un débogueur Vuex très performant
 - › Il nous permet d'accéder au state actuel et de le modifier !



VueX & DevTools

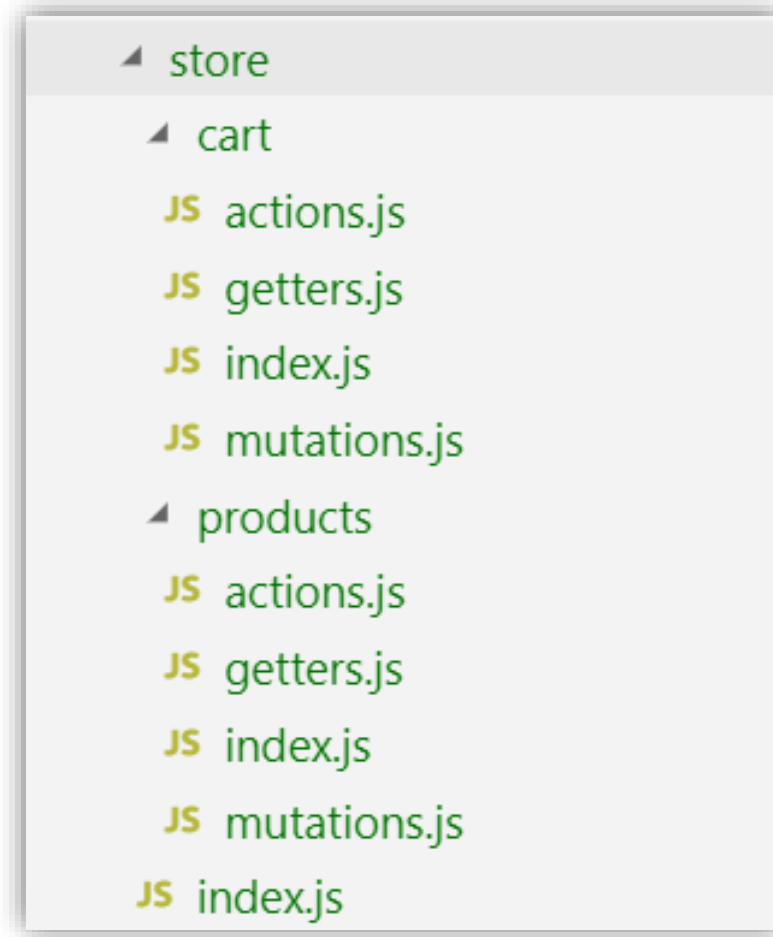
- ▷ On peut même très facilement revenir à un state précédent :



VueX pour les projets conséquents

- ▷ L'écriture que l'on en fait actuellement n'est pas adapté pour les gros projets
 - › 1 fichiers pour gérer tous les states
- ▷ VueX a prévu un système de modules pour améliorer l'organisation
- ▷ Un module contient un state, les getters, mutations et actions ;
- ▷ Le store peut contenir autant de module qu'on le souhaite ;
 - › Un module peut aussi contenir d'autre module !
- ▷ Il est recommandé de créer un module par dossier (ou fichier)

Architecture de dossiers & fichiers



Déclarer un module

► Pour indiquer à Vuex que vous souhaitez créer un module, il est nécessaire d'ajouter l'option « namespaced » :

```
export default {  
  namespaced: true,  
  state: initialState,  
  getters,  
  actions,  
  mutations  
};
```

store/products/index.js

Impossible de dispatcher les actions sans ce paramètre

```
export default createStore({  
  modules: {  
    cart,  
    products  
  },  
  strict: true // Permet d'interdire la modification d'un state en dehors des mutateurs  
});
```

store/index.js

strict: true // Permet d'interdire la modification d'un state en dehors des mutateurs

Dispatcher nos actions

▷ Lors du binding des actions & getters, il va maintenant être nécessaire de spécifier à quel module on fait référence :

```
computed: {  
  ...mapState({  
    checkoutStatus: state => state.cart.checkoutStatus  
  }),  
  ...mapGetters("cart", {  
    products: "cartProducts",  
    total: "cartTotalPrice"  
  })  
},
```

components/ShoppingCart.vue

```
// Le premier paramètre correspond au nom du module du store  
methods: mapActions("cart", ["addProductToCart"]),
```

components/ProductList.vue

Dispatcher nos actions

```
components/ProductList.vue  
  
created() {  
  // Avec plusieurs modules, il est désormais nécessaire de préfixer l'action par le nom du module!  
  this.$store.dispatch("products/getAllProducts");  
}
```

- ▷ Dans une action, il n'est pas nécessaire d'indiquer le nom du module lorsque l'on souhaite appeler la méthode « commit » :

```
store/cart/actions.js  
  
.then(() => {  
  commit("setCheckoutStatus", "réussi");  
})
```

- ▷ Sauf si l'on souhaite appeler le mutateur d'un autre module !

```
store/cart/actions.js  
  
// Retirer 1 produit du stock  
commit(  
  "products/decrementProductInventory",  
  { id: product.id },  
  { root: true }  
);
```

Mutateur du module
« product » !

VueX

- ▷ [Téléchargez ici](#) le code source de ce projet
- ▷ VueX n'est ni un réflexe à avoir
 - › Adaptée pour les grosses applications mais pas pour les petites
- ▷ Ni une obligation
 - › D'autres librairies existent ([puex](#), [vue-stash](#) etc.)

Plugins

- ▷ VueX vous permet d'implémenter des plugins
- ▷ Permettent une meilleure flexibilité de votre application
- ▷ On peut appliquer un plugin pour sauvegarder le state actuel dans le localStorage à chaque modification

Création d'un plugin

▷ On crée d'abord le plugin dans un fichier à part :

```
const monPlugin = store => {  
  // Cette méthode est appelée à la création du store  
  
  store.subscribe((mutation, state) => {  
    // Celle-ci à chaque fois qu'une modification survient sur le state  
    // L'objet mutation est constitué ainsi : '{ type, payload }'  
    // Le type correspond au nom du mutateur  
    // Le payload est un objet contenant les paramètres  
  });  
};
```

▷ Puis on l'applique à notre store :

```
createStore({  
  ...  
  plugins: [monPlugin]  
  ...  
});
```

Plusieurs plugins peuvent être affectés à un même store

Questions

TP - Implémenter VueX

- ▷ Récupérez le TP précédent
- ▷ Votre objectif, gérer les films avec un **Store** !
 - › Le store stocke le **tableau des films**
- ▷ C'est les **mutations** qui devront se charger de :
 1. Ajouter un film ;
 2. Modifier un film ;
 3. Activer / Désactiver l'édition d'un film ;
 4. Supprimer un film.

