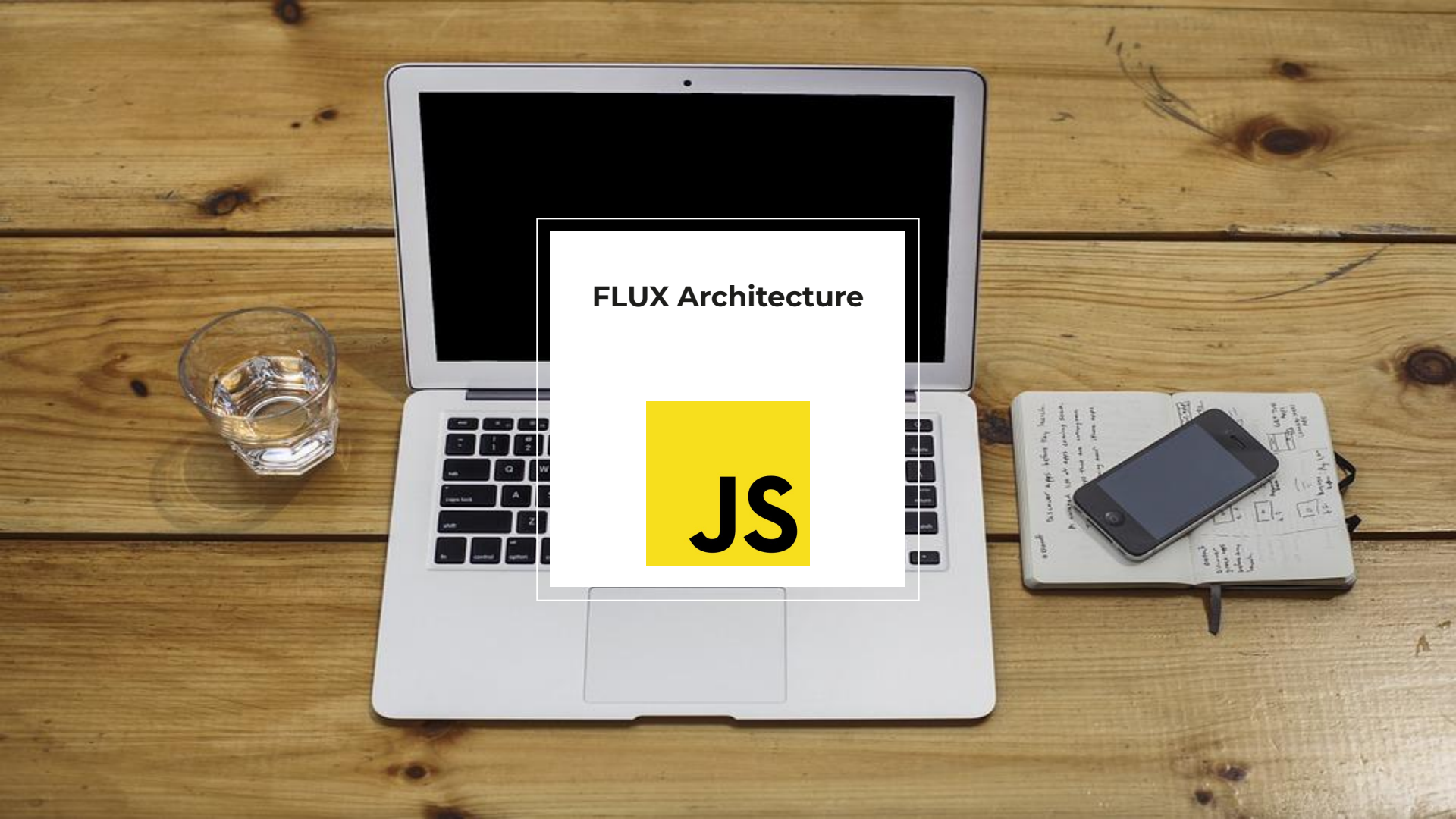
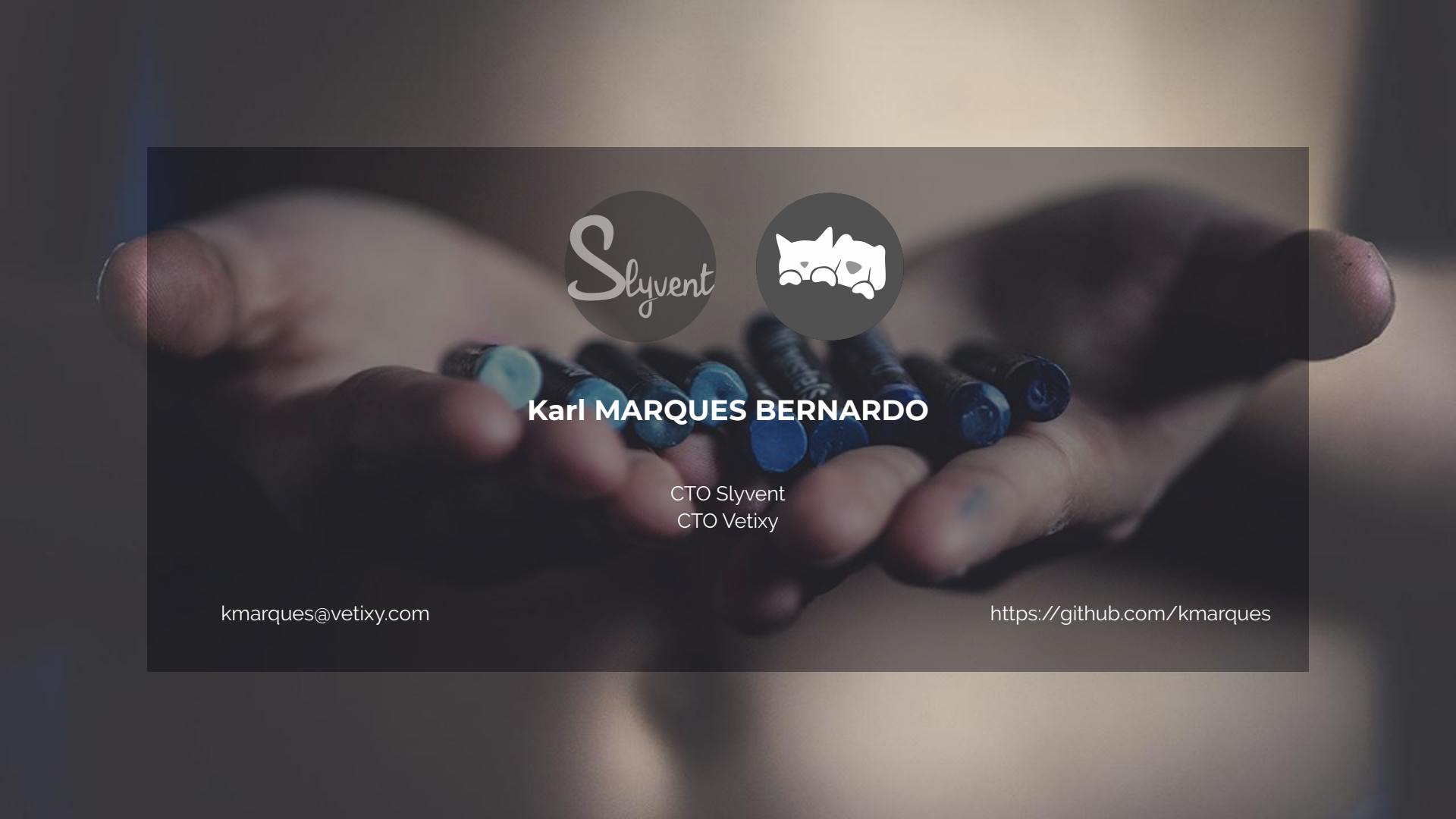

FLUX Architecture
JS

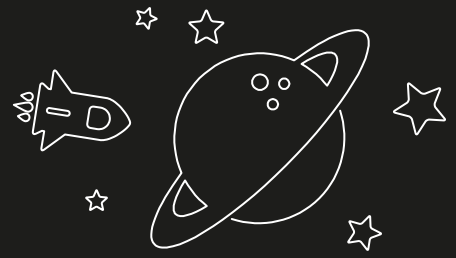


Karl MARQUES BERNARDO

CTO Slyvent
CTO Vetixy

kmarques@vetixy.com

https://github.com/kmarques



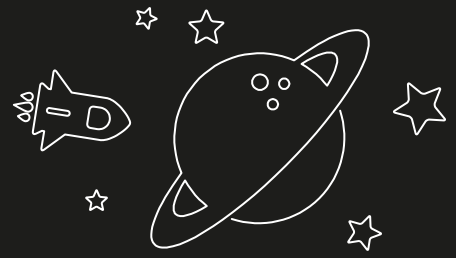
HISTORIQUE

3

4

- Créateur : Facebook
- Date de divulgation: 2014
- Dépôt original: facebook/flux

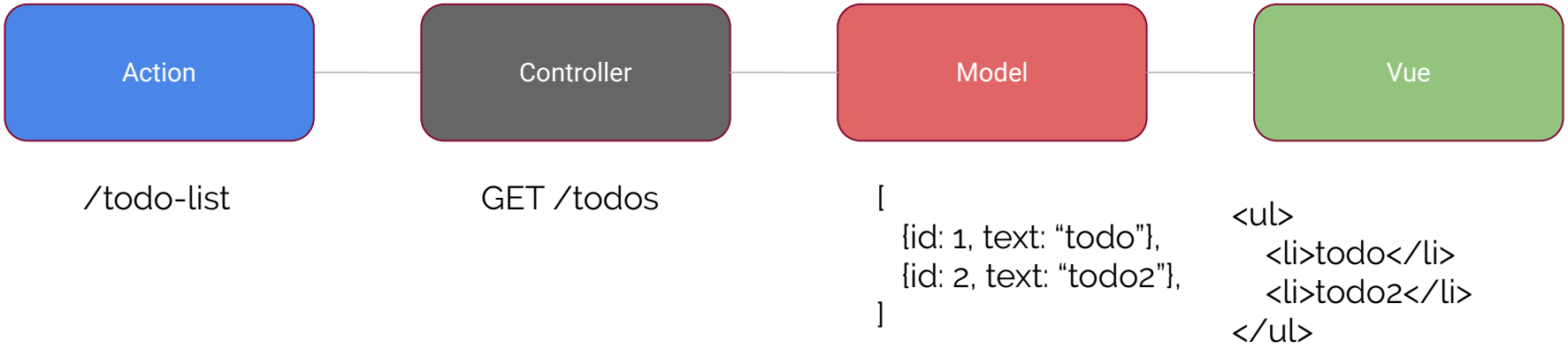
- Flux frameworks connus:
  - reduxjs/react-redux (ReactJS)
  - vuejs/vuex (VueJS)
  - ngrx/platform (Angular)

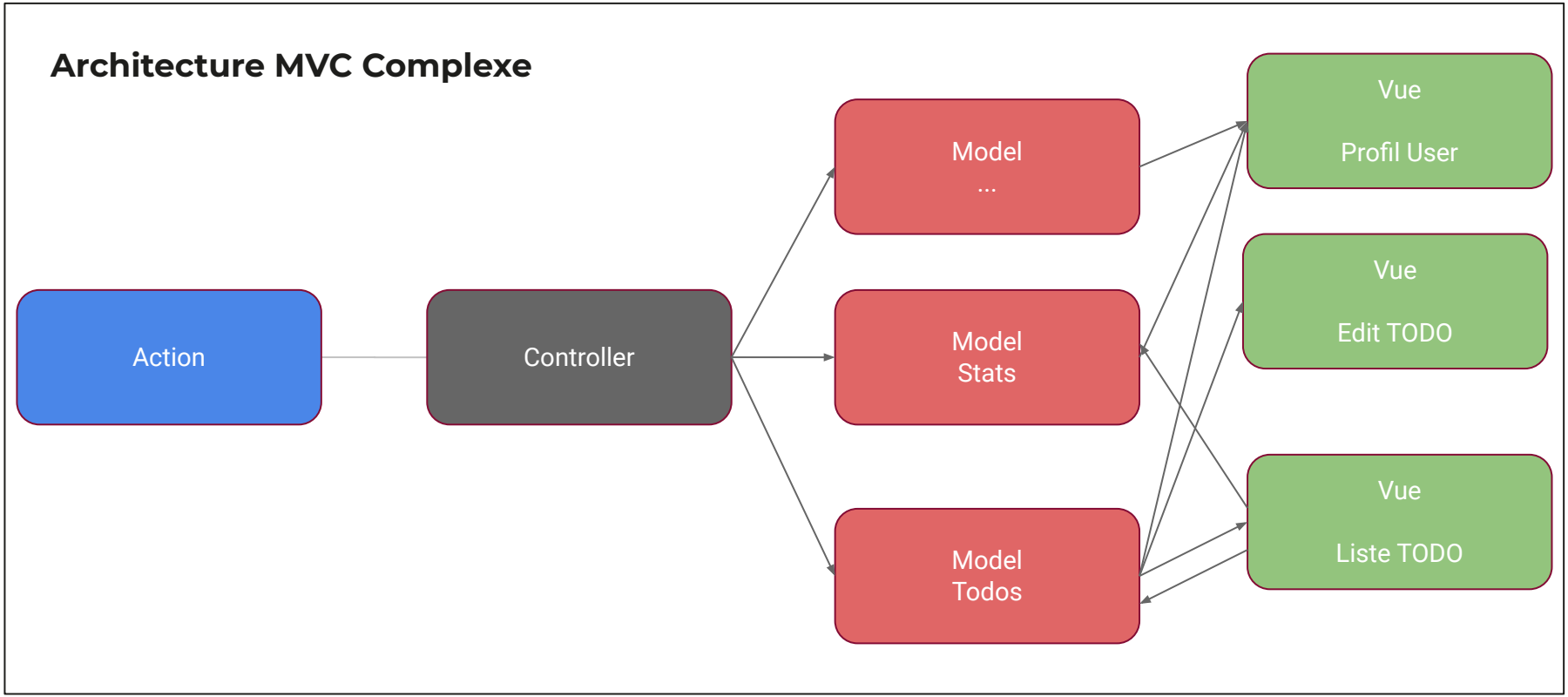


Pourquoi ?

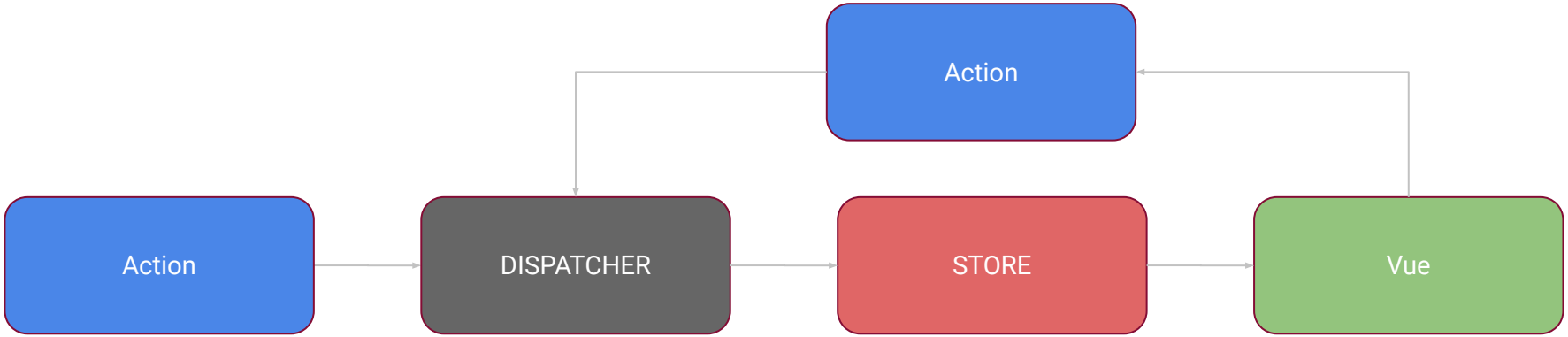
5

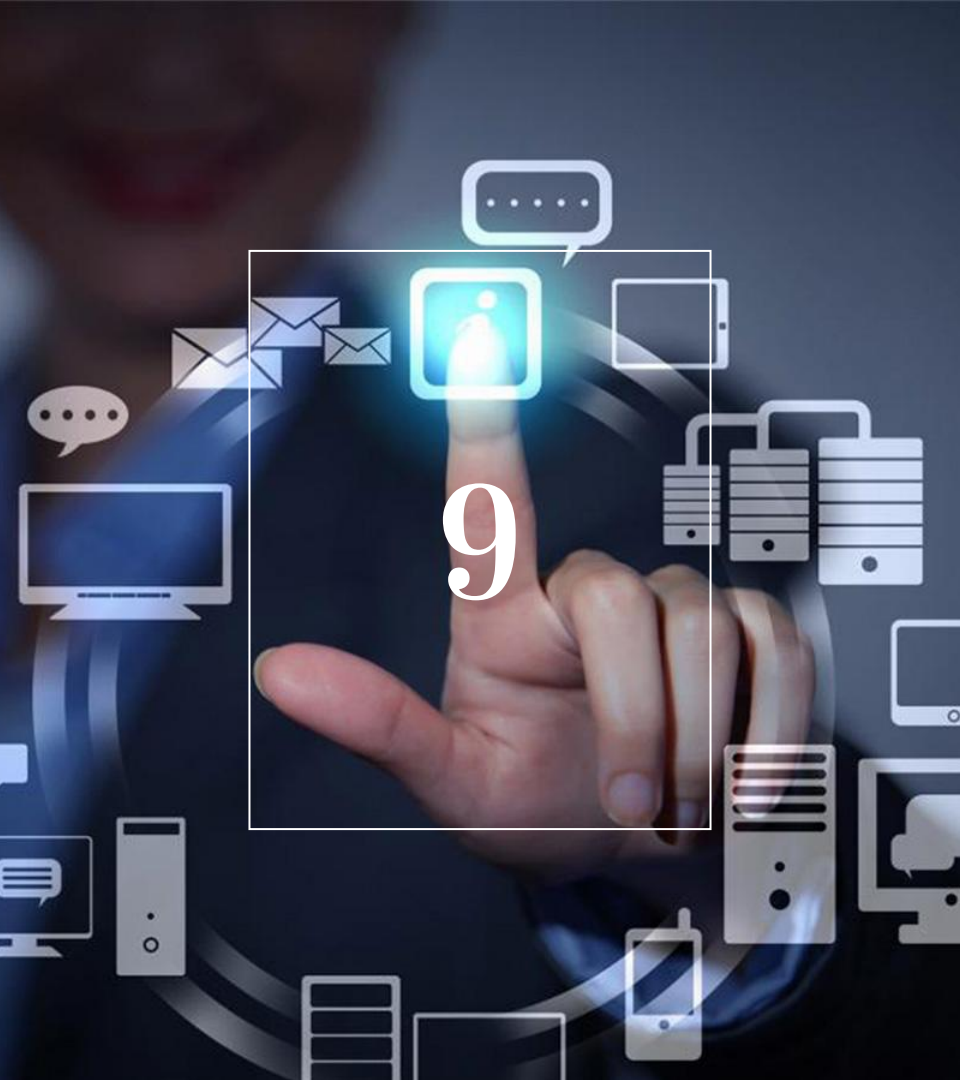
Architecture MVC Simple




Architecture MVC Complexe
Action
Controller
Model
...
Model
Stats
Model
Todos
Vue
Profil User
Vue
Edit TODO
Vue
Liste TODO

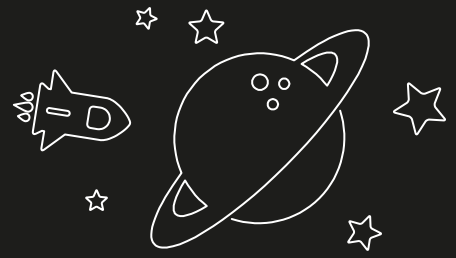
Architecture FLUX





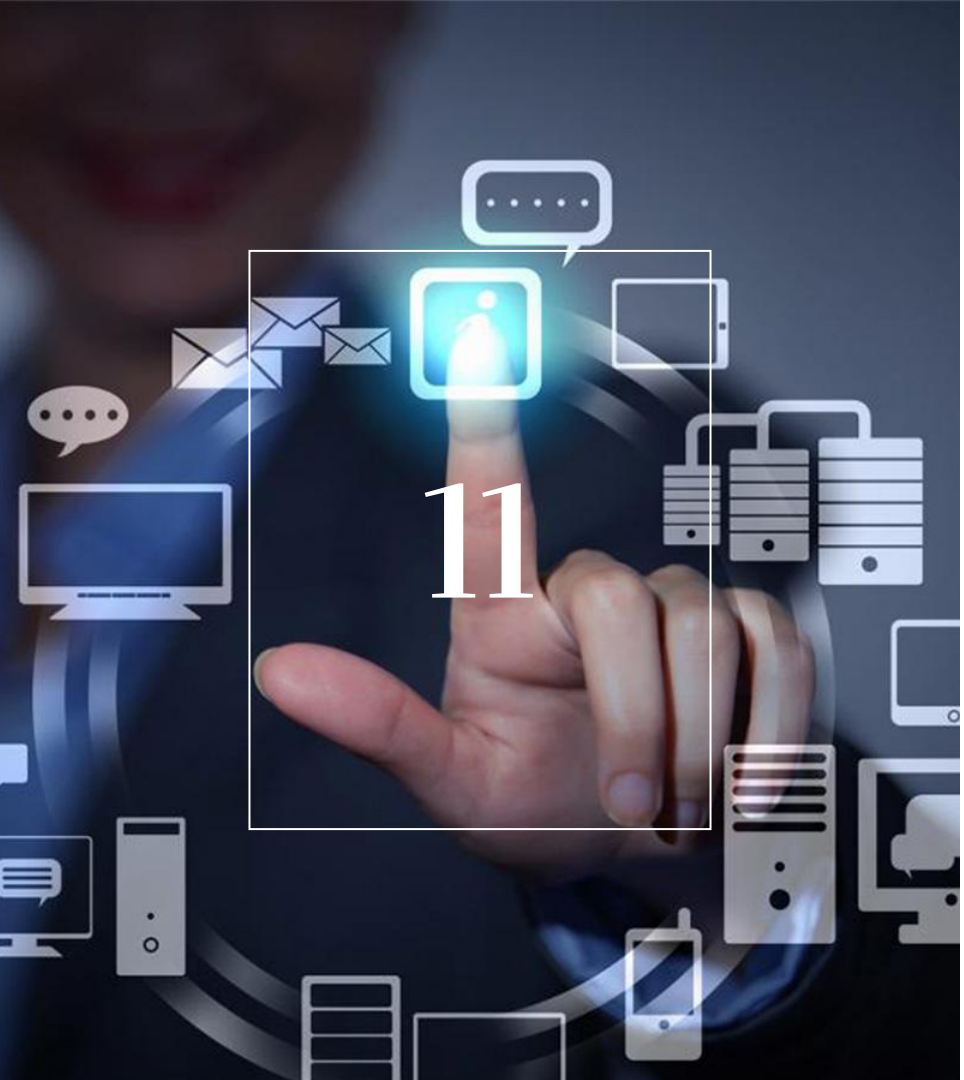
Les avantages

- Meilleur lisibilité du flux de données
- Plus facilement scalable
- Risque d'effet de bord moindre
- Test unitaire simplifié (une source de données



Spécifications

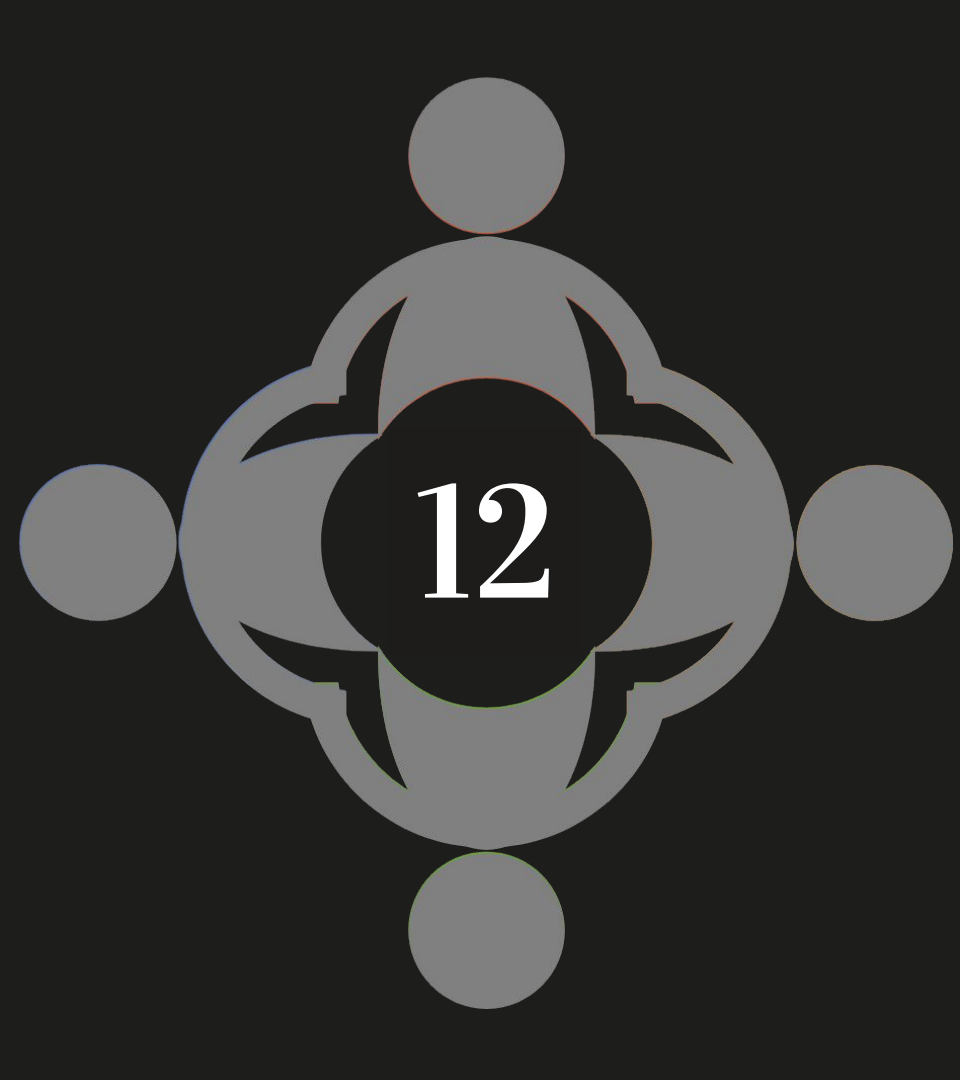
10



- L'architecture Flux est un design pattern de données

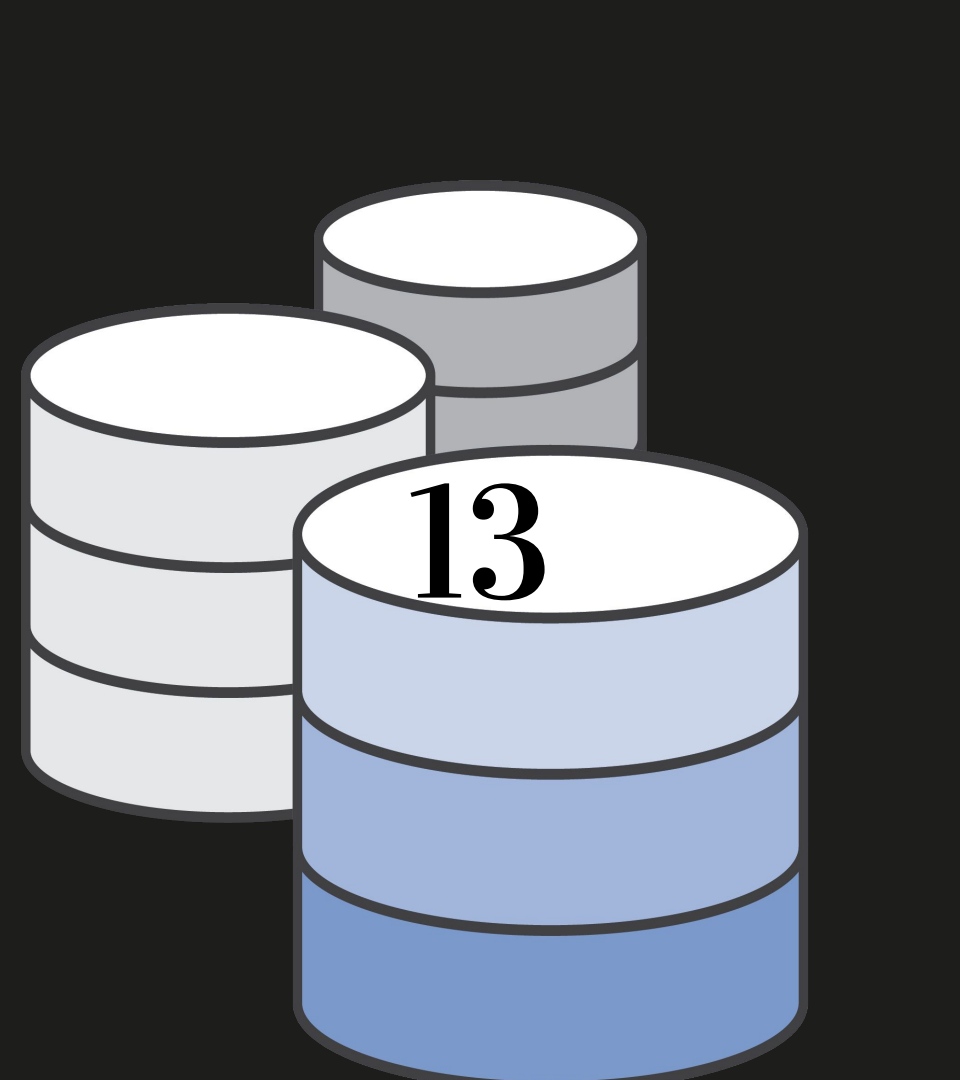
- Composée de 3 éléments essentiels
    - Dispatcher
    - Stores
    - Actions

- Les données sont en One-Way Binding



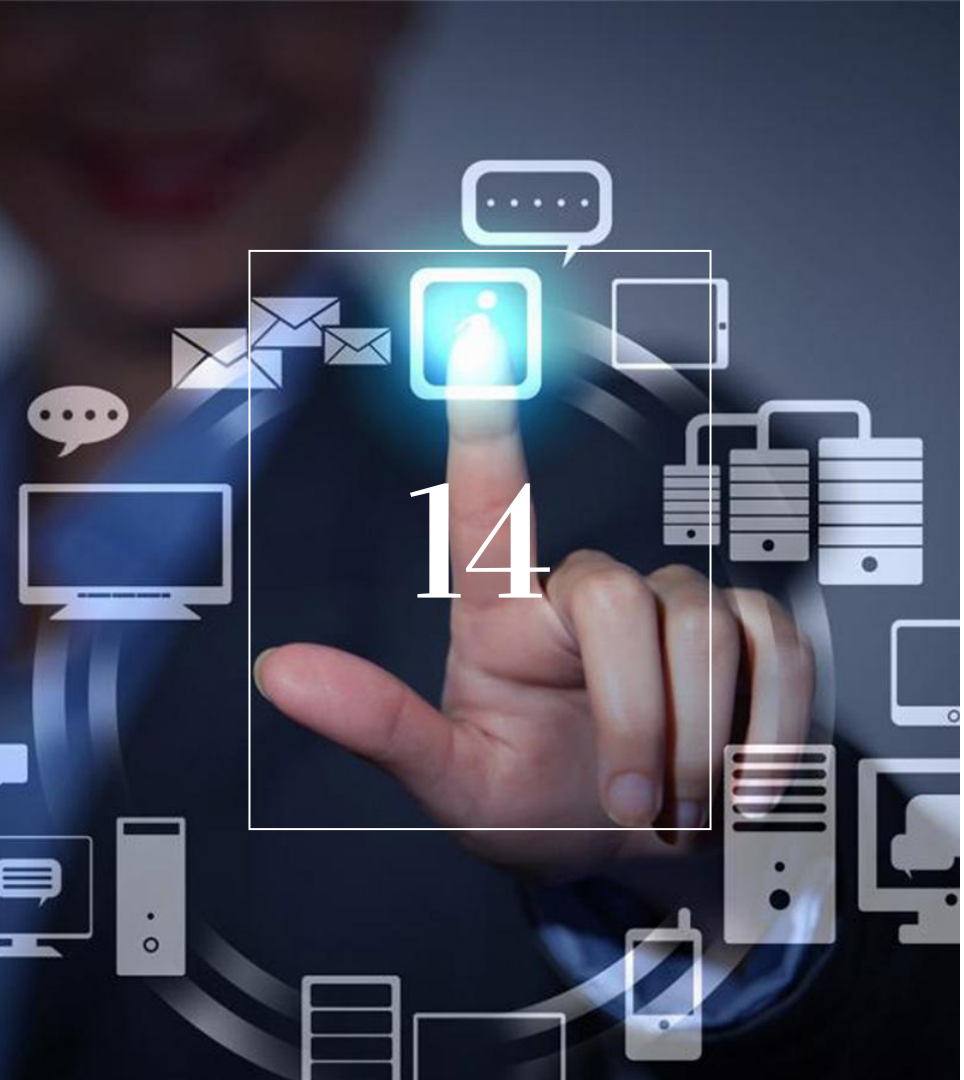
Le Dispatcher

- Unique au sein de l'application
- Enregistre les différents Stores
  méthode register(store.callback)
- Reçoit les différentes actions
  méthode dispatch(action)
- Distribue toutes les actions à tous
  les stores



Le Store

- Possibilité d'en avoir plusieurs souvent sectoriser par domaine d'application
- Représente le State général d'une application et la logique de l'application, équivalent de la partie Controller/Model d'un MVC
- Récoit les actions du Dispatcher
- Process les actions par des Reducers
- Broadcast un event si son état a changé



Les Actions

- Possibilité autant que nécessaire
- Représenté sous la forme d'une fonction
- Appelé par une vue (ex: click, submit)
- Retourne un objet
- Norme de l'objet retourné:
    - type
    - payload

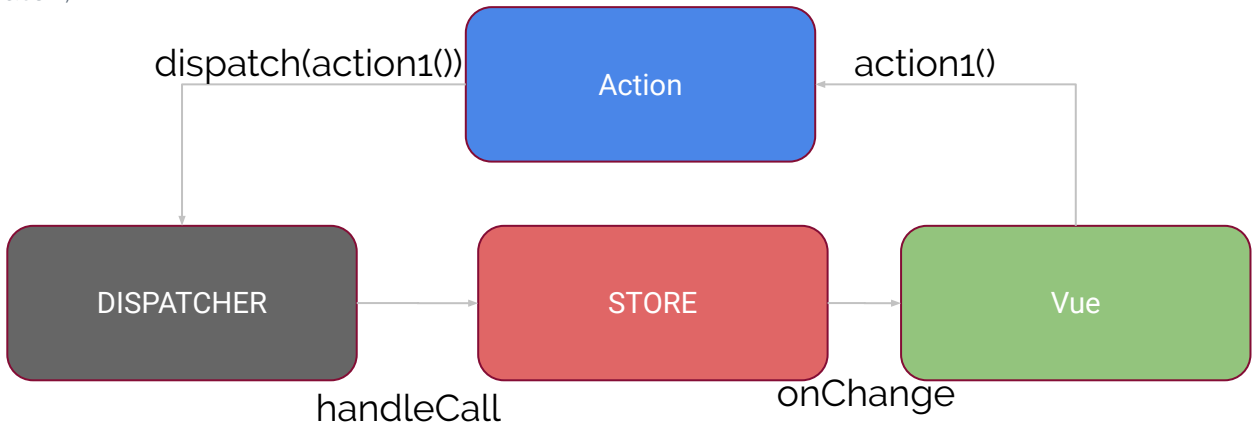
Exemple de processus 1/2

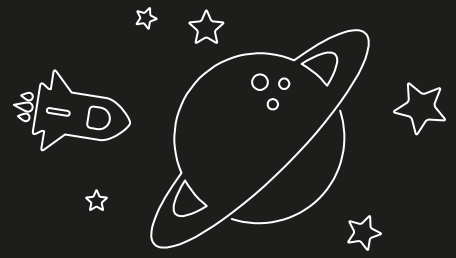
```
// Initialisation
dispatcher = new Dispatcher();
store = new Store();

dispatcher.register(store.handleCall);

// Définiton des actions
action1 = function() {
    return {
        type: "ACTION1",
        payload: {userId: 1}
    };
};
```

```
// Définiton des reducers
reducer = function(state, action) {
        switch(action.type) {
            case 'ACTION1':
                return Object.assign({}, state, {
                    loggedUser: action.payload.userId
                };
            default:
                break;
        }
}

store.addReducer(reducer);
```

Exemple de processus 2/2

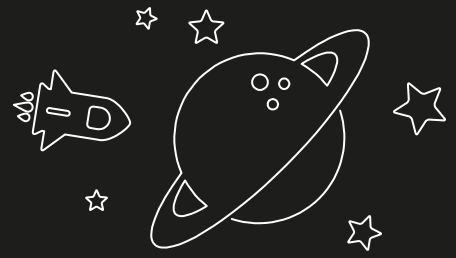
```
// Déclaration d'une vue
  constructor(dispatcher, store) {
    this.dispatch = dispatcher.dispatch;
    store.onChange((state) => {
        this.setState({
            user = state.loggedUser
        });
    });
  }

  handleClick() {
     this.dispatch(action1());
  }

  render() {
     user = this.state.user
     return '<div onClick={this.handleClick}>user</div>'
  }
```





LiveCode
React/Redux

17



Exemples Vuex

18

Redux	Vuex
-	Module
Selector	Getter
Reducer	Mutation
Action	Action

Vuex / Gestion du Store

```
// Initialisation du Store
Vue.use(Vuex)
store = new Vuex.store({
    state: {}     // initial state,
    mutations: myMutations, // Liste des mutations
    actions: myActions,  // Liste des actions
    gettters: myGetters // Liste des getters
});
```

```
// Définiton d'une Action
action1 = ({ commit }, payload) => {
      // traitement
     commit('mutation1', payload);
}
```

```
// Définiton d'une Mutation
mutation1 = (state, payload) => {
     state.myValue = payload.value;
}
```

Vuex / Gestion des composants

```
// Initialisation du composant
comp1 = {
  computed: {
      storeValue: this.$store.state.myValue,
      ...mapGetters([
          'getAll'
      ])
  },
  methods: {
      ...mapActions([
          'action1'
      ]),
  }
}
```