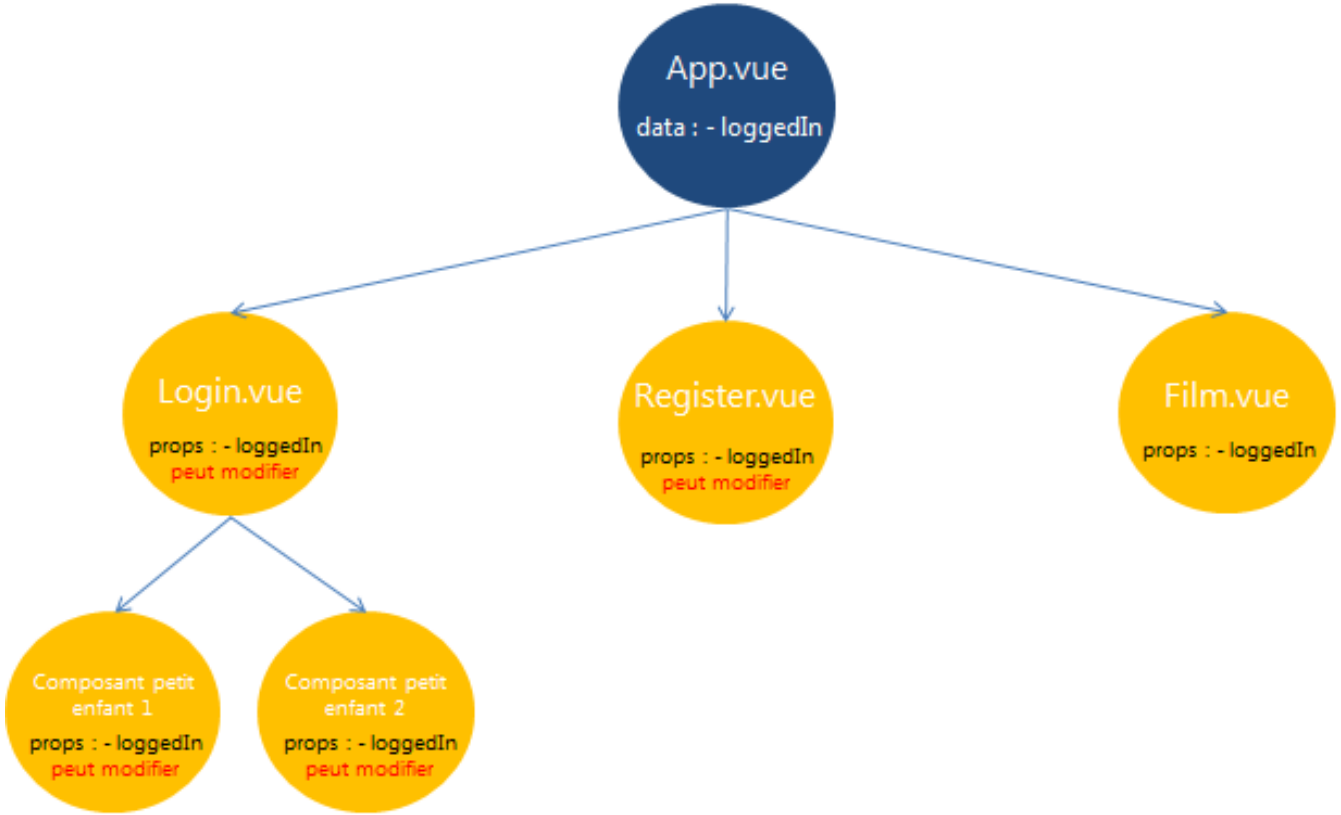
UN FRAMEWORK, ET BIEN PLUS...

QU'EST-CE QUE VUE.JS ?

- Framework progressif
- Réactif
- Arborescence de composants



COMPARATIF

Html + Javascript

- Non réactif, event based
- Eléments chargés dans le DOM
- Navigue entre plusieurs pages (donc site plus lourd)
- Code verbeux

Vue.js

- Réactif
- DOM virtuel
- Component based
- Système de rendu déclaratif

RENDU DÉCLARATIF

```html
<div id="counter">
  Counter: {{ counter }}
</div>
```

```js
const Counter = {
  data() {
    return {
      counter: 0
    }
  }
}

Vue.createApp(Counter).mount('#counter')
```

```
    <span>Picked: {{ picked }}</span>
</div>
```

```
Vue.createApp({
  data() {
    return {
      picked: ''
    }
  }
}).mount('#v-model-radiobutton')
```

LIAISON DES DONNEES

CONDITIONS ET BOUCLES

CONDITIONS ET BOUCLES

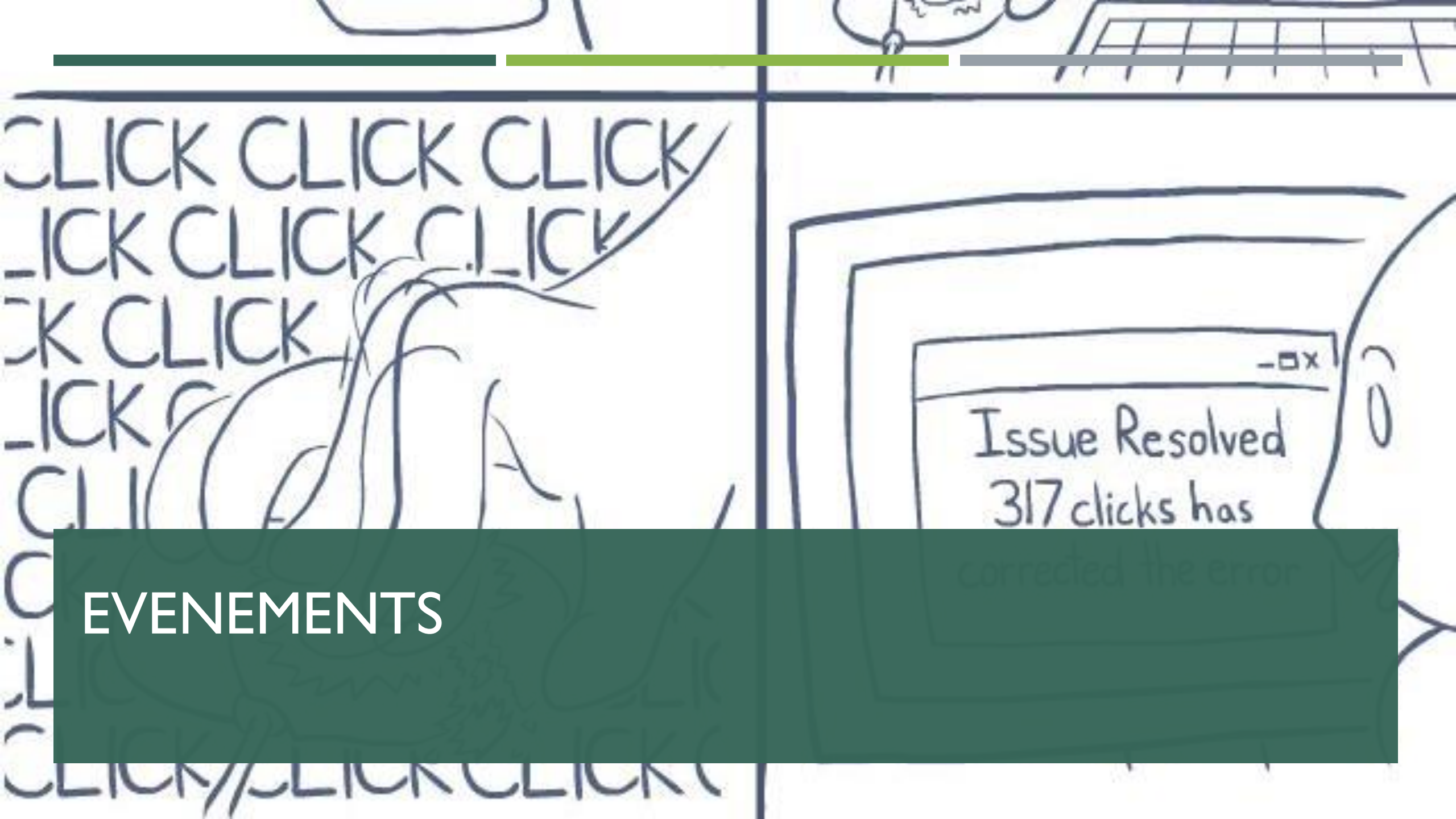
Conditions

```html
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

Boucles

```html
<ul id="array-rendering">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

```js
Vue.createApp({
  data() {
    return {
      items: [{ message: 'Foo' }, { message: 'Bar' }]
    }
  }
}).mount('#array-rendering')
```



EVENEMENTS

```html
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Message retourné</button>
</div>
```

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js !'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

EVENTS ET MÉTHODES

UNE DIRECTIVE POUR PLUSIEURS EVENTS...

- La directive "v-model" lie de manière bi-dimensionnelle entre les champs d'un formulaire et une donnée.

HTML

```html
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

JS

```js
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue !'
  }
})
```

PETIT EXERCICE PRATIQUE

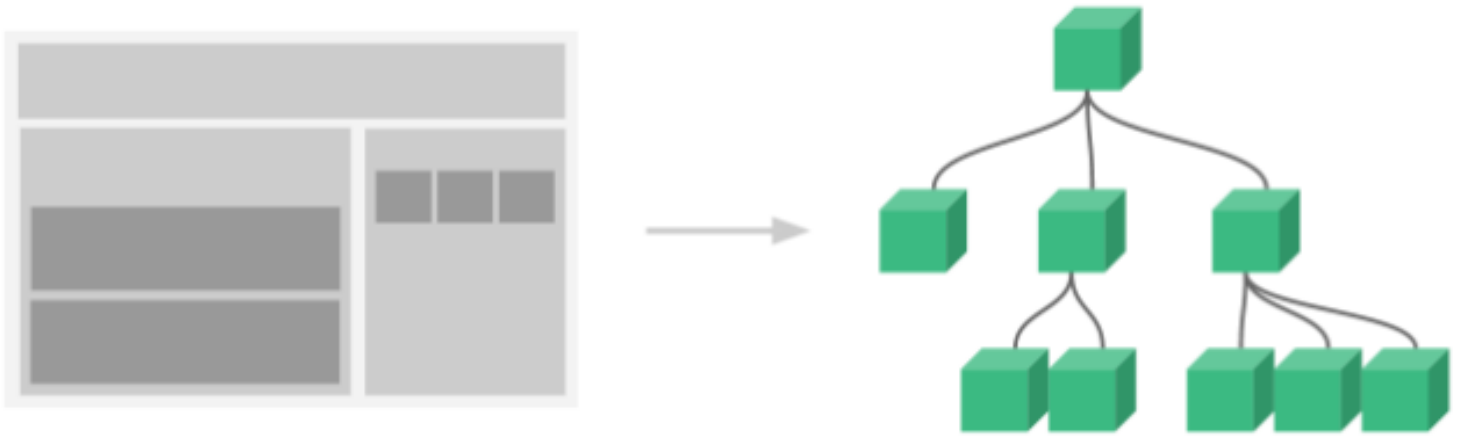
Créer une application vue

Créer une data qui contiendra une donnée texte et une data qui contiendra un tableau d'objets (Chaque objet se compose d'un attribut id et d'un attribut name)

Dans le template html, créer une zone qui contiendra un input texte simple et une autre zone où l'on affichera la valeur de l'input, et une dernière zone qui affichera une liste basée sur le tableau d'objets

Créer un bouton, qui au clic, appellera une méthode qui affichera le texte simple et au deuxième clic, affichera la liste.

LES COMPOSANTS



```vue
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  components: {},
  directives: {},
  props: {
    msg: String
  },
  data: () => {
    return {
      testMsg: 'test',
    }
  },
  computed: {
    reactiveTestMessage() {
      return testMsg.reverse().join('');
    }
  },
  methods: {},
  watch: {}
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
  h3 {
    margin: 40px 0 0;
  }
  ul {
    list-style-type: none;
    padding: 0;
  }
  li {
    display: inline-block;
    margin: 0 10px;
  }
  a {
    color: #42b983;
  }
</style>
```

QU'EST-CE QU'UN COMPOSANT?

- Une instance de vue avec des options prédéfinies
- Déclaré dans un fichier .vue

PROPS D'UN COMPOSANT

Dans le composant :

```
export default {
  name: 'HelloWorld',
  components: {},
  directives: {},
  props: {
    msg: String
  },
  data: () => {
    return {
      testMsg: 'test',
    }
  },
```

En appelant le composant :

```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Bonjour"/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue';

export default {
  name: 'App',
  components: {
    HelloWorld,
  }
}
</script>
```

COMPUTED

Valeurs calculées

- Permettent de retourner des valeurs en se basant sur des conditions, en les transformant etc

```
<template>
  <div class="hello">
    <h1>{{ reactiveTestMessage }}</h1>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  components: {},
  directives: {},
  props: {
    msg: String
  },
  data: () => {
    return {
      testMsg: 'test',
    }
  },
  computed: {
    reactiveTestMessage() {
      return testMsg.reverse().join('');
    }
  },
```

WATCHERS

- Permettent de surveiller la valeur d'une propriété(data, computed, etc)

HTML

```
<div id="demo">{{ fullName }}</div>
```

JS

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

PRACTICE TIME!

- Créer une appli vue et 3 composants
- Contenu des composants :
  - Composant 1 : Un input, un texte et le composant 2
  - Composant 2 : 3 objets listes en data, un paramètre en entrée(id de liste) et un composant 3
  - Composant 3 : Affiche une liste d'objets prise en paramètre et un paramètre de tri
- Le contenu de l'input du Composant 1 attendra un id de liste choisie arbitrairement
- A chaque fois que la valeur de l'input correspond à un id de liste, le Composant 2 doit mettre à jour le Composant 3 avec la liste correspondante et la trier avec un paramètre permettant de trier de manière ascendante ou descendante la liste

EVENEMENTS PERSONNALISÉS

COMMUNICATION ENFANT - PARENT

Transmission d'événement

```
methods: {
  sendText(target) {
    if (target)
     this.$emit('my-event', target.innerHTML);
  }
}
```

Récupération de l'événement

```
<!-- parent template -->
<div id="my-app">
   <my-component @my-event="doSomething"></my-component>
</div>
```

PRACTICE... AGAIN!

- Reprendre l'exercice précédent
- Créer un nouveau composant contenant à minima un bouton
- Appeler ce composant à partir du composant 3
- Au clic sur le bouton, incrémenter l'id de la liste à afficher

VUE ROUTER

Permet une navigation basée sur les composants

Routes et sous-routes

Gestion fine de la navigation avec parametres

Classes css pour les liens actifs

VUE-ROUTER

CONFIGURER UNE ROUTE AVEC VUE-ROUTER

```
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld
    },
    {
      path: '/Home/:msg',
      component: Home
    },
    {...
    },
  ]
})
```

- Ajouter le router à l'instance de Vue (Vue.use)
- Ajouter un tableau de chemins par composant

COMMENT L'UTILISER?

Appeller le moteur vue-router

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>

<script>
```

Naviguer grace à vue-router

```
<router-link to="/">Go to HelloWorld by router link</router-link>
```

ou ...

```
this.$router.push({path: '/'});
```

ET POUR PASSER DES PARAMETRES?

Les déclarer

```
{
  path: '/Home/:msg',
  component: Home
},
```

Les passer

```
<router-link to="/Home/Bienvenue">Go to Home with message = "Bienvenue"</router-link>
```

```
this.$router.push({path: '/Home', params: { msg: 'Bienvenue' }})
```

Les récupérer

```
this.$route.params.msg
```

ET LES PROPS C'EST PAREIL?

```
routes: [
  { path: '/promotion/from-newsletter', component: Promotion, props: { newsletterPopup: false } }
]
```

ET LES QUERY?

- Permettent de passer des paramètres ayant pour but de filtrer le contenu

```
// avec une requête « query » résultant de `/register?plan=private`
router.push({ path: 'register', query: { plan: 'private' }})
```

ROUTES NOMMÉES

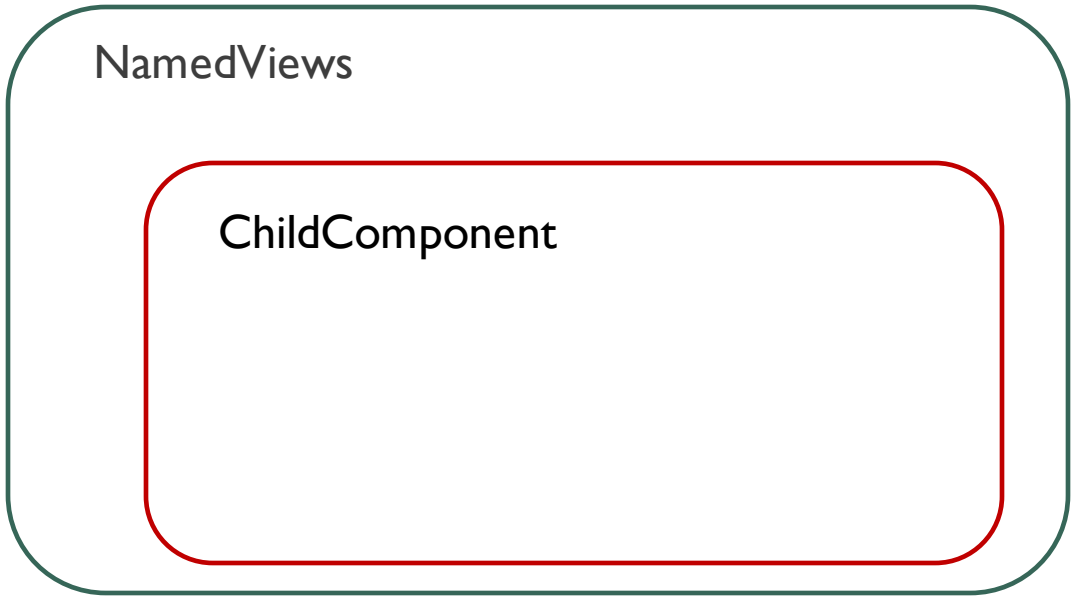
```js
const router = new VueRouter({
  routes: [
    {
      path: '/utilisateur/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

```html
<router-link :to="{ name: 'user', params: { userId: 123 }}">Utilisateur</router-link>
```

```js
router.push({ name: 'user', params: { userId: 123 }})
```

ROUTES NOMMÉES IMBRIQUÉES

/NamedViews/childComponentExample



/NamedViews/namedSubView