



Vue.js

Introduction à VueJS

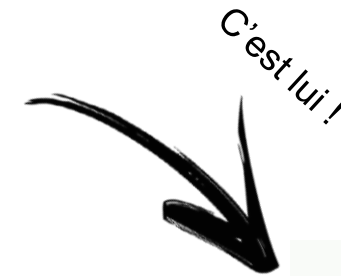
Animé par Mazen Gharbi



Vue d'ensemble

Création du framework

- ▷ Créé par « Evan You », un ex-employé de Google
 - › Dans l'équipe **AngularJS**
- ▷ Son objectif : Reconstruire un framework léger !
 - › Et **performant** bien sûr
- ▷ VueJS en version 3 !
 - › Plus léger / rapide / facile 😊
- ▷ Une communauté très active
- ▷ Grandement inspiré de React, Polymer et Angular



C'est lui !



Philosophie du framework

- ▷ VueJS vous propose une logique de développement **incrémentale**
- ▷ Développez avec un petit noyau solide au départ, et faites grossir votre application très facilement
 - › Router / Vuex / etc.
- ▷ Framework volontairement très **accessible**
- ▷ **Performant** d'après les nombreux benchmarks sur le net ([exemple](#))
- ▷ **Aucun support pour IE**

Logique de conception

- ▷ Modèle Vue / Vue Modèle (MVVM)
- ▷ Tout dans un seul fichier !?
- ▷ Ré-utilisabilité en priorité
- ▷ Les version futures prévoient une meilleure intégration avec des environnements tels que [Weex](#) ou [NativeScript](#) (et [Ionic](#) depuis peu)
- ▷ Aucune dépendance nécessaire autre que la librairie VueJS !

Références



NETFLIX



euronews.





Initialisation d'un projet Vue

Contexte

- ▷ Pour faciliter l'accès, nous utiliserons l'IDE en ligne « stackblitz »
 - › stackblitz.com
- ▷ Durant ce cours, les bonnes pratiques apparaitront en vert

Création d'un premier projet Vue

► Commençons par mettre en place une page .html basique

```
<!doctype html>
<html class="no-js" lang="fr">
<head>
  <meta charset="UTF-8">
  <!--[if IE]><meta http-equiv="X-UA-Compatible" content="IE=edge"><![endif]-->
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  <title>Cours VueJS</title>
</head>

<body>

</body> <!-- FIN BODY -->
</html>
```

index.html

Importer VueJS

▷ Nous avons différentes façons d'y arriver

▷ CDN

index.html

```
...  
    <script src="https://unpkg.com/vue@next"></script>  
    <script src="./index.js"></script>  
</body> <!-- FIN BODY -->
```

▷ Local

index.html

```
...  
    <script src="./libs/vue.js"></script>  
    <script src="./index.js"></script>  
</body> <!-- FIN BODY -->
```

Text interpolation

▷ Nous allons initialiser notre application VueJS sur un bloc HTML donné

```
<body>
  <div id="app">
    Bonjour à tous, bienvenue dans la formation donnée par <u>{{ societe }}</u>
  </div>
```

Mustache Tags !

index.html

▷ Une fois le bloc créé, on va lui appliquer un comportement grâce à VueJS

```
const App = {
  data() {
    return {
      societe: 'Macademia'
    }
  }
};

Vue.createApp(App).mount('#app');
```

index.js

Bonjour à tous, bienvenue dans la formation donnée par Macademia

Débugger sur chrome & firefox

▷ Nous aurons des cas de figures bien plus complexe à gérer

```
const vueApp = {  
  data: {  
    societe: 'Macademia',  
    nom_enseigne: 'Krusty Krab',  
    nb_plats: 4,  
    ...  
  }  
});
```

index.js

▷ Pour nous faciliter la vie, Vue ont créé un débogueur à installer via le [Chrome store](#)



Vue.js devtools

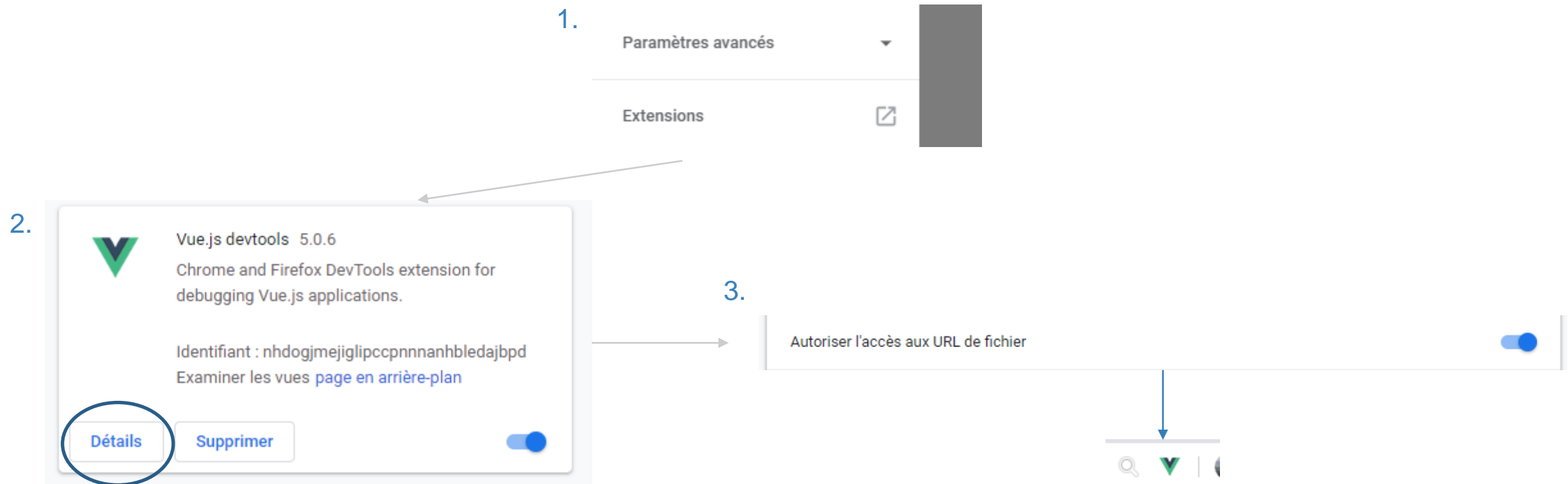
Proposé par : <https://vuejs.org>

★★★★★ 1415 | Outils de développement | 873 817 utilisateurs

Ajouter à Chrome

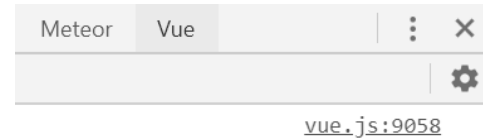
Débugger sur chrome & firefox

► Une fois cela fait, rendez-vous sur la page des paramètres, puis ouvrez la page relative aux extensions :

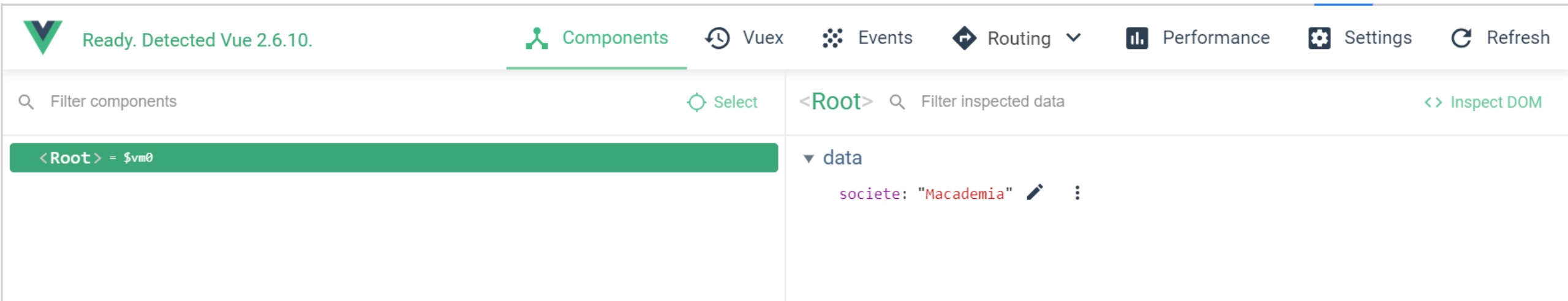


Débugger sur chrome & firefox

► Une fois l'installation réalisée, un nouvel onglet apparaît dans votre console :



► Grâce à lui, nous sommes désormais de consulter et modifier l'état de notre application en live :



Mise en place d'un projet

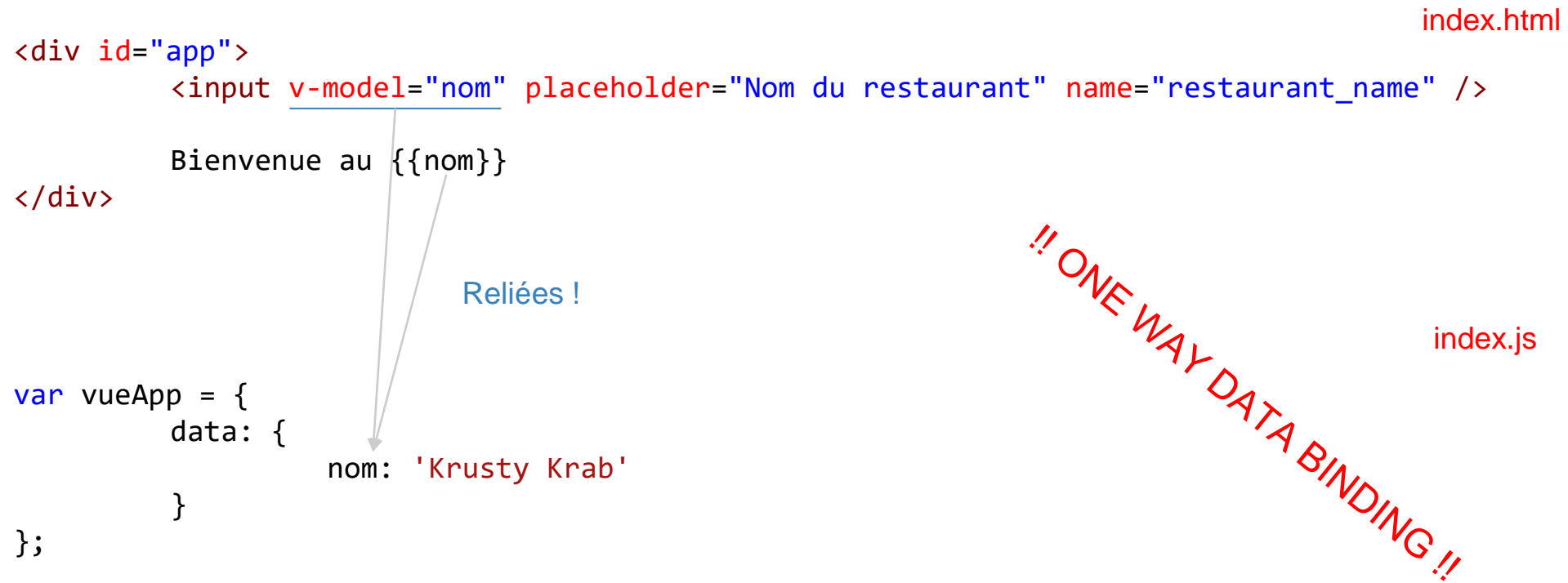
- Mettez en place un **nouveau projet local** et vérifiez que **Vue Devtools** le reconnaisse correctement



Relier le modèle et la vue

[Testez ce code !](#)

► VueJS permet de relier des variables du modèle avec le DOM




Bienvenue au Krusty Krab

Relier le modèle à la vue

- Une autre écriture est possible pour la définition des variables du modèle :

```
var vueApp = {  
  data() {  
    return {  
      nom: 'Krusty Krab'  
    }  
  }  
};
```



Une fonction ici !

index.js

- En javascript, l'écriture `{a}` est équivalente à `{a: a}`

Binding des propriétés

- ▷ Certaines propriétés peuvent être dynamique dans une page
 - › Placeholder / name / class / aria
- ▷ La directive v-bind fournie par Vue nous permet de dynamiser n'importe quel attribut d'un node HTML :

```
<div id="app">  
  <input v-model="nom" v-bind:placeholder="'Valeur dynamique : ' + nom" name="restaurant_name" />  
  
  <p>Bienvenue au {{nom}} </p>  
</div>
```

Comparaison avec JQuery

```
<div id="app">
  <input id="nom" name="restaurant_name" />

  <p id="content"></p>
</div>
```

index.html

```
$('#nom').on('keyup', function () {
  var message = $('#nom').val();
  $('#content').text(message);
});
```

index.js

Méthodes et interpolation

- ▷ Nous souhaitons afficher le **nombre de client actuel**
 - › Pour ce faire, nous allons passer par une fonction qui nous renverra un nombre aléatoire entre 0 et 5
- ▷ En VueJS, il est possible de déclarer des **méthodes** qui pourront être appelées lors d'un évènement
 - › Ou tout simplement entre des `{{ mustache tags }}` pour renvoyer une vue !
- ▷ Toutes les méthodes doivent être présentes dans la propriété « **methods** » de votre instance

Méthodes et interpolation

[Testez ce code !](#)

index.js

```
...
methods: {
  getNbClients() {
    let nbClients = Math.round(Math.random() * 5);

    if (nbClients === 0) {
      return `aucun client`;
    } else if (nbClients === 1) {
      return `${nbClients} client`;
    } else {
      return `${nbClients} clients`;
    }
  }
}
```

index.html

```
...
  Bienvenue au {{nom}}, nous avons {{ getNbClients() }}
</div>
```

Evènements globaux

- Pour notre restaurant, il va être plus pertinent d'afficher des boutons pour permettre aux client d'y rentrer et d'en sortir :

Bienvenue au Krusty Krab, nous avons 1 client

Entrer Sortir

- Pour y arriver, VueJS nous offre plusieurs directives permettant de réagir aux évènements utilisateurs

```
...  
Bienvenue au {{nom}}, nous avons {{getNbClients()}}  
  
<div class="actions">  
  <button v-on:click="enter()">Entrer</button>  
  <button @click="leave()">Sortir</button>  
</div>
```

v-on: et @ sont équivalents

index.html

Evènements globaux

[Testez ce code !](#)

► Les réactions correspondent à des méthodes définies dans l'instance :

```
data() {  
  return {  
    nom: 'Krusty Krab',  
    nbClients: 0  
  }  
},  
methods: {  
  ...  
  enter() {  
    this.nbClients++;  
  },  
  leave() {  
    if (this.nbClients <= 0) return;  
    this.nbClients--;  
  }  
  ...  
}
```

Le this référence l'instance de la vue actuelle

index.js

Modificateurs d'évènements

[Testez ce code !](#)

▷ Le framework offre la possibilité d'**ajouter des contrôles** sur les évènements : interdire, limiter le nombre d'appel ou même limiter le scope d'appel !

```
<div id="app">
  <!-- Ce bouton ne réagira QU'UNE FOIS ! -->
  <button @click.once="displayBox('Bonjour !')">Une fois</button>
</div>
```

index.html

▷ Voici la liste des modificateurs disponibles :

- › **.stop** : Stop la propagation de l'évènement auquel il est associé
- › **.prevent** : Annule le comportement par défaut de l'évènement (pour un formulaire, empêche l'évènement *submit* de recharger la page)
- › **.capture** : Réagit à chaque étape durant la phase de « capture » d'un évènement
- › **.self** : Ne lance la fonction que si l'évènement est bien sur cet élément et non pas sur un des fils
- › **.once** : N'appelle l'évènement qu'une et unique fois

▷ Il est possible de **chaîner** plusieurs modificateurs

Ecouter les touches claviers

- ▷ Pour les évènements clavier, la logique est similaire avec les « key modifiers »
 - › Le framework récupère le code ASCII ou la valeur définie dans la configuration pour réagir :

```
<input v-on:keyup.65="maMethode"> <!-- Touche A majuscule -->  
<input v-on:keyup.enter="maMethode"> <!-- Touche entrée -->
```

index.html

- ▷ On peut évidemment chaîner ces modifieurs :

```
<input v-on:keyup.ctrl.enter="maMethode"> <!-- CTRL + Entrée -->
```

index.html

- ▷ *Avancé - Vous pouvez définir vos propres raccourcis en configurant VueJS :*

```
// Binder la touche F1 (ascii 112) au mot-clé f1  
Vue.config.keyCodes.f1 = 112;
```

index.js

Directives « structurelles »

- ▷ Nous avons à notre disposition pléthore de directives que nous découvrirons au long du cours
- ▷ Parmi celles-ci, deux se démarquent : **v-if** et **v-for**
- ▷ On les appelle directives structurelles car elles modifient la structure du DOM en ajoutant / supprimant des *nodes* HTML

V-IF

[Testez ce code !](#)

- ▷ « v-if » prend en valeur un booléen et se chargera d'afficher, ou non, l'élément sur lequel il est appliqué
- ▷ Cette directive est ré-évaluée à chaque changement sur la page

index.html

```
<div id="app">
  <input v-model="nom" placeholder="Nom du restaurant" name="restaurant_name" />

  Bienvenue au {{nom}}
  <span v-if="nbClients > 0">, nous avons {{ getNbClients() }}</span>

  <div class="actions">
    <button v-on:click="enter()">Entrer</button>
    <button @click="leave()">Sortir</button>
  </div>
</div>
```

V-IF

- En complément du `v-if`, il existe « `v-else-if` » et « `v-else` »

```
<div v-if="condition1">  
    Premier cas de figure  
</div>  
<div v-else-if="condition2">  
    Deuxieme  
</div>  
<div v-else>  
    Sinon...  
</div>
```

index.html

- Une directive semblable à `v-if` existe : `v-show`, mais contrairement à la directive structurelle, celle-ci se contente d'ajouter un « `display: none` » à l'élément

V-IF vs V-SHOW

- ▷ Question à 1 million, quand utiliser l'un, et quand utiliser l'autre ?
- ▷ Dans la plupart des cas, v-if est la meilleure solution ;
- ▷ En terme de performance, le v-if a un coût de changement d'état plus cher que v-show
 - › Donc préférez *v-show* si le toggle va être *provoqué plusieurs fois* !
- ▷ L'avantage du v-if est que si la condition est faux au rendu initial, il ne fera rien, donc la page s'affichera plus rapidement
 - › Donc préférez *v-if* si l'affichage est *peu probable au chargement de la page*

V-FOR

▷ « v-for » permet d'itérer sur un tableau de votre modèle afin d'afficher l'élément HTML sur lequel il est appliqué, autant de fois qu'il n'y a d'élément dans le tableau

▷ Commençons par créer notre tableau dans le modèle

```
var vueApp = new Vue({  
  el: '#app',  
  data: {  
    nom: 'Krusty Krab',  
    nbClients: 0,  
    menu: [  
      'Filet de saumon - 15€',  
      'Salade de fruit - 9€',  
      'Salade niçoise - 120€'  
    ]  
  },  
  ...  
})
```

index.js

V-FOR (2)

[Testez ce code !](#)

- ▷ Il s'agit maintenant d'itérer le tableau pour afficher la carte des menus

```
<ul>  
  <li v-for="food of menu">  
    {{ food }}  
  </li>  
</ul>
```

index.html

- ▷ La vue se met à jour automatiquement dès la moindre modification faite sur le tableau

On récupère ainsi l'index actuel

```
<li v-for="(food, index) of menu">  
  [{{index}}] - {{ food }}  
</li>
```

index.html

- ▷ Ne JAMAIS utiliser v-if et v-for sur un même élément

V-FOR – Optimisation des performances

- ▷ Lors du réaffichage de la vue, certaines problématiques peuvent provoquer la lourdeur pour la mise à jour d'un v-for
- ▷ Pour optimiser les performances, il est recommandé d'appliquer une propriété « :key » unique à chaque itération :

```
<tr v-for="movie in movies" :key="movie.id"></tr>
```


Afficher l'état de vos variables

▷ Si pour une raison quelconque, vous souhaitez afficher votre objet « data » dans le DOM, vous avez la possibilité d'appliquer une écriture simplifiée pour transformer son format en string (*lightweight interchange format*). La vue se met à jour automatiquement

```
data: {  
  obj: {  
    a: 1,  
    b: 2,  
    c: {  
      d: 3  
    }  
  }  
}
```

index.js

```
<div id="app">  
  {{ $data }}  
</div>
```

index.html

Résultat : { "obj": { "a": 1, "b": 2, "c": { "d": 3 } } }

Questions