



# Introduction à Vue.js (v3)

Philippe Genoud

*Philippe.Genoud@univ-grenoble-alpes.fr*

Dernière mise à jour : 07/02/2024 12:41



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

# Qu'est-ce que Vue.js ?

- Vue.js : un framework JavaScript open-source et progressif pour faciliter la construction d'interfaces utilisateur (UI) web.
- Développé initialement par Evan You (2014)
- Vue3 dernière version (3.0 sept. 2020 , 3.2.47 fév. 2023)
  - <https://github.com/vuejs/core/blob/main/CHANGELOG.md>
- Deux principales caractéristiques
  - **Rendu déclaratif** : Vue étend le HTML standard avec une syntaxe de modèle (templates) qui nous permet de décrire de manière déclarative la sortie HTML en fonction de l'état JavaScript.
  - **Réactivité** : Vue suit automatiquement les changements d'état de JavaScript et met efficacement à jour le DOM lorsque des changements se produisent.
- Conçu comme un framework flexible, léger et extensible il permet de couvrir une large gamme d'applications
- Souvent comparé aux autres frameworks front-end les plus populaires que sont

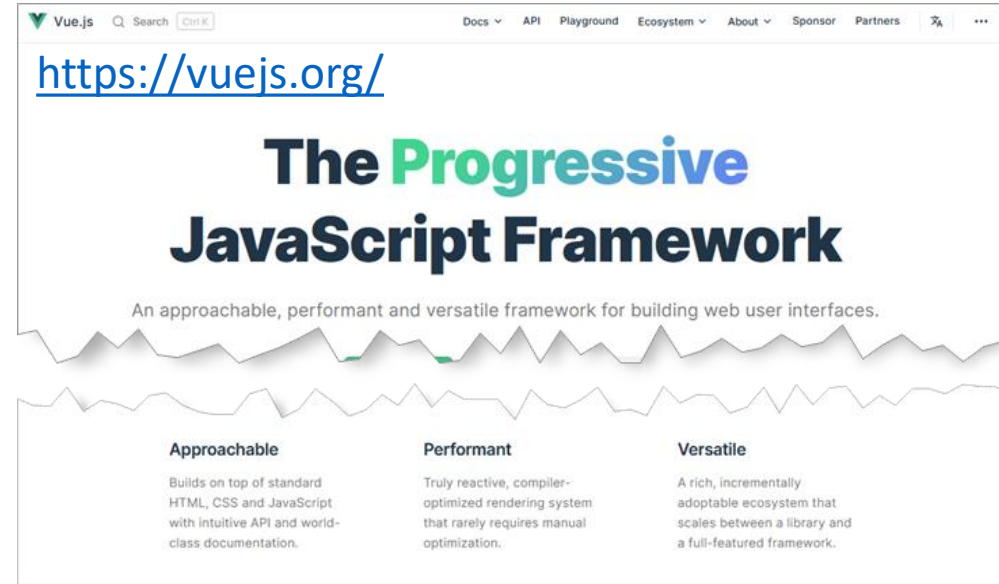
Angular  
(Google)



React.js  
(Meta)  
(ex Facebook)



Svelte  
(projet open source  
initié par Rich Harris)



## The Progressive Framework

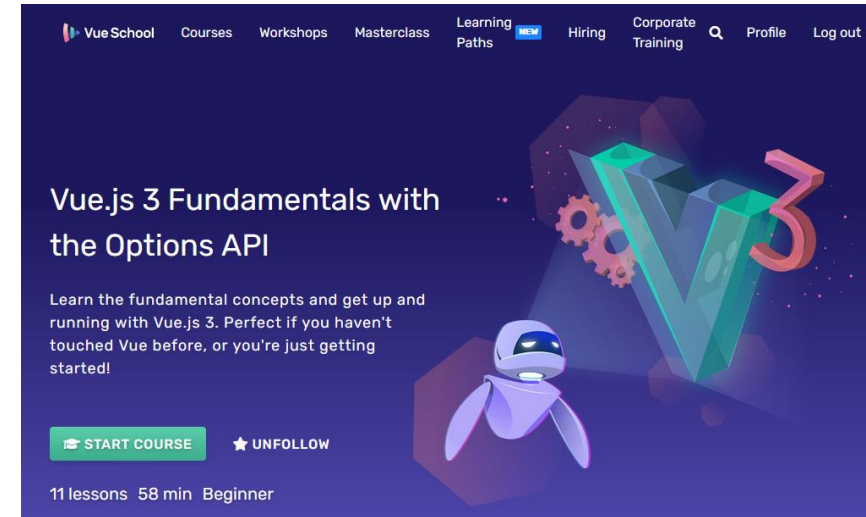
Vue is a framework and ecosystem that covers most of the common features needed in frontend development. But the web is extremely diverse - the things we build on the web may vary drastically in form and scale. With that in mind, Vue is designed to be flexible and incrementally adoptable. Depending on your use case, Vue can be used in different ways:

- Enhancing static HTML without a build step
- Embedding as Web Components on any page
- Single-Page Application (SPA)
- Fullstack / Server-Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- Targeting desktop, mobile, WebGL, and even the terminal

<https://vuejs.org/guide/extras/ways-of-using-vue.html>

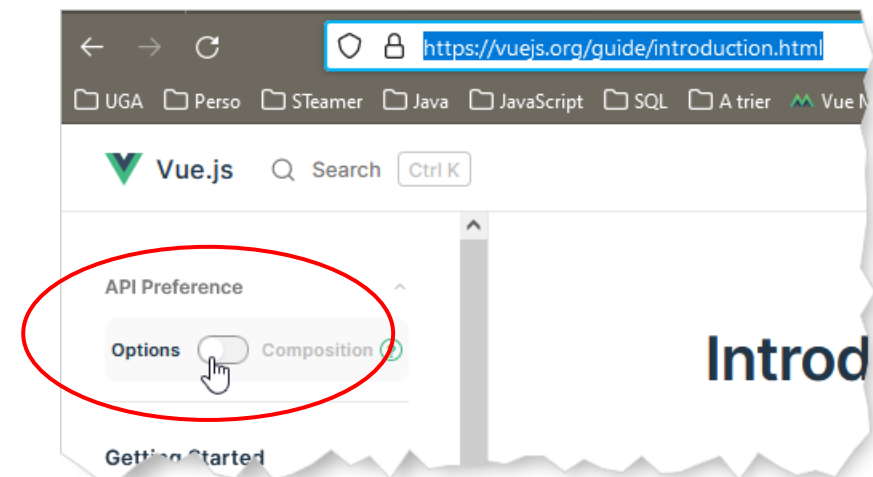
# Tutorial d'introduction à Vue3

- Tutorial inspiré de "*Vue.js 3 Fundamentals with the Options API*" de vueschool.io  
<https://vueschool.io/courses/vuejs-3-fundamentals>
  - Syntaxe de template
  - Liaison bidirectionnelle des données de formulaires
  - Gestion des événements utilisateur
  - Méthodes
  - Rendu conditionnel
  - Liaison d'attributs HTML



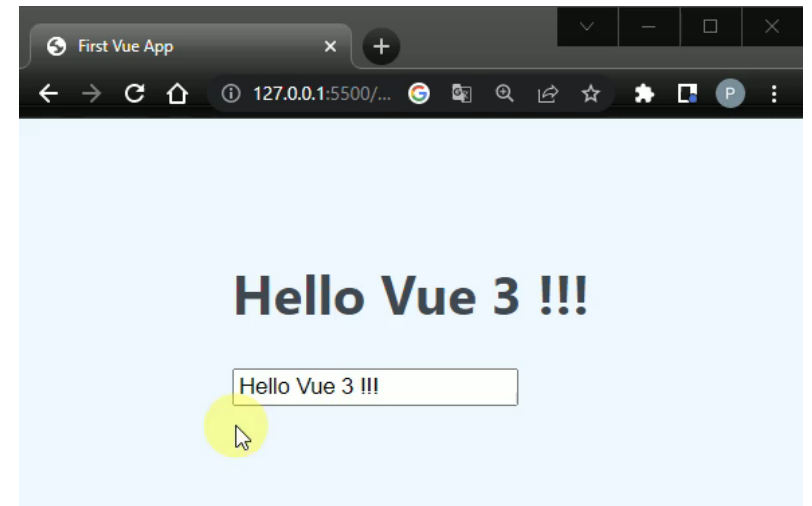
- Attention : dans le guide Vue3 activer la préférence Options API

<https://vuejs.org/guide/introduction.html>



# Hello Vue !

- Utilisation de Vue dans un simple fichier HTML



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>First Vue App</title>
  <link rel="stylesheet" href="main.css">
</head>
<body>
  <div id="hello-message">
    <h1>{{message}}</h1>
    <input v-model="message">
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const helloMessage = Vue.createApp({
      data() {
        return {
          message: "Hello Vue 3 !!!"
        }
      }
    })
    helloMessage.mount("#hello-message");
  </script>
</body>
</html>
```

`{{ ... }}` doubles moustaches

Récupère la valeur de la propriété **message** de l'objet données de l'instance de Vue

Directive **v-model** qui permet de lier la valeur de l'input à la propriété **message** de l'objet données de l'instance de Vue

Importe Vue3 depuis son CDN

`Vue.createApp({ ... })` création d'un objet Vue, avec en paramètre un objet définissant les options de configuration

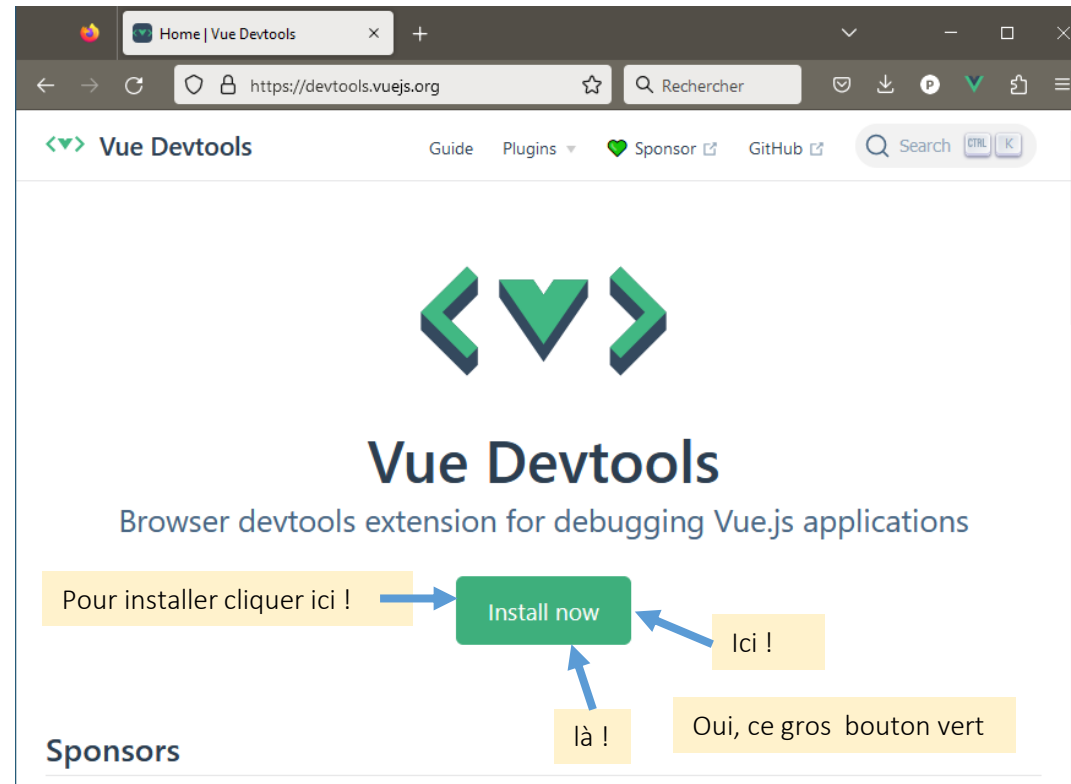
Fonction **data** qui retourne un objet avec une propriété **message**.  
L'objet retourné par **data()** définit les données utilisées par l'instance de Vue

Associe l'objet Vue au **div** d'identifiant **hello-message**

# Vue Devtools

- Vue Devtools extension du browser pour Firefox, Chrome et Edge qui permet d'interagir et de déboguer une application Vue
  - Indispensable au développement avec Vue

<https://devtools.vuejs.org/>



# Vue Devtools sur Chrome

Attention si vous passez par le web store de Chrome.... choisissez la bonne version (6.5.0 au 22/02/2023)

1 Visiter le site du webstore de chrome (<https://chrome.google.com/webstore>)

2 Chercher Vue Devtools

3 Sélectionner Version actuelle 6.x (supporte Vue3 et Vue 2)

Version bêta (pour la tester la v 6.x, mais n'est plus active)

Version legacy 5.x pour les versions antérieures de Vue (Vue2)

The screenshot shows the Chrome Web Store search results for 'Vue.js devtools'. The search bar contains 'Vue devtools'. The results list three extensions: 'Vue.js devtools' (beta), 'Vue.js devtools' (current version 6.x), and 'Vue.js devtools' (legacy 5.x). The current version is highlighted in green.

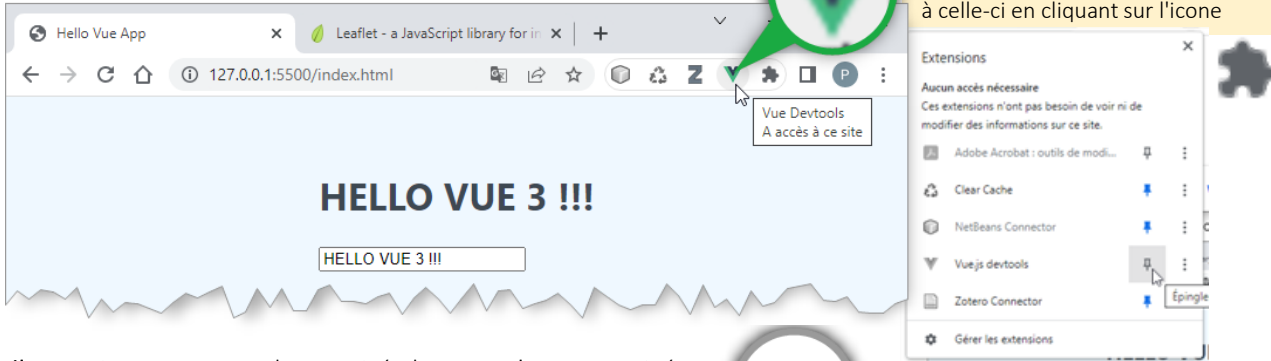
4 Ajouter l'extension

5 Ajouter l'extension

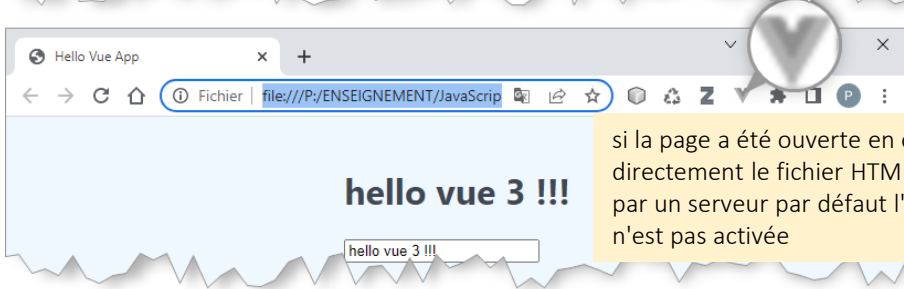
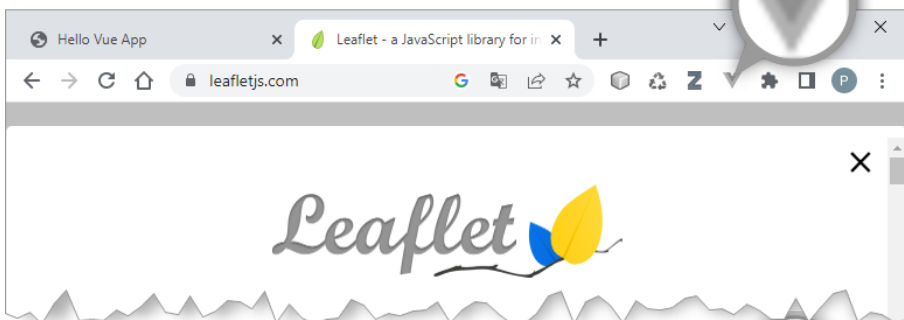
The screenshot shows the detail page for 'Vue.js devtools' on the Chrome Web Store. The 'Add to Chrome' button is highlighted with a yellow box and the number 4. A modal dialog is open, asking to install the extension, with the 'Ajouter l'extension' button highlighted with a yellow box and the number 5.

# Vue Devtools sur Chrome

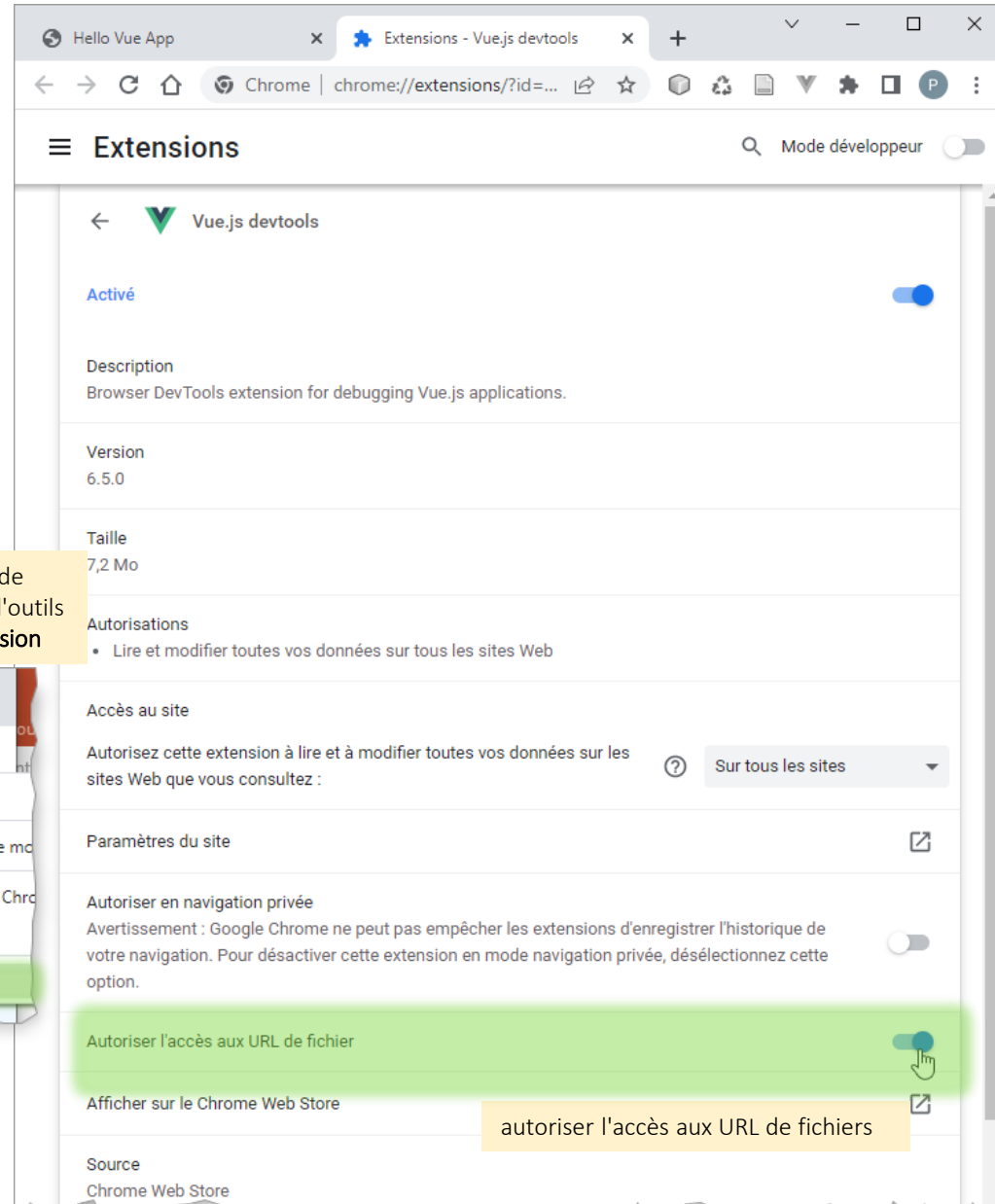
Si la page contient une Vue Vue.js l'extension est activée



l'extension Vue Devtools est grisée lorsque n'est pas activée



Pour activer l'extension Vue Devtools sur les URLs de fichiers





# Vue Devtools sur Chrome

The image shows a Chrome browser window with a Vue.js application running. The application displays "hello vue 3 !!!" in a text input field. The Vue Devtools interface is overlaid on the browser, showing the component tree and the data of the selected component. The component tree shows a root component with a data property containing a message. The console shows the Vue instance logs, including the message being displayed.

version de Vue.js

Sélectionner l'onglet Vue

possibilité de les modifier directement

Instance de l'application et des éventuels composants

les données associées au composant sélectionné

Vue Devtools injecte directement les instances de **Vue** dans la console pour que l'on puisse les manipuler directement sans avoir à déclarer explicitement une variable

`$vm0` désigne la première instance (ici il n'y en a qu'une) de **Vue**



# Syntaxe de templates Vue

- `{{ ... }}` doubles moustaches pour lier des données au DOM

```
<body>
  <div id="hello-message">
    <h1>{{message}}</h1>
    <input v-model="message">
  </div>
  <script
src="https://unpkg.com/vue@3"></script>
  <script>
    const helloMessage = Vue.createApp({
      data() {
        return {
          message: "Hello Vue 3 !!!"
        }
      }
    })
    .mount("#hello-message");
  </script>
</body>
```

Hello Vue 3 !!!

Hello Vue 3 !!!

- Possibilité d'utiliser n'importe quelle expression JavaScript
- Mais, uniquement une et une seule expression est acceptée

```
<h1>{{message.toLocaleUpperCase()}}</h1>
```

```
<h1>{{console.log(message); message.toLocaleUpperCase()}}</h1>
```



```
<h1>{{ if (! message) { return "Welcome !" } }} </h1>
```

❌ instruction `if` n'est pas une expression

```
<h1>{{ message?message:"Welcome !" }} </h1>
```

✅ expression ternaire

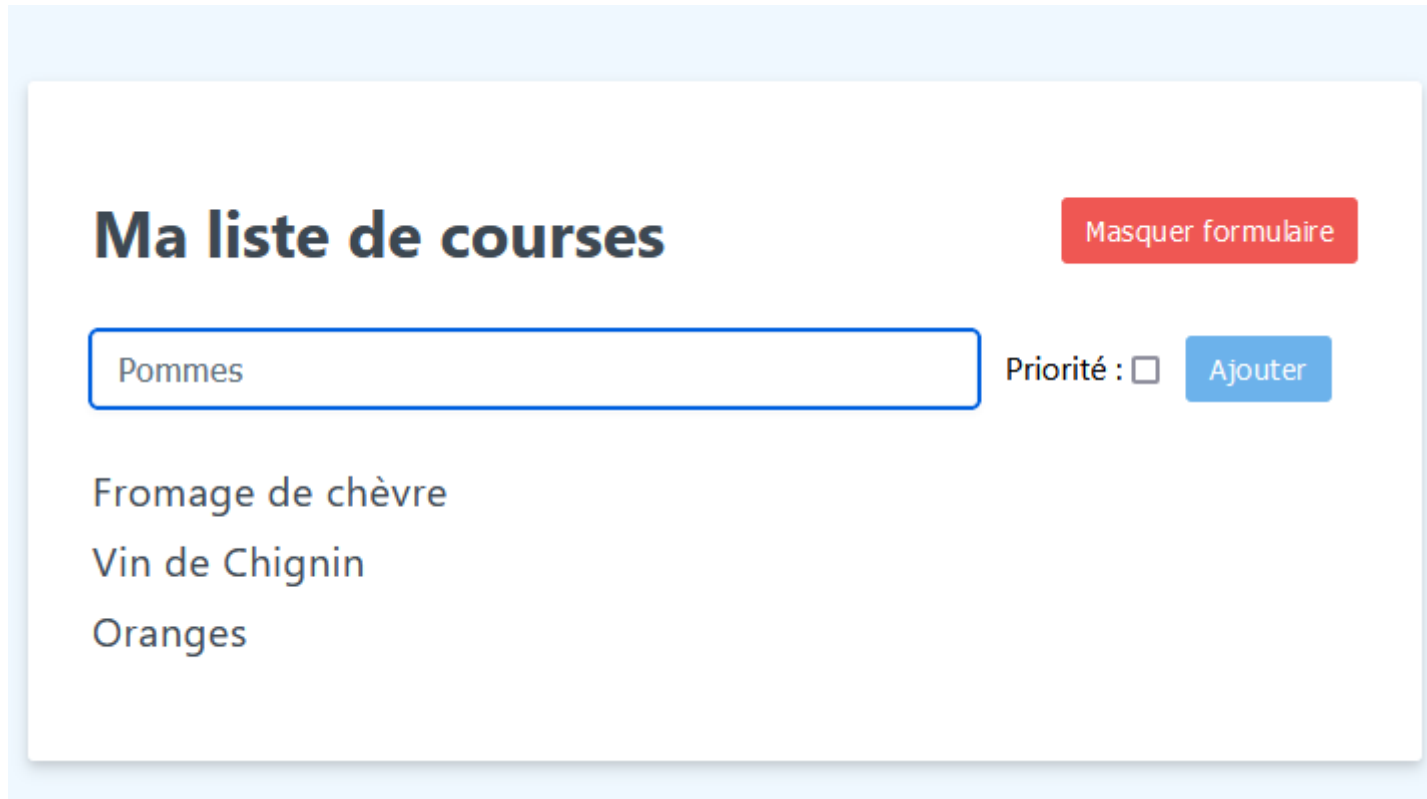
```
<h1>{{ message || "Welcome !" }} </h1>
```

✅ expression booléenne

```
[Vue warn]: Template compilation error: Error parsing JavaScript expression: Unexpected token ';'
1 |
2 |   <h1>{{console.log(message); message.toLocaleUpperCase()}}</h1>
3 |   ~~~~~
4 |   <input v-model="message">
   at <App>
Uncaught SyntaxError: missing ) after argument list
at new Function (anonymous)
    compileFunction (vue@3:15927:23)
```

# Exemple : gérer une liste de courses

- Pour aller plus loin avec la découverte de Vue on va développer une application de gestion d'une liste de courses



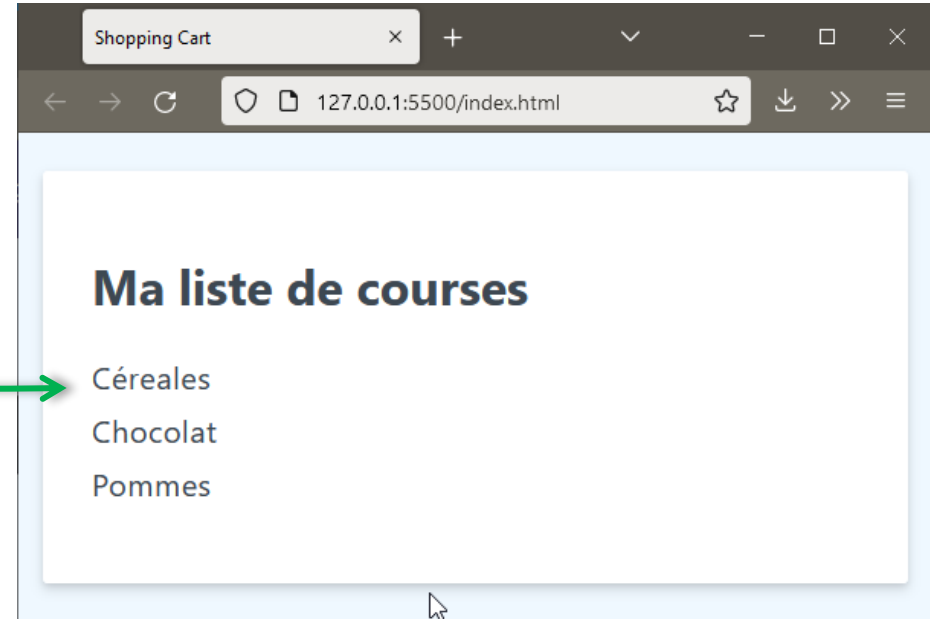
The screenshot shows a web application interface for managing a shopping list. The title is "Ma liste de courses". There is a red button labeled "Masquer formulaire" in the top right. Below the title, there is a text input field containing "Pommes". To the right of the input field is a "Priorité :  Ajouter" label and a blue "Ajouter" button. Below the input field, the list of items is displayed: "Fromage de chèvre", "Vin de Chignin", and "Oranges".

- Syntaxe de template
- Liaison bidirectionnelle des données de formulaires
- Gestion des événements utilisateur
- Méthodes
- Rendu conditionnel
- Liaison d'attributs HTML

# Syntaxe de templates Vue

- Liste de courses affichée sous la forme d'une liste HTML ( `<ul>` )

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>
<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li>Céréales</li>
      <li>Chocolat</li>
      <li>Pommes</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses"
        }
      }
    })
    .mount("#shopping-list");
  </script>
</body>
</html>
```



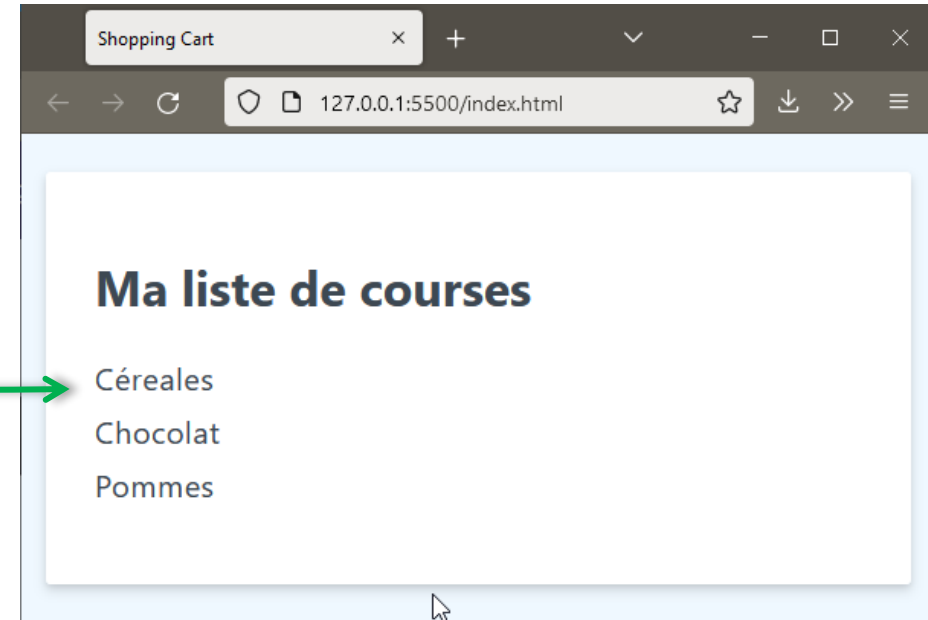
```
ul {
  list-style: none;
  padding: 0;
}
```

# Syntaxe de templates Vue

- Directive **v-for** : pour itérer et afficher le contenu de tableaux ou d'objets

```
<!DOCTYPE html>
<html lang="en">
<head>
  ...
</head>
<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li>Céréales</li>
      <li>Chocolat</li>
      <li>Pommes</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses"
        }
      }
    })
    .mount("#shopping-list");
  </script>
</body>
</html>
```

Liste de courses affichée sous la forme d'une liste HTML ( `<ul>` )



plutôt que de *coder en dur* la liste de courses dans le code HTML, l'idée est de la définir comme une propriété (*data property*) de l'objet Vue par exemple sous la forme d'un tableau

# Vue templating syntax

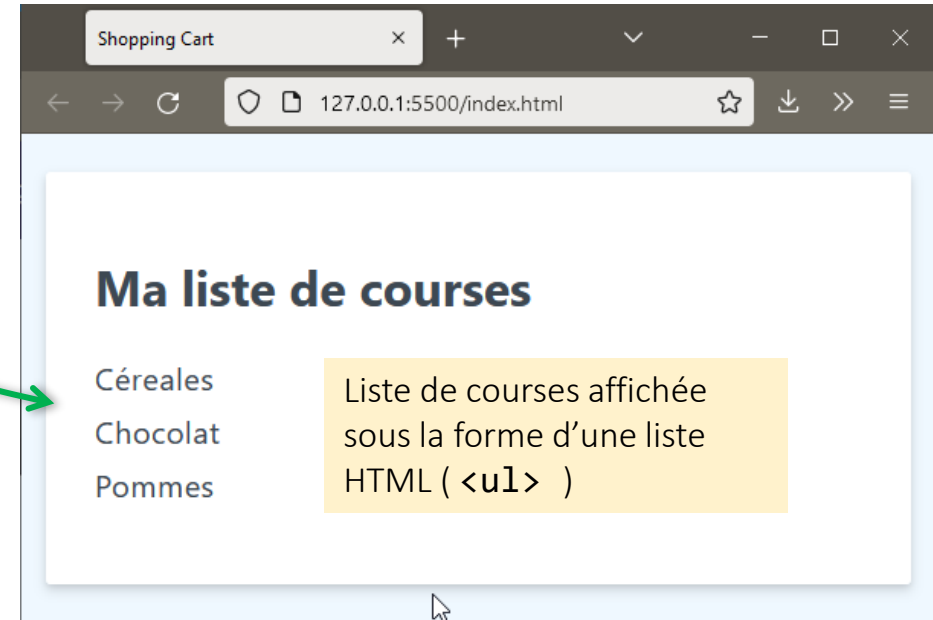
- Directive **v-for** : pour itérer et afficher le contenu de tableaux ou d'objets

```
<head>
  ...
</head>

<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li v-for="item in items">{{item}}</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses",
          items: [
            "Céréales",
            "Chocolat",
            "Pommes",
          ]
        }
      }
    })
    shoppingListApp.mount("#shopping-list");
  </script>
</body>
</html>
```

Pour chaque élément (**item**) du tableau **items** on crée un élément **li** qui affiche sa valeur

la liste est un attribut de l'objet instance de Vue (**items** tableau de string)



# Vue templating syntax

- Directive **v-for** : pour itérer et afficher le contenu de tableaux ou d'objets

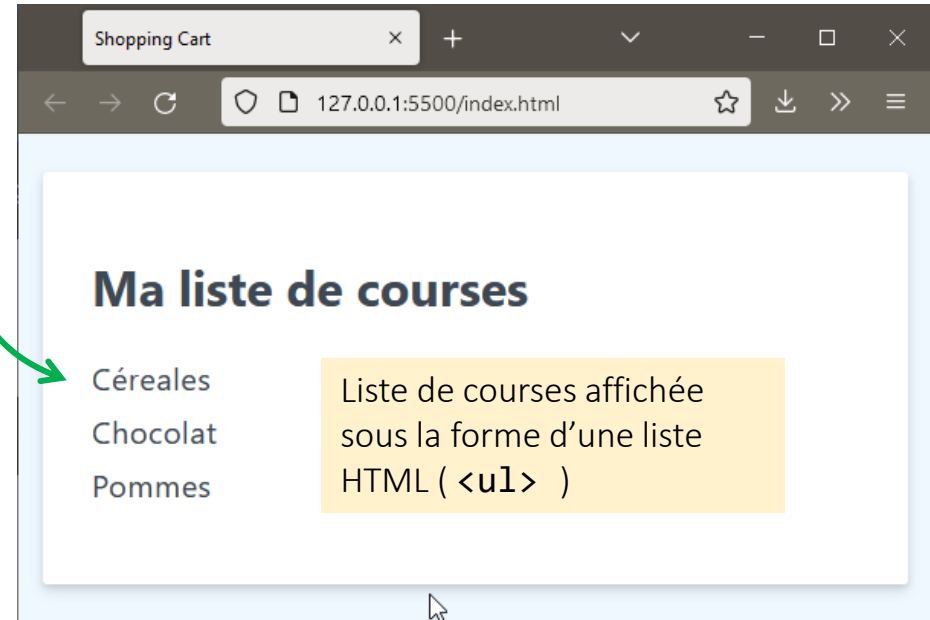
```
<head>
  ...
</head>

<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li v-for="item in items" :key="item.id">{{item.label}}</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses",
          items: [
            {id : 1, label : "Céréales"},
            {id : 2, label : "Chocolat"},
            {id : 3, label: "Pommes"},
          ]
        }
      }
    })
    .mount("#shopping-list");
  </script>
</body>
</html>
```

Pour chaque élément (**item**) du tableau **items** on crée un élément **li** qui affiche sa valeur

Attribut **:key** avec un identifiant unique pour chaque élément permet d'optimiser le rendu itératif d'une liste

[Tips and Gotchas for Using key with v-for in Vue.js 3](#)



Pour afficher devant chaque **item** son **id**

```
<li v-for="item in items" :key="item.id">{{item.id}} {{item.label}}</li>
```

Possibilité d'utiliser la *déstructuration* ou décomposition d'objets pour simplifier l'écriture

```
<li v-for="{id, label} in items" :key="id">{{id}} {{label}}</li>
```

déstructure un objet **item** en deux variables, évite ensuite d'utiliser la notation pointée

**Ma liste de courses**

- 1 Céréales
- 2 Chocolat
- 3 Pommes

# (JavaScript : affectation par décomposition ...

- **Object** et **Array** deux des structures de données les plus utilisées en JavaScript
- affectation par décomposition permet de *déstructurer* des tableaux ou objets dans un ensemble de variables
  - par exemple pour passer des données à une fonction, on n'a pas besoin d'un tableau ou d'un objet mais d'un ensemble de valeurs (éventuellement ne correspondant qu'à une partie de l'objet ou du tableau)

exemples déstructuration de tableaux :

```
let tab = ["Joe", "Moose", 30, "Canada", "CA"];
```

```
let prenom = tab[0];  
let nom = tab[1]; } let [prenom, nom] = tab;
```

```
function afficher(prenom, nom) {  
  console.log(`prénom : ${prenom}`);  
  console.log(`nom : ${nom}`);  
}
```

```
afficher([firstname, lastname] = tab);
```

```
let [prenom, , age] = tab;
```

```
let [prenom, nom, , ...pays] = tab;
```

```
let [prenom, nom, age] = ["Joe", "Moose"];
```

```
let [prenom, nom, age = 30] = ["Joe", "Moose"];
```

instruction de décomposition

si le tableau est plus long que la liste à gauche, les éléments supplémentaires sont ignorés

instruction de décomposition utilisée directement en lieu et place d'une liste d'arguments

possibilité d'ignorer des éléments du tableau à l'aide de virgules supplémentaires

possibilité de récupérer la fin du tableau (le tableau des éléments restants) en utilisant un paramètre supplémentaire précédé de ... `pays[0] → "Canada" pays[1] → "CA"`

si le tableau est plus court que la liste de valeurs à gauche, les valeurs manquantes sont **undefined**  
`age → undefined`

possibilité de définir des valeurs par défaut `age → 30`

les valeurs par défaut peuvent être des expressions ou même des appels de fonction



# JavaScript : affectation par décomposition ...

liste de variables      objet

- décomposition d'objets : diviser un objet en variables

```
let {var1, var2} = {var1:..., var2:...}
```

```
let joe = {  
  prenom : "Joe", nom : "Moose", age: 30, pays: "Canada", codePays : "CA"  
};
```

```
let nom = joe.nom;  
let prenom = joe.prenom;  
let pays = joe.pays;
```

```
let { nom, prenom , pays } = joe;
```

toutes les propriétés de l'objet en partie droite ne sont pas nécessairement en partie gauche  
l'ordre des propriétés n'a pas d'importance

```
let {prenom: firstname, nom: lastname} = joe;
```

possibilité d'affecter la valeur d'une propriété à une variable ayant un autre nom

```
let marie = {prenom : "Marie", nom: "Dupont", age: 27 };
```

```
let {nom, prenom, pays = "France", codePays = "FR"} = marie;
```

possibilité de définir des valeurs par défaut pour les propriété potentiellement manquantes  
les valeurs par défaut peuvent être des expressions ou même des appels de fonction

```
let {nom, prenom, age, ...infosPays} = joe;
```

si il y a moins de variables que de propriétés dans l'objet, possibilité de récupérer les propriétés restantes dans un objet    infosPays → { pays: "Canada", codePays : "CA" }

```
let nom, prenom;  
{ nom , prenom } = joe; ❌      { nom , prenom } = joe;  
SyntaxError: Unexpected token '='
```

erreur syntaxique { ... } est interprété comme un bloc de code

```
( { nom , prenom } = joe ); ✅
```

dans ce cas il faut envelopper l'expression entre parenthèses ( ... )

# JavaScript : affectation par décomposition

```
let grenoble = {  
  nom : "Grenoble",  
  codePostal : "38000",  
  coordonnees : {  
    lat : 45.1885,  
    lon : 5.7245  
  },  
  temperatures : [5, 12.5]  
};  
  
let {  
  nom, codePostal, coordonnees : {  
    lat, lon  
  },  
  temperatures : [ tmin, tmax ],  
  pays = "France"  
} = grenoble;
```

Si un objet ou un tableau contient d'autres objets et tableaux imbriqués possibilité de déstructurer les données imbriquées

```
let {  
  nom, codePostal, coordonnees : { lat, lon }, temperatures : [tmin, tmax], pays = "France"  
} = grenoble;
```

seules les variables définies au niveau le plus profond sont créées

- décomposition très utile pour des fonctions ayant plusieurs paramètres dont la plupart sont facultatifs (exemple constructeur de **Vue**)

```
function showMenu(title = "Untitled", width = 200, height = 100, items = []) {  
  // ...  
}  
  
showMenu("My Menu", undefined, undefined, ["Item1", "Item2"]);
```



Difficulté à se rappeler de l'ordre des paramètres  
Nécessité de passer valeur **undefined** lorsque l'on veut utiliser les valeurs par défaut

```
function showMenu({ title = "Untitled", width = 200, height = 100, items = [] } = {}) {  
  // ...  
}  
  
let options = {  
  title: "My menu",  
  items: ["Item1", "Item2"],  
};  
showMenu(options);
```



Les paramètres sont passés sous forme d'objets et la fonction les décompose en variables

Pas de problèmes d'ordre des paramètres  
Pas de problèmes de valeurs par défaut, dans options on ne met que les paramètres nécessaires

Permet d'appeler la fonction sans paramètre si on veut utiliser toutes les options par défaut  
`showMenu();`

exemple d'après [JAVASCRIPT.INFO](https://javascript.info/destructuring-assignment)  
<https://javascript.info/destructuring-assignment>

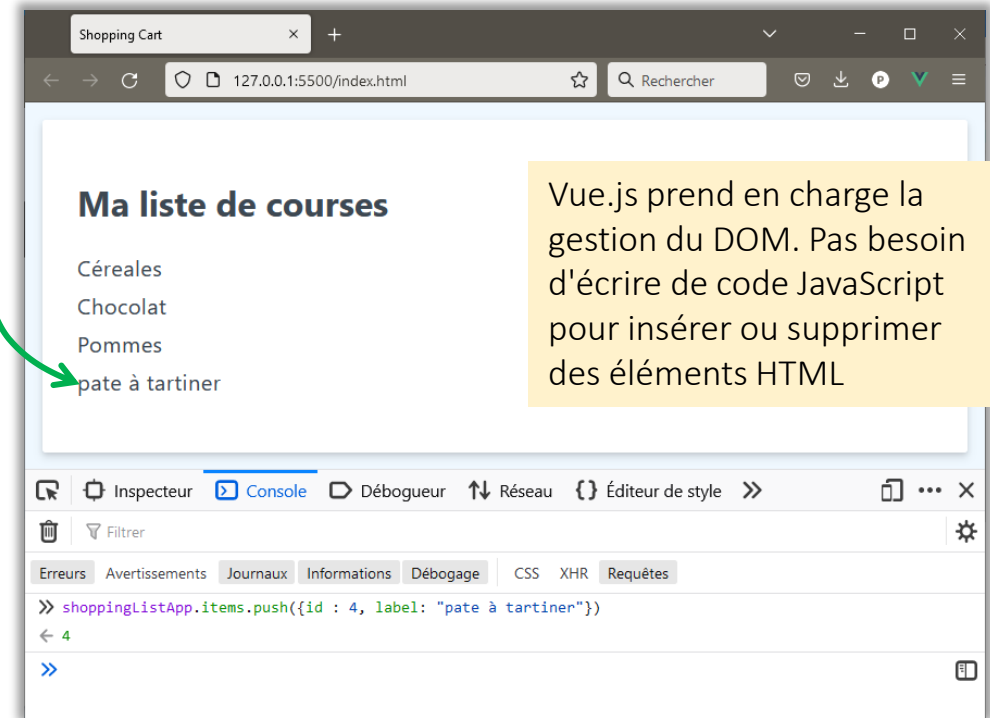
# Vue templating syntax

- Directive `v-for` : réagit aux modifications de l'objet lié

```
<head>
  ...
</head>

<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li v-for="item in items" :key="item.id">{{item.label}}</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses",
          items: [
            {id : 1, label : "Céréales"},
            {id : 2, label : "Chocolat"},
            {id : 3, label: "Pommes"},
          ]
        }
      }
    })
    .mount("#shopping-list");
  </script>
</body>
</html>
```

Liste de courses affichée sous la forme d'une liste HTML (`<ul>`) est réactive aux modifications du tableau `items`.

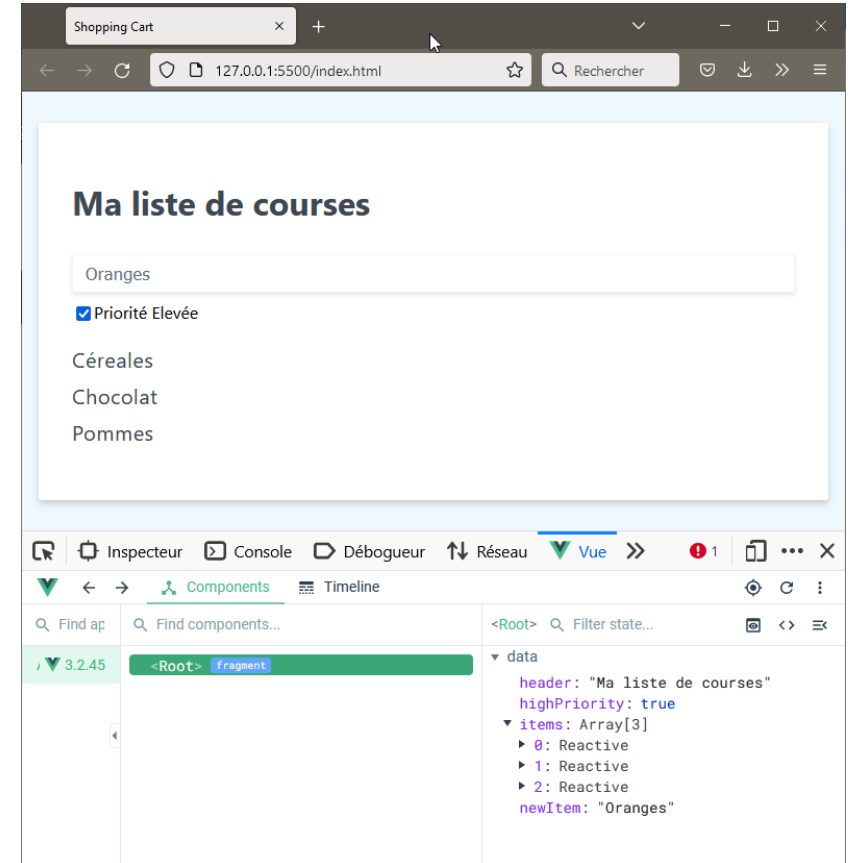


Rajoute un item au tableau items

```
shoppingListApp.items.push({id : 4, label: "pate à tartiner"})
```

# Vue : saisie de valeurs à l'aide d'inputs

```
<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <input v-model="newItem" type="text" placeholder="Ajouter un item">
    <label>
      <input type="checkbox" v-model="highPriority">Priorité Elevée
    </label>
    <ul>
      <li v-for="item in items" :key="item.id" >{{item.label}}</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "Ma liste de courses",
          newItem: "",
          highPriority: false,
          items: [
            {id: 1, label: "Céréales"},
            {id: 2, label: "Chocolat"},
            {id: 3, label: "Pommes"},
          ]
        }
      }
    })
    shoppingListApp.mount("#shopping-list");
  </script>
</body>
```



- la directive `v-model` permet de faire un binding bidirectionnel entre les inputs et les données du modèle
  - Lorsque l'input change le modèle est mis à jour
  - Lorsque le modèle change, l'input est mis à jour

# Vue : saisie de valeurs à l'aide d'inputs

- on peut associer aux directives `v-model` des *modifieurs* permettant d'altérer leur comportement par défaut
- syntaxe : `v-model.modifier`
- exemple

```
<input v-model="newItem"  
      type="text" placeholder="Ajouter un item">
```

## Ma liste de courses

 I  
Froma  
Céréales  
Chocolat  
Pommes

Par défaut la propriété `newItem` est mise à jour à chaque événement `keyup` (à chaque caractère saisi)

- autres modifieurs : `.trim`, `.number`

```
<input v-model.lazy="newItem"  
      type="text" placeholder="Ajouter un item">
```

## Ma liste de courses

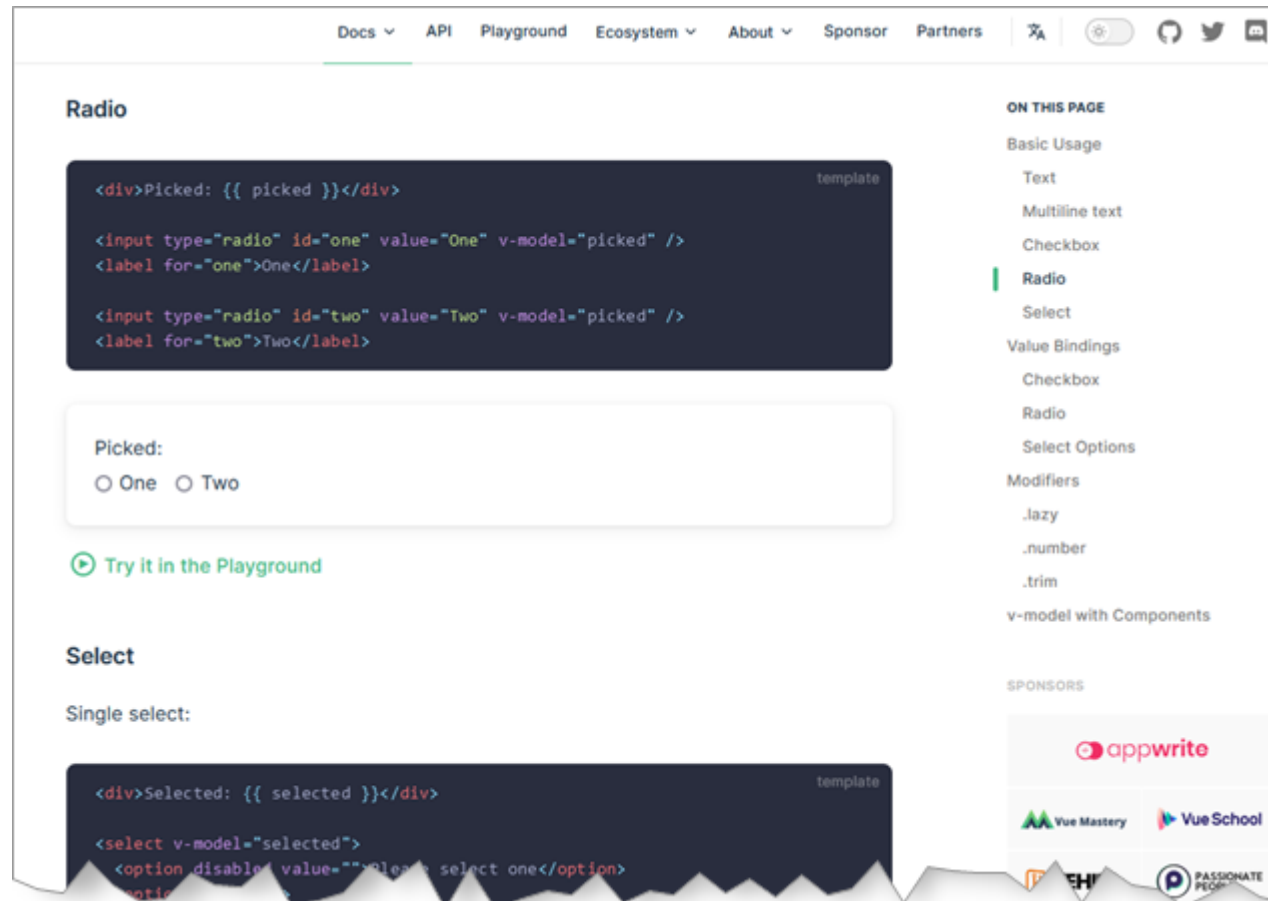
 I  
Céréales  
Chocolat  
Pommes

Avec le *modifieur* `lazy` la propriété `newItem` n'est mise à jour qu'après un événement `blur` (lorsque le champ de saisie perd le focus)

# Vue : saisie de valeurs à l'aide d'inputs

- `v-model` ne s'applique pas uniquement aux inputs de type texte, mais à toutes sortes d'input HTML5 : textareas, selects, checkboxes, radios boutons, ....

<https://vuejs.org/guide/essentials/forms.html>



The screenshot shows the Vue.js documentation page for the `Radio` component. It features a code editor with the following HTML code:

```
<div>Picked: {{ picked }}</div>
<input type="radio" id="one" value="One" v-model="picked" />
<label for="one">One</label>
<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
```

Below the code, there is a live preview of the component showing two radio buttons labeled "One" and "Two". A "Try it in the Playground" button is visible. The page also includes a table of contents on the right side, listing various topics like "Basic Usage", "Text", "Multiline text", "Checkbox", "Radio", "Select", "Value Bindings", "Modifiers", and "Sponsors".



The screenshot shows the Vue.js documentation page for the API Preference section. The "API Preference" section is highlighted with a red circle, and the "Options" button is selected. The "Composition" button is also visible. The page title is "Intro" and the URL is <https://vuejs.org/guide/introduction.html>.

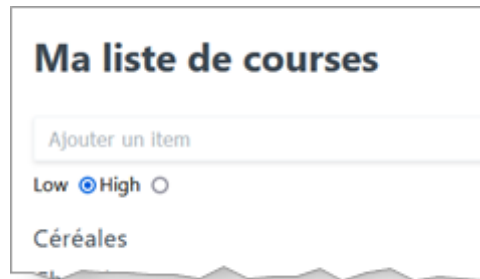
Attention : dans le guide Vue3  
activer la préférence  
Options API

# Vue : saisie de valeurs à l'aide d'inputs

- Ajout d'une propriété permettant de donner une priorité au nouvel item

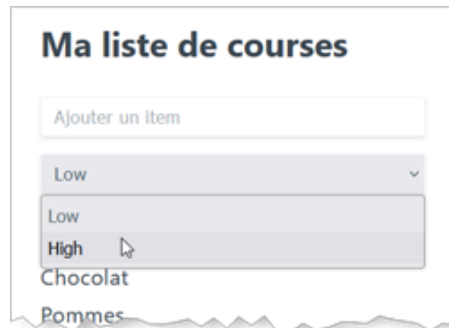
`newItemPriority: "low",`

- avec des Radios Boutons



```
<label>Low <input type="radio" v-model="newItemPriority" value="low"></label>  
<label>High <input type="radio" v-model="newItemPriority" value="high"></label>
```

- avec un liste de sélection



```
<select v-model="newItemPriority">  
  <option value="low">Low</option>  
  <option value="high">High</option>  
</select>
```

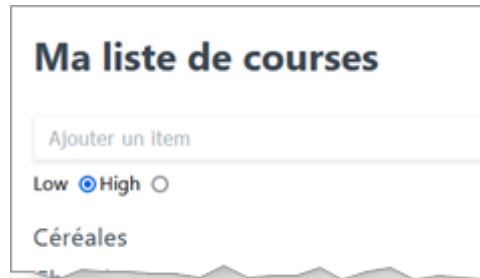


# Vue : saisie de valeurs à l'aide d'inputs

- Ajout d'une propriété permettant de donner une priorité au nouvel item

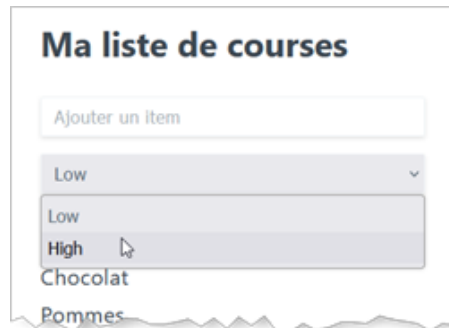
`newItemPriority: "low",`

- avec des Radios Boutons



```
<label>Low <input type="radio" v-model="newItemPriority" value="low"></label>  
<label>High <input type="radio" v-model="newItemPriority" value="high"></label>
```

- avec un liste de sélection



```
<select v-model="newItemPriority">  
  <option value="low">Low</option>  
  <option value="high">High</option>  
</select>
```

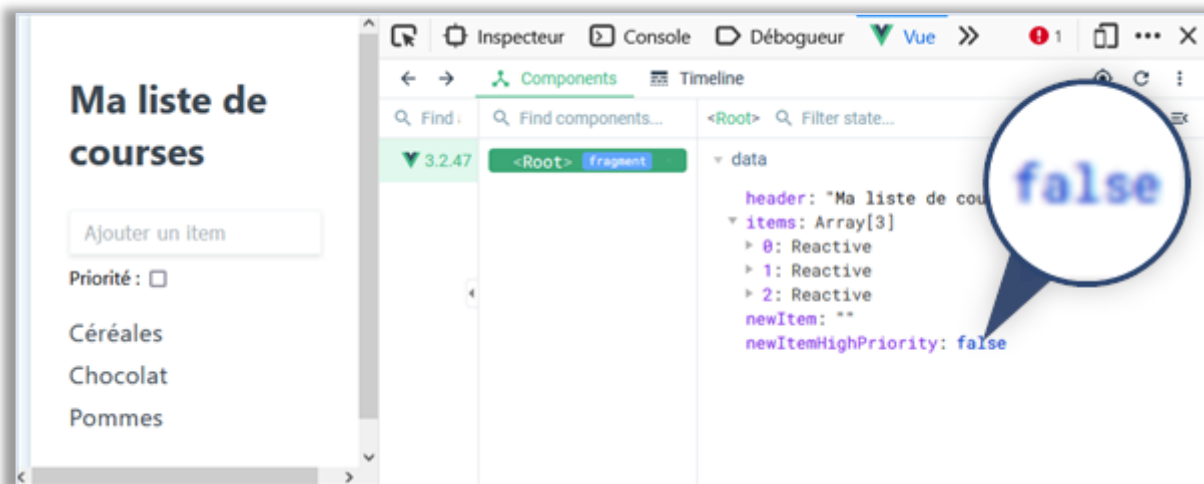
# Vue : saisie de valeurs à l'aide d'inputs

- définition de la priorité comme étant un booléen

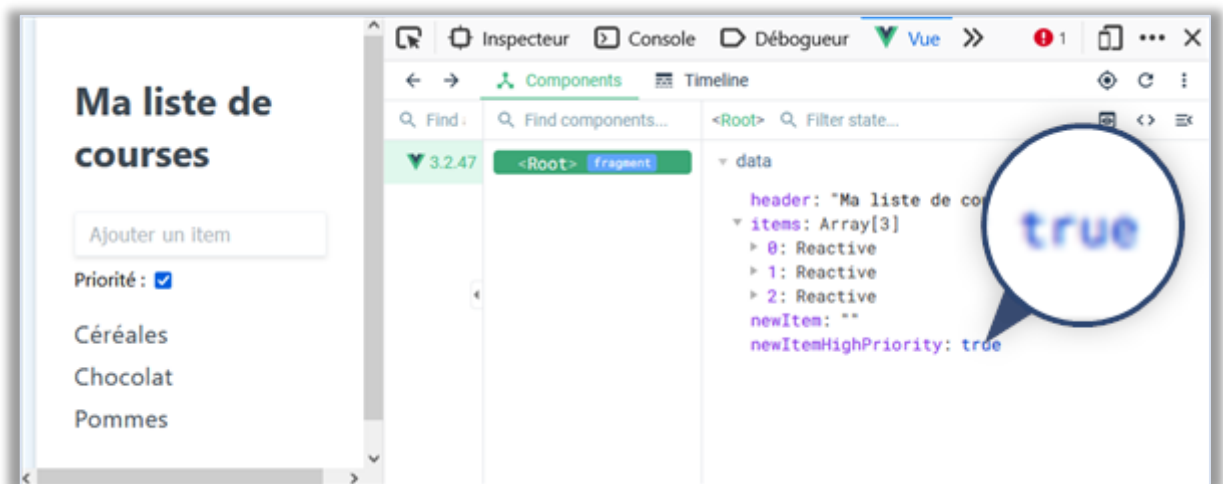
newItemHighPriority: `false`,

- utilisation d'une case à cocher (checkbox) pour définir son niveau

```
<label>  
  Priorité : <input type="checkbox" v-model="newItemHighPriority">  
</label>
```



case non cochée : valeur par défaut `false`



case cochée : valeur `true`

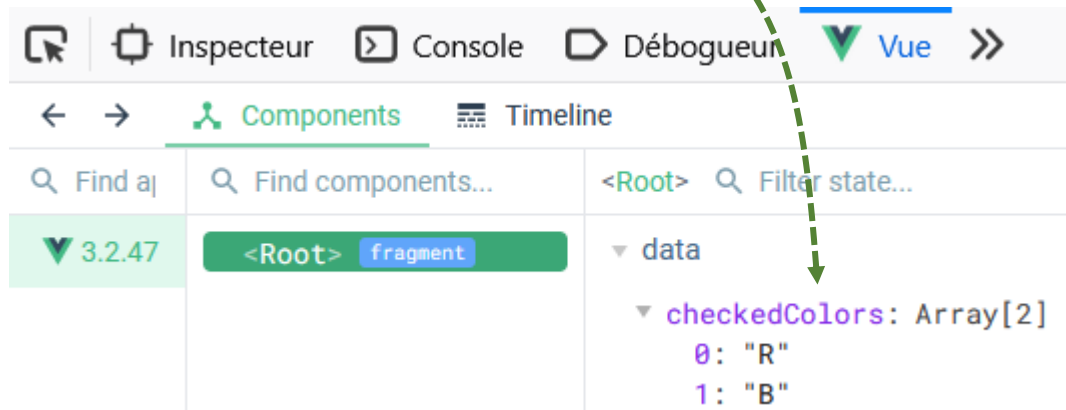
# Vue : saisie de valeurs à l'aide d'inputs

- cases à cocher pas limitées à une valeur booléenne, possibilité de l'associer à des choix multiples
  - plusieurs checkboxes liées a une propriété du modèle
  - cette propriété contiendra la liste (tableau) des valeurs des checkboxes sélectionnées
- exemple

Couleur : Rouge  Vert  Bleu

```
<label>Rouge <input type="checkbox" value="R" v-model="checkedColors"></label>  
<label>Vert<input type="checkbox" value="V" v-model="checkedColors"></label>  
<label>Bleu<input type="checkbox" value="B" v-model="checkedColors"></label>
```

le tableau et les checkboxes sont synchronisés



```
Vue.createApp({  
  data() {  
    return {  
      ...  
      checkedColors: [ ],  
      ...  
    }  
  }  
  ...  
}).mount(...)
```

# gérer les événements utilisateur

- Vue exploite la puissance des événements JavaScript au travers d'une syntaxe déclarative simple qui permet de réagir facilement aux interactions des utilisateurs
- la directive `v-on` permet d'associer un gestionnaire d'événement à un élément HTML
  - `v-on:type-event="code javascript à exécuter"`
- exemple : bouton pour ajouter le nouvel item à la liste des courses



le gestionnaire d'événement associé à un click sur le bouton doit ajouter un nouvel item à la liste

```
items.push({id : items.length + 1, label : newItem})
```

```
<div class="add-item-form">  
  <input v-model="newItem" type="text" placeholder="Ajouter un item">  
  <label>  
    Priorité : <input type="checkbox" v-model="newItemHighPriority">  
  </label>  
  <button class="btn btn-primary">Ajouter</button>  
  <br>  
</div>
```

- Directive `v-on` pour gérer les événements utilisateur

```
v-on:click="items.push({id:items.length + 1, label: newItem})"
```

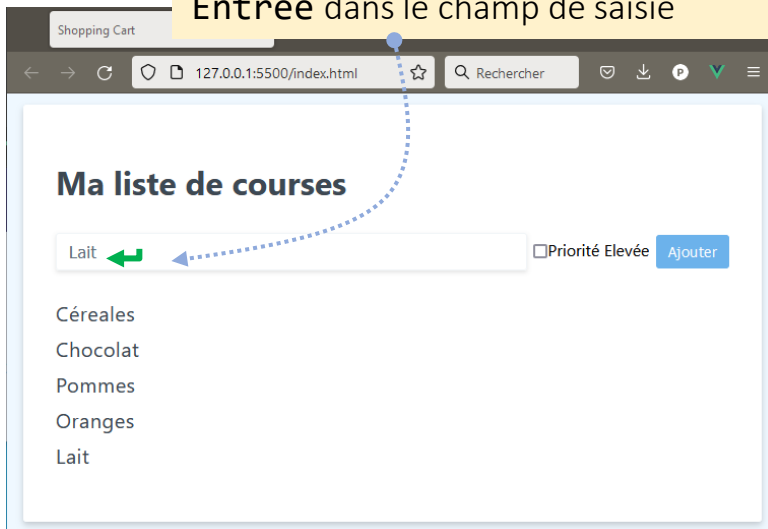
L'évènement

Le code JavaScript à exécuter

# Vue : gérer les évènements utilisateur

- Comme pour les directives v-model , possibilité d'ajouter des 'modifiers' à la directive **v-on** pour altérer le comportement des gestionnaires d'évènements en utilisant une syntaxe déclarative
- forme générale **v-on:evenement.modifieur**
- exemple :

Possibilité d'ajouter le nouvel item à la liste `items` en tapant simplement **Entrée** dans le champ de saisie



évènement : l'utilisateur relâche une touche

modifieur : gestionnaire d'évènement n'est exécuté que quand l'utilisateur relâche la touche **Entrée**

```
v-on:keyup.enter="items.push({id:items.length + 1, label: newItem})"
```

```
<div id="shopping-list">
  <h1>{{header}}</h1>
  <div class="add-item-form">
    <input v-model="newItem" type="text" placeholder="Ajouter un item">
    <label>
      <input type="checkbox" v-model="highPriority">Priorité Elevée
    </label>
    <button
      v-on:click="items.push({id:items.length + 1, label: newItem})"
      class="btn btn-primary">
      Ajouter
    </button>
  </div>
  <ul>
    <li v-for="item in items" :key="item.id">{{item.label}}</li>
  </ul>
</div>
```

# Vue : gérer les évènements utilisateur

- @ raccourci pour la directive v-on:
  - @*evenement.modifieur* ⇔ v-on:*evenement.modifieur*

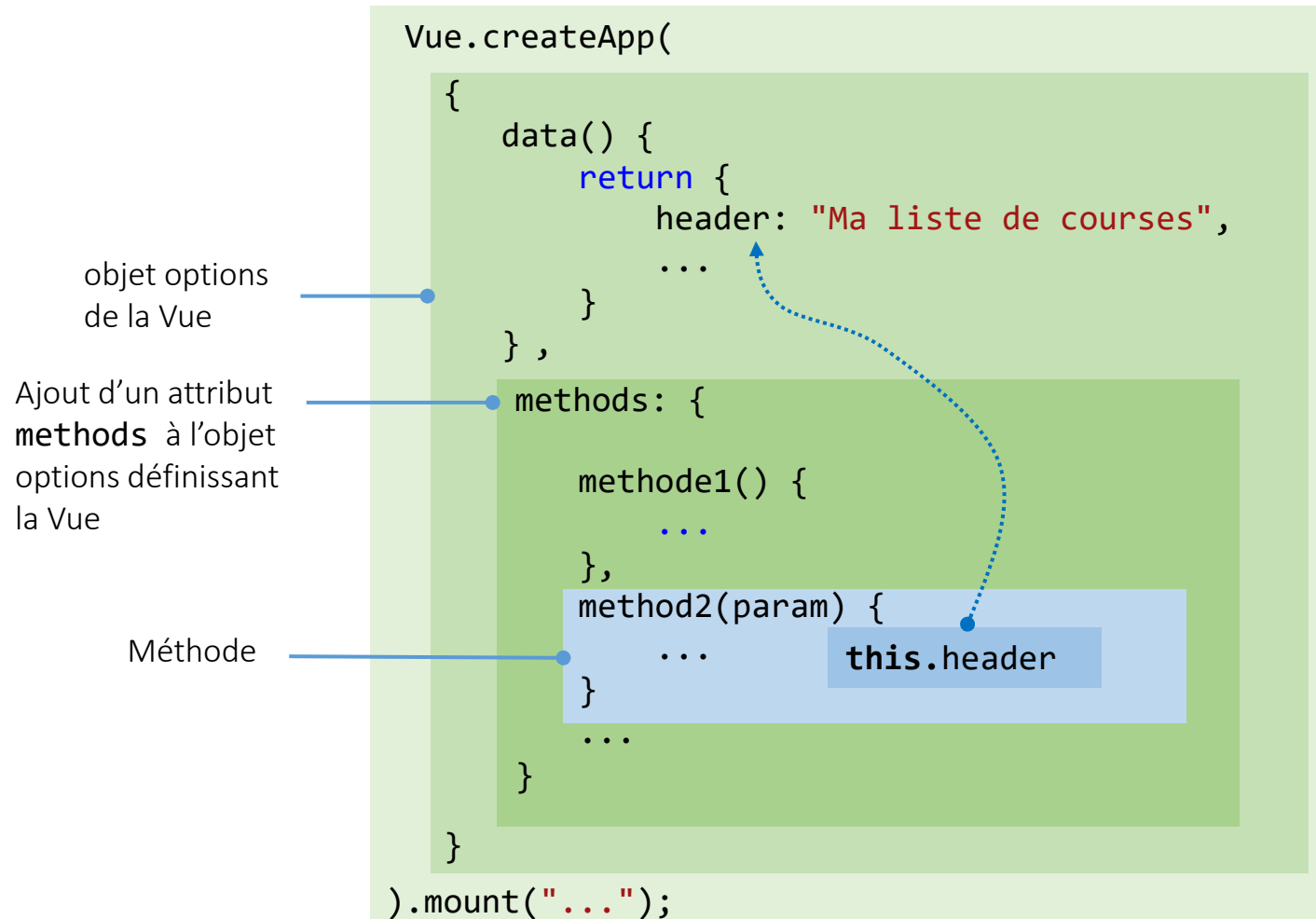
- exemple

```
<div id="shopping-list">
  <h1>{{header}}</h1>
  <div class="add-item-form">
    <input v-model="newItem"
      @keyup.enter="items.push({id:items.length + 1, label: newItem})"
      type="text"
      placeholder="Ajouter un item"
    >
    <label>
      <input type="checkbox" v-model="highPriority">Priorité Elevée
    </label>
    <button
      @keyup.click="items.push({id:items.length + 1, label: newItem})"
      class="btn btn-primary">
      Ajouter
    </button>
  </div>
  <ul>
    <li v-for="item in items" :key="item.id">{{item.label}}</li>
  </ul>
</div>
```

# Vue : Méthodes

- Pas toujours efficace d'exécuter directement du code JavaScript dans un attribut de directive
  - Lisibilité, lorsque le code à exécuter est plus complexe qu'une simple instruction
  - Éventuelle duplication de code

➔ Extraction de la logique dans des fonctions, plus précisément des méthodes, associées à l'instance de **Vue**



il faut passer par la référence **this** pour accéder aux données de l'objet Vue



On ne peut pas utiliser de fonction fléchée car **this** ne serait pas utilisable

```
method3: (param) => {  
  ...  
}
```



# Vue : Méthodes

- Méthode `addItem` pour rajouter `newItem` au tableau `items`

```
<input v-model="newItem"
      type="text"
      @keyup.enter="addItem()"
      placeholder="Ajouter un item"
/>
...
<button @click="addItem" class="btn btn-primary">Ajouter</button>
```

appel de la méthode.  
Les () ne sont pas  
obligatoires

Template  
(Vue)

objet options  
de la Vue

```
Vue.createApp(
  {
    data() {
      return {
        header: "Ma liste de courses",
        ...
      }
    },
    methods: {
      addItem() {
        this.items.push({id: this.items.length + 1, label: this.newItem});
        this.newItem = "";
      }
    }
  }
).mount("#shopping-list");
```

pour effacer  
l'input avec  
l'intitulé de  
l'item

Modèle  
(Vue)

# Rendu Conditionnel : `v-if` et `v-else`

- Parfois il est nécessaire de n'afficher du code HTML que quand certaines conditions sont remplies → directives `v-if` et `v-else`
- Directive `v-if="condition"` si *condition* est vraie l'élément auquel s'applique la directive est visible, sinon il n'apparaît pas (plus précisément, il est retiré du DOM)
- Directive `v-else` ne peut s'appliquer qu'à un élément suivant un élément avec une directive `v-if`, l'élément s'affiche si la directive `v-if` associée est fausse, et inversement ne s'affiche pas si elle est vraie
- exemples
  - afficher un message si la liste est vide

## Ma liste de courses

  Priorité : 

Super ! Tu as fait tous tes achats !

```
<p v-if="items.length === 0">  
  Super ! Tu as fait tous tes achats !  
</p>
```

# Rendu Conditionnel : v-if et v-else

- exemples

- afficher ou masquer le formulaire de saisie d'un nouvel item

Pour Masquer

### Ma liste de courses

Masquer formulaire

 Priorité:

Pain  
Vin (Chignin)

Pour afficher

### Ma liste de courses

Pain  
Vin (Chignin)

Rendu conditionnel des bouton :

- le bouton **Masquer** est affiché si l'édition est possible,
- sinon c'est le bouton **Ajouter un Item** qui est afficher

```
<div class="header">  
  <h1>{{header}}</h1>  
  <button v-if="editing" @click="doEdit(false)" class="btn btn-cancel">  
    Masquer formulaire  
  </button>  
  <button v-else @click="doEdit(true)" class="btn btn-primary">  
    Ajouter un item  
  </button>  
</div>
```

Un clic sur les boutons bascule l'état de la propriété **editing** du modèle

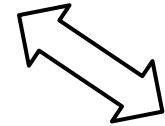
```
Vue.createApp({  
  data() {  
    return {  
      ...,  
      editing: false,  
    };  
  },  
  methods: {  
    addItem() {  
      ...  
    },  
    doEdit(editing) {  
      this.editing = editing;  
      this.newItem = ""  
    }  
  },  
}).mount("#shopping-list");
```

ajout d'une propriété indiquant si le formulaire d'édition doit apparaitre ou non

ajout d'une méthode permettant de modifier la propriété **editing**

# Liaison (binding) d'attributs HTML

- directive **v-bind**: permet de lier n'importe quel attribut HTML aux données du modèle
- syntaxe générale **v-bind:nom-attribut="expression JavaScript"**
- exemples



: tout seul peut être utilisé comme raccourci à **v-bind:nom-attribut="expression JavaScript"**

Ma liste de courses Masquer formulaire

Priorité:  Ajouter

[Dynamic link](#)

Super ! Tu as fait tous tes achats !

https://www.google.com

```
<a v-bind:href="newItem">Dynamic link</a>
```

La valeur du lien hypertexte est liée à la valeur de la propriété `newItem` du modèle

Ma liste de courses Masquer formulaire

Priorité:  Ajouter

Super ! Tu as fait tous tes achats !

Le bouton est désactivé si le champ de saisie est vide

```
<button  
  v-bind:disabled="newItem.length === 0"  
  @click="addItem"  
  class="btn btn-primary">  
  Ajouter  
</button>
```

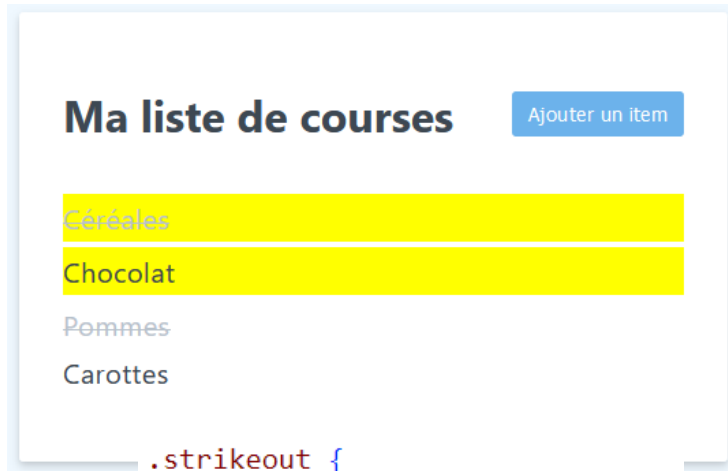
# Liaison d'attributs (binding) de styles CSS

- Quand on utilise une liaison d'attributs HTML (directive `v-bind:` ou `:`) les attributs de classe de style sont un cas particulier car on peut passer en plus des données permettant de contrôler quand certaines classes s'appliquent ou non.

```
:class="{ nom-classe : expression [ , nom-classe : expression ] }"
```

la classe de style est appliquée sur l'expression JS est vraie

- exemple : barrer dans la liste de courses les items qui ont été achetés et surligner ceux qui sont prioritaires.



```
.strikeout {  
  text-decoration: line-through;  
  color: #b8c2cc;  
}  
  
.priority {  
  background-color: yellow;  
  margin-bottom: 4px;  
}
```

Dans le modèle (JS)

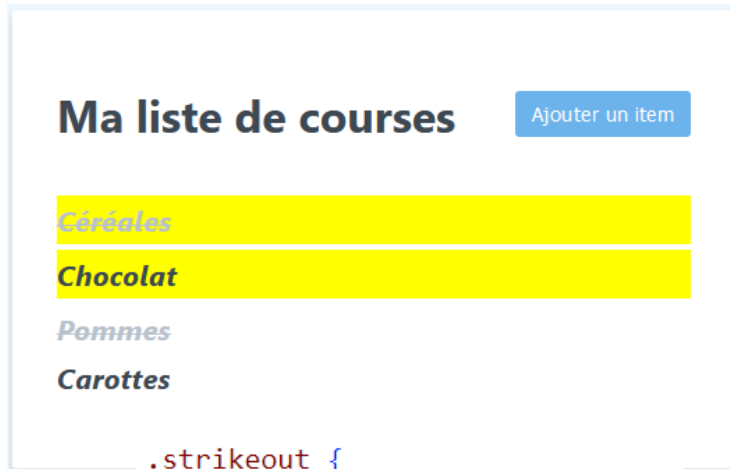
```
items: [  
  { id: 1, label: "Céréales", highPriority: true, purchased:true },  
  { id: 2, label: "Chocolat", highPriority: true, purchased:false },  
  { id: 3, label: "Pommes", highPriority: false, purchased:true },  
  { id: 4, label: "Carottes", highPriority: false, purchased:false },  
],
```

Dans la Vue (template HTML)

```
<ul>  
  <li v-for="item in items" :key="item.id"  
    :class="{strikeout: item.purchased, priority: item.highPriority}">  
    >{{ item.label }}</li>  
</ul>
```

# Liaison d'attributs (binding) de styles CSS

- Pour les classes de style *statiques* (qui s'appliquent de manière inconditionnelle) il suffit de les déclarer à l'aide d'un attribut `class` sans liaison (*binding*)
- exemple : tous les items de la liste des courses sont en gras italique



```
.strikeout {
  text-decoration: line-through;
  color: #b8c2cc;
}

.priority {
  background-color: yellow;
  margin-bottom: 4px;
}

.shopping-item {
  font-style: italic;
  font-weight: bold;
}
```

Dans le modèle (JS)

```
items: [
  { id: 1, label: "Céréales", highPriority: true, purchased:true },
  { id: 2, label: "Chocolat", highPriority: true, purchased:false },
  { id: 3, label: "Pommes", highPriority: false, purchased:true },
  { id: 4, label: "Carottes", highPriority: false, purchased:false },
],
```

Dans la Vue (template HTML)

```
<ul>
  <li v-for="item in items" :key="item.id"
    :class="{strikeout: item.purchased, , priority: item.highPriority}"
    class="shopping-item">
    >{{ item.label }}</li>
</ul>
```

# Liaison d'attributs (binding) de styles CSS

- Une autre syntaxe existe pour définir les style permettant de combiner styles liés et styles statiques

```
:class="[ element-de-style [ , element-de-style ] ]"
```

où *element-de style* peut être

*'nom-classe'* pour appliquer statiquement style *nom-classe*

{ *nom-classe* : *expression* } pour appliquer style *nom-classe* si *expression* est vraie

*expression1* ? *expression2* : *expression3* pour appliquer les styles définis par *expression2* si *expression1* est vraie et les styles définis par *expression2* sinon

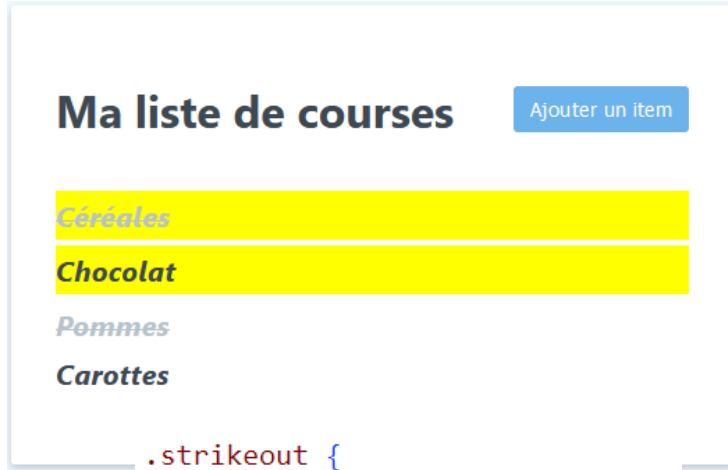
exemple

```
:class="[  
  'maclasse1',           applique statiquement la classe maclasse1  
  { maclasse2: cond1 }, applique maclasse2 si cond1 est vraie  
  { maclasse3: cond2 }, applique maclasse3 si cond2 est vraie  
  cond3 ? 'maclasse4 maclasse5' : 'maclasse6' applique maclasse4 et maclasse5 si cond3 est vraie  
  ]" ou maclasse6 si cond3 est fausse
```



# Liaison d'attributs (binding) de styles CSS

- dans l'exemple avec la liste de courses



```
.strikeout {  
  text-decoration: line-through;  
  color: #b8c2cc;  
}  
  
.priority {  
  background-color: yellow;  
  margin-bottom: 4px;  
}  
  
.shopping-item {  
  font-style: italic;  
  font-weight: bold;  
}
```

Dans le modèle (JS)

```
items: [  
  { id: 1, label: "Céréales", highPriority: true, purchased:true },  
  { id: 2, label: "Chocolat", highPriority: true, purchased:false },  
  { id: 3, label: "Pommes", highPriority: false, purchased:true },  
  { id: 4, label: "Carottes", highPriority: false, purchased:false },  
],
```

Dans la Vue (template HTML)

```
<ul>  
  <li v-for="item in items" :key="item.id"  
    :class="{strikeout: item.purchased, , priority: item.highPriority}"  
    class="shopping-item"  
    >{{ item.label }}</li>  
</ul>
```

```
:class="[  
  {strikeout: item.purchased},  
  {priority: item.highPriority},  
  'shopping-item'  
]"
```

# Propriétés calculées

- fonctionnalité de Vue qui permet de transformer ou d'effectuer des calculs sur les données du modèle, puis de réutiliser facilement le résultat en tant que variable à jour dans notre modèle.
- très utiles pour remplacer les expressions complexes dans les templates



```
<div class="add-item-form" v-if="editing">
  <input
    v-model="newItem"
    type="text"
    @keyup.enter="addItem"
    placeholder="Ajouter un item"
  />
  <p class="counter">{{characterCount}}/200</p>
  ...
</div>
```

```
Vue.createApp({
  data() {
    return {
      header: "Ma liste de courses",
      ...
    };
  },
  methods: {
    ...
  },
  computed: {
    characterCount() {
      return this.newItem.length;
    }
  }
})
.mount("#shopping-list");
```

une propriété calculée **ne doit pas** modifier les données du modèle contrairement aux méthodes qui peuvent le faire