



# Introduction à Vue.js(v3)

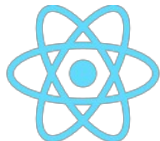
# Qu'est-ce que Vue.js ?

- 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 , 3.5.13 nov. 2024)
  - <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)



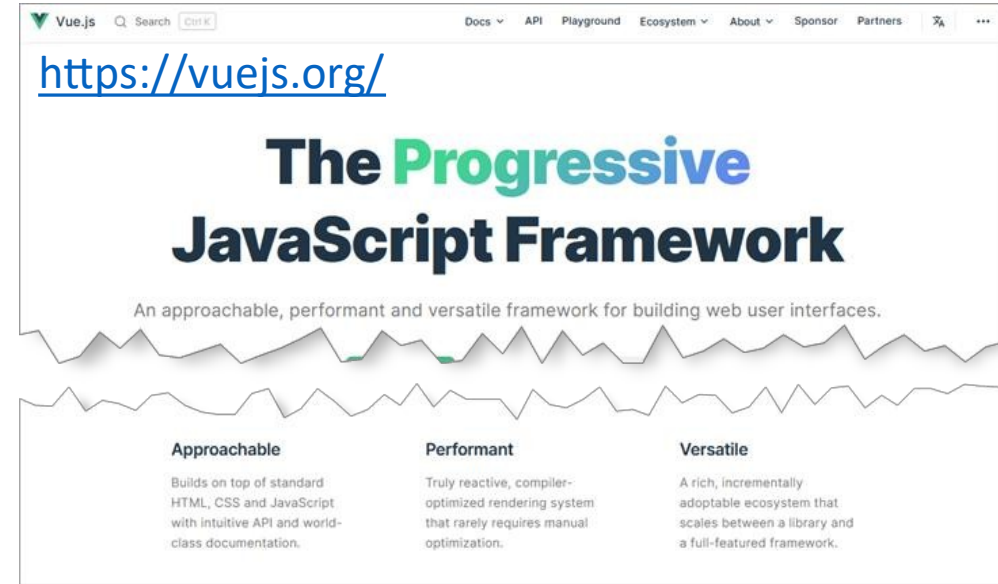
React.js  
(Meta)  
(ex Facebook)



Svelte  
(projet open  
source  
initié par Rich  
Harris 2016)



SolidJS  
(projet open  
source  
initié par Brian



## 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>

# Comparaison des frameworks

Chaque framework a ses forces et faiblesses

le choix dépend souvent des besoins spécifiques du projet et des préférences des développeurs.



## 1. Vue.js :

- **Progressif** : Peut être utilisé pour des parties spécifiques d'une application ou pour des applications complètes[2].
- **Facilité d'apprentissage** : Connu pour sa courbe d'apprentissage douce et sa documentation claire[2].
- **Reactivité** : Utilise un système de réactivité basé sur des observateurs et des proxies[1].

## 2. Angular.js :

- **Framework complet** : Offre une solution complète pour le développement d'applications web, incluant le routage, les formulaires, et bien plus[2].
- **TypeScript** : Utilise TypeScript, ce qui peut être un avantage pour les développeurs qui préfèrent un typage statique[2].
- **Complexité** : Peut être plus complexe à maîtriser en raison de sa richesse fonctionnelle[2].

## 3. React.js :

- **Bibliothèque UI** : Principalement une bibliothèque pour construire des interfaces utilisateur, souvent utilisée avec d'autres bibliothèques pour gérer le routage et l'état[2].
- **JSX** : Utilise JSX, une syntaxe qui combine JavaScript et HTML[2].
- **Hooks** : Introduit les Hooks pour gérer l'état et les effets dans les composants fonctionnels[1].

## 4. Solid.js :

- **Reactivité** : Utilise une approche fine de la réactivité, où les composants ne se re-rendent que lorsque les données qu'ils utilisent changent[1].
- **Performance** : Très performant grâce à une gestion efficace des mises à jour du DOM[1].

[1] [Solid.js, React, and Vue - reactivity systems compared](#)

[2] [Angular vs React vs Vue: Core Differences - BrowserStack](#)

# Tutorial d'introduction à Vue3

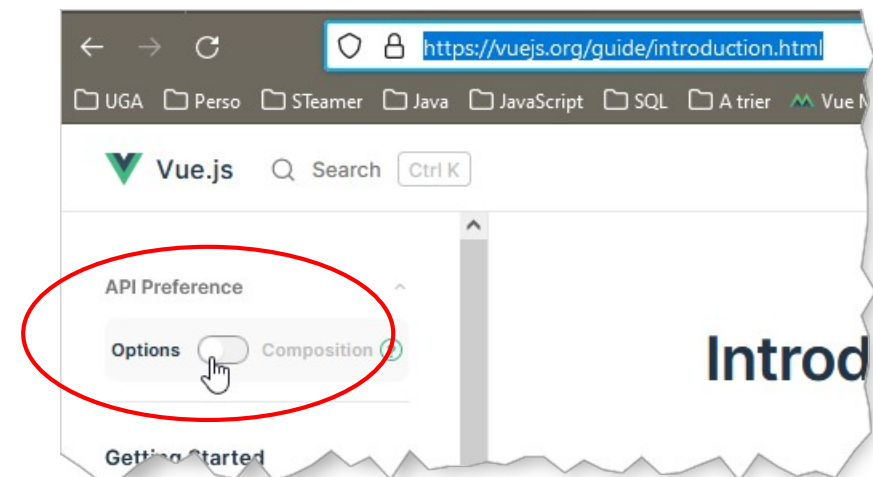
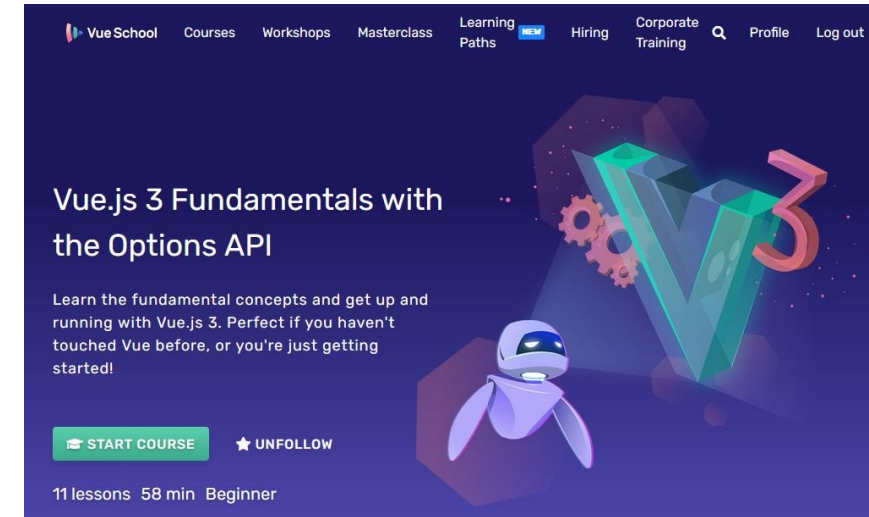
- Tutorial inspiré de "*Vue.js 3 Fundamentals with the Options API*" de vueschool.io

<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 !!!"
        }
      }
    })
    .mount("#hello-message");
  </script>
</body>
</html>
```

{{ ... }} doubles moustaches

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

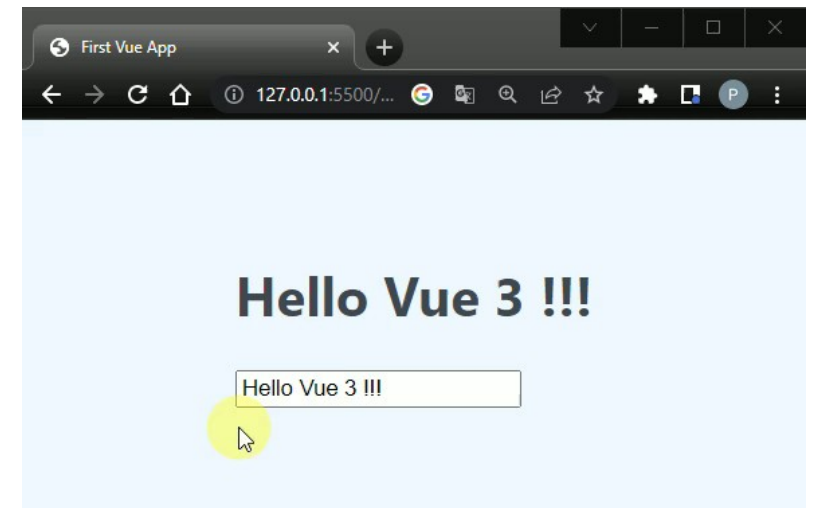
Directive `v-model` qui permet de lier la valeur de l'input à la propriété `message` de l'objet données de l'instance de Vue

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

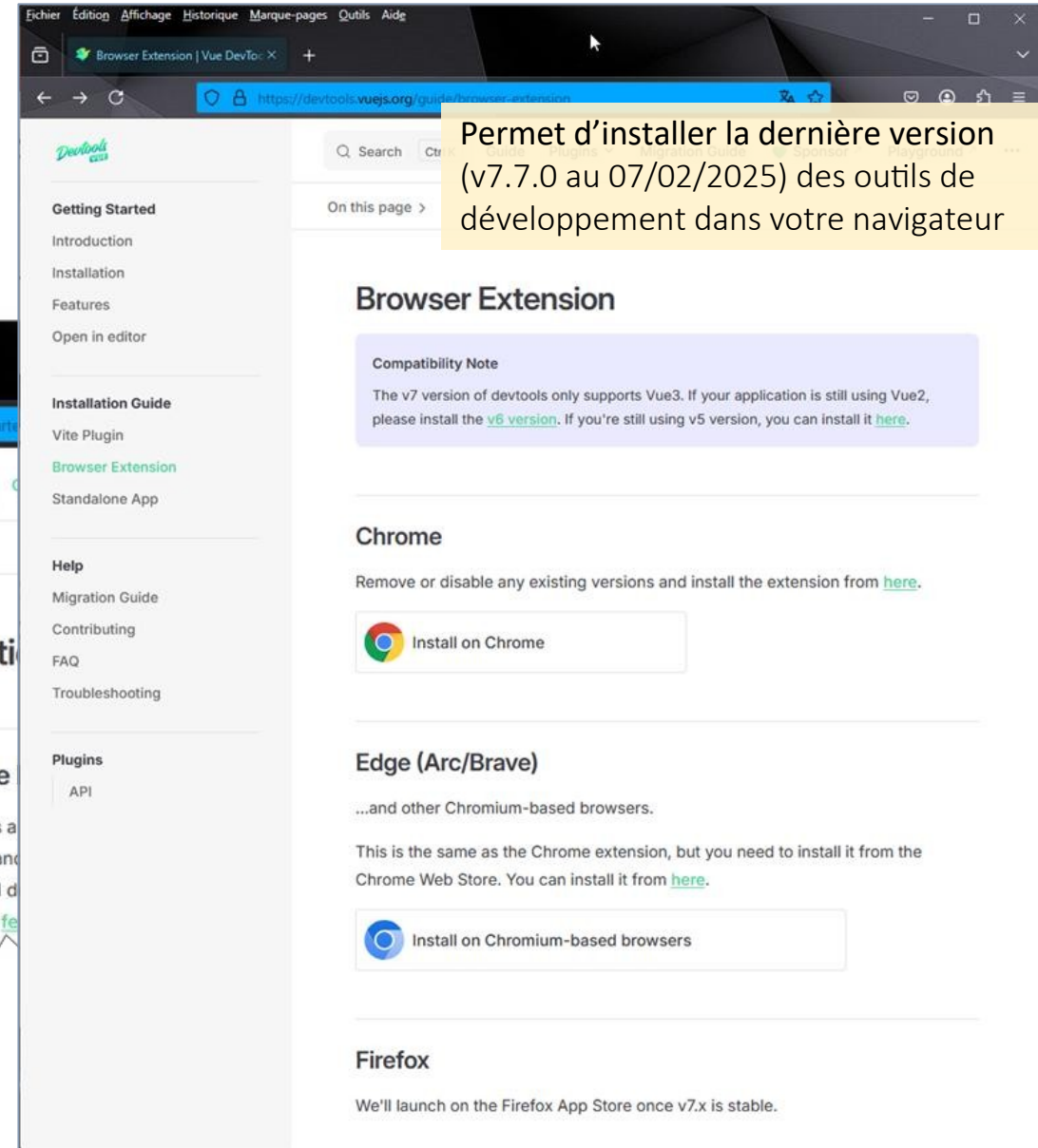
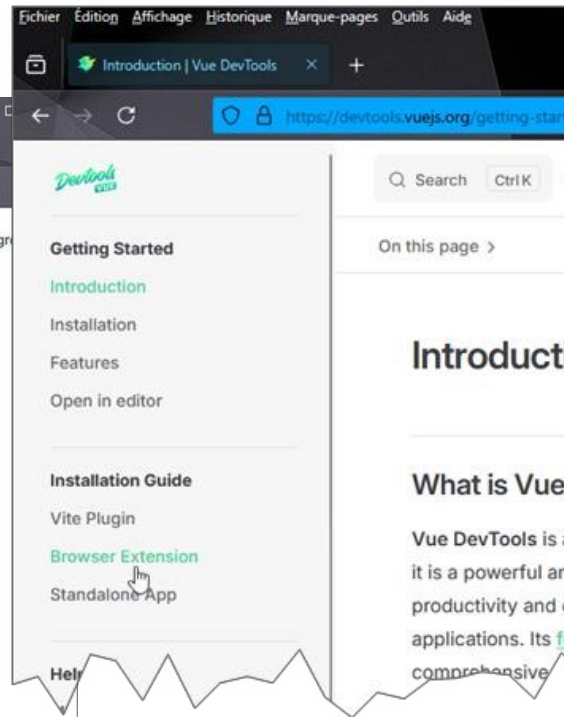
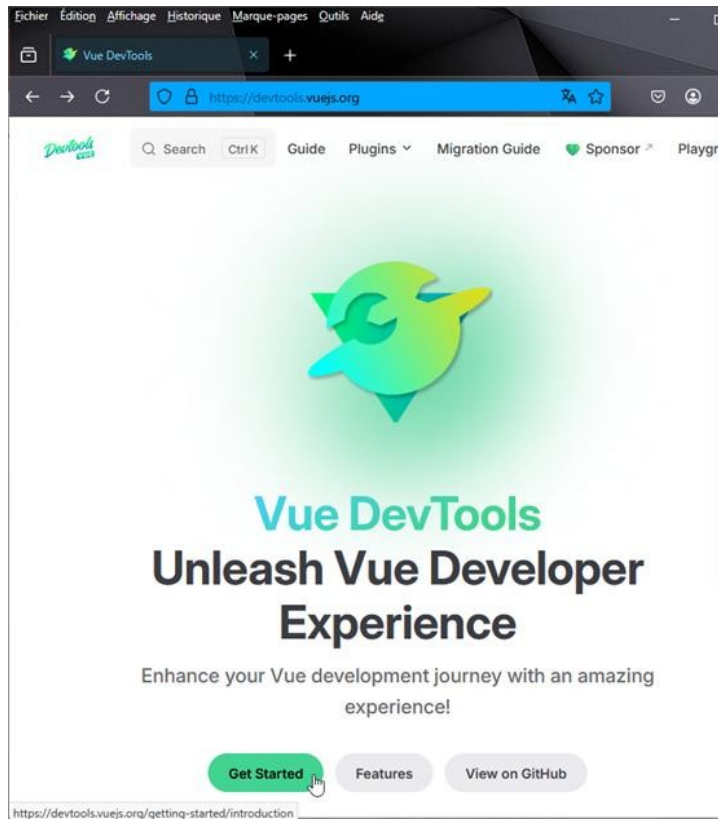
Associe l'objet Vue au div d'identifiant `hello-message`



# Vue Devtools

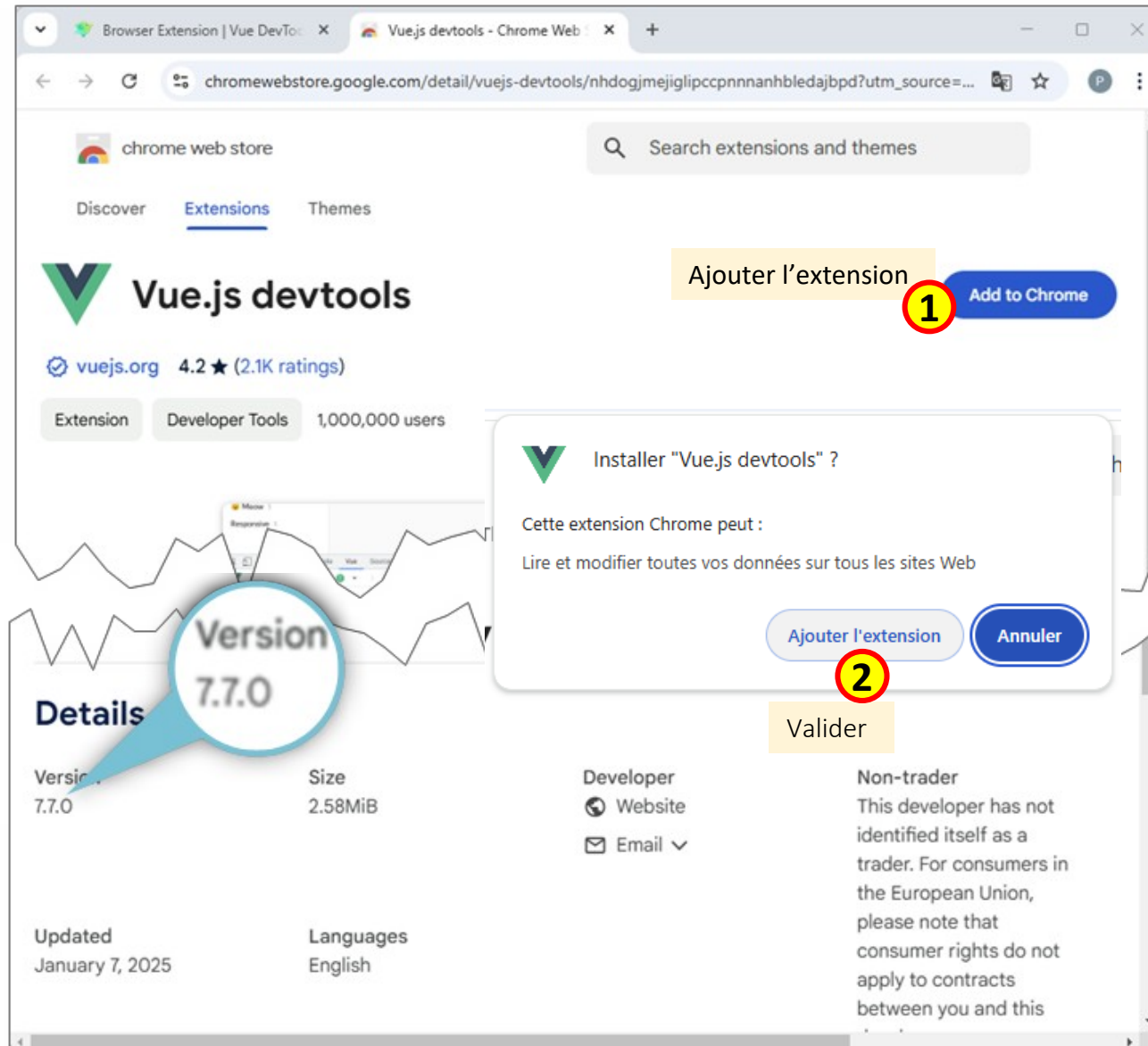
- Vue Devtools extension du browser pour Firefox, Chrome et Edge qui permet d'interagir avec une application Vue
  - Très utile au développement et débogage avec Vue

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

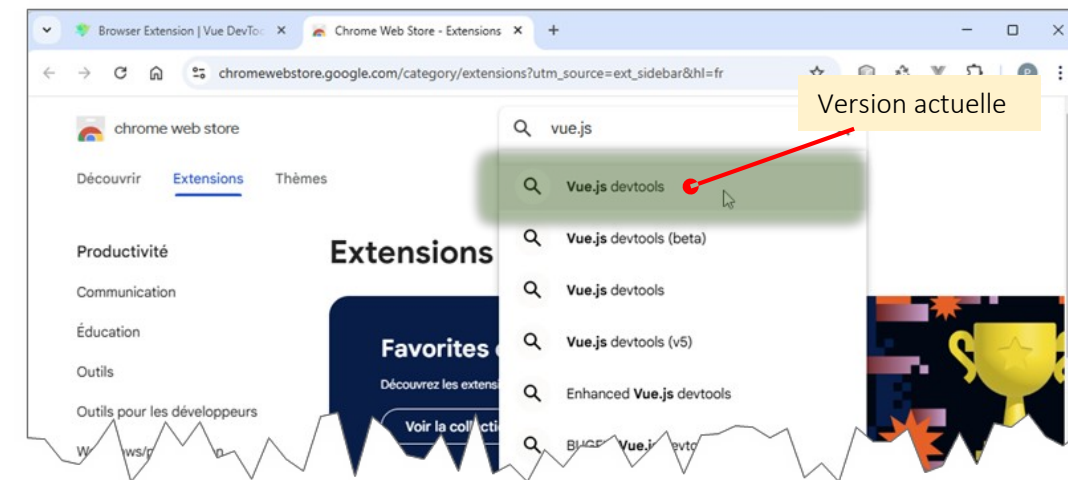




# Vue Devtools sur Chrome



Attention si vous passez par le web store de Chrome (<https://chromewebstore.google.com/>) choisissez la bonne version (7.7.0 au 07/02/2025)



# 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

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

- Mais, uniquement une et une seule expression est acceptée

```
<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

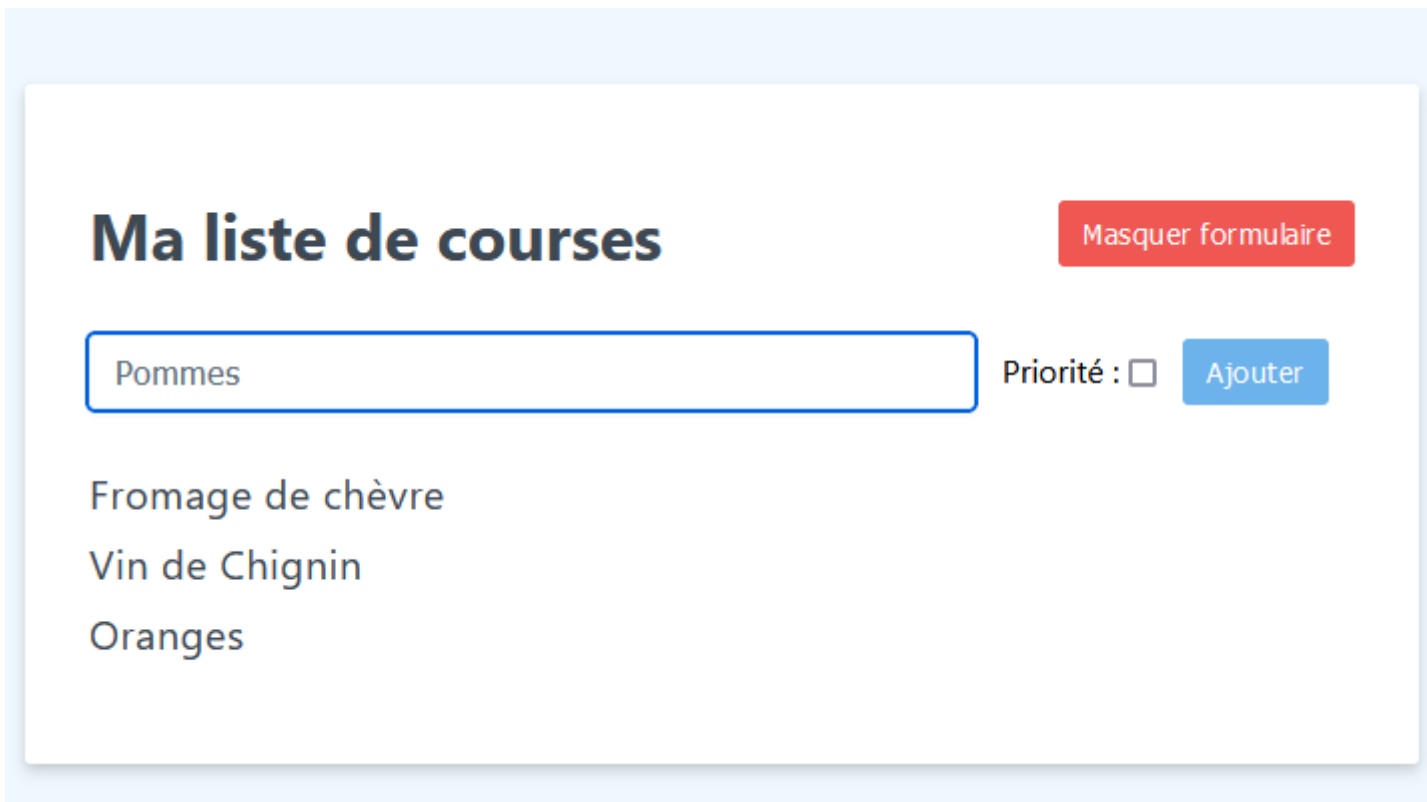
```
[Vue warn]: Template compilation error: Error parsing JavaScript expression: Unexpected token ';'
1 |
2 |   <h1>{{console.log(message); message.toLocaleUpperCase()}}</h1>
  |   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
3 |   <input v-model="message">
4 |
at <App>

Uncaught SyntaxError: missing ) after argument list
    at new Function (<anonymous>)
    at compileFunction (vue@3:15920:23)
```



# 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

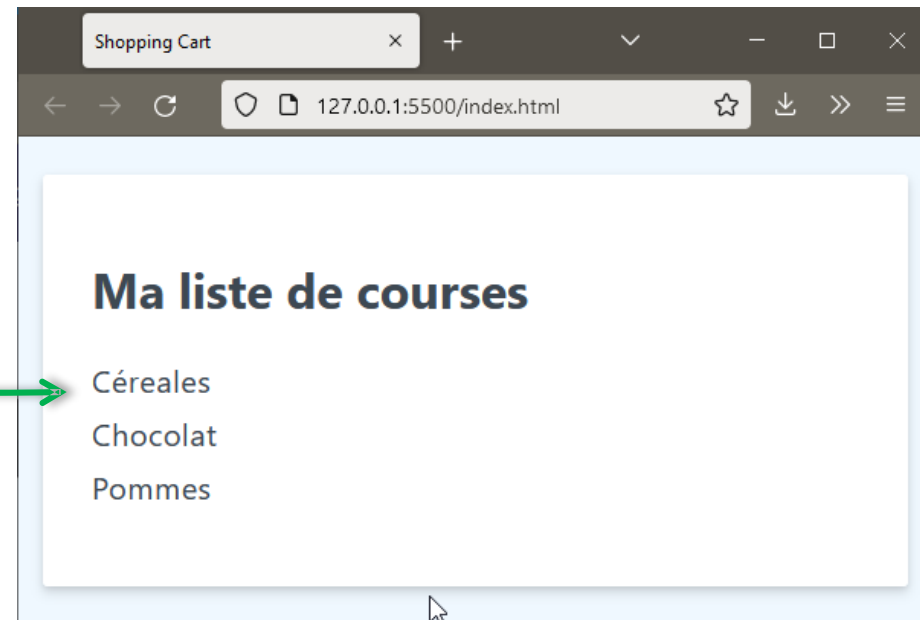
The image shows a web application interface for managing a shopping list. It features a title 'Ma liste de courses' in bold black text. To the right of the title is a red button labeled 'Masquer formulaire'. Below the title is a text input field containing the word 'Pommes'. To the right of the input field is a label 'Priorité : ' followed by an unchecked checkbox and a blue button labeled 'Ajouter'. Below the input field, there is a list of items: '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>
```



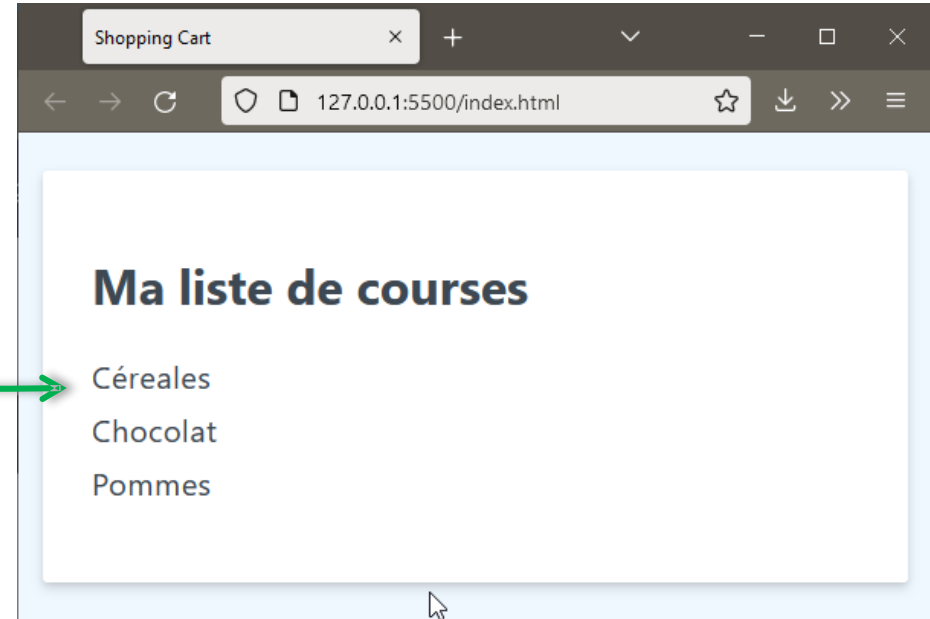
```
ul {
  list-style: none;
  padding: 0;
}
```

# 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

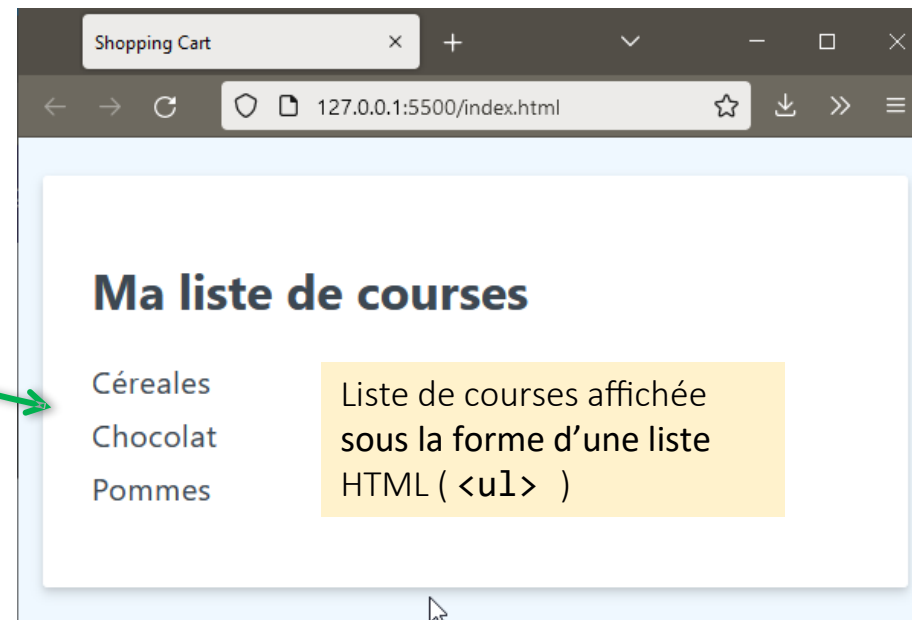
```
<head>
...
</head>

<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>

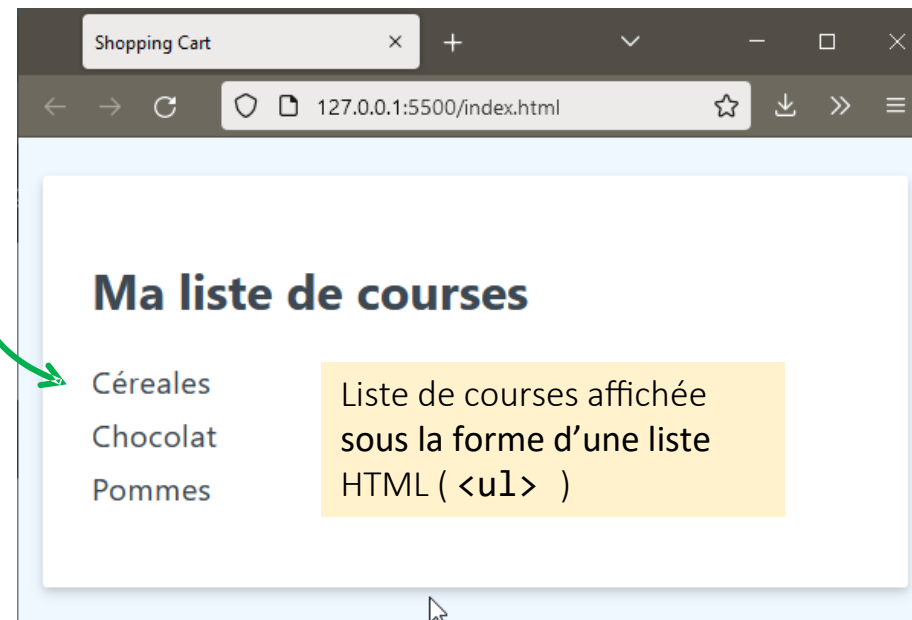
<body>
  <div id="shopping-list">
    <h1>{{header}}</h1>
    <ul>
      <li v-for="item in items" :key="item.id">{{item.label}}</li>
    </ul>
  </div>
  <script src="https://unpkg.com/vue@3"></script>
  <script>
    const shoppingListApp = Vue.createApp({
      data() {
        return {
          header: "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);
```

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

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

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

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

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



# 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 Strasbourg = {  
  nom : "Strasbourg",  
  codePostal : "67000",  
  coordonnees : {  
    lat : 48.5833,  
    Lon : 7.75  
  },  
  temperatures: [5, 12.5]  
};
```

*Diagram illustrating variable destructuring:*

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

Arrows indicate the mapping from the original object properties to the destructured variables.

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"  
} = Strasbourg;
```

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);
```

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

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();**

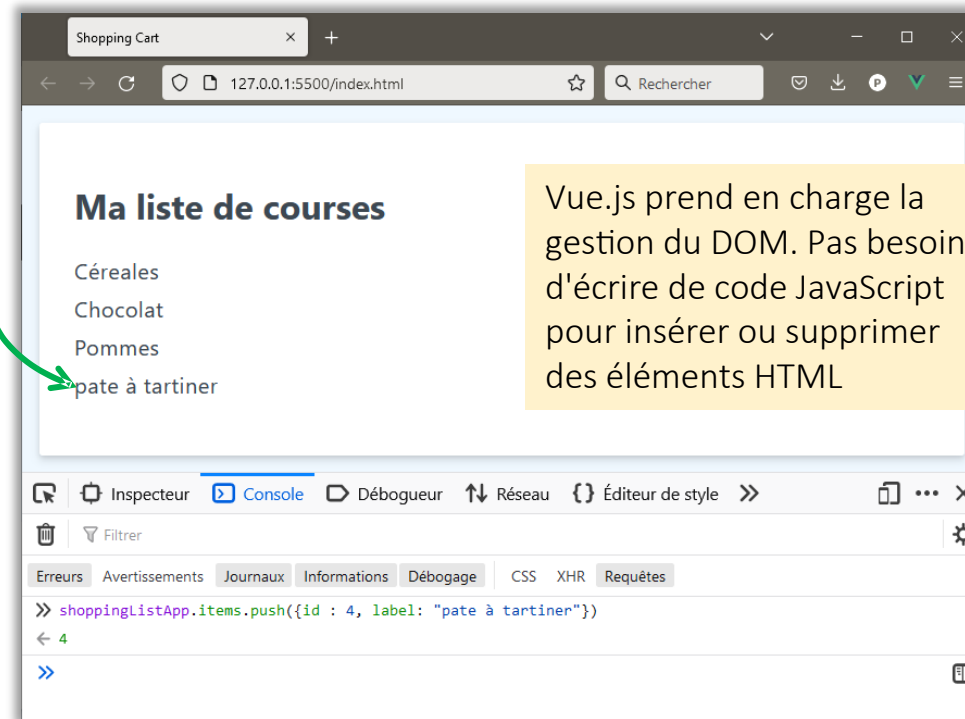
# Vue templating syntax

- 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.

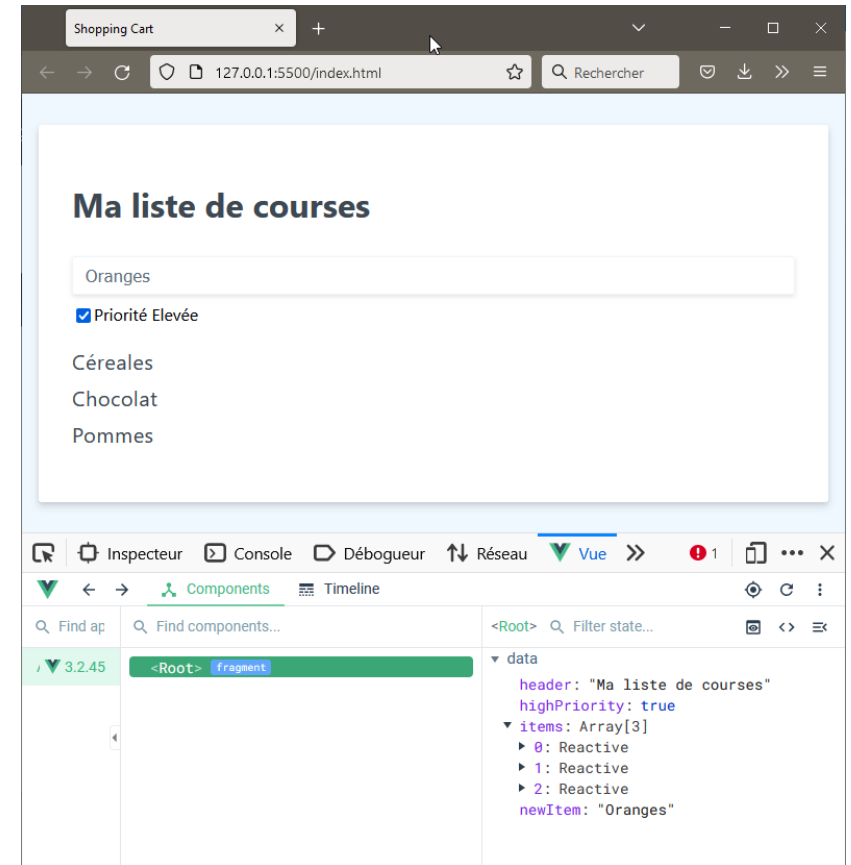


Rajoute un item au tableau items

```
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

  
Fromage  
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

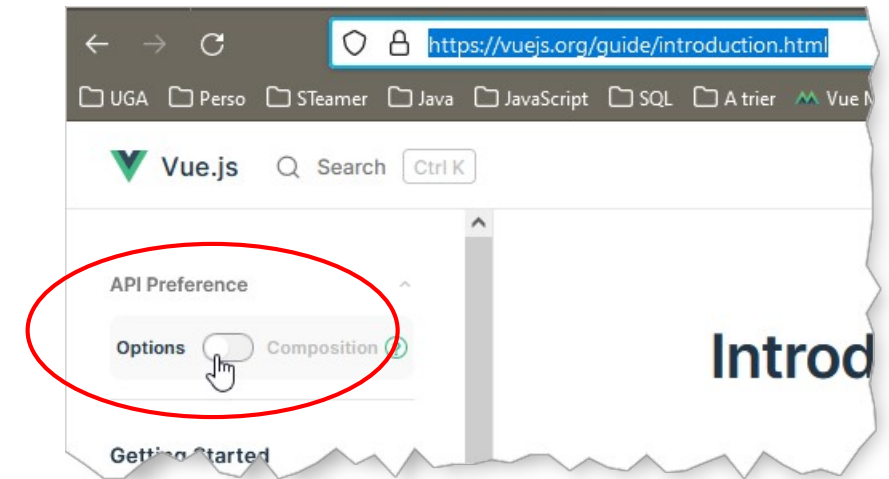
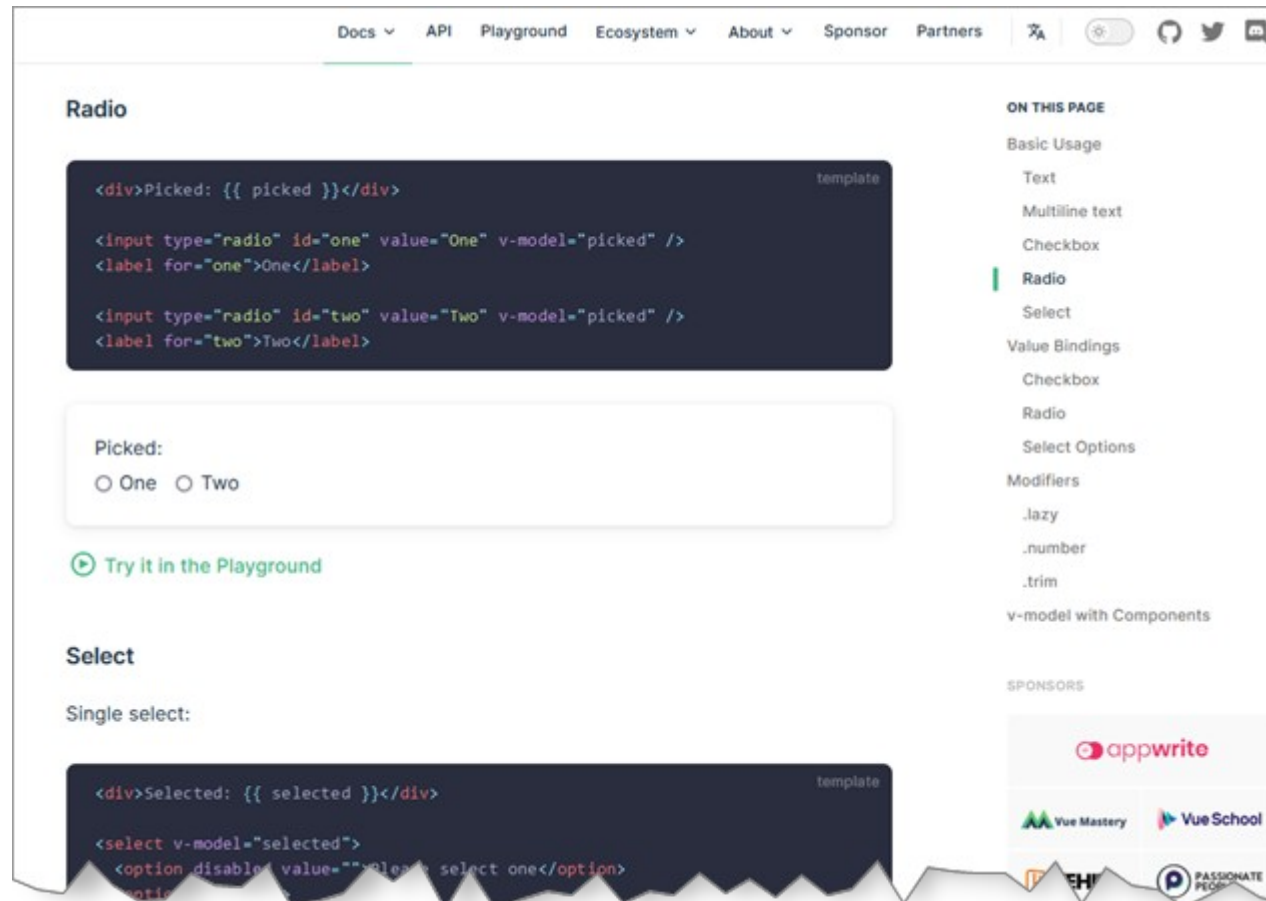
  
Céréales  
Chocolat  
Pommes

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>



Attention : dans le guide Vue3  
activer la préférence  
Options API

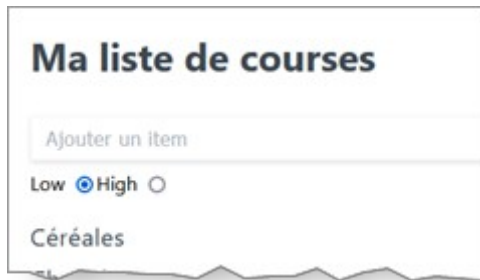


# Vue : saisie de valeurs à l'aide d'inputs

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

newItemPriority: "low",

- avec des Radios Boutons



Ma liste de courses

Ajouter un item

Low ☒ High ☐

Céréales

```
<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



Ma liste de courses

Ajouter un item

Low ▼

Low  
High

Chocolat

Pommes

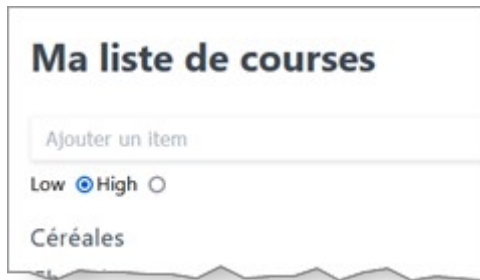
```
<select v-model="newItemPriority">  
  <option value="low">Low</option>  
  <option value="high">High</option>  
</select>
```

# 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



The screenshot shows a web form titled "Ma liste de courses". It has a text input field labeled "Ajouter un item". Below the input field, there are two radio buttons: "Low" (which is selected, indicated by a blue dot) and "High". Below the radio buttons, the text "Céréales" is visible, suggesting it's the item being added.

```
<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



The screenshot shows a web form titled "Ma liste de courses". It has a text input field labeled "Ajouter un item". Below the input field, there is a dropdown menu with "Low" selected. The dropdown menu is open, showing "Low" and "High" as options. Below the dropdown menu, the text "Chocolat" and "Pommes" are visible, suggesting they are items in the list.

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

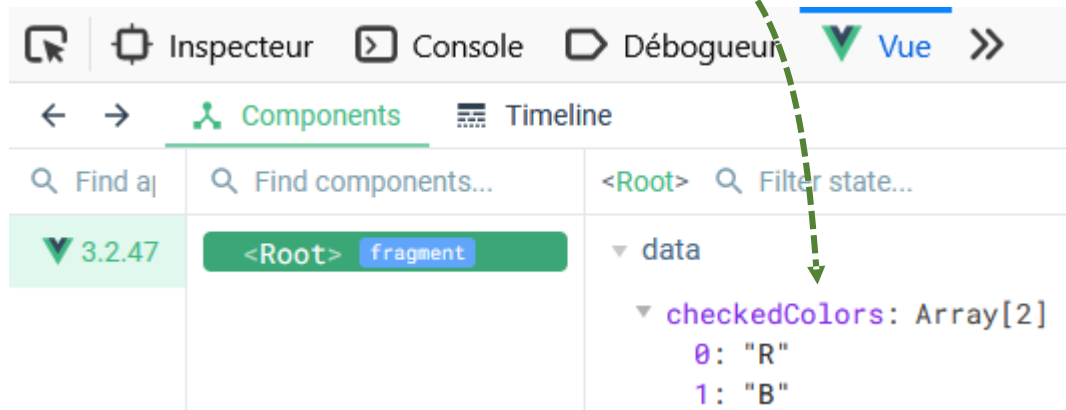
# Vue : saisie de valeurs à l'aide d'inputs

- 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})"
```

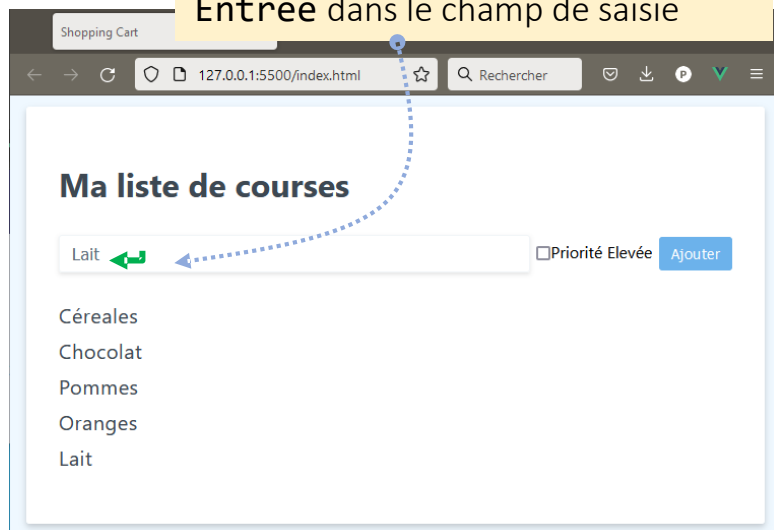
L'évènement

Le code JavaScript à exécuter

# Vue : gérer les événements 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:événement.modifieur**
- exemple :

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



événement : l'utilisateur relâche une touche

modifieur : gestionnaire d'évènement n'est exécuté que quand l'utilisateur relâche la toucheEntrée

**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 événements utilisateur

- @ raccourci pour la directive **v-on**:
  - **@evenement.modifieur**  $\Leftrightarrow$  **v-on:evenement.modifieur**

- exemple

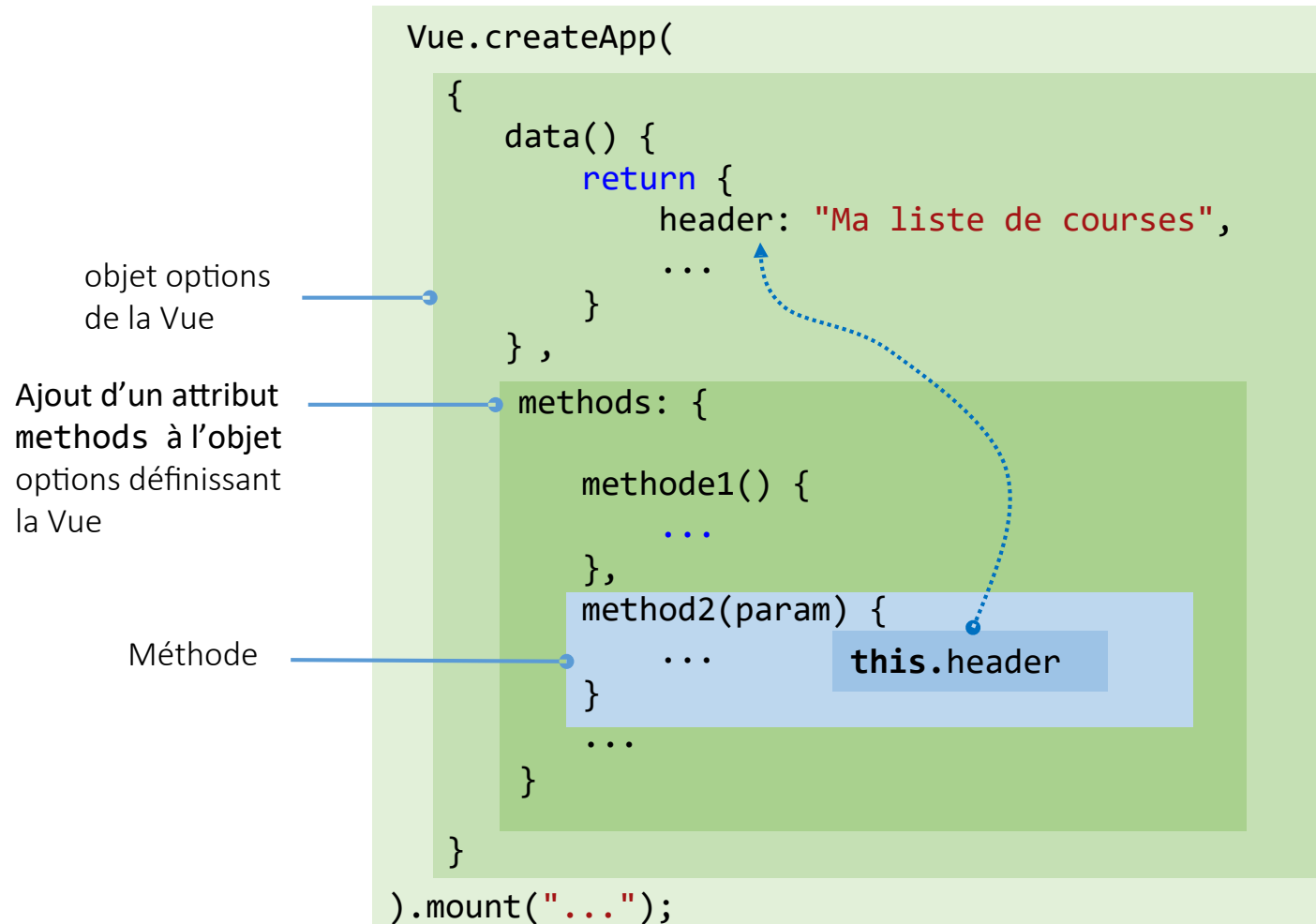
```
<div id="shopping-list">
  <h1>{{header}}</h1>
  <div class="add-item-form">
    <input v-model="newItem"
      @keyup.enter="items.push({id:items.length + 1, label: newItem})"
      type="text"
      placeholder="Ajouter un item"
    >
    <label>
      <input type="checkbox" v-model="highPriority">Priorité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

```
methods (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 : ☐ Ajouter

Super ! Tu as fait tous tes achats !

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

D'autres directives pour le rendu conditionnel existent (**v-else-if**, **v-show** ...), pour en savoir plus

- <https://vuejs.org/guide/essentials/conditional.html>
- <https://vueschool.io/lessons/conditional-rendering-in-vue-3?friend=vuejs>

# 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**

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

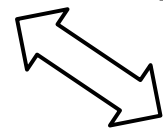
```
Vue.createApp({
  data() {
    return {
      ...,
      editing: false,
    };
  },
  methods: {
    addItem() {
      ...
    },
    doEdit(editing) {
      this.editing = editing;
      this.newItem = ""
    }
  },
}).mount("#shopping-list");
```

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"**

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

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

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>
```

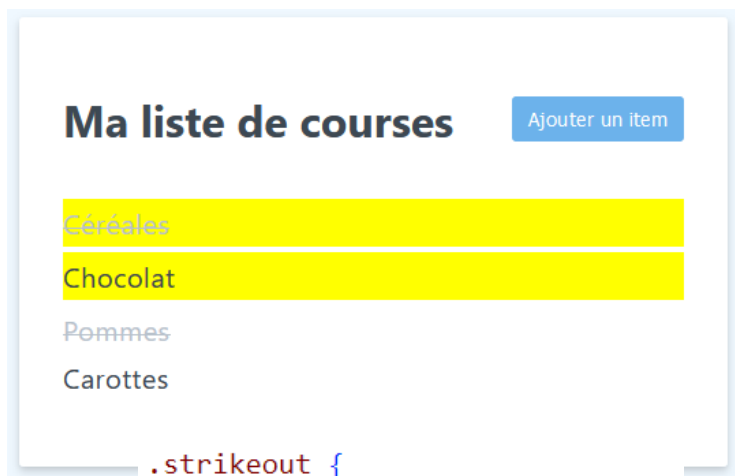
# Liaison d'attributs (binding) de styles CSS

- 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 si 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 styles 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 *expression3* 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



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

# Instances multiples d'une application

- Possibilité de faire cohabiter plusieurs applications Vue (créés par `Vue.createApp()`) sur une même page

```
const vueApp1 = createApp({  
  /* ... */  
});  
vueApp1.mount("#container-1");  
  
const vueApp2 = createApp({  
  /* ... */  
});  
vueApp2.mount("#container-2");
```

"si vous utilisez Vue pour améliorer l'HTML rendu par le serveur et que vous n'avez besoin de Vue que pour contrôler des parties spécifiques d'une grande page, évitez de monter une seule instance d'application Vue sur la page entière. Au lieu de cela, créez plusieurs petites instances d'application et montez-les sur les éléments dont elles sont responsables."

<https://vuejs.org/guide/essentials/application.html#multiple-application-instances>

# Syntaxe de template

- basée sur HTML<sup>1</sup>
- permet de lier de manière déclarative le DOM rendu aux données de l'instance du composant sous-jacent.
- en interne,
  - Vue compile les templates en code JavaScript hautement optimisé.
  - Combiné avec le système de réactivité, Vue est capable de déterminer intelligemment le nombre minimal de composants à restituer et d'appliquer la quantité minimale de manipulations DOM lorsque l'état de l'application change.

```
<table class="table">
  <thead>
    ...
  </thead>
  <tbody>
    <tr v-for="lieu in lieuxTrouvés"
      @click="sélectionnerLieu(lieu)"
      :class="{ 'table-primary' : lieu.selected }" >
      <td> {{lieu.name}} </td>
      <td> {{lieu.country}} </td>
      <td> {{lieu.lat}} </td>
      <td> {{lieu.lon}} </td>
    </tr>
  </tbody>
</table>
```

<sup>1</sup> Tous les templates Vue sont du HTML syntaxiquement valide pouvant être analysé par des navigateurs et des analyseurs HTML conformes aux spécifications

# Syntaxe de template

- Interpolation de texte
  - forme élémentaire de liaison de données en utilisant la syntaxe Moustaches (doubles accolades)
    - remplacée par valeur de la propriété **prop** de l'instance du composant correspondant
    - mise à jour chaque fois que la propriété **prop** change

<p>texte interpolé : {{ prop }}</p>



Texte interpolé : Hello Vue 3 !!!

- n'affiche rien si la valeur de la propriété est **undefined** ou **null**

<p>prop1: {{ prop1 }}</p>  
<p>prop1: {{ prop2 }}</p>  
<p>prop1: {{ prop3 }}</p>



prop1 :  
prop2 :  
prop3 :

```
const vueApp = Vue.createApp({  
  data() {  
    return {  
      prop: "Hello Vue 3 !!!",  
      prop1 : undefined,  
      prop2 : null  
    }  
  }  
});  
vueApp.mount("#hello-message");
```

# Syntaxe de templates

```
const vueApp = Vue.createApp({
  data() {
    return {
      ...,
      prop4 : ["chaîne 1", "chaîne 2", "chaîne 3"],
      prop5 : { nom: "MOOSE", prenom : "Joe", age : 25},
      texteHTML : '<strong>Texte renforcé</strong>'
    }
  }
});
vueApp.mount("#hello-message");
```

```
<p>prop1 = {{ prop4 }}</p>
<p>prop1 = {{ prop5 }}</p>
```

```
<p>Interpolation de texte : {{texteHTML}}</p>
```

```
<p>
  Avec directive v-html : <span v-html="texteHTML"></span>
</p>
```

- interpolation de texte suite

- si la valeur de la propriété n'est pas un type primitif (**number**, **string**, **boolean**) la valeur littérale de l'objet est affichée

prop4 = [ "chaîne 1", "chaîne 2", "chaîne 3" ]  
prop5 = { "nom": "MOOSE", "prenom": "Joe", "age": 25 }

- si la valeur de la propriété est une chaîne HTML elle est interprétée comme texte brut

Interpolation de texte : <strong>Texte renforcé</strong>

- la directive **v-html** permet d'interpréter la propriété comme du HTML

Avec directive v-html : Texte renforcé

la propriété **innerHTML** de l'élément HTML est affectée avec la valeur de la directive **v-html**



# Syntaxe de Templates

- Directives

- attributs de balises HTML spéciaux fournis par Vue, préfixés par **v-**
- appliquent un comportement réactif spécial au DOM rendu pour l'élément HTML auquel s'applique la directive
  - exemples :
    - **v-if** supprime/insère un élément selon la valeur booléenne de l'expression liée
    - **v-for** rend une liste d'éléments en itérant sur une collection de données
    - ...

- Directive **v-bind** pour la liaison d'attributs

`{{ }}` ne peuvent pas être utilisées dans attributs HTML



`<a v-bind:href="{{url}}">Lien</a>`

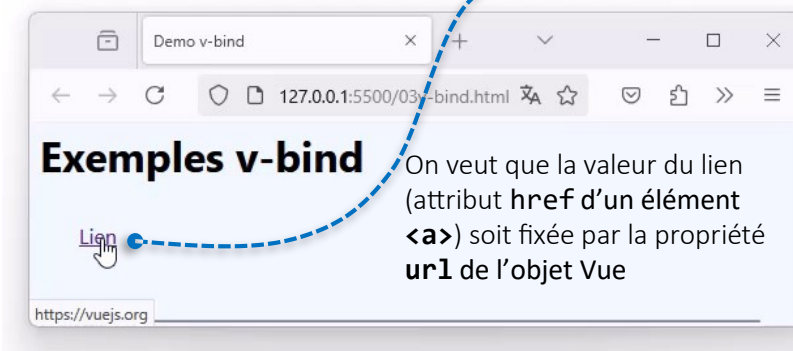
à la place il faut utiliser directive **v-bind** ou son raccourci :



`<a v-bind:href="url">Lien</a>`

`<a :href="url">Lien</a>`

```
const bindVue = Vue.createApp({
  data() {
    return {
      url: "https://vuejs.org"
    }
  }
});
bindVue.mount("#bind");
```



# Syntaxe de templates

- Liaisons d'attributs (suite...)

- raccourcis

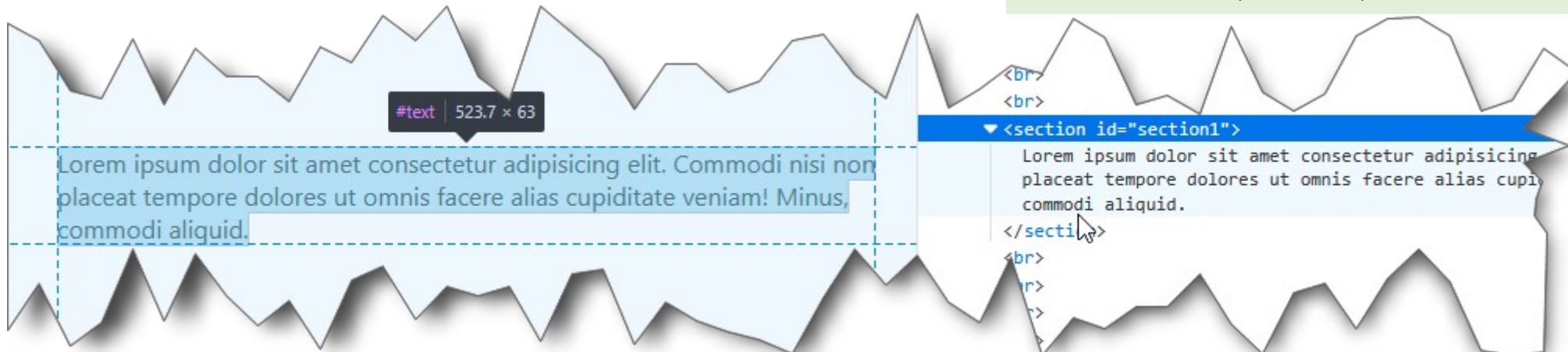
- `:` au lieu de **v-bind:**
- omission de la valeur de l'attribut si la propriété a le même nom que l'attribut lié

Vue3.4 +

```
<section :id>
  Lorem ipsum dolor sit amet consectetur
  adipisicing elit.
</section>
```

équivalent à **v-bind:id="id"** ou **v-bind:id**

```
const bindVue = Vue.createApp({
  data() {
    return {
      id: "section01",
      attrImage: {
        src: "images/logoUGA.png",
        width: "100"
      }
    }
  }
});
bindVue.mount("#bind");
```



# Syntaxe de template

- Utilisation d'expressions JavaScript
  - possibilité d'utiliser des expressions JavaScript dans les liaisons entre le template HTML et les données du composant
    - dans une interpolation de texte `{{ }}`
    - dans la valeur d'attribut de toutes les directives Vue `v-`

```
{{ number + 1 }} → 11
```

```
{{ ok ? 'YES' : 'NO' }} → NO
```

```
{{ message.split('').reverse().join('') }}
```

```
→ ! 3euV olleH
```

```
<div :id="`list-${id}`">
  Lorem ...
</div>
```

```
→ <div :id="list-courses">
  Lorem ...
</div>
```

```
const bindExprApp = Vue.createApp({
  data() {
    return {
      number: 10,
      ok: false,
      message: "Hello Vue3 !",
      id: "courses",
    };
  },
});
const bindExpr = bindVueApp.mount("#app");
```



Les expressions de template sont en bac à sable (*sandboxed*). On n'a accès qu'aux données et méthodes définies sur le composant et à une [liste restreinte de variables globales](#)<sup>1</sup>.

<sup>1</sup> possibilité de définir explicitement des variables globales supplémentaires en les ajoutant à [app.config.globalProperties](#).

# Syntaxe de template

- Arguments des directives
  - Certaines directives peuvent prendre un "argument", distingué par un double-point ( : ) après le nom de la directive

```
<a v-bind:href="url">Lien</a>
```

Nom de la  
directive

Argument de  
la directive

- Possible d'utiliser des arguments dynamiques définis par une expression

```
<a v-bind:[expression]="url"> ... </a>
```

La valeur de l'expression entre [ ] définit la valeur de l'attribut lié

```
<div id="app">
  <div>
    <label>width : </label>
    <input type="radio" v-model="dimension" value="width">
    <label>height : </label>
    <input type="radio" v-model="dimension" value="height">
  </div>
  <div>
    <input type="number" v-model="size"
      placeholder="valeur de width ou height">
  </div>
  
</div>
```

```
const bindArgsApp = Vue.createApp({
  data() {
    return {
      dimension: "width",
      size: 200
    };
  },
});
```

# Gérer les erreurs dans une application VueApp

- on peut utiliser un gestionnaire d'erreurs défini de manière globale au niveau de l'application (*app-level error handler*) qui capture les erreurs de tous les composants descendants.
- celui-ci est défini via l'objet **config**<sup>1</sup> exposé par l'objet application

```
const vueApp = Vue.createApp({ ... } );
```

```
vueApp.config.errorHandler = (err, instance, info) => { ... } ;
```

```
const vm = vueApp.mount("#app");
```

**vueApp** : l'instance de l'application.

**vm** : l'instance du  
composant racine

<sup>1</sup> Pour plus d'informations voir la documentation de l'API : <https://vuejs.org/api/application.html>