**Contexte** : PPE 2ème année – étude et utilisation du Framework Javascript Vue.js

Principes de Vue.js. Instance Vue. Propriétés, directives

**Objectif du PPE**

Durant cette séance, nous allons introduire le Framework Vue, présenter ses interactions avec HTML, CSS, JavaScript.

**Installation :**

Nous allons utiliser PHPStorm, Chrome et l’extension Chrome pour le débogage Vue.js.

**Partie n°0** : Installer l’extension Vue pour Chrome

<https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=fr>

**Partie n°1** : Installation et introduction aux instances Vue

Nous allons partir d’un template extrêmement basique HTML, appelé **000\_Template.html** et fourni.

Etape 1 : Utilisation de Vue

1. Copiez le fichier fourni dans le répertoire créé pour initialiser un dépôt Git

(rappel : création d’un répertoire sous la racine c:\laragon\www et git init)

1. Tout d’abord ouvrez PHPStorm puis ouvrez le répertoire créé
2. Ajoutez le Plugin Vue.js

Ce plugin fournit de la complétion de code et de l’aide lorsque vous écrivez du code.

File 🡪 Settings 🡪 Plugins 🡪 Cherchez et installer le plugin Vue.js

1. Ouvrez le template fourni puis, dans le BODY, création d’une balise Script, et référencement du Framework Vue.js.

Lire la documentation suivante pour savoir comment référencer le script qui contient le Framework Vue.js : https://v3.vuejs.org/guide/introduction.html#handling-user-input

Utilisez le CDN (dépôt) pour référencer dans votre balise SCRIPT le framework Vue.

Etape 2: Création d’une nouvelle instance Vue.js

Ouvrir une balise HTML Script et dedans, créez une nouvelle instance Vue placée dans une variable app :

***L’objet data*** : Documentation : « Quand une instance de Vue est créée, cela ajoute toutes les propriétés trouvées dans son objet data au **système réactif** de Vue. Quand une valeur de ces propriétés change, la vue va « réagir », se mettant à jour pour concorder avec les nouvelles valeurs. »

Ici on crée une ***propriété*** message qu’on veut utiliser pour passer un message.

Etape 3 : ajout dans le Div d’un input

A l’intérieur de la balise DIV sur laquelle va s’accrocher notre instance Vue, ajoutez un INPUT :



On a un élément HTML Input de type Text, qu’on relie à la ***propriété message*** de data (réactive) avec la ***directive*** Vue v-model :

Documentation : « Vous pouvez utiliser la directive v-model pour créer une liaison de données bidirectionnelle sur les champs de formulaire (input, select ou textarea). Elle choisira automatiquement la bonne manière de mettre à jour l’élément en fonction du type de champ. Bien qu’un peu magique, v-model est essentiellement du sucre syntaxique pour mettre à jour les données lors des évènements de saisie utilisateur sur les champs, ainsi que quelques traitements spéciaux pour certains cas particuliers. »

Ensuite on affiche la ***propriété message*** de data, via la syntaxe moustache

Cf : <https://mustache.github.io/>

Logic-less template veut dire : « there are no if statements, else clauses, or for loops. Instead there are only tags. Some tags are replaced with a value, some nothing, and others a series of value. »

**Checkpoint n° 1** :

1. Vérifiez que lorsque vous modifiez le champ input, le message se modifie en même temps.
2. Dans la console Chrome, ouvrez le panneau Vue et modifiez directement le message :

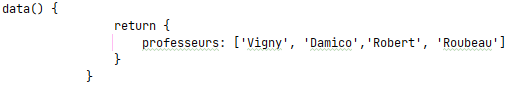
Pour ouvrir la console : F12 puis allez chercher le panneau Vue.

log.$date.message =’message modifié’

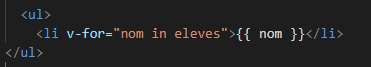
log est le nom donné à mon instance vue.

Etape 4 : manipulation de tableau (listes) d’éléments

Eventuellement, vous pouvez enregistrer votre programme sous un nouveau nom pour tracer par la suite les modifications.

Modifiez le data : 

Remplacez la balise Input et le paragraphe par une liste d’éléments :



Explications :

On retrouve les balises HTML de liste (UL et LI) .

Voici la description de la directive v-for. Documentation : «Nous pouvons utiliser la directive v-for pour faire le rendu d’une liste d’éléments en nous basant sur un tableau. La directive v-for utilise une syntaxe spécifique de la forme item in items, où items représente le tableau source des données et où item est un **alias** représentant l’élément du tableau en cours d’itération »

La directive v-for est souvent utilisée conjointement avec l’attribut :key. Documentation : «Pour expliquer à Vue comment suivre l’identité de chaque nœud, afin que les éléments existants puissent être réutilisés et réordonnés, vous devez fournir un attribut unique key pour chaque élément. »

**Checkpoint 2 :**

1. Vérifiez que la liste s’affiche. Si rien ne s’affiche ou si vous avez une erreur, corrigez pour que ça s’affiche bien
2. Dans la console vue ; *ajoutez un élément à la liste*, en utilisant la syntaxe suivant : nomInstance.$data.nomTableau.push(‘valeur’)

Etape 5: à l’ancienne : façon JavaScript pur : utilisation d’une fonction DOM pour ajouter dynamiquement un nom au tableau

**Difficile** (instructions à interpréter et compléter) :

1. D’abord on ajoute un élément **Input** de type text, avec un id
2. On ajoute un élément **button** avec un id
3. Ensuite on ajoute du code DOM / JS dans la fonction ***mounted*** de l’objet Vue

Cette fonction est appelée juste après la création de l’instance Vue.

document.querySelector('#ajout').addEventListener('click',() => {

let nvNom = document.querySelector('#nouveauNom');

app.noms.push(nvNom.value);

nvNom.value = '';})

La méthode ***querySelector*** permet de retourner le 1er élément qui correspond à un sélecteur CSS dans le document.

La méthode ***addEventListener : « Documentation***

La méthode **addEventListener()** d'[EventTarget](https://developer.mozilla.org/fr/docs/Web/API/EventTarget" \o "EventTarget est une interface implémentée par des objets qui peuvent recevoir des évènements et avoir des auditeurs.) met en place une fonction à appeler chaque fois que l'événement spécifié est remis à la cible. Les cibles courantes sont un [Element](https://developer.mozilla.org/fr/docs/Web/API/Element" \o "Element est la classe de base la plus générale à partir de laquelle tous les objets d'un Document héritent. Il n'a que des méthodes et des propriétés communes à tous les types d'éléments. Les classes plus spécifiques héritent d'Element. Par exemple, l'interfac), le [Document](https://developer.mozilla.org/fr/docs/Web/API/Document) lui-même et une [Window](https://developer.mozilla.org/fr/docs/Web/API/Window" \o "L'objet window représente une fenêtre contenant un document DOM ; la propriété document pointe vers le document DOM chargé dans cette fenêtre.), mais elle peut être tout objet prenant en charge les évènements (comme [XMLHttpRequest](https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest)). »

La méthode **addEventListener** permet donc d’ajouter un gestionnaire d’évènement, associé à un élément DOM, qui « écoute » un certain type d’évènement, par exemple le click de souris.

Cette façon de faire est traditionnelle mais laborieuse, elle se base sur les fonctions DOM.

Etape 6: utilisation de gestionnaire d’évènements Vue

**Documentation** : « Nous pouvons utiliser l’instruction v-on pour écouter les évènements du DOM afin d’exécuter du JavaScript lorsque ces évènements surviennent.»

Eventuellement enregistrez sous un nouveau nom. Supprimez le code de la méthode **mounted**

1. Dans le bouton, ajout d’une directive :

<button @click="ajoutNom" >

@click est le raccourci pour la directive v-on :click

ajoutNom est le nom de la méthode Vue qui est appelée lorsqu’on clique sur le bouton.

1. Ajout d’une méthode ajoutNom

Ajout d’une méthode ajoutNom dans le composant Vue

methods:{

ajoutNom(){

}}

1. Complétez le code

Indication : pensez à rajouter un attribut réactif à l’instance vue avec une liaison bidirectionnelle.

Etape 7: accès à une API (Application Programming Interface) à partir du navigateur

Nous allons utiliser dans nos composants Vue le composant Axios, qui permet de faire des requêtes [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) à partir du navigateur. Ces requêtes se font de manière asynchrone, pour ne pas bloquer le programme en attente de réponse de l’API.

Axios est actuellement l’outil le plus utilisé pour effectuer des requêtes http dans des programmes JavaScript : <https://www.npmtrends.com/ajax-vs-axios-vs-fetch-vs-jquery>

Pour une discussion technique : <https://www.quora.com/Is-Axios-a-good-replacement-for-Ajax-to-make-HTTP-calls>

**Tableau comparatif de l’ensemble des librairies.**



A mon avis, aujourd’hui, c’est ce qu’il y a de mieux actuellement. Il faut également surveiller l’API fetch qui pourrait devenir rapidement un nouveau standard … (<https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch>) …. Ou bien ne jamais percer.

Axios est une bibliothèque JavaScript : <https://github.com/axios/axios>

1. Récupérez et copiez dans le projet le fichier « 030\_IntroAxios.html »

Combien y a-t-il de balises script ? A quoi sert chaque balise ?

1. Récupérez et copiez dans le projet le fichier 030\_axios.js

Dans le composant Vue, sans donnée, rattachée à l’ancre définie en a. , insérez le code suivant dans la fonction **mounted()** :



**Checkpoint 3 :**

1. Naviguez sur le fichier et en mode débogage vérifiez que les valeurs sont retournées dans la console, à l’intérieur de l’objet JavaScript renvoyé par la requête axios (qui se nomme object).
   1. Qu’est ce que abilities ?
   2. Quelle est la valeur de l’attribut name ?
   3. Qu’est ce que stat
2. Faites une recherche et indiquez à quoi sert la fonction Vue mounted

**Exercice 2 :**

* 1. Affichez dans la page le nom du pokemon obtenu
  2. Affichez dans la page la valeur de toutes les stats
  3. Affichez dans la page la valeur de la statistique « defense »