Passer à Vue CLI

Préparation du TP

Il vous faut installer la dernière version de Node afin d'avoir accès au gestionnaire de paquet NPM dans votre terminal.

Puis rendez-vous à l'url ci-dessous pour créer votre propre dépôt. Ensuite clonez le en local via Git.

URL dépôt tutoriel : https://classroom.github.com/a/7etjLAJi

Une fois le projet cloné vous devrez ouvrir un terminal et vous rendre dans ce dossier puis taper la commande suivante:

```
npm install
```

Cela va installer les paquets et dépendance nécessaires pour le projet. De plus vous possédez un projet contenant plusieurs fichiers/dossier. Le descriptif de ces fichiers est indiqué dans le cours.

Après l'installation vous pourrez utiliser les commandes vue. Pour lancer votre projet en local exécuter la commande suivante:

```
npm run serve
```

Enfin n'oubliez pas de prendre la première partie du Tutoriel, nous allons utiliser le code qui a été écrit dedans pour le replacer aux bons endroits de nos fichiers .vue .

Travailler avec les composants

Nous allons récupérer le code fait dans le tutoriel précedant et le replacer dans notre projet pour revenir au même état qu'à la fin du premier tuto.

Vous noterez que le fichier index.html est contenu dans le dossier public. On pourrait duppliquer notre code dedans mais ça n'a pas vraiment de sens dans un projet vue-cli. De même votre fichier app.js a disparu au profit d'un fichier main.js. Son utilisé est sensiblement la même mais là aussi nous n'allons pas écrire de code dedans.

Nous avons cependant accès à un fichier App.vue.

Pour rappel un fichier .vue est un fichier qui peut contenir à la fois du html dans sa partie `<template>`, du css dans sa partie `<style>` et du javascript dans sa partie `<script>`. Vous l'avez sans doute deviné c'est dans ce type de fichier que l'on va venir mettre à la fois le code de notre ancien app.js et notre index.html de la première partie du tuto.

En vous aidant du cours, essayez de replacer votre code dans les bonnes parties.

Remarque: Le projet par défaut intègre déjà notre ancien code css. Pour le moment nous allons mettre de côté la balise `<style>`. Elle est censée contenir le css qui était présent dans le fichier default.css du premier tutoriel mais nous verrons cela à la toute fin.

Une fois le code replacé de la bonne manière vous devriez executer votre application avec un `npm run serve` et verifier sur l'adresse http://localhost:8080/ que vous ayez bien le même résultat qu'à la partie 1.

Créer son composant

Nous voyons bien que le code permettant d'afficher la carte de personnage se répète. C'est un peu dommage de devoir recopier sans cesse. Et imaginons si nous voulions donner plus de détails dans la carte, le code plus lourd se retrouverait en plus duppliqué.

Vue donne accès à la notion de composant et permet de simplifier les répétitions de code. Et ce à travers un type de fichier: les fichiers .vue.

Et oui, le fichier App.vue utilisé précedemment était déjà un composant, celui par défaut qui va appeler tous les autres. En quelque sorte le point d'entrée de notre application ! Mais voyons ça plus en détail.

Rendez vous dans le dossier components

Nous allons essayer de placer le code qui génère la carte de personnage dans un composant appelé `<CartePersonnage>`. Pour ce faire nous allons d'abord créer un fichier CartePersonnage.vue dans le dossier component. C'est ce fichier qui viendra recevoir notre code. Pour rappel un fichier .vue se décompose en 3 balises `<template>`,`<script>` et `<style>`.

Ce composant aura pour but d'afficher les informations d'un personnage. Nous l'appelerons ensuite autant de fois que nécessaire. Cependant si on s'amuse à déplacer le code qui se répète dans ce composant on va rapidement se heurter à plusieurs problèmes:

- La data characters qui contient la liste des personnages se trouve dans l'App.vue, si on cherche à appeler le composant CartePersonnage, qui est censé utiliser les données d'UN et UN seul personnage, comment connaîtra t-il cette liste ? Pire comment même saura t-il de quel personnage il s'agit à chaque pas de la boucle ?

C'est là que vont intervenir les props !

Les props, un moyen de paramétrer notre composant.

Nous avons donc créer un composant CartePersonne.vue. On va essayer de faire en sorte qu'il affiche à chaque fois les informations d'un seul Personnage, et ce personnage pourrait-être n'importe lequel. Le mieux serait d'injecter le personnage en paramètre de ce composant et c'est exactement ce que nous allons faire.

Les props sont comme les attributs d'un élément Html classique sauf que là c'est vous qui décidez comment s'appelle cet attribut et ce qu'il contient.

Partie `<script>` du composant CartePersonnage

On va rajouter l'attribut props qui sera un tableau de valeur. Notre composant possèdera au moins une props: *personnage* qui représentera un objet de type personnage.

Partie `<template>` du composant CartePersonnage

Maintenant vous pouvez duppliquer la partie de code Html qui sert à afficher les infos de la carte Personnage et l'insérer dans le template. Sauf que cette fois l'objet *character* duquel vous récuperez l'*image*, le *name* est en fait l'objet issu de votre props.

Partie `<template>` du composant App

Dans ce fichier vous pouvez à la place du code qui se répète appeler votre composant `<CartePersonnage>` n'oublier pas la props qui contiendra l'objet *character* de votre boucle.

App.vue

Il ne faut pas oublier dans le `<script>`d'importer le composant et de l'ajouter à l'attribut *components* du composant principal App.vue. Ainsi que l'appeler dans la partie `<template>`.

Nous sommes loin d'en avoir terminé en effet le bouton Ajouter ne semble plus fonctionner. C'est le sujet de notre prochaine partie

Emmettre des évènements entre composants

Le bouton Ajouter ne fonctionne donc plus. C'est en réalité logique, sa méthode *add()* se trouve actuellement dans l'App.vue et pourtant le bouton est dans le CartePersonnage.vue.

Le premier réflexe serait de déplacer la méthode *add()* dans CartePersonnage.vue. Mais un autre problème surviendrait, la méthode *add()* utilise la data team qui elle se situe dans App.vue et ne peut pas être déplacé dans CartePersonnage sans perdre tout son sens. Un vrai casse-tête !

Il faudrait que lorsqu'on appuie sur le bouton dans notre composant CartePersonnage, l'information remonte dans son composant père App et ce dernier agira en fonction de cette information. Et bien c'est tout à fait faisable grâce au évènement et la directive *v-on* que nous avons déjà utilisé. Sauf que cette fois nous allons créer nos propres évènements!

Partie `<script>` du composant CartePersonnage

On va créer une méthode *sendAdd()* dans ce composant aussi. Sauf que cette fonction add ne fera pas la même chose. A la place elle va émettre un évènement. Vous pouvez émettre un évènement grâce à la fonction $emit. Nous allons envoyer l'évènement 'ajout' et envoyer aussi en paramètre le *character* qui a été cliqué.

Partie `<template>` du composant CartePersonnage

On vérifie bien que notre bouton appelle la fonction *sendAdd()* lors d'un click.

Partie `<script>` du composant App

La fonction add() que nous avions déjà définit devrait normalement rester identique.

Partie `<template>` du composant App

On vérifie bien que notre bouton appelle la fonction *add()* lors de la réception de l'évènement 'sendAdd'.

On remarque cependant un problème le bouton est présent dans l'équipe choisie ce qui n'a pas de sens. Essayez de rajouter une props dans votre composant pour éviter cela!

Consommer une API

Notre petit tutoriel touche à sa fin mais il reste cependant un aspect d'importance. Depuis le début nous utilisons un tableau de personnages que nous avons défini en le récupérant depuis un fichier JSON. Mais je souhaiterai afficher la totalité des personnages du jeu SmashBros.

La première idée en tête serait d'étendre notre tableau, mais non seulement c'est long mais aussi peu pratique et maintenable. Il est évident que pour de plus gros projets nous aurions accès à une base de données ou tout autre structure qui permet de recevoir des données en quantités et effiacement. L'une de ces solutions c'est la requête à une API.

Tester notre API

Pour rappel une requête API se fait depuis une URL. Pour ce TP nous allons utiliser une API en libre accès qui permet de récupérer des données sur les personnages du jeu Nintendo Smash Bros.

Voici la DOC de l'API: https://smashbros-unofficial-api.vercel.app/ Et ici vous trouverez l'URL que nous allons utiliser pour nos requêtes: https://smashbros-unofficial-api.vercel.app/api/v1/ultimate/characters

Testez cet API dans Postman pour voir ce qu'elle renvoie. Par chance les données sont au même format que celle contenues dans notre fichier JSON. Quelle chance !

Axios et les requêtes API.

Nous allons maintenant chercher à remplacer notre tableau *character* de data par le tableau de la réponse JSON de l'API. Mais avant de faire ça, il faudrait encore être capable d'appeler notre API depuis notre application Vue comme nous le faisons depuis Postman.

Pour cela nous allons utiliser un package (une sorte de librairie Javascript) très pratique et simple d'utilisation: Axios. Axios permet de faire des requêtes HTPP de manière très simple et rapide.

Partie `<script>` de notre composant App

Vous trouverez dans le cours un exemple d'utilisation d'axios. Essayer de l'intégrer à votre composant App en utilisant l'url de l'API donnée plus haut.

Remarque: L'appel à l'API devra se faire avant que votre composant se n'affiche dans l'idéal. Peut-être que les hooks de cycle de vie pourraient vous êtres utile.

Voilà maintenant que vous avez correctement appelé votre API et qu'elle a remplacé votre data character vous pouvez maintenant afficher une liste de personnages bien plus grande !

Attention: Si votre liste ne se charge pas sur le navigateur, veuillez vérifier votre console et qu'il n'y ait pas le message suivant:

> Access to XMLHttpRequest at 'https://smashbros-unofficial-api.vercel.app/api/v1/ultimate/characters' from origin 'http://localhost:8080' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

Si c'est le cas alors vous êtes bloqué par un proxy, pas de panique au lieu d'utiliser l'url de l'API fournie utilisez la suivante: "http://localhost:8080/api/v1/ultimate/characters

Dernier travail corriger le css

Pour terminer nous pouvons adapter le CSS au composant App et CartePersonnage. Tout d'abord nous allons devoir désactiver la feuille de style default.css implementer sur la page.

- Rendez-vous dans le public/index.html et commenter la ligne qui permet d'intégrer le script default.css
- Le css est maintenant désactivé
- Ensuite récuperer les éléments css contenue dans le public/default.css et replacer les propriétés css dans la partie `<style>` de vos composants CartePersonnage et App en faisant correspondre les bonnes règles aux bons endroits. Voilà votre projet est maintenant terminé.