***VUE(CLI)***

**Dure totale du module : 21H00**

Table des matières

[Objectifs pédagogiques : 3](#_Toc111649287)

[Components Communication 4](#_Toc111649288)

[$emit() (communication composant enfant vers composant parent) 4](#_Toc111649289)

[Exercice : component communication ($emit()) test :réagir à un évènement custom 6](#_Toc111649290)

[Exercice : component communication ($emit()) test :réagir à un évènement custom + données 7](#_Toc111649291)

[.$emit() (plus en détails) 8](#_Toc111649292)

[Exercice : component communication ($emit()) en mode objet 9](#_Toc111649293)

[Exercice : component communication ($emit()) + nouveau composant 11](#_Toc111649294)

[Exercice : component communication ($emit()) + nouveau composant + suppression 12](#_Toc111649295)

# Objectifs pédagogiques :

Être capable de comprendre les enjeux des Frameworks

Être capable d'initialiser et paramétrer un projet grâce à un Frameworks front-end

Être capable de réaliser des interface utilisateur grâce à un Frameworks front-end

Être capable de mettre en place un système de Routing (url)

Être capable de gérer des requêtes HTTP (BDD)

Être capable de connecter une application à une base de données

# Components Communication

## $emit() (communication composant enfant vers composant parent)

Nous avons vu la notion de **props** dans VUEJS, cela nous permettait de transmettre des informations (des données) depuis un composant parent vers son composant enfant, nous avons également constaté que pour établir cette communication de données entre les composants VUE se base sur un flux de données UNIDIRECTIONNEL (**unidirectional dataflow**), c’est-à-dire que pour des raisons d’optimisation / sécurité / etc…, on ne pouvait pas directement faire des modifications d’une **props** (une données injectée) dans le composant enfant (pour résoudre ce problème nous avions assigné une **data** à la **props** que l’on voudrait modifier).

Rappel : dans UnAmi .vue, pour pouvoir modifier la **props** premium dans ce composant on était passé par une **data** à laquelle on assigne la props.

export default {

props:{

premium:{

type:Boolean,

required:false,

default:false

}

},

data(){

return{

premiumData: this.premium

}

},

Dans VUE c’est une approche au cœur du système de composant (et leur imbrication), sur des applications SPA (Single Page Application).  
🤔 Quand on y pense :   
Généralement c’est le root composant (racine) soit App.vue, qui va héberger / faire appel à des composants enfant (UnAmi.vue).  
Mais quand on commence à complexifier les applications (gérant beaucoup de données, de fonctions, etc…), nous allons vouloir communiquer dans le sens inverse , que le composant enfant communique des informations au composant parent. (C’est du **unidirectional dataflow** mais dans l’autre sens).

Pour faire en sorte qu’un composant enfant envoie une information vers le composant parent, le composant enfant va devoir émettre un évènement (le changement d’une variable dans par exemple), on peut faire le parallèle où en JS on capte un évènement (‘click’,’keyup’, ‘mouseout’,…) qui contient des informations (les coordonnées du click, la lettre tapée au clavier, etc..), ici c’est le même principe mais on crée des évènements « custom » qui correspondent aux besoins précis des données de l’application.

Dans le composant enfant cette communication se fait via une fonction de VUE :   
**$emit(‘leNomdeDuEvent’)**

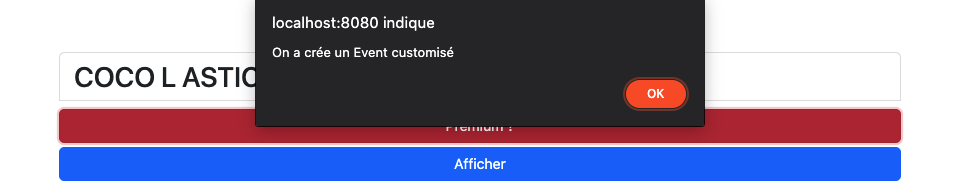
Et dans le composant parent on va réagir à cet évènement :   
**@leNomdeDuEvent** = “ du code JS / une variable / une fonction / etc…“.

Dans l’exercice suivant nous allons faire en sorte que le composant enfant émette un évènement très simple (qui ne contient pas d’informations (pour le moment)) et dans le composant parent on va réagir à cet évènement en exécutant un console.log.

### Exercice : component communication ($emit()) test :réagir à un évènement custom

**Dans unAmi.vue :**   
° Au niveau des **methods**, modifier la méthode afficherPremium, mettre en commentaire la manipulation de la variable premiumData, à la place on va utiliser this.$emit() avec une chaine de caractère en paramètre « mon-event-premium » (c’est le nom de l’évènement que le composant enfant va … émettre).

**Dans App.vue :**   
° Au niveau des **data**, pour chaque objet du tableau lesAmis, rajouter une propriété premium (à true ou false).  
° Au niveau des **methods**, une méthode afficherStatusPremium dans laquelle on fait juste un console log (message au choix).  
° Au niveau du template sur le composant **<un-ami>** :   
 °modifier le **v-bind** de premium pour le lier à la propriété premium de unAmi.  
 °mettre en place un **v-on** qui réagit à notre évènement « mon-event-premium » et appelle la   
 méthode afficherStatusPremium



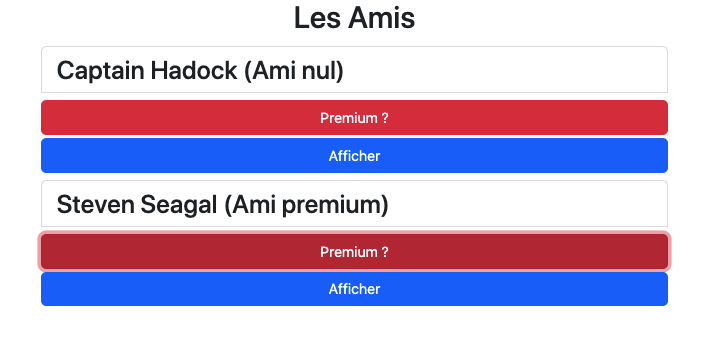
[GitHub](https://github.com/jefff404/cours-vue-cli/tree/8-component-communication-emit())

On vient d’utiliser la fonction $emit(), en passant 1 paramètre : le nom de l’évènement …

### Exercice : component communication ($emit()) test :réagir à un évènement custom + données

**Dans UnAmi.vue :**   
° rajouter une props id (de type String, et required)  
°techniquement on utilise plus premiumData (on peut mettre en commentaire dans les **data**), et dans la condition du <h2> (dans le template) utiliser premium (la **props** directement)  
° Dans les **methods**, rajouter cette props id en second paramètre de $emit()  
  
Le composant enfant émét un event qui contient un id, on va le capter dans le composant parent

**Dans App.vue :**   
°Dans le template sur le composant <un-ami>, on peut rajouter un v-bind de id pour le lier à unAmi.id  
° modifier la methods afficherStatusPremium pour qu’elle prenne une variable en paramètre (leIdDansUnAmi).  
° Dans afficherStatusPremium :   
 ° créer une variable unAmiIdentified, lui assigner le tableau lesAmis (**data**), sur lesAmis utiliser la  
 fonction. find(unTruc => unTruc.id === leIdDansUnAmi)  
 ° à la propriété premium de unAmiIdentified, lui assigner son inverse



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/9-component-communication-emit()-data)

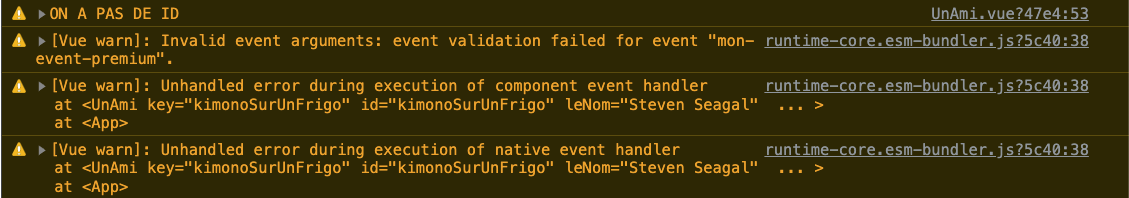
## .$emit() (plus en détails)

Nous venons une première manière d’utiliser la fonction emit(), mais pareillement à props, on peut aller plus loin dans la gestion des évènements avec $emit() en passant par un tableau et même un objet, nous allons voir la version objet pour rajouter dans notre évènement, une fonction qui va vérifier si l’id est bien transmit.

### Exercice : component communication ($emit()) en mode objet

**Dans UnAmi.vue :**

° Dans l’export default, après les props, déclarer emits , un objet { }  
° Dans cet objet on a une chaine de caractère ('mon-event-premium') à laquelle on assigne une fonction (anonyme ou fléchée) qui prend id en paramètre.  
°Dans cette fonction on va faire ceci :   
**SI** id est définie ALORS on return true **SINON** on fait un console.warn(‘MSG au choix’) et on return false  
°Pour tester (voir si le warning se déclenche), dans les methods retirer le paramètre this.id dans la fonction $emit  
°Pensez à le remettre par la suite.



Cette approche permet de renforcer la sécurité des données quand elles communiquent entre composant en passant par des évènements.

[GitHub](https://github.com/jefff404/cours-vue-cli/tree/10-component-communication-emit()-data-details)

Schéma communication composants



### Exercice : component communication ($emit()) + nouveau composant

Nous allons rajouter un composant permettant d’ajouter un ami   
**Dans le dossier components :**   
°Créer un nouveau fichier « NewAmi.vue »  
**Dans main.js :**   
°importer le composant NewAmi, puis déclarer ce composant sous le nom « new-ami » dans l’app

**Dans NewAmi.vue :**   
° Dans le **template** : 1 formulaire avec 3 inputs (pour ajouter un nom, un tél, un mail), et un bouton  
° Pour la partie **script** juste la balise <script>: avec l’export default pour le moment  
° Dans le **script**, dans export default, rajouter un tableau **emits** qui contient une chaine de caractère   
« add-ami » (c’est le nom de l’event)  
° Rajouter des **data** (des string vide) enteredName, enteredPhone, enteredMail  
° Rajouter une **methods** submitData dans laquelle :  
 ° on utilise la fonction **$emit** en lui passant en paramètre « add-ami » (le nom de l’event) puis les data enteredName/Phone/Mail  
° Dans le **template :**   
placer un v-model sur chaque input pour les relier aux **data**sur la balise <form> placer un v-on qui écoute l’évènement « submit » (ajouter le **modifiers** prevent) et qui déclenche notre **methods** submitData  
**Dans App.vue :**   
° sur un composant <new-ami> placer un v-on qui écoute notre event add-ami et qui déclenche une methods add-ami  
° Rajouter une methods ajouterAmi qui prend en param name, phone et email.  
° Dans cette fonction ajouterAmi, créer une variable newAmiContact, lui assigner un objet.  
° Dans cet **objet** on a :   
 - une proprieté id, lui assigner new Date().toISOString(),  
- des proprietés name, phone et email, leur assigner name, phone et email,  
- une proprieté premium, lui assigner false  
° Dans la data lesAmis, faire un push() de newAmiContact  
[GitHub](https://github.com/jefff404/cours-vue-cli/tree/11-component-communication-emit()-new-component)



### Exercice : component communication ($emit()) + nouveau composant + suppression

**Dans unAmi.vue :**   
° Dans le **template**, Sous la <ul> rajouter un bouton avec un **v-on** réagir au click et exécuter la fonction **$emit(),** en lui passant ‘delete’ et id en paramètre  
°Dans le script, mettre en commentaire l’objet emits (plus besoin maintenant qu’on a automatisé l’injection des données, le v-for sur <un-ami> dans App.vue)  
°A la place on va juste passer par **emits** en mode **tableau** (qui contient les nom des event ‘mon-event-premium’ et ’delete).

**Dans App.vue :**   
° Dans le **template**,sur le composant <un-ami> rajouter un **v-on** qui écoute l’event ‘delete’ et qui déclenche une methods effacerAmi   
°Dans le script, rajouter une methods effacerAmi qui a unId en parametre  
°Dans effacerAmi on assigne au tableau lesAmis, lesAmis.filter(), c’est une fonction native de JS  
° Dans les paramètres de filter(), on passe une fonction fléchée ()=>,   
(unePersonne)=> unePersonne.id !== unId

[GitHub](https://github.com/jefff404/cours-vue-cli/tree/12-component-comuunication-emit()-suppression)

