***VUE(CLI)***

**Dure totale du module : 21H00**

Table des matières

[Objectifs pédagogiques : 3](#_Toc111648485)

[Components Communication 4](#_Toc111648486)

[Props (communication composant parent vers composant enfant) 4](#_Toc111648487)

[Exercice : component communication (props) 5](#_Toc111648488)

[Exercice : component communication (props) problème 7](#_Toc111648489)

[Exercice : component communication (props) correction du problème 9](#_Toc111648490)

[Exercice : component communication (props) plus détaillé 11](#_Toc111648491)

[Props + v-for : composants dynamiques 13](#_Toc111648492)

[Exercice : component communication (props + v-for) Composants dynamiques 14](#_Toc111648493)

# 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

## Props (communication composant parent vers composant enfant)

Dans l’application précédente nous avions recrée le setup de base d’une application VUE, nous avons créé le ROOT Component App.vue (le composant principal/racine de toutes les application VUE), et un second composant qui se charge d’afficher un contact (UnAmi.vue), le problème c’est que l’on se retrouve avec 2 fois le même composant avec 2 fois les mêmes data.

L’idée serait plutôt de pouvoir utiliser ce composant en lui injectant des données.

Ce concept d’imbrication de composant est au cœur de la conception des applications web SPA (des application « component oriented »)

Le système de composant améliore la flexibilité lors du développement d’une app (on peut réutiliser facilement un composant dans plusieurs parties de l’application).

Nous allons donc maintenant nous intéresser à comment faire communiquer des composants.

Avec ce système de composants imbriqués, VUE dispose d’un concept pour faire communiquer un composant « parent » et son composant Child (enfant).

Il s’agit de **props**, un TABLEAU dans lequel :

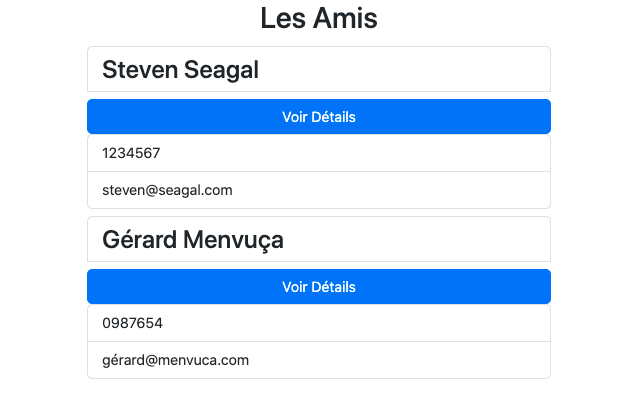
* + - 1. On va définir les propriétés du composant enfant que l’on rend accessible au composant parent

On va pouvoir utiliser des attributs HTML « custom » pour injecter des données dans un composant enfant.

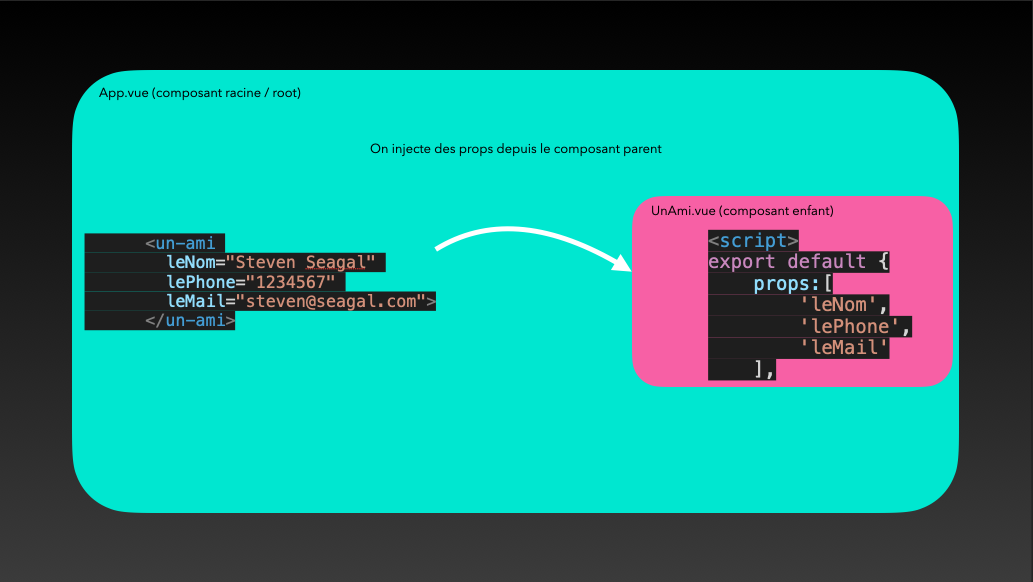
### Exercice : component communication (props)

**Dans UnAmi.vue :**   
° Dans l’export default, rajouter une propriété props, un tableau qui va contenir plusieurs chaines de caractères : « leNom », « lePhone » et « leMail »  
° Pensez à modifier les données que l’on affiche dans le Template, avec l’interpolation désormais on va afficher les **props**

**Dans App.vue :**° Sur nos balises <un-ami>, on va rajouter des attributs HTML qui ont strictement le même nom que les **props** en leur attribuant des valeurs au choix.



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/3-component-communication-props)



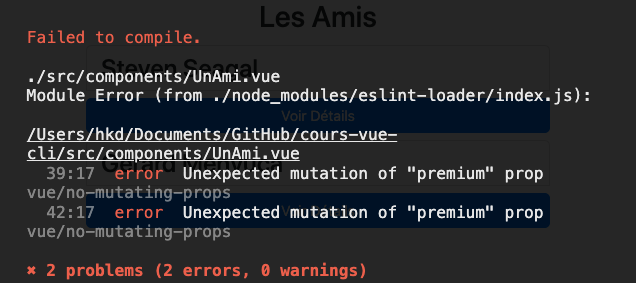
### Exercice : component communication (props) problème

On va tout casser !

**Dans UnAmi.vue :**   
° Au niveau des props, rajouter une props « premium » (pour savoir si on a des ami de qualité 😁)  
° Dans methods, rajouter une méthode « afficherPremium » dans cette méthode :   
 °SI premium est bien égal à la chaine de caractère « 1 », alors on met la props premium à « 0 »,   
 SINON on met la props premium à « 1 »  
° Dans le template, au niveau du <h2>, on va rajouter une interpolation {{}}, dans laquelle on aura une condition ternaire : SI premium est bien égal à « 1 » on retourne « Ami Premium » sinon on retourne « Ami Nul »

**Dans App.vue :**   
° Sur les balises <un-ami> on rajoute un attribut premium en injectant soit « 1 » ou « 0 ».

Yes 🤙 :



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/4-component-communication-props-problem)

Explication du « Bug » :

Pour la communication entre des composants avec props, Vue se base sur le concept du   
Unidirectionnal DataFlow (flux de données unidirectionnel), principalement pour des raisons de sécurité, de performances et de maintenance (debugging).

Dans notre cas les props que l’on injecte depuis App.vue vers UnAmi.vue ne devraient être modifiée QUE dans App.vue.

En revanche nous, dans UnAmi.vue, on essaie de modifier la props premium :

afficherPremium(){

if(this.premium ==='1'){

this.premium = '0';

}

else{

this.premium = '1';

}

}

Nous allons à l’encontre de ce flux de données **UNIDIRECTIONNEL**, c’est pour cela que nous avons ce type d’erreur :

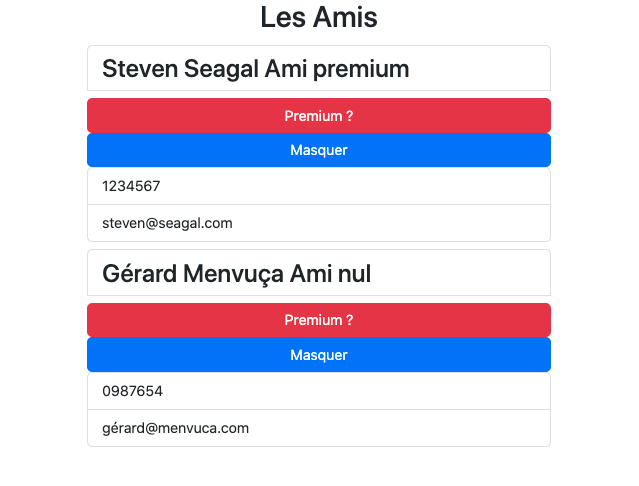
error Unexpected mutation of "premium" prop vue/no-mutating-props

Pour corriger ce problème il faudrait que le composant parent soit informé que nous voulons modifier les **props** dans l’enfant. De cette manière le parent pourra mettre à jour ses data (plus tard).

Une autre manière de corriger cela serait de créer au sein du composant enfant (UnAmi.vue), une **data** liée à notre **props** premium.

### Exercice : component communication (props) correction du problème

**Dans UnAmi.vue :**   
° Au niveau des data, rajouter une data « premiumData » à laquelle on assigne la **props** premium  
° Dans la methods « afficherPremium », remplacer les props « premium » par les data « premiumData »  
° Dans le template, dans l’interpolation, remplacer les props « premium » par les data « premiumData »  
° Dans le template, sous le <h2> rajouter un bouton qui réagit au click pour exécuter la methods afficherPremium



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/5-component-communication-problem-solved)

Nous avons vu la configuration de base pour utiliser les props :

props:[

'leNom',

'lePhone',

'leMail',

'premium'

],

Avec cette configuration de base on sait que côté HTML on va injecter des données correspondantes à leNom, lePhone, leMail, premium (pas plus, pas moins).

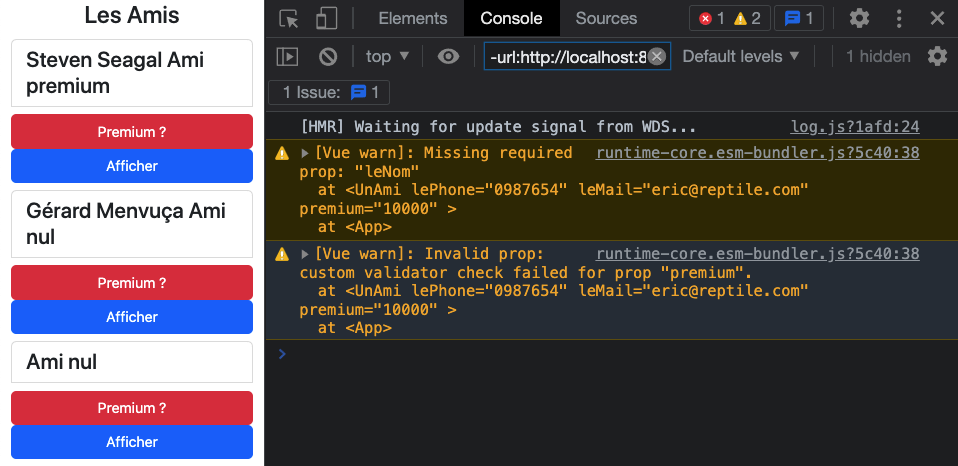
Cette configuration est largement suffisante dans des petites applications, mais quand ça devient complexe ou que l’on travaille en équipe on va vouloir partager + d’informations sur les **props**  
pour que VUE puisse savoir si l’on injecte de mauvaises données ou non, ou si l’on a oublié d’injecter une **props** par exemple.

VUE permet d’aller plus en détails avec l’utilisation des **props**, on modifiant légèrement sa syntaxe, **props** serait désormais un objet { }, plutôt qu’un tableau [ ], on pourrait alors rajouter le typage des props et ou indiquer si la **props** est obligatoire.

### Exercice : component communication (props) plus détaillé

**Dans UnAmi.vue :**   
°Mettre en commentaire tout le tableau **props**, on va créer un objet **props** plutôt.  
°Dans l’objet **props**, on va rajouter plusieurs objets (1 objet pour chaque **props** techniquement) :  
 °leNom{}  
 °lePhone{}  
 °leMail{}  
 °premium{}  
°Dans chacun de ces objets on va avoir une propriété **type**:String (toujours Majuscule pour le type) et propriété **required** à true (sauf premium sa propriété **required** est à false)  
° pour l’objet de la **props** premium, on rajoute une propriété **default** à ‘0’, et une propriété **validator  
°** à la propriété **validator** on lui attribue une fonction qui a une variable **value** en paramètre et qui **retourne** : cette condition value==='1' || value==='0'

**Dans App.vue :**   
° Rajouter un composant <un-ami>, et nous allons créer des erreurs en injectant des props erronées par exemple :   
° On oubli de renseigner la **props** leNom, et dans la **props** premium on va passer une mauvaise valeur « 1000 » par exemple.   
°Ensuite dans le navigateur on va explorer la console pour voir les erreurs qui sont généré par cette nouvelle approche des **props** (plus sécurisé)



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/6-component-communication-props-details)

Pour résumer avec l’approche objet des props :

props:{

premium:{

type:String,

required:false,

default:'0',

validator: function(value){return value==='1' || value==='0'}

}

Nous pouvons gérer le **type** des données qui seront injectées dans le composant enfant, les rendre obligatoire ou non avec **required**, mettre une valeur par **défaut**(on peut utiliser également une fonction dans default), enfin les **validator** permette d’avoir des contrôles très précis et customisé avec des conditions spécifiques (ici par exemple le **validator** n’autorise que la chaine de caractère « 1 » ou « 0 »).

## Props + v-for : composants dynamiques

A ce stade nous savons faire communiquer (injecter) des données d’un composant vers son composant enfant… Mais nous connaissons aussi la directive v-for… 😉

Afin de rendre notre composant encore plus dynamique et plus facilement réutilisable, dans le sens ou cela devient répétitif d’injecter une par une les props depuis le composant parent comme ceci :

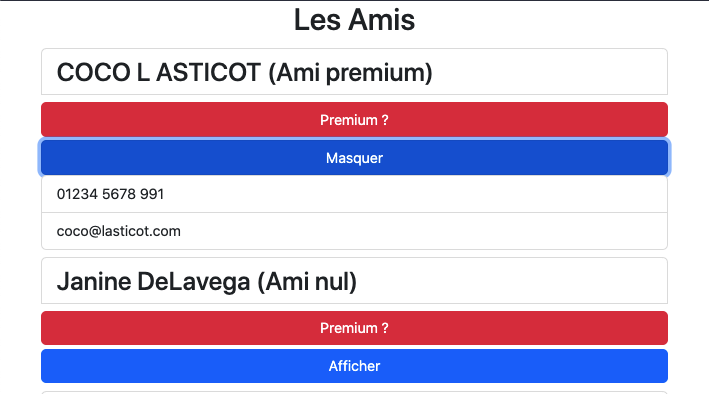
<un-ami leNom="Gérard Menvuça" lePhone="0987654" leMail="gérard@menvuca.com"premium="0"></un-ami>

Dans l’exercice suivant nous allons gérer toutes les données dans les data de App.vue (composant parent) et optimiser leurs injections dans les **props** du composant enfant (UnAmi.vue).

### Exercice : component communication (props + v-for) Composants dynamiques

**Dans UnAmi.vue :**° on va modifier l’objet de la **props** premium (le **type** sera booléen, le **required** et **default** seront à false)  
on peut mettre en commentaire le **validator** (plus besoin si c’est de type booléen)  
° Au niveau des **data** on peut mettre en commentaire l’objet unPote  
° Au niveau des methods, mettre en commentaire (ou supprimer) la partie IF ELSE, on va faire plus simple : on assigne premiumData à son inverse.  
° Dans le template au niveau du <h2> modifier la condition ternaire : on vérifie juste si premiumData est définie

**Dans App.vue :**° Mettre en commentaire (ou supprimer) les composants <un-ami> (on va en utiliser 1 seul).  
° Sur un composant <un-ami> :   
 ° utiliser une directive v-for : avec une variable temporaire unAmi, parcourir le tableau lesAmis.  
 ° Ensuite on va faire des v-bind de :   
 - key pour le lier à l’id de unAmi  
 - leNom pour le& lier à name de unAmi  
 - lePhone pour le lier à phone de unAmi  
 - leMail pour le lier à email de unAmi  
 - premium pour lui assigner true



[GitHub](https://github.com/jefff404/cours-vue-cli/tree/7-component-communication-props-v-for-(dynamic-component))