

ZK



Créer votre première application

vue.js



FRANCK ABGRALL

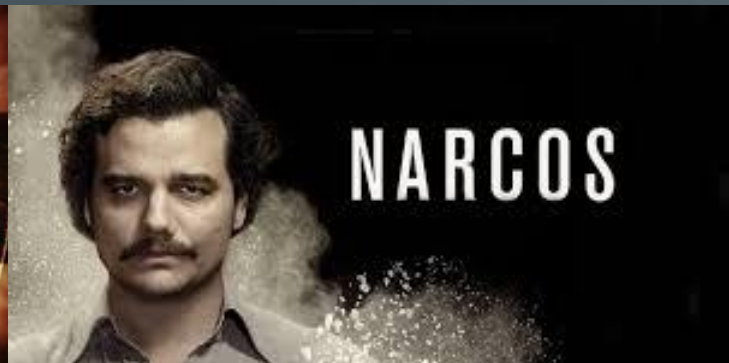
Consultant & Formateur
Zenika Nantes



GREGORY BEVAN

Consultant & Formateur
Zenika Nantes

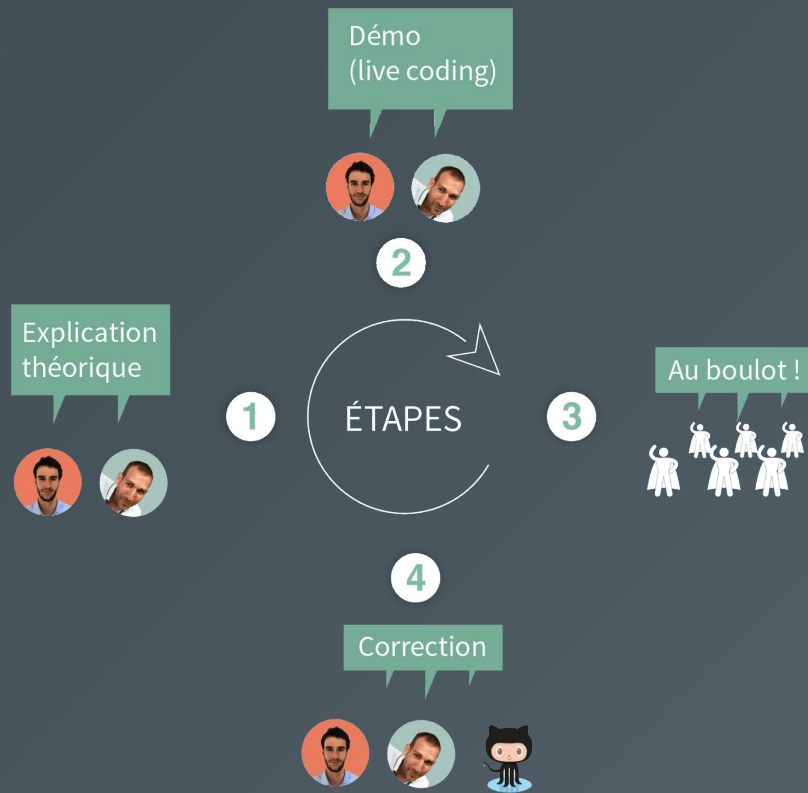
Développer une application pour la gestion de vos séries favorites



Déroulement du codelab



Déroulement du codelab



Déroulement du codelab



// Cloner le repo

```
$ git clone https://github.com/GregoryBevan/devfest-vuejs
```

// Au début de chaque étapes

```
$ git checkout step1
```

// A la fin de chaque étapes

```
$ git add -A && git commit -m "Wow, vuejs is amazing...."
```

Let's go !





Evan YOU

- Framework.js
- 1st Release : 2014
- v2
- ≈ 70000 ★ sur github
- Popularité ↗

Alternatives ...



Vue.js en 3 mots ...



MINIMALISTE

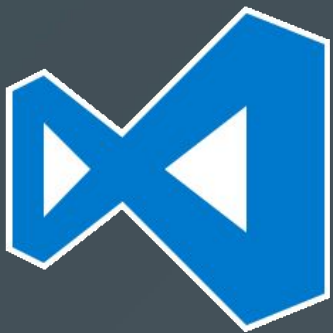


PERFORMANT

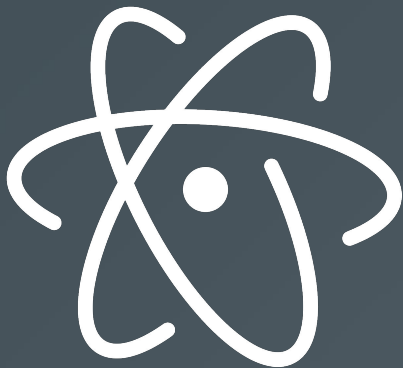


SIMPLE

Editeurs et plugins



vetur



language-vue



vue-js







Étape 1

Initialiser un projet Vue.js

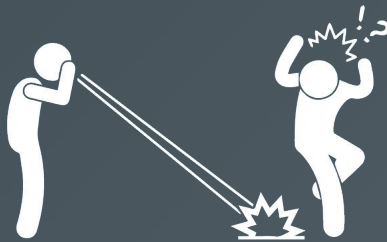


Vue-cli

Principaux **avantages** du template webpack :

- Webpack 
- Single file component (.vue) 
- Rechargement à chaud de l'application 
- ES6 et ESLint  
- Unit et e2e tests 
- ...

Live coding



Objectifs étape 1

5 min



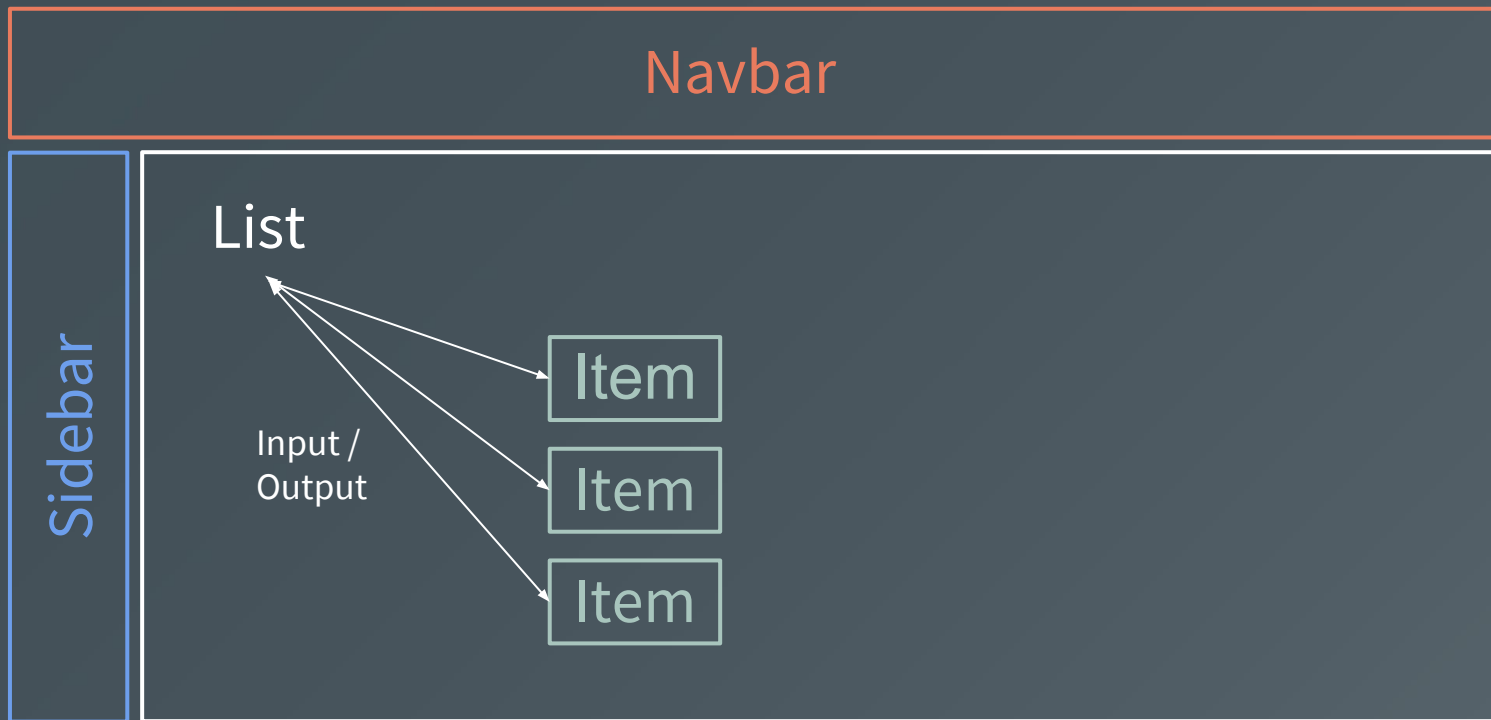
1. \$ git clone <https://github.com/GregoryBevan/devfest-vuejs> && cd devfest-vuejs
2. \$ git checkout step1
3. \$ npm install -g vue-cli
4. \$ vue init webpack . (Générer le projet dans le dossier courant)
5. Renseigner les options du projet **!/ Ne pas accepter ESLint + tests unitaires !/**
6. \$ npm install
7. \$ npm run dev

Étape 2

Créer un composant



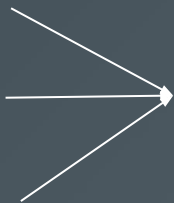
Structure d'une application js en 2017



Architecture orientée composant

Composition “classique” :

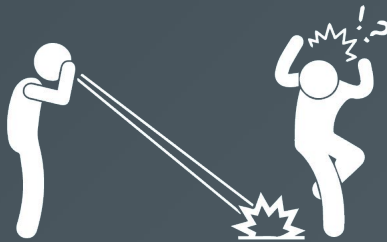
- 1 fichier html (template)
- 1 fichier js (controller)
- 1 fichier css/scss... (style)



Single File Component :

- 1 fichier ... (.vue) 🙌

Live coding



Objectifs étape 2

5 min



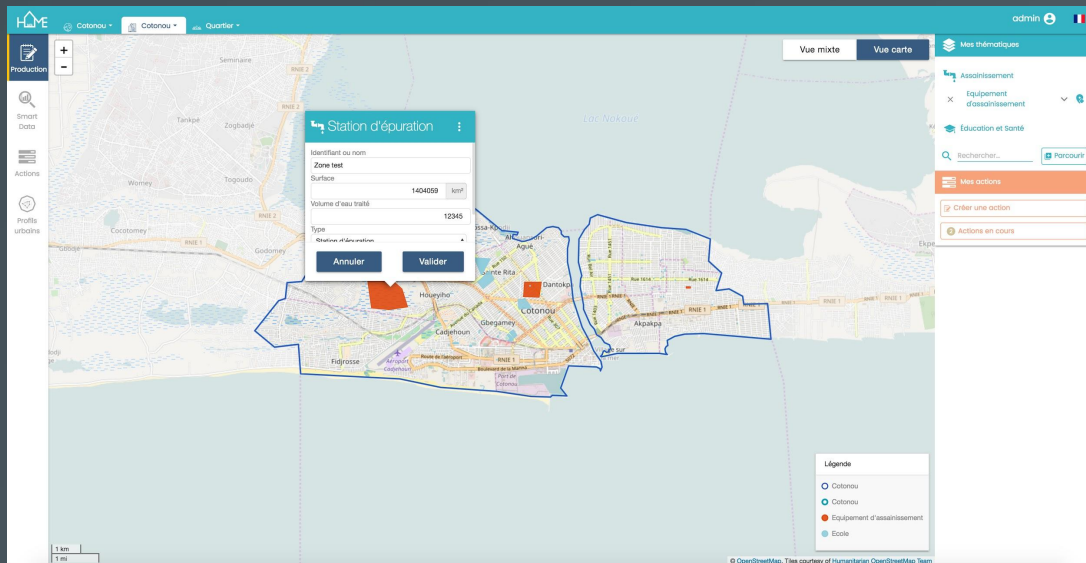
1. `$ git checkout step2 && npm i`
2. Créer un composant **Home.vue** dans **src/components**
💡 Inspirez-vous de **Hello.vue** pour créer le composant
3. Créer un titre dans le template du composant **Home.vue**
4. Remplacer les références à **Hello** par **Home** dans le fichier **router/index.js**. Nous verrons plus loin l'utilité de ce fichier...
5. Supprimer le composant **Hello.vue**

Étape 3

Créer une route 

Vue-router

<https://localhost:4200/map>

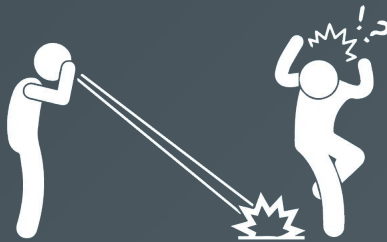


Vue-router

<https://localhost:4200/stats>



Live coding



Objectifs étape 3

4 min



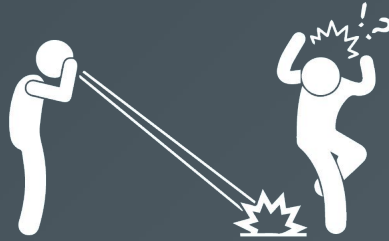
1. `$ git checkout step3 && npm i`
2. Créer un composant ***Favorites.vue*** qui représentera une page secondaire et afficher un titre dans le template du composant
3. Dans le fichier ***router/index.js***, créer la route pour rediriger vers ***Favorites.vue***
 - 💡 *Appuyez-vous sur la syntaxe de la route existante pour créer la nouvelle*
 - 💡 *N'oubliez pas d'importer le fichier ***Favorites.vue*** !*
4. Tester le fonctionnement en tapant l'url **`localhost:8080/#/le-nom-de-ma-route`** dans le navigateur

Étape 4

Créer une jolie nav-bar 🎀

```
<router-link to="home">Home</router-link>
```

Live coding



Objectifs étape 4 :

4 min



1. `$ git checkout step4 && npm i`
2. Dans cette étape, vous devrez compléter le fichier **Navbar.vue** dans **src/components**.
Premièrement, vous devrez placer le composant navbar au dessus du composant `router-view` dans le template de **App.vue**.

💡 *Pour rendre un composant disponible dans la portée d'un autre, il suffit de l'importer et de l'inscrire dans l'option **components** de ce dernier :*

```
components: {  
  'my-component': MyComponent  
}
```

3. Dans le template de **Navbar.vue**, ajouter des composants `router-link` pour naviguer vers les différentes pages de l'application

💡 *Rappel de la syntaxe:*

```
<router-link to="my-new-page-url">My new page</router-link>
```



vue.js

Étape 5

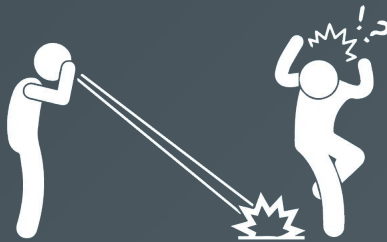
Préparer l'affichage d'une série



L'objet *data* d'un composant



Live coding



Objectifs étape 5 :

6 min



1. \$ git checkout step5 && npm i
2. Créer un composant **Serie.vue**
3. Dans la fonction **data** du composant, créer les propriétés suivantes :

```
name: 'NightClazz'  
summary: 'What an amazing codelab ....',  
image: { medium: 'static/logo.png' }
```

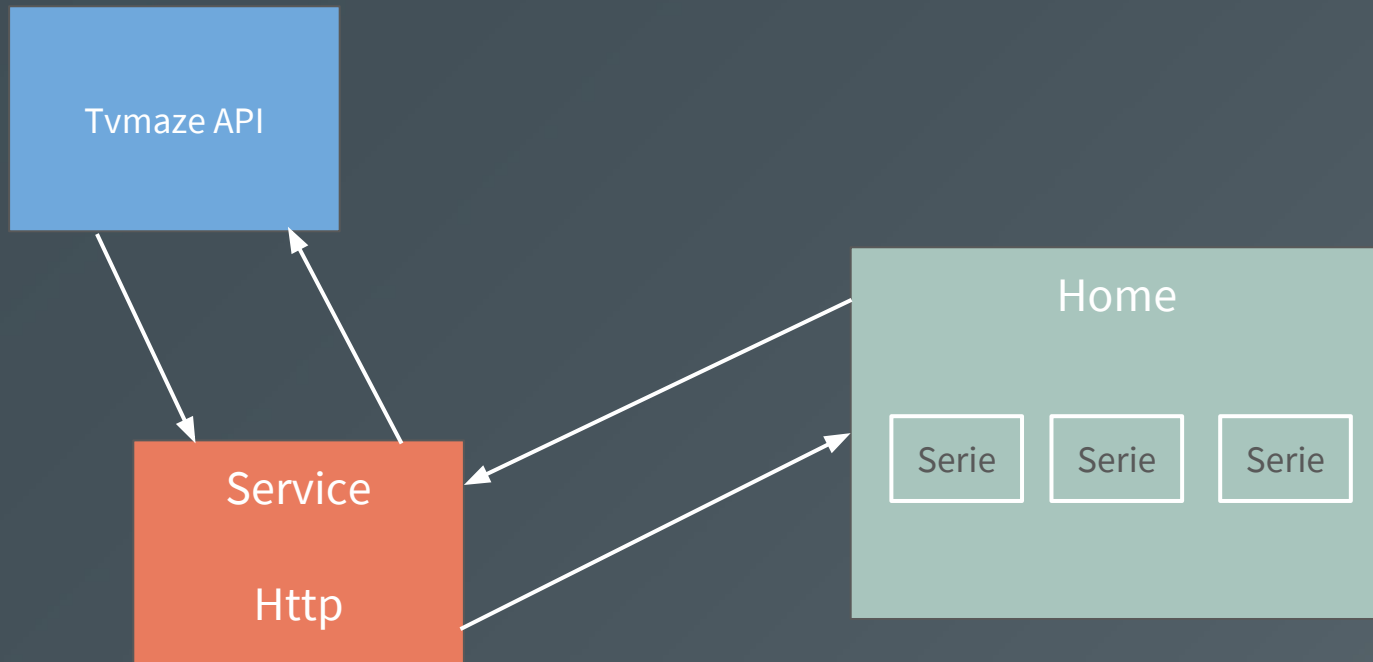
4. Afficher les données dans le **template** du composant
💡 *Pour binder la propriété image sur un élément HTML , utiliser la syntaxe suivante: (Nous expliquerons cette syntaxe dans les étapes suivantes)*
5. Afficher plusieurs composants **Serie** dans le template de **Home.vue**
💡 *N'oubliez pas d'importer **Serie** dans la propriété **components** de **Home.vue***

Étape 6

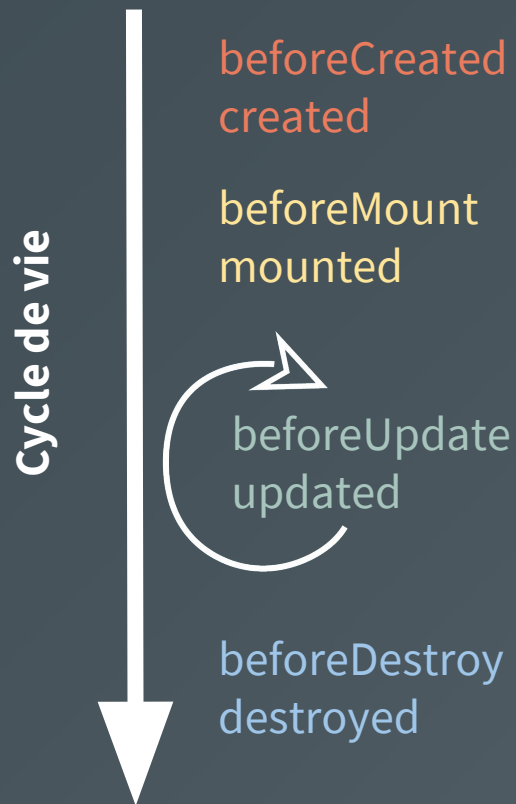
Récupérer les données



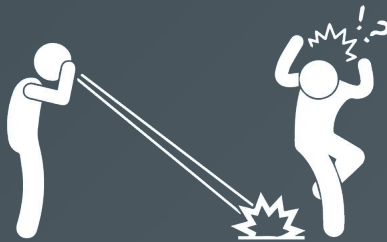
Client HTTP



Cycle de vie d'un composant




Live coding



Objectifs étape 6

5 min

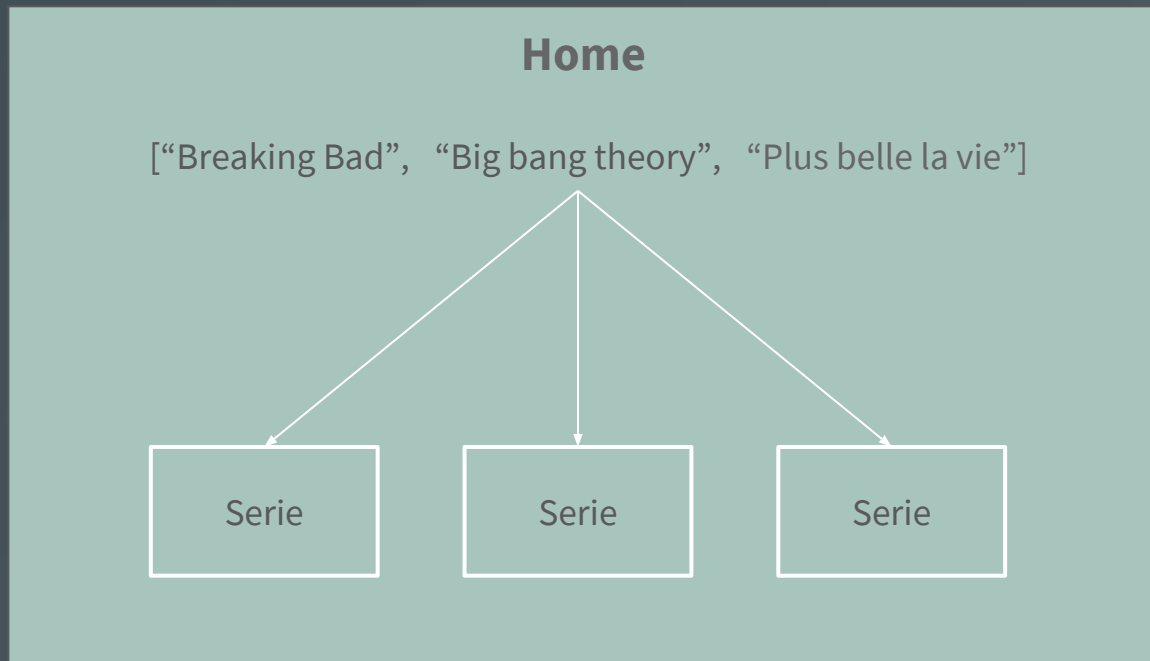


1. \$ git checkout step6
2. Importer le service **series.service.js** dans **Home.vue**
3. Appeler le service depuis le composant **Home.vue** à son **initialisation**. Afficher le résultat dans la console pour s'assurer du bon fonctionnement.
 *Utiliser la fonction hook **mounted** d'un composant pour exécuter du code à son initialisation*

Étape 7

Afficher les séries 

Créer **dynamiquement** des composants



Directives **Vuejs**

- `v-for` => Créer dynamiquement du contenu HTML
- `v-if` => Rendu conditionnel (Insertion dans le DOM)
- `v-show` / `v-hide` => Rendu conditionnel (Afficher / Masquer)
- `v-on` ou `@` => Gérer un événement
- `v-bind` ou `:` => Lier des données
- `v-model` => Met en place un two-way binding sur un formulaire

Directives **Vuejs**

- `v-for` => Itérer sur une liste
- `v-if` => Rendu conditionnel (Insertion dans le DOM)
- `v-bind` ou `:` => Lier des données

Créer **dynamiquement** à partir d'une liste

```
<div v-for="serie in series">{{ serie }}</div>
```



Rendu

```
<div>Breaking Bad</div>  
<div>Plus belle la vie</div>  
<div>Big bang theory</div>
```

Transmettre de la donnée à un composant

v-bind (:) + props

Composant parent (Template)

```
<serie v-bind:dataSerie="data"></serie>
```

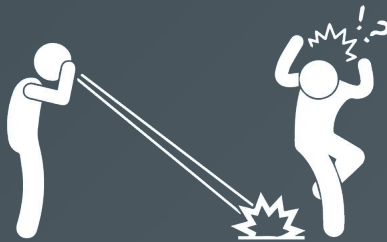
```
<serie :dataSerie="data"></serie>
```



Composant enfant (Controller)

```
props: [  
  'dataSerie'  
]
```

Live coding



Objectifs étape 7 :

8 min



1. `$ git checkout step7 && npm i`
2. Remplacer la fonction **data** par la propriété **props** dans le composant **Serie.vue**. Adapter le template de **Serie.vue** pour qu'il fonctionne avec la props
💡 Rappel de la syntaxe: `props: ['myProp']`
3. Stocker les séries dans la fonction **data** de Home.vue
4. Créer une boucle sur le composant **Serie.vue** avec la directive **v-for**
💡 Rappel de la syntaxe: `v-for="serie in series"`
5. Passer les données des séries à la **props** de Serie.vue
💡 Rappel de la syntaxe: `:<my_props_name>="data"`
6. Certaines séries récupérées via l'api ne possèdent pas d'image. Si une série ne possède pas de propriété "image", vous devrez la masquer grâce à la directive **v-if** pour éviter une erreur
💡 Rappel de la syntaxe: `v-if="myCondition"`

Étape 8

Gestion des favoris 

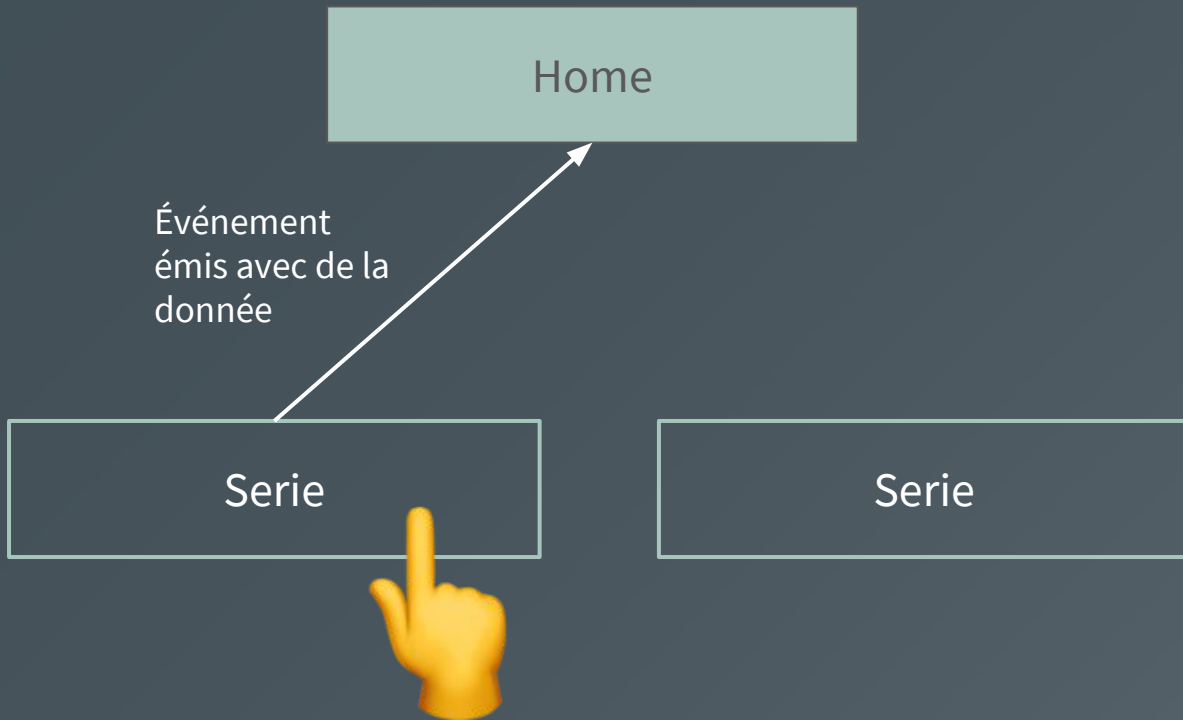
Les événements **natifs**

- click
- focus
- blur
- ...

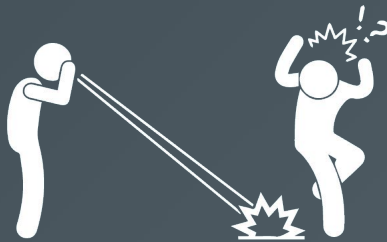
```
<div v-on:click="myFunction()"></div>
```

```
<div @click="myFunction()"></div>
```

Les événements personnalisés



Live coding



Objectifs étape 8 :

8 min



1. `$ git checkout step8 && npm i`
2. Dans **Serie.vue**, créer un événement permettant de remonter la série cliquée au parent
 - 💡 Récupérer un événement de type `click`: `@click="onClick()"`
 - 💡 Les fonctions d'un composant sont définies dans la propriété **methods**:

```
methods: {  
  onClick() {}  
}
```
 - 💡 Émettre un événement depuis le controller: `this.$emit('event-name', data)`
3. Dans **Home.vue**, ajouter / supprimer un favori à la réception de l'événement. Pour cela vous devrez importer et utiliser le service déjà pré-codé dans cette étape (`favorites.service.js`)
 - 💡 Rappel de la syntaxe: `@event-name="myFunc($event)"`
4. Si tout s'est bien passé, vous devriez voir les séries sur lesquels vous avez cliqué sur la page favoris (la récupération des favoris a déjà été pré-codée sur cette page).

Étape 9

Afficher en fonction d'un état



Computed property

Template :

```
{{ message }}
```

Computed property

Template :

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

Computed property

Template :

```
{{ reversedMessage() }}
```

Controller :

```
methods: {  
  reversedMessage() {  
    return this.message.split('').reverse().join('')  
  }  
}
```

Computed property

Template :

```
{{ reversedMessage }}
```

Controller :

```
computed: {  
  reversedMessage() {  
    return this.message.split('').reverse().join('')  
  }  
}
```

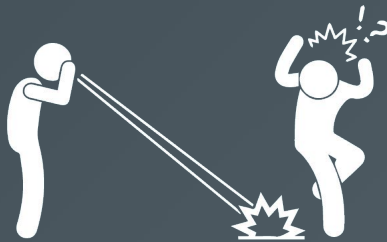
Class binding

```
<div :class="{ 'red': isRed }"></div>
```

```
<div :class="{ 'red': isRed, 'title': isTitle }"></div>
```

```
<div :class="[ isRed ? 'red' : 'blue' ]"></div>
```


Live coding



Objectifs étape 9 :

8 min



1. `$ git checkout step9 && npm i`
2. Dans cette étape, ajouter l'élément `<i class="glyphicon"></i>` au template de **Serie.vue**. Vous devez ajouter une classe sur l'icône en fonction de l'état de la série (favorite ou non). Si la série fait partie des favoris, ajouter la classe **glyphicon-star** sinon ajouter la classe **glyphicon-star-empty**. Vous devez réaliser cette étape grâce aux **computed properties** et au **class binding** :

💡 Résultat attendu: `<i class="glyphicon" :class="???"></i>`

💡 Rappel de la syntaxe class binding:

```
<div :class="[ isRed ? 'red' : 'blue' ]"></div>
```

💡 Syntaxe computed properties :

```
computed: {  
  isRed() { return ... }  
}
```

💡 Stocker **favoritesService** dans la fonction data du composant et utiliser la méthode **isFavorite** de **favoritesService** dans la computed property

Étape 10


Rechercher une série 🙄🙄

v-model

“Controller”

```
data () {  
  return {  
    name: ""  
  }  
}
```

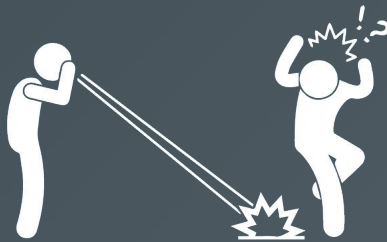
Two way data
binding



Template

```
<template>  
  <div>  
    <input v-model="name" />  
  </div>  
</template>
```

Live coding



Objectifs étape 10 :

8 min



1. Créer un input de type text dans **Home.vue**
2. Ajouter une propriété `search` dans la section data de **Home.vue**
3. Binder la variable `search` sur l'input grâce à `v-model`
💡 Rappel de la syntaxe class binding `v-model="myVariable"`
4. Filtrer la liste des séries sur leurs noms grâce à la variable `search` et aux `computed properties`

This is the end...

...but the beginning of your
Vue.js experience

