

# Bachelier en Informatique de Gestion

## Web : principes de base Projet de Développement Web

Enseignement supérieur économique de type court

Code FWB : 7534 29 U32 D1, 7534 30 U32 D3

Code ISFCE : 4IWPB, 4IPW3



# Table des matières

## Généralités

- 01. Introduction au web
- 03. Outils
- 05. Format XML
- 06. Format JSON

## Front-End

- 12. Structure HTML
- 13. Formulaire HTML
- 14. Mise en forme CSS
- 15. Adaptabilité
- 17. Javascript
- 18. Bibliothèque jQuery
- 19. Composant Vue.js

## Back-End

- 21. Middleware PHP
- 22. Traitement du formulaire
- 23. Architecture MVC
- 24. Données SQL
- 25. Données NoSQL
- 27. Requête asynchrone



= BUROTIX 0

# 19. Component-Based Design

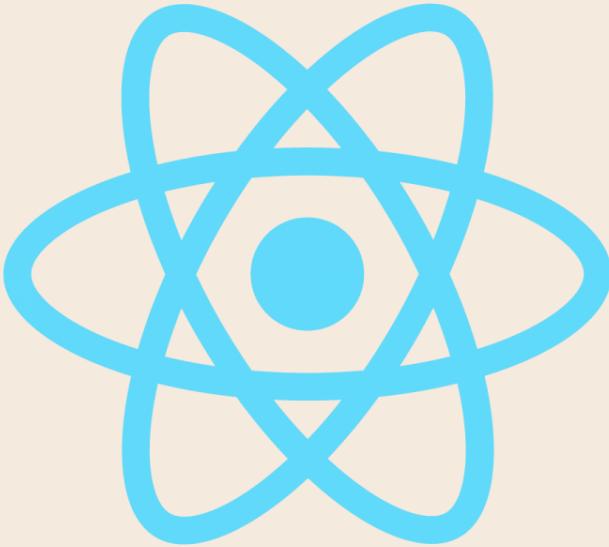
Motivation

Etat de l'art

Vue.js



= BUROTIX 0



Etat de l'Art



= BUROTIX 0

# Qu'est-ce qu'un composant ?

- Contexte : Single Page Application
- Brique de base d'une interface utilisateur
  - exemple : un bouton, une carte, un menu
- Analogie :
  - Lego®
  - assemblage de petits éléments pour construire un tout
- Pourquoi utiliser des composants ?
  - réutilisabilité
  - modularité
  - maintenance simplifiée

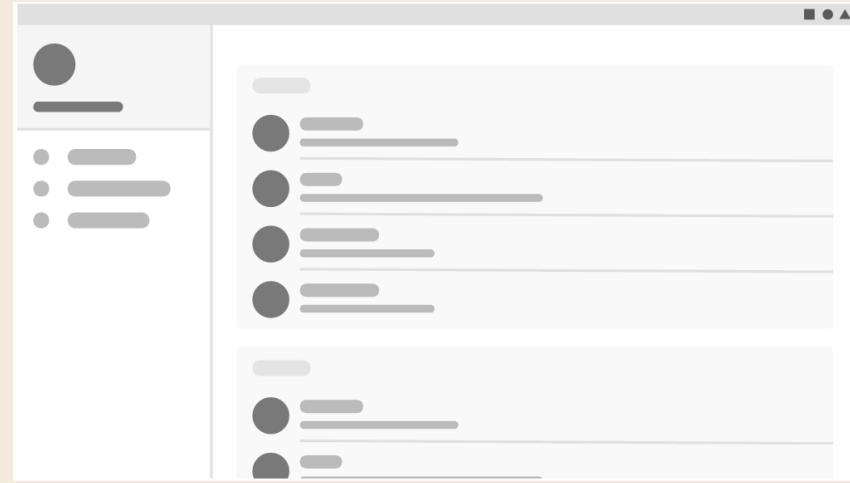


# Exemples

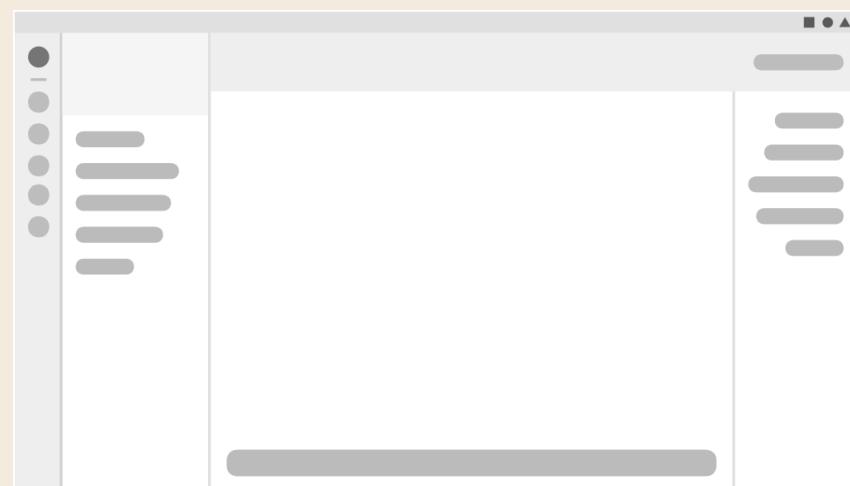
**Projects**

The interface displays three projects in a grid:

- Learning Laravel**: Due on 27-07-2022. Description: Learning basic laravel and create dummy project. Status: 28 Days Left.
- Bachelorthesis**: Due on 28-07-2022. Description: Finish bachelor thesis until to get a degree. Status: 28 Days Left.
- Donating my blood**: Due on 29-07-2022. Description: Donating my blood to get healthier. Status: 29 Days Left.



"inbox"



"discord"



= BUROTIX 0

# Exemples

The screenshot shows the Amazon.fr homepage with a light blue header bar containing the Amazon logo, search bar, and navigation links like 'Meilleures Ventes', 'AmazonBasics', 'Prime', 'Dernières Nouveautés', 'High-Tech', 'Livres', 'Service Client', 'Cuisine et Maison', 'Idées cadeaux', and 'Informatique'. A prominent banner at the top right is titled 'COVID-19 : livraisons, retours et sécurité'.

**Ajoutez Alexa dans toutes les pièces**

echo dot echo echoshow echoshow

**Les top catégories**

- Épicerie (image of a product)
- Rangements (image of a storage box)
- Objets connectés (image of a smartwatch)
- Livres (image of a book cover)
- Meilleures ventes (image of a product)

**Soutenons les entreprises françaises**

- Entreprises françaises (image of a product)
- Boutique des producteurs (image of a product)
- Jouets français (image of a toy car)
- Startup françaises (image of a laptop)
- Découvrir

**Apprendre à coder gratuitement**

8-18 ans

En savoir plus

**Identifiez-vous pour une meilleure expérience**

Se connecter en toute sécurité

chèques-cadeaux

Vous offrez, ils choisissent

Découvrir



# Les composants, l'avenir du "front-end" ?

- Modularité et réutilisabilité
  - Problème
    - avant, interfaces monolithiques, difficiles à maintenir
    - fichier HTML/CSS/JS géants
  - Solution
    - composants => découper l'interface en petits blocs réutilisables
    - exemples : un bouton "print", une barre de navigation
- Collaboration facilitée
  - Travail en équipe
    - Chaque développeur peut travailler sur un composant différent sans conflit.
  - Exemple
    - Une équipe de 5 personnes peut développer une application en parallèle (un composant par personne).



# Les composants, l'avenir du "front-end" ?

- Maintenance simplifiée
  - Mise à jour ciblée
    - Modifier un composant n'interfère pas avec le reste de l'application.
  - Exemple
    - Changer le style d'un bouton se fait en un seul endroit, même s'il est utilisé 100 fois.
- Écosystème et communauté
  - Bibliothèques de composants
    - milliers de composants disponibles
  - Standardisation
    - frameworks modernes tous basés sur les composants
    - norme industrielle



# Les composants, l'avenir du "front-end" ?

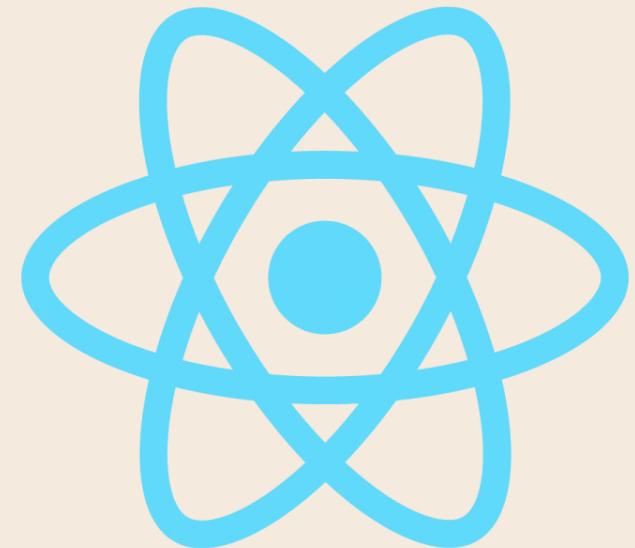
- Adaptation aux besoins modernes
  - Applications complexes
    - gestion d'interfaces dynamiques
    - ex : réseaux sociaux, dashboards
  - Performance
    - uniquement composants nécessaires chargés
- Apprentissage
  - Intégration facilitée des nouveaux collaborateurs
    - interface découpé en "morceaux logiques" plus facilement compréhensible



= BUROTIX 0

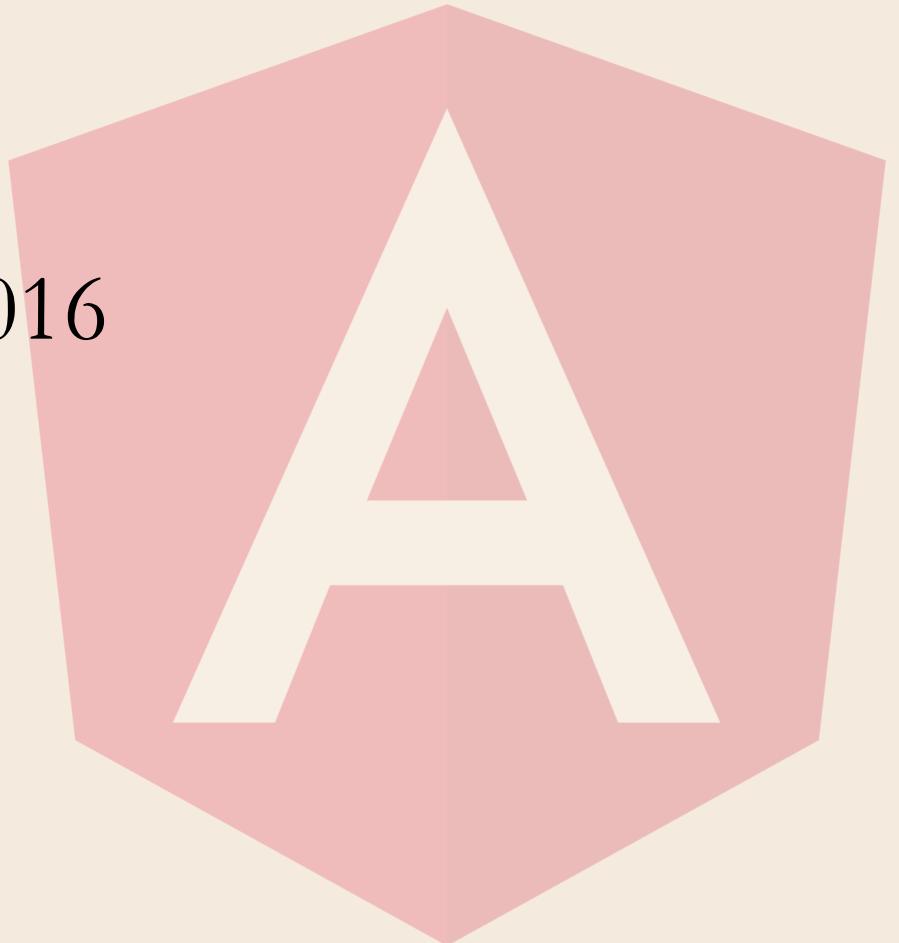
# React

- Développé par Facebook
- Introduction en 2013
- Basé sur Javascript (syntaxe JSX)
- Librairie JavaScript
- Facilité de développement
- Performance
- Utilisé par Meta, Airbnb



# Angular: Introduction

- Développé par Google
- Introduction en 2010. Réécrit en 2016
- Basé sur TypeScript
- Framework complet
- Structure claire et modulaire
- Facilité de maintenance
- Utilisé par Google, Microsoft

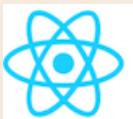


# Vue: Introduction

- Développé par Evan You (ancien employé de Google)
- Introduction en 2013
- Basé sur JavaScript
- Framework progressif
- Open Source
- Faible courbe d'apprentissage
- Performances
- Utilisé par Alibaba, Xiaomi, Adobe



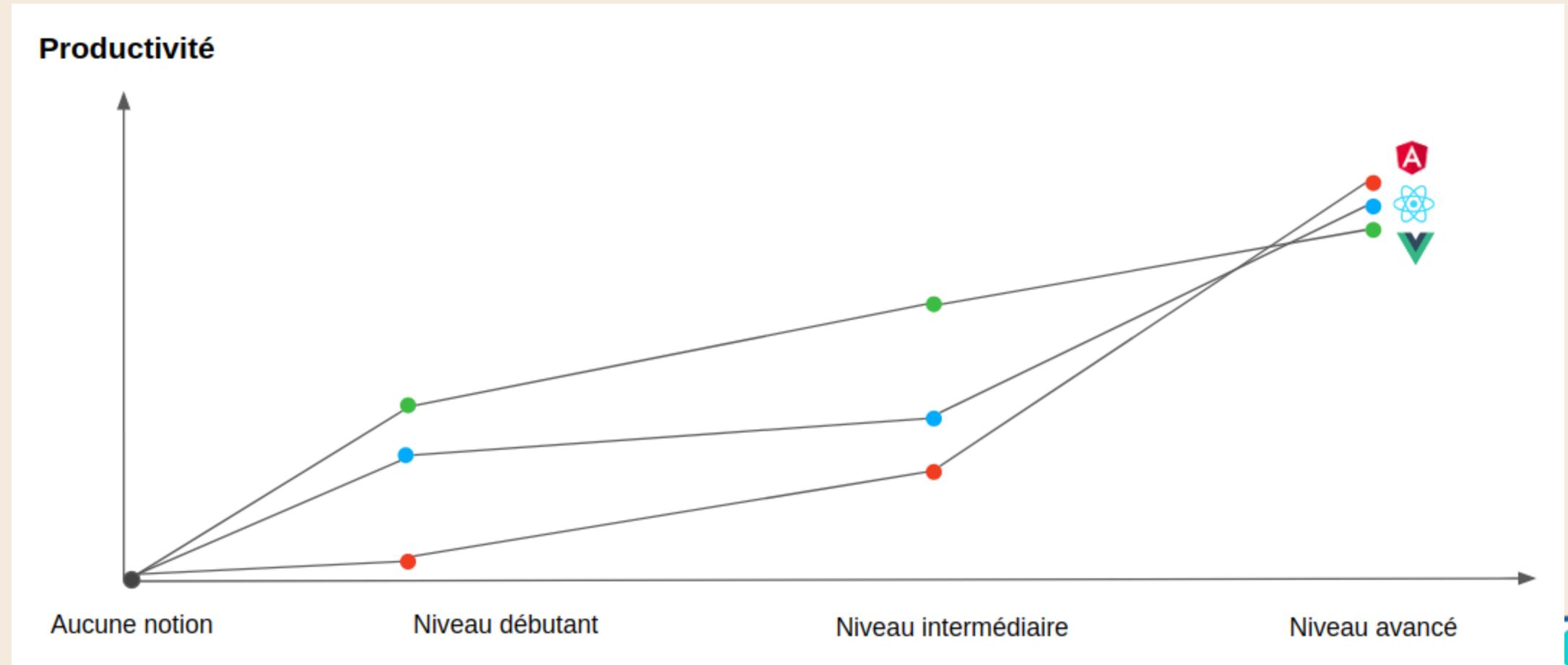
# Comparaison



	React JS	Angular	Vue JS
Ecosystème	Modulaire, Dynamique	Robuste, Structuré	Simple, Léger
Courbe d'apprentissage	Graduelle	Prononcée	Douce
Performances	Rapides	Acceptables	Bonnes
Adoption	Très élevée	Élevée	En croissance
Flexibilité	Elevée	Moyenne	Élevée
Communauté	Large et solide	Grande et soutenue	En augmentation



# Détail: Productivité & maintenabilité





# Vue.js, la philosophie



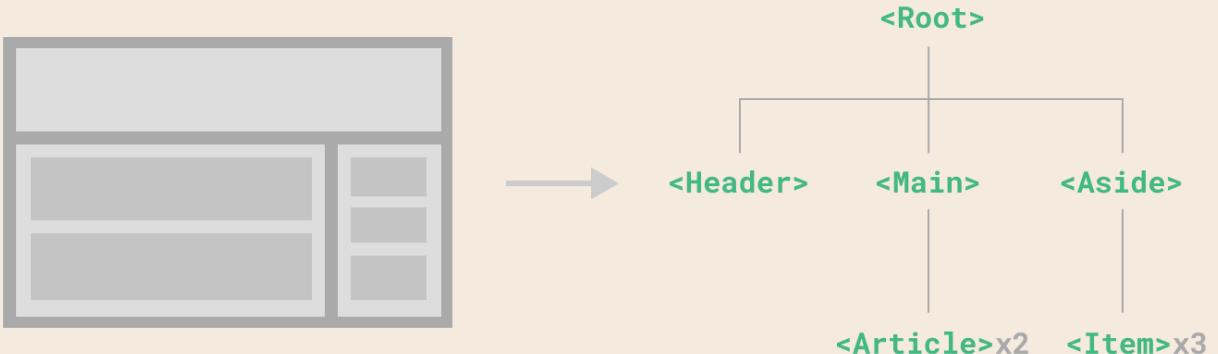
= BUROTIX 0

# Philosophie

- Framework progressif
  - intégration progressive dans un projet
- Approche déclarative
  - on décrit "ce qu'on veut faire" (et non "comment le faire")
- Caractéristiques des composants Vue.js :
  - Encapsulation : HTML, CSS, JS dans un même fichier
  - Communication entre composants
    - via des **props** et **événements**



# Structuration



- Imaginez un interface comme une maison en Lego®
  - Chaque pièce est un composant : (cuisine, salon, chambre) = (header, sidebar, carte).
  - On réorganise les pièces ou on ajoute de nouvelles pièces sans tout casser.
  - Si vous changez la couleur du salon, le reste de la maison reste intact !



= BUROTIX 0

# Cas d'usage concrets

- Quand utiliser Vue.js ?
  - sites vitrines
  - applications légères
  - prototypes rapides
- Exemples
  - Alibaba
  - GitLab
  - Nintendo



# Vue.js : interprété ou compilé ?

## interprété

- **MonModule.js**
  - fichier JavaScript standard
  - CDN
- simple et léger
  - navigateur suffisant
- moins intégré
  - style non intégré
  - template complexe non intégré
- moins lisible

## compilé

- **MonModule.vue**
  - Single File Component (SFC)
- complexe
  - bundler nécessaire
  - Vite, Webpack, Vue CLI
- intégré
  - template + script + style
- lisible



# Vue.js : interprété ou compilé ?

- Par la suite, la **version interprétée** sera présentée.
- La version compilée sera abordée par l'étudiant de son propre chef.



# Vue.js "statique", en pratique

approche progressive



= BUROTIX 0

# exo 11 : version HTML pure

- exo11\_pure\_html.html
  - examinez ce code
- <body> divisé en 3 blocs
  - header
  - main
  - footer
- <main> divisé en 3 blocs
  - titre
  - deux articles de contenu

**Mon Site Web**

Accueil À propos Contact

Bienvenue sur notre site !

**Article 1**

Un article de journal.

**Article 2**

Un autre exemple de contenu.

© 2025 - Tous droits réservés



= BUROTIX 0

# exo 13 : version Vue.js statique

- exo13.html
- Le framework Vue.js est intégré par un lien CDN dans `<head>`

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js">
```



# exo 13 : version Vue.js statique

- <body> contient un attribut **id** et un code (non-html) qui place les **composants personnalisés vue.js** sur l'écran :

```
<body id="app">
  <header-component></header-component>
  <main-component></main-component>
  <footer-component></footer-component>
</body>
```
- **composants personnalisés**
  - définis par le développeur
  - réutilisables
  - encapsulation de HTML, CSS et JavaScript



# exo 13 : version Vue.js statique

- <head> contient un script lancé après le chargement de la page :

```
document.addEventListener(  
  'DOMContentLoaded', () => {  
    // définition et montage  
    // des composants vue.js  
  } )
```



# exo 13 : version Vue.js statique

- Ce script définit d'abord les composants `Vue.js`, ici sous forme d'objets JavaScript :

```
const FooterComponent = {  
    template: `<footer class="footer">  
        <p>&copy; 2025 - Tous droits réservés</p>  
    </footer>  
`};
```

- Il y a plusieurs manières de définir les composants.



# exo 13 : version Vue.js statique

- Ensuite, ce script crée et "monte" l'application **Vue.js** en associant les composants aux variables.

```
Vue.createApp({  
  components: {  
    'header-component': HeaderComponent,  
    'main-component': MainComponent,  
    'footer-component': FooterComponent  
  }  
}).mount('#app');
```

nom du  
composant  
(cf HTML)

nom de l'objet  
(cf JavaScript)

- On dit à **Vue.js** : "Quand tu vois la balise <footer-component> dans le template, utilise le composant défini par l'objet **FooterComponent**."



# exo 13 : version Vue.js statique

- Attention à la syntaxe :
  - kebab-case
    - dans les templates HTML
    - ex: **header-component**
  - PascalCase
    - dans le code en JavaScript
    - ex:  **HeaderComponent**

# exo 15 : Vue.js, statique, multi-composants

- exo15.html
- exo15-app.js
- dir. exo15-components/
  - Header.js
  - Main.js
  - Footer.js

- Vue.js est intégré par une "import-map" (optionnel)

```
<script type="importmap">
{
  "imports": {
    "vue": "...vue.global.js"
  }
}</script>
```



# exo 15 : Vue.js, statique, multi-composants

- inchangé par rapport à exo 13
- <body> contient un attribut **id** et un code (non-html) qui place les **composants personnalisés** Vue.js sur l'écran :

```
<body id="app">
  <header-component></header-component>
  <main-component></main-component>
  <footer-component></footer-component>
</body>
```



# exo 15 : Vue.js, statique, multi-composants

- <head> contient un script lancé après le chargement de la page, cette fois-ci dans un fichier séparé :

```
<script  
  type="module"  
  src="../exo15-app.js"  
  defer>  
</script>
```



# exo 15 : Vue.js, statique, multi-composants

- Ce script importe les composants Vue.js.
- Chaque composant est défini dans son propre fichier.

```
import FooterComponent  
from './exo15-components/Footer.js'
```



# exo 15 : Vue.js, statique, multi-composants

- inchangé par rapport à exo 13
- Ensuite, ce script crée et "monte" l'application **Vue.js** en associant les composants aux variables.

```
Vue.createApp({  
    components: {  
        'header-component': HeaderComponent,  
        'main-component': MainComponent,  
        'footer-component': FooterComponent  
    }  
}).mount('#app');
```

nom du  
composant  
(cf HTML)

nom de l'objet  
(cf JavaScript)



# exo 15 : Vue.js, statique, multi-composants

- Chaque composant Vue.js est défini comme un objet JavaScript dans un fichier propre, avec la syntaxe suivante :

```
export default {  
    name: 'FooterComponent', // Nom composant  
    template :`  
        <footer class="footer">  
            <p>&copy; 2025 - ...</p>  
        </footer>  
    `,  
}
```



# Vue.js "dynamique", en pratique

un tour des lieux



# exo 25 : Vue.js, dynamique, multi-composants

**Mon Beau Site Web**

Mon Beau Site Web Mon Beau Site Web Mon Beau Site Web

Accueil À propos Contact

Bienvenue sur notre site !

3 articles

**Article 1**

Un article de journal.  
bla bla blahhh

[cliquez-moi](#)

**Article 2**

Un autre exemple de contenu.  
bla bla blahhh

[cliquez-moi](#)

**Article 3**

Encore un autre exemple de contenu sans intérêt.  
bla bla blahhh

[cliquez-moi](#)

Nombre d'articles listés : 3

© 2025. Tous droits réservés.

You are a clown !

[OK](#)

Bienvenue sur notre site !

3 articles

**Article 1**

Un article de journal.  
bla bla blahhh

[cliquez-moi](#)

**Article 2**

Un autre exemple de contenu.  
bla bla blahhh

[cliquez-moi](#)

**Article 3**

Encore un autre exemple de contenu sans intérêt.  
bla bla blahhh

[cliquez-moi](#)

Nombre d'articles listés : 3

© 2025. Tous droits réservés.



= BUROTIX 0

# concepts de base

- name
- template
- data()
- props
- computed
- methods
- v-for, v-if
- v-bind
- v-show
- @mouseover,  
@mouseleave



# HTML : exo25.html

```
<body id="app">  
  <header-component  
    title="Mon Site"  
    :verbose_b="verbose"  
  >  
  </header-component>  
  <main-component><...>  
  <footer-component><...>  
</body>
```

- La page HTML envoie des paramètres aux composants.
- Paramètre de valeur hard-codée
  - **title="Mon Site"**
  - le composant **header** reçoit une propriété **title** de valeur "**Mon Site**"
- Paramètre de valeur calculée
  - **:verbose\_b="verbose"**
  - le composant **header** reçoit une propriété **verbose\_b** dont la valeur se trouve dans l'application sous le nom **verbose**



# App : exo25.js

```
createApp({  
  components: { ... },  
  data() {  
    return {  
      verbose:true,  
    }  
  }  
}).mount('#app');
```

- L'app définit des composants (cf exos précédents).
- L'app définit des variables que les composants peuvent utiliser
  - **verbose : true**
- L'app se "monte" sur un élément HTML (cf exos précédents).



# Composant : Header.js

```
export default {  
  name: ...  
  template : `...`,  
  props : {  
    title : {  
      type : String,  
      required : true ,  
    },  
    verbose_b : {  
      type : Boolean,  
      required : false,  
      default : false,  
    },  
  },  
}
```

## ■ Fonctionnalités

- définir un composant par son **name** et son **template**
  - cf exos précédents
- déclarer les variables fournies par le parent
  - **props**
    - **type** : Boolean, String, etc.
    - **required** : true, false
    - **default** : default value



# Composant : Header.js

template :

```
<header class="header">  
  <h1 v-bind:class="theme">  
    {{ title }}  
  </h1>  
  <h2 v-if="this.verbose_b">  
    {{ title }}{{ title }}  
  </h2>  
  <h2 v-else>{{ title }}</h2>  
  <nav>...</nav>  
</header>
```

## ■ Fonctionnalités

- utiliser une variable fournie par le parent
  - déclarée dans **props**
  - **{{ title }}**
    - var **title** de **props**
  - **v-bind:class="theme"**
    - var **theme** de **props**
- structure de contrôle
  - **v-if="this.verbose\_b"**
    - var **verbose\_b** de **props**
  - **v-else**



# Composant : Main.js

```
data() {  
  return {  
    title : "Bienvenue !",  
    contents : [  
      {  
        id : 1001,  
        title : "Article 1",  
        body : "Un article",  
        more : "bla bla",  
      },  
      { ... },  
    ],  
  },  
},  
};  
},
```

## ■ Fonctionnalités

- déclarer les variables définies en interne

- `data()` {  
 return <JSON str.>  
 },

- `<JSON structure>` peut contenir tous les types de variable : simple, complexe, liste, dictionnaire, integer, string, boolean, etc.



# Composant : Main.js

```
template : `<main class="main">
  <h2>{{ title }}</h2>
  <section v-for="c in contents">
    <h3>{{ c.title }}</h3>
    <p v-show="this.verbose_b" >
      {{ c.more }}
    </p>
  </section>
</main>
`,
```

- Fonctionnalités :
  - structure de contrôle
    - **v-for="c in contents"**
      - var **contents** de **data()**
      - var **c** pour l'itération
    - montrer / cacher
      - **v-show="this.verbose\_b"**
        - var **verbose\_b** de **props**



# Composant : Main.js

```
template :`  
  <div  
    @mouseover = "HoveredId = content.id;"  
    @mouseout = "HoveredId = -1"  
    :class = " HoveredId == content.id  
              ? 'card_mouseover'  
              : 'card_mouseout' " >  
  </div>  
`,  
  
data() {  
  return {  
    HoveredId: -1,  
  }  
},
```

- Fonctionnalités :
- gestion des évènements
  - `@mouseover`
  - `@mouseout`
- implantation d'une logique
  - `HoveredId`
- impact éventuel sur d'autres éléments
  - `class`



# Composant : Main.js

```
template : `

<h3>{{ contents.length }} art.</h3>
<p> {{ nombreArticles }} art.</p>
<button @click="popup" >click</button>
`,

computed: {
  nombreArticles() {
    return this.contents.length;
  }
},
methods: {
  popup() {
    alert("You are a clown !")
  }
},
```

## ■ Fonctionnalités

- utiliser les attributs des variables dans le template
  - **contents.length**
- propriété calculée
  - valeur actualisée en temps réel
  - **computed**
  - **nombreArticles**
  - **return**
- méthode
  - notamment, le code appelé lors d'un évènement (click, ...).
  - **methods**
  - **popup()**



## Exo 26 : drill

- Reprendre l'exo 25.
- Implantez les fonctionnalités suivantes dans **Footer.js**
  - Quand on clique sur le bouton "cliquez-moi" d'un article, ses méta-données apparaissent.
  - Choix de la couleur du texte : bleu ou noir
    - en passant l'objet **event** à la fonction JS
    - alternative : en utilisant **v-model**
- Seul **Footer.js** est modifié



# concepts avancés (non vus au cours)

- provide()
- watch
- emits
- setup()
- inject
- v-model

