Web Client - **#2**

[Vue.js - les bases]

Cyril Rouyer

https://vuejs.org

Sommaire

- MVC Client
- Vue.js (v3) - les grands principes
- Syntaxe de base
- Requêtes HTTP

MVC Client

MVC Client 1

Single Page Interface

- Conception “classique” : une page = une url, 1 clic et on passe à une autre page

- SPI : 1 seule URL pour toute l’application

- Toutes les Interactions Utilisateurs sont traitées en Javascript

- Aucun rafraîchissement de page

- MVC quand même ?

MVC Client 2

Modèles Vues Contrôleurs

- M : les données, (ex : stockées de manière persistante dans une BDD distribuées par un WebService, météo, tweets, API Métier),

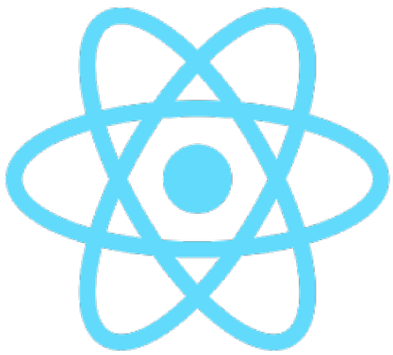
- V : vues dessinées en HTML5, CSS3,

- C : Récupèrent les événements en JS (click, change, hover, mouseenter, focus, etc…), s'occupent des interactions utilisateurs

MVC Client 5

Frameworks

ANGULARJS by Google



React

BACKBONE.JS

Vue.js

Les grands principes

Vue.js 1

Présentation

- Permet les applications hybrides
- Ultra performant
- Courbe d’apprentissage progressive
- Logique de composants WEB (
<**todo**>1</**todo**><**todo**>2</**todo**>)
- DOM Virtuel / Update dynamique basé sur la réactivité

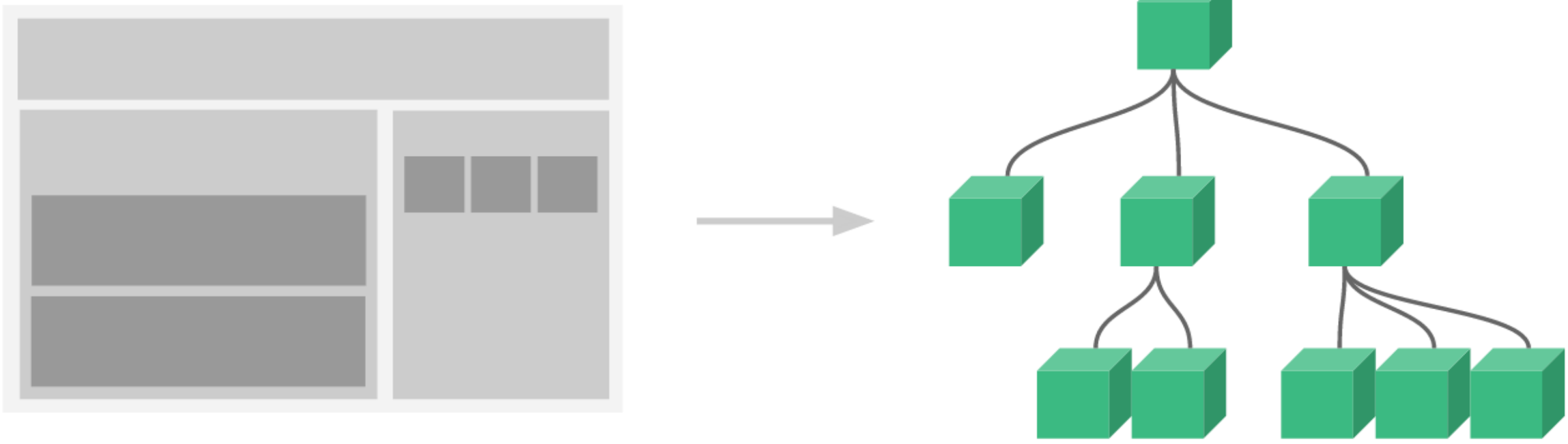
Vue.js 2

Prérequis (pour les TDs)

- EcmaScript 6
- Node / NPM
- vue-cli
- Permet de suivre le guide et les tutos en ligne
- Permet de fonctionner avec les fichiers .vue qui décomposent les trois aspects fondamentaux de chaque composant : template / interactions / styles.

Vue.js 3

Composants



```
Vue.component('todo-item', {
  template: '<li>Ceci est un élément de liste</li>'
})
```

https://fr.vuejs.org/v2/guide/index.html

Vue.js 4

Réactivité

- L’instance de votre Vue peut prendre des datas dans ses propriétés

- data est un objet

- Chaque propriété de data est réactive = si on change la donnée en JS l’affichage s’adaptera automatiquement ; si on change sa valeur dans l’affichage (input), la propriété prendra la nouvelle valeur automatiquement

Vue.js 5

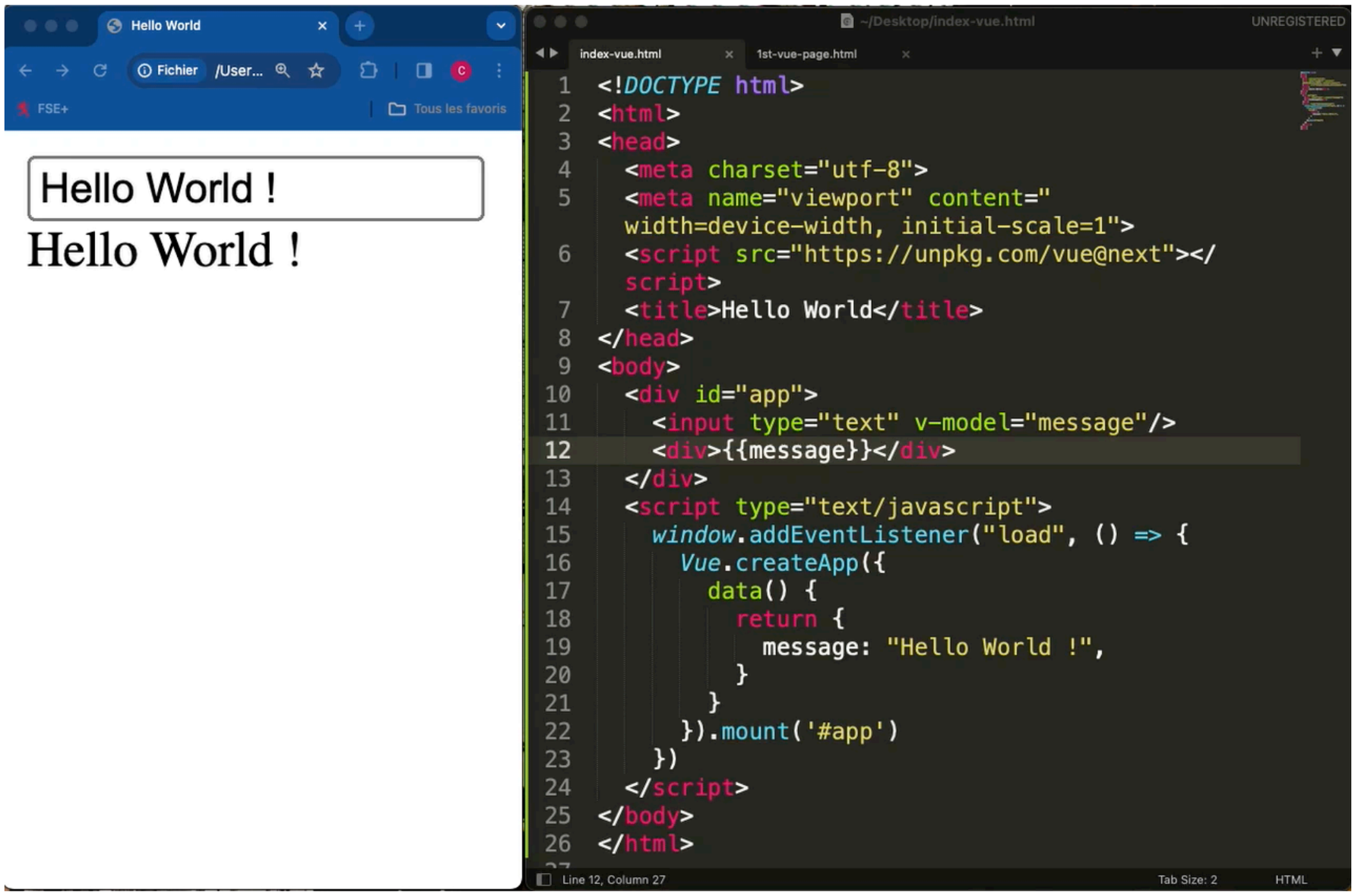
Exemple basique

```
<script src=“https://
unpkg.com/vue@latest"></
script>
…
<div id="app">
  {{ message }}
</div>
```

```
Vue.createApp({
  data() {
    return {
      message: 'Hello World !’
    }
  }
}).mount(‘#app’)
```

Vue.js 6

<-> Réactivité



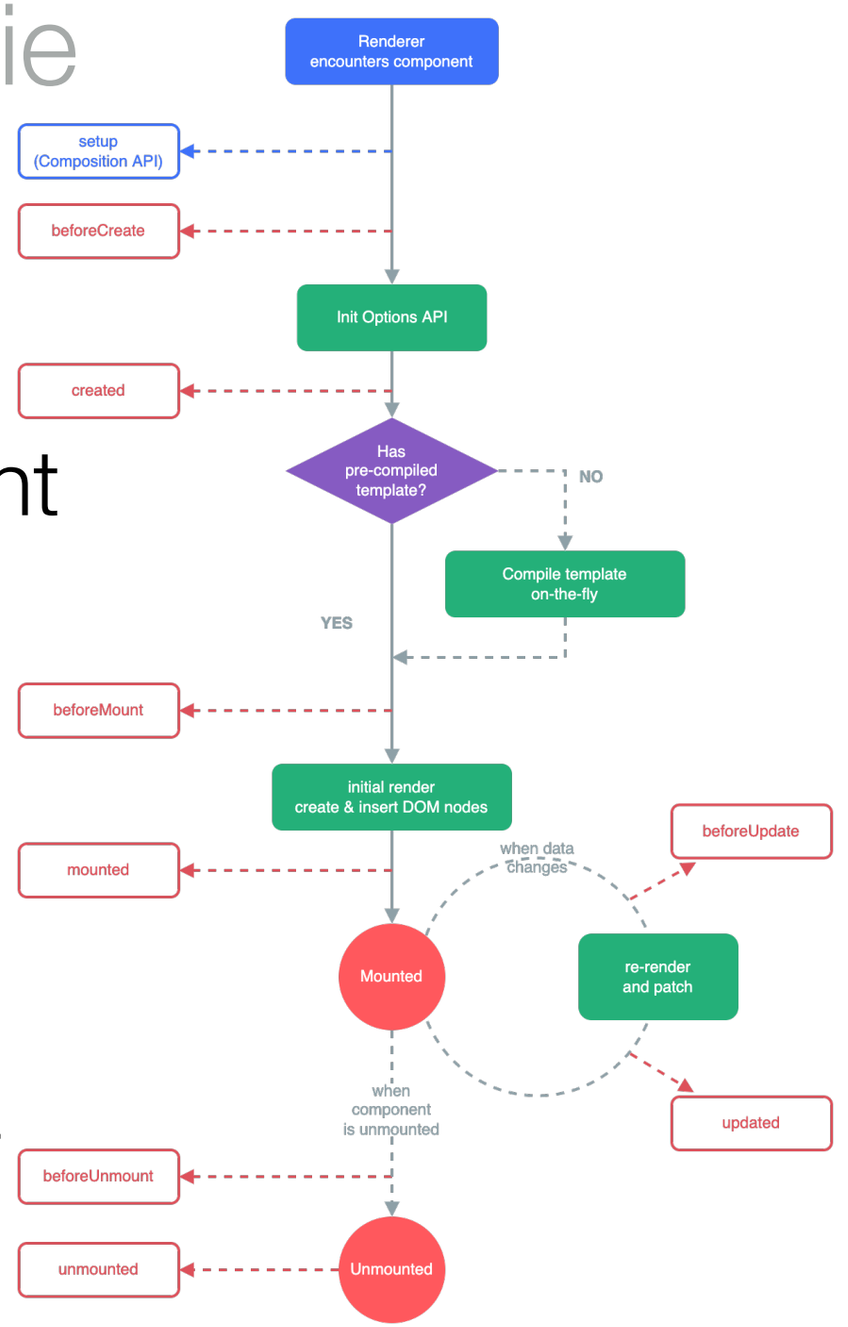
Vue.js 7

Hooks / Cycle de vie

- L'initialisation d'une instance de Vue passe par des étapes qui déclenchent chacune des hooks

- beforeCreate, created, mounted, … unmounted

- Vrai aussi pour les composants de la vue



Vue.js 8

Instance Hooks

```
Vue.createApp({
  data() {
    return {
      items: []
    }
  },
  created: function () {
    api.get('/items').then(response => {
        this.items = response.data
    })
  }
}).mount('#app')
```

Syntaxe de base

Syntaxe de base 1

Binding model <-> template

Texte simple

Prénom :{{user.firstname}}

Contenu HTML trusté

Attribut HTML

{{user.fullname}}

Syntaxe de base 2

Binding model <-> template

Booléens dans les attributs

<button v-bind:disabled=“user.actif”>Supprimer</button>

Syntaxe JS supportée (mais limitée - liste blanche de variables globales) au sein des templates

{{number + 1}}

<div v-bind:id=“‘user_’ + user.id”>

Syntaxe de base 3

Directives

- Les directives sont des attributs commençant par v-
- Exemples déjà vus : v-bind, v-html
- Leur rôle est de modifier “réactivement” le DOM en fonction des modèles et des interactions utilisateurs
- Certains prennent un argument : v-bind**:id**=“…”, v-bind**:href**=“…” Les deux points permettent de spécifier l’attribut visé

Syntaxe de base 4

Directives

- Autres exemples qui seront détaillés plus loin dans le cours :
  - v-if : permet de faire du rendu conditionnel
  - v-on : permet de catcher des événements
  - v-for : permet d’afficher les éléments d’une liste

Syntaxe de base 5

Abréviations

- v-bind et v-on sont les deux directives les plus utilisées dans une application SPI
- Vue.js propose deux abréviations pour gagner du temps :
  - v-bind -> “:” / v-on -> “@”
  - <a :href=“user.link”>Go (:href <=> v-bind:href)
  - <button @click=“saveUser()”>Enregistrer</button> (@click <=> v-on:click)

Syntaxe de base 6

Rendu conditionnel

- v-if permet un affichage conditionnel d'un élément
  - …

- Prend une condition booléenne

- <template> permet d'englober plusieurs éléments dans la condition :
- **<template** v-if="company.suscribed">
  {{company.name}}
  {{company.tel}}
  </template>

Syntaxe de base 7

Rendu conditionnel

- v-else-if
- v-else
- Exemple :

```
<div v-if="condition">…</div>

<div v-else-if="autre_condition">…</div>

<div v-else>…</div>
```

Syntaxe de base 8

Rendu de listes

- v-for : permet d'itérer un tableau ou les propriétés d'un objet

```
<div id="app">
  <ul>
    <li v-for="item in items">
     {{item.name}}
    </li>
   </ul>
</div>
```

```
Vue.createApp({
 data () {
   return {
     items: [
       {name: "John"},
       {name: "Lisa"}
     ]
   }
  }
}).mount('#app')
```

Syntaxe de base 9

Rendu de listes

- v-for="(item, index) in items" —> **index** permet d'obtenir la position de l'item au sein du tableau

- v-for="(value, key) in object" —> **key** permet d'obtenir le nom de la propriété de l'objet parcouru

- v-for="(value, key, index) in object" —> **combo**

Syntaxe de base 10

Rendu de listes

- Rendu intelligent des listes et de ses modifications (in-place patch)
- Besoin d’un pivot unique
- <li v-for=“item in items” **:key=“item.id”**>…

Syntaxe de base 11

Rendu de listes

- Réactivité sur les listes :
  - Vue surcharge les méthodes de mutation habituelles [push(), pop(), splice(), shift(), unshift(), sort(), reverse(), filter(), concat(), slice()] pour rendre réactives ses listes
- Avec Vue3 on peut maintenant modifier directement un tableau sans passer par les méthodes citées plus haut :
  - this.items[2] = 42 //est à présent réactif
  - this.items.length +=5 //est à présent réactif

Syntaxe de base 12

Objets

- Modification des propriétés d’objets & réactivité
- l’ajout d’une nouvelle propriété à la racine des data est désormais possible avec Vue3
- On peut ajouter directement une propriété à un sous-élément de la racine (this.myobject.newprop = 3)

Syntaxe de base 13

Événements

- v-on ou @ (abréviation)
- Permet de catcher n’importe quel type d’événement permis par votre navigateur
- @click, @mouseenter, @keyup, …

Syntaxe de base 14

Événements

- Permet d’exécuter directement du code une fois l’événement catché
  - <button @click=“counter += 1”>+ 1</button>
- Mais permet aussi d’appeler une méthode déclarée dans votre Vue
  - <button @click=“addToCart(product)”>Ajouter</button>

Syntaxe de base 15

Événements - point sur les méthodes

```
Vue.createApp({
  data () {
     return { name: ‘Vue.js' }
  },
  // Définissez les méthodes de l'objet
  methods: {
    greet: function (event) {
      // `this` fait référence à l'instance de Vue à l'intérieur de `methods`
      alert('Bonjour ' + this.name + ' !')
      // `event` est l'événement natif du DOM
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
}).mount(‘#example-2')
```

Syntaxe de base 16

Événements

- On peut injecter l'événement lui-même dans l'appel de la méthode avec $event en dernière position

```
<button v-on:click="warn('ceci est un message de warning.', $event)">
  Soumettre
</button>
```

```
// ...
methods: {
  warn: function (message, event) {
    if (event) event.preventDefault()
     alert(message)
  }
}
```

Syntaxe de base 17

Modificateurs

- Les modificateurs sont des raccourcis dans la déclaration du listener qui vous font gagner du temps. Exemples :
  - @click**.stop**=“…” : fera un stopPropagation pour vous
  - @click**.prevent**=“…” : fera un preventDefault
  - @click**.capture**=“…” : permet de capturer l’event depuis les parents jusqu’à l’élément cliqué
  - @click**.self**=“…” : déclenché uniquement si l’élément cliqué est celui-ci, et pas un de ses fils
  - @click**.once**=“…” : l’événement ne sera déclenché qu’une seule fois

Syntaxe de base 18

Modificateurs

- On peut les chaîner :
  - <div @click**.stop.prevent**=“doSomething()”>…

Syntaxe de base 19

Modificateurs

- **Modificateurs de touches**
  - <input type=“text” @keyup.13=“submit()”/>
  - .enter, .tab, .delete, .esc, .space, .up, .down, .left, .right
  - Personnalisation : Vue.config.keyCodes.f1 = 112
  - Touches “système” : .ctrl, .alt, .shift, .meta
  - .exact : permet de s’assurer que la touche est la bonne :
    - <button @click.ctrl.exact=“onCtrlClick”>…</button>

Syntaxe de base 20

Modificateurs

- **Modificateurs pour les clics souris**
  - .left,
  - .right,
  - .middle

Syntaxe de base 21

Binding & Formulaires

- **v-model** permet de lier un modèle (data) à un champ (input, select, textearea)

```
Vue.createApp({
  data() {
    return {
      message: 'Hello Vue !',
      checked: false,
      selected: [],
      bigtext: ‘’
    }
  }
}).mount(‘#app’)
```

```
<template>
  <input v-model=“message”/>
  <input type=“checkbox” v-model=“checked”
         true-value=“A”
         false-value=“B”/>
  <select v-model=“selected” multiple>
    <option value=“A”>A</option>
    <option value=“B”>B</option>
    <option value=“C”>A</option>
  </select>
  <textarea v-model=“bigtext”></textarea>
</template>
```

Syntaxe de base 22

Binding & Formulaires

- **Modificateurs :**
  - .lazy : permet de synchroniser les données au “change” du navigateur plutôt qu’à chaque input utilisateur
  - .number : permet de forcer Vue à récupérer un nombre et pas une chaîne de caractères
  - .trim : permet de supprimer les espaces superflus

Syntaxe de base 23

Computed

- **Computed :**
  - Une des propriétés possibles de votre instance Vue.js
  - C’est un objet
  - Chaque propriété est une fonction permettant de retourner une valeur calculée qui pourra être utilisée directement dans le template

Syntaxe de base 24

Computed

```
<div id="example">
  <p>Message original : "{{ message }}"</p>
  <p>Message inversé calculé : "{{ reversedMessage }}"</p>
</div>

var vm = new Vue({
  el: '#example',
  data: {
    message: 'Bonjour'
  },
  computed: {
   reversedMessage: function () {
     return this.message.split('').reverse().join('')
    }
  }
})
```

Syntaxe de base 25

Computed vs Methods

- Les méthodes permettent aussi de réaliser ce type de calculs

- Quelle différence entre les computed et les methods ?

- Si un template appelle directement une méthode pour afficher une valeur, la méthode sera exécutée à chaque event loop, même si rien n’a changé

- Avec les computed, Vue.js détecte les changements sur les dépendances et ne rafraîchit la vue que si c’est nécessaire

Requêtes HTTP

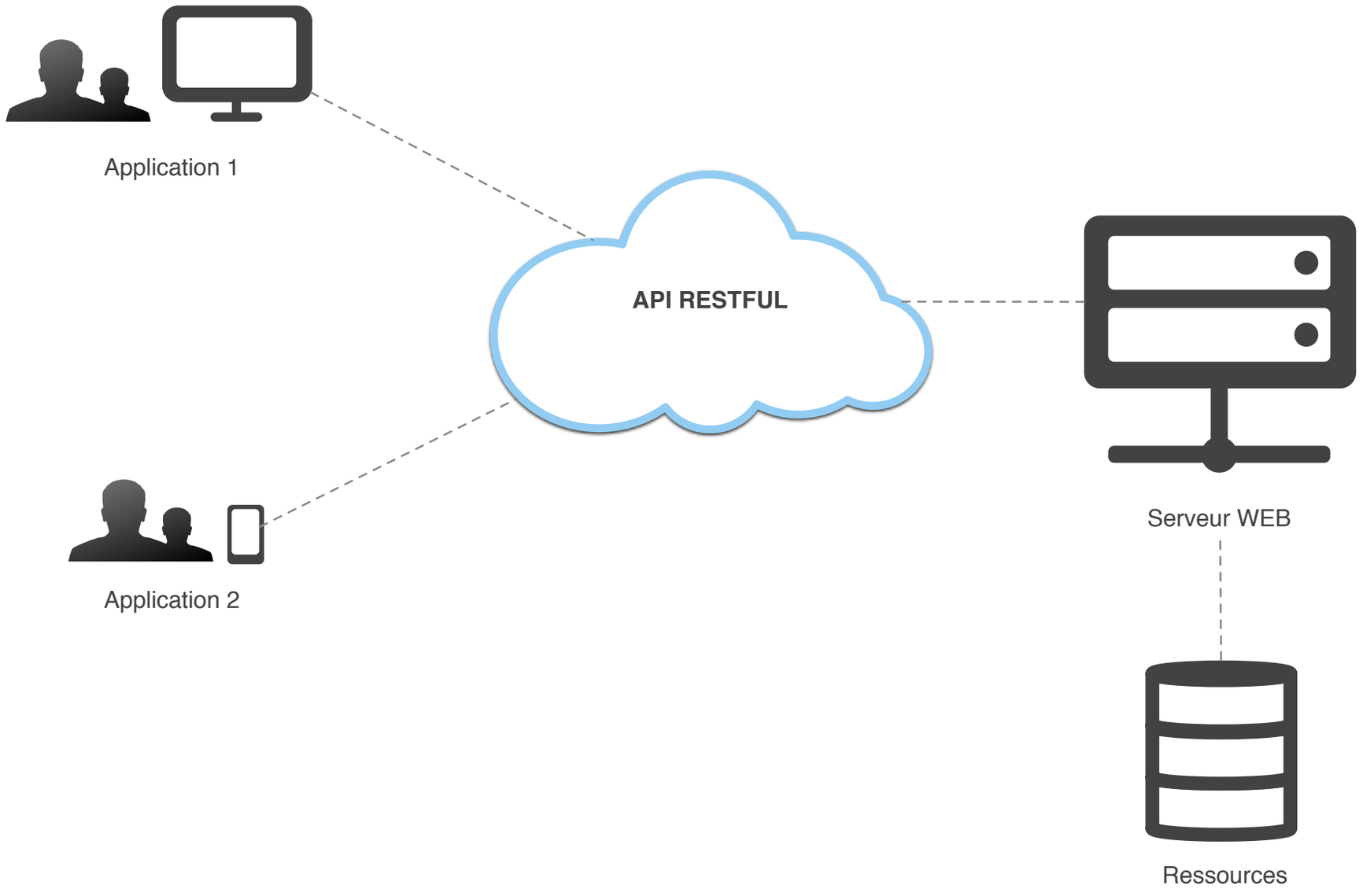
REST 1

Une architecture Backend

- = une architecture : ni un protocole ni un format de données
- Sépare les données et leur traitement de l'affichage
- les données sont des ressources
- Chaque requête du client se suffit à elle-même pour que le serveur comprenne ce qui est demandé (sans stockage « contextuel » sur le serveur - ex : pas de session).
- On parle d' « API REST » côté Backend quand le serveur publie des routes « RESTFUL » permettant de répondre aux besoins d'une ou plusieurs applications Frontend.

REST 2

REST - Representational State Transfer

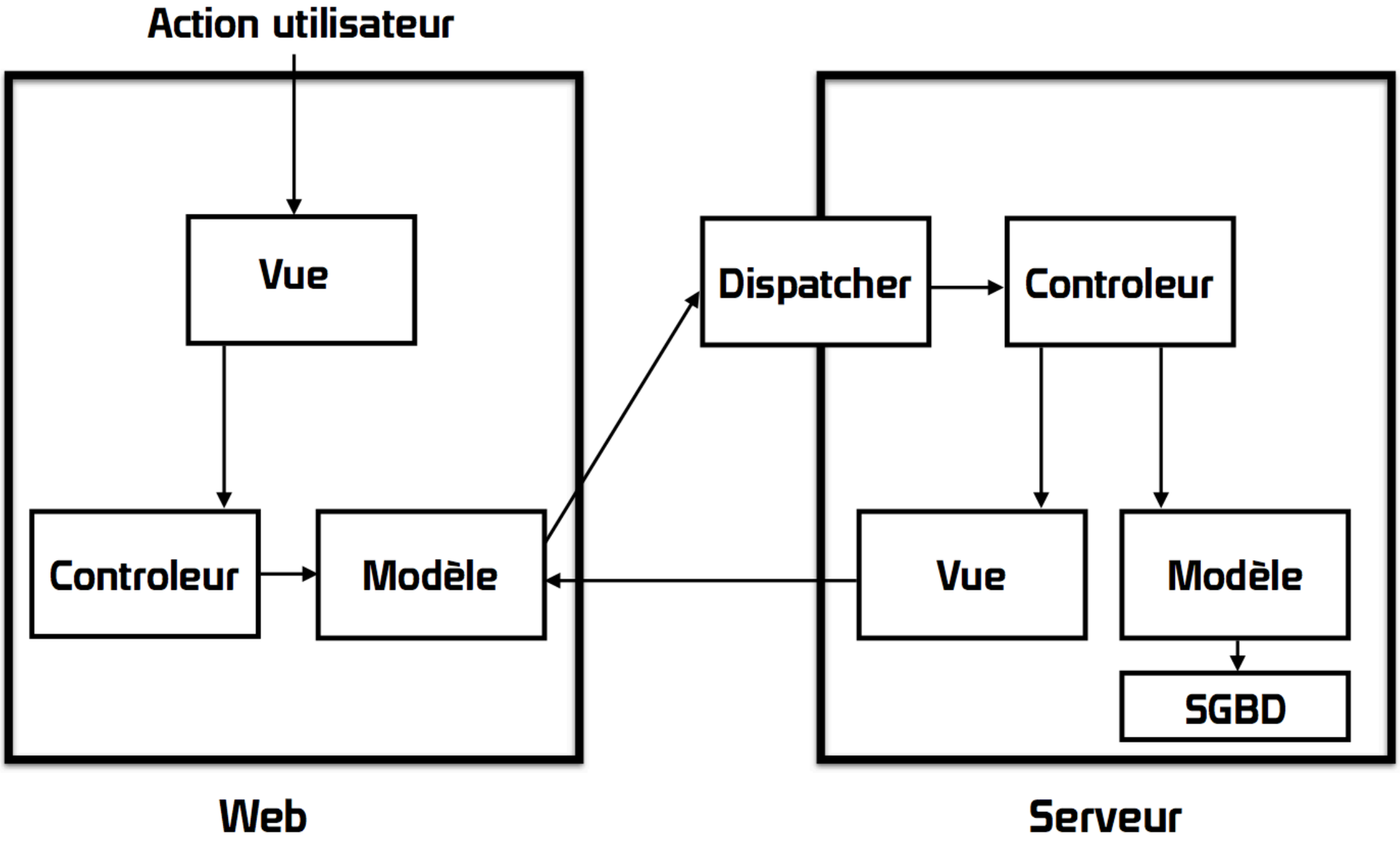


REST 3

Des routes et non des pages

- GET /albums
- POST /albums
- GET /albums/1
- PUT /albums/1
- PATCH /albums/1/label
- DELETE /albums/1/tracks/4

FrontEnd / Backend



Vue.js & requêtes HTTP

- Pas d’outil natif permettant de réaliser des requêtes Ajax

- S’occupe uniquement de la Vue et des interactions avec l’utilisateur

- Vue.js recommande d’utiliser Axios (https://github.com/axios/axios)

Axios 1

- Librairie qui s'occupe uniquement de la partie HTTP
- Installation facile avec npm : npm install axios
- Librairie basée sur les promesses. Permet d'utiliser le .then((data) {…}) en cas de succès et le .catch((error){…}) en cas d'erreur

Axios 2

- Exemples d’appels :
- axios.get(url, {params: {ID: 34, order: ‘name’}})
- axios.post(url, {name: ‘John’, age: 56})
- axios.put(…)
- axios.delete(…)
- Tous les verbes HTTP sont gérés

Axios 3

- Permet de créer des instances différentes et de configurer précisément chaque instance :

```
const basicConfig = {
  baseURL: conf.url,
  headers: {
    'Accept': 'application/json;version=1;'
    'Authorization': conf.apiKey
   },
  params: {} // Prevents null check in interceptors
}

export const api = axios.create(basicConfig)
export const signin = axios.create({baseURL: conf.url})
```