# Vue.js : Axios

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

# Plan

# Vue.js

## **HTTP** : **H**yper**t**ext **T**ransfer **P**rotocol

- Protocole de communication entre client et serveur
- Basé sur la notion requête - réponse appelée généralement (HTTP Request - HTTP Response)
- Plusieurs types de requêtes = méthodes ou verbes **HTTP**
  - `GET`
  - `POST`
  - `DELETE`
  - `PUT`
- Toutes ces méthodes prennent en paramètre l'**URL** de la ressource (+ pour certaines méthodes les données à manipuler)

# Vue.js

## **Vue.js** et les serveurs

- **Vue.js** : Framework **JavaScript** qui permet de réaliser des applications **Web** qui s'exécutent coté client
- **Axios** : module **Vue.js** facilitant la réalisation de requête **HTTP** vers n'importe quel serveur via les classes suivantes :
- Pour le coté serveur : on peut utiliser un serveur **JSON** qui recevra les requêtes **HTTP** envoyées par **Axios** et retourner une réponse

# Vue.js

## json-server

- Serveur **HTTP** de test pour les développeurs Front-End
- Open-source
- Utilisant par défaut le port 3000
- Documentation : `https://github.com/typicode/json-server`

# Vue.js

## json-server

- Serveur **HTTP** de test pour les développeurs Front-End
- Open-source
- Utilisant par défaut le port 3000
- Documentation : `https://github.com/typicode/json-server`



### Pour l'installer

```
npm install -g json-server
```

# Vue.js

**Créons un fichier `db.json`, à la racine du projet, qui va nous servir de base de données**

```
{
    "personnes": [
        {
            "age": 45,
            "nom": "Wick",
            "prenom": "John",
            "id": 1
        },
        {
            "age": 40,
            "nom": "Dalton",
            "prenom": "Jack",
            "id": 2
        }
    ]
}
```

# Vue.js

**Pour pouvoir envoyer des requêtes HTTP, il faut démarrer json-server sur le port `5555` en précisant le nom de la base de données (`db.json`)**

```
json-server -p 5555 db.json
```

# Vue.js

**Pour pouvoir envoyer des requêtes HTTP, il faut démarrer json-server sur le port `5555` en précisant le nom de la base de données (`db.json`)**

```
json-server -p 5555 db.json
```

**Résultat (parmi toutes les lignes affichées)**

```
Resources
http://localhost:5555/personnes
```

# Vue.js

## http://localhost:5555/personnes

- C'est l'URL qui sera utilisée par le client pour réaliser des requêtes **HTTP**
- Si on copie cette adresse et qu'on la colle dans le navigateur, on obtient toutes les données de notre base de données

# Vue.js

## Explication

- Pour récupérer la liste de toutes les personnes
  `GET : http://localhost:5555/personnes`
- Pour récupérer une personne selon l'identifiant (33 par exemple)
  `GET : http://localhost:5555/personnes/33`
- Pour ajouter une nouvelle personne
  `POST : http://localhost:5555/personnes`
- Pour modifier les valeurs d'une personne existante (ayant l'identifiant 33)
  `PUT : http://localhost:5555/personnes/33`
- Pour supprimer une personne existante (ayant l'identifiant 33)
  `DELETE : http://localhost:5555/personnes/33`

# Vue.js

**Pour installer** `axios`

```
npm install axios
```

# Vue.js

**Pour installer** `axios`

```
npm install axios
```

**Pour utiliser** `axios`**, il faut l'importer**

```
<script>
import axios from 'axios'

export default { }
</script>
```

# Vue.js

## Objectif

Dans `PersonneShowView`

- Suppression du contenu du tableau `personnes`
- Utilisation d'**Axios** pour la récupération et l'affichage des données gérées par le serveur **json-server**

# Vue.js

**Dans `PersonneShowView`, commençons par supprimer le contenu du tableau `personnes`**

```
<template>
    <h1>Gestion de personnes</h1>
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            <router-link :to="{ name: 'personne-details', params: { id: elt.id } }">
                {{ elt.nom }} {{ elt.prenom }}
            </router-link>
        </li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    }
}
</script>
```

# Vue.js

**Ensuite, ajoutons l'import d'`Axios`**

```
<template>
    <h1>Gestion de personnes</h1>
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            <router-link :to="{ name: 'personne-details', params: { id: elt.id } }">
                {{ elt.nom }} {{ elt.prenom }}
            </router-link>
        </li>
    </ul>
</template>

<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    }
}
</script>
```

# Vue.js

**Utilisons maintenant Axios pour envoyer une requête HTTP de type `GET` pour récupérer la liste de personnes**

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    },
    mounted() {
        axios({
            method: 'GET',
            url: 'http://localhost:5555/personnes',
        })
            .then(response => this.personnes = response.data)
    }
}
</script>
```

# Vue.js

**Utilisons maintenant Axios pour envoyer une requête HTTP de type `GET` pour récupérer la liste de personnes**

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    },
    mounted() {
        axios({
            method: 'GET',
            url: 'http://localhost:5555/personnes',
        })
            .then(response => this.personnes = response.data)
    }
}
</script>
```

Vérifiez dans la console du navigateur que l'objet `response` affiché contient une clé `data` dont la valeur est un tableau de personnes.

# Vue.js

**Nous pouvons aussi utiliser le raccourci** `axios.get()`

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    },
    mounted() {
        axios
            .get('http://localhost:5555/personnes')
            .then(response => console.log(response))
    }
}
</script>
```

## Récupérons les données de l'objet `response`

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    },
    mounted() {
        axios
            .get('http://localhost:5555/personnes')
            .then(response => this.personnes = response.data)
    }
}
</script>
```

## Récupérons les données de l'objet `response`

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: []
        }
    },
    mounted() {
        axios
            .get('http://localhost:5555/personnes')
            .then(response => this.personnes = response.data)
    }
}
</script>
```

Vérifiez que la liste de personnes s'affiche dans la page.

# Vue.js

**Dans le `script` de `PersonneAdd`, modifions la méthode `ajouterPersonne()` afin de permettre d'ajouter les valeurs saisies dans la base de données**

```
ajouterPersonne(values) {
    axios
        .post('http://localhost:5555/personnes', values)
        .then(() => {
            this.personne = {}
        })
},
```

# Vue.js

**Dans le `script` de `PersonneAdd`, modifions la méthode `ajouterPersonne()` afin de permettre d'ajouter les valeurs saisies dans la base de données**

```
ajouterPersonne(values) {
    axios
        .post('http://localhost:5555/personnes', values)
        .then(() => {
            this.personne = {}
        })
},
```



**N'oublions pas d'importer `axios`**

```
import axios from 'axios'
```

# Vue.js

## Si on essaye d'ajouter une personne

- La personne est ajoutée dans le fichier `personnes.json`
- Mais elle n'est pas affichée dans la page qu'après actualisation
- Car on ne met pas à jour la liste des personnes

# Vue.js

## Si on essaye d'ajouter une personne

- La personne est ajoutée dans le fichier `personnes.json`
- Mais elle n'est pas affichée dans la page qu'après actualisation
- Car on ne met pas à jour la liste des personnes

## Deux solutions possibles

- Soit on ajoute la personne au tableau `personnes` du composant `PersonneShowView` lorsqu'on l'ajoute dans `db.json`
- Soit on refait une requête **HTTP** de type `GET` pour récupérer la nouvelle liste `personnes`

# Vue.js

**Modifions la méthode `ajouterPersonne()` pour qu'elle émette un évènement au composant parent `PersonneShowView`**

```
ajouterPersonne(values) {
    axios
        .post('http://localhost:5555/personnes', values)
        .then((reponse) => {
            console.log(reponse)
            this.personne = {}
            this.$emit('sendData', reponse.data)
        })
},
```

# Vue.js

**Modifions le** `template` **de** `PersonneShowView` **pour intercepter l'évènement du composant enfant** `PersonneAdd`

```
<template>
    <h1>Gestion de personnes</h1>
    <PersonneAdd @send-data="ajouterDansListe"/>
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            <router-link :to="{ name: 'personne-details', params: { id: elt.id } }">
                {{ elt.nom }} {{ elt.prenom }}
            </router-link>
        </li>
    </ul>
</template>
```

# Vue.js

**Dans le `script` de `PersonneShowView`, définissons la méthode `ajouterDansListe`**

```
methods: {
    ajouterDansListe(values) {
        this.personnes.push(values)
    }
}
```

# Vue.js

**Dans le `script` de `PersonneShowView`, définissons la méthode `ajouterDansListe`**

```
methods: {
    ajouterDansListe(values) {
        this.personnes.push(values)
    }
}
```



Vérifiez qu'en ajoutant une nouvelle personne, cette dernière est immédiatement affichée dans la liste.

# Vue.js

**Considérons le contenu suivant pour le `template` de `PersonneShwoView`**

```
<template>
    <h1>Gestion de personnes</h1>
    <PersonneAdd @send-data="ajouterDansListe" />
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            {{ elt.nom }} {{ elt.prenom }}
            <router-link :to="{ name: 'personne-details', params: { id: elt.id } }">
                <i class="far fa-edit"></i>
            </router-link>
            <button class="btn" @click="supprimerPersonne(elt.id)">
                <i style="color:tomato" class="fas fa-trash-alt"></i>
            </button>
        </li>
    </ul>
</template>
```

# Vue.js

**Considérons le contenu suivant pour le `template` de `PersonneShwoView`**

```
<template>
    <h1>Gestion de personnes</h1>
    <PersonneAdd @send-data="ajouterDansListe" />
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            {{ elt.nom }} {{ elt.prenom }}
            <router-link :to="{ name: 'personne-details', params: { id: elt.id } }">
                <i class="far fa-edit"></i>
            </router-link>
            <button class="btn" @click="supprimerPersonne(elt.id)">
                <i style="color:tomato" class="fas fa-trash-alt"></i>
            </button>
        </li>
    </ul>
</template>
```



**Exercice 1**

Faites les modifications nécessaires pour permettre à l'utilisateur d'ajouter 0, 1 ou plusieurs sports.

# Vue.js

**Correction**

```
methods: {
    ajouterDansListe(values) {
        this.personnes.push(values)
    },
    supprimerPersonne(id) {
        console.log(id)
        axios
            .delete(`http://localhost:5555/personnes/${id}`)
            .then(() => this.personnes = this.personnes.filter(elt => elt.id != id))
    }
}
```

# Vue.js

## Exercice 2

Dans le composant `PersonneDetailsView`

- Supprimez le tableau `personnes`,
- Faites une requête **HTTP** pour récupérer la personne depuis la base de données,
- Affichez la personne récupérée dans un formulaire afin de permettre la modification de ses valeurs,
- Ajoutez un bouton `Enregistrer` qui permettra :
  - d'enregistrer les modifications dans la base de données et
  - de rediriger vers le composant `PersonneShowView`.

# Vue.js

**Correction (`template` de `PersonneDetailsView`)**

```
<template>
    <h1>Détails de la personne {{ id }}</h1>
    <p>Personne recherchée</p>
    <Form @submit.prevent="enregistrer">
        <div>
            Nom :
            <input type=text v-model="personne.nom" />
        </div>
        <div>
            Prénom :
            <input type=text v-model="personne.prenom" />
        </div>
        <div>
            Age :
            <input type=number v-model="personne.age" />
        </div>
        <div>
            <button>
                Enregistrer
            </button>
        </div>
    </Form>
    <router-link :to="`/personne/${id}/adresse`">Adresse</router-link> |
    <router-link :to="`/personne/${id}/sport`">Sport</router-link>
    <router-view />
    <button @click="pagePrecedente()">Retour</button>
</template>
```

# Vue.js

**Correction (`script` de `PersonneDetailsView`)**

```
<script>
import axios from 'axios'

export default {
    name: 'PersonneDetailsView',
    props: ['id'],
    data() {
        return {
            personne: { nom: '', prenom: '', age: null }
        }
    },
    mounted() {
        axios
            .get(`http://localhost:5555/personnes/${this.id}`)
            .then(response => this.personne = response.data)
    },
    methods: {
        pagePrecedente() {
            this.$router.go(-1)
        },
        enregistrer() {
            axios
                .put(`http://localhost:5555/personnes/${this.id}`, this.personne)
                .then(() => this.$router.push('/personne'))
        }
    }
}
</script>
```

# Vue.js

## Objectif

Ne plus démarrer séparément les deux serveurs **JSON** et **Vue.js**.

# Vue.js

## Objectif

Ne plus démarrer séparément les deux serveurs **JSON** et **Vue.js**.

## Solution

Utiliser le package **concurrently**.

# Vue.js

## Package **concurrently**

- Package **NodeJS**
- Permettant d'exécuter plusieurs commandes simultanément
- Syntaxe : `concurrently "command1 arg" "command2 arg"`

# Vue.js

## Package **concurrently**

- Package **NodeJS**
- Permettant d'exécuter plusieurs commandes simultanément
- Syntaxe : `concurrently "command1 arg" "command2 arg"`



**Pour l'installer**

```
npm install concurrently --save
```

# Vue.js

**En exécutant la commande**

```
npm start
```

# Vue.js

**En exécutant la commande**

```
npm start
```

**C'est la section `scripts` qui sera consultée et plus précisément la partie `serve`**

```
"scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
},
```

# Vue.js

**Modifions la partie `serve` pour pouvoir démarrer les deux serveurs simultanément**

```
"scripts": {
    "serve": "concurrently \"vue-cli-service serve\" \"json-server -p 5555 ./db.json\""
      ,
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
},
```

# Vue.js

## Question

Et en cas d'erreur (utilisateur ou serveur), comment faire ?

# Vue.js

## Question

Et en cas d'erreur (utilisateur ou serveur), comment faire ?



## Réponse

On peut traiter ça dans le `catch` de la promesse.

# Vue.js

**Dans `script` de `PersonneShowView`, modifions l'URL pour avoir une erreur et utilisons le `catch` pour la capturer**

```
data() {
    return {
        personnes: [
        ]
    }
},
mounted() {
    axios
        .get('http://localhost:5555/personne')
        .then(response => this.personnes = response.data)
        .catch(() => console.log("Erreur"))
},
```

# Vue.js

**Dans `script` de `PersonneShowView`, modifions l'URL pour avoir une erreur et utilisons le `catch` pour la capturer**

```
data() {
    return {
        personnes: [
        ]
    }
},
mounted() {
    axios
        .get('http://localhost:5555/personne')
        .then(response => this.personnes = response.data)
        .catch(() => console.log("Erreur"))
},
```

L'utilisateur ne regarde pas la console, affichons donc un message d'erreur dans `template`.

# Vue.js

**Commençons par déclarer une variable `erreur` dans `data` et initialisons sa valeur dans `catch`**

```
data() {
    return {
        erreur: null,
        personnes: [
        ]
    }
},
mounted() {
    axios
        .get('http://localhost:5555/personne')
        .then(response => this.personnes = response.data)
        .catch((error) => this.erreur = error)
},
```

# Vue.js

**Affichons un message d'erreur dans** `template`

```
<template>
    <h1>Gestion de personnes</h1>
    <template v-if="!erreur">
        <PersonneAdd @send-data="ajouterDansListe" />
        <ul>
            <li v-for="(elt, index) in personnes" :key="index">
                {{ elt.nom }} {{ elt.prenom }}
                <router-link :to="{ name: 'personne-details', params: { id: elt.id }}">
                    <i class="far fa-edit"></i>
                </router-link>
                <button class="btn" @click="supprimerPersonne(elt.id)">
                    <i style="color:tomato" class="fas fa-trash-alt"></i>
                </button>
            </li>
        </ul>
    </template>
    <template v-else>
        <h2>Erreur : la liste n'est pas disponible pour le moment</h2>
    </template>
</template>
```

# Vue.js

**Affichons un message d'erreur dans** `template`

```
<template>
    <h1>Gestion de personnes</h1>
    <template v-if="!erreur">
        <PersonneAdd @send-data="ajouterDansListe" />
        <ul>
            <li v-for="(elt, index) in personnes" :key="index">
                {{ elt.nom }} {{ elt.prenom }}
                <router-link :to="{ name: 'personne-details', params: { id: elt.id }}">
                    <i class="far fa-edit"></i>
                </router-link>
                <button class="btn" @click="supprimerPersonne(elt.id)">
                    <i style="color:tomato" class="fas fa-trash-alt"></i>
                </button>
            </li>
        </ul>
    </template>
    <template v-else>
        <h2>Erreur : la liste n'est pas disponible pour le moment</h2>
    </template>
</template>
```

Nous pouvons aussi utiliser l'interpolation pour afficher le contenu de la variable `erreur`.

# Vue.js

**Nous pouvons aussi préciser la source de l'erreur**

```
<template>
    <h1>Gestion de personnes</h1>
    <template v-if="!erreur">
        <PersonneAdd @send-data="ajouterDansListe" />
        <ul>
            <li v-for="(elt, index) in personnes" :key="index">
                {{ elt.nom }} {{ elt.prenom }}
                <router-link :to="{ name: 'personne-details', params: { id: elt.id }}">
                    <i class="far fa-edit"></i>
                </router-link>
                <button class="btn" @click="supprimerPersonne(elt.id)">
                    <i style="color:tomato" class="fas fa-trash-alt"></i>
                </button>
            </li>
        </ul>
    </template>
    <template v-else-if="erreur.response">
        <h2>Erreur : la liste n'est pas disponible pour le moment</h2>
    </template>
    <template v-else-if="erreur.request">
        <h2>Erreur : Serveur indisponible</h2>
    </template>
    <template v-else>
        <h2>Erreur inconnue : {{ erreur.message }}</h2>
    </template>
</template>
```

# Vue.js

## Question

Comment faire pour éviter d'importer `axios` dans tous les composants qui l'utilisent ?

# Vue.js

## Question

Comment faire pour éviter d'importer `axios` dans tous les composants qui l'utilisent ?

## Solution

Utiliser `vue-axios`.

# Vue.js

**Pour installer** `vue-axios`

```
npm install vue-axios
```

# Vue.js

**Dans `main.js`, importons `VueAxios` d'une manière globale**

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import axios from 'axios'
import VueAxios from 'vue-axios'

createApp(App)
    .use(router)
    .use(VueAxios, axios)
    .mount('#app')

import "bootstrap/dist/css/bootstrap.min.css"
import "bootstrap/dist/js/bootstrap.bundle.min.js"
import "@fortawesome/fontawesome-free/css/all.css"
import "bootstrap-icons/font/bootstrap-icons.css"
import "./assets/css/style.css"
import '@/validators/min-max';
```

# Vue.js

## Ensuite

- Supprimez l'import d'`axios` dans tous les composants
- Remplacez l'objet `axio` par `this.axios` avant l'appel des méthodes `get`, `post`...
- Actualisez la page et vérifiez que tout fonctionne correctement

# Vue.js

## Constat

Les URL utilisées dans les requêtes **AJAX** ont une base commune.

# Vue.js

## Constat

Les URL utilisées dans les requêtes **AJAX** ont une base commune.

## Question

Peut-on simplifier cet appel en définissant la base de l'URL dans un seul fichier afin de faciliter la maintenance ?

# Vue.js

## Constat

Les URL utilisées dans les requêtes **AJAX** ont une base commune.

## Question

Peut-on simplifier cet appel en définissant la base de l'URL dans un seul fichier afin de faciliter la maintenance ?

## Réponse

Oui, on peut la définir comme une propriété globale dans `main.js`.

# Vue.js

**Dans** `main.js`**, définissons une propriété globale appelée** `baseUrl`

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import axios from 'axios'
import VueAxios from 'vue-axios'

const app = createApp(App);

app.config.globalProperties.baseUrl = 'http://localhost:5555';

app.use(router)
    .use(VueAxios, axios)
    .mount('#app')

import "bootstrap/dist/css/bootstrap.min.css"
import "bootstrap/dist/js/bootstrap.bundle.min.js"
import "@fortawesome/fontawesome-free/css/all.css"
import "bootstrap-icons/font/bootstrap-icons.css"
import "./assets/css/style.css"
import '@/validators/min-max';
```

# Vue.js

**Pour référencer `baseUrl` dans les appels HTTP (exemple : `mounted` de `PersonneShowView`)**

```
mounted() {
    this.axios
        .get(`${this.baseUrl}/personnes`)
        .then(response => this.personnes = response.data)
        .catch((error) => this.erreur = error)
},
```

# Vue.js

**Pour référencer `baseUrl` dans les appels HTTP (exemple : `mounted` de `PersonneShowView`)**

```
mounted() {
    this.axios
        .get(`${this.baseUrl}/personnes`)
        .then(response => this.personnes = response.data)
        .catch((error) => this.erreur = error)
},
```



**Remarque**

Faire la même chose dans tous les appels des **API REST**.