Vue.js

Qu’est-ce que VueJS ?

Christopher Blassiaux

Développeur d'Application Web Front-end & Formateur Consultant chez ALT RH Consulting

chrisblassiaux@gmail.com

https://chrisb.fr/

Je travaille avec (liste non exhaustive) :

Je travaille chez :

14H41

Qu'est-ce que c'est ?

↪ **Framework** *Javascript* *(cadre/espace de travail permettant de développer des sites/applications)*

Quelle est l'utilité ? → Une organisation conventionnée / Un gain de temps

Chaque langage a son/ses framework.s

ANGULAR

RAILS

React JS

Laravel

Symfony

django

Version

Première version en **2014**, développé par **Evan You***, VueJS est actuellement à la version :

3.2.41

***Evan You :** développeur Chinois ayant travaillé chez Google et Meteor.*

Avantages

Avantages

- ↪ Léger (23 Ko)
- ↪ Appréhendable (faible courbe d’apprentissage)
- ↪ Flexible
- ↪ Performant dans son exécution
- ↪ Sa syntaxe d’écriture est simple
- ↪ Couvre toutes les fonctionnalités qu’on peut attendre d’un framework Front-end
- ↪ Une documentation moderne

Single Page Application (SPA)

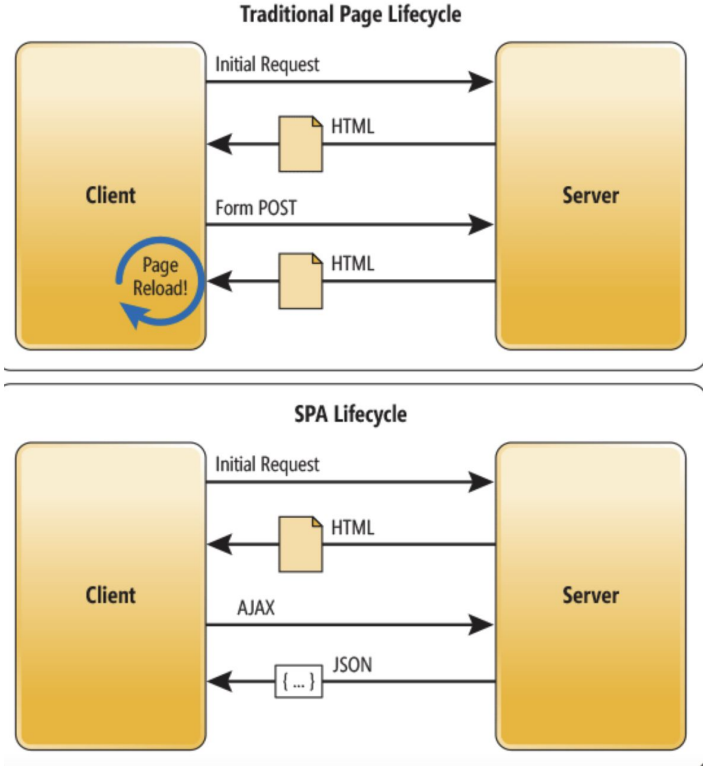
↪ Site/App web composé d'une seule page qui fonctionne sans que l'utilisateur n'ait besoin de recharger la page.

Gmail, Google Analytics, Trello, Dropbox

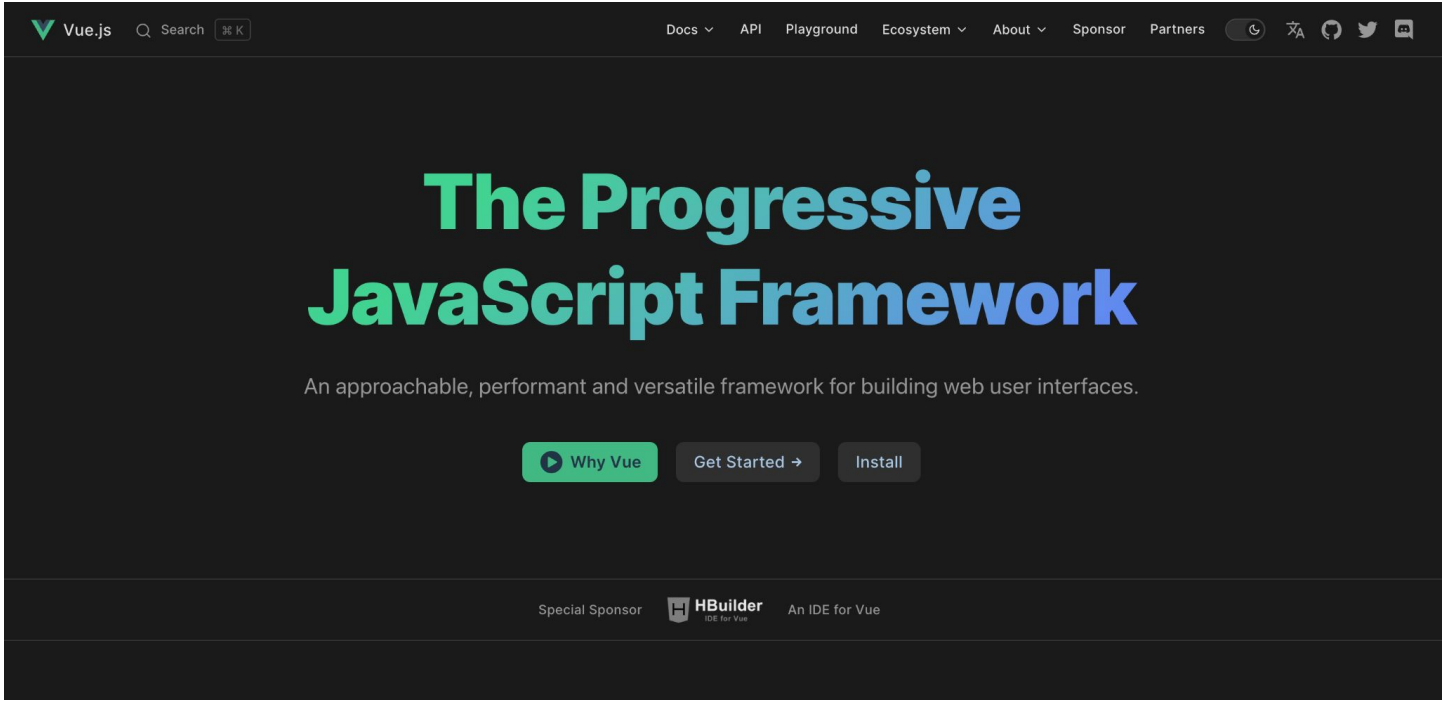
↪ De nombreux frameworks front ont adopté cette architecture (Angular, React, Vue, Ember, Meteor etc)

↪ Avantages : rendu côté client | un seul chargement de page | seules les données sont transmises | développement mobile simplifié

↪ Inconvénients : chargement initial de l'application prend du temps | SEO



Documentation



Les composants

Composant / LOGIQUE

↪ Le composant ou vue (component / view) contrôle une partie de l'écran appelée une **vue**.

↪ Dans le composant, la logique -le code pour contrôler la vue- est réalisée dans un **script.**

Template / Vue

↪ Un template est une forme d'HTML qui dit à VueJS comment doit être effectué le **rendu du composant.**

Style

↪ La partie qui contrôle le style du composant

Programmation orientée composant

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <button class="btn" @click="buttonClick($event)">Affiche un console.log()</button>
  </div>
</template>

<script>
export default {
  name: 'AppIndexPlanets',
  props: {
    msg: String
  },   You, 28/07/2022 15:13 • Uncommitted changes
  data() {
    return {
      maVariable: 'test',
    }
  },
  methods: {
    helloWorld() {
      console.log('Salut je suis un méthode');
    },
    buttonClick(event) {
      console.log(event)
    }
  },
  mounted() {
    this.helloWorld();
  }
}
</script>
```

Créer et comprendre un projet

Créer un projet

1 → Installer npm : https://nodejs.org/en/

```
1 | node -v
```

2 → Installation du VueJS CLI

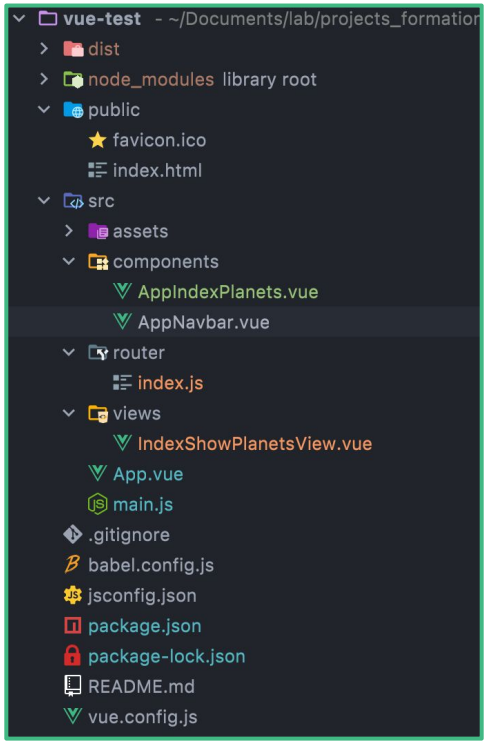
```
npm install -g @vue/cli
```

3 → Créer le projet VueJS

```
vue create vue-test
```

```
vue serve
```

Comprendre la structure



BILAN

Nous avons vu jusqu'à présent :

↪ Introduction à VueJS (SPA)

↪ Qu'est-ce qu'un composant VueJS

↪ Comment une application VueJS démarre

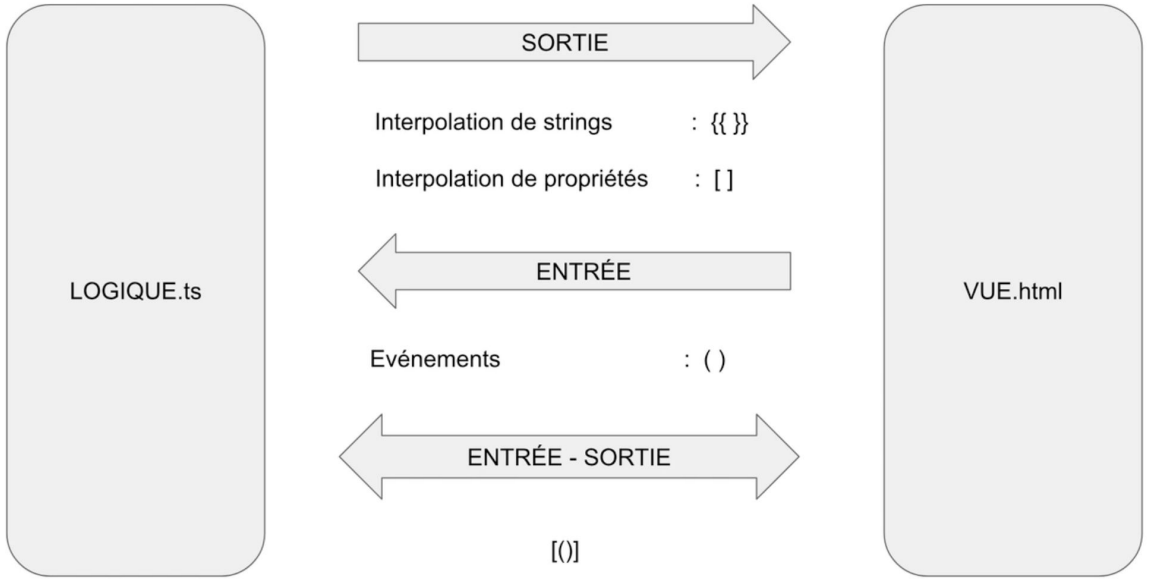
↪ Comment déclarer des templates et des styles

EST-CE COMPRIS ?

Data-binding (réactivité)

↪ Les data bindings possibles dans VueJS sont les suivants :

- L'interpolation
- La liaison de propriété (property binding)
- La liaison d'attribut (attribute binding)
- La liaison d'événement (event binding)
- La liaison de classe (class binding)
- La liaison de style (style binding)
- La liaison de données dans les deux sens (two-way data binding)



String interpolation

↪ Interpolation en JS {{}} → Convertis en chaîne de caractère

```
<template>
  <div class="hello">
    <h1>{{ maVariable }}</h1>
    <button class="btn" @click="buttonClick($event)">Affiche un console.log()</button>
  </div>
</template>

<script>
export default {
  name: 'AppIndexPlanets',
  data() {
    return {
      maVariable: 'test',
    }
  },
```

newapp 2

42

true

[object Object]

Une instruction JS peut-être lu par les interpolations {{ 1 + 1 }} ou {{ sayHello() }}

La liaison des propriétés

```
<input :type="dynamicType">
```

```
data() {
  return {
    dynamicType: 'text'
  }
},   You, A minute ago • Uncommitted char
```

La liaison d'événements

↪ @click

↪ $event

Objet d'événement permettant de récupérer des informations à propos de l'événement.

template

```
<template>
  <div class="hello">
    <h1>{{ maVariable }}</h1>
    <button class="btn" @click="buttonClick($event)">Affiche un console.log()</button>
  </div>
</template>
```

logique

```
methods: {
  helloWorld() {
    console.log('Salut je suis un méthode');
  },
  buttonClick(event) {
    console.log(event)
  }
},
mounted() {
  this.helloWorld();
}
```

Directives

↪ ***Instruction qui est donnée au DOM***

```
<p backgroundBlue>Hello</p>
```

Instruction

↪ composants : Directive

↪ :style, :class : Directives d'attributs

Modifient le comportement ou le l'apparence

↪ v-if , v-for : Directives structurelles

Modifient la structure du DOM

```
<app-index-planets></app-index-planets>
```

Directive

Documentation & directives

Découverte de la documentation : https://vuejs.org/api/

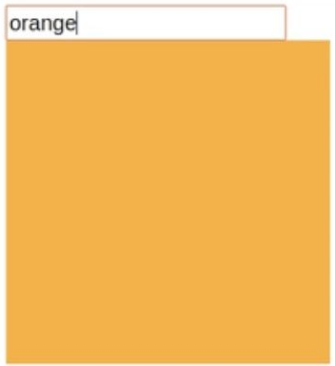
↪ :style :class : https://vuejs.org/guide/essentials/class-and-style.html#binding-html-classes

↪ v-if : https://vuejs.org/api/built-in-directives.html#v-if

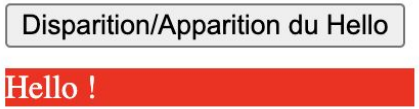
↪ v-for : https://vuejs.org/guide/essentials/list.html#v-for

Exos Directives

Exercice 1 :



Exercice 2 :



Exercice 3 :

```
users = [
  {
    firstName: 'Jean',
    lastName: 'Bon'
  },
  {
    firstName: 'John',
    lastName: 'Doe'
  }
]
```

0

Prénom : Jean

Nom : Bon

1

Prénom : John

Nom : Doe

Début de notre première application !

Planification

↪ Intro :

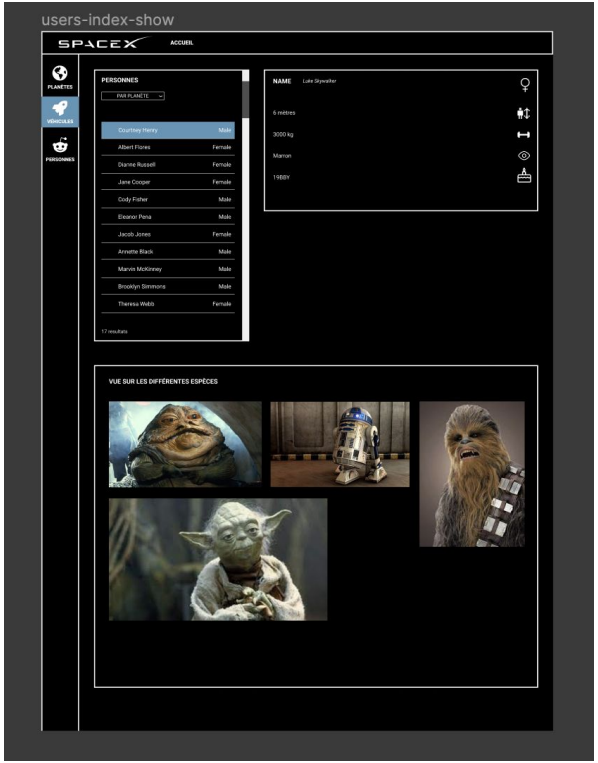
- ↪ Imaginer les différentes tâches
- ↪ Imaginer les différents composants (*arborescence*)

↪ Découvrir et comprendre la maquette : https://www.figma.com/file/GEbL7cnSwyRIrt5C0iw6p0/about_the_universe_app?node-id=0%3A1

Le site d'exemple : https://adoring-dubinsky-fed927.netlify.app/#/

↪ Découverte de l'API : https://swapi.dev/

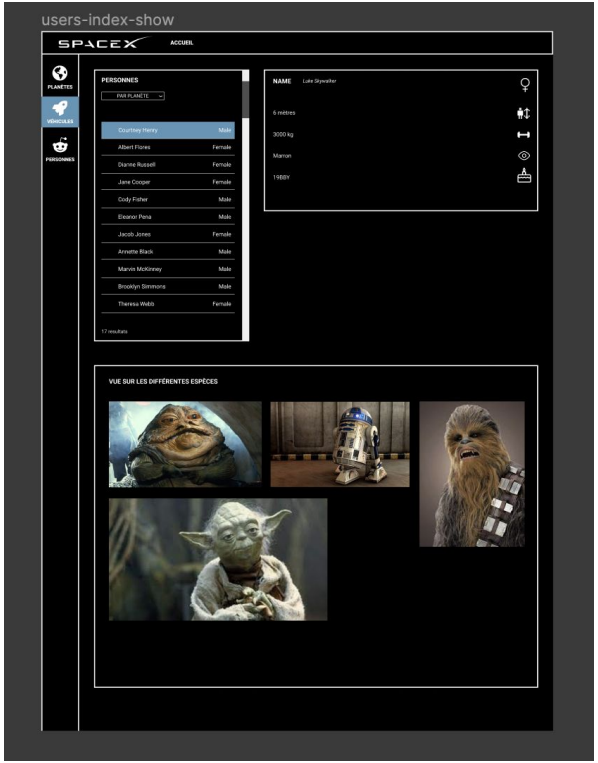
↪ Découverte du KIT UI (HTML / CSS) : https://framagit.org/ChrisBlassiaux/about_the_universe_kit_ui



Planification

↪ Tâches :

- → Add favicon to application
- → Integration of Kit UI (SCSS)
- → Ajout de la font & de font-awesome
- → Develop navbar component
- → Develop home-page component
- → Init first HTTP Request (API)
- → Init/Develop Routing
- → Develop index-users component
- → Develop index-show-users component
- → Develop index-planets component
- → Develop index-show-planets component
- → Develop index-vehicles component
- → Develop index-show-vehicles component
- → Develop filters with selects for index components



Mise en place du projet

```
vue create about-the-universe
```

Premières tâches :

↪ Amener le style (kit ui)

↪ Lancement du projet (serveur)

↪ Ajout de la font, de font-awesome / Modification de la favicon

Navbar

↪ Créer un composant navbar

↪ Rechercher le **HTML** concerné dans le KIT UI et l'amener dans le composant navbar

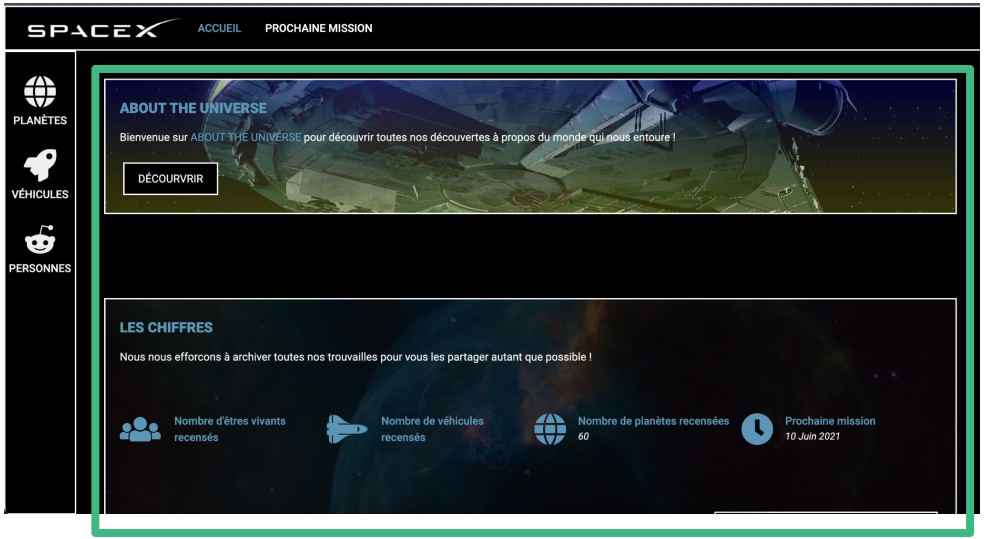
↪ Afficher/déclarer le composant navbar dans le composant principale

Home page

↪ Créer un composant home-page

↪ Rechercher le **HTML** concerné dans le KIT UI et l'amener dans le composant home-page

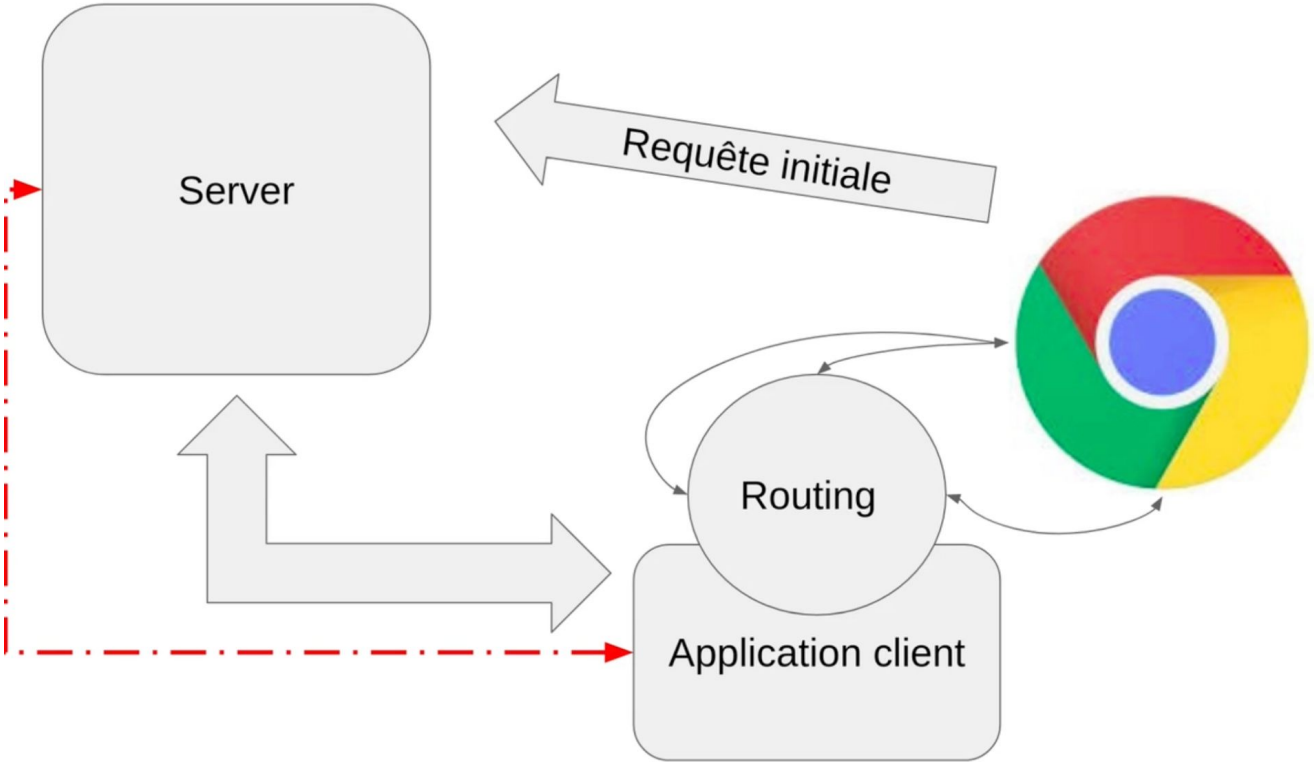
↪ Afficher/déclarer le composant home-page dans le composant principale



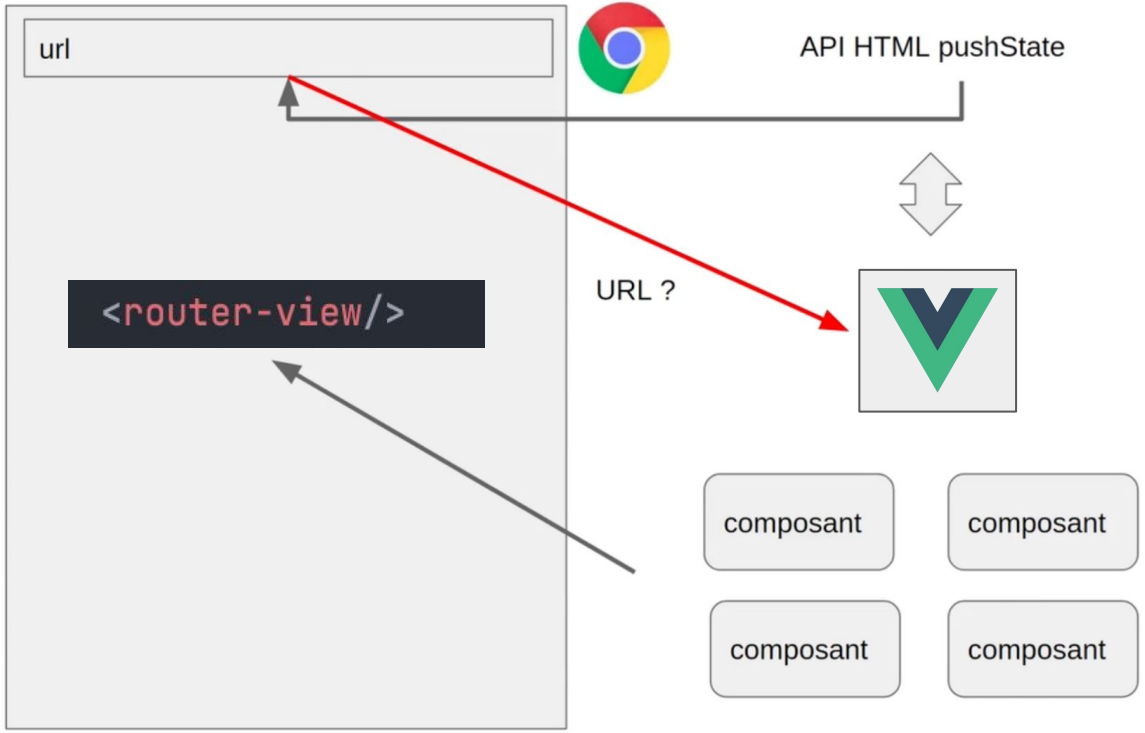
home-page

Routing

Routing : Introduction et mise en place



Routing : Introduction et mise en place



Routing : Les bases du Router Angular

1.

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: IndexShowPlanetsView
  },
]
```

2.

```
<template>
  <AppNavbar></AppNavbar>
  <router-view/>
</template>
```

Le router-link

↪ https://router.vuejs.org/guide/#router-link

```
<router-link to="/">Go to Home</router-link>
<router-link to="/about">Go to About</router-link>
```

Routing : La directive RouterLinkActive

↪ Route active dans la navbar

https://v3.router.vuejs.org/api/#active-class

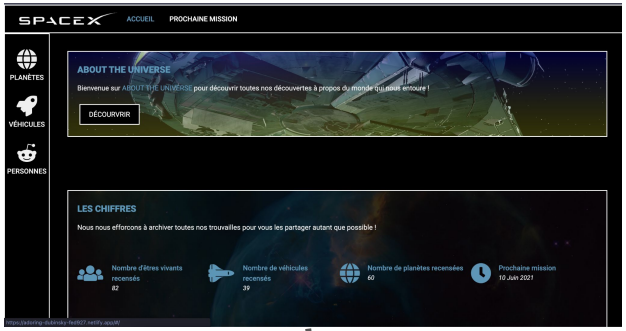
https://v3.router.vuejs.org/api/#exact

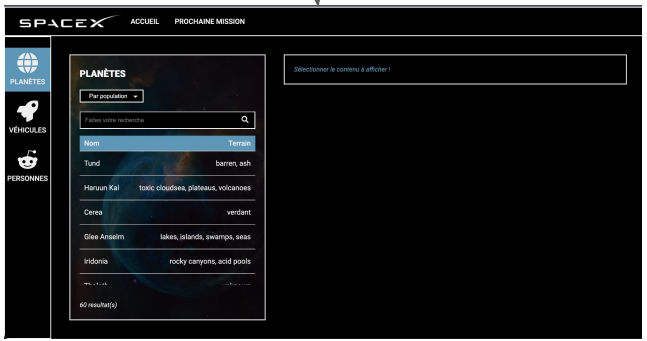
Projet : Init/Develop routing

↪ Développer le système de route dans notre projet (création des routes)

↪ Générer le composant “index-show-planets”

↪ Rendre la barre de navigation fonctionnelle avec les deux composants actuellement présents “index-show-planets” et “home-page”





Faire des requêtes HTTP

Axios

↪ Il existe plusieurs méthodes pour discuter avec une base de données (via des APIs).

↪ Axios est un client HTTP basé sur le **promesses**.

GET

POST

PUT/PATCH

DELETE

Requêtes AJAX

requête http classique

Barre d'url

API HTML (ex: form)

requête http AJAX

JavaScript

(via XmlHttpRequest ou fetch)

Exemple

```
new Vue({
  el: '#app',
  data () {
    return {
      info: null
    }
  },
  mounted () {
    axios
      .get('https://api.coindesk.com/v1/bpi/currentprice.json')
      .then(response => (this.info = response))
  }
})
```

```
<div id="app">
  {{ info }}
</div>
```

Exercice : Votre première requête HTTP

↪ Découvrir la documentation de l'API (database en ligne) : https://swapi.dev/

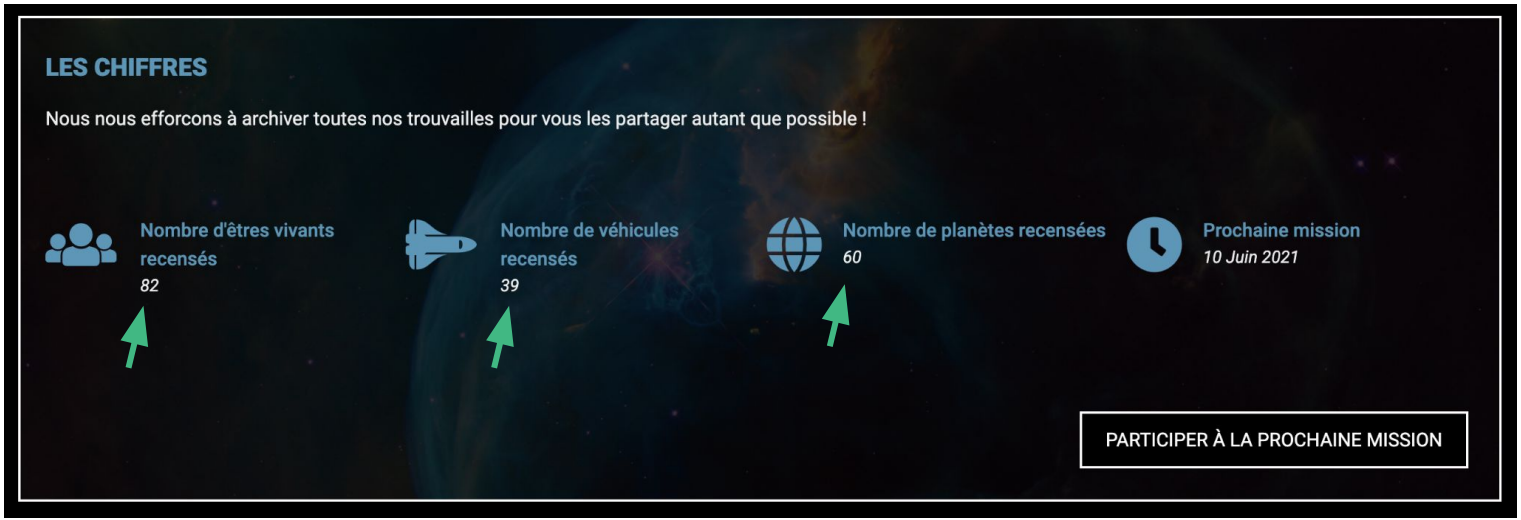
↪ Requêtes HTTP : Via Axios

↪ Réaliser une fonction appelée getCharacter() qui permettra d'aller chercher un personnage sur la base de donnée. Cette fonction doit être appelée dans la fonction mounted(). **Objectif :** Afficher le nom d'un personnage sur la home page.

Documentation :
https://v2.fr.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html

Develop home-page component

↪ Aller rechercher les informations en base de données pour rendre le composant home-page dynamique.

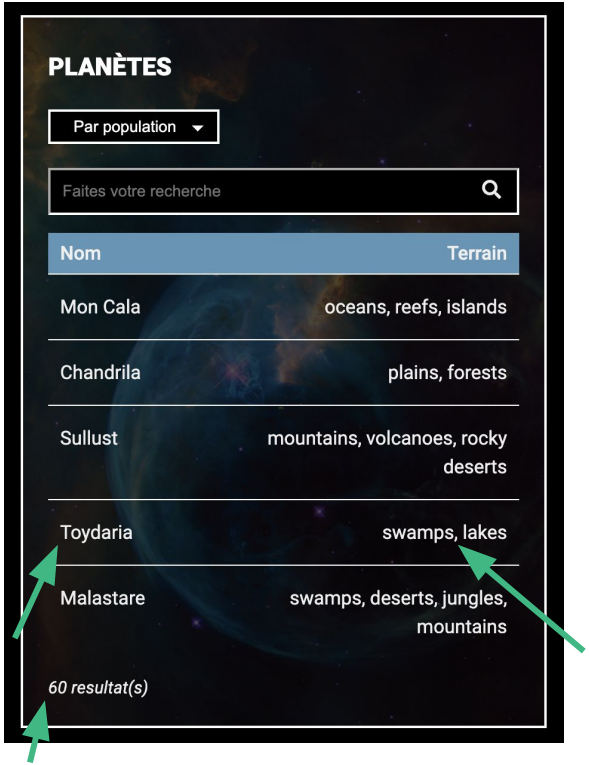


Develop index-planets component

↪ Générer le composant index-planets

↪ Placer le composant dans le composant index-show-planets

↪ Aller rechercher les informations en base de données pour rendre le composant index-planets dynamique.



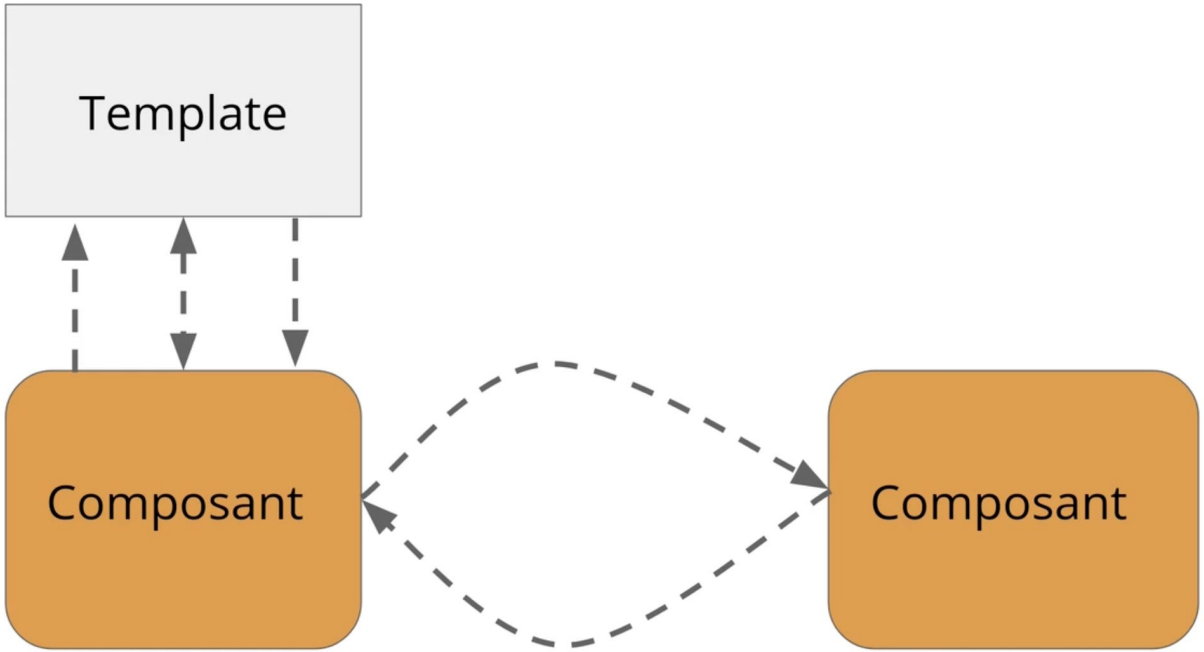
Develop le filtre de index-planets

↪ Lorsque l'utilisateur choisi une des options, celle-ci vient modifier le nombre des planètes affichées.

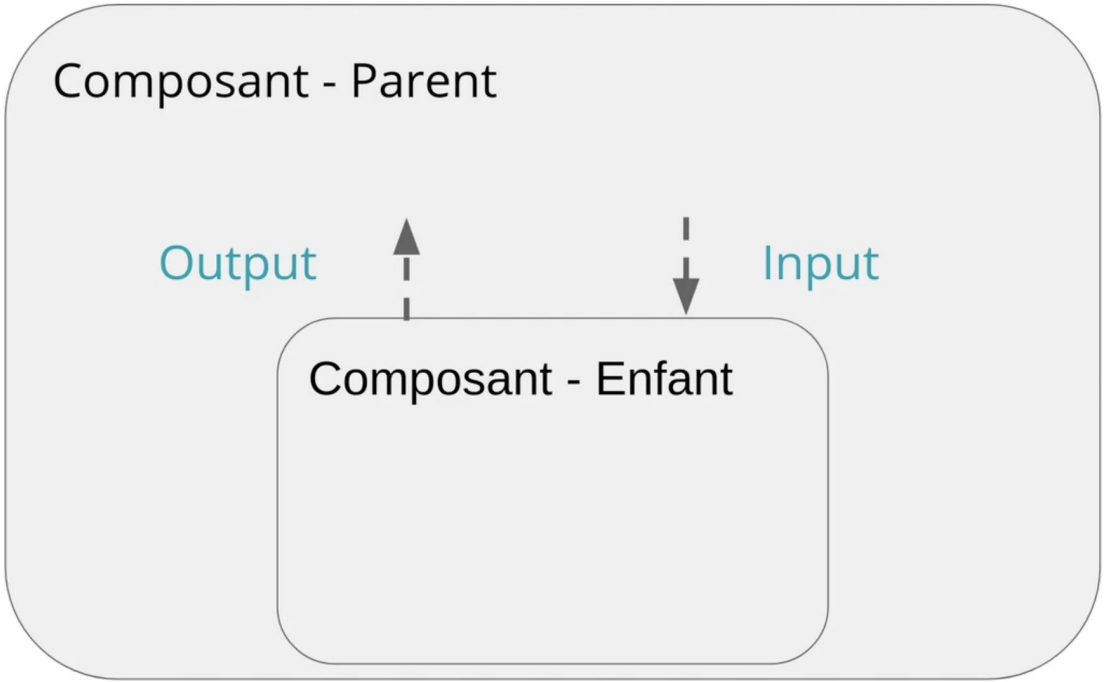
↪ Exemple : 0 à 100.000 affichera uniquement les planètes ayant moins de 100.000 habitants

Approfondissement des composants

Communication entre les composants



Communication entre les composants



About the universe

Quelle propriété va nous servir dans notre cas ?

Propriété Output (notre cas)

↪ Se rendre dans le composant enfant :

```
import axios from 'axios';

export default {
  name: 'AppNavbar',
  data() {
    return {
      planets: [],
      planetsSaved: []
    }
  },
  methods: {
    showPresentation(p) {
      this.$emit('presentation-planet', p);
    },
```

```
<tr v-for="p in planets" v-bind:key="p.name" @click="$event => showPresentation(p)">
  <td>{{ p.name }}</td>
  <td>{{ p.terrain }}</td>
</tr>
```

Propriété Output

↪ Se rendre dans le composant parent :

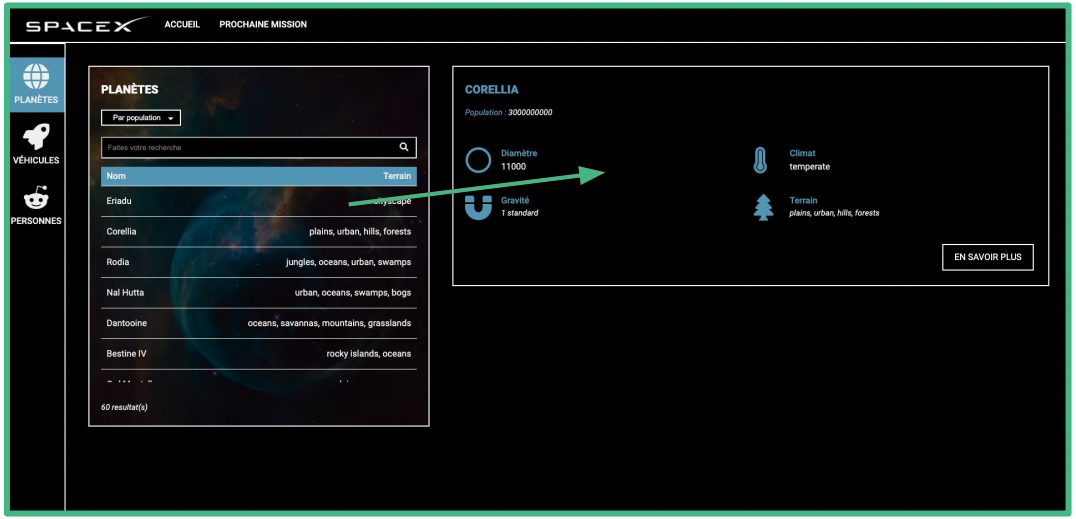
```
<script>
import AppListPlanets from '@/components/AppListPlanets.vue';

export default {
  name: 'PresentationPeople',
  data() {
    return {
      planet: false,
    }
  },
  methods: {
    setPlanet(p) {
      this.planet = p;
    },
  },
  components: {AppListPlanets},
}
</script>
```

```
<app-list-planets @presentation-planet="setPlanet"></app-list-planets>
```

Projet : Output

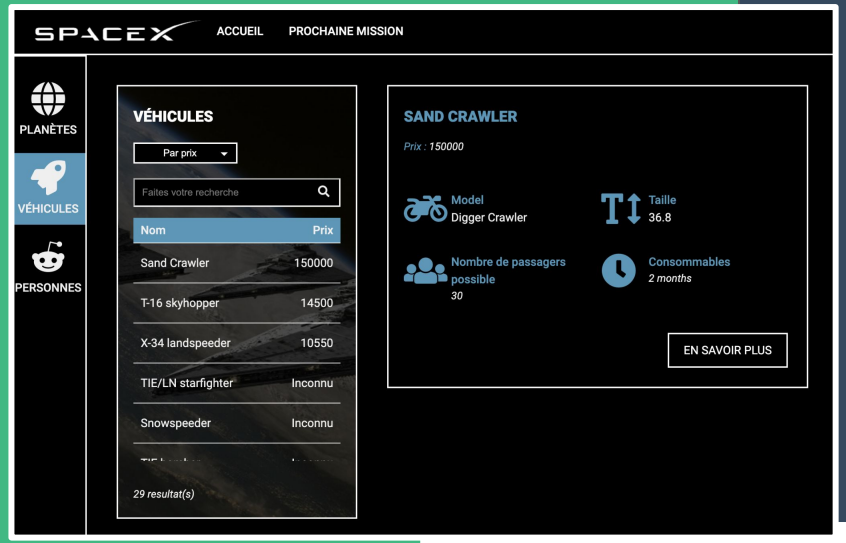
↪ Lorsque l'utilisateur clique sur une planète, la donnée de celle-ci doit-être envoyé au composant parent index-show-planets



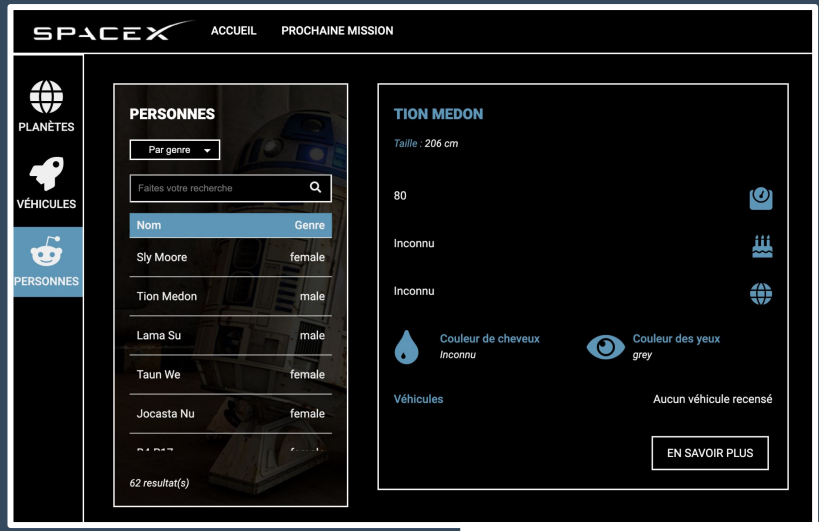
↪ index-planets → index-show-planets

Autonomie

Développer les autres parties :



Partie Véhicules



Partie Personnes

Kahoot : QCM

QCM : *Les bases VueJS*

Fin de la formation

VueJS Découverte

JOUR 1

Par Christopher Blassiaux