Module : Base du Vue.JS

Version 1

Formateur :

DANGLA Loïc
loic.dangla1@ecoles-epsi.net

Déroulement du module

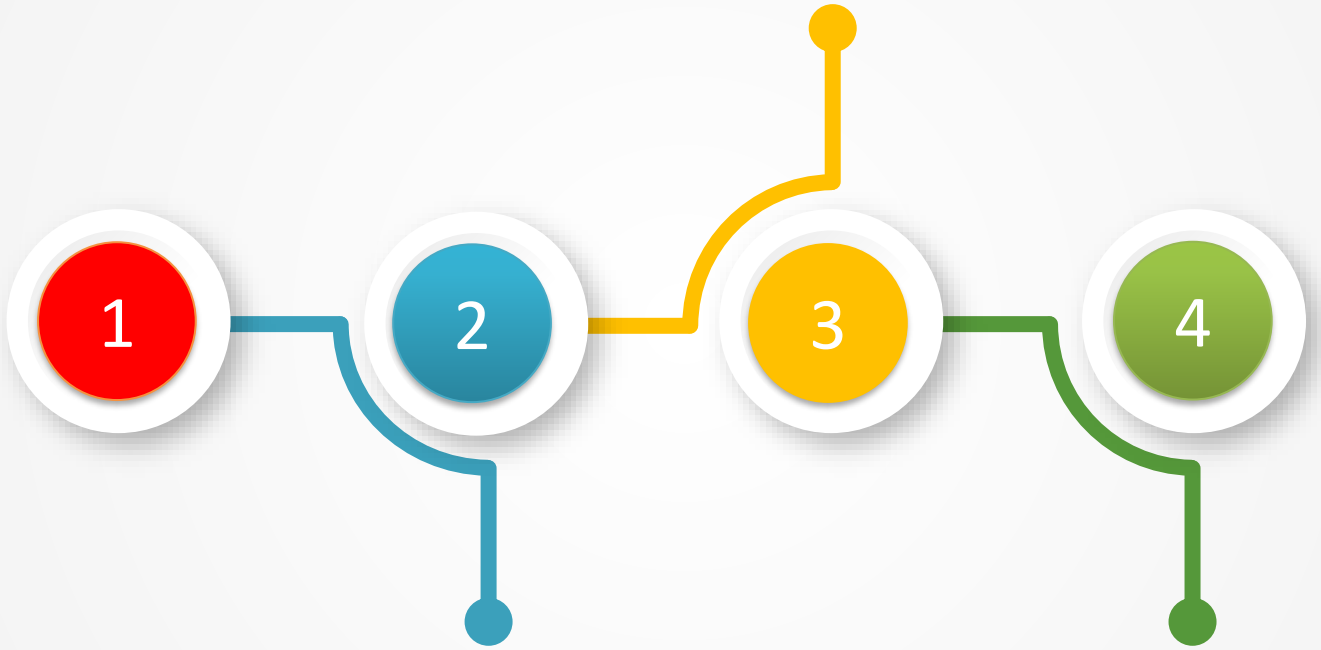
- Durée du module : 16 heures
- Nombre d’évaluations : 2 (TD + QCM)

Introduction

Histoire du Vue.JS

Composants Vue

Composant avancés
Communication en composant
Transition et animation



Fondamentaux et concepts avancés

Directives avancées
Filtrage et calculs dans les modèles
Mixins et directives globales

Routage avec Vue Router

Routage Vue
Gestion de la navigation et de la transition

Vue.JS pour vous, c’est ?

Un peu d’histoire

- Vue.js est un Framework JavaScript moderne pour la construction d'interfaces utilisateur.
- 2009: Création par Evan You - Vue.js a débuté en tant que projet individuel.
- 2014: Version 1.0 - Première version publique.
- 2016: Version 2.0 - Introduction des composants et de la Virtual DOM.
- 2020: Vue.js 3.0 - Restructuration majeure avec une architecture plus performante.

Un peu d’histoire

- Vue.js a gagné en popularité grâce à sa simplicité, sa flexibilité et sa courbe d'apprentissage douce.

- Vue.js a évolué grâce à une communauté active et un engagement continu de l'équipe de développement.

L’Univers Vue.JS

- Outre le cœur de Vue.js, l'écosystème est enrichi par des bibliothèques et des outils complémentaires.

- Vue Router pour la navigation, Vuex pour la gestion d'état, Vue CLI pour le développement rapide.

Comparaison entre Vue.js, React, et Angular

Décortiquer les Principaux Frameworks Front-End

- Vue.js, React, et Angular sont trois frameworks front-end populaires.
- Nous allons examiner différents aspects pour vous aider à choisir celui qui correspond le mieux à vos besoins.

Vue.js - Principales Caractéristiques

- Vue.js se distingue par sa simplicité et son approche progressive.
- Système de composants flexible.
- Courbe d'apprentissage douce, idéal pour les petits et grands projets.

Comparaison entre Vue.js, React, et Angular

React - Principales Caractéristiques

- React est soutenu par Facebook et est largement utilisé dans l'industrie.
- Approche basée sur les composants.
- Utilisation de JSX (JavaScript XML) pour écrire des composants de manière déclarative.

Angular - Principales Caractéristiques

- Angular est un framework complet développé par Google.
- Architecture robuste avec une large gamme de fonctionnalités.
- Utilisation de TypeScript pour une typage statique.

Comparaison entre Vue.js, React, et Angular

Comparaison des Courbes d'Apprentissage

- Vue.js offre une courbe d'apprentissage douce, idéale pour les débutants.
- React demande une compréhension de JSX et de concepts avancés.
- Angular nécessite une courbe d'apprentissage plus raide en raison de sa complexité.

Cas d'Utilisation et Scénarios Recommandés

- Vue.js est idéal pour des applications légères et des projets où la simplicité est cruciale.
- React est souvent utilisé dans des projets de grande envergure, en particulier dans les applications à rendu côté client.
- Angular excelle dans les projets d'entreprise complexes, en particulier lorsque l'évolutivité et la maintenabilité sont des priorités.

Comparaison entre Vue.js, React, et Angular

Écosystèmes et Communautés

- Examinons la taille et la vitalité des communautés Vue.js, React et Angular.
- Les écosystèmes influent sur la disponibilité des bibliothèques tierces, des tutoriels et de la documentation.

Performances et Optimisation

- Vue.js, React et Angular ont tous des mécanismes de rendu virtuel pour optimiser les performances.
- Les différences peuvent être notables dans des situations spécifiques.

Comparaison entre Vue.js, React, et Angular

Récapitulatif - Choix du Framework

- Choisir entre Vue.js, React et Angular dépend des besoins du projet.
- Considérez la taille du projet, les compétences de l'équipe, et les objectifs à long terme.

Installation de Vue.js via Vue CLI

- Vue CLI (Interface de Ligne de Commande) simplifie grandement le processus d'installation de Vue.js.

- Commande pour installer Vue CLI :

npm install -g @vue/cli

- Commande pour créer votre projet :

vue create mon-projet-vue

Crée un nouveau projet Vue avec des options interactives.

Structure de base d'un projet Vue.js

- Un projet Vue généré par Vue CLI possède une structure organisée.

- Examinons les dossiers et fichiers clés :
  - **node_modules:** Dépendances du projet.
  - **public:** Contient les fichiers statiques.
  - **src:** Le code source de l'application Vue.
    - **components:** Emplacement des composants Vue.
    - **App.vue:** Composant racine de l'application.
    - **main.js:** Point d'entrée de l'application.

Structure de base d'un composant Vue

- Un composant Vue est une unité réutilisable.
- Exemple de '**App.vue**' :

```vue
<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Bonjour, Vue.js!'
    };
  }
};
</script>
```

La section <template> contient le HTML, <script> contient la logique JavaScript, et <style> pour le CSS.

Exercice

Exercice 1 :

1. Ouvrez un terminal.

1. Utilisez les commandes :
   - **npm install -g @vue/cli**
   - **vue create mon-nouveau-projet**

1. Suivez les étapes interactives pour configurer votre projet.

1. Explorez la structure générée.

Exercice

Exercice 2 :

1. Dans le dossier src/components, créez un nouveau fichier ‘**MonComposant.vue**’.
1. À l'intérieur, créez un composant simple avec une propriété de données.
1. Importez et utilisez ce composant dans ‘**App.vue**’.
1. Lancez votre application et observez les résultats.

Exercice

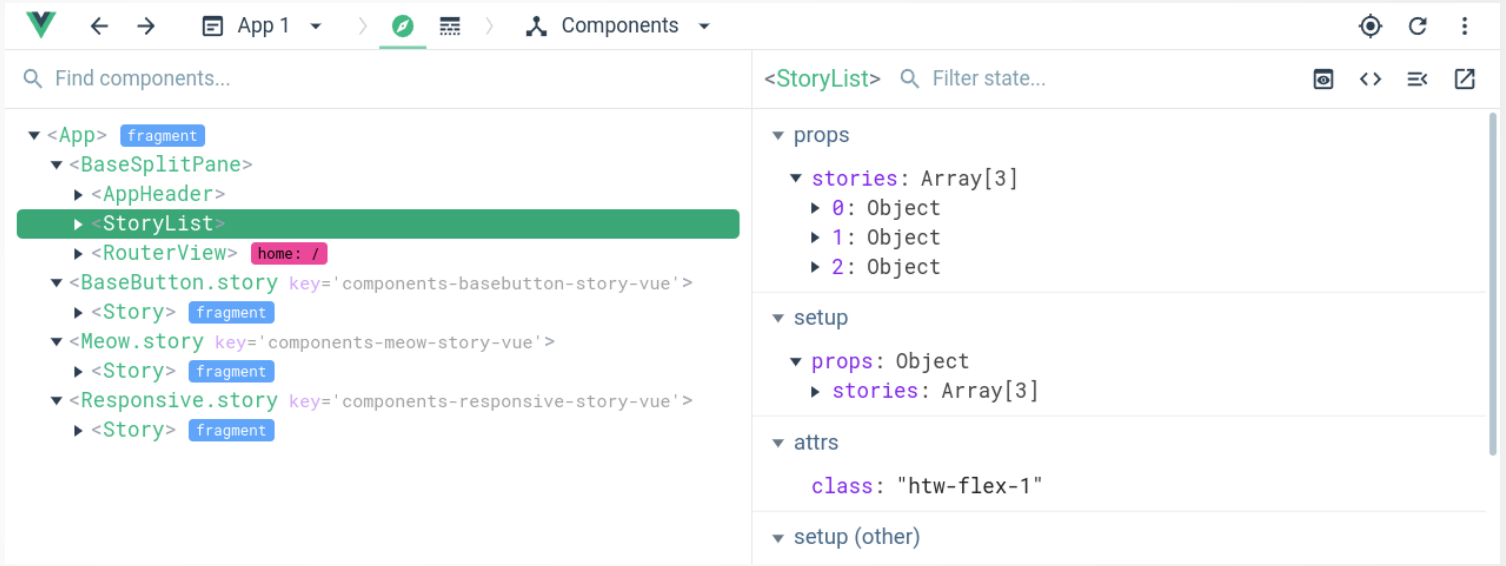
Exercice 2 :

```vue
<!-- MonComposant.vue -->

<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: "Bonjour depuis MonComposant!"
    };
  }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

```vue
<!-- App.vue -->

<template>
  <div id="app">
    <MonComposant />
  </div>
</template>

<script>
import MonComposant from "@/components/MonComposant.vue";

export default {
  components: {
    MonComposant
  }
};
</script>

<style>
/* Styles globaux de l'application */
</style>
```

Vue Devtools et Outils de Développement

- Vue Devtools est une extension de navigateur pour déboguer les applications Vue.



Installation et utilisation facilitées par Vue CLI.

- L'utilisation de Vue CLI simplifie la configuration initiale.
- La structure de base d'un projet Vue.js offre une organisation logique.
- Devtools facilitent le débogage et l'optimisation du code.

Récapitulatif

Configuration et Outils

- L'utilisation de Vue CLI simplifie la configuration initiale.
- La structure de base d'un projet Vue.js offre une organisation logique.
- Devtools facilitent le débogage et l'optimisation du code.

Fondamentaux et concepts avancés

Directives Avancées de Vue.js

Maîtriser les Directives Puissantes

- Les directives telles que **v-if, v-else-if, v-else, v-show, v-for, v-bind, v-on, v-model** offrent une flexibilité accrue.

- Exemple de v-bind :

```
<a v-bind:href="url">Lien</a>
```

Ceci lie dynamiquement l'attribut href à la variable url.

Directives Personnalisées

Personnaliser Votre Vue

- Vue permet la création de directives personnalisées.
- Exemple :

HTML

```html
<p v-highlight>Texte surligné</p>
```

Vue + No-Library (pure JS)

```js
Vue.directive('highlight', {
  bind(el, binding) {
    el.style.backgroundColor = binding.value;
  }
});
```

Cette directive crée un surlignage avec la couleur spécifiée.

Exercice

Exercice 3 : Utilisation de Directives

- Ajoutez un champ de saisie (‘**input**’) dans votre composant.
- Utilisez la directive ‘**v-model**’ pour lier le champ de saisie à une variable dans vos données.
- Utilisez ‘**v-show**’ pour afficher ou masquer un élément en fonction d'une condition.

Exercice

Exercice 3 : Utilisation de Directives

```vue
<!-- MonComposant.vue -->

<template>
  <div>
    <input type="text" v-model="inputValue" />
    <p>La valeur saisie est : {{ inputValue }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      inputValue: ""
    };
  }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

```vue
<!-- MonComposant.vue -->

<template>
  <div>
    <input type="text" v-model="inputValue" />
    <p v-show="inputValue !== ''">La valeur saisie est : {{ inputValue }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      inputValue: ""
    };
  }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

Filtrage dans les Modèles

Affiner les Données avec les Filtres (Vue.JS 2)

- Les filtres permettent de formater les données avant de les afficher.
- Exemple :

```
<p>{{ message | uppercase }}</p>

filters: {
  uppercase(value) {
    return value.toUpperCase();
  }
}
```

Ceci convertit le texte en majuscules.

Calculs Complexes dans les Modèles

Plus qu'une Simple Affichage

- Les modèles permettent des calculs dynamiques.

- Exemple :

```
<p>Total : {{ quantity * price }}</p>
```

Exercice

Exercice 4: Utilisation de Filtres et Calculs dans les Modèles

- Ajoutez une propriété **price** et **quantity** à votre composant.
- Utilisez une expression pour calculer le total à afficher.

Exercice

Exercice 4: Utilisation de Filtres et Calculs dans les Modèles

```vue
<!-- MonComposant.vue -->

<template>
  <div>
    <p>Prix : {{ price }}</p>
    <p>Quantité : {{ quantity }}</p>
    <p>Total : {{ total }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      price: 20.5,
      quantity: 3
    };
  },
  computed: {
    total() {
      return this.price * this.quantity;
    }
  }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

Mixins et Directives Globales

Partager la Logique - Mixins et Directives Globales

- Les mixins permettent de réutiliser la logique entre composants.
- Directives globales facilitent l'ajout de comportements à l'échelle de l'application.

Mixins en Action

Exemple de l'utilisation des mixins :

```
// Définition du mixin
const myMixin = {
  data() {
    return {
      mixinData: 'Données du Mixin'
    };
  }
};

// Utilisation du mixin
export default {
  mixins: [myMixin]
};
```

Exercice

Exercice 5 : Création et Utilisation de Mixins

- Créez un fichier mixin avec des données partagées.
- Utilisez ce mixin dans deux composants différents.
- Vérifiez si les données du mixin sont accessibles dans les deux composants.

Exercice

Exercice 5 : Création et Utilisation de Mixins

src/mixins

```
// MonMixin.js

export default {
  data() {
    return {
      sharedData: 'Données partagées depuis le mixin'
    };
  }
};
```

```
<!-- ComposantA.vue -->

<template>
  <div>
    <p>Composant A - {{ sharedData }}</p>
  </div>
</template>

<script>
import MonMixin from "@/mixins/MonMixin.js";

export default {
  mixins: [MonMixin],
  // ... autres options et méthodes
};
</script>
```

```
<!-- ComposantB.vue -->

<template>
  <div>
    <p>Composant B - {{ sharedData }}</p>
  </div>
</template>

<script>
import MonMixin from "@/mixins/MonMixin.js";

export default {
  mixins: [MonMixin],
  // ... autres options et méthodes
};
</script>
```

Création de Directives Globales

Directive Globale en Vue

Création d'une directive globale :

```
Vue.directive('red-color', {
  bind(el) {
    el.style.color = 'red';
  }
});
```

Cette directive peut être utilisée dans n'importe quel composant.

Exercice

Exercice 6 : Création et Utilisation de Directives Globales

- Créez une directive globale qui modifie l'apparence d'un élément.
- Utilisez cette directive dans plusieurs composants.
- Observez comment la directive affecte le style des éléments.

Exercice

Exercice 6 : Création et Utilisation de Directives Globales

src/directives

```
// ApparenceDirective.js

export default {
  // Hook bind pour appliquer la directive lorsqu'elle est liée à un élément
  bind(el, binding) {
    // Modification de l'apparence en fonction de la valeur de la directive
    el.style.color = binding.value;
  }
};
```

```
// main.js

import Vue from 'vue';
import App from './App.vue';
import ApparenceDirective from './directives/ApparenceDirective';

// Enregistrement de la directive globale
Vue.directive('apparence', ApparenceDirective);

new Vue({
  render: h => h(App),
}).$mount('#app');
```

```
<!-- ComposantA.vue -->

<template>
  <div>
    <p v-apparence="'red'">Composant A - Texte rouge</p>
  </div>
</template>
```

```
<!-- ComposantB.vue -->

<template>
  <div>
    <p v-apparence="'blue'">Composant B - Texte bleu</p>
  </div>
</template>
```

Récapitulatif

Directives et Logique Réutilisable

- Les directives avancées offrent un contrôle précis sur le DOM.
- Filtres et calculs enrichissent les modèles.
- Mixins et directives globales permettent une logique réutilisable.

Composants Vue - Approfondissement

Composants Avancés de Vue.js

Plongée Profonde dans les Composants

- Les composants avancés vont au-delà du simple HTML et JavaScript.
- Nous explorerons les slots, les composants fonctionnels et comment ils améliorent la flexibilité de Vue.js.

Flexibilité avec Slots et Portées dans Vue.js

- Les slots permettent d'injecter du contenu dans un composant parent.
- Exemple :

```
<button>
  <slot>Contenu par défaut</slot>
</button>
```

La portée (‘scope’) permet de passer des données dans le slot.

Exercice

Exercice 7: Utilisation de Slots et Portées

- Créez un composant avec un slot.
- Utilisez la portée pour afficher des données dans le slot.
- Testez le composant en l'utilisant dans un autre composant parent.

Exercice

Exercice 7: Utilisation de Slots et Portées

- src/components

```vue
<!-- ComposantSlot.vue -->

<template>
  <div>
    <h2>Composant avec Slot</h2>
    <slot :message="message"></slot>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: "Données transmises via la portée du slot"
    };
  }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

Exercice

Exercice 7: Utilisation de Slots et Portées

```vue
<!-- ComposantParent.vue -->

<template>
  <div>
    <h1>Composant Parent</h1>
    <ComposantSlot>
      <template v-slot="{ message }">
        <p>{{ message }}</p>
      </template>
    </ComposantSlot>
  </div>
</template>

<script>
import ComposantSlot from "@/components/ComposantSlot.vue";

export default {
  components: {
    ComposantSlot
  }
};
</script>

<style>
/* Styles globaux du composant parent */
</style>
```

Composants Fonctionnels Vue.js

Efficacité avec les Composants Fonctionnels

- Les composants fonctionnels sont des composants légers, sans état ni instance.
- Idéaux pour améliorer les performances dans certains cas.

Exemple :

```
export default {
  functional: true,
  render(h, context) {
    return h('div', `Je suis un composant fonctionnel`);
  }
};
```

Exercice

Exercice 8: Création de Composants Fonctionnels

- Créez un composant fonctionnel simple.
- Testez sa performance en comparaison avec un composant standard.

Communication Complexe entre Composants

Transmettre l'Information avec Habileté

- Communication entre composants va au-delà des événements simples.
- Nous explorerons les événements personnalisés avancés et la communication entre composants non parents/enfants.

Événements Personnalisés Avancés

- Élever les Événements à un Niveau Supérieur
- Utilisation de l'émission d'événements pour transmettre des données.
- Exemple :

```
this.$emit('mon-evenement', data);
```

Exercice

Exercice 9 : Émission d'Événements Personnalisés

- Créez deux composants.
- Utilisez un événement personnalisé pour transmettre des données du composant enfant au parent.
- Affichez les données dans le parent.

Communication entre Composants Non Parents/Enfants

Surmonter les Limites de la Hiérarchie

- Utilisation d'un bus d'événements ou d'un gestionnaire d'état global pour la communication entre composants distants.

Exercice

Exercice 10: Communication Globale entre Composants

- Créez deux composants non liés.

- Utilisez un bus d'événements ou un gestionnaire d'état global pour partager des données entre eux.

Transition et Animation avec Vue.js

Ajouter du Flair avec des Transitions

- Vue.js facilite l'ajout d'effets de transition et d'animation aux composants.
- Utilisation de la balise <transition> et des classes CSS.

Animer les Composants avec Vue.js

Élévation Visuelle des Composants

```
<transition name="fade">
  <p v-if="show">Je m'anime!</p>
</transition>
```

Classes CSS associées :

```
.fade-enter-active, .fade-leave-active {
  transition: opacity 1s;
}
.fade-enter, .fade-leave-to {
  opacity: 0;
}
```

Exercice

Exercice 11: Ajout de Transitions et Animations

- Intégrez une transition à un composant de votre choix.
- Expérimentez avec différentes propriétés de transition (durée, délai, etc.).
- Explorez l'utilisation de classes CSS pour personnaliser l'animation.

Récapitulatif

Composants Évolués de Vue.js

- Les slots et les composants fonctionnels étendent la flexibilité.
- La communication complexe entre composants ouvre de nouvelles possibilités.
- Les transitions et animations améliorent l'expérience visuelle.

Routage avec Vue Router - Approfondissement

Routage Avancé avec Vue Router

Exploration des Fonctionnalités Avancées du Routeur Vue

- Vue Router offre des fonctionnalités avancées pour la navigation et la gestion des transitions.

npm install vue-router

- Dans cette section, nous explorerons les routeurs imbriqués, la navigation programmée, la gestion de la navigation et les transitions entre les routes.

https://router.vuejs.org/

Routage Avancé avec Vue Router

Configuration de Vue Router 4

- Crée un fichier `router.js` à la racine du projet.
- Configure Vue Router à l'intérieur de ce fichier.

Exemple de configuration de base :

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import { createRouter, createWebHistory } from 'vue-router';

const router = createRouter({
  history: createWebHistory(),
  routes: [
    // Définissez vos routes ici
  ],
});

createApp(App).use(router).mount('#app');
```

Routeurs Imbriqués avec Vue Router

Structurer avec des Routes Imbriquées

- Les routeurs imbriqués permettent de structurer une application en sections modulaires.
- Exemple :

```
// main.js (suite)
const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
    { path: '/about', component: About },
    { path: '/contact', component: Contact },
  ],
});
```

```
routes: [
  {
    path: '/section',
    component: Section,
    children: [
      {
        path: 'sous-section',
        component: SousSection
      }
    ]
  }
]
```

Cela crée une route /section avec une sous-route /section/sous-section.

Exercice

Exercice 12: Implémentation de Routes Imbriquées

- Configurez un routeur avec une route principale et une route enfant.
- Créez des composants correspondant à ces routes.
- Testez la navigation entre les routes imbriquées.

Intégration

Utilisation des Routes dans les Composants

Utilisation des Routes dans les Liens :

```
<!-- Dans un composant -->
<router-link to="/">Accueil</router-link>
<router-link to="/about">A Propos</router-link>
<router-link to="/contact">Contact</router-link>
```

Utilisez **router-link** pour créer des liens vers vos routes.

Affichage des Composants Associés

Utilisez **router-view** pour afficher le composant associé à la route actuelle.

```
<!-- Dans un composant -->
<router-view></router-view>
```

Navigation Programmée avec Vue Router

Contrôle de la Navigation avec Vue Router

- La navigation programmée permet de changer de route en réponse à des actions utilisateur ou à des événements.

- Exemple :

```
// Naviguer vers une route
this.$router.push('/nouvelle-route');
```

```
// Dans un composant
methods: {
  redirectToAbout() {
    this.$router.push('/about');
  }
}
```

Exercice

Exercice 13: Navigation Programmée avec Vue Router

- Utilisez la navigation programmée pour changer de route en réponse à une action utilisateur (clic sur un bouton, par exemple).

- Assurez-vous que la transition entre les routes est fluide.

Paramètres

Gestion des paramètres de route

Supposons que vous ayez un composant **UserProfile.vue** qui affiche les détails d'un utilisateur, et vous souhaitez utiliser un paramètre de route pour déterminer quel utilisateur afficher.

```vue
//UserProfile.vue

<template>
  <div>
    <h2>Profil de l'utilisateur</h2>
    <p>Nom d'utilisateur: {{ $route.params.username }}</p>
    <!-- Autres détails de l'utilisateur -->
  </div>
</template>

<script>
export default {
  name: 'UserProfile',
};
</script>
```

```js
//main.js

import { createApp } from 'vue';
import App from './App.vue';
import { createRouter, createWebHistory } from 'vue-router';
import Home from './components/Home.vue';
import UserProfile from './components/UserProfile.vue';

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
    { path: '/user/:username', component: UserProfile }, // Route a
":username"
  ],
});

const app = createApp(App);
app.use(router);
app.mount('#app');
```

Avec ce code, lorsque l'URL est quelque chose comme /user/johndoe, le composant UserProfile sera rendu et le nom d'utilisateur "johndoe" sera extrait du paramètre de route :**username**. Ce paramètre est ensuite affiché dans le composant UserProfile. Vous pouvez évidemment adapter cela à vos besoins spécifiques.

Gestion de la Navigation avec Vue Router

Contrôler le Flux de Navigation

- Vue Router offre des hooks de navigation pour contrôler et personnaliser le flux de navigation.

- Exemple :

```
router.beforeEach((to, from, next) => {
  // Logique de validation ou d'autorisation
  // Appel à next() pour permettre la navigation
  next();
});
```

Exercice

Exercice 14: Utilisation des Hooks de Navigation

- Utilisez le hook beforeEach pour implémenter une logique de validation ou d'autorisation avant la navigation.
- Testez en naviguant entre les différentes routes de votre application.

Transition entre les Routes avec Vue Router

Animation et Transition entre les Routes

- Vue Router permet d'ajouter des transitions entre les routes.
- Utilisation du <router-view> avec des transitions.
- Exemple :

```
<router-view v-slot="{ Component }">
  <transition name="fade" mode="out-in">
    <component :is="Component" />
  </transition>
</router-view>
```

Cela applique une transition entre les composants lors du changement de route.

Exercice

Exercice 15: Ajout de Transitions entre les Routes

- Intégrez des transitions entre les routes en utilisant les classes CSS.
- Expérimentez avec différents effets de transition (fade, slide, etc.).
- Testez en naviguant entre les routes.

Récapitulatif

Routage Avancé avec Vue Router

- **V**ous avez exploré les fonctionnalités avancées du routage avec Vue Router.
- Routeurs imbriqués permettent une structure modulaire.
- Navigation programmée offre un contrôle dynamique sur le flux de navigation.
- Gestion de la navigation et transitions rendent l'expérience utilisateur plus agréable.

EPSI l'École d'ingénierie informatique

Fin