Cahier de corrections d'exercice - Réutilisation des Web Components cross-frameworks

> Ce document n'est qu'un premier brouillon en attente d'une relecture complète et de tests en situation réelle. Merci de me signaler toute erreur, incohérence ou manque de clarté.

Correction de l'Exercice 1 : Créer et exporter un Web Component réutilisable avec modern-web / open-wc

Étapes

1. **Initialiser le projet :**
   - Si vous utilisez `open-wc`, vous pouvez initialiser un projet avec la commande suivante :

```
npm init @open-wc
```

2. **Créer le composant `weather-widget` :**
   - Dans un fichier `weather-widget.js`, créez un Web Component qui accepte l'attribut `city` et fait une requête à l'API Open-Meteo pour récupérer la température.

```
import { html, LitElement } from 'lit';
>
  constructor() {
    super();
    this.city = '';
    this.temperature = null;
  }

  async updated(changedProperties) {
    super.updated(changedProperties);
    if (changedProperties.has('city') && this.city) {
      await this.fetchWeather();
    }
  }

  async fetchWeather() {
    try {
      const geoData = await this.getCoordinates(this.city);
      if (geoData) {
        const { latitude, longitude } = geoData;
        const weatherData = await this.getWeather(latitude, longitude);
        this.temperature = weatherData.current_weather.temperature;
      }
```

```
    } catch (error) {
      console.error('Error fetching weather:', error);
      this.temperature = 'N/A'; // Show an error message if fetching fails
    }
  }

  async getCoordinates(city) {
    const geoResponse = await fetch(`https://api.open-meteo.com/v1/geocode?city=${city}
    const geoData = await geoResponse.json();
    return geoData.results[0]; // Return coordinates of the first result
  }

  async getWeather(latitude, longitude) {
    const weatherResponse = await fetch(
      `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitud
    );
    const weatherData = await weatherResponse.json();
    return weatherData;
  }

  render() {
    return html`
      <div>
        <h3>Weather in ${this.city}</h3>
        <p>Temperature: ${this.temperature !== null ? this.temperature + '°C' : 'Loadin
      </div>
    `;
  }
}

customElements.define('weather-widget', WeatherWidget);
```

3. **Exporter le composant :**
   - Le composant `weather-widget` est exporté comme un module JavaScript, ce qui le rend réutilisable dans d'autres projets.
4. **Test du composant :**
   - Testez le composant dans un fichier HTML minimaliste en utilisant la commande suivante :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Weather Widget</title>
  </head>
```

```
  <body>
    <weather-widget city="Paris"></weather-widget>
    <script type="module" src="weather-widget.js"></script>
  </body>
</html>
```

5. **Publication du composant (optionnel) :**
   - Pour rendre le composant réutilisable dans d'autres projets, vous pouvez le publier sur npm ou via un CDN (comme un fichier statique hébergé sur un serveur).

Question

Pourquoi est-il important d'exporter le composant en tant que module JavaScript ?

- L'exportation en tant que module JavaScript permet de réutiliser le composant dans d'autres projets sans dupliquer le code. Cela permet de l'intégrer facilement dans d'autres bibliothèques ou applications, que ce soit via des imports classiques ou en l'hébergeant sur un CDN. Les modules JavaScript favorisent également l'utilisation de l'ESM (ECMAScript Modules), qui est un standard moderne permettant une gestion plus propre des dépendances et du code.

Correction de l'Exercice 2 : Créer et exporter un Web Component réutilisable avec StencilJS

Objectif

- Créer un Web Component réutilisable qui affiche la température actuelle d'une ville donnée, en utilisant l'API **Open-Meteo** (https://open-meteo.com/en/docs/meteofrance-api).
- Le composant doit accepter un attribut `city` pour spécifier la ville.
- Le composant doit être exporté en tant que module JavaScript, afin de pouvoir l'utiliser facilement dans différents projets.

Instructions

1. **Initialiser le projet StencilJS**
   - Créez un projet StencilJS en exécutant la commande :

     ```
     npm init stencil
     ```

2. **Créer le composant `weather-widget`**
   - Dans le fichier `weather-widget.tsx`, créez un Web Component qui accepte l'attribut `city` et fait une requête à l'API Open-Meteo pour récupérer la température.

3. **Exporter le composant**
    - Exportez le composant `weather-widget` en tant que module JavaScript.
4. **Test du composant**
    - Testez le composant dans un fichier HTML minimaliste en utilisant la commande suivante.
5. **Publication du composant (optionnel)**
    - Préparez le composant pour qu'il puisse être publié sur npm ou mis à disposition via un CDN.

Questions

- Quelles sont les différences principales entre un Web Component créé avec StencilJS et un Web Component créé avec `open-wc` ? Dans quel cas préférez-vous l'un par rapport à l'autre ?

Correction de l'Exercice 2 : Créer et exporter un Web Component réutilisable avec StencilJS

Étapes

1. **Initialiser le projet StencilJS** :
    - Créez un projet StencilJS en exécutant la commande suivante :

```
npm init stencil
```

2. **Créer le composant `weather-widget`** :
    - Dans le fichier `weather-widget.tsx`, créez un Web Component qui accepte l'attribut `city` et fait une requête à l'API Open-Meteo pour récupérer la température.

```
import { Component, Prop, State, h } from '@stencil/core';

@Component({
  tag: 'weather-widget',
  styleUrl: 'weather-widget.css',
  shadow: true,
})
export class WeatherWidget {
  @Prop() city: string;
  @State() temperature: string | null = null;

  async componentWillLoad() {
    await this.fetchWeather();
```

```
  }

  async fetchWeather() {
    try {
      const geoData = await this.getCoordinates(this.city);
      if (geoData) {
        const { latitude, longitude } = geoData;
        const weatherData = await this.getWeather(latitude, longitude);
        this.temperature = weatherData.current_weather.temperature;
      }
    } catch (error) {
      console.error('Error fetching weather:', error);
      this.temperature = 'N/A'; // Show an error message if fetching fails
    }
  }

  async getCoordinates(city: string) {
    const geoResponse = await fetch(`https://api.open-meteo.com/v1/geocode?city=${city}
    const geoData = await geoResponse.json();
    return geoData.results[0]; // Return coordinates of the first result
  }

  async getWeather(latitude: number, longitude: number) {
    const weatherResponse = await fetch(
      `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitud
    );
    const weatherData = await weatherResponse.json();
    return weatherData;
  }

  render() {
    return (
      <div>
        <h3>Weather in {this.city}</h3>
        <p>Temperature: {this.temperature !== null ? this.temperature + '°C' : 'Loading
      </div>
    );
  }
}
```

3. **Exporter le composant** :
   - Le composant `weather-widget` est exporté en tant que module JavaScript, ce qui le rend réutilisable dans d'autres projets.
4. **Test du composant** :
   - Testez le composant dans un fichier HTML minimaliste en utilisant

la commande suivante :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Weather Widget</title>
  </head>
  <body>
    <weather-widget city="Paris"></weather-widget>
    <script type="module" src="weather-widget.js"></script>
  </body>
</html>
```

5. **Publication du composant (optionnel)** :
   - Pour rendre le composant réutilisable dans d'autres projets, vous pouvez le publier sur npm ou via un CDN (comme un fichier statique hébergé sur un serveur).

Question

Quelles sont les différences principales entre un Web Component créé avec StencilJS et un Web Component créé avec `open-wc` ? Dans quel cas préférez-vous l'un par rapport à l'autre ?

- **StencilJS** : Permet de créer des composants hautement optimisés et produit des fichiers JavaScript modernes. Il prend en charge les `shadow DOM`, la gestion de l'état, et est plus adapté pour des projets de grande envergure ou des bibliothèques de composants.
- **open-wc** : Fournit une série d'outils et de pratiques pour développer des Web Components avec des outils comme LitElement et d'autres bibliothèques. Il est plus simple et rapide à configurer pour des projets plus petits ou des Web Components basiques.
- **Préférence** : Si vous avez besoin d'une application avec de nombreuses fonctionnalités avancées (comme l'état, les événements personnalisés, la gestion des styles), **StencilJS** peut être plus adapté. En revanche, si vous souhaitez un développement rapide et léger de Web Components réutilisables, **open-wc** est une excellente option.

Correction de l'Exercice 3 : Ajouter des tests unitaires (open-wc)

Étapes

1. **Installation de `@open-wc/testing`** :
   - Installez `@open-wc/testing` pour pouvoir utiliser les outils de testing des Web Components :

```
npm install @open-wc/testing --save-dev
```

2. **Création du fichier de test** :
   - Créez un fichier de test `weather-widget.test.js` pour tester le comportement du composant.

```
import { html, fixture } from '@open-wc/testing';
import './weather-widget.js';

describe('weather-widget', () => {
  it('affiche "Loading..." avant que les données ne soient récupérées', async () => {
    const el = await fixture(html`<weather-widget city="Paris"></weather-widget>`);
    const p = el.shadowRoot.querySelector('p');
    expect(p.textContent).to.equal('Loading...');
  });

  it('affiche la température après récupération des données', async () => {
    const el = await fixture(html`<weather-widget city="Paris"></weather-widget>`);
    await el.fetchWeather(); // Simule la récupération des données
    const p = el.shadowRoot.querySelector('p');
    expect(p.textContent).to.include('°C');
  });
});
```

3. **Lancer les tests** :
   - Vous pouvez maintenant exécuter les tests avec un framework comme Mocha ou Karma, ou utiliser simplement la commande suivante si vous avez configuré les tests avec un bundler.

```
npm run test
```

Question

Pourquoi est-il important d'écrire des tests unitaires pour vos composants ?

- Les tests unitaires sont essentiels pour vérifier que chaque composant fonctionne comme prévu, indépendamment du reste du système. Ils permettent de détecter des erreurs tôt dans le processus de développement, assurent la stabilité du code à mesure que de nouvelles fonctionnalités sont ajoutées et facilitent la maintenance du projet à long terme. > Quelles sont les bonnes pratiques en matière de tests pour les Web Components ?
- Utiliser des tests isolés pour chaque composant.
- Vérifier les interactions avec le DOM (contenu, événements).
- Simuler les comportements asynchrones (comme les requêtes API).
- Utiliser des outils comme `@open-wc/testing` pour simplifier l'écriture et l'exécution des tests.

Correction de l'Exercice 4: Documentation

Étapes

1. **Ajouter des commentaires TSDoc :**
   - Ajoutez des commentaires TSDoc au-dessus des propriétés et des méthodes du composant pour documenter leur fonctionnement.

```
/**
 * Composant pour afficher la température actuelle d'une ville.
 * @prop {string} city La ville dont on veut afficher la température.
 * @prop {string} temperature La température actuelle en °C.
 */
class WeatherWidget extends LitElement {
  /** Ville dont on veut afficher la température */
  @Prop() city: string;

  /** Température actuelle en °C */
  @State() temperature: string | null = null;

  /**
   * Méthode appelée au chargement du composant pour récupérer la température.
   */
  async componentWillLoad() {
    await this.fetchWeather();
  }
  //...
}
```

2. **Générer la documentation avec TypeDoc** :
   - Installez `TypeDoc` et générez la documentation à partir des commentaires TSDoc :

```
npm install typedoc --save-dev
npx typedoc --out docs src
```

3. **Vérifier la documentation générée** :
   - Ouvrez le dossier `docs` et vérifiez que la documentation HTML a bien été générée.

Question

Pourquoi est-il important de bien documenter ses composants ?

- Une bonne documentation permet aux autres développeurs de comprendre rapidement le fonctionnement d'un composant, de savoir comment l'utiliser et quelles sont ses dépendances. Cela facilite également la collaboration

dans une équipe de développement et permet de maintenir un code propre et bien documenté à long terme.

Comment une bonne documentation peut-elle améliorer la collaboration dans une équipe de développement ?

- Elle permet à chaque membre de l'équipe de comprendre rapidement l'objectif et les fonctionnalités des composants sans avoir à se plonger dans le code source. Une bonne documentation rend également l'intégration des nouveaux membres plus facile et assure que le projet reste compréhensible même à mesure qu'il grandit.

Correction de l'Exercice 5 : StencilJS

Étapes

1. **Initialiser un projet StencilJS** :
   - Exécutez la commande suivante pour initialiser un projet StencilJS :

```
npm init stencil
```

2. **Créer le composant `weather-widget` avec StencilJS** :
   - Dans le fichier `weather-widget.tsx`, créez un Web Component similaire à celui des exercices précédents.

```
import { Component, Prop, State, h } from '@stencil/core';

@Component({
  tag: 'weather-widget',
  styleUrl: 'weather-widget.css',
  shadow: true,
})
export class WeatherWidget {
  @Prop() city: string;
  @State() temperature: string | null = null;

  async componentWillLoad() {
    await this.fetchWeather();
  }

  async fetchWeather() {
    try {
      const geoData = await this.getCoordinates(this.city);
      if (geoData) {
        const { latitude, longitude } = geoData;
        const weatherData = await this.getWeather(latitude, longitude);
        this.temperature = weatherData.current_weather.temperature;
      }
```

```
    } catch (error) {
      console.error('Error fetching weather:', error);
      this.temperature = 'N/A';
    }
  }

  async getCoordinates(city: string) {
    const geoResponse = await fetch(`https://api.open-meteo.com/v1/geocode?city=${city}
    const geoData = await geoResponse.json();
    return geoData.results[0];
  }

  async getWeather(latitude: number, longitude: number) {
    const weatherResponse = await fetch(
      `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitud
    );
    const weatherData = await weatherResponse.json();
    return weatherData;
  }

  render() {
    return (
      <div>
        <h3>Weather in {this.city}</h3>
        <p>Temperature: {this.temperature !== null ? this.temperature + '°C' : 'Loading
      </div>
    );
  }
}
```

3. **Ajouter des tests unitaires** pour vérifier que le composant fonctionne correctement :

```
import { html, fixture } from '@open-wc/testing';
import './weather-widget.ts';

describe('weather-widget', () => {
  it('affiche "Loading..." avant que les données ne soient récupérées', async () => {
    const el = await fixture(html`<weather-widget city="Paris"></weather-widget>`);
    const p = el.shadowRoot.querySelector('p');
    expect(p.textContent).to.equal('Loading...');
  });

  it('affiche la température après récupération des données', async () => {
    const el = await fixture(html`<weather-widget city="Paris"></weather-widget>`);
    await el.fetchWeather();
    const p = el.shadowRoot.querySelector('p');
```

```
    expect(p.textContent).to.include('°C');
  });
});
```

4. **Documenter le composant avec TSDoc** :
   - Ajoutez des commentaires TSDoc pour documenter le composant et ses propriétés.

```
/**
 * Composant pour afficher la température actuelle d'une ville.
 * @prop {string} city La ville dont on veut afficher la température.
 * @prop {string} temperature La température actuelle en °C.
 */
class WeatherWidget {
  /** Ville dont on veut afficher la température */
  @Prop() city: string;

  /** Température actuelle en °C */
  @State() temperature: string | null = null;

  /**
   * Méthode appelée au chargement du composant pour récupérer la température.
   */
  async componentWillLoad() {
    await this.fetchWeather();
  }
  //...
}
```

5. **Exporter le composant pour qu'il soit réutilisable** :
   - Le composant sera automatiquement exporté en tant que module JavaScript, comme dans les exercices précédents.

6. **Tester et valider le composant** dans un fichier HTML minimaliste en utilisant la commande suivante :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Weather Widget</title>
  </head>
  <body>
    <weather-widget city="Paris"></weather-widget>
    <script type="module" src="weather-widget.js"></script>
  </body>
</html>
```

Questions

Quelles sont les différences entre StencilJS et `open-wc` en termes de fonctionnalités et de configuration ? Pourquoi choisir l'un ou l'autre ?

- **StencilJS** : Permet de créer des composants hautement optimisés avec des fonctionnalités avancées comme le shadow DOM, la gestion de l'état, et la génération de composants prêts pour la production avec un minimum de configuration. Il est particulièrement adapté pour des applications complexes et des bibliothèques de composants.
- **open-wc** : Fournit une approche plus légère et modulaire pour la création de Web Components, basée sur des pratiques courantes et des outils comme LitElement. Il est plus simple et rapide à configurer pour des projets plus petits ou des composants basiques.
- **Choix entre les deux** : Si vous avez besoin de fonctionnalités avancées et d'une configuration automatisée, **StencilJS** est le bon choix. Pour un projet plus simple ou pour des composants qui n'ont pas besoin de toute la complexité de Stencil, **open-wc** est plus adapté.

Correction de l'Exercice 6 : A,gular

Étapes

1. **Initialiser un projet Angular** :
   - Créez un projet Angular minimaliste en exécutant la commande suivante :

   ```
   ng new weather-app
   ```

2. **Configurer Angular pour accepter les Web Components** :
   - Déclarez `CUSTOM_ELEMENTS_SCHEMA` dans le module Angular pour accepter les éléments non natifs.

   ```
   import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
   import { BrowserModule } from '@angular/platform-browser';
   import { AppComponent } from './app.component';

   @NgModule({
     declarations: [AppComponent],
     imports: [BrowserModule],
     schemas: [CUSTOM_ELEMENTS_SCHEMA], // Ajouter ceci
     bootstrap: [AppComponent],
   })
   export class AppModule {}
   ```

3. **Intégrer le Web Component `weather-widget` dans Angular** :

- Importez le fichier `weather-widget.js` dans le fichier `index.html` de votre projet Angular.
- Utilisez le composant Web dans le template Angular.

```
<!-- index.html -->
<script type="module" src="path/to/weather-widget.js"></script>
<!-- Ajouter cette ligne -->

<!-- app.component.html -->
<weather-widget city="Paris"></weather-widget>
```

4. **Test du projet** :
   - Lancez le projet Angular avec la commande suivante et vérifiez que le Web Component fonctionne correctement dans l'application.

   ```
   ng serve
   ```

Question

Pourquoi Angular nécessite-t-il la déclaration de `CUSTOM_ELEMENTS_SCHEMA` ?

- `CUSTOM_ELEMENTS_SCHEMA` permet à Angular de reconnaître et de gérer des éléments HTML non natifs, comme les Web Components, sans générer d'erreurs. Par défaut, Angular s'attend à ce que les éléments dans les templates correspondent à des composants Angular. Cette configuration informe Angular qu'il peut accepter et gérer des éléments personnalisés (comme les Web Components) dans le DOM, ce qui permet leur utilisation dans les templates sans conflit.

Correction de l'Exercice 7

Étapes

1. **Initialiser un projet Angular** :
   - Si vous n'avez pas déjà initialisé un projet Angular, suivez les étapes de l'Exercice 6 pour le faire.
2. **Importer les Web Components** :
   - Ajoutez les fichiers des composants `weather-widget.js` et `weather-widget.stencil.js` dans le dossier de votre projet Angular.
   - Importez les fichiers dans `index.html` de votre projet Angular.

   ```
   <script type="module" src="path/to/weather-widget.js"></script>
   <script type="module" src="path/to/weather-widget.stencil.js"></script>
   ```bash
   ```
3. **Exporter les composants** :
   ```

- Ces composants sont automatiquement exportés en tant que modules JavaScript et peuvent être importés dans d'autres projets ou bibliothèques.

4. **Test du projet** :
   - Testez l'intégration des composants dans `app.component.html` en utilisant les deux composants avec des villes différentes.

```
<!-- app.component.html -->
<weather-widget city="Paris"></weather-widget>
<weather-widget city="London"></weather-widget>
```bash
```

5. **Lancer l'application** :
   - Utilisez la commande `ng serve` pour lancer l'application et tester les composants.

Question

Quelles difficultés avez-vous rencontrées lors de l'intégration des deux Web Components dans Angular ?

- Une difficulté courante est l'importation correcte des fichiers de composants dans Angular, en particulier lorsqu'il s'agit de composants créés avec des outils différents comme `open-wc` et `StencilJS`. Pour résoudre ce problème, nous avons utilisé des imports dynamiques dans `index.html` et configuré correctement `CUSTOM_ELEMENTS_SCHEMA` pour permettre à Angular de reconnaître les Web Components.

Correction de l'Exercice 8

Étapes

1. **Créer un composant Angular** :
   - Créez un composant Angular avec les mêmes fonctionnalités que dans l'Exercice 6, en affichant la température de la ville.

2. **Convertir le composant Angular en Angular Element** :
   - Utilisez `@angular/elements` pour transformer le composant en Angular Element.

```
import { Component, Injector, Input, OnInit } from '@angular/core';
import { createCustomElement } from '@angular/elements';
import { NgModule } from '@angular/core';

@Component({
  selector: 'app-weather-widget',
```
```

```
  template: `<h3>Weather in {{ city }}</h3><p>{{ temperature }}°C</p>`,
})
export class WeatherWidgetComponent implements OnInit {
  @Input() city: string;
  temperature: number;

  ngOnInit() {
    // Logic for fetching weather data
  }
}

@NgModule({
  declarations: [WeatherWidgetComponent],
  imports: [],
  entryComponents: [WeatherWidgetComponent],
})
export class WeatherWidgetModule {
  constructor(private injector: Injector) {}

  ngDoBootstrap() {
    const el = createCustomElement(WeatherWidgetComponent, { injector: this.injector })
    customElements.define('weather-widget', el);
  }
}
```bash
```

3. **Exporter le Web Component** :
   - Exposez le composant Angular comme un Web Component en l'exportant sous forme de module JavaScript via la méthode `createCustomElement`.
4. **Test du projet** :
   - Testez l'intégration de l'Angular Element dans un projet HTML minimaliste pour vous assurer qu'il fonctionne comme un Web Component classique.

Question

Quels sont les avantages de l'utilisation d'Angular Elements par rapport aux Web Components classiques ?

- Les **Angular Elements** offrent une intégration plus facile avec des applications Angular existantes et permettent de convertir des composants Angular en Web Components sans avoir à tout réécrire. L'avantage principal est la possibilité de réutiliser la logique d'Angular tout en offrant un composant compatible avec n'importe quel environnement basé sur le Web.
```

Correction de l'Exercice 9

Étapes

1. **Initialiser un projet React minimaliste** :
   - Exécutez la commande suivante pour initialiser un projet React :

```
npx create-react-app weather-app
```bash
```

2. **Importer le Web Component** :
   - Ajoutez le fichier `weather-widget.js` dans le dossier public de votre projet React.
   - Assurez-vous que le fichier est bien importé dans `index.html`.

```
<script type="module" src="path/to/weather-widget.js"></script>
```bash
```

3. **Utiliser le Web Component dans React** :
   - Utilisez le Web Component dans le JSX d'un composant React comme suit :

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Weather App</h1>
      <weather-widget city="Paris"></weather-widget>
    </div>
  );
}

export default App;
```bash
```

4. **Test du projet** :
   - Lancez le projet React avec `npm start` et vérifiez que le Web Component fonctionne correctement dans l'application.

Question

Quelle est la différence entre l'intégration de Web Components dans un projet React et l'intégration dans un projet Angular ?

- Dans React, vous pouvez intégrer un Web Component comme un élément HTML natif dans JSX. Cependant, React nécessite une gestion spécifique des événements et des propriétés (par exemple, en utilisant
```

`addEventListener` pour les événements personnalisés). En revanche, Angular gère nativement les Web Components via `CUSTOM_ELEMENTS_SCHEMA`, et la gestion des événements est plus intégrée dans son système. React nécessite donc un peu plus de configuration manuelle pour l'intégration des Web Components.

Correction de l'Exercice 10

Étapes

1. **Initialiser un projet React minimaliste** :
   - Si vous n'avez pas encore initialisé le projet, suivez les étapes de l'Exercice 9 pour le faire.
2. **Importer les Web Components** :
   - Assurez-vous que les fichiers `weather-widget.js` et `weather-widget.stencil.js` sont ajoutés à votre projet React.
   - Vous pouvez les importer dans `index.html` ou utiliser des imports dynamiques pour charger les composants.

```
<script type="module" src="path/to/weather-widget.js"></script>
<script type="module" src="path/to/weather-widget.stencil.js"></script>
```

3. **Utiliser les Web Components dans React** :
   - Dans le fichier `App.js`, intégrez les deux Web Components avec des villes différentes :

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Weather App</h1>
      <weather-widget city="Paris"></weather-widget>
      <weather-widget city="London"></weather-widget>
    </div>
  );
}

export default App;
```

4. **Test du projet** :
   - Lancez l'application avec `npm start` et vérifiez que les deux Web Components affichent correctement la température pour les villes spécifiées.

Question

Quels sont les défis que vous avez rencontrés lors de l'intégration de Web Components dans un projet React et comment les avez-vous résolus ?

- L'intégration des Web Components dans React peut poser des problèmes de gestion des événements et des propriétés. React gère les événements de manière différente des Web Components natifs, il est donc nécessaire d'utiliser `addEventListener` pour gérer les événements personnalisés émis par les Web Components. De plus, certaines incompatibilités de style peuvent apparaître, notamment si des styles globaux de React affectent l'apparence du Web Component, ce qui peut être résolu en utilisant le Shadow DOM ou en isolant les styles des Web Components.

Correction de l'Exercice 11

Étapes

1. **Initialiser un projet Next.js** :
   - Exécutez la commande suivante pour initialiser un projet Next.js :

```
npx create-next-app weather-app
```

2. **Importer le Web Component** :
   - Ajoutez le fichier `weather-widget.js` dans le dossier public de votre projet Next.js.
   - Assurez-vous que le fichier est bien importé dans `pages/_document.js`.

```
<script type="module" src="path/to/weather-widget.js"></script>
```

3. **Utiliser le Web Component dans Next.js** :
   - Intégrez le Web Component dans le fichier `pages/index.js` :

```
import React from 'react';

function Home() {
  return (
    <div>
      <h1>Weather App</h1>
      <weather-widget city="Paris"></weather-widget>
    </div>
  );
}

export default Home;
```

4. **Test du projet** :

- Lancez le projet Next.js avec `npm run dev` et vérifiez que le Web Component fonctionne correctement.

Question

Quelle est la différence entre l'intégration de Web Components dans un projet Next.js et dans un projet React ?

- Dans **Next.js**, l'intégration de Web Components nécessite souvent un peu plus de configuration au niveau de la gestion des fichiers externes et de leur importation dans `_document.js`. Next.js étant un framework basé sur le rendu côté serveur (SSR), il faut parfois gérer l'importation dynamique des Web Components pour s'assurer qu'ils ne sont rendus que côté client. En revanche, dans **React**, l'intégration des Web Components est plus directe grâce à la flexibilité du JSX, bien qu'il faille gérer les événements et propriétés différemment.

Correction de l'Exercice 12

Étapes

1. **Initialiser un projet Vue.js** :
   - Si vous utilisez **Vue 3**, vous pouvez créer un projet avec la commande suivante :

   ```
   npm install vue@next
   ```

2. **Importer le Web Component** :
   - Ajoutez le fichier `weather-widget.js` dans le dossier `public` de votre projet Vue.js.
   - Importez le fichier dans `index.html` ou utilisez un import dynamique.

   ```
   <script type="module" src="path/to/weather-widget.js"></script>
   ```

3. **Utiliser le Web Component dans Vue.js** :
   - Utilisez le Web Component dans le template Vue.js comme suit :

   ```
   <template>
     <div>
       <weather-widget city="Paris"></weather-widget>
     </div>
   </template>

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

4. **Test du projet** :
    - Lancez le projet Vue.js avec la commande `npm run serve` et vérifiez que le Web Component fonctionne correctement.

Question

> Quels sont les principaux avantages et inconvénients de l'intégration des Web Components dans Vue.js par rapport aux autres frameworks ?

- **Avantages :** Vue.js est facile à configurer pour l'utilisation des Web Components grâce à son système de composants flexible. Les Web Components peuvent être intégrés dans Vue.js sans avoir besoin d'outils supplémentaires comme dans Angular.
- **Inconvénients :** Vue.js a son propre système de gestion des événements et des propriétés, ce qui peut rendre l'intégration des événements personnalisés des Web Components plus complexe. De plus, la gestion du DOM et du cycle de vie peut entraîner des conflits avec l'architecture de Vue.js si le Web Component n'est pas correctement configuré pour fonctionner de manière autonome.