Réutilisation des Web Components cross-frameworks

Objectifs de la matinée

1. Comprendre comment développer des Web Components réutilisables dans différents projets et frameworks.
2. Découvrir les méthodes pour intégrer des Web Components dans Angular, React, Vue.js, et d'autres environnements.
3. Explorer comment encapsuler le code Angular avec Angular Elements.
4. Créer et utiliser des Web Components avec React et Preact.
5. Comprendre les opportunités et les limites des Web Components dans des applications modernes.

Développement de Web Components “cross-plateformes/cross-projet”

Pourquoi développer des Web Components “cross-projet” ?

1. **Réutilisabilité :** Les Web Components permettent de partager des éléments UI entre projets sans dépendance à un framework spécifique.
2. **Interopérabilité :** Fonctionnent nativement dans tous les navigateurs modernes et peuvent être utilisés avec Angular, React, Vue.js, etc.
3. **Standardisation :** Utilisent les standards du DOM natif, garantissant la pérennité du code.
4. **Indépendance des outils :** Pas besoin de lier des frameworks ou bibliothèques spécifiques pour les utiliser.

Étapes pour développer un Web Component réutilisable

1. Créer un Web Component en suivant les normes des standards ouverts.
2. S'assurer qu'il n'y a pas de dépendances spécifiques à un framework ou outil particulier.
3. Tester le composant dans différents environnements pour garantir sa compatibilité.
4. Documenter clairement les propriétés, événements et méthodes exposées.

Exemple d'un Web Component simple

Pour illustrer, nous allons créer un composant `my-alert` qui affiche un message et utilise une propriété personnalisée `message`.

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

@Component({
  tag: 'my-alert',
  shadow: true,
})
export class MyAlert {
  @Prop() message: string;

  render() {
    return <div>{this.message}</div>;
  }
}
```

Points à surveiller lors de la création de Web Components

1. **Isolation des styles :** Utilisez le Shadow DOM pour éviter les conflits avec le reste de l'application.
2. **Interopérabilité :** Évitez les dépendances non standardisées (exemple : API spécifiques à un framework).
3. **Compatibilité :** Testez les composants dans des navigateurs variés et avec des frameworks populaires.
4. **Accessibilité :** Assurez-vous que les composants respectent les bonnes pratiques d'accessibilité (ARIA, clavier, etc.).

Documenter un Web Component avec TSDoc

Pourquoi utiliser TSDoc ?

1. **Clarté du code :** Les développeurs comprennent facilement les propriétés, événements et méthodes des composants.
2. **Standardisation :** TSDoc est basé sur des normes bien établies pour la documentation TypeScript.
3. **Génération automatique :** Facilite la création de documentation HTML ou PDF à partir du code source.

Exemple de documentation avec TSDoc

```
/**
 * Un composant d'alerte personnalisable.
 * @tag my-alert
 */
@Component({
  tag: 'my-alert',
  shadow: true,
})
export class MyAlert {
  /**
```

Introduction à Storybook pour les Web Components

Qu'est-ce que Storybook ?

1. **Outil open-source :** Storybook est un environnement de développement pour construire des composants UI de manière isolée.
2. **Support natif des Web Components :** Fonctionne avec les Web Components vanilla, sans dépendre d'un framework particulier.
3. **Personnalisation interactive :** Permet de tester différentes variantes des composants en modifiant leurs propriétés en temps réel.

Pourquoi utiliser Storybook ?

1. **Documentation interactive :**
   - Génère des aperçus interactifs des composants.
   - Utile pour partager une bibliothèque de composants avec une équipe.
2. **Tests visuels :**
   - Vérifiez rapidement les rendus et les interactions de vos composants.
3. **Amélioration de la collaboration :**
   - Permet aux designers et développeurs de travailler ensemble en visualisant les composants isolément.

Étapes pour utiliser Storybook avec les Web Components

1. **Installation :** Ajoutez Storybook à votre projet.
2. **Configuration :** Configurez le support des Web Components vanilla.
3. **Création de stories :** Créez des fichiers `*.stories.js` pour définir les cas d'utilisation.

Étape 1 : Installer Storybook

```
# Installez Storybook pour les Web Components
npx sb init --type web_components
```

Étape 2 : Configurer le projet

1. Storybook génère automatiquement un dossier `.storybook`.
2. Modifiez `main.js` pour spécifier que vous utilisez des Web Components.

```
// Fichier de configuration Storybook
module.exports = {
  stories: ['../src/**/*.stories.js'],
  addons: [],
  framework: '@storybook/web-components',
};
```

Étape 3 : Créer une story pour un Web Component

1. Les stories permettent de documenter et visualiser vos composants.
2. Créez un fichier `my-alert.stories.js` pour le composant `my-alert`.

```
// Exemple de story pour un Web Component
export default {
  title: 'MyAlert',
};

export const Default = () => `
  <my-alert message="Ceci est une alerte."></my-alert>
`;
```

Points d'attention avec Storybook

1. **Performances :** Assurez-vous que vos composants sont optimisés pour éviter les ralentissements dans Storybook.
2. **Interopérabilité :** Testez vos composants dans Storybook avec différentes configurations de navigateurs pour garantir leur compatibilité.
3. **Collaboration :** Utilisez l'interface interactive pour recueillir des retours des membres de l'équipe (design, QA, etc.).

Présentation des outils modern-web.dev

Histoire et raison d'être

- **Création :** modern-web.dev est une initiative open-source lancée pour répondre aux besoins des développeurs travaillant avec des standards modernes du Web.
- **Objectif :** Proposer des outils simples, performants et compatibles avec les ES modules natifs, sans dépendances inutiles.
- **Philosophie :** Faciliter le développement et les tests des applications web en exploitant au maximum les standards natifs.

Web Dev Server

Description

1. **Qu'est-ce que c'est ?**
   - Un serveur de développement léger et rapide conçu pour les ES modules.
   - Utilisé pour servir les fichiers directement au navigateur sans étape de bundling complexe.
2. **Fonctionnalités clés :**
   - Support des ES modules natifs.
   - Reload à chaud pour une expérience de développement fluide.
   - Fonctionne sans configuration pour les projets basés sur les standards modernes.

Exemple de commande

```
# Installation de Web Dev Server
npm install @web/dev-server --save-dev

# Démarrage du serveur
npx web-dev-server --open --watch
```

Testing Library

Description

1. **Qu'est-ce que c'est ?**
   - Une bibliothèque conçue pour tester les Web Components et autres éléments natifs.
   - Fournit des outils pour interagir avec le DOM et valider les comportements.
2. **Pourquoi l'utiliser ?**
   - Simplifie l'écriture de tests unitaires et fonctionnels pour les Web Components.
   - Basé sur une approche centrée sur l'utilisateur, inspirée de `@testing-library/react`.

Exemple de test

```
// Exemple de test avec Testing Library
import { fixture, expect } from '@open-wc/testing';

it('affiche le message correct', async () => {
  const el = await fixture('<my-alert message="Hello"></my-
  expect(el.message).to.equal('Hello');
});
```

Dev Tools

Description

1. **Qu'est-ce que c'est ?**
   - Une collection d'outils pour le développement avec les standards Web.
   - Inclut des fonctionnalités pour le debugging et la performance.
2. **Fonctionnalités clés :**
   - Aide au debugging des ES modules.
   - Analyse des performances des applications basées sur les standards modernes.

Présentation des outils open-wc.org

Histoire et raison d'être

- **Création :** open-wc.org est une initiative de la communauté pour normaliser et simplifier la création de Web Components.
- **Objectif :** Fournir des recommandations, des outils, et des bonnes pratiques pour les développeurs travaillant avec les Web Components.
- **Philosophie :** Rendre les Web Components accessibles et faciles à adopter grâce à des modèles et des outils adaptés.

Starter Kit

Description

1. **Qu'est-ce que c'est ?**
   - Un modèle de projet prêt à l'emploi pour démarrer rapidement avec les Web Components.
   - Basé sur des outils modernes comme Rollup et Web Dev Server.
2. **Pourquoi l'utiliser ?**
   - Fournit une structure de projet standardisée.
   - Permet de démarrer rapidement sans configuration complexe.

Exemple d'utilisation

```
# Créez un nouveau projet avec le Starter Kit
npm init @open-wc
```

Linting et Testing

Description

1. **Qu'est-ce que c'est ?**
   - Des outils pour garantir la qualité du code et tester les Web Components.
   - Inclut des configurations prêtes à l'emploi pour ESLint et Testing Library.
2. **Pourquoi l'utiliser ?**
   - Détecte les erreurs courantes dans les Web Components.
   - Facilite l'intégration de tests dans les workflows CI/CD.

Exemple de configuration ESLint

```
// Exemple de configuration ESLint
module.exports = {
  extends: ['@open-wc/eslint-config'],
};
```

Documentation

Description

1. **Qu'est-ce que c'est ?**
   - Des guides et tutoriels pour apprendre les bonnes pratiques des Web Components.
   - Inclut des exemples détaillés pour différents cas d'utilisation.
2. **Pourquoi l'utiliser ?**
   - Idéal pour les développeurs débutants ou expérimentés souhaitant perfectionner leur maîtrise des Web Components.
   - Fournit des solutions aux problèmes courants dans les projets basés sur les standards modernes.

Mauvaises pratiques dans le développement de Web Components

Isolation des styles

- **Problème :** Styles globaux affectant les Web Components ou styles internes fuyant vers le reste de l'application.
- **Solution :** Utiliser le Shadow DOM pour encapsuler les styles.

Interopérabilité et compatibilité

- **Problème :** Utilisation de dépendances non standard ou API incompatibles avec certains navigateurs.
- **Solution :** S'en tenir aux standards ouverts et tester les composants avec des polyfills si nécessaire.

Accessibilité

- **Problème :** Absence de support pour la navigation au clavier ou les lecteurs d'écran.
- **Solution :** Respecter les bonnes pratiques ARIA et tester les composants avec des outils d'accessibilité.

Intégration d'un Web Component dans un projet Angular

Pourquoi utiliser des Web Components avec Angular ?

1. **Interopérabilité :** Angular permet d'utiliser des Web Components comme n'importe quel élément HTML standard.
2. **Réduction des dépendances :** Permet d'introduire des fonctionnalités spécifiques sans ajouter de bibliothèques Angular supplémentaires.
3. **Réutilisabilité :** Les mêmes composants peuvent être utilisés dans d'autres projets ou frameworks.

Étapes pour intégrer un Web Component dans Angular

1. Ajouter le Web Component dans le projet Angular (copie des fichiers ou utilisation d'un package npm).
2. Déclarer le `CUSTOM_ELEMENTS_SCHEMA` dans le module Angular pour autoriser les éléments non natifs.
3. Ajouter le Web Component dans le HTML du composant Angular.

Exemple d'intégration dans Angular

```
// Importer les schémas Angular pour les Web Components
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

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

Explications

- **CUSTOM_ELEMENTS_SCHEMA :** Permet à Angular de reconnaître et d'utiliser les Web Components sans générer d'erreurs.
- **Bootstrap avec AppComponent :** La structure classique d'une application Angular est préservée, même avec des Web Components.

Points d'attention lors de l'intégration dans Angular

1. Vérifiez que les Web Components sont correctement chargés avant leur utilisation (via les scripts ou imports nécessaires).
2. Assurez-vous que les événements personnalisés déclenchés par le composant sont bien capturés dans Angular (via `@Output()` ou les gestionnaires d'événements natifs).

Intégration d'un Web Component dans un projet React

Étapes pour intégrer un Web Component dans React

1. Installer et importer le Web Component dans votre projet React.
2. Utiliser le Web Component comme un élément HTML natif dans les composants React.
3. Gérer les événements personnalisés et les propriétés du Web Component dans React.

Exemple d'intégration dans React

```
// Exemple d'utilisation d'un Web Component dans React
import React from 'react';

function App() {
  return (
    <div>
      <my-alert message="Hello, React!"></my-alert>
    </div>
  );
}

export default App;
```

Explications

- **Utilisation de Web Components dans JSX :** React permet l'utilisation d'un Web Component directement dans JSX comme s'il s'agissait d'un élément HTML.
- **Gestion des propriétés :** Les propriétés sont transmises comme attributs dans JSX et sont gérées de manière réactive dans le Web Component.

Points d'attention lors de l'intégration dans React

1. Vérifiez que les Web Components sont chargés avant d'être utilisés dans React.
2. Gérez les événements personnalisés à l'aide de `addEventListener()` dans React, car les Web Components peuvent utiliser des événements non traditionnels.
3. Testez les Web Components pour garantir qu'ils n'interfèrent pas avec le système de gestion d'état ou le cycle de vie des composants React.

Intégration d'un Web Component dans un projet React, Next.js et Remix

Pourquoi utiliser des Web Components avec React, Next.js et Remix ?

1. **Web Components dans React** :
   - Utilisation traditionnelle des Web Components dans des applications React.
   - Compatibilité native avec JSX.
2. **Web Components dans Next.js et Remix** :
   - Ces frameworks offrent un rendu côté serveur (SSR) et un routage intégré, tout en permettant l'utilisation de composants Web pour une interopérabilité maximale.
   - Les Web Components s'intègrent bien dans l'architecture de ces frameworks, particulièrement pour des applications modernes nécessitant du rendu statique ou dynamique.

Étapes pour intégrer un Web Component dans Next.js ou Remix

1. **Créer et publier un Web Component** comme expliqué précédemment.
2. **Installer et importer le Web Component** dans votre projet Next.js ou Remix.
3. Utilisez le Web Component dans les composants React comme un élément HTML natif.
4. Assurez-vous que les Web Components sont chargés côté serveur (dans le cas de Next.js et Remix) pour éviter tout problème de SSR.

Exemple d'intégration dans Next.js

```
// Importation du Web Component dans un projet Next.js
import React from 'react';

const HomePage = () => {
  return (
    <div>
      <my-alert message="Hello from Next.js!"></my-alert>
    </div>
  );
};

export default HomePage;
```

Exemple d'intégration dans Remix

```
// Importation du Web Component dans Remix
import { useEffect } from 'react';

const HomePage = () => {
  useEffect(() => {
    // Assurez-vous que le Web Component est chargé avant u
    import('my-alert');
  }, []);

  return (
    <div>
      <my-alert message="Hello from Remix!"></my-alert>
    </div>
  );
};

export default HomePage;
```

Explications

- Next.js et Remix prennent en charge le rendu côté serveur. Lors de l'utilisation de Web Components dans ces frameworks, il est important de s'assurer qu'ils sont correctement rendus côté client après le SSR.
- L'importation dynamique des Web Components dans Remix (avec `useEffect`) ou Next.js permet de les charger uniquement côté client, évitant ainsi des problèmes de rendu côté serveur.

Points d'attention lors de l'intégration dans Next.js ou Remix

1. **Rendu côté serveur (SSR) :** Testez vos Web Components pour garantir qu'ils sont correctement gérés lors du SSR, notamment en vérifiant qu'ils sont chargés après le rendu côté client.
2. **Importation dynamique :** Utilisez `React.lazy()` ou `useEffect()` pour importer dynamiquement vos Web Components dans Next.js ou Remix, afin de gérer leur chargement côté client.
3. **Optimisation des performances :** Les Web Components peuvent être lourds si mal optimisés, assurez-vous de ne charger que ce qui est nécessaire et d'éviter le double chargement dans les deux environnements (SSR et client).

Web Components et React Server Components

Introduction aux React Server Components

1. **Qu'est-ce que c'est ?**
   - Les React Server Components (RSC) permettent de rendre une partie de l'UI côté serveur tout en maintenant une expérience utilisateur réactive côté client.
2. **Compatibilité avec les Web Components :**
   - Les Web Components peuvent être intégrés dans des applications React Server Components pour fournir des éléments réutilisables et performants tout en réduisant le JavaScript envoyé au client.

Intégration d'un Web Component dans React Server Components

1. **Utilisation côté serveur :** React Server Components peuvent rendre des Web Components côté serveur, mais ces derniers ne sont pas toujours compatibles avec le rendu serveur de React.
2. **Assurez-vous que les Web Components sont correctement rendus et chargés côté client**.

Exemple d'utilisation avec React Server Components

```
// Exemple d'utilisation avec React Server Components
import { Suspense } from 'react';

export default function Page() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <my-alert message="Rendered with React Server Compone
    </Suspense>
  );
}
```

Explications

- **Suspense et React Server Components :** Le composant `Suspense` est utilisé pour gérer les éléments de l'UI qui sont chargés de manière asynchrone, comme les Web Components.
- **Chargement côté client et serveur :** Assurez-vous que les Web Components sont rendus côté serveur sans perdre leur comportement interactif lorsqu'ils sont rendus côté client.

Intégration d'un Web Component dans un

Exemple d'intégration dans Vue.js

```
// Exemple d'utilisation d'un Web Component dans Vue.js
<template>
  <div>
    <my-alert :message="'Hello from Vue.js!'"></my-alert>
  </div>
</template>

<script>
export default {
  mounted() {
    // Vous pouvez importer dynamiquement le Web Component
    import('my-alert');
  },
};
</script>
```

Explications

- **Utilisation dans le template Vue.js :** Vous pouvez utiliser le Web Component comme n'importe quel autre élément HTML natif, en passant des propriétés via des attributs et en écoutant des événements personnalisés.
- **Importation dynamique :** Dans l'exemple, le Web Component est importé dynamiquement dans la méthode `mounted()`, ce qui garantit qu'il est chargé une fois le composant Vue.js monté.

Points d'attention lors de l'intégration dans Vue.js

1. **Gestion des événements :** Vue.js utilise un système d'événements basé sur la gestion des props et des événements via des directives. Assurez-vous que vos événements personnalisés sont bien gérés en utilisant `addEventListener` si nécessaire.
2. **Utilisation du cycle de vie de Vue.js :** Lorsque vous travaillez avec des Web Components, il peut être nécessaire de gérer leur cycle de vie dans des hooks comme `mounted()` ou `updated()` pour s'assurer que les composants sont bien initialisés et actualisés.
3. **Compatibilité avec les autres versions de Vue :** Vérifiez la compatibilité de vos Web Components avec Vue 2.x et 3.x, car il peut y avoir des différences dans la gestion du DOM et des composants.

Conclusion du chapitre

1. **Web Components cross-frameworks :** Les Web Components permettent de créer des éléments réutilisables dans des projets utilisant des frameworks modernes comme Angular, React, Vue.js, et d'autres.
2. **Interopérabilité :** Ils offrent une solution idéale pour intégrer des composants dans des projets existants sans dépendre d'un framework spécifique.
3. **Meilleures pratiques :** Assurez-vous de respecter les bonnes pratiques liées à l'**isolation des styles**, l'**interopérabilité**, et l'**accessibilité** pour garantir une expérience optimale et une maintenance facile.