LES WEB COMPONENTS

ATELIER WAIGÉO N°1
25 MAI 2018

Le futur c'est maintenant avec les composants web natifs

SOMMAIRE

- Présentation des Web Components
- Introduction à Stencil
- Stencil API
- Demo
- Ressources

PRÉLUDE

BUILDING UI AT ENTERPRISE SCALE WITH WEB COMPONENTS BY EA SPORTS

Cette présentation est plus une prise de conscience de l'impact des web components sur le développement web d'aujourd'hui et surtout de demain. Pour comprendre les bénéfices de cette technologie dans le milieu professionnel, il n'y a qu'une seule vidéo à voir, un must have, même si cela utilise une autre technologie que celle que je vais décrire dans la deuxième partie Cas d'utilisation en entreprise par EA Sports (polymer)

WEB COMPONENTS

- Composants JavaScript normalisés pris en charge de manière native
- S'exécute dans n'importe quel framework ou par son propre chef
- Répond au problème des composants partagés
- Propulsé par la spécification Custom Elements v1

Les Web Components est une suite de technologies connectées permettant de créer des éléments personnalisés réutilisables - avec leurs fonctionnalités encapsulées à l'écart du reste de votre code pour les utiliser dans vos applications Web.
L'essentiel des discussions a tourné autour du DOM fantôme (shadow DOM), mais la technologie qui apportera sans doute le plus de transformations est ce qu'on appelle les custom elements (les éléments personnalisés), une méthode vous permettant de définir vos propres éléments, avec leurs comportements et propriétés.

4 TECHNOLOGIES REPRÉSENTENT LES WEB COMPONENTS

- Custom Elements
- Shadow DOM
- HTML Templates
- HTML Imports

4 technologies normées derrière ce nom

CUSTOM ELEMENTS

Crée des éléments personnalisés qui encapsulent vos fonctionnalités sur une page HTML

VS

Se contenter de spaghettis à la sauce balises HTML définissant vos fonctionnalités

L'un des aspects les plus importants des composants web est la possibilité de créer des éléments personnalisés qui encapsulent correctement vos fonctionnalités sur une page HTML, plutôt que de devoir se contenter d'une soupe de balises définissant des fonctionnalités personnalisées.
Cette technologie vous permet de définir vos propres éléments, avec leurs comportements et propriétés, émettant ou réagissant à des événements personnalisés à travers la page

SHADOW DOM

permet d'encapsuler du JavaScript et du CSS au sein d'un Web Component afin que ces éléments soient séparés du DOM du document principal

Le Shadow DOM ou DOM fantôme permet d'encapsuler du JavaScript et du CSS au sein d'un Web Component. Le Shadow DOM fait en sorte que ces éléments soit séparés du DOM du document principal.
Le Shadow DOM est utilisable seul, en dehors d'un Web component.
Pourquoi rechercher à séparer le code du reste de la page ?
Une des raisons est que sur des sites de tailles importantes, par exemple, si le CSS n'est pas correctement organisé, les styles de certains composants peuvent impacter d'autres parties du site bien que ce ne soit pas prévu, et vice-versa.
Quand un site ou une application grandit, ce genre de chose est difficile a éviter.

HTML TEMPLATES

met à disposition les éléments <template> et <slot> pour créer un modèle flexible de fragments de contenu, côté client, qui peut ensuite être utilisée pour remplir le shadow DOM d'un composant Web

Au chargement de la page, le contenu de chaque balise <template> n'est pas affiché mais peut être instancié par la suite via JavaScript

L'élément HTML <template> (ou Template Content ou modèle de contenu) est un mécanisme utilisé pour stocker du contenu côté client et qui ne doit pas être affiché lors du chargement de la page mais qui peut être instancié par la suite via JavaScript. Cet élément est un fragment de contenu mis de côté pour être utilisé par la suite dans le document. Lorsque le moteur traite le contenu de l'élément <template> lors du chargement de la page, il ne fait que vérifier la validité du contenu, ce dernier n'est pas affiché

HTML IMPORTS

est conçue à l'origine pour être le mécanisme d'empaquetage des composants Web

Mais est également utilisée pour importer un fichier HTML seul en utilisant une balise <link>

```
<link rel="import" href="myfile.html" />
```

Les importations HTML sont conçues pour être le mécanisme de packaging des composants Web, mais vous pouvez également utiliser les importations HTML seules. HTML imports est un peu particulier car toujours en état Working Draft et est un peu controversé par certains grandes entreprises. Vous importez un fichier HTML en utilisant une balise link dans un document HTML comme celui-ci

NAVIGATEURS COMPATIBLES

- Support natif : Chrome, Safari, Opéra
- Scripts polyfills performant
- Firefox très proche (version 60/61)

Browser support	CHROME	OPERA	SAFARI	FIREFOX	EDGE
TEMPLATES	STABLE	STABLE	STABLE	STABLE	STABLE
CUSTOM ELEMENTS	STABLE	STABLE	STABLE	POLYFILL DEVELOPING	POLYFILL CONSIDERING
SHADOW DOM	STABLE	STABLE	STABLE	POLYFILL DEVELOPING	POLYFILL CONSIDERING
IMPORTS	STABLE	STABLE	POLYFILL ON HOLD	POLYFILL ON HOLD	POLYFILL CONSIDERING

Support de navigateurs natifs en desktop et en mobile : Safari (et oui!)
Des scripts polyfills performant sont disponible pour rendre les autres principaux navigateurs pleinement compatibles (IE 11)
Firefox en est très proche, activable via un flag dev et prévu par défaut dans la version de production 60/61
et le mauvais élève reste IE

EXEMPLE DE CUSTOM ELEMENT

```
<my-component size="large" theme="light"/></my-component>
```

EXEMPLE DE CUSTOM ELEMENT

```
class MyComponent extends HTMLElement {
        createdCallback() {
                // Standard DOM/fetch/etc. code...
        }
        attachedCallback() {}
        detachedCallback() {}
        attributeChangedCallback() {}
}

document.registerElement('my-component', MyComponent);
```

POUVONS-NOUS RENDRE PLUS FACILE LA CONSTRUCTION D'ÉLÉMENTS PERSONNALISÉS ?

- Souhait de continuer à utiliser des fonctionnalités que seul les frameworks proposent
- Désir de gérer facilement des bundles de composants
- Volonté d'utiliser un language typé comme TypeScript

- Souhait de continuer à utiliser des fonctionnalités que seul les frameworks proposent
- Désir de gérer facilement des bundles de composants par thématique
- Beaucoup d'équipes ont choisi TypeScript pour unifier et typer leur développement Javascript à travers leurs équipes

STENCIL

UN SIMPLE COMPILATEUR POUR CRÉER DES WEB COMPONENTS RAPIDE ET RÉACTIF

STENCILJS.COM

STENCIL ? KÉSAKO

- Un **compilateur** qui génère des Custom Elements, une partie de la spécification des composants Web
- **Ce n'est pas un framework** : la sortie est 100% conforme aux normes Custom Elements
- Ajoute de **puissantes fonctionnalités** venant du monde des frameworks aux Web Components
- Créé et utilisé par l'équipe derrière Ionic Framework. **Ionic 4+** et ionicons sont construits dessus !

Un compilateur qui génère des éléments personnalisés, partie intégrante de la spécification des Web Components. Ajoute de puissantes fonctionnalités venant du monde des frameworks aux Web Components

POURQUOI STENCIL ?

- **Performance** : les frameworks traditionnels s'avèrent trop lourds pour désservir une expérience mobile exigeante via des applications Web progressives (PWA)
- **Stabilité** : désir d'utiliser les standards Web et éviter les réécritures constantes lors d'un changement de framework

POURQUOI STENCIL ?

- **Interopérabilité** : capacité à créer des composants qui fonctionnent à l'identique à travers tous les principaux frameworks
- **Familiarité** : fonctionnalités prisées par les frameworks JS mais dans un package plus léger et conforme aux normes

Le compilateur stencil n'est pas là pour remplacer l'utilisation de frameworks même si il capable de gérer un site ou une application professionnelle elle-même.

EXEMPLE D'UN COMPOSANT

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

@Component({
  tag: 'my-name',
  styleUrl: 'my-name.scss'
})
export class MyName {
  @Prop() name: string;

  render() {
    return (
      <p>
        Hello, my name is {this.name}
      </p>
    );
  }
}
```

```
<my-name name="Ch'Big" /></my-name>
```

La nomenclature est importante avec un dash dans le nom du composant pour le dissocier des balises HTML5
Stencil utilise le meilleur d'angular (la déclaration d'une classe et ses décorateurs sous typescript) et de react (JSX et système de rendu performant)
On peut directement utiliser un fichier css pour remplcer sass ou n'importe quel autre moteur et tiré profit des CSS variables
Utilisation dans votre fichier HTML ou dans un autre composant JSX

LES COMPOSANTS COMPILÉS PAR STENCIL ONT

- **Virtual DOM** : mises à jour DOM rapides sans les pièges communs de performance DOM
- **Lazy Loading** : par défaut, les composants sont chargés de manière asynchrone et peuvent être associés à d'autres composants
- **Réactivité** : mises à jour efficaces basées sur les changements de propriété et d'état

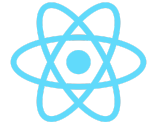
- **Virtual DOM** : fast DOM updates without common DOM performance pitfalls
- **Lazy Loading** (chargement paresseux) : By default components load asyncronously and can be bundled with related components
- **Réactivité** : Efficient updates based on property and state changes

LES COMPOSANTS COMPILÉS PAR STENCIL ONT

- **Rendu haute performance** : système de rendu asynchrone similaire à React Fiber
- **JSX** : système de markup populaire et familier lancé par React
- **Server Side Rendering** : hydratez les composants précompilés sur le serveur sans l'utilisation d'un navigateur sans tête (headless browser)

- **High-performance Rendering** : async rendering system, similar to React Fiber
- **JSX** : Popular and familiar markup system pioneered by React
- **Server Side Rendering** : Hydrate pre-compiled components on the server without a headless browser : « navigateur sans tête », navigateur web sans interface graphique

LES FONCTIONNALITÉS DES FRAMEWORKS INCLUS


JSX / Virtual DOM	×	✓	✓
Async Rendering	×	✓	✓
TypeScript	✓	×	✓
Decorators	✓	×	✓
Server side rendering	✓	✓	✓

STENCIL API - CYCLE DE VIE

- **componentWillLoad()** : le composant est sur le point de charger et il n'a pas encore été rendu
- **componentDidLoad()** : le composant a été chargé et a déjà été rendu
- **componentWillUpdate()** : le composant est sur le point d'être mis à jour et rendu.
- **componentDidUpdate()** : le composant vient d'effectuer son rendu
- **componentDidUnload()** : le composant a déchargé et l'élément sera détruit

componentWillLoad : le composant est sur le point de charger et il n'a pas encore été rendu.
C'est le meilleur endroit pour faire des mises à jour de données avant le premier rendu.
componentWillLoad ne sera appelé qu'une seule fois.

componentDidLoad : le composant a été chargé et a déjà été rendu.
La mise à jour des données dans cette méthode provoquera le rendu du composant.
componentDidLoad ne sera appelé qu'une seule fois.

componentWillUpdate : le composant est sur le point d'être mis à jour et rendu.
componentDidUpdate : Le composant vient de se re-rendre.
Appelé plusieurs fois tout au long de la vie du composant quand il se met à jour.
componentWillUpdate n'est pas appelé lors du premier rendu.

STENCIL API - DÉCORATEURS

- **@Component()** : définir le nom du tag et la feuille de style associée (Sass ou plain CSS)

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

@Component({
  tag: 'todo-list',
  styleUrl: 'todo-list.scss',
  host: {
    theme: 'todo',
    role: 'list'
  }
})
export class TodoList {
  ...
}
```

Chaque composant Stencil doit être décoré avec un décorateur @Component() du paquet @stencil/core.
Dans le cas le plus simple, les développeurs doivent fournir un nom de balise HTML pour le composant. Souvent, un styleUrl est également utilisé, ou même styleUrls, où plusieurs feuilles de style différentes peuvent être fournies pour différents modes/thèmes d'application.

Le décorateur de composants a également une option hôte. Cela vous permet de définir des classes et des attributs CSS sur le composant que vous construisez.

STENCIL API - DÉCORATEURS

- **@Prop()** : créer une propriété sur le composant
- **@State()** : créer un état local et le surveiller pour mettre à jour le rendu si détection d'un changement

```
import { Prop, State } from '@stencil/core';
...
export class MyName {

@Prop({ mutable: true }) name: string = 'oùestcharly';
@State() completedTodos: Todo[];
...
```

Le décorateur @Prop() sont des attributs/propriétés exposés publiquement sur l'élément. Par défaut à chaque changement du décorateur, le composant est rendu de nouveau
Il est important de savoir que @Prop est par défaut immuable à l'intérieur de la logique du composant.
Une fois qu'une valeur est définie par un utilisateur, le composant ne peut pas la mettre à jour en interne.
Cependant, il est possible d'autoriser explicitement la mutation suivante, comme dans l'exemple suivant:

Le décorateur @State() peut être utilisé pour gérer les données internes d'un composant. Cela signifie qu'un utilisateur ne peut pas modifier ces données de l'extérieur du composant, mais le composant peut le modifier comme il le souhaite.
Toute modification apportée à une propriété @State () entraîne l'appel de la fonction de rendu des composants.

STENCIL API - DÉCORATEURS

- **@Method()** : exposer une méthode à l'api public

```
import { Method } from '@stencil/core';

export class TodoList {

  @Method()
  showPrompt() {
    // show a prompt
  }
}
```

On peut appeler cette méthode comme cela :

```
const todoListElt = document.querySelector('todo-list')
todoListElt.showPrompt();
```

Le décorateur @Method() est utilisé pour exposer des méthodes sur l'API publique. Les fonctions décorées avec le décorateur @Method() peuvent être appelées directement depuis l'élément.

STENCIL API - DÉCORATEURS

- **@Watch()** : lorsqu'un utilisateur met à jour une propriété, ce décorateur lancera la méthode à laquelle elle est attachée et transmettra la nouvelle et l'ancienne valeur

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

export class LoadingIndicator {
  @Prop() activated: boolean;

  @Watch('activated')
  watchHandler(newValue: boolean, oldValue: boolean) {
    console.log('New value is: ', newValue);
  }
}
```

Pour faire la validation d'une propriété, vous pouvez utiliser le décorateur @Watch()

STENCIL API - DÉCORATEURS

- **@Element()** : récupèrer l'élément DOM pour ce composant

```
import { Element } from '@stencil/core';

export class TodoList {

  @Element() todoListEl: HTMLElement;

  addClass(){
    this.todoListEl.classList.add('active');
  }
}
```

Le décorateur @Element() est comment avoir accès à l'élément host dans l'instance de classe. Cela renvoie une instance d'un HTMLElement, donc les méthodes / événements DOM standards peuvent être utilisés ici.

STENCIL API - EVENEMENTS

- @Event() : déclencher des événements sur un composant

```
import { Event, EventEmitter } from '@stencil/core';

...
export class TodoList {

  @Event() todoCompleted: EventEmitter;

  todoCompletedHandler(todo: Todo) {
    this.todoCompleted.emit(todo);
  }
}
```

Les composants peuvent émettre des données et des événements à l'aide du décorateur Event Emitter.

Le code ci-dessus enverra un événement DOM personnalisé appelé todoCompleted.

STENCIL API - EVENEMENTS

- @Listen() : écouter les événements envoyés par ses enfants

```
import { Listen } from '@stencil/core';

...
export class TodoApp {

  @Listen('todoCompleted')
  todoCompletedHandler(event: CustomEvent) {
    console.log('Received todoCompleted event: ', event.det
  }
}
```

Le décorateur @Listen() permet de gérer les événements envoyés depuis @Events.

Dans cet exemple, on suppose qu'un composant enfant, TodoList, émet un événement todoCompleted à l'aide de EventEmitter.

STENCIL VS X

- **Angular/React/Vue/...** : Stencil construit des composants Web standard qui s'exécutent nativement dans le navigateur.
- **Polymer** : Stencil fonctionne à la compilation plutôt qu'à l'exécution. Pragmatique sur JSX, Virtual DOM et d'autres fonctionnalités des frameworks.
- **Vanilla Web Components** : Stencil fournit des fonctionnalités complexes venant des frameworks, comme si vous les aviez écrites vous-même.

Comment Stencil est différent de X ?
Stencil fonctionne principalement à la compilation plutôt qu'à l'exécution. Produit des composants Web "vanilla". Pragmatique sur JSX, Virtual DOM et d'autres fonctionnalités des frameworks

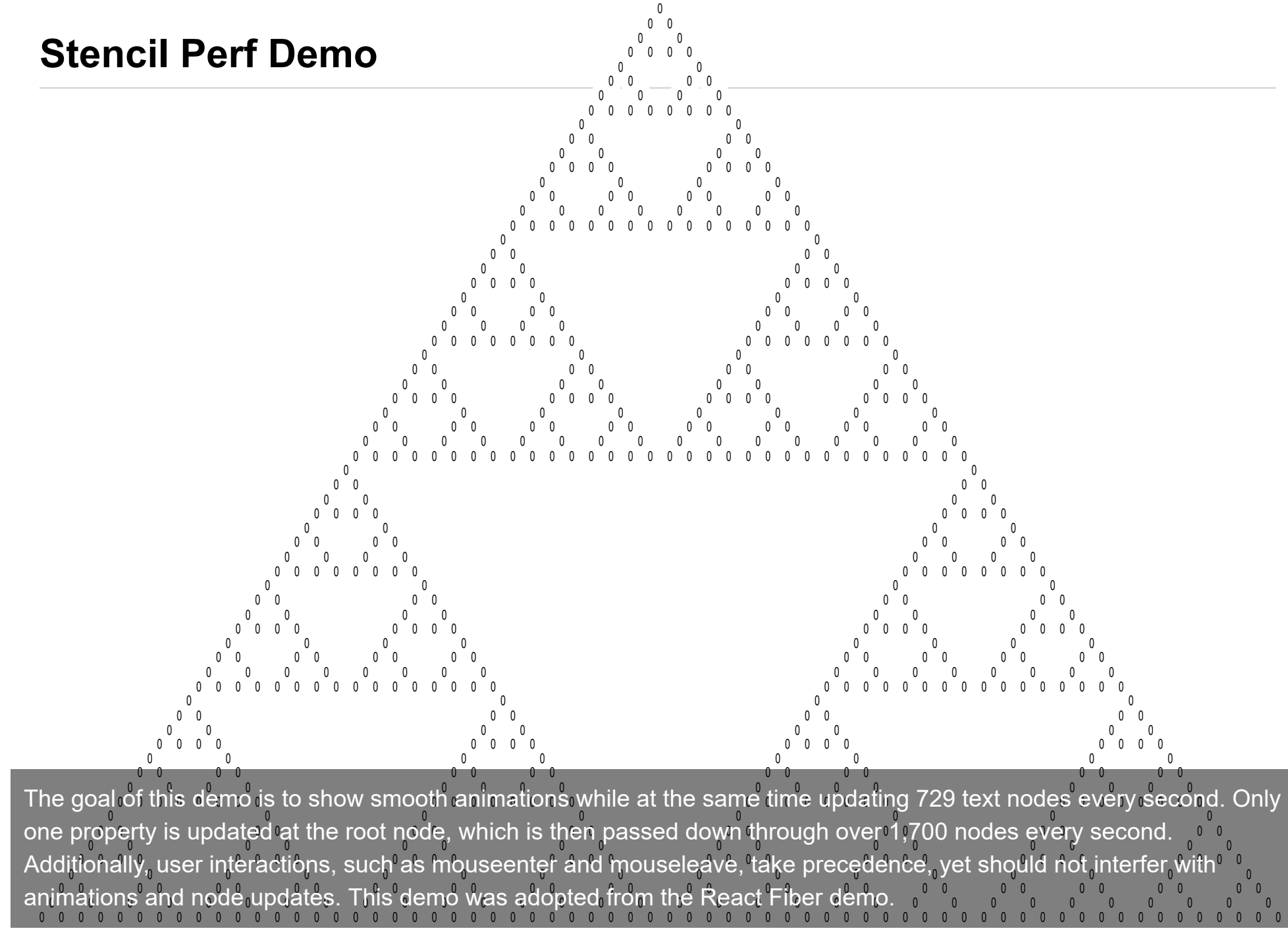
Stencil fournit des fonctionnalités complexes venant des frameworks, comme si vous les aviez écrites vous-même dans des composants Web "vanilla".

DEMO

C'est parti !

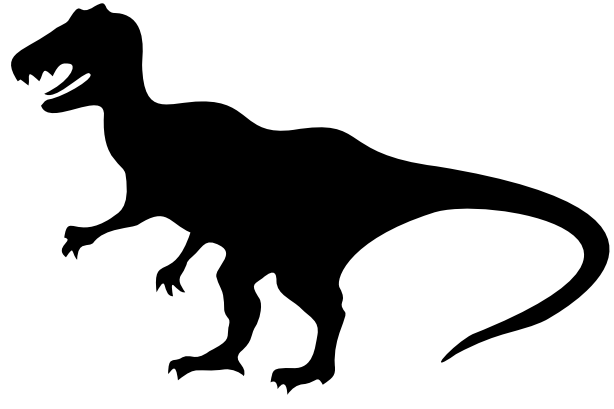
Un composant n'est pas forcément lié à une interaction graphique, il peut faire un fetch d'une requete ou appeler une autre api (web share, web payment)
Il est existe aussi un bundle pour construire une appli PWA avec ionic & stencil avec un score de 100% sur lighthouse

Stencil Perf Demo



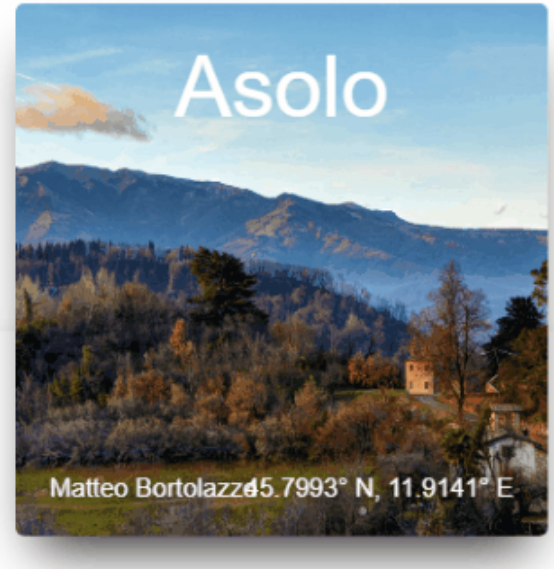
The goal of this demo is to show smooth animations while at the same time updating 729 text nodes every second. Only one property is updated at the root node, which is then passed down through over 1,700 nodes every second. Additionally, user interactions, such as mouseenter and mouseleave, take precedence, yet should not interfer with animations and node updates. This demo was adopted from the React Fiber demo.

SVGPATHS MORPHING



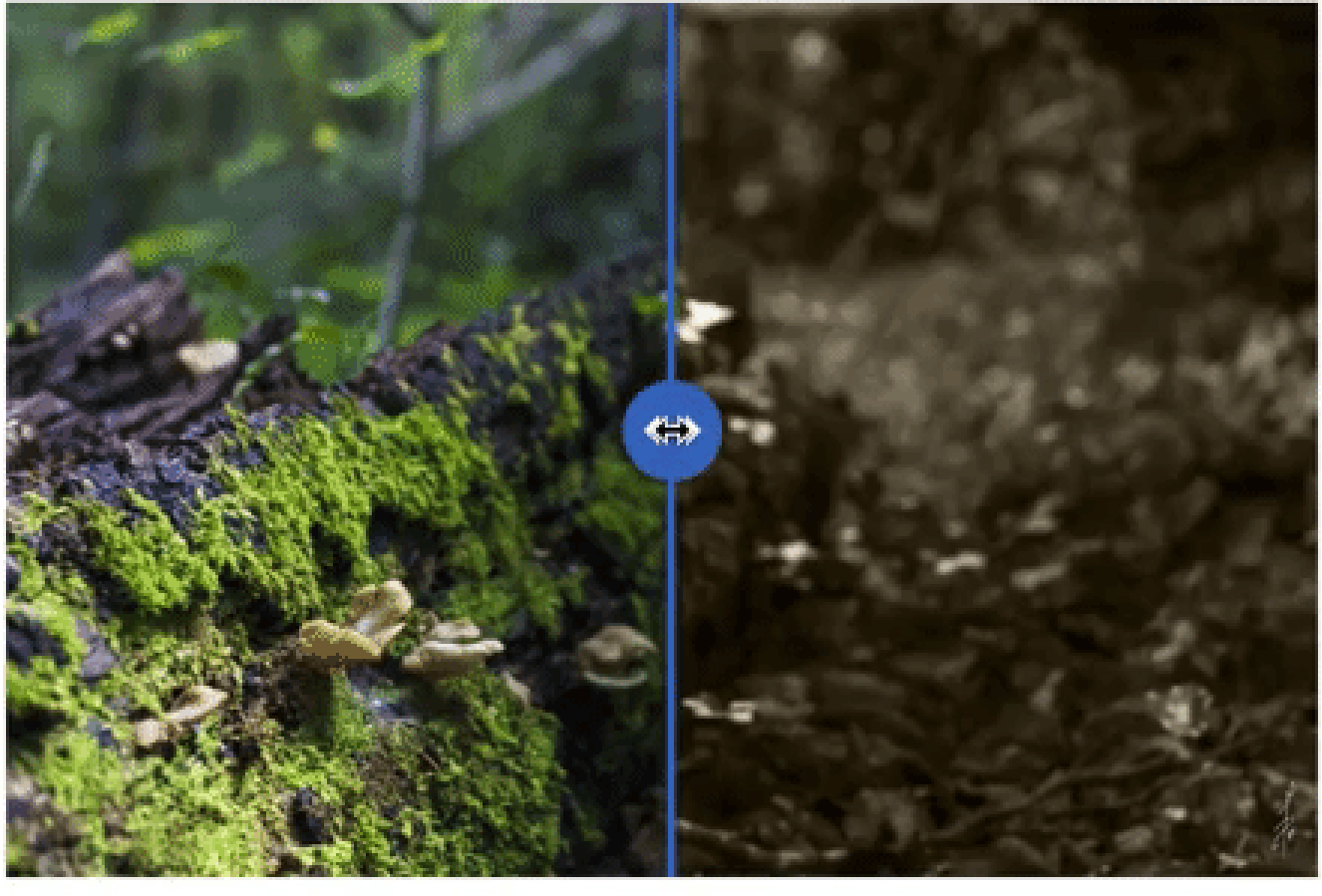
MORPHING UI

Sign In



Asolo is a town and comune in the Veneto Region of Northern Italy.

IMAGE COMPARISON SLIDER



TWITTER COMPONENT

Examples of using this web component.

repo for this component. st-twitter

npm i st-twitter

Follow Button

<st-twitter type="follow" user="stenciljs"></st-twitter>

Follow @stenciljs

Hashtag Tweet

<st-twitter type="hashtag" hashtag="stenciljs" text="this is the text of the tweet">
</st-twitter>

RESSOURCES

- StencilJS.com
- github.com/ionic-team/stencil
- github.com/ionic-team/ionic-pwa-toolkit
- Stencilcomponents.com
- wc-todo.firebaseapp.com

RESSOURCES (DEMO)

- stencil-fiber-demo.firebaseapp.com
- stencilcomponents.com/component/svg-paths-morphing
- github.com/matteobortolazzo/fast-morph
- stencilcomponents.com/component/image-comparison-slider
- stencilcomponents.com/component/twitter