# stencil

The magical, reusable web component compiler

## Web Component ?

### Plaît-il ?

## Web Component

```
A set of web platform APIs that allow you to create new custom, reusable,
encapsulated HTML tags to use in web pages and web apps...

...will work across modern browsers, and can be used with
any JavaScript library or framework that works with HTML.
```

```
<whoop-whoop></whoop-whoop>
```

## Concepts

- Custom Element API

## Concepts

- Custom Element API
- Shadow DOM

## Concepts

- Custom Element API
- Shadow DOM
- HTML imports

## Concepts

- Custom Element API
- Shadow DOM
- HTML imports
- HTML Template

```
<template>
  <p>Hello <strong></strong></p>
</template>

<script>
(function(window, document, undefined) {

    var thatDoc = document;
    var thisDoc =  (thatDoc._currentScript || thatDoc.currentScript).ownerDocument;

    var template = thisDoc.querySelector('template').content;

    var MyElementProto = Object.create(HTMLElement.prototype);
    MyElementProto.who = 'World';

    MyElementProto.createdCallback = function() {
        var shadowRoot = this.createShadowRoot();
        var clone = thatDoc.importNode(template, true);
        shadowRoot.appendChild(clone);
        this.strong = shadowRoot.querySelector('strong');
        if (this.hasAttribute('who')) {
            var who = this.getAttribute('who');
            this.setWho(who);
        }
        else {
            this.setWho(this.who);
        }
    };
    MyElementProto.attributeChangedCallback = function(attr, oldVal, newVal) {
        if (attr === 'who') {
            this.setWho(newVal);
        }
    };
    MyElementProto.setWho = function(val) {
        this.who = val;
        this.strong.textContent = this.who;
    };

    window.MyElement = thatDoc.registerElement('hello-world', {
        prototype: MyElementProto
    });
})(window, document);
</script>
```

```
<template>
  <p>Hello <strong></strong></p>
</template>

<script>
(function(window, document, undefined) {

    var thatDoc = document;
    var thisDoc =  (thatDoc._currentScript || thatDoc.currentScript).ownerDocument;

    var template = thisDoc.querySelector('template').content;

    var MyElementProto = Object.create(HTMLElement.prototype); // Custom Element API
    MyElementProto.who = 'World';

    MyElementProto.createdCallback = function() { // Custom Element API
        var shadowRoot = this.createShadowRoot();
        var clone = thatDoc.importNode(template, true);
        shadowRoot.appendChild(clone);
        this.strong = shadowRoot.querySelector('strong');
        if (this.hasAttribute('who')) {
            var who = this.getAttribute('who');
            this.setWho(who);
        }
        else {
            this.setWho(this.who);
        }
    };
    MyElementProto.attributeChangedCallback = function(attr, oldVal, newVal) { // Custom Element API
        if (attr === 'who') {
            this.setWho(newVal);
        }
    };
    MyElementProto.setWho = function(val) {
        this.who = val;
        this.strong.textContent = this.who;
    };

    window.MyElement = thatDoc.registerElement('hello-world', { // Custom Element API
        prototype: MyElementProto
    });
})(window, document);
</script>
```

```
<template>
  <p>Hello <strong></strong></p>
</template>

<script>
(function(window, document, undefined) {

    var thatDoc = document;
    var thisDoc =  (thatDoc._currentScript || thatDoc.currentScript).ownerDocument;

    var template = thisDoc.querySelector('template').content;

    var MyElementProto = Object.create(HTMLElement.prototype);
    MyElementProto.who = 'World';

    MyElementProto.createdCallback = function() {
        var shadowRoot = this.createShadowRoot(); // Shadow DOM
        var clone = thatDoc.importNode(template, true);
        shadowRoot.appendChild(clone);
        this.strong = shadowRoot.querySelector('strong');
        if (this.hasAttribute('who')) {
            var who = this.getAttribute('who');
            this.setWho(who);
        }
        else {
            this.setWho(this.who);
        }
    };
    MyElementProto.attributeChangedCallback = function(attr, oldVal, newVal) {
        if (attr === 'who') {
            this.setWho(newVal);
        }
    };
    MyElementProto.setWho = function(val) {
        this.who = val;
        this.strong.textContent = this.who;
    };

    window.MyElement = thatDoc.registerElement('hello-world', {
        prototype: MyElementProto
    });
})(window, document);
</script>
```

```
<template> <!-- HTML Template -->
  <p>Hello <strong></strong></p>
</template>

<script>
(function(window, document, undefined) {

    var thatDoc = document;
    var thisDoc =  (thatDoc._currentScript || thatDoc.currentScript).ownerDocument;

    var template = thisDoc.querySelector('template').content; // HTML Template

    var MyElementProto = Object.create(HTMLElement.prototype);
    MyElementProto.who = 'World';

    MyElementProto.createdCallback = function() {
        var shadowRoot = this.createShadowRoot();
        var clone = thatDoc.importNode(template, true); // HTML Template
        shadowRoot.appendChild(clone); // HTML Template
        this.strong = shadowRoot.querySelector('strong');
        if (this.hasAttribute('who')) {
            var who = this.getAttribute('who');
            this.setWho(who);
        }
        else {
            this.setWho(this.who);
        }
    };
    MyElementProto.attributeChangedCallback = function(attr, oldVal, newVal) {
        if (attr === 'who') {
            this.setWho(newVal);
        }
    };
    MyElementProto.setWho = function(val) {
        this.who = val;
        this.strong.textContent = this.who;
    };

    window.MyElement = thatDoc.registerElement('hello-world', {
        prototype: MyElementProto
    });
})(window, document);
</script>
```

## et stencil ?

## The magical, reusable web component *compiler*

https://stenciljs.com

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

@Component({
  tag: 'hello-world',
  styleUrl: 'hello-world.scss'
})
export class HelloWorldComponent {

  @Prop() who: string;

  render() {
    return (
      <p>
        Hello {this.who}
      </p>
    );
  }
}
```

## Quels intêrets ?

## Quels intêrets ?

- Syntaxe légère

## Quels intêrets ?

- Syntaxe légère
- Polyfills chargés dynamiquement

## Quels intêrets ?

- Syntaxe légère
- Polyfills chargés dynamiquement
- Prerendering

## Quels intêrets ?

- Syntaxe légère
- Polyfills chargés dynamiquement
- Prerendering
- Server Side Rendering

## Quels intêrets ?

- Syntaxe légère
- Polyfills chargés dynamiquement
- Prerendering
- Server Side Rendering
- Service Workers (Progressive Web App)

## Quels intêrets ?

- Syntaxe légère
- Polyfills chargés dynamiquement
- Prerendering
- Server Side Rendering
- Service Workers (Progressive Web App)
- Facilités de dévelopement (livereload, test unitaire, distribution ...)

## Getting started

Composant

```
git clone https://github.com/ionic-team/stencil-component-starter my-co
```

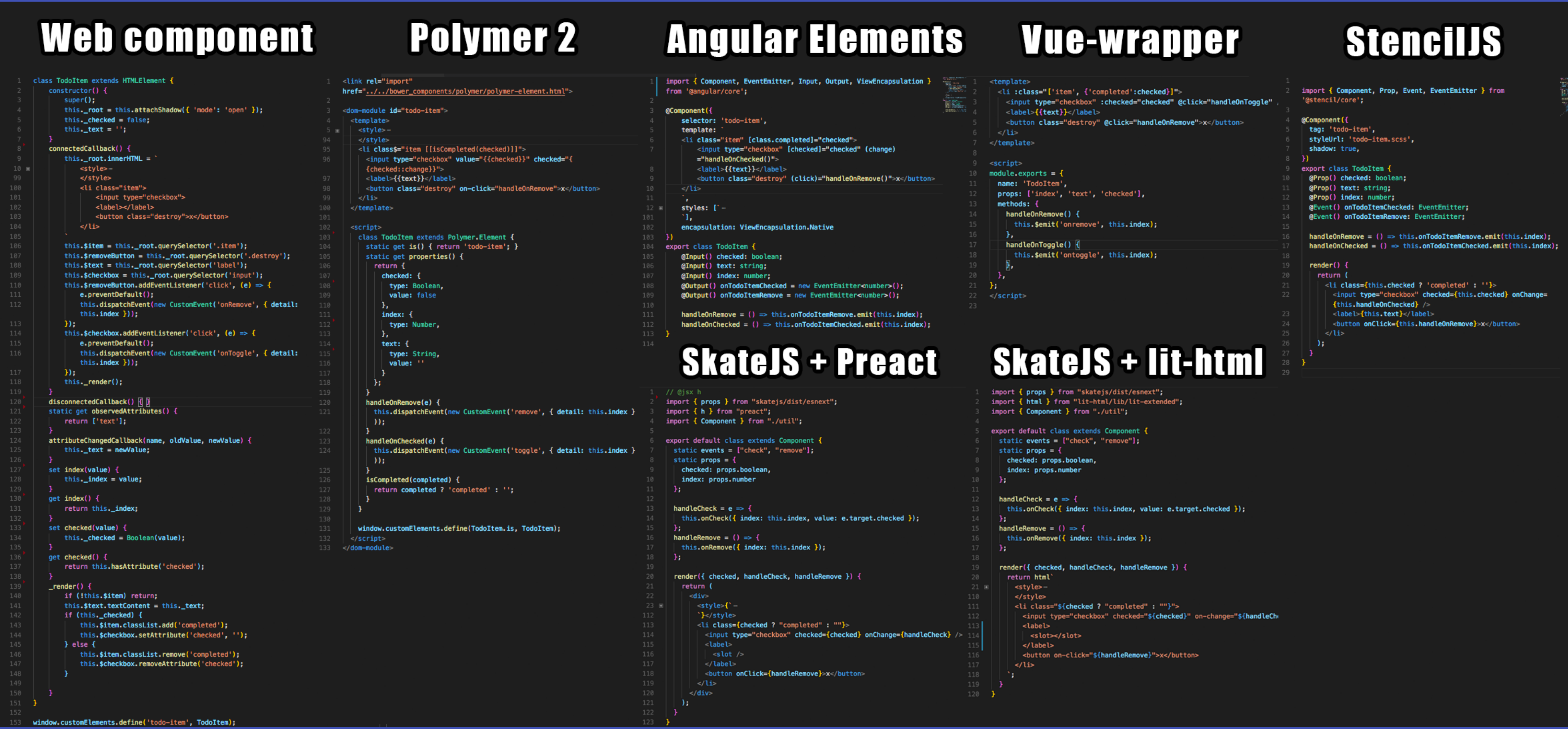
Application

```
git clone https://github.com/ionic-team/stencil-app-starter my-app
```

***Grrr***

***Waouf***

# Questions ?



## Web component

```
class TodoItem extends HTMLElement {
    constructor() {
        super();
        this._root = this.attachShadow({ 'mode': 'open' });
        this._checked = false;
        this._text = '';
    }
    connectedCallback() {
        this._root.innerHTML = `
            <style>…
            </style>
            <li class="item">
                <input type="checkbox">
                <label></label>
                <button class="destroy">x</button>
            </li>
        `
        this.$item = this._root.querySelector('.item');
        this.$removeButton = this._root.querySelector('.destroy');
        this.$text = this._root.querySelector('label');
        this.$checkbox = this._root.querySelector('input');
        this.$removeButton.addEventListener('click', (e) => {
            e.preventDefault();
            this.dispatchEvent(new CustomEvent('onRemove', { detail: this.index }));
        });
        this.$checkbox.addEventListener('click', (e) => {
            e.preventDefault();
            this.dispatchEvent(new CustomEvent('onToggle', { detail: this.index }));
        });
        this._render();
    }
    disconnectedCallback() { }
    static get observedAttributes() {
        return ['text'];
    }
    attributeChangedCallback(name, oldValue, newValue) {
        this._text = newValue;
    }
    set index(value) {
        this._index = value;
    }
    get index() {
        return this._index;
    }
    set checked(value) {
        this._checked = Boolean(value);
    }
    get checked() {
        return this.hasAttribute('checked');
    }
    _render() {
        if (!this.$item) return;
        this.$text.textContent = this._text;
        if (this._checked) {
            this.$item.classList.add('completed');
            this.$checkbox.setAttribute('checked', '');
        } else {
            this.$item.classList.remove('completed');
            this.$checkbox.removeAttribute('checked');
        }

    }
}

window.customElements.define('todo-item', TodoItem);
```

## Polymer 2

```
<link rel="import" href="../../bower_components/polymer/polymer-element.html">

<dom-module id="todo-item">
  <template>
    <style>…
    </style>
    <li class$="item [[isCompleted(checked)]]">
      <input type="checkbox" value="{{checked}}" checked="{{checked::change}}">
      <label>{{text}}</label>
      <button class="destroy" on-click="handleOnRemove">x</button>
    </li>
  </template>

  <script>
    class TodoItem extends Polymer.Element {
      static get is() { return 'todo-item'; }
      static get properties() {
        return {
          checked: {
            type: Boolean,
            value: false
          },
          index: {
            type: Number,
          },
          text: {
            type: String,
            value: ''
          }
        };
      }
      handleOnRemove(e) {
        this.dispatchEvent(new CustomEvent('remove', { detail: this.index }));
      }
      handleOnChecked(e) {
        this.dispatchEvent(new CustomEvent('toggle', { detail: this.index }));
      }
      isCompleted(completed) {
        return completed ? 'completed' : '';
      }
    }

    window.customElements.define(TodoItem.is, TodoItem);
  </script>
</dom-module>
```

## Angular Elements

```
import { Component, EventEmitter, Input, Output, ViewEncapsulation } from '@angular/core';

@Component({
    selector: 'todo-item',
    template: `
    <li class="item" [class.completed]="checked">
        <input type="checkbox" [checked]="checked" (change)="handleOnChecked()">
        <label>{{text}}</label>
        <button class="destroy" (click)="handleOnRemove()">x</button>
    </li>
    `,
    styles: [`…
    `],
    encapsulation: ViewEncapsulation.Native
})
export class TodoItem {
    @Input() checked: boolean;
    @Input() text: string;
    @Input() index: number;
    @Output() onTodoItemChecked = new EventEmitter<number>();
    @Output() onTodoItemRemove = new EventEmitter<number>();

    handleOnRemove = () => this.onTodoItemRemove.emit(this.index);
    handleOnChecked = () => this.onTodoItemChecked.emit(this.index);
}
```

## Vue-wrapper

```
<template>
  <li :class="['item', {'completed':checked}]">
    <input type="checkbox" :checked="checked" @click="handleOnToggle" ,
    <label>{{text}}</label>
    <button class="destroy" @click="handleOnRemove">x</button>
  </li>
</template>

<script>
module.exports = {
  name: 'TodoItem',
  props: ['index', 'text', 'checked'],
  methods: {
    handleOnRemove() {
      this.$emit('onremove', this.index);
    },
    handleOnToggle() {
      this.$emit('ontoggle', this.index);
    },
  },
};
</script>
```

## StencilJS

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

@Component({
  tag: 'todo-item',
  styleUrl: 'todo-item.scss',
  shadow: true,
})
export class TodoItem {
  @Prop() checked: boolean;
  @Prop() text: string;
  @Prop() index: number;
  @Event() onTodoItemChecked: EventEmitter;
  @Event() onTodoItemRemove: EventEmitter;

  handleOnRemove = () => this.onTodoItemRemove.emit(this.index);
  handleOnChecked = () => this.onTodoItemChecked.emit(this.index);

  render() {
    return (
      <li class={this.checked ? 'completed' : ''}>
        <input type="checkbox" checked={this.checked} onChange={this.handleOnChecked} />
        <label>{this.text}</label>
        <button onClick={this.handleOnRemove}>x</button>
      </li>
    );
  }
}
```

## SkateJS + Preact

```
// @jsx h
import { props } from "skatejs/dist/esnext";
import { h } from "preact";
import { Component } from "./util";

export default class extends Component {
  static events = ["check", "remove"];
  static props = {
    checked: props.boolean,
    index: props.number
  };

  handleCheck = e => {
    this.onCheck({ index: this.index, value: e.target.checked });
  };
  handleRemove = () => {
    this.onRemove({ index: this.index });
  };

  render({ checked, handleCheck, handleRemove }) {
    return (
      <div>
        <style>{`…
        `}</style>
        <li class={checked ? "completed" : ""}>
          <input type="checkbox" checked={checked} onChange={handleCheck} />
          <label>
            <slot />
          </label>
          <button onClick={handleRemove}>x</button>
        </li>
      </div>
    );
  }
}
```

## SkateJS + lit-html

```
import { props } from "skatejs/dist/esnext";
import { html } from "lit-html/lib/lit-extended";
import { Component } from "./util";

export default class extends Component {
  static events = ["check", "remove"];
  static props = {
    checked: props.boolean,
    index: props.number
  };

  handleCheck = e => {
    this.onCheck({ index: this.index, value: e.target.checked });
  };
  handleRemove = () => {
    this.onRemove({ index: this.index });
  };

  render({ checked, handleCheck, handleRemove }) {
    return html`
      <style>…
      </style>
      <li class="${checked ? "completed" : ""}">
        <input type="checkbox" checked="${checked}" on-change="${handleCh
        <label>
          <slot></slot>
        </label>
        <button on-click="${handleRemove}">x</button>
      </li>
    `;
  }
}
```