

# FRAMEWORK ANGULAR

---

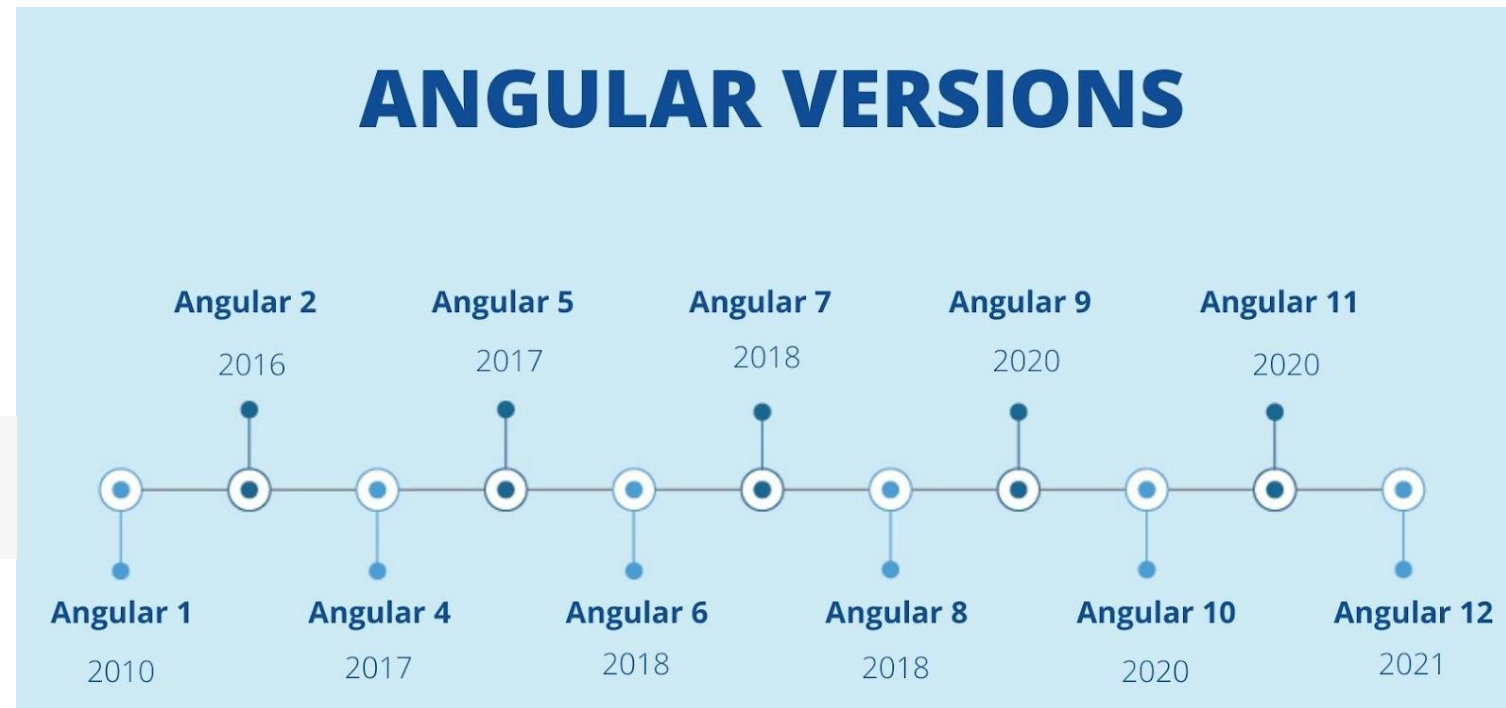
Utilisation de Framework

Notion de composant web

Concept d'Angular

# FRAMEWORK ANGULAR

---



# FRAMEWORK ANGULAR

---

## Impérative vs déclarative

JQuery

```
<input type="text" id="yourName">
<h1 id="helloName"></h1>
<script type="text/javascript">
  $(function() {
    $("#yourName").keyup(function () {
      $("#helloName").text("Hello " + this.value + "!");
    });
  });
</script>
```

Angular

```
<input type="text" [(ngModel)]="yourName">
<h1>Hello {{yourName}}!</h1>
```

# FRAMEWORK ANGULAR

---

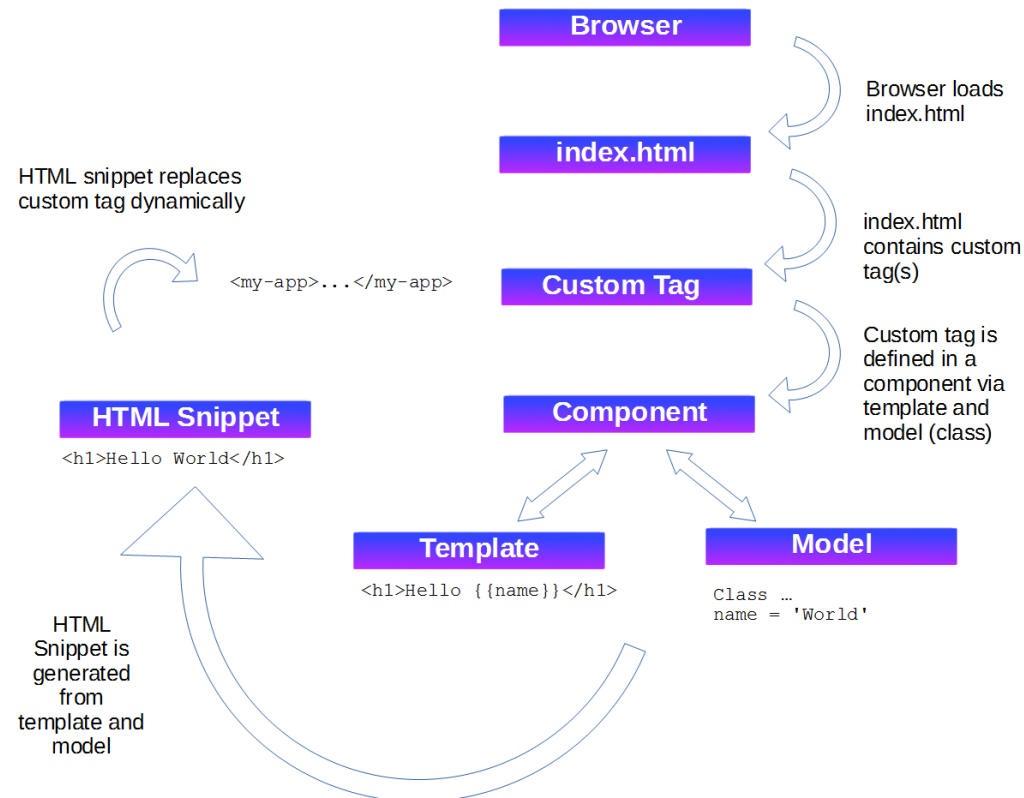
## **Abstraction :**

- jQuery a déjà une abstraction par rapport au fonctionnement du navigateur (traversé du DOM, event binding, etc.)
- Angular rend les relations abstraites (entre le modèle et la vue, différents éléments de la vue, etc.)

# FRAMEWORK ANGULAR

---

→ Angular '*compile*' HTML : utilisation de custom tag



# FRAMEWORK : COMPOSANT WEB

---

- Faciliter la réutilisation d'éléments dans la monde du web (création d'élément réutilisables, encapsulés et versatiles sans risquer une collision avec d'autres morceaux de code).

## Standard du web :

- **Custom Elements** (permet de créer et enregistrer des nouveaux éléments HTML)
- **HTML Templates** (squelette pour créer des éléments HTML instanciables)
- **Shadow DOM** (permet d'encapsuler le JavaScript et le CSS des éléments)
- **HTML Imports** (abandonnées au profit des imports JavaScript)

# FRAMEWORK : COMPOSANT WEB

---

→ Enrichi le web avec des nouveaux tag.

## **Principe :**

Créer des nouveaux tags










Encapsuler le code pour masquer et isoler sa complexité

Etre capable d'importer et de déclarer les tags dans d'autres pages ou projets.

# FRAMEWORK : COMPOSANT WEB

---

- Standardisation des composants web en 2012 par W3C.
- Google via son projet ***Polymer*** à aider au développement des web composants.

Browser support	 CHROME	 OPERA	 SAFARI	 FIREFOX	 EDGE
 HTML TEMPLATES	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
 CUSTOM ELEMENTS	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
 SHADOW DOM	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
 ES MODULES	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE



# COMPOSANT WEB : CUSTOM ELEMENT

---

→ Extension de l'API JavaScript *CustomElements* pour créer des nouveau éléments.

**Etape 1 :** Créer une classe (syntaxe de classe ES6) dans laquelle est spécifié la fonctionnalité du composant web.

→ Hérite de *HTMLElement*

→ Possibilité de définir des rappels se déclenchant à différents points du cycle de vie de l'élément.

```
class HelloWorld extends HTMLElement {
  constructor(){
    super();
    console.log("constructor");
    //Fonctionnalité de l'élément
  }
  //Rappels du cycle de vie
  connectedCallback(){
    this.innerHTML = '<p>Composant connecté pour la 1ère fois au DOM : '+
      'Ajout du contenu initial / fetch data</p>'
  }
  disconnectedCallback(){
    //l'élémentt personnalisé est déconnecté du DOM du document
  }
  adoptedCallback(){
    //l'élément personnalité est déplacé vers un nouveau document
  }
  attributeChangedCallback(){
    //un des attributs de l'élément personnalisé est ajouté, supprimé ou modifié
  }
}
```

# COMPOSANT WEB : CUSTOM ELEMENT

---

**Etape 2 :** Enregistrer le nouvel élément personnalisé avec la commande *define*.

→ Le nom de l'élément doit contenir un « - »

```
//customElements.define(name, constructor, options);  
customElements.define('hello-world', HelloWorld);
```

125

→ Possibilité d'ajouté des options

```
customElements.define('word-count', WordCount, { extends: 'p' });
```

```
class WordCount extends HTMLParagraphElement { /*...*/ }
```

# COMPOSANT WEB : CUSTOM ELEMENT

---

**Deux types d'élément customisé :**

→ Éléments customisés autonomes (ne dépend pas d'un autre élément HTML)

```
<hello-world></hello-world>
```

```
document.createElement('hello-world');
```

→ Éléments intégrés personnalisés (héritant d'un élément HTML de base)

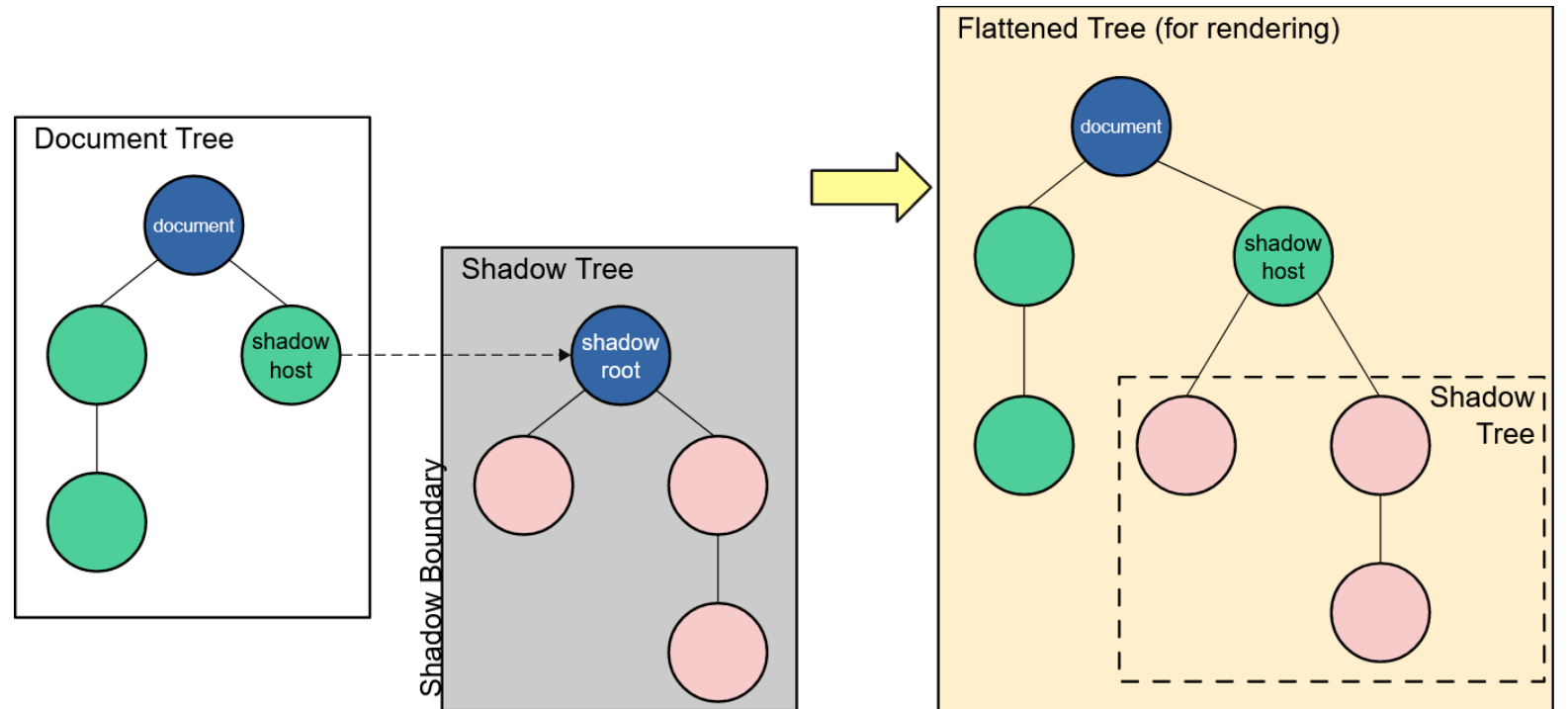
```
<p is="word-count"></p>
```

```
document.createElement("p", {is: "word-count"})
```

# COMPOSANT WEB : SHADOW DOM

- **Encapsulation** possible grâce au **Shadow DOM** : possible d'utiliser JavaScript et du CSS dans un élément customisé sans affecter les autres composants.

Lors de la création du shadow DOM, un sous arbre est rattaché à un élément du DOM.



# COMPOSANT WEB : SHADOW DOM

---

**Etape 3 :** Connecter un shadow DOM à l'élément personnalisé, puis ajouter des éléments fils, des écouteurs d'évènements, etc.

→ Associer une racine fantôme à un élément :

```
let fantome = element.attachShadow({mode: 'open'});  
let fantome = element.attachShadow({mode: 'closed'});
```

→ Ajout d'élément au Shadow DOM :

```
let paragraphe = document.createElement('p');  
fantome.appendChild(paragraphe);
```

# COMPOSANT WEB : EXEMPLE 1

→ Création d'un Shadow DOM

```
class HelloWorld extends HTMLElement {
  constructor(){
    super();
    console.log("constructor");
    //Fonctionnalité de l'élément
    this.nom = "Hello !"
    //Shadow DOM
    let shadowRoot = this.attachShadow({mode: 'open'});
    let newParagraph = document.createElement('p');
    newParagraph.setAttribute('class', 'paragraphe');
    newParagraph.innerHTML = 'Hello word du shadow DOM !';
    let otherParagraph = document.createElement('p');
    otherParagraph.innerHTML = this.afficherNom();
  }
}
```

Web Component

Nom de la balise : Hello !

Hello word du shadow DOM !

```
let newStyle = document.createElement('style');
newStyle.textContent = `
  .paragraphe {
    border: 1px solid black;
    border-radius: 5px;
    background-color: yellow;
  }`;
shadowRoot.appendChild(otherParagraph);
shadowRoot.appendChild(newStyle);
shadowRoot.appendChild(newParagraph);
}
afficherNom(){
  return 'Nom de la balise : ' + this.nom;
}
}
```

# COMPOSANT WEB : SHADOW DOM

---

→ Possibilité d'ajouter une feuille de style externe au lieu de la balise style :

```
const linkElem = document.createElement('link');  
linkElem.setAttribute('rel', 'stylesheet');  
linkElem.setAttribute('href', 'style.css');  
  
shadow.appendChild(linkElem);
```

# COMPOSANT WEB : HTML TEMPLATE

---

- Portion de code réutilisable
- Le moteur ne vérifie que la validé du contenu, le contenu n'est pas affiché
- Les scripts et les images ne sont pas chargés et le contenu n'est pas attaché au DOM (*document.getElementById()* et *querySelector()* ne fonctionneront pas)/

131

## Rappel d'utilisation :

```
<template id="hello">  
  <p>Autre Hello World !</p>  
</template>
```

```
let template = document.getElementById('hello');  
let templateContent = template.content.cloneNode(true);  
document.body.appendChild(templateContent);
```

Code JavaScript pour ajouter le template au DOM et l'afficher



# COMPOSANT WEB : HTML TEMPLATE

---

**Etape 4** : Possibilité de définir un template HTML. Utilisation des méthodes DOM pour cloner le template et le connecter au shadow DOM.

→ Constructeur de la classe *HelloWorld* :

```
constructor() {  
  super();  
  let template = document.getElementById('hello');  
  let templateContent = template.content;  
  
  const shadowRoot = this.attachShadow({mode: 'open'});  
  shadowRoot.appendChild(templateContent.cloneNode(true));  
}
```

# COMPOSANT WEB : EXEMPLE

→ Ajout d'un template

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <p>Web Component</p>
    <hello-world></hello-world>
    <template id="hello">
      <p>Autre Hello World avec template !</p>
    </template>
    <script src='js/composantWeb.js'></script>
  </body>
</html>
```

```
//Shadow DOM
let shadowRoot = this.attachShadow({mode: 'open'});

let template = document.getElementById('hello');
let templateContent = template.content;

shadowRoot.appendChild(templateContent.cloneNode(true));
```

133

Web Component

Autre Hello World avec template !

Nom de la balise : Hello !

Hello word du shadow DOM !

# COMPOSANT WEB : EXEMPLE

→ Page HTML :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <template id="counter">
      <style>
        button {
          background-color: ■ red;
          color: ■ white;
          padding: 4px;
        }
      </style>
      <button>Click me</button>
      <span id="times">0</span>
    </template>
    <my-counter></my-counter>
    <button>Pas dans le même composant</button>
    <script src='js/composantWeb2.js'></script>
  </body>
</html>
```

# COMPOSANT WEB : EXEMPLE

---

→ Page JavaScript :

```
class MyCounter extends HTMLElement {
  times = 0;
  constructor() {
    super();
    const template = document.getElementById('counter');
    const shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.appendChild(template.content.cloneNode(true));
    this.onClick = this.onClick.bind(this);
    this.shadowRoot.querySelector('button')
      .addEventListener('click', this.onClick);
  }
  onClick() {
    this.times += 1;
    this.shadowRoot.querySelector('#times').textContent = this.times;
  }
}
customElements.define("my-counter", MyCounter);
```

# COMPOSANT WEB : EXEMPLE

---

→ Résultat :

