Cómo documentar tu desarrollo con web-components

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Versión

Normativa front guia implementacion documentación WC Vigente

Versión: 1.0.0

Estado: **ACTIVO**

Entrada en vigor desde: 1 jun 2023

Obligado cumplimiento desde: Sept-2023

Normativa front guia implementacion documentación WC PRE-RELEASE

Versión: 1.0.0

Estado: RELEASE

Entrada en vigor desde: 28 nov 2022

Obligado cumplimiento desde: -

Tabla de Contenido

Cumplimiento Normativo

Las normas expuestas son de obligado cumplimiento. La STIC podrá estudiar los casos excepcionales los cuales serán gestionados a través de los responsables del proyecto correspondiente y autorizados por el Área de Gobernanza de la STIC. Asimismo cualquier aspecto no recogido en estas normas deberá regirse en primera instancia por las guías técnicas correspondientes al esquema nacional de seguridad y esquema nacional de interoperabilidad según correspondencia y en su defecto a los marcos normativos y de desarrollo software establecidos por la Junta de Andalucía, debiendo ser puesto de manifiesto ante la STIC.

La STIC se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata según la planificación de cada proyecto.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y /o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STIC para proceder a su validación técnica.

Contacto Arquitectura: l-arquitectura.stic@juntadeandalucia.es

Histórico de cambios

Los cambios en la normativa vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución. Ordenándose de mas recientes a menos recientes, prestando especial cuidado a las cabeceras de la tablas dónde se indican las fechas de entrada en vigor y versión.

Versión	Pre-release	Adopción	Activa	Retiro	Alcance
v01r00	15 jun 2022				• Versión inicial del documento

1. Introducción

En el desarrollo de componentes web, así cómo para cualquier elemento software orientado a la reutilización, la correcta gestión de la documentación es fundamental, de cara a su reaprovechamiento y mantenimiento. Esta guia define los aspectos básicos a documentar en la implementación de soluciones de front-end, teniedo en mente las siguientes cuestiones:

- La documentación del desarrollo, compondrá y contendrá el contrato de articulación del componente, definiendose este como aquellas caracteristicas necesarias a definir para el correcto funcionamiento de la solucióin, así como el conjunto de eventos esperables, que harán posible la interacción entre los distintos componentes.
- El modelo de documentación será expuesto a terceros mediante:
    - Storybook, solución de mockup y exposición del sistema de diseño
    - JSDOC, solución documental básica
    - README.md
- Los mecanismos documentales abordados en esta guia, van orientados a reutilizar al máximo la documentación provista en el propio código fuente.

---

2. Descripción técnica de la solución

La solución se basa en la generación automática de una serie de descriptores, extrayéndose de los comentarios definidos en el código fuente. Los ficheros generados de forma automática son:

- custom-elements.json
- Conjunto de ficheros que conforman la JSDOC.

Como se ha dicho en el punto 1, se deben generar los siguientes ficheros en los que se visualice nuestra documentación:

- **README.md**: Fichero markdown con la informacion de la API del componente.
- **custom-element.json**: Fichero en el que se basa toda la información que aparecen en las PropsTables del addon DOCS de storybook.

La **herramienta base para crear ambos ficheros es WCA (web-component-analyzer)** , y se trata de un CLI que se integra en nuestro proyecto y con el que se analizan de manera sencilla los componentes para obtener la documentación en distintos formatos.

Una vez se generaron los ficheros con WCA, se observó que la herramienta no agrupaba la documentación de los distintos tags que podría tener en el componente, de modo que solo aparecia en el fichero README.md o en storybook el primero que analizase. Por ello, el departamento de arquitectura crea una herramienta llamada CustomElementMerger mediante la cual, con la salida del WCA, se obtiene:

- Unir todas las tags generadas en el fichero custom-element.json en una sola.
- En la pestaña DOCS del componente en storybook, añade en la descripción una serie de chips que nos llevaran a la documentación de las clases generada por JSDOC.

Por último se usará la herramienta JSDOC, mediante la cual se analizarán los comentarios sobre el código fuente (JS) y se generará automáticamente una estructura de páginas HTML que el desarrollador pueda seguir para observar la API del componente.

---

3. Instalación de herramientas

La siguiente sección aclara los pasos necesarios para instalar todas estas herramientas en nuestro proyecto y así poder generar la documentación necesaria.

3.1 Consideraciones generales iniciales

- Es importante aclarar que, de momento, una de las herramientas (CustomElementMerger) sólo funciona en linux por los scripts que se utilizan, así que si nuestro sistema operativo es Windows se necesitará ejecutarlo con WSL (Subsistema de Windows para Linux).
- Otra consideración previa a la instalación de estas herramientas es que están almacenadas en la DML del SAS, por lo que se se debe estar registrado en la misma previamente para poder descargarlas (guía de registro).

3.2 Instalación

Desde consola, estando en la raíz del proyecto, se ejecutará lo siguiente:

- **web-component-analyzer (WCA)**

```
npm install -g web-component-analyzer
```

- **CustomElementMerger** (siempre usar la versión más actual de la herramienta en el momento de su instalación)

```
npm install @sas/custom-element-merger@1.0.3
```

- **JSDOC**

```
npm install @sas/custom-element-merger@1.0.3
```

A parte de instalar este paquete, se ha de configurar para que haga lo que se necesita a nivel de documentación. Para ello descargar el siguiente fichero (.jsdoc.zip) y descomprimir en la raíz del proyecto.

El contenido de este fichero es una carpeta (.jsdoc) cuyo contenido es el siguiente:

Los ficheros que contiene el archivo zip son los siguientes:

- **.jsdoc.config.json**: Contiene toda la configuración de la herramienta. Toda la información acerca de este fichero, su generación y sus distintas secciones, se podrá encontrar aquí.
- **README.md**: Fichero markdown con las cabeceras personalizadas para incluir en el fichero raíz del JSDOC (index.html).
- **plugins/cssprop.js**: Código añadido al JSDOC para que pueda analizar y documentar el tag @cssprop dentro de todos los ficheros que se analicen.

---

4. Cómo realizar correctamente la documentación del WC

El objetivo de esta sección es facilitar al desarrollador el modo de realizar los comentarios dentro del código fuente del WC para que dichos comentarios puedan ser parseados por WCA (para storybook) y por JSDoc (HTML Doc) y que generen la documentación necesaria en cada uno de sus sistemas. Se intentará unificar de una manera lógica todos los comentarios para que estos sean lo mas precisos posibles dentro del modelo Model-View-ViewModel de la manera que se explica a continuación:

4.1 Consideraciones generales iniciales

Observaciones previas que toda documentación de componentes debe cumplir:

- Dentro de la vista se deberá definir el nombre con el que se referenciará al componente de la siguiente manera:

```
window.customElements.define("<nombre-componente>", <ClaseViewDelComponente>);
```

y siempre como última línea del documento. **No se deberá hacer nunca con el decorador @customElement** :

- Para que el storybook tome del custom-element.json la documentación y la adapte a las PropsTables del addon DOCS, añadir a **.storybook/preview.js** el siguiente código:

```
import { setCustomElementsManifest } from '@storybook/web-components';
import customElements from '../custom-elements.json';

setCustomElementsManifest(customElements);
```

- Añadir a las stories de los distintos componentes:

```
export default {
  title: 'Nombre de la story',
  component: 'nombre-componente', // El que se encuentra también como nombre en el fichero `custom-
elements.json`
};
```

4.2 Tags disponibles para la generación de comentarios

La siguiente tabla incluye todos los tags disponibles para comentar el código fuente:

Tag	Definición	Scope	Visible en	Ejemplo

@element	Se usa para asociar el nombre del componente con la documentación	Ficheros View, ViewModel, CSS	Storybook	**Simple property doc example**
				`/**`
				`*`
				`* @element my-custom-element`
				`*`

@file	Se utiliza para indicar una descripción del fichero	Todos los ficheros	JSDOC	**Simple property doc example**
				`/**`
				`*`
				`* @file`
				`<shortFileDescription>`
				`*`

@author	Se usa para identificar el autor de un fichero o elemento	Todos los ficheros	JSDOC	**Simple property doc example**
				`*`
				`* @author <authorName>`
				`[<authorEmail>]`
				`*`

@version	Se usa para documentar la versión de un fichero o elemento	Todos los ficheros	JSDOC	**Simple property doc example**
				`*`
				`* @version`
				`<versionNumber>`
				`*`

@class	Se usa para identificar una clase	Todos los ficheros que incluyan clases	JSDOC	**Simple property doc example**
				`*`
				`* @class [<type>]`
				`[<name>]`
				`*`

@classdesc	Se usa para dar la descripción de una clase, y suele usarse en combinación con el tag @class (o @constructor)	Todos los ficheros que incluyan clases	JSDOC	**Simple property doc example**
				`*`
				`* @class [<type>]`
				`[<name>]`
				`* @classdesc`
				`<classDescription>`
				`*`
				NOTA: Para usar **@classdesc** es obligatorio haber definido el tag **@class** previo a este

@extends	Se usa para indicar que un elemento hereda de otro elemento padre	Todos los ficheros que incluyan clases y extiendan de otra clase padre	JSDOC	**Simple property doc example**
				`*`
				`* @extends <namepath>`
				`*`

@interface	Se usa para documentar una interfaz que otros elementos podrían implementar	Ficheros Model, ViewModel, Utils o ficheros de interfaz	JSDOC	**Simple property doc example** `* * @interface [<name>] *`
@function	Se usa para marcar un elemento como función o método	Ficheros Model, ViewModel y Utils	JSDOC	**Simple property doc example** `* * @function [<functionName>] *`
@param	Se usa para establecer nombre, tipo y descripción de un parámetro de una función	Ficheros Model, ViewModel y Utils con parámetros definidos dentro de las funciones	JSDOC	**Simple property doc example** `* * @param [{type}] <name> [- <description>] *`
@return	Se usa para describir el tipo y resultado devuelto por una función o método	Ficheros Model, ViewModel y Utils con funciones que devuelvan algún resultado	JSDOC	**Simple property doc example** `* * @return [{type}] [<description>] *`
@static	Se usa para indicar que un elemento esta contenido en su elemento padre y puede ser accedido sin crear una instancia del padre	Todos los ficheros que definan elementos estáticos	JSDOC	**Simple property doc example** `* * @static *`
@protected	Se usa para indicar que un elemento es protegido	Todos los ficheros que definan elementos protegidos	JSDOC	**Simple property doc example** `* * @protected *`
@fires	Se usa para documentar eventos	Ficheros Model, ViewModel y Utils	Storybook y JSDOC	**Simple property doc example** `* * @fires <name> [- <description>] *`

@slot	Se usar para documentar slots (área dentro del componente que permite la inyección de HTML)	Ficheros Model, ViewModel	Storybook	**Simple property doc example** `* * @slot <name> [%- <description>] *`
@attr	Se usa para documentar los atributos (propiedades internas al componente - Internal properties) del componente	Todos los ficheros que definan atributos sobre algunos de sus elementos	Storybook y JSDOC	**Complex property doc example** `/** * Attribute description * @attr [{type}] attrTest [=defaultValue] - Attribute description **/ attrTest=defaultvalue`

@prop	Se usa para documentar las propiedades (propiedades expuestas al exterior a través de la API) del componente	Todos los ficheros que definan propiedades sobre algunos de sus elementos	Storybook y JSDOC	**Simple property doc example**
				`/** * Property description **/ @property({type: propertyType}) propName=defaultvalue`
				Complex property doc example
				`/** * Property description * @prop [{propertyType}] propName[=defaultValue] - Property description **/ @property({type: propertyType}) propName=defaultvalue`

@cssprop	Se usa para documentar las propiedades custom CSS del componente	Ficheros CSS	Storybook	**Simple property doc example** `* * @cssprop --<name>= [defaultValue] [- <description>] *`
@csspart	Se usa para documentar CSS shadow parts (áreas que permiten el uso de propiedades CSS dentro del shadow dom) del componente	Ficheros CSS	Storybook	**Simple property doc example** `* * @cssprop <name> *`

IMPORTANTE:

- Dentro de la documentación, siempre que se usen los tags **@attr** o **@prop** y se quiera definir su tipo con {type}, este **siempre deberá escribirse en minúsculas**, aunque el tipo sea una clase (Boolean - boolean). Solo se admitirá CamelCase en el caso que el tipo sea un Literal Type.

NOTA: Para los ejemplos, indicar:

- Los símbolos de mayor y menor que **< >** se usan para delimitar los elementos, es decir, para indicar que se necesita un nombre **<name>** o una descripción **<description>.**
- Si un elemento está entre corchetes **[ ]** indica que dicho elemento es opcional - por ejemplo, en **@fires <name> [- <description>] <name>** es obligatorio y **<description>** es opcional -.
- Las llaves se usan para definir elementos determinados que los generadores de documentación puedan interpretarlos como tal - por ejemplo el tipo de un elemento **{type}** siempre se define entre llaves.

4.3 Estructura de los comentarios dentro de los ficheros

- **Siempre se dispondrán los comentarios con la siguiente estructura para ficheros Typescript o Javascript**:

A nivel de ejemplo, la estructura global de los comentarios dentro del fichero será la siguiente:

```
/**
* @file STIC Button, Patron MVVM -> View
*
* @author STIC Arquitectura
*
* @version 0.0.3
*/

import { html } from 'lit-element';
import { ButtonTheme } from './css/ButtonTheme.css';
import '@material/mwc-button';
import '@material/mwc-icon';
import { WcSticButtonViewModel } from './WcSticButtonViewModel';

/**
*
* @element wc-stic-button
*
* @class WcSticButtonView
*
* @classdesc La Vista define cómo la información y las funcionalidades se mostrarán gráficamente.<br>
* Tiene la responsabilidad de definir la estructura que saldrá de la pantalla del componenete
WCSticButton.
*
* @extends WcSticButtonViewModel
*/
export class WcSticButtonView extends WcSticButtonViewModel {

static getStyles() {
  return [ButtonTheme.buttonTheme];
}

render() {
  return html`
    <mwc-button
      .label=${this.label} .icon=${this.icon} ?trailingIcon=${this.trailingIcon}
      ?disabled=${this.disabled} ?dense=${this.dense} ?fullwidth=${this.fullwidth}
      ?outlined=${this.outlined} ?raised=${this.raised} ?unelevated=${this.unelevated}></mwc-button>
    `;
  }
}

window.customElements.define('wc-stic-button', WcSticButtonView);
```

Y dentro de esta estructura, profundizando sobre cada uno de sus elementos:

- Se deberá incluir un bloque de comentarios al inicio del mismo para describir su contenido y características globales del mismo:

```
/**
* @file <descripción a alto nivel del fichero>
*
* @author <nombreAutor>
*
* @version <versionFichero> (versionado semántico)
*/
```

comentario que se situará en la primera línea del fichero y llegará hasta la línea anterior de los imports.

- En todos los ficheros que incluyan documentación para incluir en storybook se deberá incluir el siguiente bloque de comentarios **OBLIGATORIO**:

```
/**
* Definición del componente (puede ser multilínea)
*
* @element [nombre-customElement-componente] (Nombre que se le da al componente en la definición del customElement)
*
*/
```

ya que con **@element** se asocia el nombre del customElement de ese componente a la documentación que aparece en el storybook.

Este bloque de comentarios vendrá a continuación de los imports y, por norma general, llegará hasta la definición de la clase exportada del fichero.

Es importante aclarar que el bloque de comentarios anterior y el bloque de comentarios de definición de la clase **HAN DE SER UN ÚNICO BLOQUE DE COMENTARIOS** (por necesidades de los parseadores de documentación WCA y JSDOC), de modo que, por ejemplo, quedaría de la siguiente manera:

```
/**
*
* @element wc-stic-button
*
* @class WcSticButtonView
*
* @classdesc La Vista define cómo la información y las funcionalidades se mostrarán gráficamente.<br>
* Tiene la responsabilidad de definir la estructura que saldrá de la pantalla del componenete WCSticButton.
*
* @extends WcSticButtonViewModel
*/
```

y, bajo ningún concepto, podría quedar así:

```
/**
*
* @element wc-stic-button
*
*/

/**
*
* @class WcSticButtonView
*
* @classdesc La Vista define cómo la información y las funcionalidades se mostrarán gráficamente.<br>
* Tiene la responsabilidad de definir la estructura que saldrá de la pantalla del componenete
WCSticButton.
*
* @extends WcSticButtonViewModel
*/
```

- Para todas elementos de las tags que se han definido en el punto 4.2, siempre llevarán un comentario justo en la línea de encima del mismo, comentario que incluirá una descripción del elemento y todas las tags que deban usarse para comentar el mismo.

Por ejemplo, dentro del ViewModel, donde se definen las propiedades:

```
/**
* @prop {string} label - Texto del botón.
*/
@property()
protected label: string = "";
```

- En caso que se necesite documentar algún elemento que no esté implícito en el código fuente, se definirá el elemento dentro del bloque de comentarios con la definición del componente y el **@element** (por ejemplo).

```
/**
*
* @element wc-stic-button
*
* @prop {string} [variant=primary] - Permite seleccionar el conjunto de valores "primarios" o
"secundarios" mediante el selector de clases definido a tal efecto.
*
* @cssprop [--stic-button-background=--theme-color-background] - Color de superficie del botón.
*
*/
```

---

5. Scripts en package.json para automatizar su uso

Una vez instaladas las herramientas (punto 3) y documentados los componentes (punto 4) se crearán una serie de scripts en el archivo package.json de la raíz del proyecto para automatizar la generación de documentación.

Estos scripts se generarán dentro del package.json del storybook de desarrollo de monorepo en que se está trabajando.

Dentro del fichero **package.json** se han creado los elementos dentro de la sección **scripts**:

```
"jsdoc:generate:docs": "jsdoc --configure .jsdoc/.jsdoc.config.json --verbose"

"storybook:generate:docs": "wca analyze sas/**/dist/src/** --outFile ./custom-elements.json && wcamerger
--inputFile 'custom-elements.json' --outputFile 'custom-elements.json' --makeJsDocsChips --jsdocURI
'http://storybook.dev.alm.sas.junta-andalucia.es/storybook/docs'"

"storybook": "npm run storybook:generate:docs && npm run jsdoc:generate:docs && start-storybook -p 6006"
```

Estos scripts son la para la parte del run storybook (dev), mientras que los siguientes:

```
"build-storybook": "npm run storybook:generate:docs && build-storybook --modern"

"build": "npm run jsdoc:generate:docs && npm run build-storybook"
```

Son para el build de storybook y su generación de ficheros estáticos.

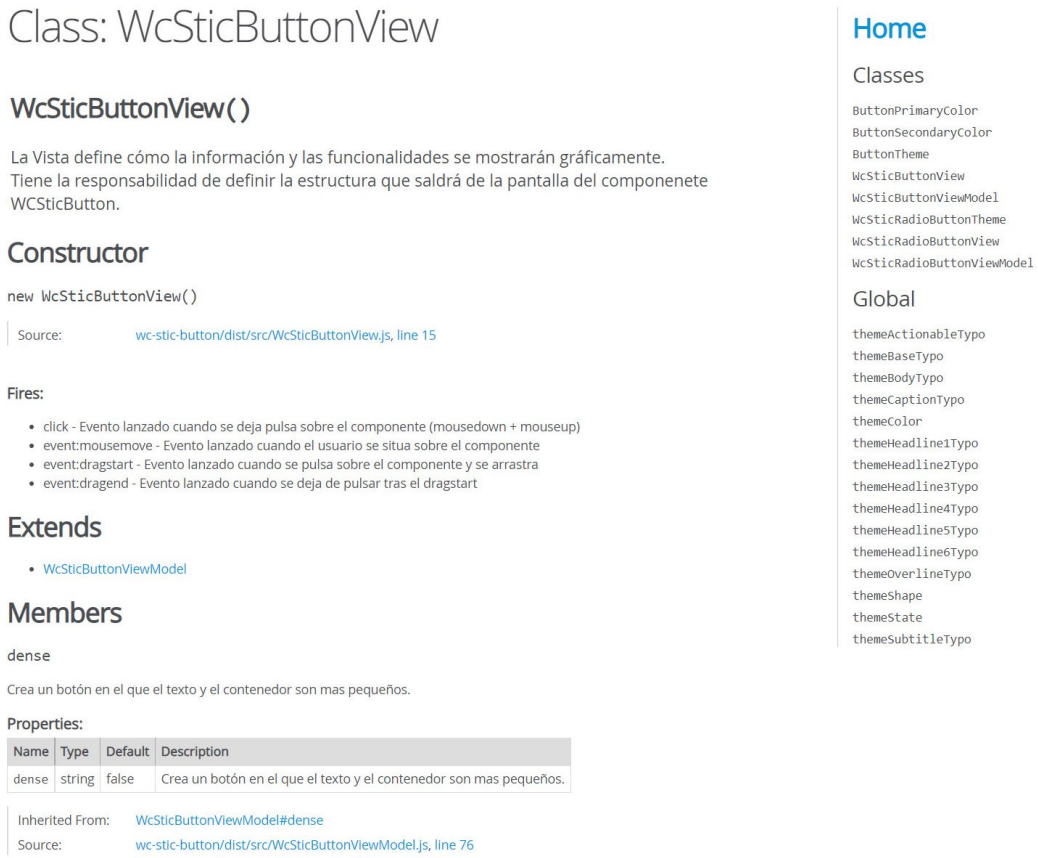
En un análisis más pormenorizado de los scripts:

```
"jsdoc:generate:docs": "jsdoc --configure .jsdoc/.jsdoc.config.json --verbose"
```

Este primer script genera toda la documentación mediante la herramienta JSDOC. Se usan los siguientes flags:

- **--configure**: Sirve para indicar dónde se encuentra el fichero de configuración personalizado de JSDOC.
- **--verbose**: Sirve para observar por consola, durante el proceso de generación de la documentación, todos los mensajes que indique la herramienta.

Esto nos genera una carpeta **jsdocs** en la raiz del proyecto mediante la cual se puede ver toda la documentación generada por la herramienta. Para ello (de manera local) se puede abrir el fichero **index.html (Home)** dentro de esta carpeta y navegar por la documentación. A modo ejemplo esta es una captura de la documentación generada por JSDOC en el storybook de DEV:



Class: WcSticButtonView

WcSticButtonView()

La Vista define cómo la información y las funcionalidades se mostrarán gráficamente.
Tiene la responsabilidad de definir la estructura que saldrá de la pantalla del componenete WCSticButton.

Constructor

new WcSticButtonView()

Source: wc-stic-button/dist/src/WcSticButtonView.js, line 15

Fires:

- click - Evento lanzado cuando se deja pulsa sobre el componente (mousedown + mouseup)
- event:mousemove - Evento lanzado cuando el usuario se situa sobre el componente
- event:dragstart - Evento lanzado cuando se pulsa sobre el componente y se arrastra
- event:dragend - Evento lanzado cuando se deja de pulsar tras el dragstart

Extends

- WcSticButtonViewModel

Members

dense

Crea un botón en el que el texto y el contenedor son mas pequeños.

Properties:

Name	Type	Default	Description
dense	string	false	Crea un botón en el que el texto y el contenedor son mas pequeños.

Inherited From: WcSticButtonViewModel#dense
Source: wc-stic-button/dist/src/WcSticButtonViewModel.js, line 76

Home

Classes

ButtonPrimaryColor
ButtonSecondaryColor
ButtonTheme
WcSticButtonView
WcSticButtonViewModel
WcSticRadioButtonTheme
WcSticRadioButtonView
WcSticRadioButtonViewModel

Global

themeActionableTypo
themeBaseTypo
themeBodyTypo
themeCaptionTypo
themeColor
themeHeadline1Typo
themeHeadline2Typo
themeHeadline3Typo
themeHeadline4Typo
themeHeadline5Typo
themeHeadline6Typo
themeOverlineTypo
themeShape
themeState
themeSubtitleTypo

```
"storybook:generate:docs": "wca analyze sas/**/dist/src/** --outFile ./custom-elements.json && wcamerger --inputFile 'custom-elements.json' --outputFile 'custom-elements.json' --makeJsDocsChips --jsdocURI 'http://storybook.dev.alm.sas.junta-andalucia.es/storybook/docs'"
```

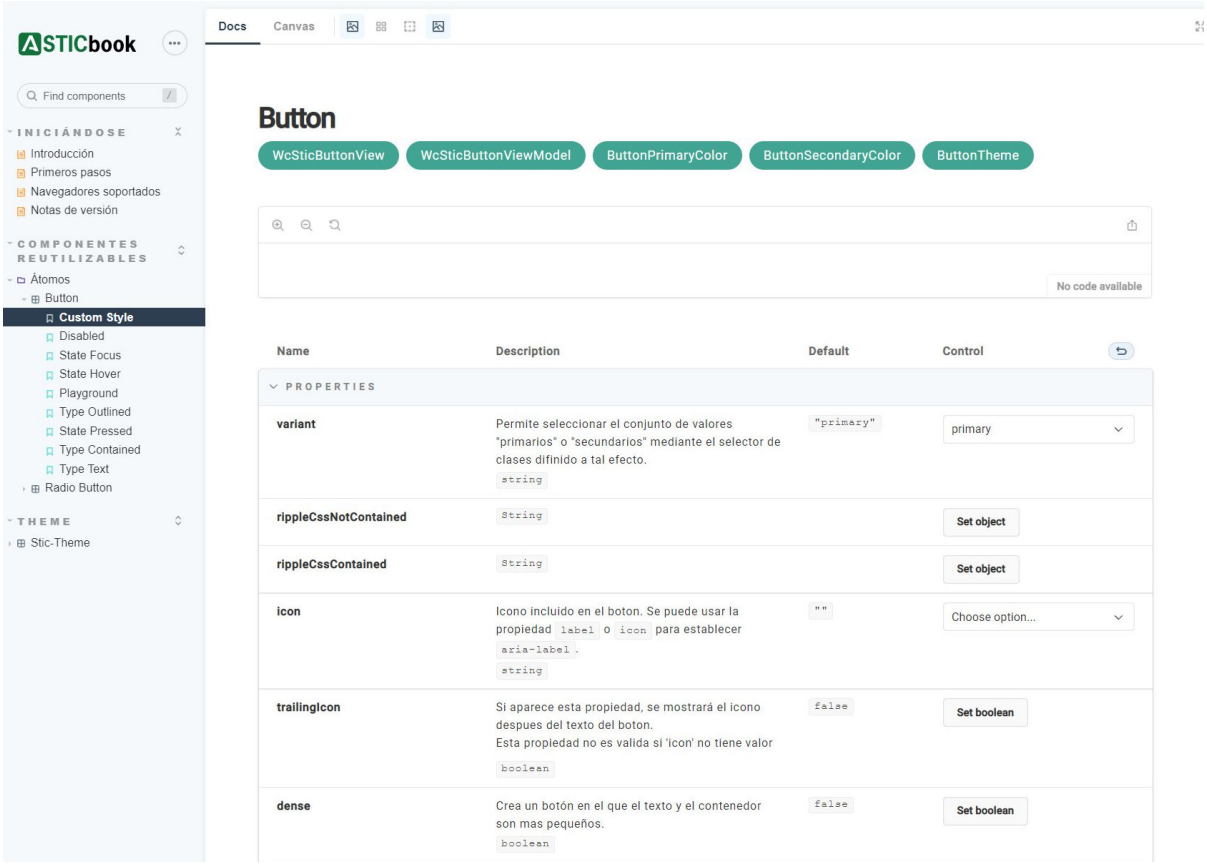
Este segundo script une las herramientas WCA y CEM de la siguiente manera:

- La primera parte del script (**wca analyze sas/**/dist/src/** --outFile ./custom-elements.json**) utiliza WCA para generar toda la documentación en el fichero **custom-element.json**, cuya ruta de salida se indica a través del flag (**--outFile**).
- La segunda parte del script (**wcamerger --inputFile 'custom-elements.json' --outputFile 'custom-elements.json' --makeJsDocsChips --jsdocURI 'http://storybook.dev.alm.sas.junta-andalucia.es/storybook/docs'**) usa CEM y utiliza los siguientes flags:
  - **--inputFile**: Indica cual es el fichero de entrada a analizar
  - **--outputFile**: Indica cual es el fichero de salida con las tags mergeadas.
  - **--makeJsDocsChips**: Genera, en la descripción del componente dentro del addon DOCS de storybook, unas chips enlazadas con la documentación de las clases generada por JSDOC.
  - **--jsdocURI 'http://storybook.dev.alm.sas.junta-andalucia.es/storybook/docs'**: Se indica, para las chips generadas, cual sera la URI de todos los elementos generados por JSDOC. Como ejemplo, se ha

puesto la URI del storybook de DEV, pero si se quiere ejecutar de manera local se podría hacer con (**http://localhost:6006/docs/**).

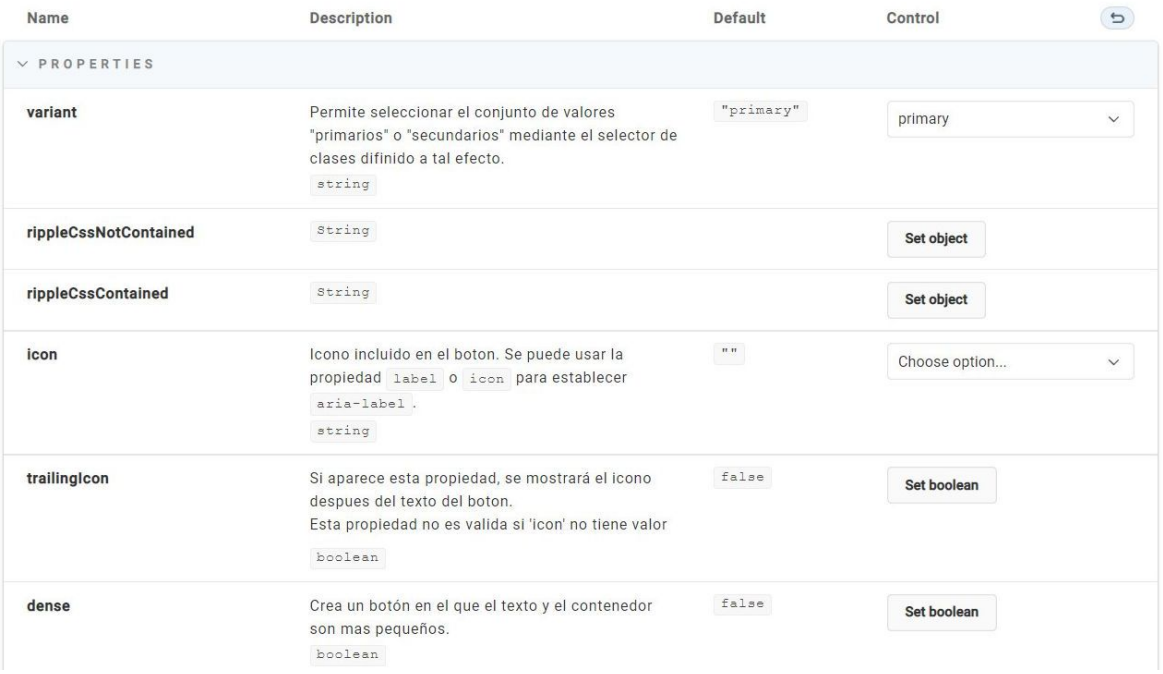
```
"storybook": "npm run storybook:generate:docs && npm run jsdoc:generate:docs && start-storybook -p 6006"
```

Este tercer script sirve para ejecutar los dos scripts anteriores y lanzar el storybook de manera local, con lo que una vez finalizada su ejecución, si se accede a un componente, en la pestaña DOCS del mismo aparecerá toda la documentación generada por WCA y CEM en el fichero custom-elements.json, si como las chips de las clases generadas por JSDOC:



Destacando los **chips de la parte superior** :

Y también la zona de **documentación de la API (imagen parcial)** :



Name	Description	Default	Control
PROPERTIES			
variant	Permite seleccionar el conjunto de valores "primarios" o "secundarios" mediante el selector de clases difinido a tal efecto. `string`	`"primary"`	primary
rippleCssNotContained	`String`		Set object
rippleCssContained	`String`		Set object
icon	Icono incluido en el boton. Se puede usar la propiedad `label` o `icon` para establecer `aria-label` . `string`	`""`	Choose option...
trailingIcon	Si aparece esta propiedad, se mostrará el icono despues del texto del boton. Esta propiedad no es valida si 'icon' no tiene valor `boolean`	`false`	Set boolean
dense	Crea un botón en el que el texto y el contenedor son mas pequeños. `boolean`	`false`	Set boolean

En la que se puede ver cómo se **integra en Storybook toda la documentación generada por WCA en el fichero custom-element.json y que posteriormente se une para todas las TAGS del componente a través de CustomElementMerger** .

---

6. Enlaces de interés

- ***Storybook en PRO***
- ***JSDOC en PRO***