Testing

Subdirección de las Tecnologías de la Información y Comunicaciones

Área de Gobernanza y Calidad

Dirección General de Tecnologías de la Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Versión

Guia de Testing en el Frontend **Versión:** 1.0.0 **Estado**: Vigente **Entrada en vigor desde: 13 nov 2024**	**Guia de Testing en el Frontend** **Versión:** 1.0.0 **Estado**: PRE-RELEASE **Entrada en vigor desde: 11 oct 2023**

Tabla de Contenido

Uso de las Guías de Desarrollo

Las guías de desarrollo son un documento que busca facilitar el onboarding y la homogeneidad de los proyectos en la STIC. 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 guías 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 guía 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 documentados, 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
1.0.0	1.0.0	13 nov 2024	-	-	• Versión inicial del documento

Version	Date	Author	Comment
21	13-11-02024	LUIS MARTINEZ FONTIVEROS	Publicación

Introducción

En este documento se describirán los principales aspectos a tener en cuenta a la hora de diseñar y ejecutar test en los desarrollos frontales, que serán ejecutables dentro de los procesos de IC de la STIC.

El código de los ejemplos compartidos en esta guía de diseño se puede encontrar en el espacio público

Configuración

Este apartado pretende compartir algunas configuraciones consideradas de utilidad para una correcta ejecución de los test.

Web test runner

- Dependencias
    - @open-wc/testing
    - @types/mocha
    - @web/test-runner
    - @web/test-runner-playwright
    - sinon
    - playwright
- Fichero de configuración del runner
    - Esta configuración permite la ejecución de los test desde typescript, realizando la transpilación de forma automática.
    - También está configurado para funcionar en un monorepo
    - También permite la correcta ejecución en contenedores, requisito indispensable para la entrega en la plataforma de Jenkins

web-test-runner.config.mjs

```
// Este archivo es la configuración de Web Test Runner. Aquí puedes configurar los plugins, los
navegadores, los reporters, etc.
import { esbuildPlugin } from '@web/dev-server-esbuild';

import { playwrightLauncher } from '@web/test-runner-playwright';
import { defaultReporter } from '@web/test-runner';

function playwrightLauncherExtended(options, browserName) {
  const launcher = playwrightLauncher(options);
  launcher.name = browserName??launcher.name;
  return launcher;
}

export default {
  plugins: [
    esbuildPlugin({
      ts: true, tsconfig: 'tsconfig.dev.json', // Asegúrate de que esbuild use tu tsconfig.json
      loaders: {
        '.ts': 'ts',
      }
    }),

  ],
  browsers: [
    playwrightLauncherExtended({
      product: 'chromium',
      launchOptions: {
        channel: 'chrome',
      },
    }, 'Chrome'), // Lanza Chromium
    playwrightLauncherExtended({
      product: 'chromium',
      launchOptions: {
        channel: 'msedge',
      },
    }, 'Microsoft Edge'),
    playwrightLauncherExtended({
      product: 'firefox',
    }, 'Mozilla Firefox') // Lanza Firefox
  ],
  reporters: [defaultReporter({ reportTestResults: true, reportTestProgress: true })],
  nodeResolve: true,
  preserveSymlinks: true,
  rootDir: '.',
  files: ['test/**/*.test.ts'],
  mimeTypes: {
    '**/*.ts': 'js',
  }
};
```

package.json

package.json

```
{
  ....
  "scripts": {
        "test": "wtr --coverage",
    "test:debug": "wtr --debug --watch",
    "build:dev":"tsc --project tsconfig.dev.json",
    "build":"npm i -ddd && tsc",
    "postinstall": "npx playwright install --with-deps"
  },
  "devDependencies": {
....
    "@open-wc/testing": "4.0.0",
    "@types/mocha": "10.0.9",
    "@web/dev-server-esbuild": "1.0.2",
    "@web/test-runner": "0.19.0",
    "@web/test-runner-playwright": "0.11.0",
    "lit": "2.7.4",
    "playwright": "1.48.1",
    "sinon": "19.0.2",
    "typescript": "5.6.3"
....
  },
 ...
}
```

tsconfig.json

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es2018",
    "module": "esnext",
    "moduleResolution": "node",
    "noEmitOnError": true,
    "lib": ["es2017", "dom"],
    "strict": true,
    "esModuleInterop": false,
    "allowSyntheticDefaultImports": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "importHelpers": true,
    "outDir": "dist",
    "sourceMap": true,
    "inlineSources": true,
    "rootDir": ".",
    "declaration": true,
    "declarationMap": true
  },
  "include": ["./src/**/*.ts"],
  "exclude": [
    "./dist","./node_modules"
  ]
}
```

tsconfig.dev.json

tsconfig.dev.json

```
 {
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "declarationDir": "./types/"
  }
}
```

Ejemplos

Componente de ejemplo base para el test

Codigo de un WC para los test

```
import { html, LitElement } from "lit";
import { property } from "lit/decorators.js";
export class Example extends LitElement{

    @property({ type: String, attribute: 'some-attribute'})
    public someProperty: string = "default value";
    constructor() {
        super();
    }
    connectedCallback() {
        super.connectedCallback();
        console.debug(`Cargado componente  ${this.tagName} `);
    }
    render(): any {
        return html`<h2>componente creado</h2>
        <p id="property-value">${this.someProperty}</p>`;
    }
}
customElements.define('example-test-component', Example);
```

Renderizar un componente

Ejemplo de test de renderizado de un WC

```
//Importante que los import vengas de @open-wc/testing para que tener la máxima compatibilidad con lit-element
import { defineCE, expect, fixture, fixtureCleanup, html, unsafeStatic} from "@open-wc/testing";
//Importamos el componente
import { Example } from "../index";

describe("Testing Lit WebComponent ", () => {
    //Se restaura el sandbox y se limpia el fixture después de cada test
    afterEach(() => {
      fixtureCleanup();
    });

    //test de renderizado del componente example-test-component
    it("should render a web component", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
    });
    it("should render a web component with random test tag", async () => {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag}></${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
    });
});
```

Propiedades Reactivas

Alerta Acoplamiento

//Cuidado!!!!! este código genera acoplamiento al html generado por el render del componente, no es recomendable ya que los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza correctamente, no solo que se ha definido

expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');

Ejemplo Testing de propiedades del componente

```
//Importante que los import vengas de @open-wc/testing para que tener la máxima compatibilidad con lit-element
import { defineCE, elementUpdated, expect, fixture, fixtureCleanup, html, unsafeStatic} from "@open-wc/testing";
//Importamos el componente
import { Example } from "../index";

describe("Testing Lit WebComponent ", () => {

    //creación de un sandbox para el mock
    before(() => {
    });
    //Se restaura el sandbox y se limpia el fixture después de cada test
    afterEach(() => {
      fixtureCleanup();
```

```
    });

    //test de renderizado del componente example-test-component
    it("should render a default value of property", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;

      expect(el.someProperty).to.equal('default value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');
    });
    it("should render a default value of property in a web component with random test tag", async () => {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag}></${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('default value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');
    });
    it("should render a initialize value property", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component .someProperty=${'init value'} ><
/example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
    });
     it("should render a initialize value property in a web component with random test tag", async () =>
{
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} .someProperty=${'init value'}></${unsafeTag}
>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
    });
 it("should render a changed value property", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component .someProperty=${'init value'} ><
/example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
```

```
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      el.someProperty = 'new value';
      await elementUpdated(el);
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');

    });
      it("should render a changed value property in a web component with random test tag", async () => {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} .someProperty=${'init value'}></${unsafeTag}
>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      el.someProperty = 'new value';
      await elementUpdated(el);
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });

   it("should render a reactive changed value property", async () => {
      let propertyReactiveBind = 'init value';
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component .
someProperty=${propertyReactiveBind} ></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      propertyReactiveBind = 'new value';
      //Tenemos que hacer esto ya que no existe un método para actualizar la propiedad de forma reactiva
a través del html desde un entorno de test.
      el.someProperty = propertyReactiveBind;
      await elementUpdated(el);
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });
    it("should render a reactive changed value property in a web component with random test tag",
async () => {
      let propertyReactiveBind = 'init value';
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} .someProperty=${propertyReactiveBind}><
/${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
```

```
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      propertyReactiveBind = 'new value';
       //Tenemos que hacer esto ya que no existe un método para actualizar la propiedad de forma reactiva a través del html desde un entorno de test.
      el.someProperty = propertyReactiveBind;
      await elementUpdated(el);
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });
});
```

Atributos Reactivos

> **Alerta Acoplamiento**
>
> //Cuidado!!!!! este código genera acoplamiento al html generado por el render del componente, no es recomendable ya que los test deben ser funcionales. Nos sirve para testear que el valor del atributo se renderiza correctamente, no solo que se ha definido
>
> expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');

Ejemplo Testing de atributos del componente

```
//Importante que los import vengas de @open-wc/testing para que tener la máxima compatibilidad con lit-element
import { defineCE, elementUpdated, expect, fixture, fixtureCleanup, html, unsafeStatic} from "@open-wc/testing";
//Importamos el componente
import { Example } from "../index";

describe("Testing Lit WebComponent ", () => {

    //creación de un sandbox para el mock
    before(() => {
    });
    //Se restaura el sandbox y se limpia el fixture después de cada test
    afterEach(() => {
      fixtureCleanup();
    });

    //test de renderizado del componente example-test-component
    it("should render a default value of attribute", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.not.exist;
      expect(el.someProperty).to.equal('default value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');
    });
```

```
    it("should render a default value of attribute in a web component with random test tag", async () =>
{
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag}></${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.not.exist;
      expect(el.someProperty).to.equal('default value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('default value');
    });
    it("should render a initialize value attribute", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component some-attribute=${'init value'} ><
/example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
    });
     it("should render a initialize value attribute in a web component with random test tag", async ()
=> {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} some-attribute=${'init value'}><
/${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
    });
    it("should render a changed value attribute", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component some-attribute=${'init value'} ><
/example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      el.setAttribute('some-attribute', 'new value');
      await elementUpdated(el);
      expect(el.getAttribute('some-attribute')).to.equal('new value');
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
```

```
    });
    it("should render a changed value attribute in a web component with random test tag", async () => {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} some-attribute=${'init value'}><
/${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      el.setAttribute('some-attribute', 'new value');
      await elementUpdated(el);
      expect(el.getAttribute('some-attribute')).to.equal('new value');
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });
    it("should render a reactive changed value attribute", async () => {
      let propertyReactiveBind = 'init value';
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component some-
attribute=${propertyReactiveBind} ></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      propertyReactiveBind = 'new value';
        //Tenemos que hacer esto ya que no existe un método para actualizar la propiedad de forma
reactiva a través del html desde un entorno de test.
        el.setAttribute('some-attribute', propertyReactiveBind);
      await elementUpdated(el);
      expect(el.getAttribute('some-attribute')).to.equal('new value');
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });
    it("should render a reactive changed value attribute in a web component with random test tag",
async () => {
      let propertyReactiveBind = 'init value';
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag} some-attribute=${propertyReactiveBind}><
/${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      expect(el.getAttribute('some-attribute')).to.equal('init value');
      expect(el.someProperty).to.equal('init value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('init value');
      propertyReactiveBind = 'new value';
```

```
      //Tenemos que hacer esto ya que no existe un método para actualizar la propiedad de forma reactiva
a través del html desde un entorno de test.
      el.setAttribute('some-attribute', propertyReactiveBind);
      await elementUpdated(el);
      expect(el.getAttribute('some-attribute')).to.equal('new value');
      expect(el.someProperty).to.equal('new value');
      //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
      expect(el.shadowRoot?.getElementById("property-value")?.textContent).to.equal('new value');
    });
});
```

Captura de eventos

Ejemplo de test de captura de eventos emitidos por un componente

```
//Importante que los import vengas de @open-wc/testing para que tener la máxima compatibilidad con lit-
element
import { defineCE, expect, fixture, fixtureCleanup, html, oneEvent, unsafeStatic} from "@open-wc
/testing";
//Importamos el componente
import { Example } from "../src/index";

describe("Testing Lit WebComponent ", () => {
    //Se restaura el sandbox y se limpia el fixture después de cada test
    afterEach(() => {
      fixtureCleanup();
    });

    //test de renderizado del componente example-test-component
    it("should receive a web component custom event", async () => {
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<example-test-component></example-test-component>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      setTimeout(() => el.fireEvent());

      const { detail } = await oneEvent(el, 'example-event');
      expect(detail).to.deep.equal ({ message: "Hello world" });
    });
    it("should receive a web component custom event with random test tag", async () => {
      const tag = defineCE(class extends Example{});
      const unsafeTag = unsafeStatic(tag);
      //await fixture espera hasta que se renderice el componente
      const el = await fixture< Example >(html`<${unsafeTag}></${unsafeTag}>`);
      expect(el).to.exist;
      // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
      expect(el.shadowRoot).to.exist;
      setTimeout(() => el.fireEvent());

      const { detail } = await oneEvent(el, 'example-event');
      expect(detail).to.deep.equal({ message: "Hello world" });
    });
});
```

Renderizar un componente customizado

Ejemplo de test de componente customizado

```
//Importante que los import vengas de @open-wc/testing para que tener la máxima compatibilidad con lit-
element
import { defineCE, expect, fixture, fixtureCleanup, html, unsafeStatic } from "@open-wc/testing";
import { LitElement } from "lit";
//Creación de un componente LitElement dinámico
const customElementWithProperties = defineCE(
  class extends LitElement{
    static properties = {
      someProperty: { type: String },
    };
    constructor() {
      super();
    }
    connectedCallback() {
      super.connectedCallback();
      console.debug(`Cargado componente  ${this.tagName} `);
    }
    render(): any {
      return html`<h2>componente creado STAND-ALONE</h2>`;
    }
  }
);

//Obtención del tag del componente para su renderizado dinámico
const customElementWithPropertiesTag = unsafeStatic(customElementWithProperties);

//Definición de la suit
describe("Testing Lit WebComponent ", () => {
  //Se restaura el sandbox y se limpia el fixture después de cada test
  afterEach(() => {
    fixtureCleanup();
  });

  //test de renderizado del componente customizado
  it("should render a web component", async () => {
     //await fixture espera hasta que se renderice el componente
    const el = await fixture< LitElement >(html`<${customElementWithPropertiesTag}><
/${customElementWithPropertiesTag}`);
    expect(el).to.exist;
    // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
    expect(el.shadowRoot).to.exist;
  });
});
```

Ejecución

La herramienta de ejecución de los test es el Web Test Runner (wtr).

La configuración ofrecida, permite la transpilación de forma directa por el wtr. Dicha transpilación se genera en memoria que, salvo algunos casos concretos (por ejemplo dentro de un iframe), no tiene implicación.

> ⚠ **Script**
>
> EL script utilizado es test ***npm run test***

Depuración

Para la depurar WTR ofrece la posiblidad de probarlo en el navegador de forma manual (opción D) y ahí poder hacer la depuración que se considere necesaria. Combinado con la opción --watch, se realizará una reejecución con cada cambio detectado.

Otra facilidad que ofrece WTR es que, de todos los test lanzados, puedes centrarte en uno solo (opción F). La lista de ficheros con test fallidos en su ejecución aparecerán en rojo.

> ⚠ **Script**
>
> EL script utilizado es test ***npm run test:debug***

Navegadores

Se puede configurar para lanzar los test en diferentes navegadores. Por normativa, los desarrollos deben ser compatibles con los navegadores evergreen (Chrome/Chromium, Microsoft Edge, Firefox). Safari no es obligatorio actualmente

Instalación de un navegador

```
npx playwright install --with-deps
```

Para ver diferentes opciones que ofrece el instalador podéis usar la opción de help

Visualización de opciones para instalar un navegador

```
npx playwright install --help
```

Con la configuración establecida se ejecutarán los test para todos los navegadores configurados.

> **Advertencia**
>
> No está permitido el uso de playwright como parte del desarrollo y definición de los test. Solo está permitido su uso como launcher.

Iframe

Advertencia sobre la carga de módulos

Un iframe genera un contexto independiente para la ejecución del código alojado dentro de él. Esto implica que se debe incluir las dependencias necesarias para la ejecución de forma manual.

Advertencia sobre el acceso a los módulos

Web Test Runner (WTR) genera la transpilación de los elemento en memoria. Esto implica que los ficheros js resultantes no están disponibles para importarlos dentro del iframe. Para poder acceder a ellos se debe:

- Antes de ejecutar un test que vaya a renderizar un componente dentro de un iframe, se deberá ejecutar una transpilación previa del código a testear.
- El import debe apuntar a la ruta donde se haya generado el código que se desea importar.

Advertencia sobre el código

El código compartido para el ejemplo solo es válido para el ámbito del testing por motivos de seguridad. En ningún caso se creará un componente que haga uso de un iframe y rellene el contenido del iframe de esta forma. Se deberá utilizar la manera tradicional (src) combinado con los CSP

Advertencia sobre el timeout

A veces la carga de iframe se ralentiza y supera los 2s de timeout, se recomiendo aumentar el timeout, como se muestra en el ejemplo

Ejemplo de test de componente renderizado dentro de un iframe

```
import { fixture, expect, html, elementUpdated} from "@open-wc/testing";
import { Example } from "../src/index";
import { LitElement } from "lit";

describe("Test Lit component inside iframe", () => {
  it("should render the Lit component inside an iframe and access shadow DOM", async function() {
    this.timeout(10000);
    // Cargar un iframe con el componente
    const el = await fixture<HTMLIFrameElement>(html` <iframe
srcdoc="${getIframeHtmlCodeForExistingComponent("example-test-component")}" width="600" height="400"><
/iframe> `);

    // Esperar a que el iframe se cargue completamente

    await new Promise((resolve) => {
      el.onload = () => resolve(true);
    });

    // Acceder al documento del iframe
    const iframeDocument = el.contentDocument!;
    const exampleComponent = iframeDocument.querySelector<Example>("example-test-component");

    // Verificar que el componente Lit se ha renderizado dentro del iframe
    expect(exampleComponent).to.exist;
```

```
    // Esperar a que se complete la actualización del componente Lit dentro del iframe
    await elementUpdated(exampleComponent!);
    // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
    expect(exampleComponent?.shadowRoot).to.exist;
    // Verificar el contenido renderizado del shadow DOM

    expect(exampleComponent?.someProperty).to.equal("default value");
    //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
    expect(exampleComponent?.shadowRoot?.getElementById("property-value")?.textContent).to.equal
("default value");
  });

  it("should render the Lit component with random tag inside an iframe and access shadow DOM", async
function() {
    this.timeout(10000);
    const tag = 'example-test-component-' + Math.random().toString(36).substring(7);
    // Cargar un iframe con el componente
    const el = await fixture<HTMLIFrameElement>(html` <iframe
srcdoc="${getIframeHtmlCodeForCustomComponent(tag)}" width="600" height="400"></iframe> `);

    // Esperar a que el iframe se cargue completamente

    await new Promise((resolve) => {
      el.onload = () => resolve(true);
    });

    // Acceder al documento del iframe
    const iframeDocument = el.contentDocument!;
    const exampleComponent = iframeDocument.querySelector<LitElement>(tag);

    // Verificar que el componente Lit se ha renderizado dentro del iframe
    expect(exampleComponent).to.exist;

    // Esperar a que se complete la actualización del componente Lit dentro del iframe
    await elementUpdated(exampleComponent!);
    // si tiene shadowRoot es que es un webcomponent y que se ha renderizado de forma correcta
    expect(exampleComponent?.shadowRoot).to.exist;
    //Cuidado!!!!! este código genera acoplamiento al render del componente, no es recomendable ya que
los test deben ser funcionales. Nos sirve para testear que el valor de la propiedad se renderiza
correctamente, no solo que se ha definido
    expect(exampleComponent?.shadowRoot?.getElementById("html-iframe-rendered")?.textContent).to.equal
("testing");
  });
});

const getIframeHtmlCodeForExistingComponent = (componentName: string) => {
  return `<html>
    <head>
        <!-- Incluir los módulos de Lit desde el propio servidor de testing-->
        <script type='module' src='${window.location.origin}/node_modules/lit/index.js'></script>
        <!-- Importar el componente -->
        <script type='module'>
        import '/dist/src/index.js';
        </script>
    </head>
    <body><${componentName}></${componentName}></body></html>`;
};

const getIframeHtmlCodeForCustomComponent = (componentName: string) => {
  return `<html>
    <head>

        <script type='module'>
       //Incluir los módulos de Lit desde el propio servidor de testing
        import { LitElement, html } from "${window.location.origin}/node_modules/lit/index.js";
```

```
    //Definir el componente
        customElements.define(
          '${componentName}',
          class extends LitElement {
               static get properties() {
                  return {
                    somePropertyInsideIframe: { type: String }
                  };
                }
                constructor() {
                  super();
                }
                connectedCallback() {
                  super.connectedCallback();
                  console.log("Cargado componente ${componentName} dentro del iframe");
                }
                //Importante los caracteres especiales se deben escapar con \ para que no se interpreten
como caracteres especiales
                render() {
                  return html\`<p id='html-iframe-rendered'>testing</p>\`;
                }
              }
        );
        </script>
    </head>
    <body><${componentName}></${componentName}></body></html>`;
};
```

Recursos

- @openwc/testing
- @openwc/testing-helpers
- @openwc/testing - accesibilidad
- @openwc/testing - semantic assertion
- eslint - accesibilidad
- ejemplos

Normativa

Normativa	Descripción	Versión Activa	Versión Pre-Release	Fecha de Pre-release
Normas de diseño estructural	Documento normativo y pautas sobre el uso de Atomic Web Design	1.0	1.0	-
Web Components	Documento normativo y pautas sobre el desarrollo de web components	1.0	1.0	pte de especificar
Dominio del color	Documento normativo y pautas sobre el uso del color en los web components a través del theming	1.0	1.0	pte de especificar
Dominio de las tipografías	Documento normativo y pautas sobre el uso de los distintos elementos tipográficos en los web components a través del theming	1.0	1.0	pte de especificar
Dominio de los estados	Documento normativo y pautas sobre el uso de los estados en los web components a través del theming	1.0	1.0	pte de especificar
Dominio de las formas	Documento normativo y pautas sobre el uso de las formas en los web components a través del theming	1.0	1.0	pte de especificar
Testing	Documento normativo y pautas sobre el testing	pte de especificar	pte de especificar	pte de especificar

Guías de desarrollo

Guía Implementación	Descripción	Versión Activa	Versión Pre-Release	Fecha de Pre-release
Paradigma atomic web design	Guia de implementación en base a Atomic Web-Design	1.0	1.0	-
Documentar los web components	Guía de implementación acerca de cómo documentar correctamente los web components con las herramientas propuestas	1.0	1.0	pte de especificar
Implementación del theme en los distintos ámbitos	Guía de implementación sobre la implementación y uso del theme en los web components y su extensión a otras áreas /proyectos	1.0	1.0	pte de especificar
Modelo model–view–viewmodel en web components	Guía de implementación sobre los distintos elementos que se deben tener en cuenta en el modelo model-view-modelview	1.0	1.0	pte de especificar
Scaffolding	Guía de implementación sobre la instalación, uso y desarrollo con el Scaffolding de la STIC	1.0	1.0	pte de especificar
Testing	Guía de implementación sobre el testing	pte de especificar	pte de especificar	pte de especificar

Guías de diseño

Guía de diseño	Descripción	Versión Activa	Fecha de activa	Versión Pre-Release	Fecha de Pre-release
Sistema de diseño	Espacio de referencia donde se establecen las pautas que deben seguir los diseños de las aplicaciones en el sas			1.5	1 ene 2024