

PRÁCTICA EVALUABLE 7



Miguel Ángel Salcedo Guijarro, 2º DAW

ÍNDICE

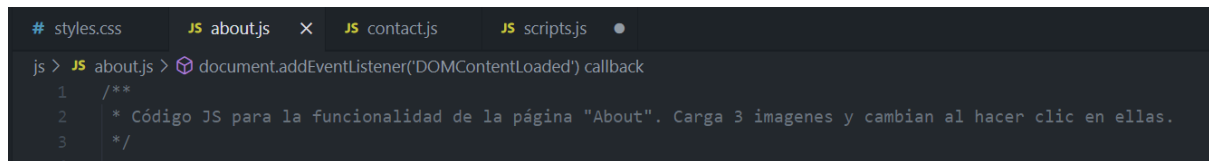
1.- Comentario de código con JSDoc.	3
1.1.- Fichero “about.js”.	3
1.2.- Fichero “contact.js”.	4
1.3.- Fichero “script.js”.	5
2.- Visualización de JSDoc en navegador.	6
3.- Instalar Git	7
4.- Subir proyecto a GitHub desde Git.	8
4.1.- Subir ficheros a una nueva rama.	10
BIBLIOGRAFÍA	11

1.- Comentario de código con JSDoc.

- En esta práctica se realizará la documentación de varios ficheros de código haciendo uso de JSDoc, herramienta que nos permite crear documentación completa de código JavaScript para facilitar su comprensión y uso por parte de otros desarrolladores.

1.1.- Fichero “about.js”.

- El primer paso, es comentar el propósito general del fichero (**Figura 1**). En este caso, es un fichero dedicado a mostrar información sobre la web en sí.



```
# styles.css JS about.js X JS contactjs JS scripts.js
js > JS about.js > document.addEventListener('DOMContentLoaded') callback
1  /**
2   * Código JS para la funcionalidad de la página "About". Carga 3 imagenes y cambian al hacer clic en ellas.
3   */
```

Figura 1.- Propósito general del fichero “about.js”.

- Después, comento las partes del código más relevantes, separándolas por secciones, haciendo que cada comentario del código se enfoque en partes específicas del código (**Figura 2**). Se comentan, entre otras cosas, la creación de una lista, o una función al hacer clic.



```
document.addEventListener('DOMContentLoaded', () => {
  /**
   * Lista de imágenes.
   * @type {string[]}
   */
  const images = [
    './images/imagen1.png',
    './images/imagen2.jpg',
    './images/imagen3.png'
  ];

  /**
   * Índice de la imagen actual.
   * @type {number}
   */
  let currentIndex = 0;

  /**
   * Carga la primera imagen.
   * @type {HTMLImageElement}
   */
  const imageElement = document.createElement('img');
  imageElement.src = images[currentIndex];

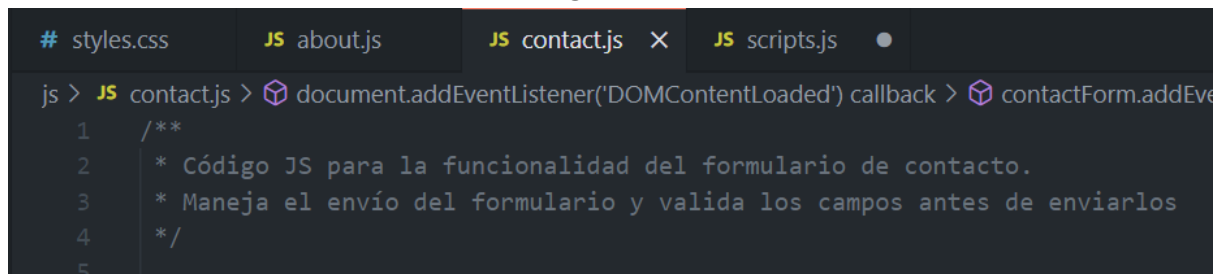
  /**
   * Añade la imagen al documento.
   */
  document.querySelector('.content').appendChild(imageElement);

  /**
   * Cambia la imagen al hacer clic.
   * @param {Event} event - El evento de clic.
   */
  imageElement.addEventListener('click', (event) => {
    currentIndex = (currentIndex + 1) % images.length;
    imageElement.src = images[currentIndex];
  });
});
```

Figura 2.- Comentario del resto de código del fichero “about.js”, separado por secciones.

1.2.- Fichero “contact.js”.

- Se repite el proceso con el fichero “**contact.js**”. Al igual que antes, empezamos haciendo un resumen sobre lo que trata el fichero (**Figura 3**).



```
# styles.css JS about.js JS contact.js X JS scripts.js
js > JS contact.js > document.addEventListener("DOMContentLoaded") callback > contactForm.addEvent
1 /**
2  * Código JS para la funcionalidad del formulario de contacto.
3  * Maneja el envío del formulario y valida los campos antes de enviarlos
4  */
5
```

Figura 3.- Propósito general del fichero “contact.js”.

- Hacemos igual que antes, comentamos el resto del código, dividiéndolo por secciones, haciendo que cada parte que comentamos del código sea una sección del mismo (**Figura 4**). Comentamos, en mi caso, partes como por ejemplo el formulario, el botón “submit” o tratar errores.



```
document.addEventListener('DOMContentLoaded', () => {
  /**
   * Elemento del formulario de contacto.
   * @type {HTMLFormElement}
   */
  const contactForm = document.getElementById('contactForm');

  /**
   * Maneja el evento de envío del formulario.
   * @param {Event} e - El evento de envío del formulario.
   */
  contactForm.addEventListener('submit', (e) => {
    e.preventDefault(); // Previene la recarga de la página

    /**
     * Valor del campo del nombre obtenido del formulario.
     * @type {string}
     */
    const name = document.getElementById('name').value;

    /**
     * Valor del campo del mensaje obtenido del formulario.
     * @type {string}
     */
    const message = document.getElementById('message').value;

    // Verifica si el nombre y el mensaje no están vacíos
    if (name.trim() !== '' && message.trim() !== '') {
      alert(`¡Gracias por tu mensaje, ${name}!`);
      contactForm.reset(); // Resetea el formulario después del envío
    } else {
      /**
       * Mensaje de alerta si el nombre o el mensaje están vacíos.
       * @type {string}
       */
      const errorMessage = 'Por favor, completa todos los campos.';
      alert(errorMessage);
    }
  });
});
```

Figura 4.- Comentario del resto de código del fichero “contact.js”, separado por secciones.

1.3.- Fichero “script.js”.

- Por último, tenemos el fichero “script.js”. Aquí, comentaremos dos clases: “Task” y “TaskManager”.
- La clase “Task” representa una tarea individualmente, con su texto y el estado actual (completada o no).
- La clase “TaskManager” gestiona las tareas, permitiendo agregar, eliminar o cambiar el estado de las mismas.
- Se comentan también las funciones, como renderizar las tareas en una lista.
- Se comentan eventos de la interfaz de usuario, como agregar nuevas tareas o borrarlas.
- Todo esto se observa en las Figuras 5 y 6.

```
class Task {
  /**
   * Crea una nueva tarea.
   * @param {string} text - El texto de la tarea.
   */
  constructor(text) {
    this.text = text; // El texto de la tarea
    this.completed = false; // Indica si la tarea está completada o no
  }
}

class TaskManager {
  /**
   * Inicializa el administrador de tareas.
   */
  constructor() {
    // Obtiene las tareas almacenadas en el localStorage, si no hay, crea una lista vacía
    this.tasks = JSON.parse(localStorage.getItem('tasks')) || [];
  }
}
```

Figura 5.- Comentario de las clases “Task” y “TaskManager”.

```
function addTask() {
  const taskInput = document.getElementById('taskInput'); // Obtiene el input de la tarea
  const text = taskInput.value.trim(); // Obtiene el texto de la tarea y elimina espacios en blanco al principio y al final
  if(text) { // Verifica si el texto no está vacío
    taskManager.addTask(text); // Agrega la tarea al administrador de tareas
    taskInput.value = ''; // Limpia el input
    renderTasks(); // Renderiza las tareas
  }
}

/**
 * Elimina una tarea.
 * @param {number} index - El índice de la tarea a eliminar.
 */
function deleteTask(index) {
  taskManager.removeTask(index); // Elimina la tarea del administrador de tareas
  renderTasks(); // Renderiza las tareas
}
```

Figura 6.- Comentarios de las funciones de agregado y borrado.

2.- Visualización de JSDoc en navegador.

- Terminado de comentar el código, hay que ejecutar varios comandos para poder ver el resultado en un navegador. Primero, instalamos JSDoc de forma global en nuestro PC. (Figura 7)

```
PS C:\Users\caeha\Desktop\2ºDAW\Despliegue\jsdoc-main> npm install -g jsdoc
```

Figura 7.- Comando de instalación de JSDoc.

- Ahora, nos situamos en la carpeta anterior a “js”, donde se sitúan todos los ficheros JavaScript, y ejecutamos el comando de la Figura 8 para generar la documentación.

```
PS C:\Users\caeha\Desktop\2ºDAW\Despliegue\jsdoc-main> jsdoc .\js\  
PS C:\Users\caeha\Desktop\2ºDAW\Despliegue\jsdoc-main>
```

Figura 8.- Generar documentación JSDoc.

- Se creará una carpeta llamada “out” con los ficheros necesarios. Al abrir el “index.html” generado, veremos toda la documentación generada, como se observa en la Figura 9.



Figura 9.- Página de inicio de la documentación JSDoc.

- En esta web de inicio, vemos en la parte derecha, las clases y funciones de nuestro programa. Al clicar en una de ellas, veremos toda la información que contienen (Figura 10)

Class: Task

Task(text)

Representa una tarea.

Constructor

new Task(text)

Crea una nueva tarea.

Parameters:

Name	Type	Description
text	string	El texto de la tarea.

Source: scripts.js, line 5

Figura 10.- Información de la clase “Task”

3.- Instalar Git

- Git es un sistema de control de versiones distribuido que permite gestionar cambios en archivos y colaborar en proyectos de desarrollo de software, facilitando el seguimiento, la colaboración y el control de versiones. Para instalarlo, nos dirigimos a su web oficial, y elegimos la versión compatible con nuestro PC (**Figura 11**) ([enlace en la bibliografía](#)).



Figura 11.- Enlaces de descarga Git.

- El proceso de instalación es simple, abrimos el instalador y aceptamos la instalación (**Figura 12**). Una vez terminada, veremos la imagen mostrada en la **Figura 13**. Si marcamos la primera opción, el programa se abrirá nada más acabar la instalación.

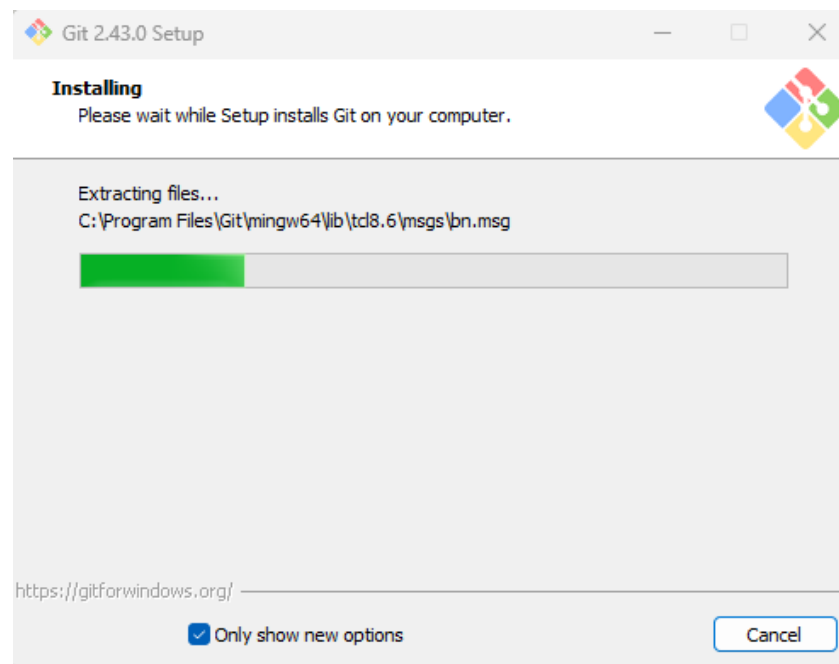


Figura 12.- Instalación de Git.

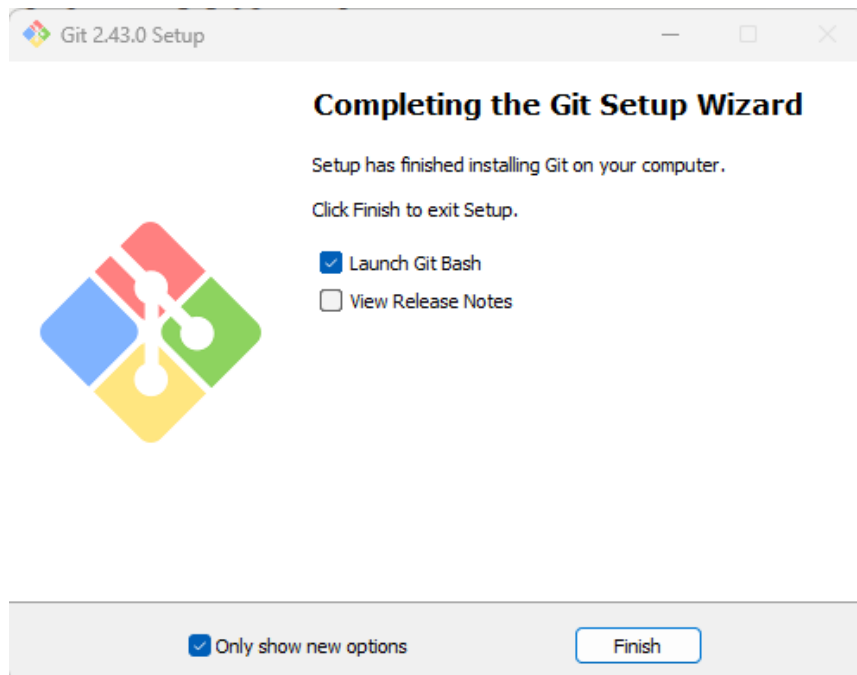


Figura 13.- Instalación de Git finalizada.

4.- Subir proyecto a GitHub desde Git.

- Para subir un repositorio a GitHub, desde Git, debemos añadir nuestro usuario y correo. (Figura 14).

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~  
$ git config --global user.name "Salcedin"  
  
caeha@DESKTOP-TC6FTIJ MINGW64 ~  
$ git config --global user.email "billy.talent01@gmail.com"
```

Figura 14.- Añadir usuario en Git.

- El siguiente paso es dirigirnos a la carpeta donde tengamos nuestro trabajo. Una vez ahí, ejecutamos el comando “git init” para inicializar un repositorio local. Después, agregamos los archivos con “git add .” (Figura 15)

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main  
$ git init  
Initialized empty Git repository in C:/Users/caeha/Desktop/2ºDAW/Despliegue/  
jsdoc-main/.git/  
  
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main (master)  
$ git add .
```

Figura 15.- Inicializar repositorio y añadir ficheros.

- Ahora, debemos confirmar los cambios con el comando “commit”, seguido de un mensaje descriptivo (Figura 16).


```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main (master)
$ git commit -m "Proyecto de JSDoc"
```

Figura 16.- Confirmar cambios.

- A continuación, creamos el repositorio en nuestro perfil de GitHub. Es importante no añadir “readme”, “.gitignore” ni licencia (**Figura 17**).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * / Repository name *
Salcedin / JSDoc
JSDoc is available.

Great repository names are short and memorable. Need inspiration? How about [special-enigma](#) ?

Description (optional)
Práctica JSDoc

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:
☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

☐ You are creating a public repository in your personal account.

Create repository

Figura 17.- Crear repositorio en GitHub.

- Creado el repositorio, hay que vincularlo con el repositorio local, para ello, hacemos el comando de la **Figura 18**.

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main (master)
$ git remote add origin https://github.com/Salcedin/JSDoc
```

Figura 18.- Vincular repositorio con el repositorio local.

- Finalmente, subimos los cambios al repositorio de GitHub con el comando “push” (**Figura 19**).

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main (master)
$ git remote add origin https://github.com/Salcedin/JSDoc
```

Figura 19.- Subir cambios a GitHub.

- Comprobamos que se ha completado el proceso correctamente desde nuestro perfil de GitHub (**Figura 20**).



Figura 20.- Repositorio subido correctamente.

4.1.- Subir ficheros a una nueva rama.

- En este punto, subiremos los ficheros generados por JSDoc a una nueva rama. Lo primero, será dirigirnos a la carpeta donde están estos ficheros, y ejecutar los comandos de las **Figuras 15 y 16** para añadir los archivos al repositorio.

- A continuación, creamos la nueva rama con el comando de la **Figura 21**.

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main/out (master)
$ git checkout -b documentacion-jsdoc
```

Figura 21.- Crear una nueva rama en nuestro repositorio.

- Agregamos el repositorio al que deseamos subir la carpeta utilizando el comando de la **Figura 22**.

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main/out (documentacion-jsdoc)
$ git remote add origin https://github.com/Salcedin/JSDoc
```

Figura 22.- Agregar repositorio para subir la nueva Carpeta.

- Para terminar, subimos los cambios a la nueva rama. Para ello, ejecutamos el comando de la **Figura 23**.

```
caeha@DESKTOP-TC6FTIJ MINGW64 ~/Desktop/2ºDAW/Despliegue/jsdoc-main/out (documentacion-jsdoc)
$ git push -u origin documentacion-jsdoc
```

Figura 23.- Subir los cambios a la nueva rama.

- Desde GitHub, en el apartado “**Branches**” de nuestro repositorio, podemos confirmar que se han realizado los cambios, como se muestra en la **Figura 24**.

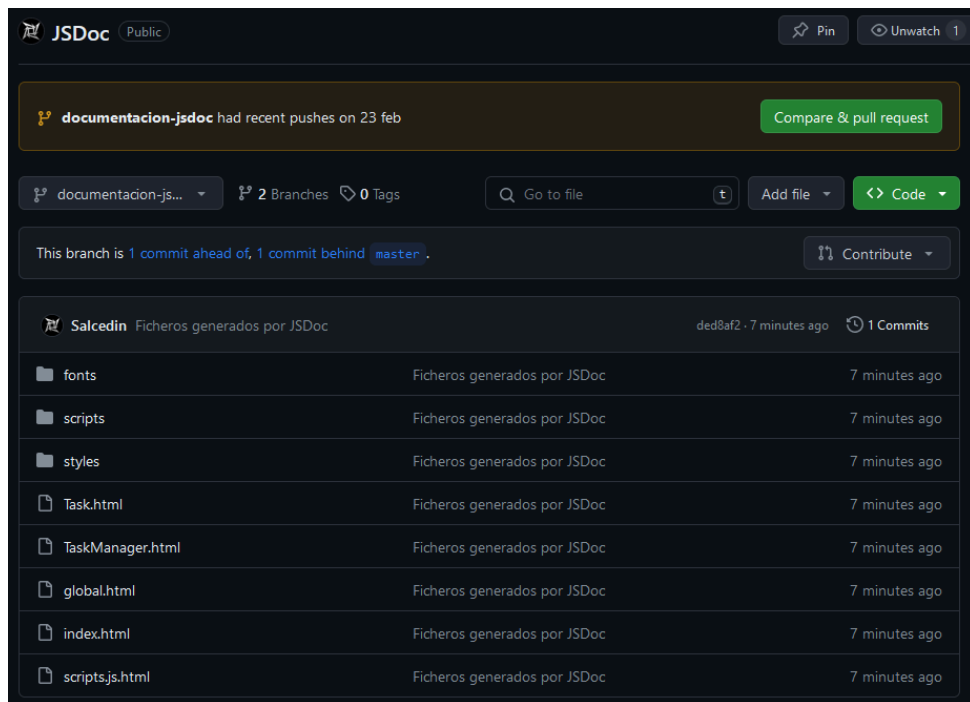


Figura 24.- Ficheros de JSDoc subidos en la nueva rama.

BIBLIOGRAFÍA

- Apuntes MEDAC.
- Enlace descarga Git: <https://git-scm.com/download/win>
- Web oficial JSDoc: <https://jsdoc.app/>