

# **UNIDAD N° 7:** **ARTEFACTOS TÉCNICOS** **REQUERIDOS**

Documentación Técnica y Código Fuente: Elementos Esenciales para el Desarrollo de Software Profesional



# ¿POR QUÉ ES VITAL LA DOCUMENTACIÓN TÉCNICA?



## COMPRENSIÓN CLARA

Facilita el entendimiento profundo del código, permitiendo que cualquier desarrollador comprenda la lógica y estructura del proyecto sin necesidad de análisis exhaustivo.



## MANTENIMIENTO EFICIENTE

Simplifica las tareas de actualización y corrección de errores, reduciendo significativamente el tiempo necesario para realizar cambios en el código.



## INCORPORACIÓN RÁPIDA

Acelera la integración de nuevos desarrolladores al equipo, minimizando la curva de aprendizaje y permitiendo contribuciones productivas desde el inicio.



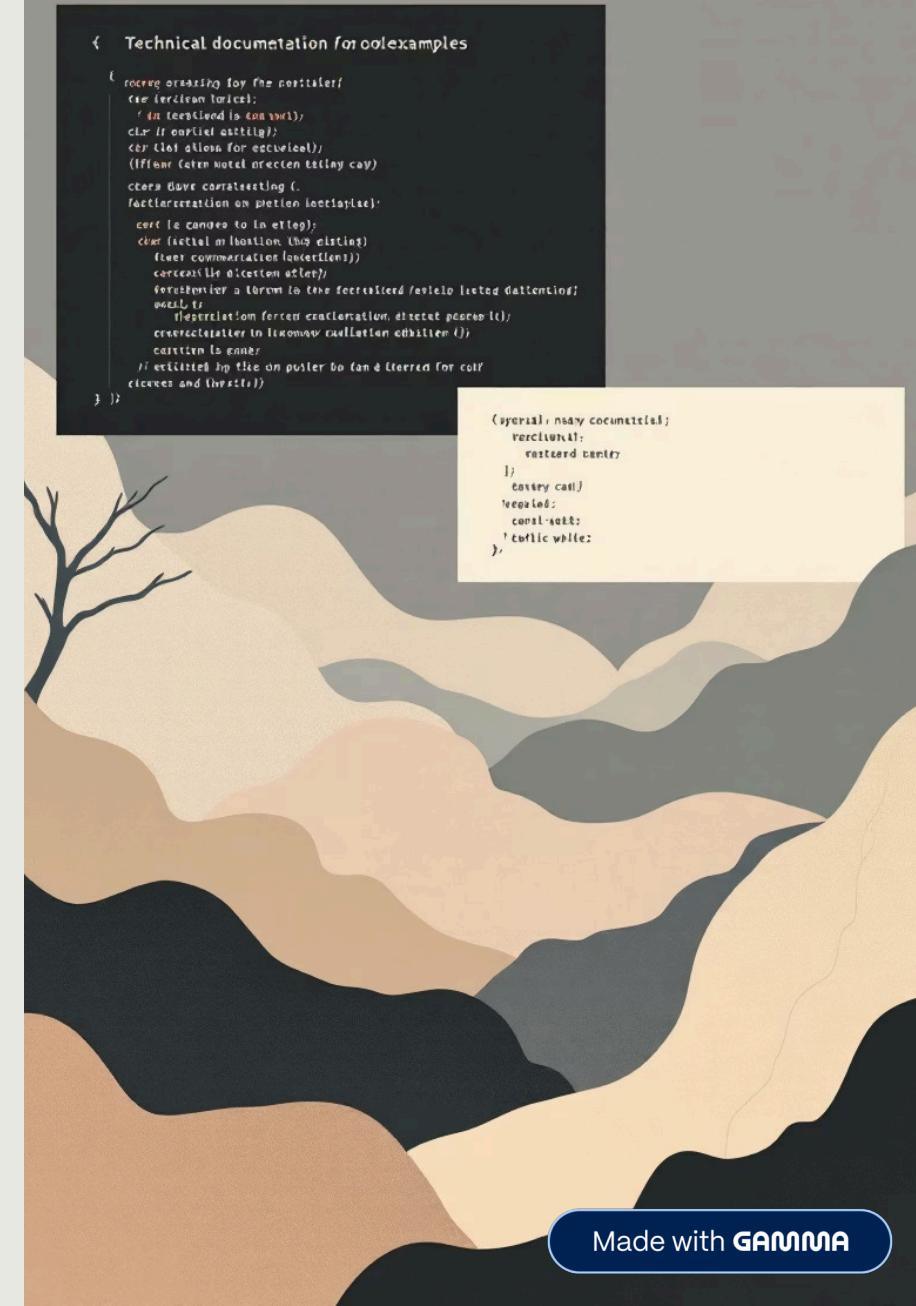
# **DOCUMENTACIÓN TÉCNICA: ¿QUÉ DEBE INCLUIR?**

# **ELEMENTOS FUNDAMENTALES**

- Descripción clara y concisa de funciones, clases y módulos
  - Especificación detallada de parámetros de entrada
  - Valores de retorno esperados con sus tipos
  - Ejemplos prácticos de uso en contextos reales

# **BUENAS PRÁCTICAS**

- Mantener estructura organizada y coherente
  - Actualizar documentación junto con el código
  - Incluir casos de uso y escenarios comunes
  - Documentar excepciones y comportamientos especiales



# JsDoc

```
/**  
 * Multiplies the input value by 2.  
 *  
 * @param {number} input The value or range of cells to multiply.  
 * @return {number} The input multiplied by 2.
```

```
{  
  "description": "Multiplies the input value by 2.",  
  "tags": [  
    {  
      "tag": "param",  
      "type": "number",  
      "name": "input",  
      "description": "The value or range of cells to multiply."  
    }  
  ]  
}
```

## INTRODUCCIÓN A JSDOC: DOCUMENTA TU CÓDIGO JAVASCRIPT

JsDoc es la herramienta estándar para crear documentación profesional automática en proyectos JavaScript y TypeScript.

θ1

### COMENTARIOS ESPECIALES

Utiliza bloques de comentarios con sintaxis específica que describen funciones, parámetros y tipos de datos de manera estructurada.

θ2

### INTEGRACIÓN CON VS CODE

Mejora significativamente IntelliSense, proporcionando autocompletado inteligente y validación en tiempo real durante el desarrollo.

θ3

### GENERACIÓN AUTOMÁTICA

Crea documentación HTML profesional de forma automática, manteniendo la coherencia y reduciendo el esfuerzo manual.

# ESTRUCTURA BÁSICA DE UN COMENTARIO JSDOC

Los comentarios JSDoc siguen una estructura específica que permite documentar código de manera clara y consistente:

```
/**  
 * Calcula el área de un rectángulo  
 * @param {number} ancho - El ancho del rectángulo  
 * @param {number} alto - El alto del rectángulo  
 * @returns {number} El área calculada  
 * @example  
 * // Retorna 50  
 * calcularArea(10, 5);  
 */  
  
function calcularArea(ancho, alto) {  
    return ancho * alto;  
}
```

- ❑ **Nota importante:** Los comentarios JSDoc siempre comienzan con `/**` y cada línea dentro del bloque utiliza un asterisco (\*) al inicio para mantener la estructura visual.

# ETIQUETAS COMUNES DE JSDOC



## @PARAM

Define los parámetros de entrada de una función, incluyendo tipo y descripción detallada.



## @RETURNS / @RETURN

Especifica el tipo y descripción del valor que retorna la función.



## @THROWS

Documenta las excepciones que puede lanzar la función en casos de error.



## @EXAMPLE

Proporciona ejemplos prácticos de cómo utilizar la función correctamente.



## @DEPRECATED

Marca funciones obsoletas que no deberían usarse en nuevo código.



## @ASYNC

Indica que la función es asíncrona y retorna una Promise.



## @TYPEDEF

Define tipos de datos personalizados para usar en la documentación.

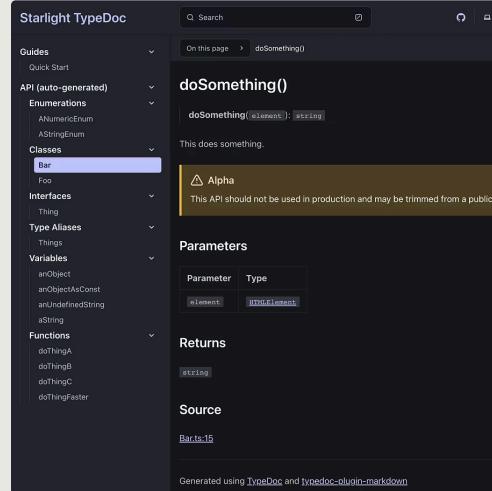


## @PROPERTY

Describe las propiedades de un objeto o tipo personalizado.

# ¿CÓMO SE GENERA DOCUMENTACIÓN AUTOMÁTICAMENTE?

```
/**  
 * Registers a new pet into the Pet Adoption Center.  
 *  
 * @memberOf Pets  
 * @param {Pets.Pet} pet - The pet.  
 * @returns {Promise<Transactions.TransactionResponse>} Details about the new pet.  
 * @example  
 * const sdk = require('pet-adoption-center');  
  
 * const transaction = sdk.pets.register({  
 *   name: 'Atticus',  
 *   type: sdk.PET_TYPES.DOG,  
 *   breed: sdk.DOG_BREEDS.FRENCH_BULLDOG,  
 *   dateOfBirth: '2010-03-10',  
 *   neutered: false  
 * });  
  
register: async function (pet : Pets.Pet) {  
  return {  
    success: true,  
    transactionId: '1234-5678-90',  
    data: {  
      // Placeholder for the pet object  
    }  
};
```



## JSDOC CLI

Herramienta de línea de comandos que genera documentación HTML completa y navegable directamente desde los comentarios del código fuente.

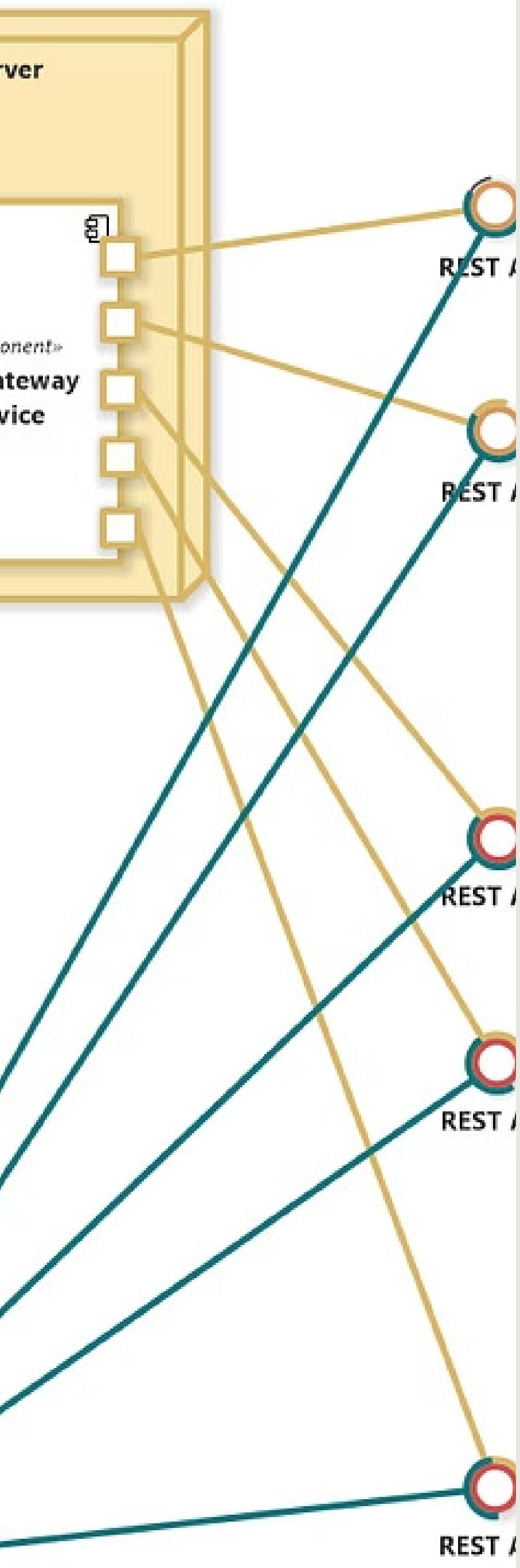
## TYPEDOC

Solución especializada para proyectos TypeScript que aprovecha el sistema de tipos para crear documentación más precisa y detallada.

```
src/vs/base/common  
export class Action extends EventEmitter implements IAction {  
  
  static LABEL: string = 'label';  
  static TOOLTIP: string = 'tooltip';  
  static CLASS: string = 'class';  
  static ENABLED: string = 'enabled';  
  static CHECKED: string = 'checked';  
  
  public _id: string;  
  public _label: string;  
  public _tooltip: string;  
  public _cssClass: string;  
  public _enabled: boolean;  
  public _checked: boolean;  
  public _actionCallback: IActionCallback;  
  public _order: number;  
  
  constructor(id: string, label: '', cssClass: '', enabled: true, actionCallback: IActionCallback = null) {  
    super();  
  
    this._id = id;  
    this._label = label;  
    this._cssClass = cssClass;  
    this._enabled = enabled;  
    this._actionCallback = actionCallback;  
  }  
  
  public getId(): string {  
    return this._id;  
  }  
  
  public get label(): string {  
    return this._label;  
  }  
  
  public set label(value: string) {  
    this._setLabel(value);  
  }  
}  
0
```

## DOCUMENT THIS (VS CODE)

Extensión que completa bloques JSDoc automáticamente, generando plantillas basadas en la firma de funciones y reduciendo el trabajo manual.



# GENERACIÓN AUTOMÁTICA DE DIAGRAMAS UML DESDE VS CODE

Visual Studio Code ofrece potentes herramientas para crear y mantener diagramas UML integrados en el flujo de desarrollo:

## DRAW.IO INTEGRATION

```

graph TD
    Start((Start)) --> Wait[Wait]
    Wait -- "mouse click(target lockon)" --> Ready[Ready]
    Ready -- "buffer 1 frame" --> Wait
    Ready -- "time > 25sec" --> SaveVID[SaveVID]
    SaveVID -- "fy_diff >= 8.0" --> Ready
    SaveVID -- "enter 'esc' key" --> Start

```

Extensión que permite crear diagramas visuales directamente en VS Code, con sincronización automática y exportación a múltiples formatos.

## PLANTUML

```

@startuml
[*] --> Wait: mouse click(target lockon)
Wait --> Ready: buffer 1 frame
Ready --> Wait: time > 25sec
Ready --> [*]: enter "esc" key
[*] --> SaveVID: SaveVID
SaveVID --> Ready: fy_diff >= 8.0
SaveVID --> [*]: enter "esc" key
Ready --> [*]: exit

```

Herramienta basada en texto que genera diagramas UML mediante código, ideal para versionado y documentación as-code con sintaxis simple.

## GITHUB COPILOT

Asistente de IA que sugiere documentación y puede generar descripciones automáticas de código, acelerando el proceso de documentación.