Documentation avec JSDoc & PHPDoc : Cours & Bonnes pratiques

Introduction

La documentation du code est essentielle pour garantir sa maintenance, sa compréhension et faciliter le travail collaboratif. Les outils comme JSDoc pour JavaScript et PHPDoc pour PHP permettent d'écrire des commentaires structurés qui aident les IDEs (comme VSCode) à fournir des infos, autocomplétion, vérifications de types, etc.

Partie 1 : Documentation en JavaScript avec JSDoc

Exemple complet et amélioré

```
"use strict";

/**
 * Cette fonction dit "bonjour" au prénom donné.
 *
 * @param {string} prenom Le prénom à saluer.
 * @returns {void} Ne renvoie rien.
 */
function bonjour(prenom) {
    console.log("Bonjour " + prenom);
}

bonjour("Paul");
```

Documentation avec types multiples, valeurs de retour et erreurs possibles

```
/**
 * Fait un console log de salutation.
 *
 * @param {string} prenom Le prénom de la personne.
 * @param {number|string} age L'âge, peut être un nombre ou une chaîne.
 * @returns {void}
 */
function salutation(prenom, age) {
    console.log("Bonjour, je m'appelle " + prenom + " et j'ai " + age + " ans.");
}

salutation("Maurice", 54);
```

Exemple avec fonction qui renvoie une valeur

```
/**
 * Calcule la somme de deux nombres.
 *
 * @param {number} a Premier nombre.
 * @param {number} b Deuxième nombre.
 * @returns {number} La somme des deux nombres.
 */
function addition(a, b) {
    return a + b;
}

const resultat = addition(5, 3);
console.log(resultat); // 8
```

Fonction avec gestion d'erreur documentée

```
/**
 * Divise un nombre par un autre.
 *
 * @param {number} a Numérateur.
 * @param {number} b Dénominateur.
 * @throws {Error} Lance une erreur si le dénominateur est zéro.
 * @returns {number} Le résultat de la division.
 */
function diviser(a, b) {
    if (b === 0) {
        throw new Error("Division par zéro impossible.");
    }
    return a / b;
}

try {
    console.log(diviser(10, 0));
} catch (e) {
    console.error(e.message);
}
```

Fonction dépréciée

```
/**
 * Donne l'heure.
 *
 * @param {Date} date Objet Date.
 * @deprecated Cette fonction est obsolète, utilisez une autre fonction à la
place.
 * @returns {void}
 */
function heure(date) {
```

```
    console.log("Il est " + date.getHours() + " heure");
}

heure(new Date());
```

Autres annotations utiles en JSDoc

- `@async` : Indique que la fonction est asynchrone.
- `@callback` : Décrit un type de fonction callback.
- `@typedef` : Définit un type personnalisé.
- `@property` : Décrit une propriété d'un objet.
- `@example` : Donne un exemple d'utilisation de la fonction.
- `@see` : Fournit un lien vers une documentation externe ou une autre fonction.

Exemple d'annotation avec `@example` et `@typedef`

```
/**
 * Représente un utilisateur.
 * @typedef {Object} Utilisateur
 * @property {string} nom Le nom de l'utilisateur.
 * @property {number} age L'âge de l'utilisateur.
 */

/**
 * Affiche les informations d'un utilisateur.
 *
 * @param {Utilisateur} user Objet utilisateur.
 * @returns {void}
 * @example
 * const u = { nom: "Alice", age: 30 };
 * afficherUtilisateur(u);
 */
function afficherUtilisateur(user) {
    console.log(user.nom + " a " + user.age + " ans.");
}
```

Partie 2 : Documentation en PHP avec PHPDoc

Syntaxe PHPDoc

```
<?php
/**
 * Cette fonction dit bonjour au prénom donné.
 *
 * @param string $prenom Le prénom à saluer.
 * @return void
 */
```

```php
function bonjour(string $prenom): void {
    echo "Bonjour " . $prenom;
}

bonjour("Paul");
```

Annotations courantes en PHPDoc

Annotation	Description	Exemple
`@param`	Type et description d'un paramètre	`@param int $age`
`@return`	Type de la valeur renvoyée	`@return string`
`@throws`	Exception que la méthode peut lever	`@throws \Exception`
`@deprecated`	Indique que la méthode est obsolète	`@deprecated`
`@var`	Type d'une variable ou propriété	`@var array<string>`

Exemple avec exceptions et retour

```php
/**
 * Calcule la division entre deux nombres.
 *
 * @param float $a
 * @param float $b
 * @return float
 * @throws \InvalidArgumentException Si $b est zéro.
 */
function diviser(float $a, float $b): float {
    if ($b === 0) {
        throw new \InvalidArgumentException("Division par zéro impossible.");
    }
    return $a / $b;
}
```

Documentation des variables et propriétés de classe

Les exemples précédents s'attardaient sur les fonctions, mais la documentation peut s'appliquer à d'autres éléments :

En JavaScript (JSDoc)

Une variable ou une constante :

```php
/**
 * Nombre maximum d'articles autorisés.
```

```
 * @type {number}
 */
const MAX_ARTICLES = 10;
```

Une classe, une propriété ou une méthode (même le constructor):

```
/**
 * Représente un utilisateur.
 */
class Utilisateur {
    /**
     * Le nom de l'utilisateur.
     * @type {string}
     */
    nom;

    /**
     * L'âge de l'utilisateur.
     * @type {number}
     */
    age;

    /**
     * Crée un nouvel utilisateur.
     * @param {string} nom
     * @param {number} age
     */
    constructor(nom, age) {
        this.nom = nom;
        this.age = age;
    }
}
```

En PHP (PHPDoc)

Les même exemples en PHP :

```
<?php
/**
 * Nombre maximum d'articles autorisés.
 *
 * @var int
 */
const MAX_ARTICLES = 10;

/**
 * Classe Utilisateur
 */
```

```php
class Utilisateur
{
    /**
     * Le nom de l'utilisateur.
     *
     * @var string
     */
    public string $nom;

    /**
     * L'âge de l'utilisateur.
     *
     * @var int
     */
    public int $age;

    /**
     * Constructeur.
     *
     * @param string $nom
     * @param int $age
     */
    public function __construct(string $nom, int $age)
    {
        $this->nom = $nom;
        $this->age = $age;
    }
}
```

Conclusion & Bonnes pratiques

- Toujours documenter les fonctions publiques et complexes.
- Utiliser des descriptions claires et précises.
- Indiquer les types, les valeurs de retour, les erreurs possibles et l'état de dépréciation.
- Profiter des outils d'automatisation pour générer la documentation.
- Maintenir la documentation à jour pour éviter les erreurs et faciliter la maintenance.