



NODE.JS

JOUR 6 : DOCUMENTATION DANS UN PROJET WEB



DEROULE MODULE



-
- ✓ Découverte de Node.JS
 - ✓ Développer une API REST avec Express
 - ✓ Gestion d'erreurs et middleware
 - ✓ Authentification et JWT
 - ✓ Routing et Accès
 - ✓ **Documentation dans un projet web**
-



DOCUMENTATION DANS UN PROJET WEB



LES OBJECTIFS

Identifier les différents types de documentation d'un projet web (technique, utilisateur, collaborative).

Rédiger de la documentation claire et conforme aux standards.

Savoir lire et comprendre la documentation d'un projet.

Utiliser des outils de documentation automatisée



NUAGE DE MOTS

Quel type de documentation connaissez vous ?

<https://www.menti.com/aldpp4773piz>



INTRODUCTION

POURQUOI DOCUMENTER UN PROJET WEB ?

- **Compréhension** : Facilite la lecture et la compréhension du code, même longtemps après.
- **Collaboration** : Permet aux membres d'une équipe (ou de futurs développeurs) de contribuer.
- **Tests et maintenance** : Aide à tester, corriger et améliorer le projet sans tout redécouvrir.
- **Onboarding** : Accueille plus facilement de nouveaux développeurs dans le projet.
- **Réutilisabilité** : Rendre des modules ou des API exploitables dans d'autres projets.
- **Communication** : Donne une vision claire du projet pour les utilisateurs ou les clients

LES DIFFERENTS TYPES DE DOCUMENTATION

Type	Pour qui ?	Sert à quoi ?
README.md	Tous	Introduction au projet
CONTRIBUTING.md	Développeurs	Collaborer avec des règles claires
API documentation	Frontend / Clients	Consommer l'API correctement
Commentaires / JSDoc	Développeurs	Comprendre la logique du code
Docs utilisateur	Utilisateurs finaux	Savoir utiliser l'interface ou le produit



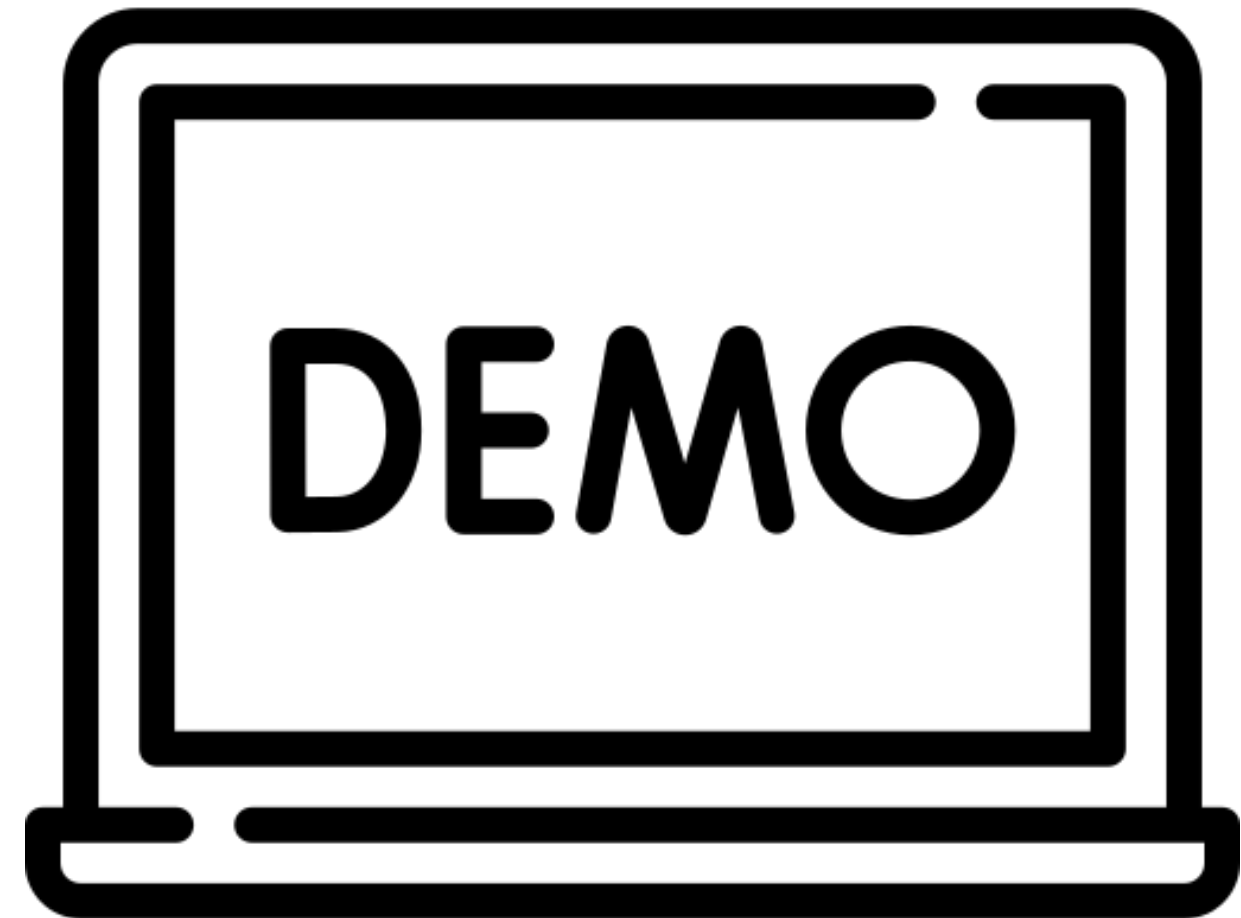
LE MARKDOWN

LE MARKDOWN C'EST QUOI

Markdown est un **langage** de **balisage léger** qui permet d'écrire du texte structuré et lisible tout en étant facilement convertible en HTML.

Il est très utilisé pour :

- **Documenter** les projets sur GitHub (fichiers .md)
- Rédiger des README, changelogs, wikis
- Écrire des contenus simples et propres sans éditeur





REDACTION TECHNIQUE

README

Qu'est ce qu'un README ?

C'est le fichier d'accueil d'un projet. Quand quelqu'un visite ton dépôt (sur GitHub, GitLab ou autre), c'est le premier document affiché. Il doit donner une vision claire et rapide du projet.

Ce fichier est écrit en Markdown (.md), un format léger pour structurer du texte (titres, listes, liens, etc.).

Pourquoi il est si important ?

Un bon README.md permet de :

- Comprendre à quoi sert le projet.
- Savoir comment l'installer et l'utiliser.
- Trouver rapidement les commandes utiles.
- Attirer des contributeurs (dans l'open source).
- Professionnaliser son code pour un employeur ou une équipe.

STRUCTURE RECOMMANDEE

1. Titre et description

En une phrase, on doit comprendre ce que fait le projet.

```
1 # MonProjet
2 Une application web pour gérer ses tâches en ligne.
```

2. Installation / Pré-requis

Expliquer les étapes à réaliser pour démarrer le projet en local

```
1 ## Installation
2
3 1. Clonez le dépôt :
4 ```bash
5 git clone https://github.com/monutilisateur/
  MonProjet.git
6 ```
7
8 2. Installer les dépendances :
9 ```bash
10 npm install
11 ```
12
13 3. Lancer le projet :
14 ```bash
15 npm start
16 ```
```

STRUCTURE RECOMMANDEE

3. Technologies utilisées

Peut aussi mentionner les bibliothèques ou les framework

```
1 ## Technologies
2
3 - HTML / CSS / JavaScript
4 - React
5 - Node.js
```

4. Fonctionnalités

Donne un aperçu clair de ce que le projet sait faire

```
1 ## Fonctionnalités
2
3 - Ajouter une tâche
4 - Supprimer une tâche
5 - Marquer comme terminée
6
```


STRUCTURE RECOMMANDEE

3. Technologies utilisées

Peut aussi mentionner les bibliothèques ou les framework

```
1 ## Technologies
2
3 - HTML / CSS / JavaScript
4 - React
5 - Node.js
```

4. Fonctionnalités

Donne un aperçu clair de ce que le projet sait faire

```
1 ## Fonctionnalités
2
3 - Ajouter une tâche
4 - Supprimer une tâche
5 - Marquer comme terminée
6
```

STRUCTURE RECOMMANDEE

5. Arborescence du projet (optionnelle)

```
1 ## Arborescence
2
3 MonProjet/
4 |
5 |— src/
6 |   |— App.js
7 |   |— utils/
8 |   |— api.js
9 |— public/
10 |   |— index.html
11 |— README.md
12
```

6. Captures d'écran (optionnel mais impactant)

```
1 ## Aperçu
2
3 ![Capture de l'application](./screenshot.png)
4
```


STRUCTURE RECOMMANDEE

7. Contribuer (si open source)

```
1 ## Contribuer
2
3 Merci pour votre aide ! Veuillez consulter le fichier
  [CONTRIBUTING.md](CONTRIBUTING.md) pour en savoir
  plus.
4
```

8. Licence

```
1 ## Licence
2
3 Ce projet est sous licence MIT.
4
```

EXEMPLE DE README MINIMAL

```
1 # Mini TODO App
2
3 Une petite application console pour gérer ses tâches.
4
5 ## Installation
6
7 Ouvrez `index.html` dans votre navigateur.
8
9 ## Fonctionnalités
10
11 - Ajouter une tâche
12 - Supprimer une tâche
13 - Afficher la liste
14
15 ## Auteurs
16
17 Fait par le groupe de stagiaires DWM 2025
18
```


BONNES PRATIQUES

Faire	Ne pas faire
Être clair et concis	Mettre du code inutile
Structurer avec des titres	Laisser le README vide
Garder à jour avec le projet	Mettre une description dépassée
Ajouter des captures si possible	Écrire tout en bloc sans sections

CONTRIBUTING

Qu'est ce qu'un CONTRIBUTING ?

Le fichier CONTRIBUTING.md explique comment contribuer à un projet.

Il est souvent utilisé dans les projets open source, mais il est tout aussi utile dans un projet d'équipe pédagogique ou professionnel.

Pourquoi il est si important ?

Il est essentiel pour :

- Encadrer les contributions (règles, étapes à suivre),
- Maintenir une qualité de code cohérente,
- Aider les nouveaux contributeurs à savoir par où commencer.

A QUOI SERT CE FICHIER EXACTEMENT

Guider les contributeurs : Explique comment installer le projet, faire une modification, proposer une amélioration.

Garder un code propre : Indique les conventions de code à respecter

Normaliser les commits : Décrit les règles de nommage des commits (ex. : feat:, fix:...).

Faciliter les revues : Précise ce qu'il faut tester ou documenter avant une pull request

STRUCTURE RECOMMANDEE

1. Installation du projet

Pour que le contributeur puisse le faire tourner en local

```
1 ## Installation
2
3 1. Forkez le projet
4 2. Clonez votre copie :
5 ```bash
6 git clone https://github.com/votre-nom/mon-projet.git
7 ```
8 3. Installez les dépendances
9 ```bash
10 npm install
11 ```
12
13 4. Lancez le projet
14 ```bash
15 npm start
16 ```
```

2. Comment contribuer

```
1 ## Comment contribuer
2
3 - Créez une issue si vous avez une suggestion ou
  un bug à signaler.
4 - Créez une nouvelle branche pour votre
  modification :
5 ```bash
6 git checkout -b ma-nouvelle-fonctionnalite
7 ```
8 - Commitez vos modifications
9 - Poussez votre branche et ouvrez une pull request
```

STRUCTURE RECOMMANDEE

3. Convention de code

```
1 ## Conventions de code
2
3 - Utilisez des noms de variables clairs et en anglais
4 - Indentation à 2 espaces
5 - Utilisez des fonctions pures si possible
6 - Commentez les fonctions complexes avec JSDoc
```

4. Convention de commit

```
1 ## Conventions de commits
2
3 Adoptez les préfixes suivants :
4 - `feat:` pour une nouvelle fonctionnalité
5 - `fix:` pour une correction de bug
6 - `refactor:` pour une amélioration de code
7 - `docs:` pour de la documentation
8
9 Exemple :
10 feat: ajout de la fonctionnalité de suppression
```

STRUCTURE RECOMMANDEE

5. Checklist avant de proposer une contribution

```
1 ## Checklist
2
3 - ☐ Le code fonctionne
4 - ☐ Aucun bug console
5 - ☐ La fonctionnalité est testée
6 - ☐ Le README est à jour si besoin
7
```


EXEMPLE DE CONTRIBUTING

```
1 # 🍌 Guide de contribution
2
3 Merci de vouloir contribuer à ce projet ! Voici les
  étapes à suivre.
4
5 ## 🚀 Installation
6
7 ```bash
8 git clone https://github.com/votre-nom/mon-projet.git
9 cd mon-projet
10 npm install
11 npm start
12 ```
13
14 ## 🗨️ Comment contribuer
15
16 1. Forkez le projet
17 2. Créez une branche (git checkout -b ma-branch)
18 3. Faites vos modifications
19 4. Commitez avec un message clair
20 5. Poussez et ouvrez une pull request
21
```

```
21
22 ## 🎨 Conventions de code
23
24 - 2 espaces pour l'indentation
25 - Variables et fonctions en anglais
26 - Commentaires JSDoc pour les fonctions
27 - Pas de console.log inutile dans le code final
28
29 ## 💬 Conventions de commit
30
31 Utilisez les préfixes suivants :
32
33 - feat: nouvelle fonctionnalité
34 - fix: correction de bug
35 - docs: mise à jour de documentation
36 - refactor: amélioration du code
37
38 Exemples :
39 feat: ajout de l'affichage des tâches terminées
40 fix: bug d'affichage dans la console
41
42 ## ✅ Checklist avant de proposer
43
44 - [ ] Code testé et fonctionnel
45 - [ ] README mis à jour si nécessaire
46 - [ ] Code propre et commenté
```

ATELIER

Réaliser un CONTRIBUTING.md pour cineclub



CONSIGNES

Nous allons reprendre notre TP de cineclub et nous allons rédiger un CONTRIBUTING en markdown comme s'il avait vocation à être open-source sujet d'améliorations.

Introduction

Présente brièvement le projet et l'objectif de ce fichier

Installation

Précise les commandes dans une instruction de code pour démarrer le projet en local

Comment contribuer

Forker le projet, créer une branche, faites vos modifications, committez avec un message clair

Convention de code

2 espaces pour l'indentation, variables et fonctions en anglais

Conventions de commit

Utilisez les prefixes suivants : feat : fix, docs, refactor...



DOCUMENTATION API

POURQUOI DOCUMENTER UNE API ?

Clarté : Permet aux développeurs de comprendre comment utiliser l'API (endpoints, méthodes, paramètres).

Maintenabilité : Facilite les mises à jour et la gestion des évolutions.

Collaboration : Facilite le travail entre équipes backend, frontend, QA, et clients.

Automatisation : Les docs OpenAPI peuvent générer des clients, des tests, des mocks automatiquement.

ELEMENTS ESSENTIELS

- **Endpoints** : URL + méthode HTTP (GET, POST, PUT, DELETE...)
- **Paramètres** : Query params, path params, body, headers
- **Réponses** : Format, codes HTTP (200, 404, 500...), messages d'erreur
- **Modèles de données** : Structure des objets envoyés et reçus (JSON)
- **Exemples d'utilisation** : Requêtes et réponses types
- **Authentication** : Comment s'authentifier (tokens, clés API...)

pet Everything about your Pets		Find out more: http://swagger.io
POST	/pet	Add a new pet to the store
PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status
GET	/pet/findByTags	Finds Pets by tags
GET	/pet/{petId}	Find pet by ID
POST	/pet/{petId}	Updates a pet in the store with form data
DELETE	/pet/{petId}	Deletes a pet
POST	/pet/{petId}/uploadImage	uploads an image

OUTILS DE DOCUMENTATION API

Swagger / OpenAPI : Spécification standard pour décrire les API REST.

Postman : Génération et tests d'API + documentation.

Apiary, Redoc, Stoplight : Outils de présentation et collaboration.

Swagger UI : Interface web interactive qui affiche la doc à partir d'un fichier OpenAPI.



POSTMAN



apiary



UTILISATION DE SWAGGER

Un fichier en JSON ou YAML

Définit toutes les routes, paramètres, réponses, et modèles

Permet de générer automatiquement la doc interactive et des clients

On peut également éditer le code en ligne à l'aide de :
<https://editor.swagger.io/>

```
1 openapi: 3.0.3
2 info:
3   title: API TODO
4   description: API pour gérer des tâches TODO
5   version: "1.0.0"
6 servers:
7   - url: http://localhost:3000/api
8 paths:
9   /todos:
10    get:
11      summary: Liste toutes les tâches
12      responses:
13        '200':
14          description: Liste des tâches
15          content:
16            application/json:
17              schema:
18                type: array
19                items:
20                  $ref: '#/components/schemas/Todo'
21    post:
22      summary: Ajoute une nouvelle tâche
23      requestBody:
24        required: true
25        content:
26          application/json:
```

LES PACKAGES SWAGGER

swagger-jsdoc

- Permet de générer automatiquement un fichier OpenAPI (Swagger) à partir de commentaires JSDoc dans votre code.
- Facile à intégrer avec Express, Koa, etc.
- Vous écrivez la doc directement dans les commentaires, et swagger-jsdoc extrait tout ça pour générer la spec.

swagger-ui-express

- Fournit une interface web interactive Swagger UI pour afficher la documentation OpenAPI.
- Souvent utilisée en conjonction avec swagger-jsdoc ou un fichier YAML/JSON OpenAPI.

```
1 const swaggerJSDoc = require('swagger-jsdoc');
2 const swaggerUi = require('swagger-ui-express');
3 const express = require('express');
4
5 const app = express();
6
7 const swaggerDefinition = {
8   openapi: '3.0.0',
9   info: {
10     title: 'API Exemple',
11     version: '1.0.0',
12     description: 'Documentation auto générée avec
    swagger-jsdoc',
13   },
14   servers: [{ url: 'http://localhost:3000' }],
15 };
16
17 const options = {
18   swaggerDefinition,
19   apis: ['./routes/*.js'], // chemins vers fichiers
    contenant les commentaires JSDoc
20 };
21
22 const swaggerSpec = swaggerJSDoc(options);
23
24 app.use('/api-docs', swaggerUi.serve,
    swaggerUi.setup(swaggerSpec));
25
26 app.listen(3000);
27
```




COMMENTAIRES DANS LE CODE

JSDOCS

- JSDoc est un standard/commentaire spécial permettant de documenter le code JavaScript.
- Les commentaires JSDoc sont placés juste au-dessus des fonctions, classes, variables, etc.
- Ils permettent de décrire les paramètres, types, valeurs de retour, exceptions, et autres infos utiles.
- Ces commentaires peuvent être utilisés pour générer automatiquement une documentation HTML lisible ou pour aider les IDE (Visual Studio Code, WebStorm) à proposer de l'autocomplétion et des vérifications de type.

```
1  /**
2   * @typedef {Object} Props
3   * @property {string} name Name of the user e.g. Lily
4   */
5
6  /**
7   * Displays a greeting to the user.
8   * @param {Props} props
9   * @example
10  * <Greeting name="Lily" />
11  * // Displays a greeting message to user called
12  * // "Lily".
13  */
14  function Greeting({ name }) {
15    return <div>Hello {name}!</div>;
16  }
17
18  export default Greeting;
19
```

SYNTAXE DE BASE

Un commentaire JSDoc commence par `/**` et se termine par `*/` :

On vient ensuite ajouter des tags

Tag	Description	Exemple
@param	Décrit un paramètre de fonction	@param {string} name
@returns / @return	Valeur retournée	@returns {boolean}
@typedef	Définit un type personnalisé	Voir plus bas
@property	Propriété d'un type personnalisé	
@throws	Exceptions pouvant être lancées	@throws {Error}
@deprecated	Indique que la fonction est obsolète	
@example	Donne un exemple d'utilisation	
@see	Référence une autre fonction ou doc	

EXEMPLE

Un commentaire JSDoc commence par `/**` et se termine par `*/` :

On vient ensuite ajouter des tags

```
/**
 * Représente une tâche TODO.
 * @typedef {Object} Todo
 * @property {number} id - Identifiant unique.
 * @property {string} title - Titre de la tâche.
 * @property {boolean} done - Statut de la tâche.
 */

/**
 * Ajoute une tâche à la liste.
 * @param {Todo} todo - La tâche à ajouter.
 * @returns {void}
 */
function ajouterTache(todo) {
  // ajout dans la liste...
}
```


ATELIER

Mettre en place une doc API pour notre
cineclub



CONSIGNES

Vous avez développé une API CineClub permettant de gérer une collection de films.

Vous allez maintenant en rédiger la documentation technique sous forme de fichier Swagger.

Documenter l'API CineClub avec Swagger

Objectif pédagogique

Apprendre à :

- Documenter une API REST avec Swagger (OpenAPI 3.x)
- Définir les routes, les paramètres, les corps de requête et les réponses
- Structurer un fichier `swagger.yaml` ou `swagger.json`
- Visualiser une documentation API à l'aide d'un outil comme Swagger Editor

Contexte

Vous avez développé une API CineClub permettant de gérer une collection de films. Vous allez maintenant rédiger une **documentation technique** complète au format Swagger.

À produire

Un fichier `swagger.yaml` (ou `swagger.json`) décrivant toute l'API CineClub.

Spécifications à documenter

1. `GET /`

Retourne la **liste complète des films**

2. `GET /:id`

Récupère un **film par son ID**

QUIZZ

