# Fiche 3 — Documenter et valoriser ses projets

## Introduction

Un bon projet mal documenté est un projet invisible.
La documentation n'est pas un supplément esthétique : c'est **la preuve de la maîtrise et de la rigueur du développeur**.

Dans le monde professionnel et open source, une documentation claire et vivante est souvent ce qui fait la différence entre un projet que l'on utilise — et un projet que l'on oublie.
Apprendre à rédiger un `README.md`, à structurer une présentation de projet et à communiquer ses choix techniques, c'est apprendre à **rendre son travail compréhensible, réutilisable et crédible**.

## 1. Comprendre la documentation comme outil professionnel

### 1.1 Pourquoi documenter ?

- Pour **vous-même** : garder une trace claire de votre logique et de vos choix.
- Pour **vos collaborateurs** : faciliter la reprise et la maintenance du code.
- Pour **vos employeurs ou clients** : prouver la qualité et la cohérence de votre travail.
- Pour **la communauté** : encourager la réutilisation, la contribution et la confiance.

### 1.2 Types de documentation

- **Documentation technique** : architecture, API, installation, configuration.
- **Documentation utilisateur** : mode d'emploi, captures d'écran, exemples d'usage.
- **Documentation contextuelle** : origine du projet, problème à résoudre, contraintes.
- **Documentation de maintenance** : changelog, todo list, roadmap.

### 1.3 La documentation comme argument

Un projet bien expliqué raconte une histoire claire :

> “Voici le problème que j’ai résolu, comment je l’ai abordé, et ce que j’ai appris.”
> Cette démarche valorise votre démarche d’ingénierie et votre progression.

## 2. Rédiger un bon `README.md`

### 2.1 Le rôle du README

Le fichier `README.md` est la **porte d’entrée** de votre projet :
c’est ce que toute personne voit en premier sur GitHub ou GitLab.
Il doit permettre de comprendre immédiatement :

- ce que fait le projet,
- comment le tester,
- et comment y contribuer.

### 2.2 Structure recommandée

Un bon `README.md` comporte idéalement :

1. **Titre et brève description**
   → une phrase claire : “Une application web de gestion de concerts développée avec Symfony 7 et React.”
2. **Badges informatifs (optionnel)**
   → version, licence, CI/CD, couverture de test.
3. **Sommaire rapide**
   → pour naviguer facilement dans le document.
4. **Présentation du projet**
   → contexte, objectifs, public visé, fonctionnalités principales.
5. **Technologies utilisées**
   → frameworks, bibliothèques, outils, hébergement.
6. **Installation / exécution**
   → étapes claires, prérequis, commandes à exécuter.

7. **Utilisation / démo**
   → captures d'écran, liens de démonstration, exemples d'usage.
8. **Structure du projet (facultatif)**
   → organigramme des dossiers, explication rapide de l'architecture.
9. **Auteurs et contributions**
   → noms, liens GitHub, remerciements.
10. **Licence**
    → indiquer clairement les droits d'usage.

### 2.3 Bonnes pratiques rédactionnelles

- Rédiger dans un **langage simple, clair et professionnel**.
- Utiliser la mise en forme Markdown : titres, listes, blocs de code.
- Vérifier l'orthographe et la cohérence terminologique.
- Illustrer vos propos : images, GIF, captures d'écran.
- Mettre à jour régulièrement le README (nouvelle version, bug connu, roadmap).

## 3. Valoriser ses projets : le storytelling technique

### 3.1 Mettre en scène vos choix techniques

Un bon projet documenté n'explique pas seulement *ce que vous avez fait*, mais aussi *pourquoi*.
Exemples :

- "J'ai choisi PostgreSQL plutôt que MySQL pour la gestion des relations complexes."
- "J'ai mis en place une API REST plutôt que GraphQL pour simplifier la maintenance."
  Ces justifications brèves montrent votre **réflexion d'ingénierie**.

### 3.2 Montrer les difficultés rencontrées

Plutôt que de masquer les erreurs ou limites :

- expliquez ce qui a posé problème,
- montrez comment vous avez corrigé,
- précisez ce que vous feriez différemment aujourd'hui.
  Cette transparence inspire la confiance et atteste de votre capacité d'apprentissage.

### 3.3 Donner de la visibilité à vos projets

- Reliez chaque projet à votre portfolio.
- Rédigez un court article technique ou un billet de blog (“comment j’ai structuré mon API Symfony”).
- Partagez sur LinkedIn, Dev.to ou Mastodon pour obtenir des retours constructifs.
- Maintenez vos projets publics à jour (même légèrement).

# 4. Activités pratiques

## Atelier 1 — Analyse comparative

- Choisissez un dépôt open source connu (ex : Symfony, Vue.js, Strapi).
- Étudiez son `README.md` : structure, ton, clarté, équilibre entre technique et communication.
- Notez ce que vous pourriez réutiliser pour vos propres projets.

## Atelier 2 — Rédaction de votre documentation

- Sélectionnez un de vos projets personnels.
- Créez ou refondez son `README.md` complet.
- Ajoutez un titre clair, une courte présentation, les instructions d’installation, les technologies, et au moins une capture d’écran.

## Atelier 3 — Mise en cohérence

- Vérifiez que la documentation de vos projets suit un modèle commun.
- Intégrez ces fiches dans votre portfolio.
- Mettez à jour les informations croisées (liens, dates, versions).

# Livrables attendus

- Un `README.md` complet et à jour pour au moins un projet.
- Trois fiches projets documentées (dans le portfolio ou en PDF).

- Un court texte d'analyse comparative (5 à 10 lignes) sur la documentation d'un projet open source étudié.

# Évaluation

| Critère | Description | Pondération |
|---|---|---|
| Qualité du README | Structure, clarté, exhaustivité, mise en forme | 40 % |
| Cohérence des fiches projets | Pertinence des informations et homogénéité de présentation | 40 % |
| Capacité d'analyse | Pertinence de la comparaison avec un projet open source | 20 % |

# Pour aller plus loin

- Consulter le guide officiel de **rédaction Markdown** : https://www.markdownguide.org/
- Explorer les templates de documentation open source :
  - *Best README Template* (https://github.com/othneildrew/Best-README-Template)
  - *Awesome README* (https://github.com/matiassingers/awesome-readme)
- Lire : *The Art of Readable Code* (Dustin Boswell, Trevor Foucher).
- Pratiquer l'amélioration continue : mise à jour mensuelle de vos documents techniques.

# Message clé

> Documenter, c'est prolonger son code par la pensée.
> Ce que vous écrivez autour de votre projet est aussi important que le projet lui-même.
> Une bonne documentation n'explique pas seulement **comment ça marche**, mais **pourquoi ça compte**.