📚 Les différentes documentations à fournir avec un logiciel

La documentation d'un logiciel est essentielle pour assurer sa **bonne utilisation, sa maintenance et son évolution**. Elle se décline en plusieurs catégories selon les publics cibles.

👤 1. Les documentations utilisateurs

Il s'agit de l'ensemble des documents destinés aux **utilisateurs finaux** du logiciel. L'objectif principal est de leur permettre de **prendre en main facilement l'application**.

👉 Elle doit être :

- claire
- intuitive
- accessible (sans jargon technique)
- agréable à consulter

⚠️ Il est important d'éviter les termes techniques ou de les expliquer simplement.

📄 Types de documents :

- **Guide de l'utilisateur / Manuel**
  - Explique comment utiliser le logiciel pour réaliser des tâches spécifiques
  - Peut inclure des exemples et tutoriels pas à pas

- **Guide de démarrage rapide / Tutoriel**
  - Permet une prise en main rapide
  - Présente les fonctionnalités essentielles

- **Aide contextuelle**
  - Intégrée directement dans l'application
  - Exemples : infobulles, icônes "?", messages d'aide

- **FAQ / Base de connaissances**
  - Réponses aux questions fréquentes
  - Résolution de problèmes courants

🛠️ 2. Les documentations techniques

Ces documents décrivent **le fonctionnement interne du logiciel**.
Ils sont destinés aux :

- développeurs
- responsables techniques
- intégrateurs
- chefs de projet

📄 Types de documents :

- **Documentation d'architecture / conception**
  - Structure globale du logiciel
  - Technologies utilisées
  - Schémas (base de données, flux, composants)
  - Design patterns et organisation du code

- **Documentation du code (API)**
  - Commentaires dans le code source
  - Documentation des API
  - Description des algorithmes
  - Conventions de codage (nommage, formatage)

- **Guide d'installation et de déploiement**
  - Étapes d'installation
  - Configuration du système
  - Procédure de déploiement
  - Processus de build (si spécifique)

- **Documentation de maintenance**
  - Procédures de correction de bugs
  - Journaux de modifications (changelog)
  - Rapports de tests
  - Guide de débogage

📊 3. Les documentations de projet

Ces documents permettent de **comprendre le contexte global du projet** et facilitent sa reprise par une autre équipe.

📄 Types de documents :

- **Cahier des charges / Expression des besoins**

    - Objectifs du projet
    - Périmètre fonctionnel
    - Contraintes
    - Attentes du client

- **Notes de version (Release Notes)**
    - Nouvelles fonctionnalités
    - Corrections de bugs
    - Améliorations

- **Plan de tests**
    - Stratégie de test
    - Scénarios
    - Résultats attendus

📢 4. Les documentations marketing

Elles sont destinées à **promouvoir le logiciel** et à mettre en avant sa valeur.

📄 Types de documents :

- **Brochures**
    - Présentation globale du produit
    - Mise en avant des avantages

- **Fiches produit**
    - Description des fonctionnalités clés
    - Cas d'usage
    - Bénéfices pour l'utilisateur

📚 La documentation technique dans la pratique

La documentation doit être intégrée directement dans le processus de développement afin de rester **à jour, utile et exploitable**. L'objectif est de documenter **sans alourdir le travail du développeur**.

⚙️ 1. Écrire la documentation au fur et à mesure

La documentation ne doit pas être réalisée à la fin du projet. Donc l'inclure directement dans le projet

```bash
/docs
  ├── README.md
  ├── installation.md
  ├── api.md
  ├── features/
  │     ├── budget.md
  │     ├── epargne.md
  │     └── notifications.md
  └── decisions.md
```

🔌 2. Ajouter la documentation API directement dans le code

Chaque endpoint ou fonction importante doit être documenté.

👉 Objectifs :

- Comprendre rapidement le rôle
- Identifier les entrées et sorties

```php
/**
 * Créer une nouvelle transaction
 *
 * @param amount int
 * @param type string (income | expense)
 * @return Transaction
 */
public function store(Request $request)
```

📬 3. Générer la documentation API avec Postman

Postman permet de générer automatiquement une documentation à partir des requêtes. Générer et daté pour chaque feature et l'inclure dans le code.

🔗 4. Ajouter des commentaires référencés

Faire des liens entre le code et la documentation détaillée.

```
// Voir docs/features/notifications.md pour la logique complète
```

🤖 5. Utiliser le CI/CD pour contrôler la documentation

Automatiser la vérification de la documentation.

👉 Exemples :

- Vérifier l'existence de fichiers essentiels (`README.md`, `/docs/api.md`)
- Bloquer un merge si la documentation est absente
- Vérifier la cohérence des endpoints

🔄 6. Versionner la documentation avec Git

La documentation doit être versionnée comme le code.

👉 Bonnes pratiques :

- Inclure la documentation dans chaque commit
- Lier les modifications de code à la documentation

📌 7. Utiliser le README comme point d'entrée

Le fichier `README.md` est la porte d'entrée du projet.

👉 Il doit permettre de :

- Comprendre rapidement le projet
- Trouver les informations essentielles
- Démarrer facilement

```Markdown
# App de gestion financière

## 🚀 Démarrage rapide
Voir docs/installation.md

## 📚 Documentation
- API → docs/api.md
- Features → docs/features/
- Base de données → docs/database.sql
- Décisions → docs/decisions.md

## 🧪 Tests
...

## 📦 Stack technique
- React Native (Expo)
- Laravel API
- PostgreSQL
```

🔄 8. Utiliser les commits comme documentation vivante

Les commits racontent l'histoire du projet.

👉 Bonnes pratiques :

- Messages clairs et structurés
- Expliquer le *pourquoi*

🗄️ 9. Documenter la base de données (PostgreSQL)

La base de données doit être versionnée et documentée.

👉 Bonne pratique :
Exporter la structure après chaque modification. Ajouter une version dans base.md daté

```Bash
pg_dump -s -U user db_name > docs/database.sql
```

🧠 10. Documenter les décisions majeures

Les décisions techniques doivent être conservées.

👉 Fichier : `docs/decisions.md`

```Markdown
## Choix de PostgreSQL
- meilleure gestion des relations
- performance sur les requêtes complexes

## Choix de React Native (Expo)
- développement rapide
- compatibilité Android/iOS

## Système de notifications
- Firebase pour le temps réel
```

🔗 11. Relier tous les éléments

👉 Une documentation efficace repose sur un système connecté :

- README → oriente
- Docs → expliquent
- Code → référence
- Commits → racontent
- Base de données → structure

⚡ 12. Workflow simple et efficace

À chaque évolution :

1. Modifier le code
2. Mettre à jour :
    - la documentation concernée
    - la base de données (si besoin)
3. Faire un commit clair

👤 La documentation technique dans la pratique

⏰ Bloquer du temps dédié à chaque livraison de feature

La documentation utilisateur ne doit pas être laissée “si on a le temps”.

👉 Bonne pratique :

- **Bloquer une plage horaire** à chaque livraison de feature
- Considérer la documentation comme une **tâche obligatoire**

➡️ Une feature n’est **pas terminée** tant que sa documentation n’est pas faite.

🧩 Intégrer la doc dans le workflow

🔁 Nouveau flow :

1. Développement de la feature
2. Tests
3. Livraison
4. 🔴 Documentation utilisateur