docShadow CLI — Spécification et Design

1. Objectif

- **Nom du projet** : `docShadow` (alias **GitShadow**)
- **But** : Générer automatiquement, à chaque commit Git, une documentation JSON pour alimenter ensuite une interface UI.
- **Philosophie** : « Silent companion » de Git : un *shadow face* qui suit chaque commit sans intervention manuelle.

2. Stack Technique

- **Langage** : Python (v3.10+)
- **CLI Framework** : Click ou [Argparse amélioré]
- **Git Integration** :
- Git hooks (`.git/hooks/post-commit`)
- Bibliothèque `GitPython` pour manipulations avancées
- **Format Config** : JSON/YAML (`docshadow.config.json`)
- **Ignore File** : `.docignore` (calqué sur `.gitignore`)
- **JSON output** : Schéma standardisé pour UI, structuré par fichier

3. Structure du Projet

```
├── .git/
├── .docignore
├── docshadow.config.json
├── .docshadow/              # dossier généré après init
│   ├── index.json           # index général de la doc pour un commit
│   ├── docshadow.json       # mapping global du projet (structure miroir +
métadonnées)
│   ├── module/              # calque l'arborescence du projet
│   │   └── foo.py.json      # doc du fichier foo.py
│   └── ...
├── docshadow/               # code source CLI
│   ├── __main__.py          # point d'entrée CLI
│   ├── hooks.py             # installation & exécution Git hooks
│   ├── generator.py         # extraction & création JSON
│   ├── utils.py             # assistants (parsing, logging, etc.)
└── setup.py                 # packaging & entry point `doc`
```

4. Commandes CLI Principales

```
# Initialisation dans un repo existant
doc init             # crée docshadow.config.json + .docignore + .docshadow/ +
```

```
installe post-commit hook

# Génération manuelle (optionnel)
doc generate [--commit <hash>]  # regen docs pour le commit courant ou spécifié

# Affichage de l'état
doc status           # liste commits/doc.json manquants ou conflit .docignore
```

5. Configuration (`docshadow.config.json`)

```
{
  "project_name": "MyAwesomeRepo",
  "languages": ["python"],
  "output_dir": ".docshadow/",
  "hooks": {
    "post_commit": true
  },
  "ignore": ".docignore"
}
```

- **project_name** : nom du projet
- **languages** : liste de langages supportés ("python" pour v1)
- **output_dir** : répertoire où stocker les fichiers `.json` par fichier source
- **hooks.post_commit** : activer/désactiver le hook
- **ignore** : chemin vers `.docignore`

6. `.docignore`

- Copie initiale de `.gitignore`
- Permet d'exclure fichiers/source non désirés au parsing

7. Git Hook Integration

1. **Installation** via `doc init` : copie du script `hooks.post_commit.py` dans `.git/hooks/post-commit`
2. **Exécution** : à chaque commit, le hook appelle `doc generate --commit $GIT_COMMIT`
3. **Sécurité** : vérification que le repo est bien initialisé (présence de `docshadow.config.json`)

8. Générateur de Documentation

- **Entrée** : commit hash + snapshot des sources (via GitPython)

- **Étapes** :

- Récupérer la liste de fichiers modifiés

- Filtrer selon `.docignore`
- Parser les fichiers Python (AST ou regex docstring)
- Extraire classes, fonctions, docstrings, signatures
- Structurer la documentation dans un fichier `.json` par fichier source
- Écrire dans `.docshadow/` en suivant l'arborescence du projet, avec un `.json` par fichier source
- Mettre à jour `docshadow.json` (mapping complet du projet → fichiers + chemins + métadonnées)
- **Sortie** :
- `.docshadow/module/foo.py.json` ← doc du fichier `module/foo.py`
- `.docshadow/index.json` ← résumé général du commit
- `.docshadow/docshadow.json` ← mapping global du projet, structure miroir exploitable côté UI
- (plus tard) `.docshadow-<commit_hash>/` ← archive envoyée vers le serveur distant après `git push`

Exemple de schéma `foo.py.json`

```
{
  "path": "module/foo.py",
  "classes": [
    {
      "name": "Foo",
      "doc": "Classe Foo...",
      "methods": [ ... ]
    }
  ],
  "functions": [ ... ]
}
```

Exemple de schéma `index.json`

```
{
  "commit": "a1b2c3d4",
  "date": "2025-07-25T14:23:11Z",
  "files": ["module/foo.py.json", "module/bar.py.json"]
}
```

Exemple de `docshadow.json`

```
{
  "project": "MyAwesomeRepo",
  "generated_at": "2025-07-25T14:30:00Z",
```

```
  "structure": {
    "module": {
      "foo.py": "module/foo.py.json",
      "bar.py": "module/bar.py.json"
    },
    "core": {
      "engine.py": "core/engine.py.json"
    }
  }
}
```

9. Roadmap & Prochaines Étapes

1. **Prototype CLI** (`init` + hook placement)
2. **Module generator.py** minimal pour un fichier Python
3. **Structure de sortie `` calquée sur le projet**
4. **Envoi d'une copie de `` renommée avec le commit hash au serveur distant après push**
5. **Mise à jour automatique de `` après chaque commit**
6. **Tests unitaires** sur extraction AST
7. **Gestion multilanguage** (v2: JS, Java)
8. **UI & SaaS** : consommation du JSON via une API + Dashboard

À toi de jouer : feedback, modules à coder, ou test en live sur repo ? 🚀