



Manu-cj

# L'organisation et l'architecture de dossiers





# Pourquoi une bonne architecture de dossiers est essentielle ?

## 1. Lisibilité et compréhension du projet :

Un projet avec des dossiers bien nommés (e.g., components, services, utils) est immédiatement compréhensible.

## 2. Facilité de maintenance et de refactoring :

Facilité de maintenance et de refactoring, modifier une fonctionnalité dans un projet bien structuré est plus facile

## 3. Collaboration efficace :

Un développeur entrant dans un projet peut se concentrer sur une fonctionnalité spécifique sans avoir à rechercher dans tout le code.

## 4. Évolutivité et modularité :

Ajouter un module dans une application sans avoir à réorganiser tout le projet.



# Architecture basée sur les types de fichiers



A screenshot of a Windows File Explorer window showing a folder structure. The folder is named "FOLDER-SKELETON [WSL : UBUN...]" and contains a "src" directory. Inside "src", there are several sub-directories: "assets", "components", "config", "hooks", "pages", "services", "utils", and "index.js". The "index.js" file is selected in the center pane, displaying the following code:

```
// /project-root
// └── src/
//     ├── components/          # Composants réutilisables
//     ├── pages/                # Pages principales de l'application
//     ├── services/             # Gestion des appels API et logique métier
//     ├── utils/                # Fonctions utilitaires réutilisables
//     ├── hooks/                # Hooks personnalisés (si React)
//     ├── assets/               # Images, icônes et styles globaux
//     ├── config/               # Fichiers de configuration
//     └── index.ts              # Point d'entrée de l'application
```

The right pane of the File Explorer shows a preview of the "index.js" file content.

**Idéale pour des petits à moyens projets.**



# Architecture feature-based

The screenshot shows a VS Code interface with the following details:

- Title Bar:** folder-skeleton [WSL : Ubuntu]
- Explorer:** FOLDER-SKELETON [WSL : UBUN...]. The tree view shows a project structure: src > features > dashboard > index.js. Inside features, there are auth and dashboard folders, each containing components, services, and index.js files. A shared folder is also present.
- Editor:** The file index.js (under dashboard) is open. The code is as follows:

```
1 // /project-root
2 // └── src/
3 //   ├── features/
4 //   |   ├── auth/
5 //   |   |   ├── components/
6 //   |   |   ├── services/
7 //   |   |   └── hooks/
8 //   |   └── dashboard/
9 //       ├── components/
10 //       ├── services/
11 //       └── index.ts
12 //   └── shared/
13 //       └── index.ts
14 // 
```

Annotations in the code explain the structure:
  - # Composants Liés à L'authentification
  - # Logique métier spécifique à L'authentification
  - # Hooks spécifiques à L'authentification
  - # Point d'entrée pour La fonctionnalité auth
  - # Code commun à plusieurs fonctionnalités
- Bottom Status Bar:** WSL : Ubuntu, 0 0 0 0 17 mins, 141hrs 48mins, Espaces : 4, UTF-8, LF, {}, Babel JavaScript, Go Live, Go Live, Go Live, Prettier, ⌂

**Text Overlay:** Idéale pour des projets à grande échelle avec plusieurs fonctionnalités indépendantes.



# Architecture MVC

The screenshot shows a VS Code interface with the following details:

- Title Bar:** folder-skeleton [WSL : Ubuntu]
- Explorer:** Shows the project structure:
  - FOLDER-SKELETON [WSL : UBUN...]
  - src
    - controllers
    - models
    - routes
    - services
    - views
    - app.js
- Editor:** The file `src/app.js` is open, displaying the following code structure:

```
1 // /project-root
2 // └── src/
3 //     ├── models/          # Modèles de données (ex: ORM, structures)
4 //     ├── views/           # Vues ou templates
5 //     ├── controllers/    # Logique qui contrôle le flux de l'application
6 //     ├── routes/          # Gestion des routes HTTP
7 //     ├── services/        # Services logiques ou API
8 //     └── app.ts           # Point d'entrée de l'application
```
- Bottom Status Bar:** WSL : Ubuntu, 0 0 0 0 18 mins, 141hrs 54mins, L 9. col 1, Espaces : 4, UTF-8, LF, {}, Babel JavaScript, Go Live, Go Live, Go Live, Prettier

**Text Overlay:** Structure classique pour des applications serveur ou des applications classiques.



# Bonne pratiques de nomage

camelCase	fichiers utilitaires : Exemple : <b>fetchData.js</b>	
PascalCase	composants et les classes : Exemple : <b>UserCard.tsx</b>	
kebab-case	noms de dossiers : Exemple : <b>user-profile/</b>	



# Git : Règles à suivre

- Ne jamais travailler dans la branche main/master
- Créer une branche Dev et une autre avec son nom
- On travaille dans sa branche
- Toujours pull avant de merge
- Prévenir le pm en cas de conflit de merge
- On fait des commits clair et régulier





# Git : Exemples de commits

## 1. Adding a New Feature

- **feat:** [short description]
  - feat: add category filtering on the products page
  - feat(auth): implement authentication with JWT

## 2. Fixing a Bug

- **fix:** [short description]
  - fix: fix form submission issue on the contact page
  - fix(api): resolve timeout issue on user request

## 3. Refactoring Code

- **refactor:** [short description]
  - refactor: simplify form validation logic
  - refactor(api): extract utility functions into a separate module

## 4. Improving Performance

- **perf:** [short description]
  - perf: optimize SQL queries to speed up data loading
  - perf(front): reduce image size to improve page load time

## 5. Adding Tests

- **test:** [short description]
  - test: add unit tests for user management
  - test(api): cover /login endpoint with integration tests

## 6. Updating Documentation

- **docs:** [short description]
  - docs: update README with installation instructions
  - docs(api): add OpenAPI specifications for new routes

## 7. Configuration or Build Changes

- **chore:** [short description]
  - chore: update npm dependencies
  - chore(ci): add job for automatic deployment

## 8. Removing Unused Code

- **chore:** [short description] or **refactor:** [short description]
  - chore: remove unused files from the project
  - refactor: remove obsolete token management

## 9. Reverting a Previous Commit

- **revert:** [short description]
  - revert: rollback axios update due to compatibility issues

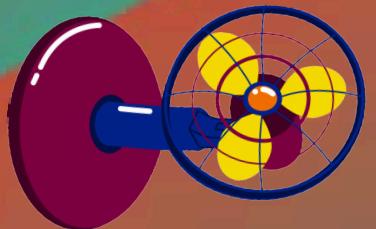
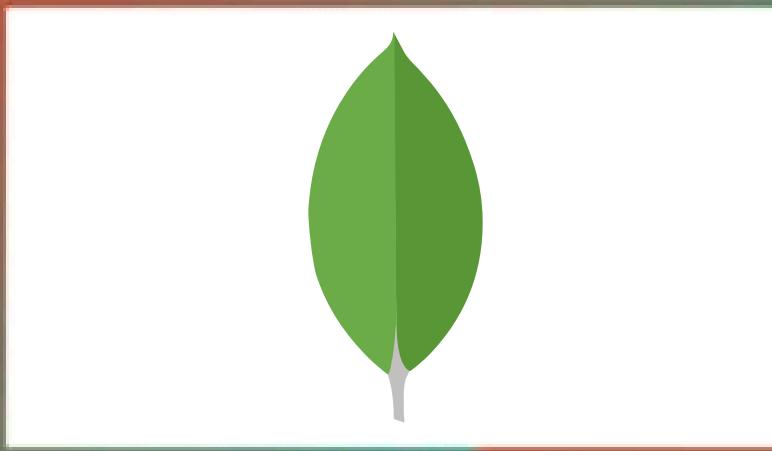


# Pour terminer

- Ne pas mélanger la logique métier et l'interface utilisateur.
- Diviser les fichiers volumineux en plus petites parties
- Documenter la structure des dossiers dans un fichier README.md
- Travailler dans la bonne branche
- Ne pas mettre en commit “Pause de 15h”



# THANKS FOR WATCHING



SUBSCRIBE [My Github](#)