

Module 4 — Pratique Guidee

Jour 1 — Projet existant + projet vierge

The Bearded Bear

Février 2026



Table des matières

Module 4 : Pratique Guidee	3
Objectifs	3
Partie 1 : Projet existant (1h)	3
1. Onboarding sur une codebase inconnue	3
1.1 Le defi	3
1.2 Methodologie en 4 etapes	3
1.3 Exercice pas-a-pas : Explorer une codebase	4
1.4 Documentation automatique	5
2. Refactoring guide	5
2.1 Methodologie	5
2.2 Exercice pas-a-pas : Refactoring	5
3. Debugging assiste	6
3.1 Methodologie	6
3.2 Exercice pas-a-pas : Debugging	7
3.3 Techniques avancees	8
4. Code review assistee	8
4.1 Methodologie	8
4.2 Exercice pas-a-pas : Code review	8
4.3 Review de PR complete	9
Partie 2 : Projet vierge (1h)	9
5. Scaffolding complet	9
5.1 Methodologie	9
5.2 Exercice pas-a-pas : Scaffolding d'un projet	10
6. Generation de code	11
6.1 Methodologie TDD	11
6.2 Exercice pas-a-pas : Generer une feature complete	11
6.3 Generer les endpoints API	13
7. Mise en place CI/CD basique	13
7.1 Exercice : Creer une pipeline CI	13
7.2 Verifier la pipeline	13
7.3 Ajouter des etapes avancees	13
8. Documentation automatique	14
8.1 Generer la documentation projet	14
8.2 Generer la documentation API	14
8.3 Generer les commentaires de code	14
9. Synthese de l'exercice complet	15
9.1 Ce que vous avez appris	15
9.2 Checklist de validation finale	15
Points Cles a Retenir	15

Ressources complémentaires	16
--------------------------------------	----

Module 4 : Pratique Guidee

Duree : 2h **Prerequis :** Module 3 - Patterns de Travail **Prochain module :** Jour 2 - Module 5

Objectifs

A l'issue de ce module, vous serez capable de :

1. Utiliser Claude Code pour explorer et comprendre une codebase inconnue
2. Conduire un refactoring guide par Claude Code
3. Debugger efficacement avec l'aide de Claude Code
4. Realiser une revue de code assistee
5. Scaffolder un projet complet depuis zero
6. Generer du code, des tests et de la documentation avec Claude Code

Partie 1 : Projet existant (1h)

Cette premiere partie vous apprend a travailler avec un projet qui existe deja. Vous allez cloner un projet open source et utiliser Claude Code pour l'explorer, le refactorer, le debugger et le reviewer.

1. Onboarding sur une codebase inconnue

1.1 Le defi

Vous arrivez sur un projet existant que vous ne connaissez pas. Comment comprendre rapidement son architecture, ses conventions et ses points sensibles?

1.2 Methodologie en 4 etapes

Etape 1 : Vue d'ensemble "Qu'est-ce que ce projet ?"
Etape 2 : Architecture "Comment est-il structure ?"

Etape 3 : Flux de donnees "Comment les donnees circulent ?"	
Etape 4 : Points sensibles "Ou sont les risques ?"	

1.3 Exercice pas-a-pas : Explorer une codebase

Preparation : Choisissez un projet open source dans le langage de votre choix, ou utilisez un de vos projets professionnels.

Etape 1 : Vue d'ensemble (5 min)

```
# Lancez Claude Code a la racine du projet
cd /chemin/vers/le/projet
claude
```

```
# Premiere question : vue globale
```

- > Analyse ce projet et donne-moi un resume en 10 lignes maximum :
- > - Quel est son objectif principal ?
- > - Quelle stack technique est utilisee ?
- > - Quelles sont les principales dependances ?
- > - Quel est l'etat general du projet (date dernier commit, tests, CI) ?

Etape 2 : Architecture (10 min)

```
# Comprendre la structure
```

- > Analyse l'architecture de ce projet :
- > - Quelles sont les couches / modules principaux ?
- > - Quel pattern architectural est utilise (MVC, Clean Architecture, etc.) ?
- > - Dessine un schema de l'architecture en ASCII
- > - Identifie les points d'entree (controllers, routes, commands)

```
# Approfondir si necessaire
```

- > Montre-moi le flux d'une requete HTTP typique
- > depuis le point d'entree jusqu'a la base de donnees

Etape 3 : Flux de donnees (10 min)

```
# Comprendre les modeles de donnees
```

- > Quels sont les modeles de donnees principaux ?
- > Montre les relations entre eux (schema ER simplifie en ASCII)

```
# Comprendre les API
```

- > Liste les endpoints API disponibles avec leurs methodes HTTP
- > et une description courte de chacun

Etape 4 : Points sensibles (5 min)

Identifier les risques

- > Identifie les 5 points les plus problematiques de ce projet :
- > - Code complexe (methodes longues, trop de responsabilites)
- > - Failles de securite potentielles
- > - Tests manquants
- > - Dette technique visible
- > Pour chaque point, indique le fichier et la ligne concernes

1.4 Documentation automatique

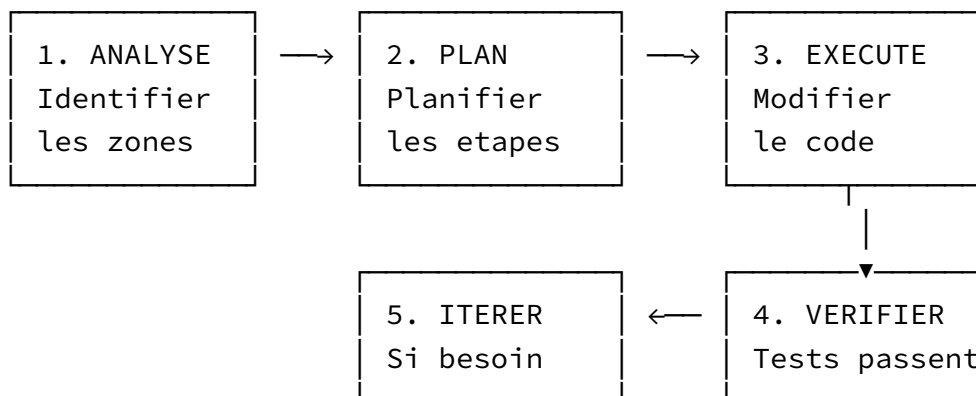
Generer une documentation d'onboarding

- > Genere un document d'onboarding pour un nouveau developpeur
- > qui rejoint ce projet. Inclus :
- > - Les prerequis d'installation
- > - Les commandes pour lancer le projet
- > - L'architecture en bref
- > - Les conventions a connaitre
- > - Les fichiers les plus importants a lire en premier

2. Refactoring guide

2.1 Methodologie

Le refactoring avec Claude Code suit un cycle precis :



2.2 Exercice pas-a-pas : Refactoring

Etape 1 : Identifier les zones a ameliorer (5 min)

Demander une analyse qualite

- > Analyse la qualite du code de ce projet et identifie :
- > - Les fichiers avec une complexite cyclomatique elevee

- > - Les methodes de plus de 20 lignes
- > - Les duplications de code
- > - Les violations des principes SOLID
- > Classe les resultats par priorite (critique, haute, moyenne, basse)

Etape 2 : Planifier le refactoring (5 min)

Activer le Plan Mode

- > Shift+Tab

Demander un plan de refactoring

- > Propose un plan de refactoring pour [le fichier identifie]
- > en respectant ces contraintes :
 - > - Ne pas casser les tests existants
 - > - Modifier un aspect a la fois
 - > - Chaque etape doit etre un commit atomique

Etape 3 : Executer le refactoring (15 min)

Valider le plan et lancer l'execution

- > Le plan est bon. Commence par l'etape 1.
- > Apres chaque modification, verifie que les tests passent.

Suivre l'execution

Claude va :

1. Modifier le code

2. Executer les tests

3. Corriger si necessaire

4. Passer a l'etape suivante

Etape 4 : Verifier (5 min)

Verifier le resultat global

- > Montre-moi un diff de tous les changements effectues
- > et confirme que :
 - > - Tous les tests passent
 - > - La couverture de tests n'a pas baisse
 - > - Le comportement fonctionnel est identique

3. Debugging assiste

3.1 Methodologie

Le debugging avec Claude Code est plus efficace quand vous fournissez un maximum de contexte :

CONTEXTE A FOURNIR

- | |
|--|
| <ol style="list-style-type: none">1. Description du comportement attendu2. Description du comportement observe3. Stack trace / logs d'erreur4. Etapes de reproduction5. Ce que vous avez deja essaye |
|--|

3.2 Exercice pas-a-pas : Debugging

Scenario : Vous avez un bug dans votre application. Un endpoint renvoie une erreur 500 dans certaines conditions.

Etape 1 : Fournir le contexte (3 min)

```
> J'ai un bug : l'endpoint [endpoint] renvoie une erreur 500
> quand [condition specifique].
>
> Voici le stack trace :
> [collez le stack trace]
>
> Le comportement attendu est : [description]
> Le comportement observe est : [description]
>
> J'ai deja verifie que :
> - La base de donnees est accessible
> - Les donnees de test sont correctes
```

Etape 2 : Laisser Claude investiguer (5 min)

```
# Claude va autonomement :
# 1. Lire les fichiers mentionnes dans le stack trace
# 2. Suivre la chaine d'appels
# 3. Identifier les variables et conditions
# 4. Proposer une hypothese
```

```
# Si Claude demande des precisions :
> Voici le contenu de la variable en question :
> [collez les donnees]
```

Etape 3 : Valider le diagnostic (2 min)

```
# Claude propose un diagnostic
# Validez ou affinez :
> Ton diagnostic semble correct. Propose un fix.
# ou
> Je ne pense pas que ce soit ca parce que [raison].
> Cherche dans une autre direction.
```

Etape 4 : Appliquer le fix (5 min)

Demander le fix avec tests

- > Applique le fix et ajoute un test de regression
- > qui reproduit ce bug specifique pour eviter
- > qu'il ne reapparaisse a l'avenir.

3.3 Techniques avancees

Debugging avec logs

- > Ajoute des logs temporaires dans [fichier] pour tracer
- > le flux d'execution. Je lancerai la requete et te montrerai les logs.

Debugging avec breakpoints (description)

- > Explique-moi a quel endroit du code je devrais mettre
- > un breakpoint pour observer le probleme

Analyse de logs existants

- > Voici les 50 dernieres lignes de logs :
 - > [collez les logs]
 - > Identifie les patterns anormaux
-

4. Code review assistee

4.1 Methodologie

Claude Code peut realiser une revue de code structured et approfondie.

4.2 Exercice pas-a-pas : Code review

Etape 1 : Definir le scope (2 min)

Review d'un diff

- > Fais une code review du diff suivant :
- > [collez le diff ou indiquez la branche]

Ou review d'un fichier

- > Fais une code review approfondie de @src/services/payment.ts

Etape 2 : Review structuree (10 min)

- > Fais une code review de @src/services/payment.ts en verifiant :
- >
- > 1. **Architecture** : Le code respecte-t-il les couches ?
- > 2. **SOLID** : Les principes sont-ils respectes ?
- > 3. **Tests** : Y a-t-il des tests suffisants ?
- > 4. **Securite** : Y a-t-il des failles ?

- > 5. **Performance** : Y a-t-il des optimisations possibles ?
- > 6. **Lisibilite** : Le code est-il comprehensible ?
- >
- > Pour chaque point, donne :
 - > - Un score de 1 a 5
 - > - Les problemes trouves avec ligne et suggestion
 - > - Un resume global

Etape 3 : Corriger les problemes (10 min)

Appliquer les corrections

- > Applique les corrections de priorite haute de la review.
- > Verifie que les tests passent apres chaque correction.

4.3 Review de PR complete

Si vous avez un diff git

- > Analyse le diff entre main et cette branche.
- > Fais une review complete :
 - > - Resume des changements
 - > - Problemes detectes (classes par severite)
 - > - Suggestions d'amelioration
 - > - Checklist de validation (tests, docs, securite)

Partie 2 : Projet vierge (1h)

Cette seconde partie vous apprend a demarrer un nouveau projet de zero avec Claude Code. Vous allez scaffold, generer du code, mettre en place les tests et la documentation.

5. Scaffolding complet

5.1 Methodologie

Le scaffolding avec Claude Code suit une approche top-down :

- | |
|---|
| 1. Definition du projet
→ Objectif, stack, contraintes |
| 2. Architecture
→ Couches, patterns, modules |

3. Structure des dossiers → Arborescence complete
4. Configuration → Package manager, linter, tests
5. Code de base → Boilerplate, premiers fichiers

5.2 Exercice pas-a-pas : Scaffolding d'un projet

Scenario : Vous devez creer une API REST pour la gestion de taches (todo list) avec votre stack de predilection.

Etape 1 : Definition du projet (5 min)

Creer le dossier et lancer Claude

```
mkdir todo-api && cd todo-api
git init
claude
```

Definir le projet

```
> Je veux creer une API REST pour une application de gestion de taches.
>
> Specifications :
> - CRUD complet pour les taches (titre, description, statut, priorite)
> - Authentification JWT
> - Pagination des resultats
> - Filtrage par statut et priorite
> - Validation des entrees
>
> Stack technique :
> - [Votre langage/framework de choix]
> - Base de donnees relationnelle
> - Tests unitaires et d'integration
>
> Architecture : Clean Architecture
>
> Commence par me proposer l'architecture et la structure
> des dossiers AVANT de generer le code.
```

Etape 2 : Valider l'architecture (5 min)

Claude propose une architecture

Validez ou ajustez :

```
> L'architecture est bonne. Quelques ajustements :
> - Ajoute un dossier pour les migrations de base de donnees
```

- > - Separe bien les DTOs des entites du domaine
- > - Prevois un dossier pour la documentation API

Ou acceptez tel quel :

- > L'architecture me convient. Cree la structure des dossiers
- > et les fichiers de configuration de base.

Etape 3 : Generation de la structure (10 min)

```
# Claude cree l'arborescence et les fichiers de config
# Il va generer :
# - Le fichier de configuration du package manager
# - La configuration du linter
# - La configuration des tests
# - Le fichier de configuration du framework
# - Le .gitignore
# - Le .claudeignore
# - Le CLAUDE.md du projet
```

```
# Verifiez que tout est correct
```

- > Montre-moi l'arborescence des fichiers crees

Etape 4 : Configuration (5 min)

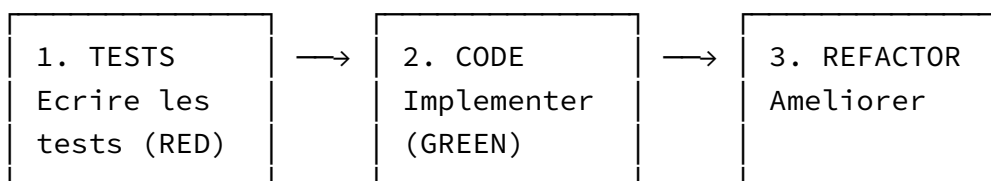
```
# Installer les dependances et verifier
```

- > Installe les dependances et verifie que le projet compile/demarre
- > sans erreur. Corrige tout probleme rencontre.

6. Generation de code

6.1 Methodologie TDD

La generation de code suit le cycle TDD : tests d'abord, puis implementation.



6.2 Exercice pas-a-pas : Generer une feature complete

Etape 1 : Generer les tests (10 min)

```
> Genere les tests unitaires pour le CRUD de l'entite Task.
> Les tests doivent couvrir :
>
> Creation :
> - Creer une tache avec des donnees valides
> - Echouer si le titre est vide
> - Echouer si le titre depasse 255 caracteres
> - Definir le statut "pending" par default
>
> Lecture :
> - Recuperer une tache par ID
> - Retourner une erreur 404 si l'ID n'existe pas
> - Lister les taches avec pagination
> - Filtrer par statut
>
> Mise a jour :
> - Modifier le titre d'une tache
> - Modifier le statut d'une tache
> - Echouer si la tache n'existe pas
>
> Suppression :
> - Supprimer une tache
> - Echouer si la tache n'existe pas
>
> Ecris les tests AVANT l'implementation.
> Ils doivent echouer (RED).
```

Etape 2 : Implementer (15 min)

```
# Verifier que les tests echouent
> Execute les tests et confirme qu'ils echouent tous
> (c'est la phase RED du TDD)

# Implementer pour faire passer les tests
> Maintenant, implemente le code necessaire pour faire passer
> TOUS les tests. Procede couche par couche :
> 1. Entite du domaine (Task)
> 2. Repository (interface + implementation)
> 3. Service / Use Case
> 4. Controller / Endpoint
>
> Apres chaque couche, execute les tests pour voir la progression.
```

Etape 3 : Verifier et refactorer (5 min)

```
# Tous les tests doivent passer
> Confirme que tous les tests passent.
> Si certains echouent encore, corrige le code.

# Refactoring
> Maintenant que les tests passent (GREEN), propose des
```

> ameliorations de code sans casser les tests (REFACTOR).

6.3 Generer les endpoints API

> Genere les endpoints REST pour le CRUD Task :

>

> POST	/api/tasks	→ Creer une tache
> GET	/api/tasks	→ Lister les taches (pagine)
> GET	/api/tasks/:id	→ Recuperer une tache
> PUT	/api/tasks/:id	→ Modifier une tache
> DELETE	/api/tasks/:id	→ Supprimer une tache

>

> Chaque endpoint doit :

- > - Valider les entrees
- > - Retourner les codes HTTP corrects
- > - Retourner des erreurs structurees en JSON
- > - Etre documente (commentaires ou annotations)

7. Mise en place CI/CD basique

7.1 Exercice : Creer une pipeline CI

> Cree une pipeline CI/CD basique avec GitHub Actions qui :

>

- > 1. Se declenche sur push et pull request vers main
- > 2. Execute les etapes suivantes :
 - > - Installation des dependances
 - > - Verification du linting
 - > - Execution des tests unitaires
 - > - Verification de la couverture (seuil 80%)
 - > - Build de l'application

>

> Genere le fichier `.github/workflows/ci.yml`

7.2 Verifier la pipeline

> Verifie que le fichier de CI est syntaxiquement correct.

> Simule mentalement l'execution et identifie les problemes potentiels (versions, cache, secrets necessaires).

7.3 Ajouter des etapes avancees

Une fois la CI de base validee

> Ajoute les etapes suivantes a la pipeline :

- > - Cache des dependances pour acclereler les builds
 - > - Matrice de versions pour tester sur plusieurs versions du runtime
 - > - Upload de la couverture de code comme artefact
 - > - Notification en cas d'**echec** (**optionnel**)
-

8. Documentation automatique

8.1 Generer la documentation projet

- > Genere un README.md complet pour ce projet avec :
- >
- > - Description du projet (**1-2 phrases**)
- > - Prerequis d'**installation**
- > - **Instructions** d'installation pas-a-pas
- > - Comment lancer le projet (**developpement**)
- > - Comment lancer les tests
- > - Structure du projet (**arborescence commentee**)
- > - Documentation API (**liste des endpoints**)
- > - Variables d'**environnement** **necessaires**
- > - **Guide de contribution**

8.2 Generer la documentation API

- > Genere une documentation API au format OpenAPI (**Swagger**)
- > pour tous les endpoints du projet. Inclus :
- > - Description de chaque endpoint
- > - Parametres (**path**, **query**, **body**)
- > - Schemas de requete et reponse
- > - Codes d'**erreur** **possibles**
- > - **Exemples de requetes**

8.3 Generer les commentaires de code

- > Ajoute des commentaires de documentation aux fonctions/methodes
 - > publiques du projet qui n'en ont pas encore. Les commentaires
 - > doivent expliquer le **POURQUOI** (pas le **QUOI**) et inclure :
 - > - Description courte
 - > - Parametres
 - > - Retour
 - > - Erreurs possibles
 - > - Exemple d'utilisation (**si non trivial**)
-

9. Synthese de l'exercice complet

9.1 Ce que vous avez appris

A la fin de cet exercice, vous avez pratique :

Competence	Projet existant	Projet vierge
Exploration codebase	Onboarding autonome	-
Architecture	Analyse existante	Conception nouvelle
Refactoring	Guide par Claude	-
Debugging	Assiste par Claude	-
Code review	Review structuree	-
Scaffolding	-	Structure complete
Generation code	-	TDD + implementation
Tests	Verification existants	Generation nouveaux
CI/CD	-	Pipeline complete
Documentation	Auto-generation	Generation initiale

9.2 Checklist de validation finale

Verifiez que vous etes a l'aise avec chaque point :

Projet existant : - ☐ Je sais explorer une codebase inconnue avec Claude Code - ☐ Je sais utiliser le Plan Mode pour planifier un refactoring - ☐ Je sais fournir le bon contexte pour un debugging efficace - ☐ Je sais demander une code review structuree

Projet vierge : - ☐ Je sais scaffold un projet complet avec Claude Code - ☐ Je sais generer du code en suivant le cycle TDD - ☐ Je sais mettre en place une pipeline CI/CD basique - ☐ Je sais generer de la documentation automatiquement

Points Cles a Retenir

1. **Exploration codebase** : Procédez du general au specifique (vue d'ensemble → architecture → details)
2. **Refactoring** : Toujours planifier avant d'agir, toujours verifier apres
3. **Debugging** : Plus vous fournissez de contexte (stack trace, logs, repro steps), plus Claude est efficace
4. **Code review** : Demandez une review structuree avec des criteres explicites

5. **Scaffolding** : Validez l'architecture AVANT de generer le code
 6. **TDD** : Tests d'abord (RED), implementation ensuite (GREEN), refactoring apres (REFACTOR)
 7. **CI/CD** : Commencez simple (lint + tests) et ajoutez progressivement
 8. **Documentation** : Claude Code peut generer README, API docs et commentaires de code
-

Ressources complementaires

Ressource	Description
Anthropic Claude Code docs	Documentation officielle
Claude Code GitHub	Code source et issues
Prompt Engineering Guide	Guide Anthropic

Prochain module : Jour 2 - Module 5 (Git avance, Hooks et automatisations)