

## Module 9 — Atelier Pratique

Jour 2 — Challenge d'équipe et mise en situation

The Bearded Bear

Février 2026



## Table des matières

<b>Module 9 : Mise en Pratique Collective</b>	<b>3</b>
Objectifs . . . . .	3
Déroulé du Module (1h30) . . . . .	3
1. Challenge d'Équipe . . . . .	3
Format . . . . .	3
Scénario . . . . .	3
Spécifications . . . . .	3
Étapes suggérées . . . . .	4
Critères d'évaluation . . . . .	4
2. Démo BMAD Sprint (15 min) . . . . .	5
Présentation . . . . .	5
Démo live : Cycle de vie d'une story . . . . .	5
Flux de statuts BMAD . . . . .	5
Quality Gates . . . . .	6
Points clés pour la démo . . . . .	6
3. Démo QA Recette (15 min) . . . . .	6
Présentation . . . . .	6
Prérequis . . . . .	6
Démo live : Tester une story . . . . .	6
La Golden Rule . . . . .	7
Points clés pour la démo . . . . .	7
4. Questions/Réponses . . . . .	8
Questions fréquentes . . . . .	8
5. Bonnes Pratiques de Collaboration . . . . .	8
Convention de commit avec Claude . . . . .	8
Partage de prompts efficaces . . . . .	9
Code Review avec Claude . . . . .	9
Documentation automatique . . . . .	9
6. Pièges à Éviter . . . . .	9
1. Over-reliance (Dépendance excessive) . . . . .	9
2. Copier-coller aveugle . . . . .	10
3. Ignorer les tests . . . . .	10
4. Négliger la sécurité . . . . .	10
5. Contexte insuffisant . . . . .	10
6. Oublier la validation humaine . . . . .	11
7. Établir les Conventions d'Équipe . . . . .	11
Template de charte IA . . . . .	11
Onboarding nouveau membre . . . . .	11
8. Ressources pour Continuer . . . . .	12
Documentation officielle . . . . .	12
Communauté . . . . .	12

Formation continue . . . . .	12
Support . . . . .	12
Évaluation de la Formation . . . . .	13
Quiz rapide (10 questions) . . . . .	13
Auto-évaluation . . . . .	13
Feedback . . . . .	13
Points Clés à Retenir . . . . .	14
Clôture . . . . .	14
Certificat de participation . . . . .	14
Prochaines étapes . . . . .	14

## Module 9 : Mise en Pratique Collective

### Objectifs

À la fin de ce module, vous serez capable de :

- Appliquer toutes les connaissances acquises sur un cas réel
- Collaborer efficacement en équipe avec Claude
- Utiliser BMAD v6 pour gérer un sprint complet
- Automatiser les tests d'acceptance avec QA Recette
- Établir les bonnes pratiques pour votre équipe
- Éviter les pièges courants

### Déroulé du Module (1h30)

Durée	Contenu	Support
5 min	Présentation du challenge	Slides
30 min	Travail en binômes	Exercice 8
15 min	Démo BMAD Sprint	Terminal
15 min	Démo QA Recette	Chrome + Terminal
15 min	Présentations (2 min/binôme)	-
10 min	Quiz + Discussion	-

### 1. Challenge d'Équipe

#### Format

- **Durée** : 30 minutes
- **Équipes** : Binômes
- **Livrable** : Feature fonctionnelle + tests

#### Scénario

Implémenter un système de **Wishlist** (liste de souhaits) pour un e-commerce.

#### Spécifications

## User Stories

1. En tant qu'utilisateur connecté, je peux ajouter un produit à ma wishlist
2. En tant qu'utilisateur, je peux voir ma wishlist
3. En tant qu'utilisateur, je peux retirer un produit de ma wishlist
4. En tant qu'utilisateur, je reçois une notification si un produit de ma  
⇨ wishlist est en promotion

## ## Contraintes techniques

- Architecture Clean/Hexagonale
- Tests TDD (coverage > 80%)
- API REST avec API Platform
- Pas de code dans les controllers (handlers uniquement)

## Étapes suggérées

### 1. Initialisation workflow et BMAD (5 min)

```
/workflow:init "Implémenter le système de Wishlist"  
/workflow:init  
/sprint:transition US-1 in-progress
```

### 2. Design (5 min)

```
@api-designer "Conçois l'API REST pour la Wishlist"  
@database-architect "Propose le schéma Doctrine"
```

### 3. Tests d'abord (10 min)

```
@tdd-coach "Écris les tests pour AddToWishlistHandler"
```

### 4. Implémentation (10 min)

```
"Implémente AddToWishlistHandler pour passer les tests"  
/symfony:generate-feature Wishlist
```

### 5. Validation Definition of Done (bonus)

```
/gate:validate-story US-1  
/sprint:transition US-1 review
```

## Critères d'évaluation

Critère	Points
Architecture Clean respectée	/25
Tests présents et passants	/25

---

Critère	Points
API fonctionnelle	/25
Code quality (PHPStan)	/15
Documentation	/10

---

## 2. Démo BMAD Sprint (15 min)

### Présentation

BMAD v6 (Business-Minded AI Development) est le framework de gestion de projet intégré à Claude-Craft. Il fournit des commandes dédiées (`/workflow:*`, `/sprint:*`, `/gate:*`) et un système de quality gates pour orchestrer vos sprints.

### Démo live : Cycle de vie d'une story

*# 1. Initialiser le framework BMAD*

`/workflow:init`

*# 2. Vérifier le statut du projet*

`/workflow:status`

*# 3. Récupérer la prochaine story prête*

`/sprint:next-story`

*# 4. Commencer à travailler sur la story*

`/sprint:transition US-1 in-progress`

*# ... travail de développement ...*

*# 5. Passer en review*

`/sprint:transition US-1 review`

*# 6. Valider la Definition of Done*

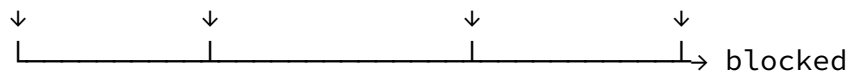
`/gate:validate-story US-1`

*# 7. Marquer comme terminée*

`/sprint:transition US-1 done`

### Flux de statuts BMAD

backlog → ready-for-dev → in-progress → review → done



## Quality Gates

Gate	Seuil	Quand
PRD Gate	$\geq 80\%$	Vision → PRD
Tech Spec Gate	$\geq 90\%$	PRD → Tech Spec
Backlog Gate	INVEST 6/6	Tech Spec → Backlog
Sprint Ready	100%	Backlog → Sprint
Story DoD	100%	Dev → Done

## Points clés pour la démo

- Les quality gates **bloquent** la progression si le seuil n'est pas atteint
- Le cycle TDD est intégré : Red → Green → Refactor
- Les rôles BMAD sont intégrés dans les commandes `/workflow:*` et `/sprint:*`
- `/workflow:status` donne une vue d'ensemble en temps réel

## 3. Démo QA Recette (15 min)

### Présentation

QA Recette est le système de tests d'acceptance automatisés via Chrome. Il applique la **Golden Rule** : un bug corrigé ne doit JAMAIS réapparaître.

### Prérequis

- Extension Chrome v1.0.36+
- Claude Code lancé avec `--chrome` ou `/chrome`

### Démo live : Tester une story

```
# 1. Dry run : voir le plan de test sans exécuter
/qa:recette --scope=story --id=US-1 --dry-run
```

```
# 2. Examiner le plan de test généré
```

```
# → Vérifier les scénarios couverts
# → Vérifier les critères d'acceptance

# 3. Lancer les tests réels via Chrome
/qa:recette --scope=story --id=US-1

# 4. Observer l'automatisation Chrome en direct
# → Claude navigue, clique, remplit les formulaires
# → Chaque assertion est vérifiée visuellement

# 5. Vérifier le statut de la session
/qa:status

# 6. Générer le rapport final
/qa:report

# 7. Corriger les bugs trouvés (dry run d'abord)
/qa:fix --session=REC-xxx --dry-run

# 8. Lancer la correction TDD
/qa:fix --session=REC-xxx --auto-commit
```

## La Golden Rule

Un bug trouvé en recette génère **automatiquement** un test de régression.

```
.recette/
├── plans/           # Plans de test (YAML)
├── sessions/        # États de session
├── regression/      # Suite de régression
│   ├── registry.yaml # Registre des régressions
│   └── tests/        # Tests auto-générés
├── metrics/         # Données historiques
└── reports/         # Rapports générés
```

## Points clés pour la démo

- Le `--dry-run` permet de valider le plan de test avant exécution
- Les sessions peuvent être reprises si interrompues (`--resume`)
- Chaque bug trouvé enrichit automatiquement la suite de régression
- Les rapports fournissent des métriques de couverture et de régression



## 4. Questions/Réponses

### Questions fréquentes

**“Claude peut-il remplacer un développeur?”** Non. Claude est un **assistant**, pas un remplacement : - Il accélère les tâches répétitives - Il aide à explorer des solutions - Il nécessite une validation humaine - Il ne comprend pas le contexte business complet

### “Comment gérer les hallucinations?”

1. Toujours vérifier le code généré
2. Exécuter les tests
3. Demander des sources/références
4. Ne pas hésiter à challenger Claude
5. Utiliser les agents spécialisés (plus fiables)

### “Quelle est la limite de tokens?”

- Sonnet/Opus : ~200K tokens
- 1 token ≈ 3-4 caractères
- Un fichier PHP moyen : ~1-2K tokens
- Stratégie : Charger seulement ce qui est nécessaire

### “Comment partager le contexte en équipe?”

1. CLAUDE.md versionné dans Git
2. Agents et skills partagés
3. Conventions documentées
4. Code review inclut le prompt utilisé

---

## 5. Bonnes Pratiques de Collaboration

### Convention de commit avec Claude

*# Indiquer l'utilisation de Claude dans le commit*  
feat(**order**): add discount calculation

Co-authored-by: Claude <claude@anthropic.com>

Prompt: "Implémente un calculateur de remise avec les règles X, Y, Z"

## Partage de prompts efficaces

## Prompt Template - Feature

### Contexte

[Décrire le contexte business]

### Objectif

[Décrire ce qu'on veut implémenter]

### Contraintes

- [Contrainte 1]

- [Contrainte 2]

### Critères de succès

- [ ] [Critère 1]

- [ ] [Critère 2]

### Exemples

[Si pertinent, donner des exemples]

## Code Review avec Claude

# Avant la PR

@symfony-reviewer "Revois ce code avant que je crée la PR"

# Dans la description de PR

## Review AI

- Points relevés par @symfony-reviewer : [liste]

- Corrections appliquées : [liste]

## Documentation automatique

# Générer la documentation après implémentation

@technical-writer "Génère la documentation pour le module Wishlist:

- README

- API endpoints

- Exemples d'utilisation"

---

## 6. Pièges à Éviter

### 1. Over-reliance (Dépendance excessive)

MAUVAIS:

"Claude, écris toute l'application"

BON:

"Claude, aide-moi à implémenter ce use case spécifique"

## 2. Copier-coller aveugle

MAUVAIS:

Copier le code sans le comprendre

BON:

- Lire et comprendre le code
- Vérifier qu'il correspond au besoin
- Adapter si nécessaire
- Tester

## 3. Ignorer les tests

MAUVAIS:

"Les tests, ce sera pour plus tard"

BON:

TDD systématique, tests en premier

## 4. Négliger la sécurité

MAUVAIS:

Implémenter sans penser sécurité

BON:

@security-auditor après chaque feature critique

## 5. Contexte insuffisant

MAUVAIS:

"Fais un service de paiement"

BON:

"Implémente un PaymentService qui:

- Utilise Stripe API
- Gère les webhooks
- Log les transactions
- Respecte PCI-DSS

Contexte: [fichiers existants, contraintes]"

## 6. Oublier la validation humaine

MAUVAIS:

Déployer directement le code de Claude

BON:

- Code review humaine
  - Tests automatisés
  - Validation métier
  - Staging avant prod
- 

## 7. Établir les Conventions d'Équipe

### Template de charte IA

# Charte d'utilisation de Claude Code - [Équipe]

#### ## Principes

1. Claude est un assistant, pas un remplacement
2. Tout code généré doit être compris et validé
3. TDD obligatoire, même avec l'IA
4. Sécurité toujours vérifiée

#### ## Workflow

1. Nouvelle feature → /workflow:init
2. Génération → Validation → Tests → Review
3. Commit avec mention Claude si pertinent

#### ## Agents autorisés

- @symfony-reviewer : Code review
- @tdd-coach : Écriture de tests
- @security-auditor : Audit sécurité

#### ## Limites

- Ne pas utiliser pour : [liste]
- Toujours valider humainement : [liste]

#### ## Métriques

- Couverture tests : > 80%
- PHPStan : Level 8
- Security audit : Avant chaque release

### Onboarding nouveau membre

#### ## Onboarding Claude Code

**### Jour 1**

- [ ] Installation Claude Code
- [ ] Configuration clé API
- [ ] Tour des commandes de base

**### Jour 2**

- [ ] Installation Claude-Craft sur le projet
- [ ] Découverte des agents
- [ ] Premier exercice guidé

**### Semaine 1**

- [ ] Première feature avec workflow Standard
  - [ ] Code review assistée
  - [ ] Session feedback
- 

**8. Ressources pour Continuer****Documentation officielle**

- Claude Code Documentation
- Anthropic Cookbook
- Claude-Craft Repository

**Communauté**

- Discord Anthropic
- GitHub Discussions Claude-Craft
- Meetups locaux IA & Dev

**Formation continue**

- Webinaires mensuels
- Nouvelles features Claude
- Mises à jour Claude-Craft

**Support**

- Email formateur
  - Canal Slack/Discord dédié (si option)
  - Documentation interne équipe
-

## Évaluation de la Formation

### Quiz rapide (10 questions)

1. Quels sont les 3 tracks de workflow ?
2. Quelle commande pour un audit complet ?
3. Comment invoquer un agent ?
4. Quel est le cycle TDD ?
5. Que fait Ralph Wiggum ?
6. Qu'est-ce que BMAD v6 et quelles commandes utilise-t-il ?
7. Que fait Ralph Wiggum et quels sont les types de DoD validators ?
8. Qu'est-ce que la Golden Rule de QA Recette ?
9. Quelle est la hiérarchie d'Extended Thinking ?
10. Qu'est-ce que MCP et comment l'utiliser ?

### Auto-évaluation

Niveau de confiance (1-5) :

- ☐ Installation et configuration Claude Code
- ☐ Utilisation des workflows
- ☐ Génération de features Symfony
- ☐ Audit de projets existants
- ☐ Utilisation des agents
- ☐ Écriture de tests avec TDD
- ☐ Utilisation de BMAD v6 (sprints, quality gates, transitions)
- ☐ Orchestration avec Ralph Wiggum (boucle continue, DoD)
- ☐ Tests d'acceptance avec QA Recette (plans, sessions, régression)
- ☐ Intégration dans mon workflow quotidien

### Feedback

1. Ce qui a été le plus utile :

-----

2. Ce qui pourrait être amélioré :

-----

3. Questions restantes :

-----

\_\_\_\_\_

## Points Clés à Retenir

1. **Claude = Assistant** : Toujours valider humainement
  2. **TDD** : Tests d'abord, même avec l'IA
  3. **Contexte** : Plus le prompt est précis, meilleur est le résultat
  4. **BMAD v6** : Structurer le développement avec quality gates et transitions
  5. **QA Recette** : La Golden Rule garantit zéro régression
  6. **Ralph Wiggum** : Automatiser les tâches longues en boucle continue
  7. **Collaboration** : Partager les bonnes pratiques en équipe
  8. **Sécurité** : Ne jamais négliger l'audit de sécurité
- 

## Clôture

### Certificat de participation

Remis à chaque participant ayant : - Assisté aux 2 jours - Complété les exercices principaux - Participé au challenge d'équipe

### Prochaines étapes

1. **Semaine 1** : Appliquer sur un petit projet/feature
  2. **Semaine 2** : Session de suivi (si option)
  3. **Mois 1** : Coaching continu (si premium)
  4. **Continu** : Veille et mise à jour des pratiques
- 

**Merci pour votre participation !**

**The Bearded CTO**