

## Module 4 — Nouveau Projet Symfony

Jour 1 — Démarrer un projet from scratch

The Bearded Bear

Février 2026



## Table des matières

<b>Module 4 : Démarrer un Nouveau Projet Symfony</b>	<b>2</b>
Objectifs . . . . .	2
1. Configuration Initiale . . . . .	2
Création du projet Symfony . . . . .	2
Installation de Claude-Craft . . . . .	2
Structure générée . . . . .	2
Configuration du contexte projet . . . . .	3
2. Architecture Clean + Hexagonale . . . . .	4
Structure des répertoires . . . . .	4
Règles de dépendances . . . . .	5
3. Génération de Features . . . . .	5
/symfony :generate-feature . . . . .	5
/symfony :generate-entity . . . . .	6
/symfony :generate-crud . . . . .	6
4. API Platform Integration . . . . .	6
Configuration . . . . .	6
Endpoint généré . . . . .	7
Documentation OpenAPI . . . . .	7
5. Respect Automatique des Patterns . . . . .	7
SOLID . . . . .	7
DRY, KISS, YAGNI . . . . .	7
PSR-12 et Standards . . . . .	8
6. Atelier Pratique . . . . .	8
Objectif . . . . .	8
Étapes . . . . .	8
Critères de succès . . . . .	9
Points Clés à Retenir . . . . .	9

## Module 4 : Démarrer un Nouveau Projet Symfony

### Objectifs

À la fin de ce module, vous serez capable de : - Configurer Claude-Craft sur un nouveau projet Symfony  
- Générer des features complètes avec les commandes - Appliquer l'architecture Clean/Hexagonale -  
Respecter automatiquement les patterns et standards

### 1. Configuration Initiale

#### Création du projet Symfony

```
# Créer un nouveau projet Symfony
```

```
symfony new mon-api --webapp
```

```
# Ou avec Composer via Docker (recommandé)
```

```
docker compose exec app composer create-project symfony/skeleton mon-api  
cd mon-api
```

```
docker compose exec app composer require webapp
```

**Règle Docker :** Toujours utiliser Docker pour les commandes afin de s'abstraire de l'environnement local.

#### Installation de Claude-Craft

```
# Avec npx (v7.13.0)
```

```
npx @the-bearded-bear/claude-craft install ~/mon-api --tech=symfony
```

```
↪ --lang=fr
```

```
# Ou depuis le répertoire claude-craft avec Make
```

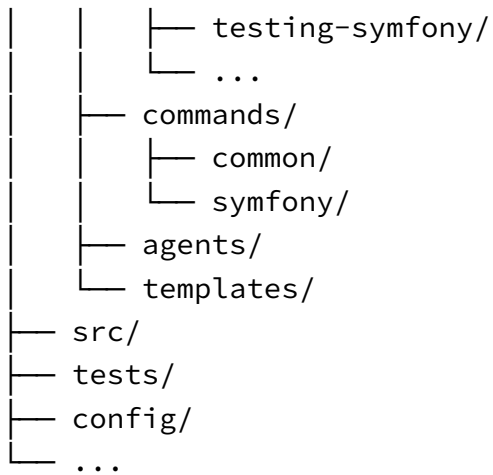
```
make install-symfony TARGET=~/mon-api LANG=fr
```

```
# Vérifier l'installation
```

```
ls ~/mon-api/.claude/
```

#### Structure générée

```
mon-api/  
├── .claude/  
│   ├── CLAUDE.md  
│   ├── skills/  
│   │   ├── architecture-clean-ddd/  
│   │   └── coding-standards/
```



### Configuration du contexte projet

*# Ouvrir Claude dans le projet*

```
cd ~/mon-api
```

```
claude
```

*# Configurer interactivement*

```
/common:setup-project-context
```

Ou manuellement éditer `.claude/rules/00-project-context.md`:

```
# MonAPI - Projet Symfony
```

```
## Project Context
```

- **Project Name**: MonAPI
- **Technology Stack**: Symfony 8.0, PHP 8.5, PostgreSQL
- **Architecture**: Clean Architecture + Hexagonal + DDD

```
## Domain Entities
```

- User (authentification)
- Product (catalogue)
- Order (commandes)
- Cart (panier)

```
## External Services
```

- Database: PostgreSQL 17
- Cache: Redis 7
- Queue: RabbitMQ (Symfony Messenger)
- Search: Elasticsearch (optionnel)

```
## Team Conventions
```

- Branch: feature/, bugfix/, hotfix/

- Commits: Conventional Commits (feat:, fix:, refactor:)
- PR: Review obligatoire, CI verte

## 2. Architecture Clean + Hexagonale

### Structure des répertoires

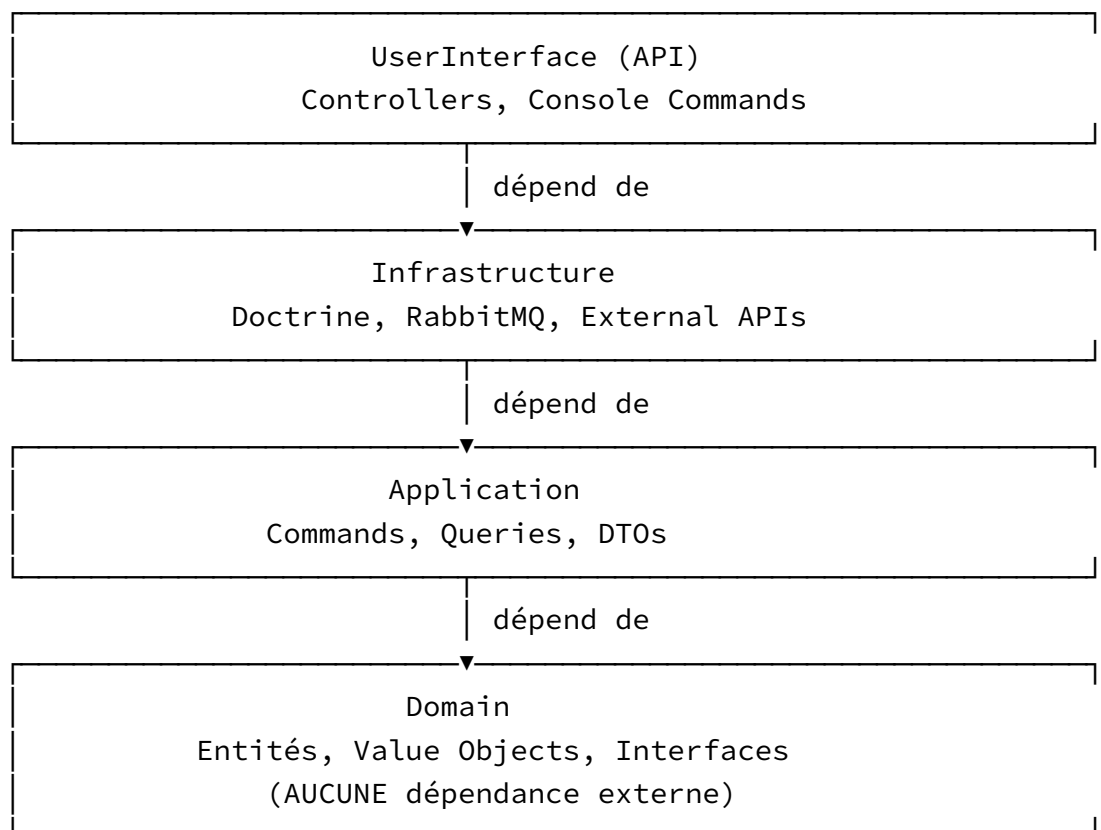
```

src/
├── Domain/                                # Cœur métier (pas de dépendances)
│   ├── Model/                            # Entités, Value Objects
│   │   ├── User/
│   │   │   ├── User.php
│   │   │   ├── UserId.php              # Value Object
│   │   │   └── Email.php               # Value Object
│   │   └── Order/
│   │       ├── Order.php
│   │       └── OrderItem.php
│   ├── Repository/                      # Interfaces (Ports)
│   │   ├── UserRepositoryInterface.php
│   │   └── OrderRepositoryInterface.php
│   ├── Service/                         # Services domaine
│   │   └── OrderCalculator.php
│   └── Event/                           # Domain Events
│       └── OrderCreated.php
├── Application/                         # Use Cases (Orchestration)
│   ├── Command/                         # Write operations
│   │   └── CreateOrder/
│   │       ├── CreateOrderCommand.php
│   │       └── CreateOrderHandler.php
│   ├── Query/                           # Read operations
│   │   └── GetOrderById/
│   │       ├── GetOrderByIdQuery.php
│   │       └── GetOrderByIdHandler.php
│   └── DTO/                             # Data Transfer Objects
│       └── OrderDTO.php
├── Infrastructure/                      # Implémentations techniques
│   ├── Persistence/                    # Doctrine repositories
│   │   └── DoctrineUserRepository.php
│   └── Messaging/                      # Queues, events

```

```
|
| | └─ RabbitMQPublisher.php
| | └─ External/                # APIs externes
| |   └─ StripePaymentGateway.php
|
└─ UserInterface/                # Présentation
    └─ Api/                      # Controllers API
        └─ OrderController.php
    └─ Console/                  # Commandes CLI
        └─ ImportProductsCommand.php
```

### Règles de dépendances



### 3. Génération de Features

#### **`/symfony:generate-feature`**

Génère une feature complète avec tous les fichiers :

```
/symfony:generate-feature Order
```

```
# Génère:  
# - Domain/Model/Order/Order.php  
# - Domain/Model/Order/OrderId.php  
# - Domain/Repository/OrderRepositoryInterface.php  
# - Application/Command/CreateOrder/CreateOrderCommand.php  
# - Application/Command/CreateOrder/CreateOrderHandler.php  
# - Infrastructure/Persistence/DoctrineOrderRepository.php  
# - UserInterface/Api/OrderController.php  
# - tests/Unit/Domain/Model/OrderTest.php  
# - tests/Integration/OrderRepositoryTest.php
```

### **/symfony:generate-entity**

Génère une entité Doctrine avec Value Objects :

```
/symfony:generate-entity Product name:string price:money stock:integer
```

```
# Génère:  
# - Domain/Model/Product/Product.php  
# - Domain/Model/Product/ProductId.php  
# - Domain/Model/Product/Price.php (Value Object pour Money)  
# - config/doctrine/Product.orm.xml
```

### **/symfony:generate-crud**

Génère un CRUD complet :

```
/symfony:generate-crud Category
```

```
# Génère:  
# - Create, Read, Update, Delete handlers  
# - Endpoints API REST  
# - Validation  
# - Tests
```

---

## **4. API Platform Integration**

### **Configuration**

```
# Installer API Platform via Docker  
docker compose exec app composer require api-platform/core
```

```
# Générer une ressource API  
/symfony:generate-api-resource Product
```

## Endpoint généré

```
// src/UserInterface/Api/Resource/ProductResource.php
#[ApiResponse(
    operations: [
        new Get(),
        new GetCollection(),
        new Post(),
        new Put(),
        new Delete(),
    ],
    normalizationContext: ['groups' => ['product:read']],
    denormalizationContext: ['groups' => ['product:write']],
)]
class ProductResource
{
    #[Groups(['product:read'])]
    public string $id;

    #[Groups(['product:read', 'product:write'])]
    public string $name;

    #[Groups(['product:read', 'product:write'])]
    public float $price;
}
```

## Documentation OpenAPI

Automatiquement générée et accessible sur `/api/docs`.

---

## 5. Respect Automatique des Patterns

### SOLID

Claude-Craft impose automatiquement :

- **Single Responsibility** : Une classe = une responsabilité
- **Open/Closed** : Extension via interfaces, pas modification
- **Liskov Substitution** : Les implémentations respectent les contrats
- **Interface Segregation** : Interfaces petites et focalisées
- **Dependency Inversion** : Dépendre des abstractions

### DRY, KISS, YAGNI

- **DRY** : Pas de duplication de code



- **KISS** : Solutions simples, pas d'over-engineering
- **YAGNI** : Seulement ce qui est demandé, pas de code "au cas où"

## PSR-12 et Standards

```
# Vérifier les standards automatiquement
/symfony:check-code-quality

# Appliquer les corrections via Docker
docker compose exec app composer run cs-fix
```

---

## 6. Atelier Pratique

### Objectif

Créer un micro-projet "Gestion de Tâches" avec : - Entité Task - CRUD complet - API REST - Tests

### Étapes

#### 1. Créer le projet

```
symfony new task-manager --webapp

# Alternative Docker (si pas de Symfony CLI locale)
docker compose exec app composer create-project symfony/skeleton
  ↳ task-manager
docker compose exec app composer require webapp
  ↳ --working-dir=task-manager

# Installer Claude-Craft (v7.13.0)
npx @the-bearded-bear/claude-craft install ./task-manager
  ↳ --tech=symfony --lang=fr

cd task-manager && claude
```

#### 2. Configurer le contexte

```
/common:setup-project-context
# Nom: TaskManager
# Stack: Symfony 8.0, PHP 8.5, SQLite (dev)
# Entités: Task, User
```

#### 3. Générer la feature Task

```
/symfony:generate-feature Task
```

#### 4. Ajouter les champs

```
"Ajoute les champs title:string, description:text,  
↪ status:enum(pending,done), dueDate:datetime à Task"
```

#### 5. Créer l'endpoint API

```
/symfony:generate-api-resource Task
```

#### 6. Écrire les tests

```
@tdd-coach "Écris les tests pour le use case CreateTask"
```

#### 7. Lancer les tests via Docker

```
docker compose exec app ./vendor/bin/phpunit
```

#### 8. Vérifier la qualité

```
/symfony:check-compliance
```

### Critères de succès

- ☐ Entité Task avec Value Objects
  - ☐ Handler CreateTaskHandler
  - ☐ Repository avec interface
  - ☐ Endpoint POST /api/tasks
  - ☐ Tests unitaires passants (docker compose exec app ./vendor/bin/phpunit)
  - ☐ Compliance check vert
- 

### Points Clés à Retenir

1. **Docker first** : Toujours exécuter les commandes via `docker compose exec app`
  2. **Clean Architecture** : Domain → Application → Infrastructure → UI
  3. **Commandes de génération** : `/symfony:generate-feature, -entity, -crud`
  4. **Inversion de dépendances** : Le Domain ne dépend de rien
  5. **API Platform** : Documentation automatique OpenAPI
  6. **Qualité intégrée** : SOLID, DRY, KISS, YAGNI, PSR-12
- 

**Durée estimée** : 2h **Prochain module** : (Jour 2) Intégrer Claude-Craft sur un Projet Existant