

1. Introduction Ce document présente l'architecture logicielle du projet Project Management, réalisé dans le cadre du module Architecture Logicielle. L'objectif principal de ce projet n'est pas de fournir une application entièrement fonctionnelle, mais de concevoir une architecture logicielle cohérente, maintenable et testable, en appliquant les principes vus en cours, notamment la séparation des responsabilités et les architectures orientées domaine.
2. Présentation du domaine fonctionnel L'application développée est une application de gestion de projet de type Kanban. Elle permet notamment : la création et la gestion de projets, l'ajout et le suivi de tâches, la gestion de l'état des tâches (à faire, en cours, terminé), la traçabilité des actions effectuées dans le système. Le périmètre fonctionnel a volontairement été limité afin de se concentrer sur les choix architecturaux et la qualité de la structure du code, plutôt que sur la richesse fonctionnelle.
3. Architecture retenue 3.1 Choix de l'architecture hexagonale Le projet repose sur une architecture hexagonale (Ports & Adapters). Ce modèle architectural place le domaine métier au centre de l'application, en l'isolant des détails techniques tels que le framework, la base de données ou les interfaces utilisateurs. Les principes fondamentaux de cette architecture sont : le domaine ne dépend d'aucune technologie externe, les dépendances sont orientées vers le cœur métier, les interactions avec l'extérieur se font via des interfaces (ports), les implémentations techniques sont fournies par des adapters. Cette approche favorise la testabilité, la lisibilité du code et la maintenabilité à long terme.
4. Choix technologiques 4.1 Utilisation de Laravel Le framework Laravel a été choisi pour servir d'infrastructure technique. Il est utilisé principalement pour : la gestion des requêtes HTTP, l'injection de dépendances, la configuration de l'application, le support des tests. Laravel n'intervient pas dans la logique métier. Le domaine et les cas d'usage ne dépendent pas du framework, ce qui permet de limiter le couplage technologique. 4.2 Langage PHP Le langage PHP a été retenu pour sa simplicité, sa maturité et sa compatibilité naturelle avec Laravel. Il permet également une écriture claire des concepts métiers et des tests unitaires.
5. Organisation du code 5.1 Couche Domaine La couche domaine contient l'ensemble de la logique métier : les entités principales (Project, Task), les value objects (ProjectId, TaskStatus), les interfaces définissant les ports, comme ProjectRepository. Les règles métier (changement d'état d'une tâche, ajout de tâches à un projet) sont encapsulées directement dans les entités, garantissant la cohérence du domaine. Cette couche ne dépend d'aucun élément du framework Laravel. 5.2 Couche Application La couche application regroupe les cas d'usage du système. Elle contient la logique d'orchestration, par exemple : la création d'un projet, l'ajout d'une tâche à un projet existant. Les cas d'usage utilisent les interfaces définies dans le domaine et ne connaissent pas les détails techniques de persistance ou de transport des données. 5.3 Couche Infrastructure La couche infrastructure regroupe les éléments techniques : les contrôleurs HTTP Laravel, les implémentations concrètes des repositories, la configuration de l'injection de dépendances. Elle implémente les ports définis dans le domaine et permet la communication avec l'extérieur (HTTP, base de données, framework).
6. Tests Des tests unitaires ont été mis en place afin de valider : le comportement des entités du domaine, le fonctionnement des cas d'usage. Les dépendances techniques sont remplacées par des implémentations factices (fakes), ce qui permet de tester les use cases de manière isolée, sans dépendre de Laravel ou d'une base de données. Cette approche renforce la fiabilité du code et facilite son évolution.
7. Décisions d'architecture Les principales décisions architecturales ont été documentées à l'aide d'Architecture Decision Records (ADR). Ces documents expliquent notamment : le choix de l'architecture hexagonale, l'utilisation de Laravel comme infrastructure, l'adoption d'un modèle event-driven interne pour le journal d'activité. Les ADR permettent de conserver une trace claire des choix réalisés et de leurs conséquences.
8. Conclusion Ce projet met l'accent sur la qualité architecturale, la séparation des responsabilités et la testabilité du code. L'architecture hexagonale choisie permet de faire évoluer l'application dans le temps tout en limitant le couplage avec les technologies utilisées. Elle constitue une base saine pour un projet de plus grande envergure.