ARCHILOG

Rapport projet

Suivis par :

MELOSSO Tom 2.1

IDOUX Clément 2.1

SCHALLER Théo 2.2

Date de réalisation :

Année 2024/2025

Analyse des Exigences et Réalisations

Architecture Hexagonale

Attendu : Implémentation d'une architecture hexagonale complète avec domaine métier central, ports d'entrée et ports de sortie.
Réalisé :

- L'architecture hexagonale est partiellement respectée avec une séparation claire entre :
    - Le domaine métier (model/)
    - Les services d'application (app/)
    - L'infrastructure (infrastructure/)
- La structure du projet montre une organisation selon les principes hexagonaux, mais certaines dépendances traversent les couches.

Ports d'Entrée (Primaires)

Attendu : Au moins deux ports d'entrée.
Réalisé :

- Interface utilisateur web (✓)
- API REST (endpoints PHP) (✓)

Ports de Sortie (Secondaires)

Attendu : Au moins trois ports de sortie dont la persistance et un webhook Discord.
Réalisé :

- Persistance des données (Base de données SQL via PHP) (✓)
- Service de stockage local (localStorage) (✓)
- Pas de webhook Discord implémenté (✗)
- Pas de troisième port de sortie (email, API externe, etc.) (✗)

Fonctionnalités Principales

Attendu : Création de tableaux, gestion des listes et cartes, déplacement des cartes entre listes.
Réalisé :

- Création et gestion des tableaux (✓)
- Ajout, modification et suppression de listes (✓)
- Ajout, modification et suppression de cartes (✓)
- Glisser-déposer pour déplacer les cartes entre listes (✓)
- Descriptions sur les cartes (✓)
- Interface intuitive avec modales (✓)

Authentification

Attendu : Non spécifié initialement dans le cahier des charges mais implicite dans les fonctionnalités utilisateur.
Réalisé :

- Système de connexion/inscription complet (✓)
- Gestion de sessions PHP (✓)

Design Patterns

Attendu : Implémentation d'au moins 5 design patterns.
Réalisé :

- DAO (Data Access Object) pour l'accès aux données (✓)
- Factory implicite dans les services (partiel)
- Pas d'Observeur implémenté (✗)
- Pas de CQRS implémenté (✗)
- Pas d'Event Sourcing (✗)
- Adapter pattern pour la gestion du DOM (✓)

Technologies

Attendu : TypeScript, HTML, CSS, JavaScript, SQL.
Réalisé :

- JavaScript (au lieu de TypeScript) (partiel)
- HTML (✓)
- CSS (✓)
- SQL (via PHP) (✓)

Tests

Attendu : Tests unitaires pour les composants du domaine et tests d'intégration.
Réalisé :

- Aucun test unitaire visible (✗)
- Aucun test d'intégration visible (✗)

Analyse Détaillée par Composant

Modèle de Domaine

Le modèle de domaine est simple mais efficace, composé de trois classes principales :

- Board: Représente un tableau contenant des listes
- List: Représente une liste contenant des cartes
- Card: Représente une carte avec titre et description

La structure est cohérente avec le domaine métier et respecte la séparation des responsabilités.

Services d'Application

Le BoardService gère les opérations de domaine et les interactions avec la persistance. Il existe une bonne séparation entre :

- Les opérations de modification du domaine
- Les appels à l'API backend
- La gestion des erreurs

Le StorageService sert d'adaptateur pour la persistance, bien que le localStorage soit encore utilisé comme solution de secours.

Infrastructure

L'adaptateur DOM (DOMAdapter) sépare correctement les préoccupations d'interface utilisateur du domaine métier. Il se charge de :

- Le rendu des composants de l'interface
- La gestion des événements utilisateur
- L'interaction avec les modales

Backend

L'implémentation du backend en PHP utilise :

- Une architecture DAO bien structurée
- Une gestion des sessions pour l'authentification
- Des endpoints API REST pour interagir avec le frontend

Points Forts du Projet

1. Interface utilisateur intuitive avec modales et glisser-déposer
2. Séparation claire des responsabilités entre domaine, application et infrastructure
3. Système d'authentification complet et fonctionnel
4. Persistance des données bien implémentée avec une architecture DAO

Points d'Amélioration

1. Design Patterns manquants : Observeur, CQRS, Event Sourcing
2. Ports de sortie incomplets : Webhook Discord et troisième port non implémentés
3. Tests absents : Aucun test unitaire ou d'intégration
4. TypeScript non utilisé : Le code est en JavaScript pur

Conclusion

Le projet Ollert présente une implémentation fonctionnelle d'un utilitaire de gestion de tâches inspiré de Trello. L'architecture hexagonale est partiellement respectée, avec une bonne séparation des couches mais des interfaces parfois insuffisamment définies.
Les fonctionnalités principales sont bien implémentées, avec un système d'authentification robuste qui va au-delà des exigences initiales. L'interface utilisateur est intuitive et offre une bonne expérience utilisateur.
Cependant, plusieurs exigences techniques n'ont pas été satisfaites, notamment l'implémentation de certains design patterns (Observeur, CQRS, Event Sourcing), de ports de sortie supplémentaires (webhook Discord), et l'absence de tests.

Perspectives d'Amélioration

Avec plus de temps, il serait possible d'implémenter :

1. Design Patterns manquants :
    - Observeur pour notifier les changements dans le tableau
    - CQRS pour séparer les lectures et écritures
    - Event Sourcing pour stocker l'historique des actions
2. Ports de sortie supplémentaires :
    - Webhook Discord pour les notifications
    - Service d'email pour les alertes et rappels
3. Tests :
    - Tests unitaires pour les composants du domaine
    - Tests d'intégration pour les adaptateurs
4. Migration vers TypeScript pour améliorer la robustesse du code
5. Améliorations fonctionnelles :
    - Dates limites pour les cartes
    - Système de filtrage pour rechercher des cartes
    - Collaboration en temps réel

Ces améliorations permettraient de respecter pleinement les exigences du projet tout en enrichissant l'expérience utilisateur.