Rapport Application de Gestion des Tâches

Pourcelot Matthis - Porchet Hugues

1. Présentation du Domaine d'Application Choisi

La gestion des tâches est un enjeu majeur dans la productivité personnelle et professionnelle. De nombreuses personnes rencontrent des difficultés à organiser leur travail, respecter des échéances et maintenir leur motivation sur le long terme. Ce projet vise à répondre à ces problématiques en proposant une application web innovante qui allie **gestion des tâches, gamification et méthodologie Pomodoro**.

L'objectif est d'offrir une solution complète permettant aux utilisateurs de **planifier, suivre et terminer leurs tâches efficacement**, tout en étant encouragés à travers un système de **récompenses et d'évolution**.

Problématiques ciblées

1. **Manque d'organisation et procrastination**
    - Les utilisateurs peinent à structurer leurs tâches et à prioriser leurs actions.
    - Solution : Interface intuitive et statuts des tâches pour mieux organiser le travail.
2. **Perte de motivation**
    - L'absence de suivi visible des progrès peut démotiver les utilisateurs.
    - Solution : Système de points d'expérience et avatar évolutif pour encourager la productivité.
3. **Difficulté à maintenir la concentration**
    - Les distractions sont nombreuses et peuvent nuire à la productivité.
    - Solution : Intégration de la méthode Pomodoro pour des sessions de travail efficaces.
4. **Oublis et non-respect des échéances**
    - Les utilisateurs oublient parfois de traiter des tâches importantes.
    - Solution : Notifications par Discord pour rappeler les deadlines.

L'application s'adresse donc aussi bien aux professionnels (freelancers, équipes de travail) qu'aux étudiants ou aux personnes cherchant à améliorer leur gestion du temps.

Complémentarité entre gamification et méthode Pomodoro

L'association de la gamification et de la méthode Pomodoro permet d'optimiser la productivité en jouant sur deux leviers psychologiques majeurs : la motivation extrinsèque et la gestion du temps.

- **Motivation et engagement** : le système de points d'expérience et d'avatar évolutif encourage l'utilisateur à rester impliqué sur le long terme. Chaque session Pomodoro

réussie peut être associée à une récompense virtuelle (succès débloqué, gain d'expérience).
- **Rythme de travail optimal** : la méthode Pomodoro favorise la concentration en segmentant le travail en périodes courtes, évitant ainsi la surcharge cognitive.
- **Objectifs clairs et progression visible** : le suivi des tâches complétées et les niveaux gagnés grâce à la gamification offrent une sensation d'accomplissement qui renforce l'engagement de l'utilisateur.

2. Présentation de l'Architecture

L'application suit une **architecture hexagonale** (ports et adaptateurs), permettant une séparation claire entre le domaine métier et les implémentations techniques.

2.1 Ports d'Entrée

- **Interface Web** : développée avec Twig/CSS pour la présentation des tâches et l'interaction avec l'utilisateur.
- **API REST** : permet aux clients externes (ex. une appli mobile ou un autre service) de créer, modifier, supprimer des tâches via des requêtes HTTP.

2.2 Ports de Sortie

- **Base de données MySQL** : persistance des données des utilisateurs et des tâches.
- **Webhook Discord** : notifications en temps réel envoyées aux utilisateurs via Discord lors de l'avancée des statuts des tâches.
- **Système de logs** : un système de fichiers automatisé qui, chaque jour, enregistre les tâches terminées avec les informations de la tâche et de l'utilisateur.

2.3. Adaptateurs

Les adaptateurs implémentent concrètement les ports pour assurer leur communication avec les systèmes externes.

- **HttpController** (adaptateur d'entrée) : Gère les requêtes HTTP et les convertit en appels métier.

- **DoctrineRepository** (adaptateur de sortie) : Implémente l'interface Repository pour interagir avec la base de données.

- **DiscordNotifier** (adaptateur de sortie) : Envoie des notifications via un webhook Discord.

- **EmailNotifier** (adaptateur de sortie) : Gère l'envoi d'emails de notification aux utilisateurs.

L'implémentation de cette architecture garantit une modularité accrue et facilite les évolutions futures sans impact majeur sur les couches métier.

Architecture détaillée du backend

1. Structure générale

Le projet est organisé en plusieurs couches logiques, séparant les responsabilités :

- **src/** : contient l'ensemble du code applicatif.
- **Application/** : implémente la logique métier et les ports d'entrée/sortie.
- **Domain/** : définit les entités et les règles métier.
- **Infrastructure/** : gère les interactions avec les services externes (base de données, notifications, etc.).

2. Détails des principaux modules

2.1 Application Layer (Couche d'Application)

Située dans **src/Application/**, cette couche contient :

- **Command/** : contient des commandes exécutables en ligne de commande.
- **Port/** : interface entre l'application et l'extérieur.
    - **Http/Controller/** : gestion des requêtes HTTP (ex: TaskController.php, PomodoroController.php).
    - **Repository/** : interfaces définissant comment accéder aux données (TaskRepositoryInterface.php).
- **Service/** : implémente la logique métier principale.
    - TaskService.php pour la gestion des tâches.
    - PomodoroService.php pour la gestion du timer Pomodoro.

2.2 Domain Layer (Couche Métier)

Située dans **src/Domain/**, elle regroupe les règles métier principales et contient :

- **Notification/** : implémente un système d'observateurs (TaskNotifier.php).
- **Task/** : définit l'entité Task.php et son repository.
- **User/** : définit les entités utilisateur et la gestion des noms d'utilisateur.

2.3 Infrastructure Layer (Couche Infrastructure)

Située dans **src/Infrastructure/**, elle gère les interactions techniques :

- **Persistence/** : interaction avec la base de données via Doctrine.

- **Notification/** : envoi des notifications (ex: DiscordNotifier.php).
- **Timer/** : gestion du timer Pomodoro avec différents modes (StandardPomodoroStrategy.php).

3. Autres composants notables

- **DataFixtures/** : gère des données de test initiales.
- **Form/** : définit les formulaires utilisés dans l'application.
- **EntityListener/** : contient des listeners pour intercepter les événements sur les entités.
- **Reports/** : stocke les fichiers de suivi des tâches.

3. Explication des Choix Technologiques

3.1 Technologies utilisées

- **Backend** : PHP avec Symfony pour la gestion des tâches et des utilisateurs.
- **Base de données** : MySQL pour stocker les informations.
- **Frontend** : Twig pour la présentation, Bootstrap et CSS pour le design.
- **Tests** : PHPUnit pour assurer la qualité du code.

3.2 Comparaison des alternatives envisagées : PHP/Symfony vs Node.js ou Django

Lors du choix du stack technologique, plusieurs options ont été étudiées :

Technologie	Avantages	Inconvénients
PHP/Symfony (choisi)	Robuste, structuré, excellent support pour les projets à long terme, gestion des dépendances avec Composer, sécurité avancée	Peut être plus verbeux que d'autres frameworks
Node.js (Express, Nest.js)	Haute performance en temps réel (ex: WebSockets), stack full JS unifiée, non bloquant	Moins structuré qu'un framework comme Symfony, gestion plus complexe des packages

Django (Python)	Rapidité de développement, ORM puissant, bon support pour l'IA et le Machine Learni	Moins performant sur des requêtes complexes, moins optimisé pour de gros volumes de trafic

Le choix de **Symfony** a été motivé par sa solidité sur des architectures complexes, sa compatibilité avec MySQL et son écosystème mature.

3.3 Explication du choix de MySQL : Scalabilité, intégrité des données et compatibilité avec Symfony

L'application repose sur **MySQL** pour le stockage des tâches et des utilisateurs. Ce choix repose sur plusieurs critères :

- **Scalabilité** : MySQL permet une bonne gestion des transactions et peut être optimisé pour supporter une charge importante (indexation, partitionnement).
- **Intégrité des données** : support des contraintes relationnelles (clés étrangères, transactions ACID), garantissant la cohérence des données.
- **Compatibilité avec Symfony** : doctrine ORM, utilisé avec Symfony, offre une interface fluide pour interagir avec MySQL tout en facilitant les migrations et l'abstraction des requêtes.

D'autres options comme PostgreSQL (plus robuste sur les transactions complexes) ont été envisagées, mais MySQL s'est imposé pour sa légèreté et sa facilité d'intégration.

3.4 Pourquoi Twig pour le frontend ? Comparaison avec React et Vue.js

L'interface utilisateur a été développée avec **Twig**, un moteur de templates utilisé avec Symfony. Voici une comparaison avec d'autres solutions :

Technologie	Avantages	Inconvénients
Twig (choisi)	Léger, rapide, sécurisé (prévention des injections XSS), parfait pour des pages dynamiques générées côté serveur	Moins interactif que les solutions SPA (Single Page Application)
React.js	Très dynamique, composants réutilisables, bonne gestion de l'état avec Redux	Nécessite un backend API REST bien structuré, peut être plus lourd à charger

Vue.js	Plus simple que React, intégration progressive possible	Moins de support natif avec Symfony qu'avec Twig

Le choix de Twig s'explique par sa simplicité d'intégration avec Symfony et l'absence de besoin d'une application très dynamique en front.

3.5 Design Patterns

Justification des Choix de Design Patterns

L'utilisation de design patterns dans l'application permet d'améliorer la flexibilité et la maintenabilité du code. Plusieurs patterns ont été choisis pour répondre aux exigences du projet.

1. Observer (Notifications et suivi des tâches)

- **Problème** : Nécessité de notifier automatiquement les utilisateurs lors d'un changement d'état d'une tâche.
- **Solution** : Implémentation du pattern Observer via un système de notifications (DiscordNotifier). Lorsqu'une tâche est mise à jour, l'observateur enregistré est automatiquement informé.
- **Avantages** : Découplage entre la gestion des tâches et l'envoi des notifications, facilitant l'ajout de nouveaux canaux de notification.

2. Strategy (Gestion du Timer Pomodoro)

- **Problème** : Différentes stratégies de gestion du temps (sessions de 15, 25 ou 45 minutes) nécessitent une flexibilité.
- **Solution** : Implémentation du pattern Strategy avec plusieurs classes concrètes (ShortPomodoroStrategy, StandardPomodoroStrategy, LongPomodoroStrategy), permettant de modifier dynamiquement le comportement du timer.
- **Avantages** : Permet d'ajouter facilement de nouvelles stratégies sans modifier la logique principale.

3. State (Gestion des statuts des tâches)

- **Problème** : Les tâches peuvent évoluer à travers différents statuts (À faire, En cours, Terminé), nécessitant une gestion flexible des transitions.

- **Solution** : Application du pattern State avec des classes représentant chaque état et des transitions bien définies.

- **Avantages** : Permet d'éviter des structures conditionnelles complexes et rend les transitions plus faciles à maintenir.

4. Repository (Gestion des accès aux données)

- **Problème** : Besoin de séparer la logique métier de l'accès aux données pour faciliter la maintenance et l'évolution du stockage.

- **Solution** : Utilisation du pattern Repository pour centraliser l'accès aux données (TaskRepositoryInterface, UserRepositoryInterface) avec des implémentations spécifiques (DoctrineTaskRepository).

- **Avantages** : Abstraction de la base de données, facilitant le changement de technologie de persistance sans impacter le domaine métier.

5. Service Layer (Gestion des Règles Métier)

- **Problème** : Éviter que la logique métier soit dispersée dans les contrôleurs.

- **Solution** : Création de services dédiés (TaskService, PomodoroService) qui centralisent les règles métier et assurent l'orchestration des actions.

- **Avantages** : Séparation claire entre la logique métier et les interfaces utilisateurs, facilitant le test unitaire et l'évolution du code.

En intégrant ces patterns, l'application devient plus robuste, modulaire et maintenable, garantissant ainsi une évolution simplifiée et une adaptation facile aux futurs besoins.

4. Difficultés Rencontrées et Solutions Apportées

Difficulté	**Solution**
Gestion des tâches finies avec un fichier de rapport	Mise en place d'un système d'écriture conditionnelle pour éviter les doublons
Envoi de notifications via Discord	Utilisation des Webhooks et formatage des messages

Respect du modèle hexagonal	Création de ports et adaptateurs distincts pour chaque service
Gestion des statuts des tâches	Implémentation d'un système de transition d'état avec suivi des modifications

5. Perspectives d'Amélioration

5.1 Fonctionnalités Futures

- **Ajout de nouvelles stratégies Pomodoro** : permettre aux utilisateurs de personnaliser leurs cycles de travail.
- **Amélioration de la gamification** : introduction d'éléments supplémentaires comme des défis ou des badges.
- **Application mobile** : développement d'une version mobile pour permettre une gestion des tâches en mobilité.
- **Tableau Kanban** : ajout d'une vue visuelle des tâches en cours, à faire et terminées.

Perspective : Ajout d'une IA pour prioriser les tâches

L'ajout d'une IA ou d'un assistant virtuel permettrait d'optimiser la gestion des tâches en fonction des habitudes de l'utilisateur. Quelques fonctionnalités possibles :

- **Analyse du comportement** : identification des tendances dans l'exécution des tâches (ex : priorisation automatique des tâches souvent reportées).
- **Recommandations intelligentes** : proposition de l'ordre optimal pour exécuter les tâches en fonction de leur complexité et des périodes de concentration maximales de l'utilisateur.
- **Prédictions et rappels personnalisés** : envoi d'alertes basées sur l'historique et le taux d'achèvement des tâches.

L'implémentation pourrait utiliser un modèle de Machine Learning entraîné sur les données des utilisateurs avec TensorFlow ou scikit-learn.

Perspective : implémentation d'une gestion collaborative

Une amélioration clé serait la possibilité de partager et de gérer des tâches en groupe :

- **Création de groupes de travail** : les utilisateurs pourraient assigner des tâches à des membres spécifiques.

- **Système de permissions** : définition de rôles (administrateur, contributeur, observateur) pour éviter les modifications non désirées.
- **Vue collaborative** : ajout d'un tableau Kanban pour visualiser la répartition des tâches en équipe.
- **Notifications partagées** : rappels envoyés aux membres concernés lorsqu'une tâche doit être complétée.

Cette évolution permettrait à l'application de s'adresser aux entreprises et aux équipes de projet, élargissant ainsi sa base d'utilisateurs.

5.2 Optimisation Technique

- **Amélioration des performances** : réduction des requêtes SQL redondantes.
- **Sécurité** : renforcement du chiffrement des données et de l'authentification.
- **Modularité** : faciliter l'ajout de nouveaux services via l'architecture hexagonale.