**Guide Technique PlanAthena - Documentation de Référence**

**🎯 Objectif**

**Cette documentation sert de référence unique pour les LLMs (Claude/AI Studio) afin d'éviter les questions répétitives et maintenir la cohérence architecturale.**

**📋 Architecture Actuelle (v0.3.9)**

**Vue d'ensemble métier**

**Processus Unifié: Qualification → Planification → Suivi**

**- Qualification des tâches (durées, contraintes, dépendances)**

**- Planification optimale (solveur Or-Tools via PlanAthena.Core)**

**- Suivi de chantier (retards, re-planification)**

**Structure des couches**

1. **IHM (WinForms)**
2. **Services Métier (Orchestration + logique)**
3. **Accès Données (Persistence + utilitaires)**

**🏗️ Concepts Métier Clés**

**Projet**

* **Définition : Conteneur principal de tous les éléments**
* **Responsabilité : ProjetService (gestion centralisée)**

**Lot**

* **Définition : Phase de chantier (GrosOeuvre → SecondOeuvre → Finitions)**
* **Contraintes : Précédence dure entre lots**
* **Responsabilité : ProjetService (vision macro)**

**Bloc**

* **Définition : Espace physique (pièce) ou logique (séquence de tâches)**
* **Contraintes :** 
  + **Dépendances entre tâches UNIQUEMENT intra-bloc**
  + **Capacité max ouvriers (découpage automatique des tâches longues)**
  + **Simplification visuelle (zoom dans les blocs)**
* **Responsabilité : BlocService (couplé aux tâches)**
* **Usage : BlocForm appelé 90% du temps depuis TacheForm**

**Tâche**

* **Définition : Unité de travail élémentaire**
* **Propriétés : durée, métier, dépendances, bloc parent**
* **Contraintes : Les dépendances ne s'exercent qu'intra-bloc**
* **Responsabilité : TacheService**

**Métier**

* **Définition : Compétence technique (Plaquiste, Électricien, etc.)**
* **Propriétés : prérequis métier, pictogramme, phases de chantier**
* **Responsabilité : Fusionné dans ProjetService (v0.3.8)**

**Ouvrier**

* **Définition : Ressource humaine avec compétences**
* **Responsabilité : OuvrierService**

**🔧 Services Principaux**

**ProjetService**

**Statut : Service principal post-fusion MetierService (v0.3.8)**

**// Gestion projet**

**+CreerNouveauProjet()**

**+SauvegarderProjet()**

**+ChargerProjet()**

**// Gestion métiers (fusionné v0.3.8)**

**+AjouterMetier()**

**+GetAllMetiers()**

**+ObtenirMetiersTriesParDependance()**

**// Gestion lots (à fusionner prochainement)**

**+AjouterLot()**

**+GererPrecedenceLots()**

**TacheService**

**Statut : Service spécialisé, reste séparé**

**// Qualification des tâches**

**+AjouterTache()**

**+ModifierTache()**

**+ObtenirTachesParLot()**

**// Import/Export**

**+ImporterTachesCSV()**

**BlocService**

**Statut : À évaluer pour fusion dans TacheService**

**+AjouterBloc()**

**+ModifierBloc()**

**+ValiderCapaciteOuvriers()**

**PlanificationService**

**Statut : Service spécialisé, reste séparé**

**+LancerPlanificationAsync()**

**+PreparerPourSolveur()**

**+ConsoliderPourGantt()**

**🔗 Dépendances Critiques**

**DependanceBuilder**

* **Rôle : Gestion des suggestions de dépendances entre tâches**
* **Contraintes : Anti-cycles, intra-bloc uniquement**
* **Algorithme : Graph First + remontée chaîne prérequis métier**
* **Usage : Appelé depuis TacheDetailForm**

**Flux de création de tâche**

**User → TacheForm → TacheDetailForm → DependanceBuilder → Suggestions**

**→ TacheService → Persistence**

**📊 Règles Métier Importantes**

**Contraintes de dépendances**

1. **Intra-bloc : Les dépendances ne traversent JAMAIS les blocs**
2. **Anti-cycles : Vérification graphe pour éviter les boucles**
3. **Suggestions métier : Basées sur prérequis et exclusions utilisateur**

**Contraintes de lots**

1. **Précédence dure : GrosOeuvre avant SecondOeuvre**
2. **Pas de dépendances : Entre blocs d'un même lot**

**Contraintes de blocs**

1. **Capacité ouvriers : Découpage automatique si tâche > capacité**
2. **Espace physique/logique : Deux types de blocs supportés**

**🚨 Points d'Attention Techniques**

**Gestion des dépendances circulaires**

* **ProjetService et TacheService s'appellent mutuellement**
* **Solution actuelle : Injection par Func<T> pour éviter les cycles**

**Import CSV**

* **Service : ImportOrchestrationService**
* **Configuration : ImportMappingConfiguration**
* **Gestion des exclusions : Support des ExclusionsDependances**

**Solveur Or-Tools**

* **Interface : PlanAthena.Core.Facade**
* **Préparation : DataTransformer pour conversion DTO**
* **Limitation : 1 ouvrier max par tâche (découpage automatique)**

**📝 Historique des Refactorings**

**v0.3.8 - Fusion MetierService**

* **Action : MetierService fusionné dans ProjetService**
* **Raison : Simplification architecture + centralisation projet**

**v0.3.9 - Prochaine étape**

* **Prévu : Fusion LotService dans ProjetService**
* **À évaluer : Fusion BlocService dans TacheService**

**💡 Guidelines pour LLMs**

**Avant modification de code**

1. **Vérifier : Quel service gère cette responsabilité ?**
2. **Respecter : Les contraintes intra-bloc pour les dépendances**
3. **Attention : Aux dépendances circulaires entre services**

**Patterns à respecter**

* **ProjetService = Vision macro (projet, lots, métiers)**
* **TacheService = Vision opérationnelle (tâches + contraintes)**
* **Services spécialisés = Logique complexe (Planification, Import)**

**Anti-patterns à éviter**

* **Dépendances entre blocs différents**
* **Bypass des services métier depuis l'IHM**
* **Logique métier dans les contrôles UI**

**🔄 Prochaines Optimisations Identifiées**

**Organisation des DTOs**

**Problème actuel : DTOs mélangés avec la logique métier dans ProjetService.cs Solution proposée :**

**/Services/Business/DTOs/**

**├── ProjetDTOs.cs (InformationsProjet, ValidationProjet, ResumeProjet)**

**├── ImportDTOs.cs (ResumeImport - à déplacer depuis DataAccess)**

**├── StatistiquesDTOs.cs (StatistiquesOuvriers, StatistiquesTaches, etc.)**

**└── ExceptionsDTOs.cs (ProjetException)**

**Simplification ProjetService**

**Actions à mener :**

1. **Extraire les DTOs vers dossier dédié**
2. **Séparer les constantes couleurs vers un service dédié**
3. **Évaluer fusion LotService → ProjetService**

**TypeEspaceBloc**

**Note métier : Distinction Espace Physique/Logique sans impact technique**

* **Espace Physique : pièce, zone délimitée**
* **Espace Logique : séquence de tâches (ex: Enduit→Ponçage→Peinture)**

**📁 Structure Actuelle des Fichiers**

**/Services/**

**├── Business/ → Services Métier (ProjetService, TacheService, etc.)**

**├── DataAccess/ → Persistence & I/O (CsvDataService, ImportService)**

**└── Processing/ → Transformations & Calculs spécialisés**

**/Data/ → Entités métier core**

**/Utilities/ → Helpers transversaux**

**/Controls/ → Contrôles UI spécialisés**

**/Forms/ → Interface utilisateur**

**Nugets identifiés :**

* **QuikGraph (graphes & tri topologique)**
* **CsvHelper (import/export CSV)**
* **Microsoft.Msagl (diagrammes)**
* **Google.OrTools (solveur via PlanAthena.Core)**
* **FluentValidation (validations)**

***Cette documentation doit être mise à jour à chaque évolution architecturale majeure.***

**📊 Matrice de Dépendances - PlanAthena v0.3.9**

**🎯 Post-Fusion LotService → ProjetService**

**📋 Légende**

* ✅ **Dépendance directe** (injection dans constructeur)
* 🔄 **Factory** (Func<T> pour éviter cycles)
* 📞 **Utilise méthodes** (sans injection)
* ❌ **Aucune dépendance**

**🏗️ Services Business Core**

| **Service → Dépend de ↓** | **ProjetService** | **TacheService** | **BlocService** | **OuvrierService** |
| --- | --- | --- | --- | --- |
| **ProjetService** | ❌ | 🔄 Func<T> | 🔄 Func<T> | ✅ Direct |
| **TacheService** | 🔄 Func<T> | ❌ | 🔄 Func<T> | ❌ |
| **BlocService** | ❌ | 🔄 Func<T> | ❌ | ❌ |
| **OuvrierService** | ❌ | ❌ | ❌ | ❌ |

**📝 Méthodes utilisées via Factories**

* **TacheService** → **ProjetService** : ObtenirLotParId(), ObtenirTousLesLots()
* **BlocService** → **TacheService** : ObtenirTachesParBloc(), ValiderDependances()
* **ProjetService** → **TacheService** : ObtenirStatistiques(), Vider()
* **ProjetService** → **BlocService** : ObtenirTousLesBlocs(), Vider()

**🗄️ Services Data & Processing**

| **Service → Dépend de ↓** | **ProjetService** | **TacheService** | **BlocService** | **CsvDataService** | **ExcelReader** |
| --- | --- | --- | --- | --- | --- |
| **ImportService** | ✅ lots | ✅ Direct | ✅ Direct | ❌ | ❌ |
| **IdGeneratorService** | ✅ lots | ✅ Direct | ✅ Direct | ❌ | ❌ |
| **DataTransformer** | ✅ lots | 📞 Utilise | 📞 Utilise | ❌ | ❌ |
| **DependanceBuilder** | ✅ lots | 📞 Utilise | ❌ | ❌ | ❌ |
| **CsvDataService** | ❌ | ❌ | ❌ | ❌ | ❌ |
| **ExcelReader** | ❌ | ❌ | ❌ | ❌ | ❌ |

**📝 Méthodes ProjetService utilisées pour les lots**

* **ImportService** : AjouterLot(), ObtenirLotParId()
* **IdGeneratorService** : ObtenirTousLesLots()
* **DataTransformer** : ObtenirTousLesLots()
* **DependanceBuilder** : ObtenirLotParId(), ObtenirTousLesLots()

**🖥️ Services UI & Controls**

| **UI Component → Dépend de ↓** | **ProjetService** | **TacheService** | **BlocService** | **OuvrierService** |
| --- | --- | --- | --- | --- |
| **MainForm** | ✅ Direct | ✅ Direct | ✅ Direct | ✅ Direct |
| **LotForm** | ✅ lots | ✅ validation | ❌ | ❌ |
| **TacheForm** | ✅ métiers+lots | ✅ Direct | ✅ Direct | ❌ |
| **OuvrierForm** | ✅ métiers | ❌ | ❌ | ✅ Direct |
| **MetierForm** | ✅ métiers | ❌ | ❌ | ❌ |
| **BlocForm** | ❌ | ✅ validation | ✅ Direct | ❌ |
| **PertDiagramControl** | ✅ lots | 📞 Utilise | 📞 Utilise | ❌ |

**📝 Méthodes ProjetService utilisées par l'UI**

* **LotForm** : AjouterLot(), ModifierLot(), SupprimerLot(), ObtenirTousLesLots()
* **TacheForm** : GetAllMetiers(), ObtenirTousLesLots()
* **OuvrierForm** : GetAllMetiers()
* **MetierForm** : AjouterMetier(), ModifierMetier(), SupprimerMetier(), GetAllMetiers()
* **PertDiagramControl** : ObtenirTousLesLots()

**🔄 Cycles Évités par les Factories**

**❌ Cycles qui existaient avant les Func<T>**

TacheService ←→ ProjetService (accès aux lots)

BlocService ←→ TacheService (validation tâches)

ProjetService ←→ TacheService (statistiques)

**✅ Solution avec Factories**

TacheService → Func<ProjetService> (lazy loading)

BlocService → Func<TacheService> (lazy loading)

ProjetService → Func<TacheService> + Func<BlocService> (lazy loading)

**📈 Impact de la Fusion LotService**

**🔥 Avant la fusion (problématique)**

ImportService(TacheService, ProjetService, BlocService, LotService, IdGenerator)

TacheForm(TacheService, ProjetService, DependanceBuilder, LotService, BlocService)

DataTransformer(ProjetService, LotService, TacheService)

**✅ Après la fusion (simplifié)**

ImportService(TacheService, ProjetService, BlocService, IdGenerator)

TacheForm(TacheService, ProjetService, DependanceBuilder, BlocService)

DataTransformer(ProjetService, TacheService)

**📊 Statistiques de la fusion**

* **Services impactés** : 12 fichiers
* **Références modifiées** : 48 occurrences
* **Constructeurs simplifiés** : 8 classes
* **Dépendances éliminées** : 1 service complet (LotService)
* **Complexité réduite** : -20% de paramètres d'injection

**🎯 Recommandations pour futurs refactorings**

**✅ Patterns à maintenir**

1. **Factory Pattern** pour éviter les cycles
2. **Service unique** pour les entités liées (ex: Projet+Métiers+Lots)
3. **Services feuilles** sans dépendances (CsvData, ExcelReader, Ouvrier)

**⚠️ Points d'attention**

1. **TacheService** est très connecté → surveiller sa complexité
2. **ProjetService** devient central → éviter qu'il devienne trop gros
3. **Factory resolution** → tester les performances si beaucoup d'appels

**📋 Prochains candidats de fusion potentiels**

* **BlocService** → **TacheService** ? (si logique métier proche)
* **IdGeneratorService** → **Service centralisé** ? (si d'autres générateurs)
* **ImportService** → **TacheService** ? (si import devient une fonction core)

**📝 Notes d'Architecture**

🎉 **Succès de la fusion LotService** : Architecture plus cohérente, maintenance simplifiée, et zero régression fonctionnelle. Le pattern Factory s'avère efficace pour éviter les cycles dans une architecture complexe.

🔧 **Leçon apprise** : Les diagrammes d'architecture détaillés sont essentiels pour anticiper l'impact des refactorings sur les 48 références dans 12 fichiers.