AGC Glass Europe

Flexi Task

Système de gestion de tâches logistique

Analyse technique

Délivrable du 5 décembre 2025

Nicolas Stoupy
05/12/2025

Introduction

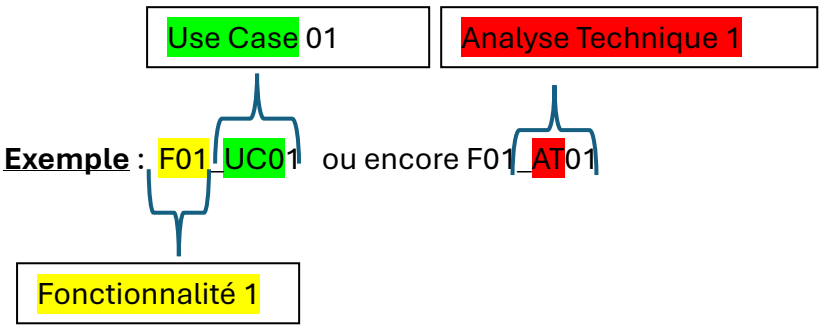
Ce document présente l'analyse technique du système de gestion de tâches logistique « Flexi Task » développé pour AGC Glass Europe. Il détaille les principales fonctionnalités de l'application, telles que l'authentification des utilisateurs, la gestion des zones de travail et le séquencement des tâches, tout en comparant différents modèles de structuration des données (sous-typage, JSON, EAV) afin de justifier le choix d'une approche hybride conciliant flexibilité, performance et maintenabilité. Enfin, l'architecture proposée s'appuie sur les principes DDD et CQRS, intègre ASP.NET Core Identity et Blazor Server, et prévoit une connexion avec SAP pour garantir la synchronisation des données logistiques.

Table des matières

Avant-propos

Normalisation

Pour maintenir un fil conducteur entre le rapport et l'ensemble des analyses que j'ai menées, chaque étude de fonctionnalité fera référence à la documentation dans le bookstack via l'identification technique.
https://bookstackesa.be/shelves/flexitask



Gestion des versions de l'analyse

L'ensemble des documents utilisé son versionné sur un git :

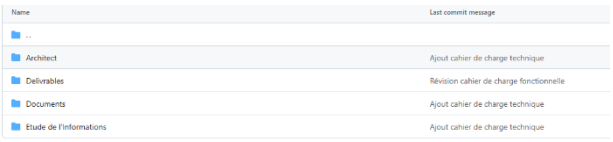
https://github.com/NicolasStoupy/FlexiTask

Gestion des versions et organisation des documents

Chaque modification apportée aux documents d'analyse est systématiquement enregistrée par un commit, ce qui garantit la traçabilité de l'évolution des fichiers. Ainsi, l'historique complet des changements reste accessible à tout moment pour consultation ou restauration.

L'organisation des documents repose sur une arborescence.

Cette arborescence se compose des dossiers suivants :



- Architect : Ce dossier contient les fichiers relatifs au projet sous Architect Sparks.
- Délivrables : Dans ce dossier sont rassemblés les documents répondant à un ou plusieurs livrables spécifiques.
- Documents : Ce répertoire regroupe l'ensemble des documents opérationnels.
- Étude de l'information : Ce dossier est consacré aux schémas conceptuels et physiques ainsi qu'aux scripts de création, constituant la base de l'analyse de l'information.

Liens et annexes

Chaque image du document permet d'accéder aux annexes en un clic.

Business Process Modeling

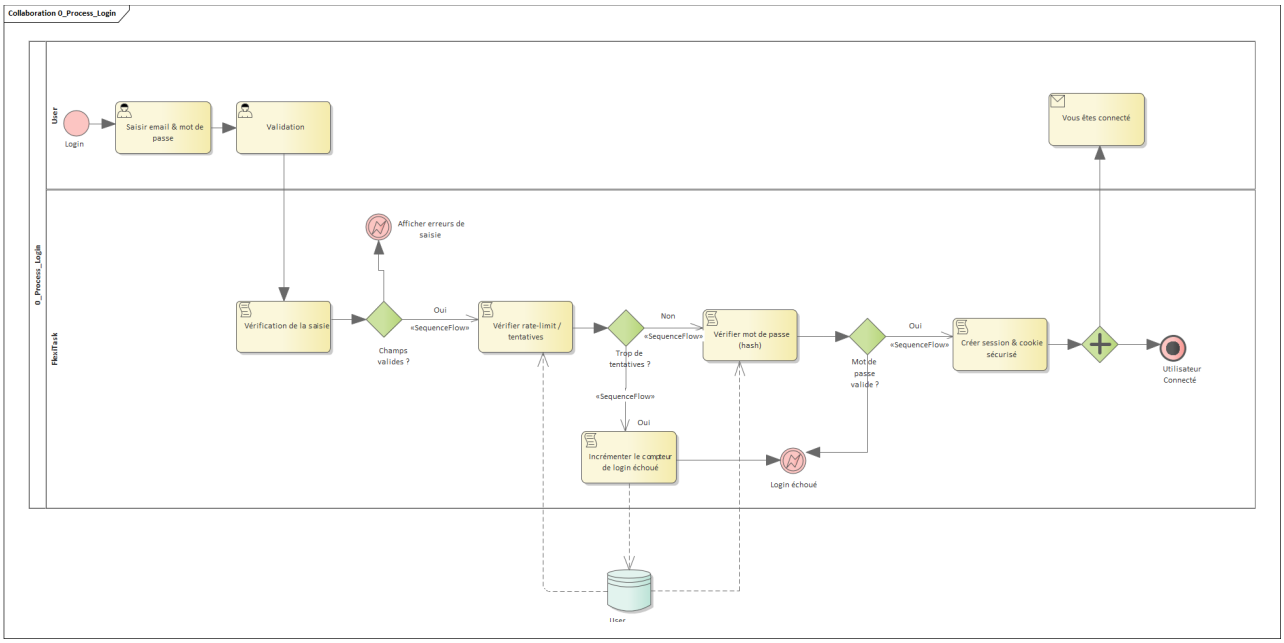
Je vais représenter ici l'ensemble des process modeling dans le scope du poc en étant légèrement plus large, les tâches qui devront être exécutées sont dépendent d'un moteur d'exécution de tâches que je défini dans la F03_UC02

Fonctionnalités :

F01 Login Authentification

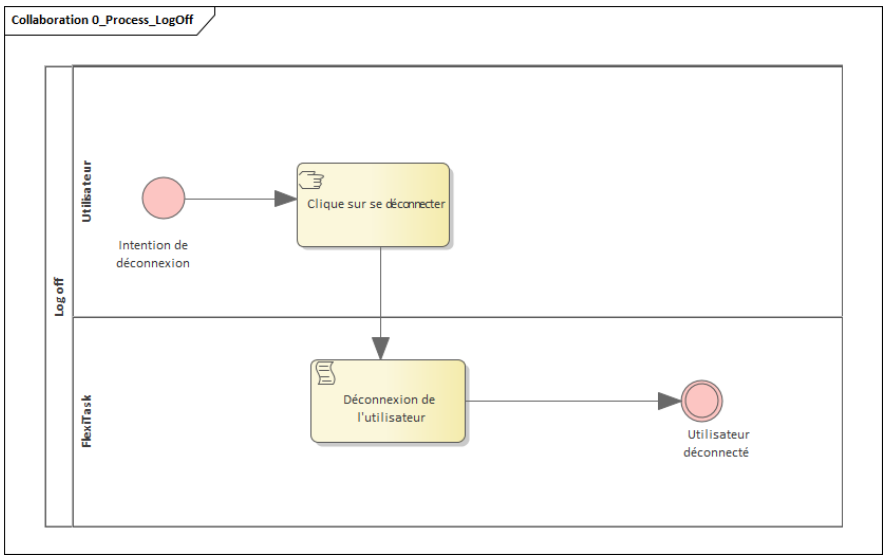
La fonctionnalité Login & Authentification permet d'assurer que seuls les utilisateurs autorisés puissent accéder à l'application en vérifiant leur identité via un nom d'utilisateur et un mot de passe, garantissant ainsi la sécurité et la confidentialité des données.

F01_UC00 : Login



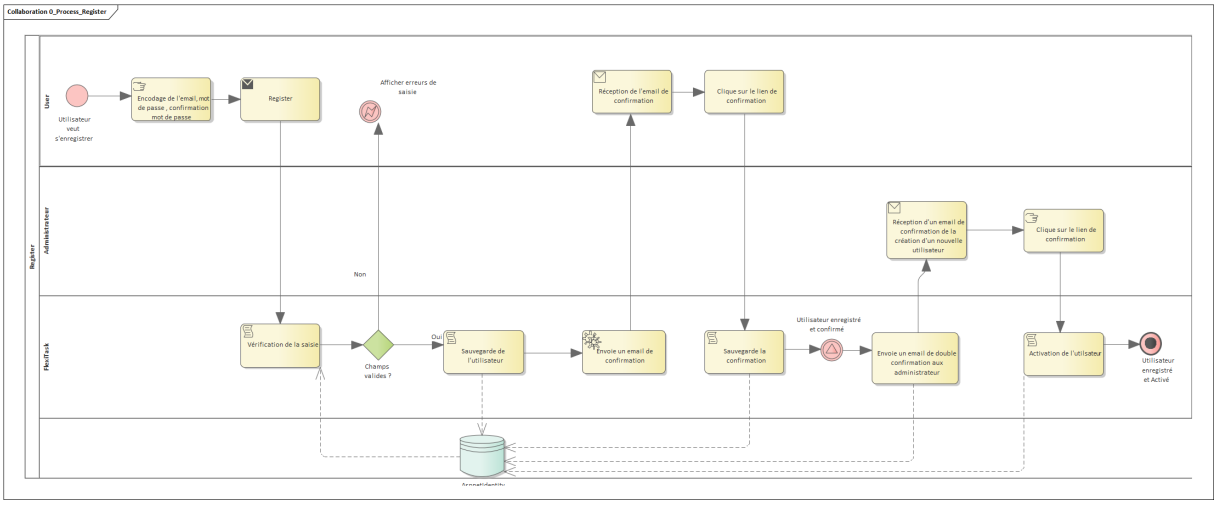
F01_ UC01 : Log off

La fonctionnalité log off intervient lorsque l'utilisateur souhaite quitter l'application ou mettre fin à sa session en cours. Elle garantit que l'accès à l'interface ne reste pas ouvert sur un poste laissé sans surveillance, ce qui pourrait entraîner un accès non autorisé à des données sensibles.



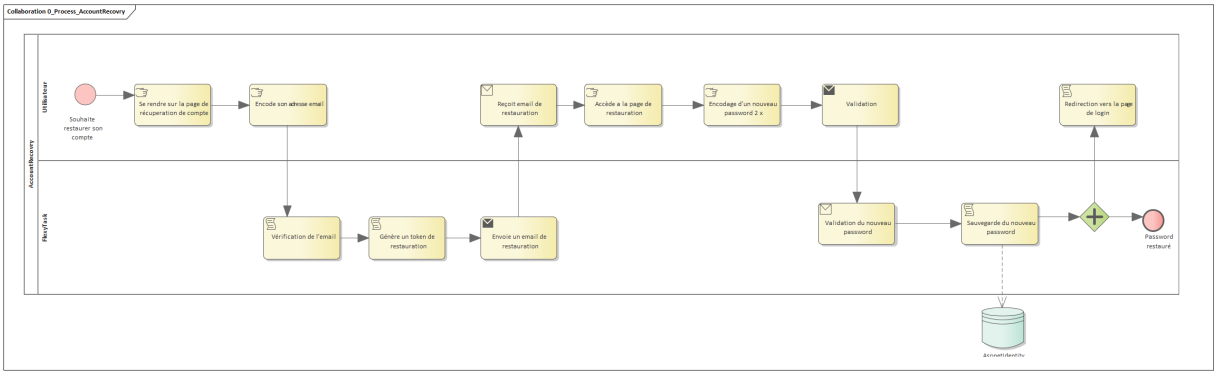
F01_ UC02 : Registrer

La fonction « Registrer » permet aux nouveaux utilisateurs de créer un compte et de le valider par e-mail de manière à vérifier leur identité et ainsi limiter l'accès aux personnes autorisées.



F01_ UC02 : Password recovery

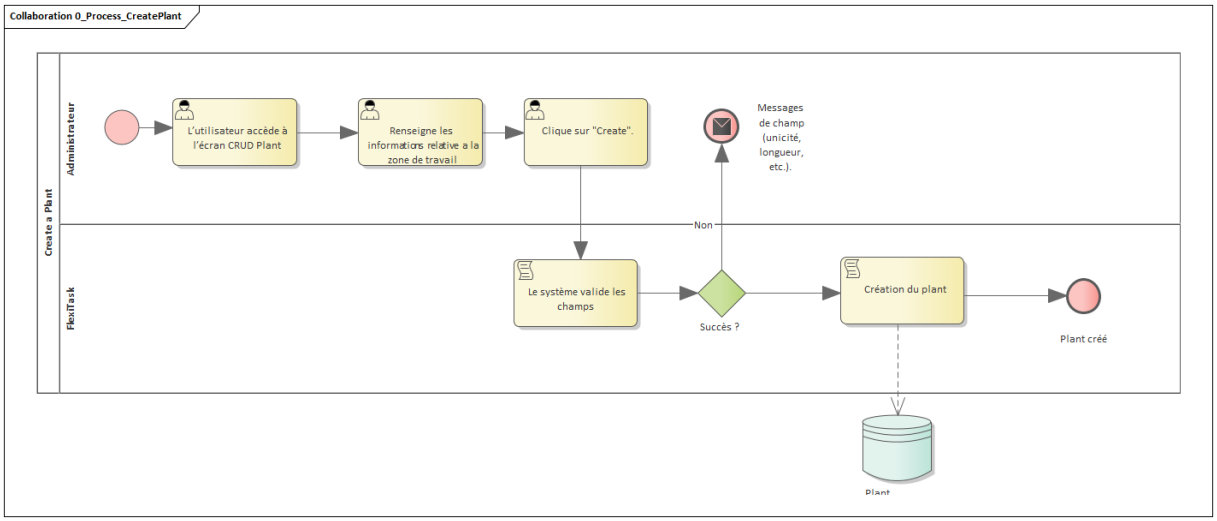
La fonctionnalité permet à un utilisateur validé de réinitialiser son mot de passe en toute autonomie.



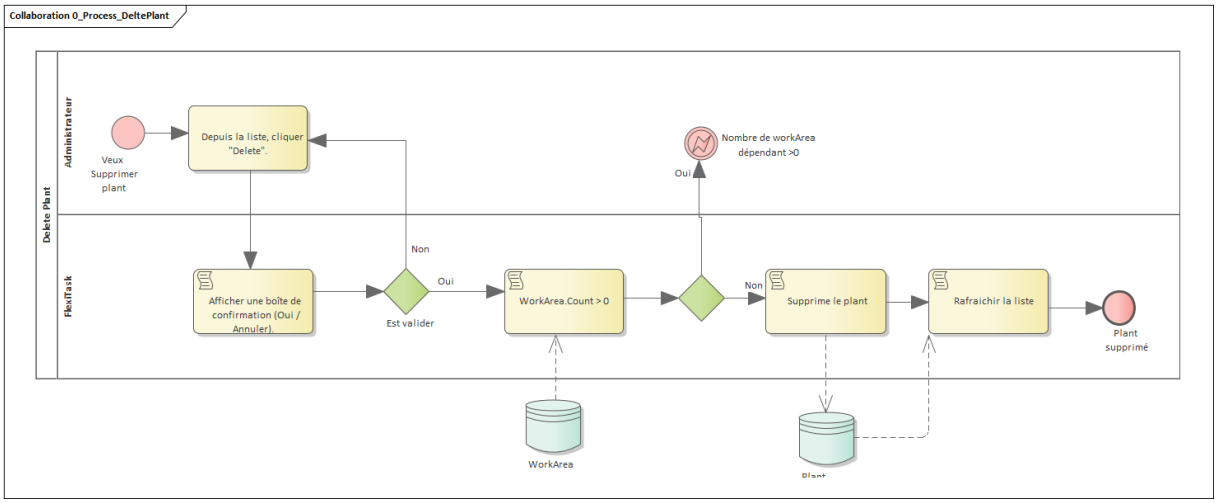
F02 Gestion des zones de travail et plant (Work Area)

Cette fonctionnalité permet de gérer toutes les entités qui représentent, sous forme informatique, les emplacements physiques à l'intérieur d'une usine, c'est-à-dire les usines ("plant") et les zones de travail ("Work Area").

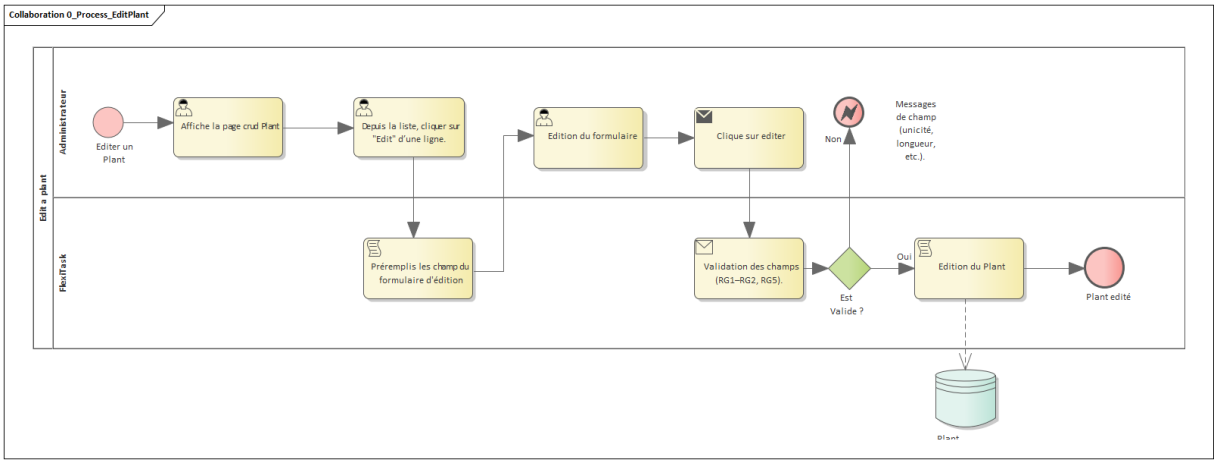
F02_UC01 : Création d'un plant



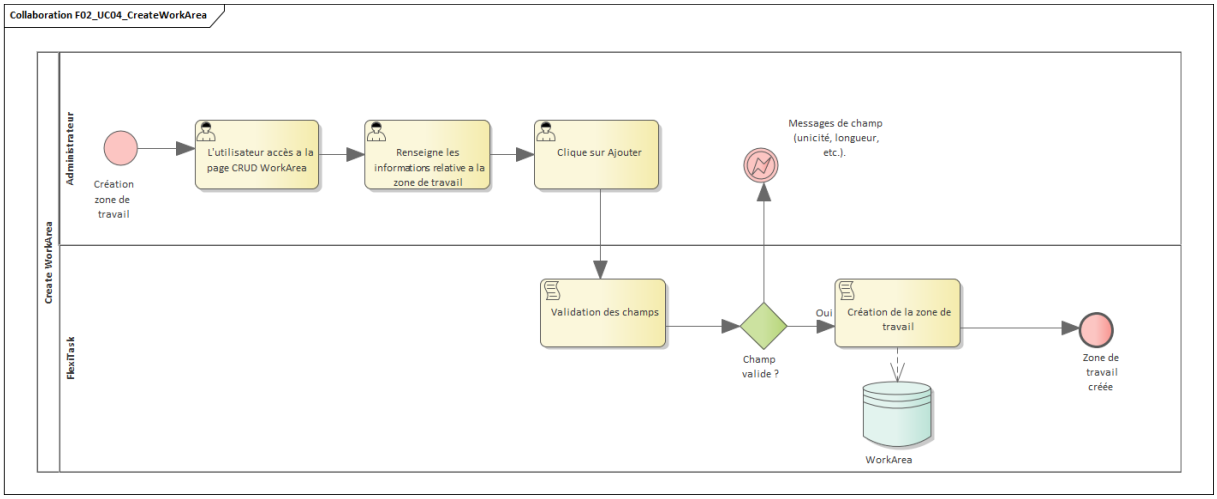
F02_UC02 : Suppression d'un plant



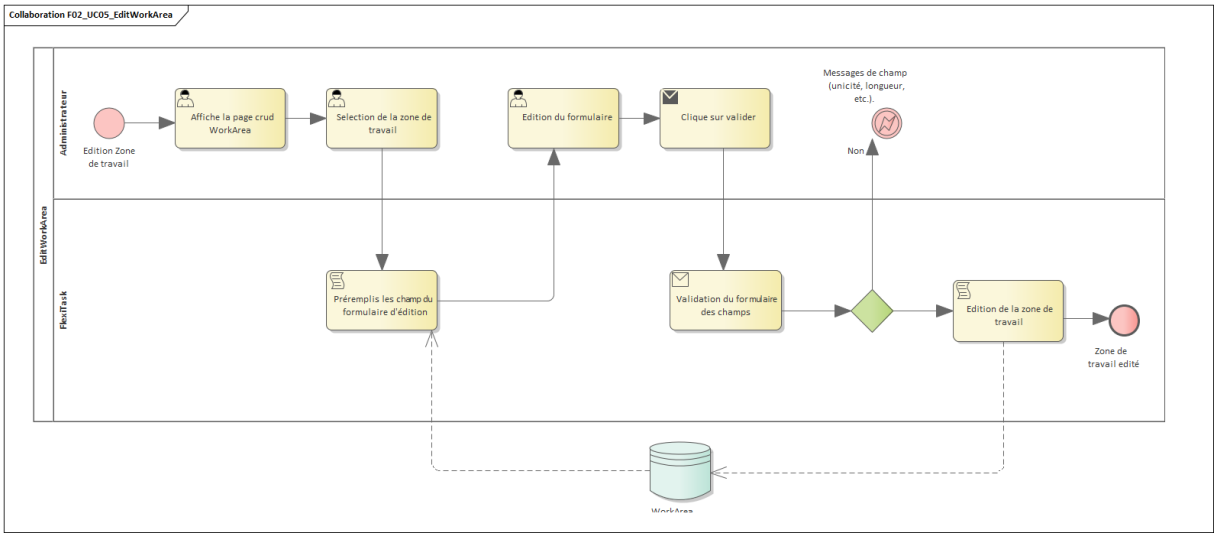
F02_UC03 : édition d'un plant



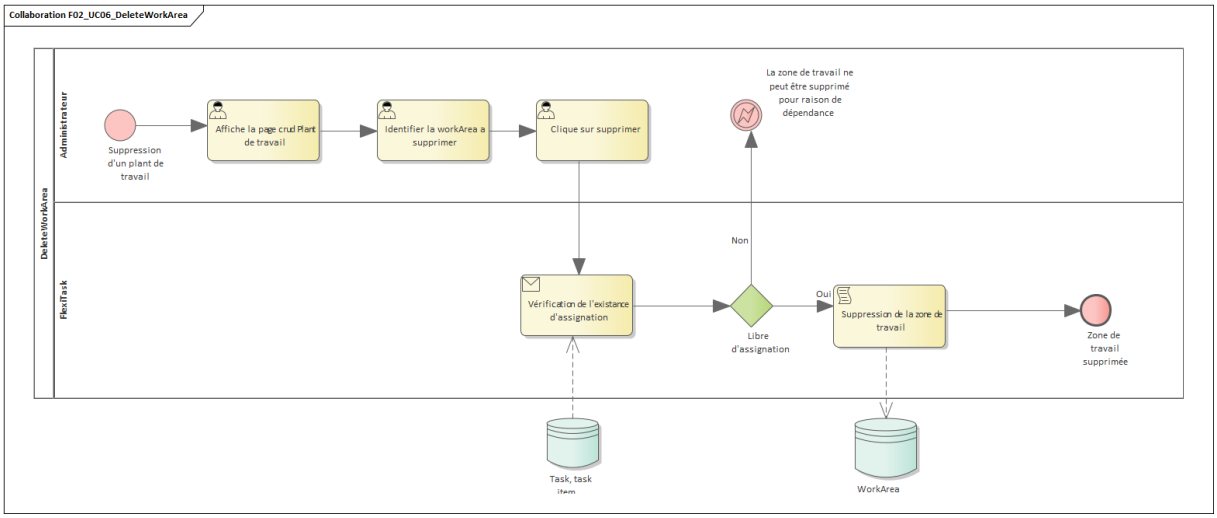
F02_UC04 : Création d'une zone de travail



F02_UC05 : Edition d'une zone de travail



F02_UC06 : Suppression d'une zone de travail



F03 Gestionnaire de tâches

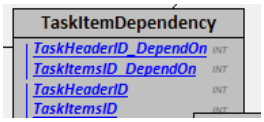
La plupart des tâches exécutées dans l'usine reposent sur des modèles de données hétérogènes, provenant de sources multiples et spécifiques aux types d'activités. Afin de simplifier leur orchestration, l'objectif est d'uniformiser la manière dont les tâches sont traitées et exécutées en les considérant comme des **porteurs** d'informations. Chaque tâche devient ainsi un conteneur d'informations utile et nécessaire à la réalisation de la tâche, et peut être exécutée selon une timeline définie, incluant des séquences, des parallélismes et des dépendances.

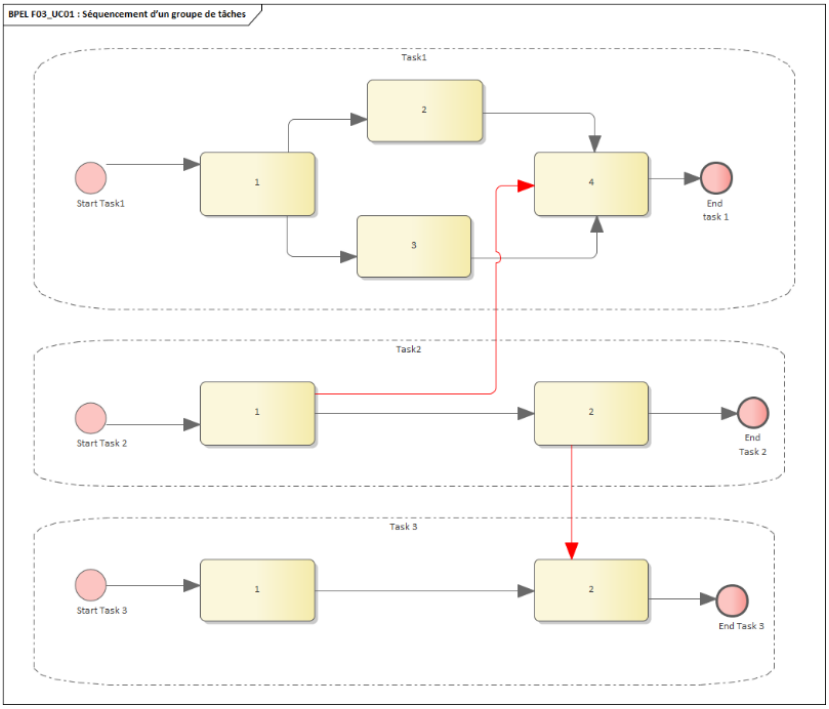
F03_UC01 : Séquencement d'un groupe de tâches

Dans un entrepôt industriel, les activités sont réparties entre plusieurs acteurs et doivent s'enchaîner selon une logique précise. Certaines tâches doivent être réalisées en série, tandis que d'autres peuvent s'exécuter en parallèle.

De plus, des dépendances existent entre tâches : aucune tâche ne peut démarrer tant que la ou les précédentes dont elle dépend n'ont pas été entièrement exécutées et validées.

Grâce à l'entité « TaskItemDependecy » nous pourrons représenter l'ensemble de ses dépendances et succession.

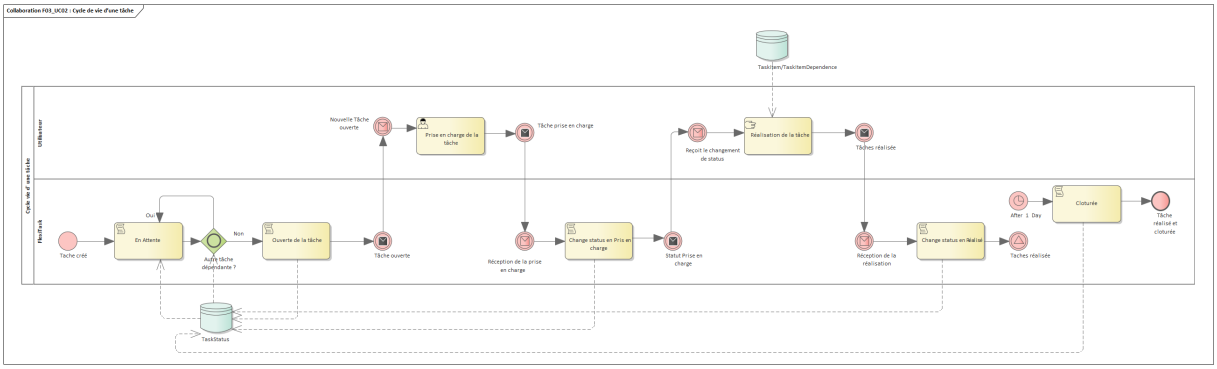




F03_UC02 : Cycle de vie d'une tâche

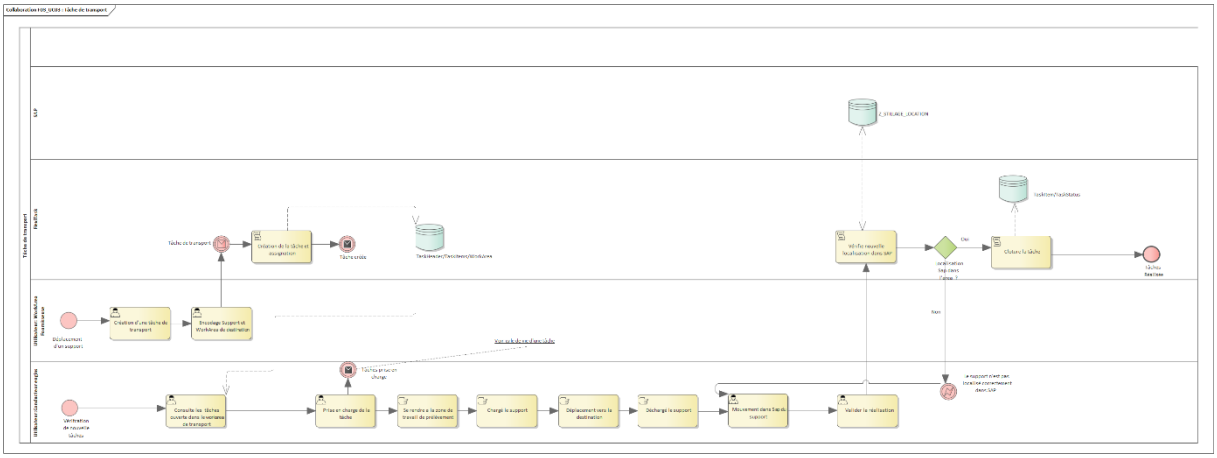
Chaque tâche passe par des états principaux comme :

Statut	Description
En attente	La tâche est liée à une dépendance elle ne peut pas être prise en charge immédiatement
Ouverte	Elle peut être exécuté
Prise en charge	La tâche est en cours d'exécution
Réalisée	La tâche est réalisée mais peut encore être modifiée
Clôturée	La tâche est clôturée et archivée
…	



F03_UC03 : Tâche de transport

Ce diagramme présente le processus d'un transport interne, de la création à la clôture de la demande. L'opérateur source crée une tâche de transport, cette tâche est prise en charge par un opérateur, effectue le déplacement et met à jour le système à chaque étape. Des contrôles automatiques (compatibilité, localisation, validations SAP) sont réalisés avant la clôture.



Modèle entité-association

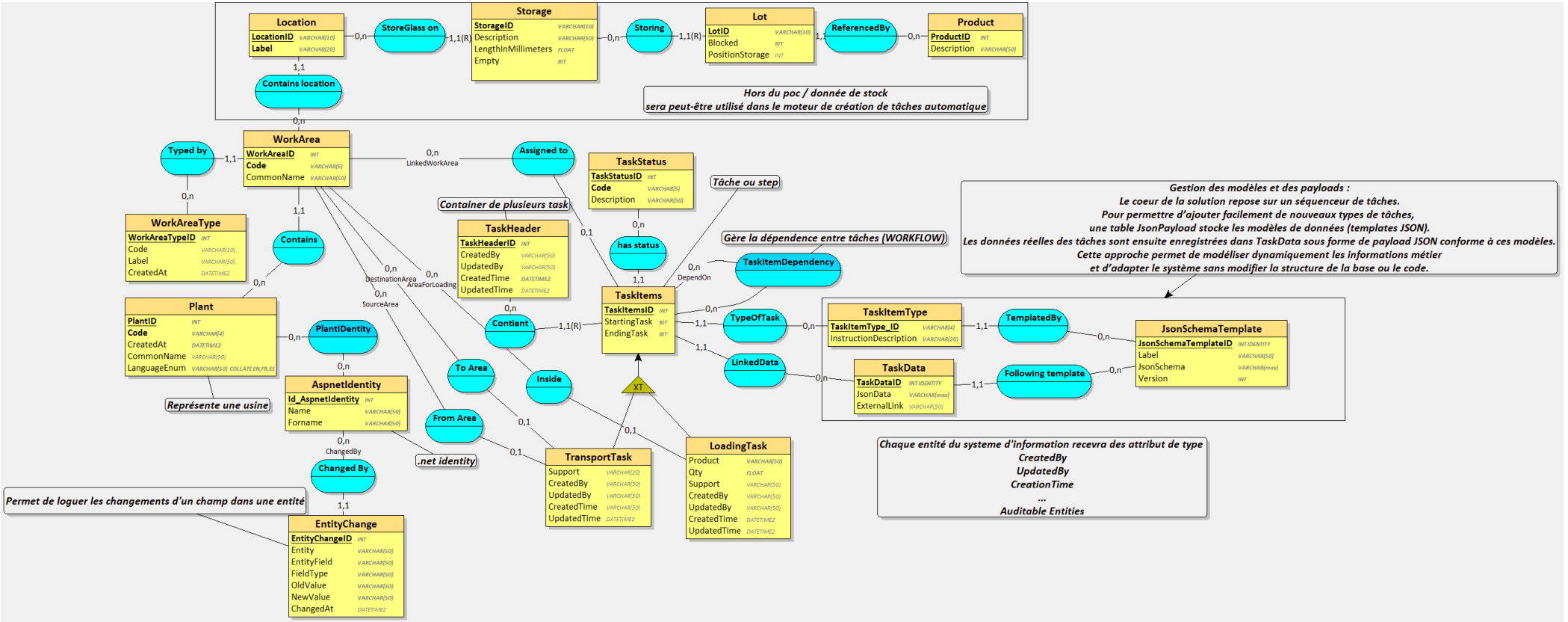
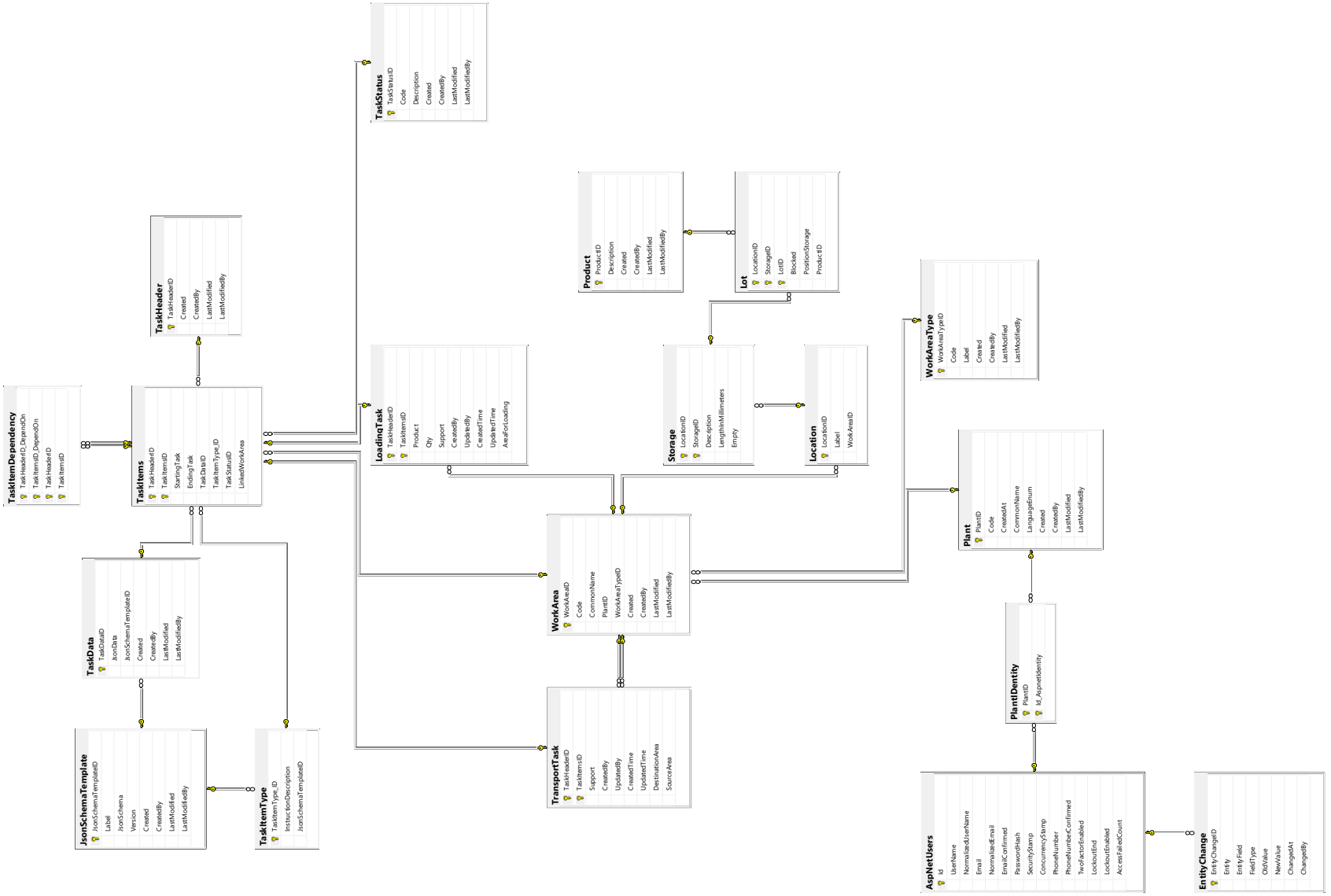


Schéma physique de la base de données



Structure des autres systèmes de persistance de l'information

SAP constitue le système principal de persistance des données opérationnelles liées au transport et à la logistique.

Il stocke et maintient toutes les informations telles que les mouvements de stock, les emplacements, les articles, les commandes, les livraisons et les statuts associés.

Flexi Task sera amené à consulter des informations comme :

- La localisation d'un support
- La localisation des lots dans le stock

Une interface dédiée assurera la communication avec SAP au moyen de RFC (Remote Function Calls), permettant de consulter ou mettre à jour les données logistiques nécessaires tout en garantissant la synchronisation entre l'application et SAP.

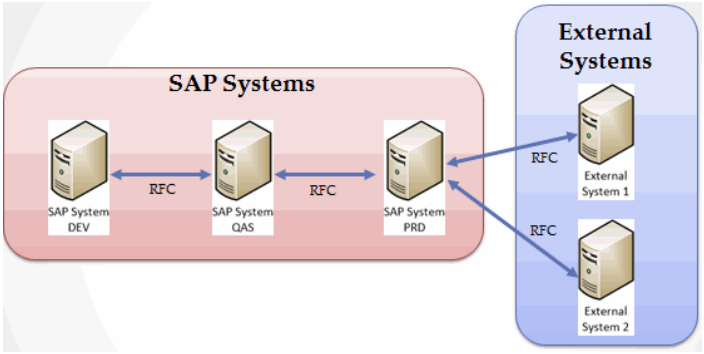
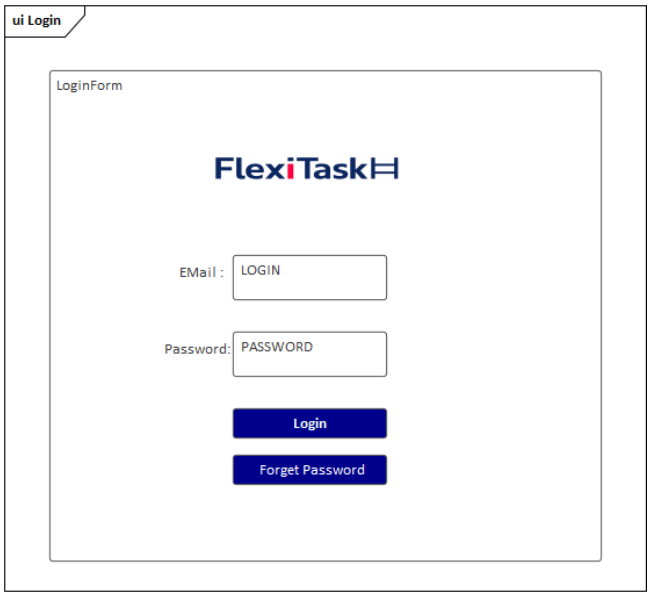


Figure 1: source http://www.sapbasisforbeginner.com/

Prototypage

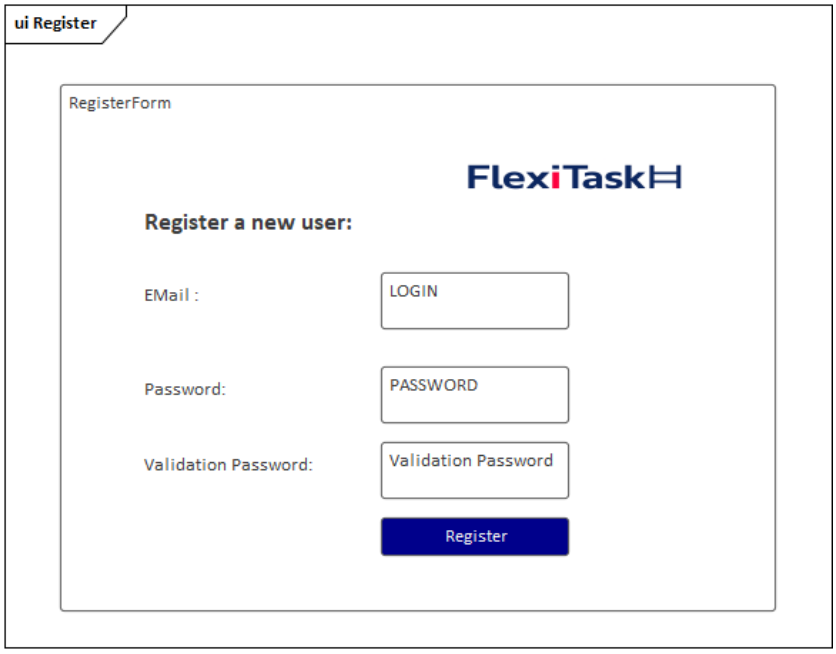
F01 Login Authentification

F01_UC00 : Login



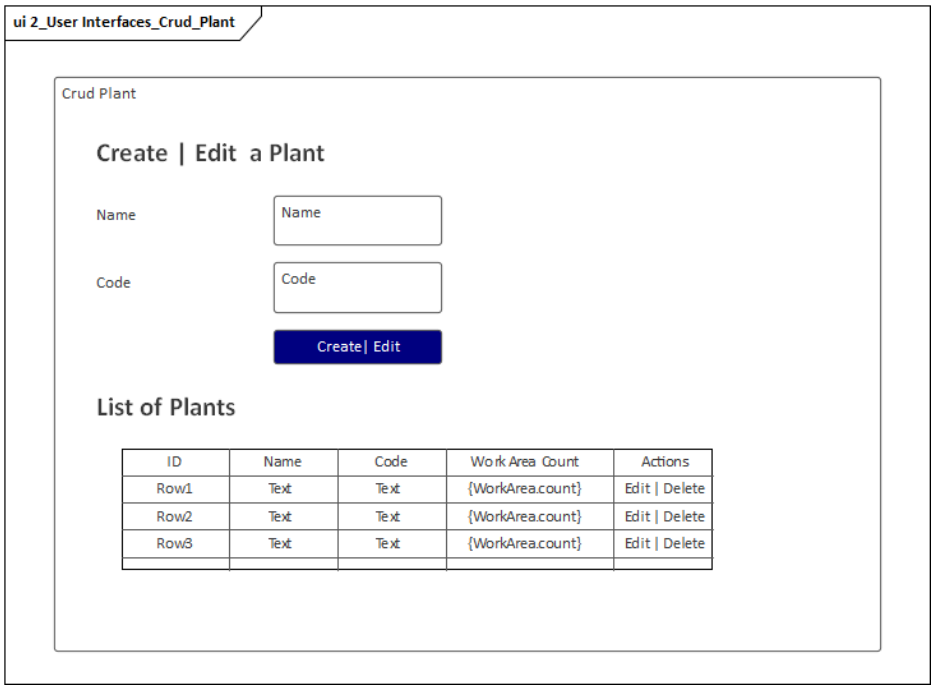
F01_ UC02 : Password recovery

F01_ UC02 : Register

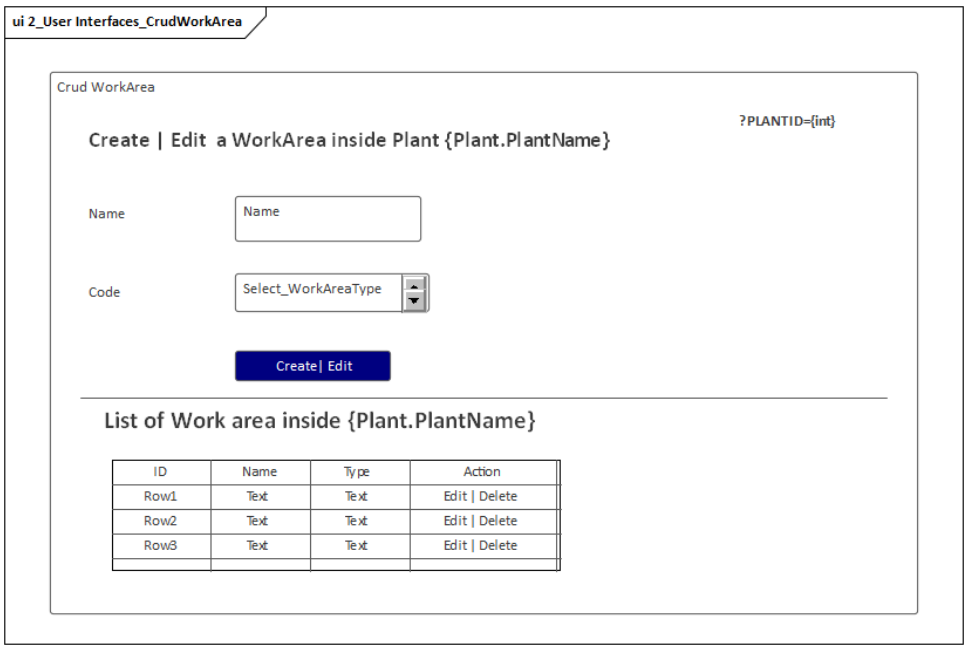


F02 Gestion des zones de travail et plant (Work Area)

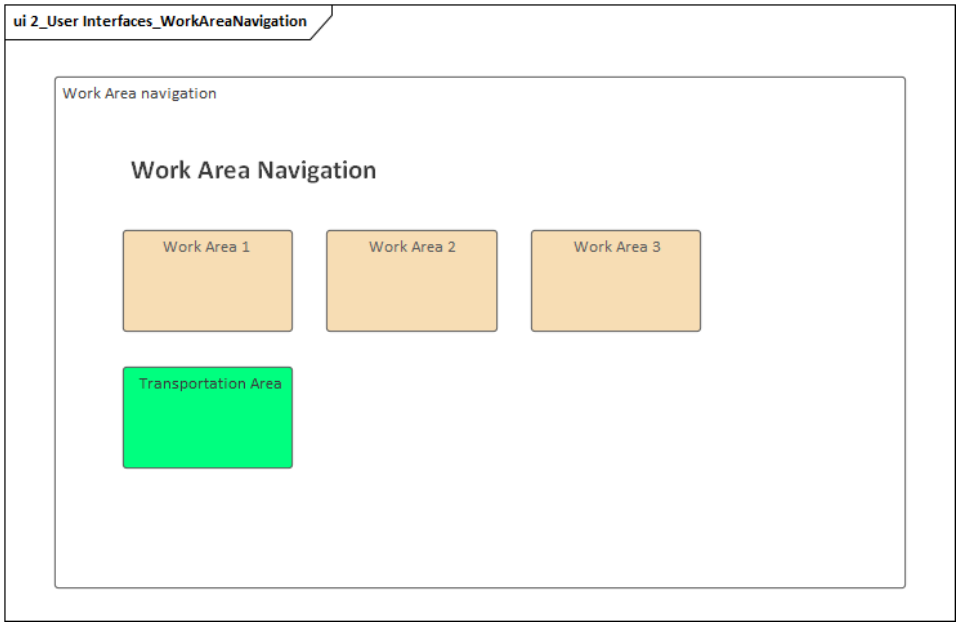
F02_UC01 : Création d'un plant & F02_UC02 : Suppression d'un plant & F02_UC03 : édition d'un plant



F02_UC04 : Création d'une zone de travail & F02_UC05 : Edition d'une zone de travail & F02_UC06 : Suppression d'une zone de travail

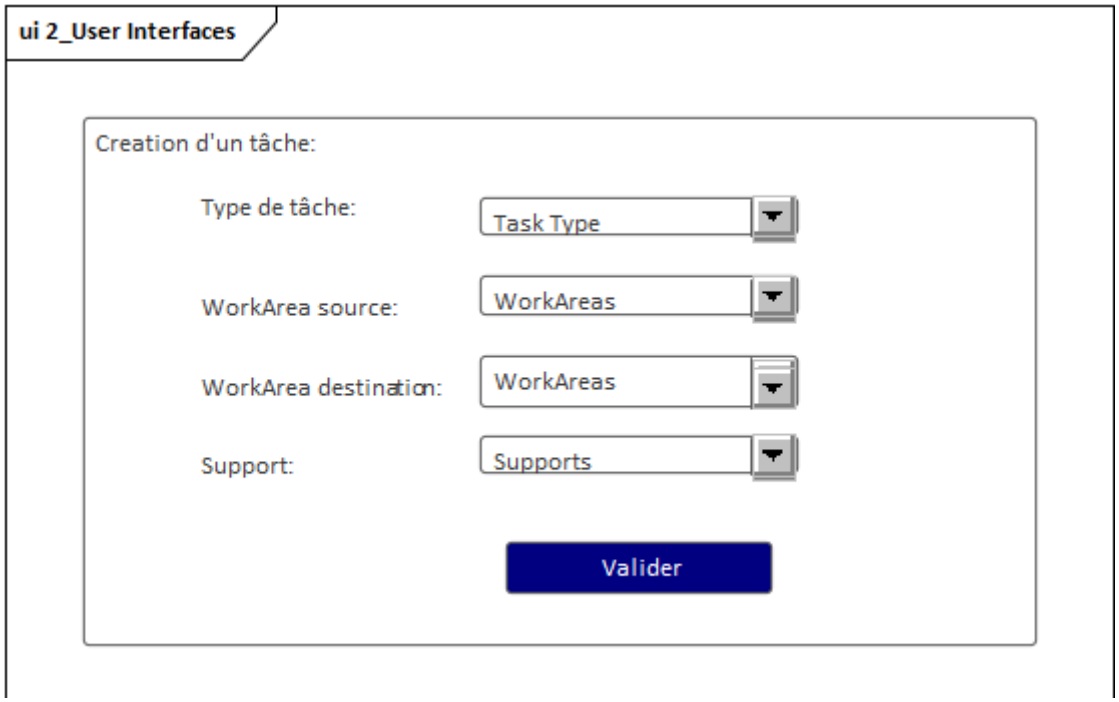


F02_UC07 : Navigation entre les Work Area

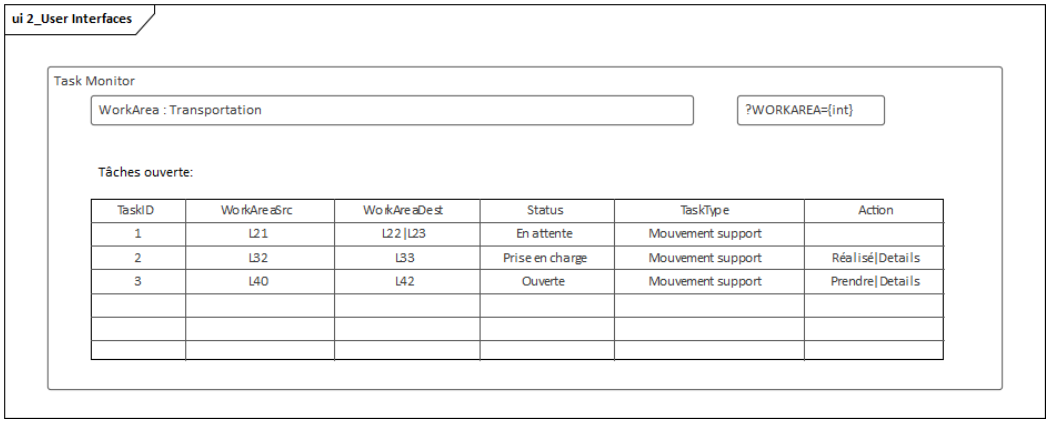


F03_UC03 : Tâche de transport

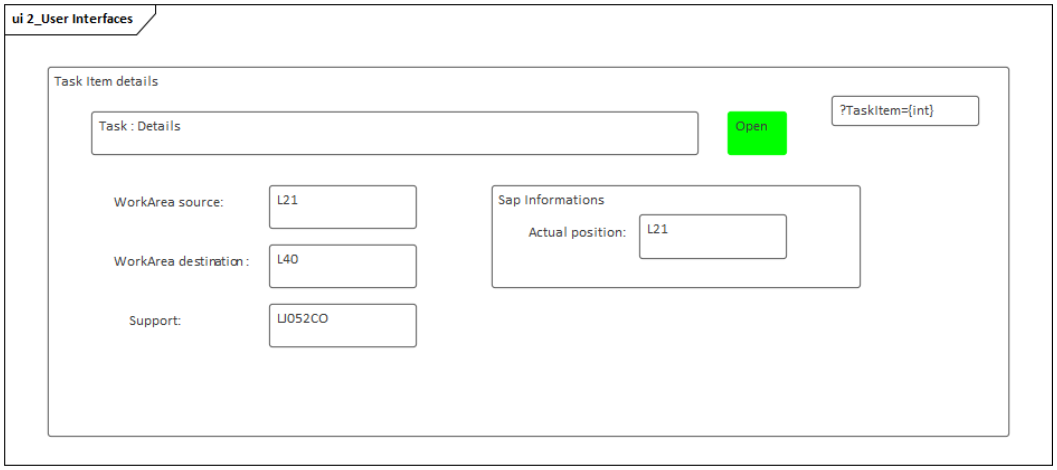
Création d'une nouvelle tâche



Moniteur des tâches dans une zone de travail



Details d’une tâche de transport



Choix techniques

F01 Login Authentification

Le choix technique ce porte sur une analyse technique disponible dans ce document , durant cette analyse j'ai mis en vis-à-vis plusieurs technologies :

- Le système d'identité asp.net
- Un système décentralisé
- Un développement maison

J'ai défini les différentes exigences techniques requise par l'application sur différent axes :

- **Sécurité** : Authentification avec gestion des rôles, politique de mot de passe, verrouillage après plusieurs échecs
- **Performance** : Authentification rapide
- **Maintenabilité** : utilisation d'un système standard permettant d'évoluer dans un environnement .net
- **Intégration** : être capable d'intervenir avec différentes technologies
- **Évolutivité** : être capable de la faire évoluer vers une utilisation centralisée (AD, Azure, ...)
- **Support / communauté** : une communauté disponible et une documentation complète

Solution retenue

ASP.NET Core Identity

Justification

Cette solution offre un bon équilibre entre sécurité, simplicité et intégration avec l'écosystème .NET. Elle reste facile à maintenir dans le temps et suffisamment flexible pour suivre l'évolution du projet.

Concrètement, la gestion des rôles devient beaucoup plus pratique : tout est centralisé, les permissions sont clairement définies et il devient très simple de préciser qui peut faire quoi dans l'application, sans complexifier le code ni l'architecture.

F03 Gestionnaire des tâches

Le gestionnaire de tâches est considéré comme un porteur d'informations, et ces informations sont hétérogènes. Le cœur de l'application repose sur l'exécution de séquences de tâches, et chaque tâche encapsule des données.

La question était donc : comment embarquer ces données dans une tâche ?

Trois solutions ont été envisagées :

- **Le sous-typage :** créer une entité spécifique pour chaque modèle de données donc par type de tâche.
- **Un champ JSON :** ajouter un champ json_data qui contient l'ensemble des informations nécessaires selon un modèle défini.
- **Modèle EAV / Propriétés dynamiques (Key/Value)**.

Option 1 – Sous-typage (un type par modèle de données)

Avantages

- Typage fort : *on sait exactement quelles données contient chaque tâche*.
- Validation plus claire : *les règles sont claires dans chaque classe*.
- Navigation & mapping plus propres : *EF Core comprend mieux les relations*.
- Requêtage SQL plus lisible : *les colonnes sont visibles dans la table*.

Inconvénients

- Explosion du nombre de classes / tables
- Chaque nouveau besoin en tâche nécessite la création d'une entité
- Rigidité à l'évolution

Option 2 – Champ JSON suivant un Template

Avantages

- Très flexible / évolutif : *changer la structure de l'information sans migration*
- Peu de modifications en base : *mettre à jour le JSON*
- Adapté aux intégrations externes

Inconvénients

- Moins de contrôle au niveau SQL
- Requêtes plus compliquées

  Solution : Créer une vue SQL qui matérialise les infos importantes

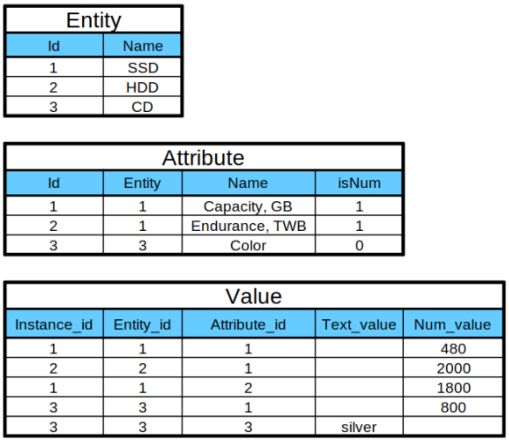
- Complexité de versioning

  Solution : Ajouter un champ version dans le JSON

Option 3 – Modèle EAV / Propriétés dynamiques (Key/Value)

Avantages :

- Très flexible : permet d'ajouter de nouvelles propriétés sans modifier le schéma ou déployer une migration.



Entity

Id	Name
1	SSD
2	HDD
3	CD

Attribute

Id	Entity	Name	isNum
1	1	Capacity, GB	1
2	1	Endurance, TWB	1
3	3	Color	0

Value

Instance_id	Entity_id	Attribute_id	Text_value	Num_value
1	1	1		480
2	2	1		2000
1	1	2		1800
3	3	1		800
3	3	3	silver	

- Adapté aux données hétérogènes : idéal lorsque les tâches utilisent des modèles de données différents.
- Économe en structure : une seule table peut gérer un grand nombre de types de données variés.
- Supporte facilement les évolutions : les règles métiers peuvent évoluer sans impacter la base.

Inconvénients :

- Complexité de requêtage : nécessite des jointures multiples ou des pivots pour reconstituer un objet complet.
- Performances variables : plus lent pour les rapports, validations complexes ou filtres sur plusieurs attributs.

Tableau comparatif

Critère	Option 1 – Modèle sous typage	Option 2 – Modèle JSON	Option 3 – Modèle EAV (Key/Value)
Flexibilité	★☆☆☆	★★★☆	★★★★
Évolution du modèle	★☆☆☆	★★★☆	★★★★
Performance	★★★★	★★★☆	★★☆☆
Complexité	★★★★ (simple)	★★★☆	★☆☆☆ (complexe)
Lisibilité des données	★★★★	★★★☆	★☆☆☆
Facilité des requêtes	★★★★	★★★☆	★☆☆☆
Adapté aux données variables	★☆☆☆	★★★☆	★★★★
Total	15	21	17

Solution retenue et justification

La solution retenue est d'adopter une approche hybride combinant json et sous typage.
Les données stables et transverses, communes à l'ensemble des tâches, sont stockées dans des colonnes typées et structurées.
À l'inverse, les données spécifiques à chaque type de tâche ou liées à des informations variables sont encapsulées dans un champ JSON.
Cette approche permet au gestionnaire de tâches de rester un porteur d'informations hétérogènes.

Retenue

À ce stade, le choix du modèle reste ouvert. Si la mise en œuvre révèle une complexité trop importante, il sera toujours possible d'adapter l'architecture, notamment en introduisant du sous-typage ou en révisant le mode de structuration des données.

N'ayant pas encore une vision totalement concrète de l'implémentation finale, je me laisse la possibilité d'ajuster ou de changer d'approche durant le développement

Choix des patterns et des technologies

Design pattern

Pour gérer la complexité liée aux tâches, à leurs dépendances et aux comportements déclenchés par les changements d'état, j'ai retenu une architecture structurée autour de principes DDD et CQRS.

Orchestration avec un médiateur : MediatR

Afin de structurer efficacement ces comportements, j'intègrerai MediatR comme bus interne de communication :

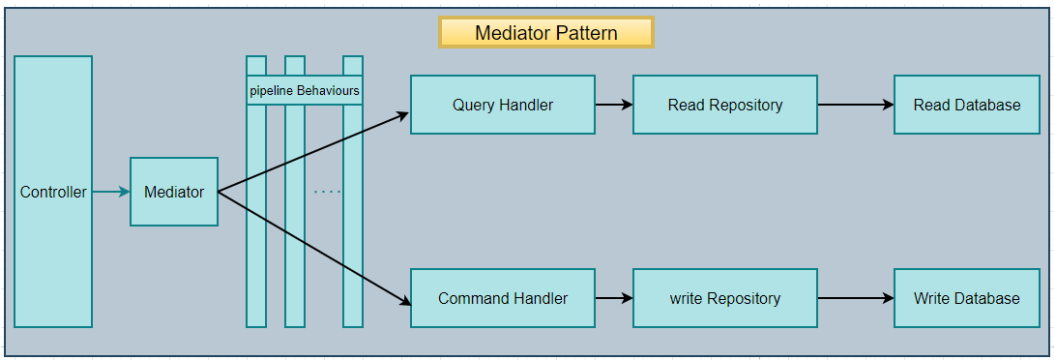


Figure 2 : Source https://medium.com/@aamallikarjuna18/mediatr-mediator-in-dotnet-one-6-4b027e31bb32

- Gestion centralisée des commandes, requêtes et événements.
- Découplage fort entre composants : aucune classe n'appelle directement une autre.
- Propagation automatique des Domain Events générés par le modèle métier.

Orchestration des workflows :

- Validations intégrées
- Transitions d'état des tâches
- Déclenchement de règles
- Mise à jour dépendantes de tâches

Testabilité accrue : chaque commande ou événement peut être testé isolément.

Persistance des données

L'application s'appuiera sur Entity Framework, ce qui me permettra de rester relativement indépendant du moteur de base de données grâce au choix du provider adapté.

Dans un premier temps, nous utiliserons toutefois SQL Server, puisque nous disposons déjà d'un environnement Microsoft opérationnel sur nos VM.

De plus, SQL Server propose des fonctionnalités avancées pour la gestion des champs JSON, telles que :

- JSON_VALUE pour extraire une valeur simple,
- JSON_QUERY pour récupérer un objet ou un tableau JSON,
- OPENJSON pour transformer dynamiquement un JSON en table exploitable, ainsi que la possibilité d'indexer des colonnes calculées dérivées du JSON.

Ces fonctionnalités s'intègrent parfaitement avec l'approche hybride retenue précédemment et me facilitera les requêtes ainsi que la validation et l'évolution des données encapsulées dans les tâches.

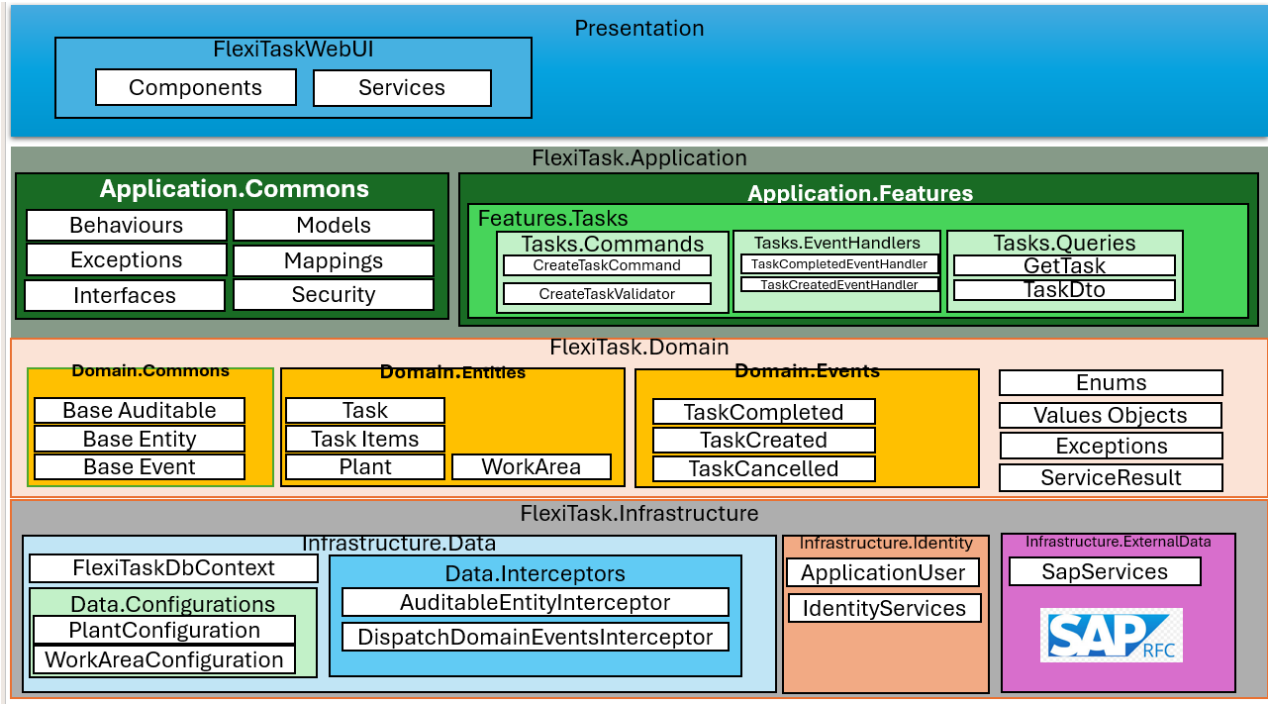
Frontend Blazor application

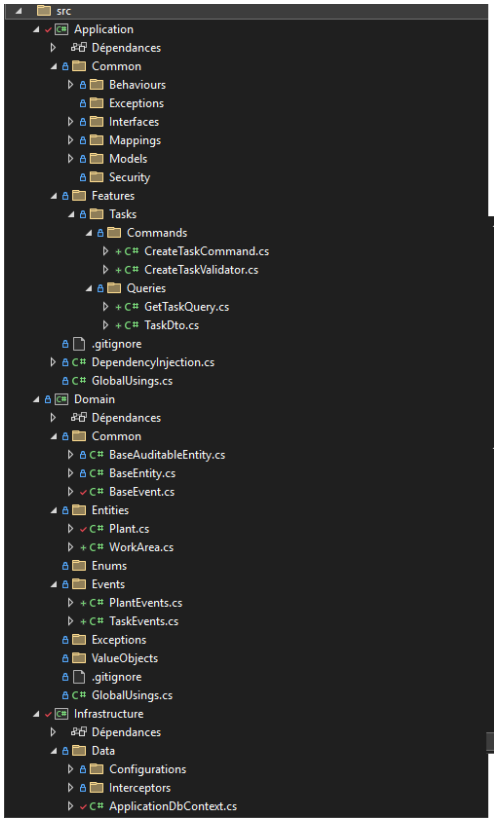
J'ai choisi Blazor Server pour la partie front-end, car son modèle de connexion permanente au serveur (via SignalR) permet de gérer efficacement une application très interactive. Dans mon cas, l'ensemble des tâches est dynamique (créations, changements de statut, dépendances, priorités, etc.), ce qui implique de nombreux échanges avec le serveur.

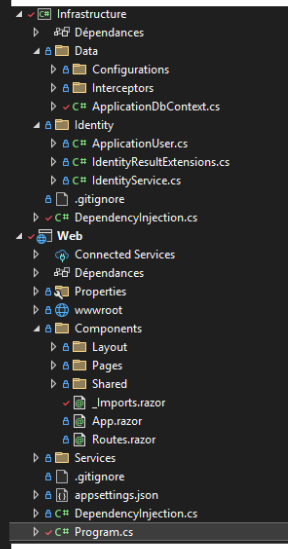
De plus, les solutions proposées par .NET sont multiplateformes, ce qui répond à l'une des contraintes techniques imposées par AGC : assurer l'accessibilité depuis un ordinateur, une tablette ou un smartphone, afin de permettre une utilisation tant en bureau que sur le terrain.

Architecture de l’application

L’application s’appuie sur une architecture DDD, reposant sur une séparation rigoureuse en plusieurs couches. Cette organisation vise à garantir l’indépendance des différentes responsabilités et à faciliter la maintenance ainsi que l’évolution du projet.







Couche Présentation (Blazor Server)

La couche Présentation sera assurée par Blazor Server, qui se chargera de l'interface utilisateur et de la gestion des interactions côté client. Cette couche est conçue pour ne communiquer qu'avec la couche Application, sans accéder directement à la logique métier ou à la persistance des données.

Couche Application

La couche Application orchestera les différents cas d'usage à l'aide du framework MediatR, en appliquant le pattern CQRS (Command Query Responsibility Segregation). Elle dépend uniquement du domaine métier et s'appuie sur des interfaces pour gérer les flux entre la Présentation et le Domaine, garantissant ainsi une séparation claire des responsabilités.

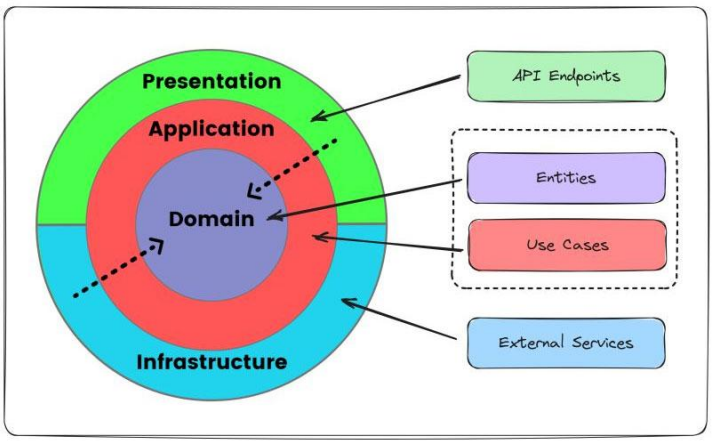
Couche Domain

Au cœur de l'architecture, la couche Domain regroupera les entités, les événements et l'ensemble des règles métier. Cette couche est conçue pour rester totalement indépendante des aspects techniques et des technologies utilisées, afin de préserver l'intégrité du modèle métier.

Couche Infrastructure

Enfin, la couche Infrastructure fournira les implémentations concrètes nécessaires au fonctionnement de l'application, telles que la persistance des données via SQL Server ou l'intégration avec SAP. Elle prendra en charge les détails techniques comme Entity Framework Core, Identity et l'accès à la base de données, tout en évitant de créer des dépendances inverses vers la couche Domain.

L’architecture s’articule ainsi autour de trois couches principales : Domain, Application et Infrastructure, chacune assumant strictement son périmètre et ses responsabilités respectives.




```
Source
|__ Core
|____ Application
|____ Domain
|__ Infrastructure
|____ Identity
|____ Persistence
|____ Shared
|__ Api
|__ ...
```

Figure 3 : source https://www.i2bglobal.com/blog/building-robust-e-commerce-systems-with-domain-driven-design-and-clean-architecture.aspx

Conclusion de la phase d’analyse technique

Durant l’analyse technique, j’ai examiné l’ensemble des solutions possibles et évalué leurs avantages ainsi que leurs inconvénients. Cette démarche m’a permis de prendre position sur certaines technologies, en tenant compte à la fois du projet et de ses besoins réels.

Je ne me sens toutefois pas entièrement à l’aise avec l’idée de mener ce projet à ce stade. Certaines décisions prises aujourd’hui pourraient influencer de manière négative le bon déroulement du développement. Cependant, le fait d’avoir exploré plusieurs pistes me permettra de rebondir plus facilement vers une solution déjà envisagée si un ajustement s’avère nécessaire.

Ce rapport a également mis en évidence plusieurs points clés, notamment la synchronisation des données entre SAP (la source de vérité) et le moteur d’exécution des tâches. Une autre difficulté réside dans le fait que de nombreuses activités au sein des plants se déroulent en dehors du système de gestion des tâches. Il faudra donc déterminer comment ces changements doivent être intégrés dans l’application, et à quel niveau ils sont réellement critiques pour le bon fonctionnement de l’application.

Annexes

Figure 4: F01_UC00:BPMN

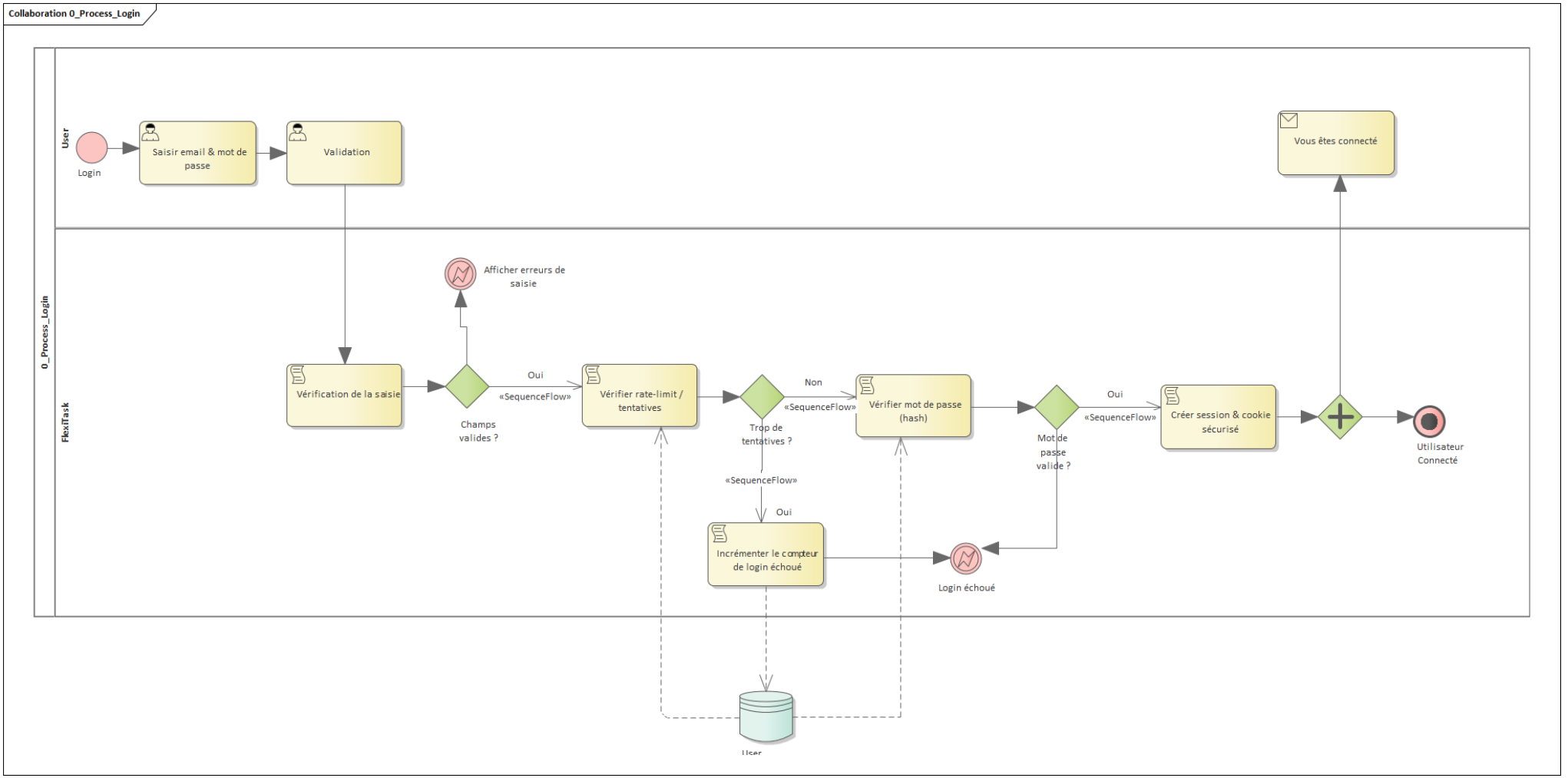


Figure 5:F01_ UC01: Log off

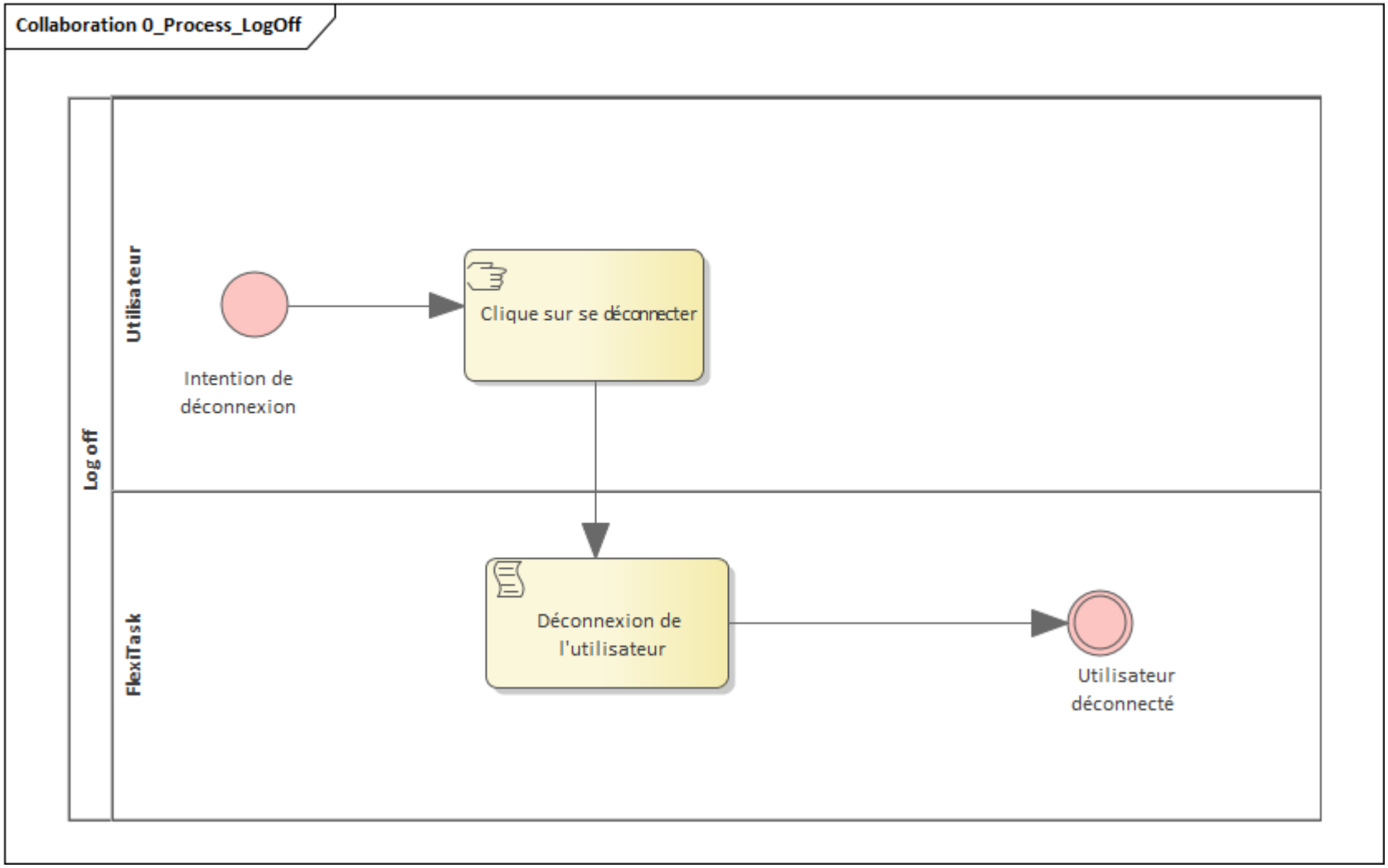


Figure 6 : F01_ UC02 : Register

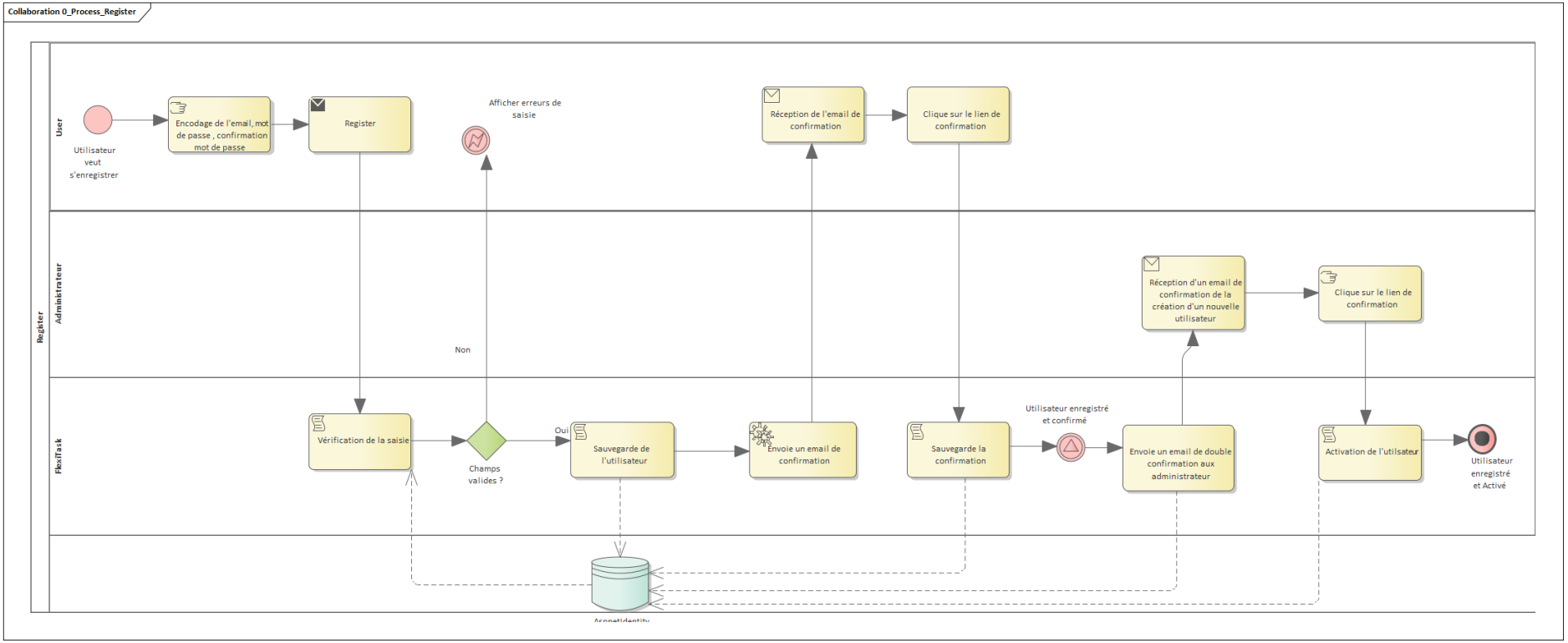


Figure 7: F01_ UC02 : Password recovery

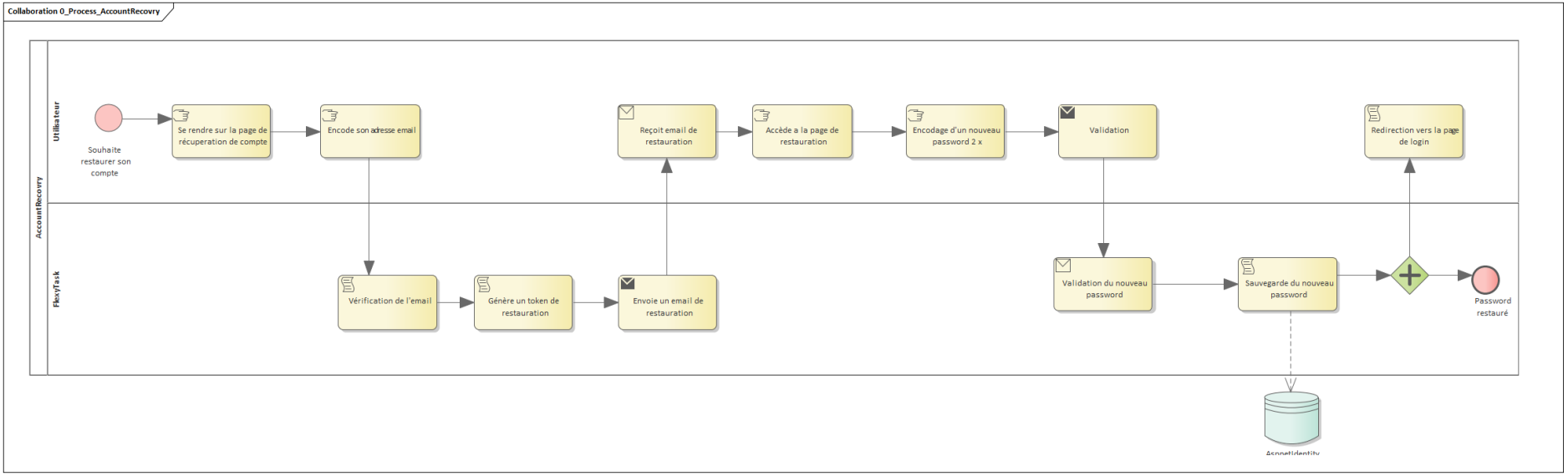


Figure 8 : F02_UC01 : Création d'un plant

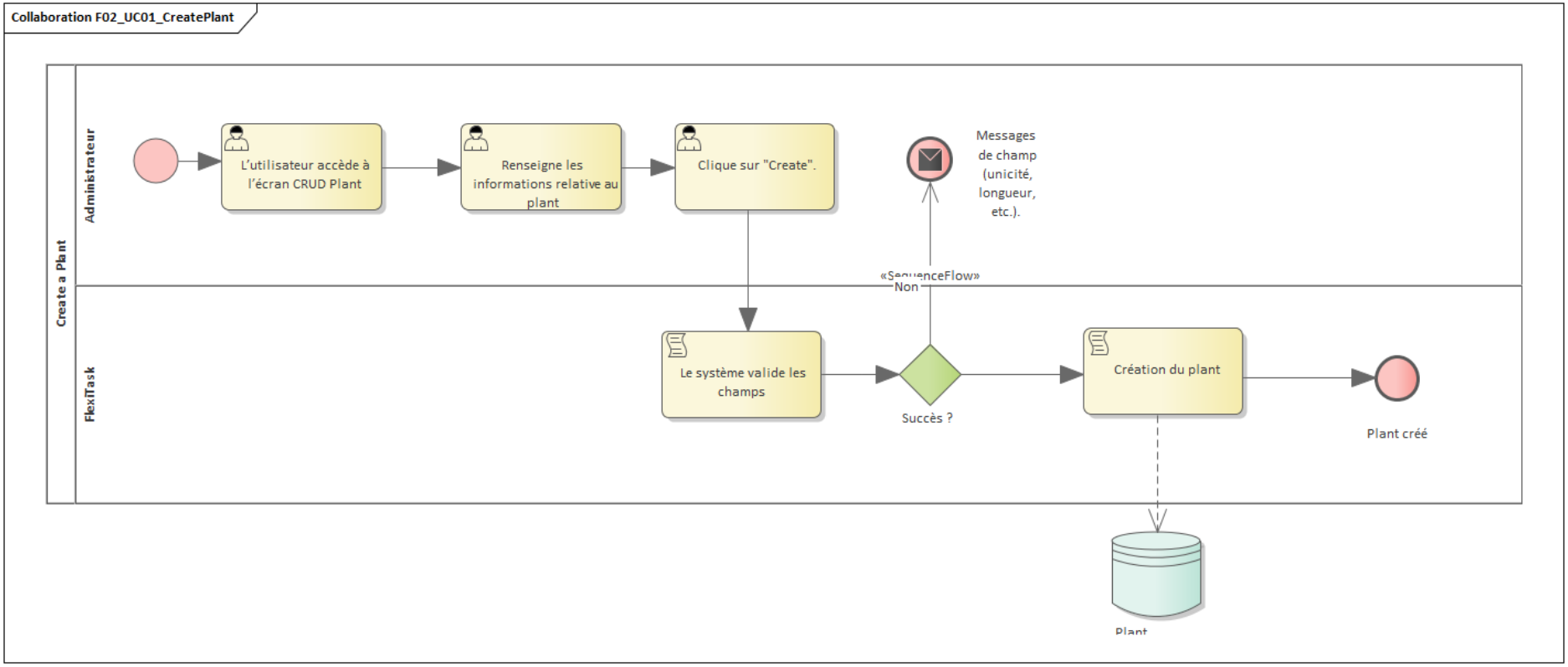


Figure 9 : F02_UC02 : Suppression d'un plant

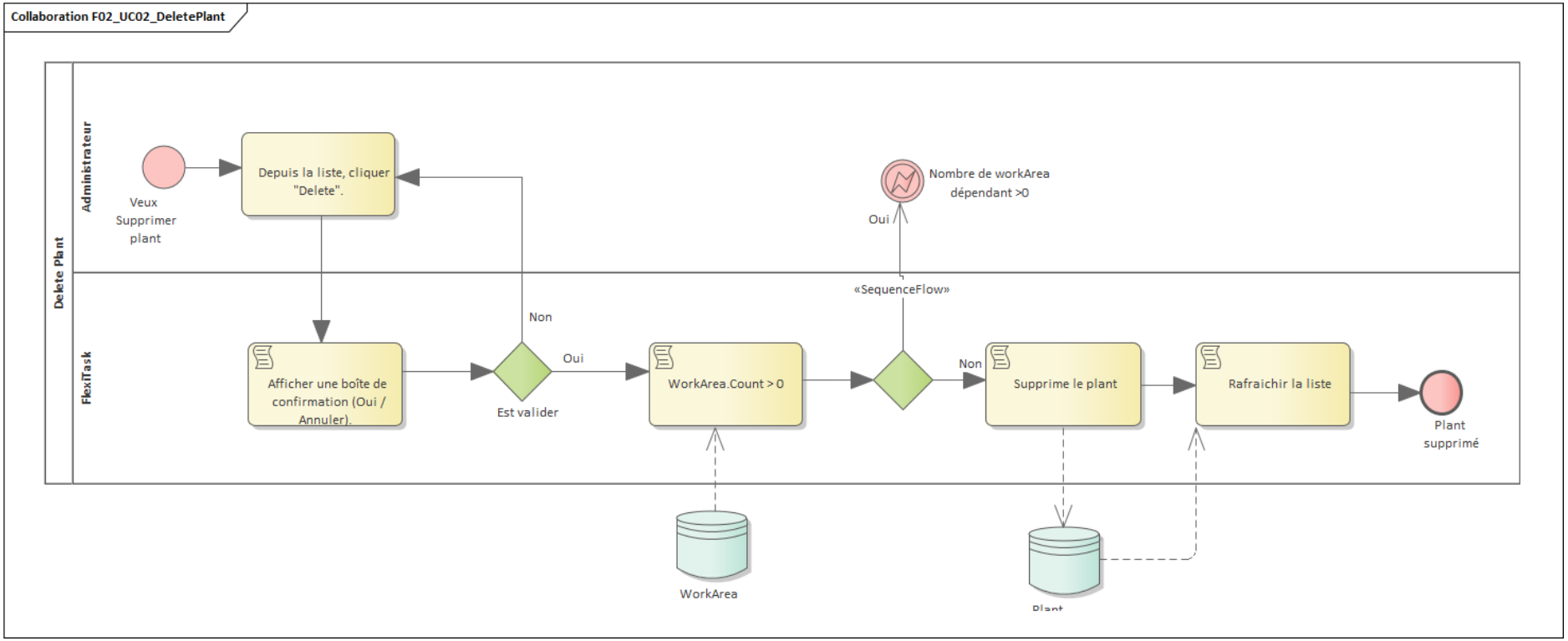


Figure 10 : F02_UC03 : édition d’un plant

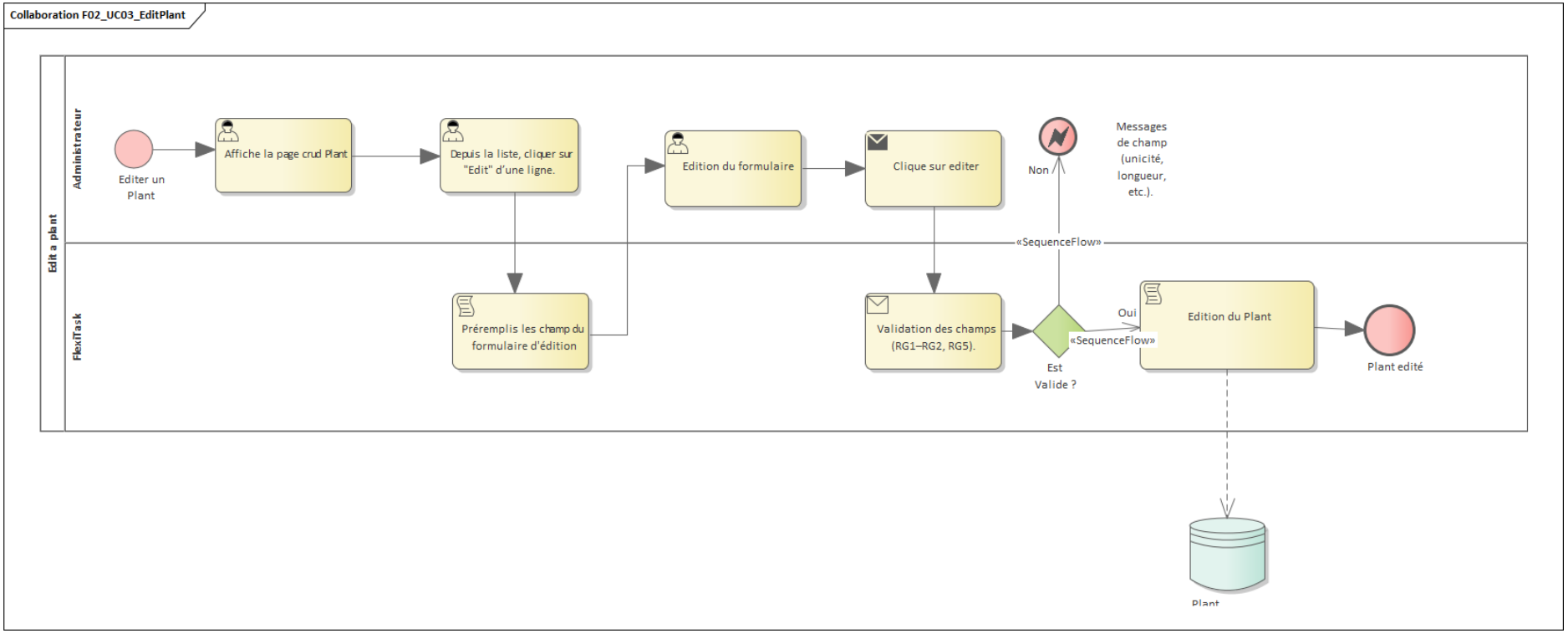


Figure 11: F02_UC04 : Création d'une zone de travail

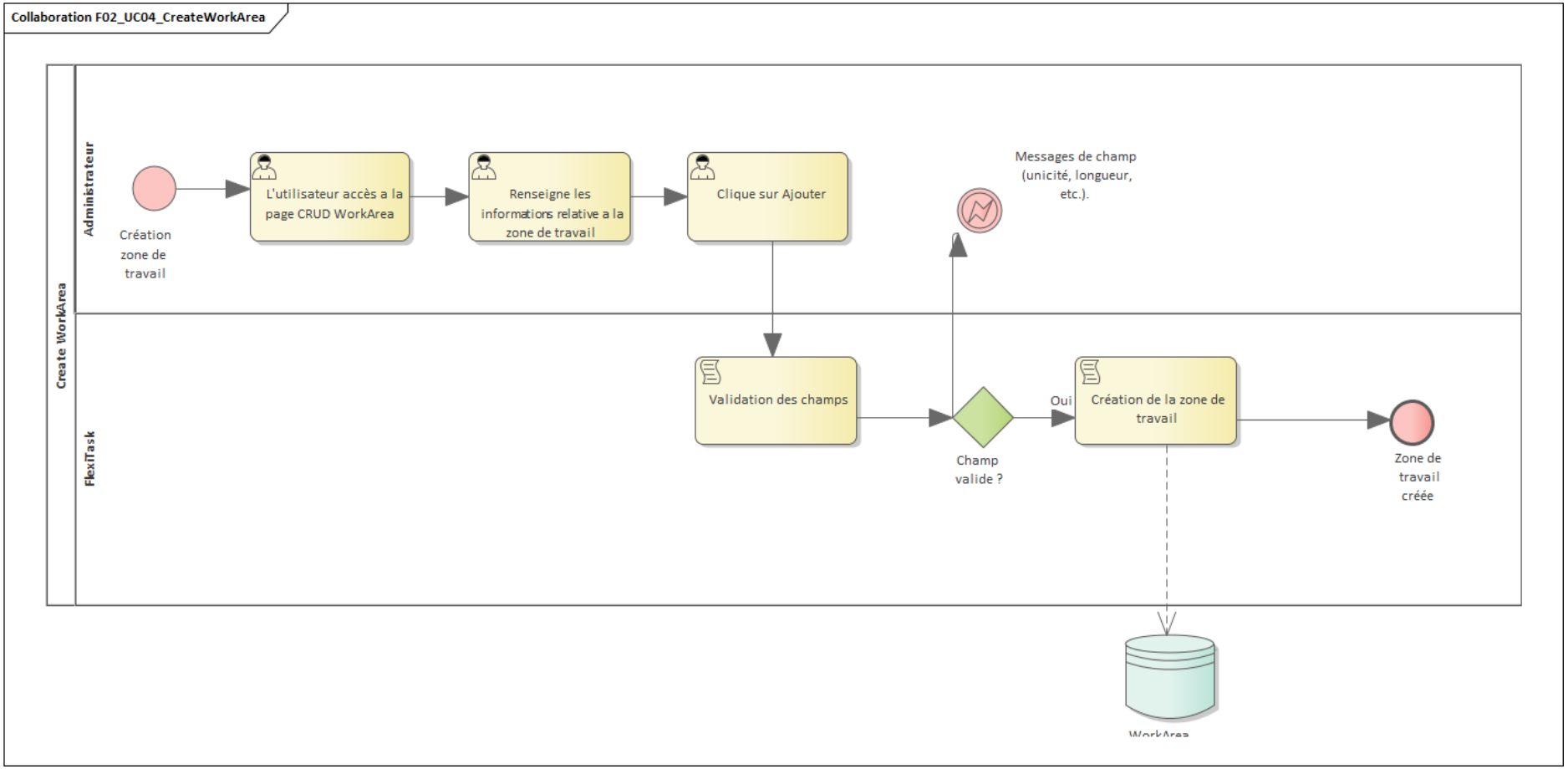


Figure 12: F02_UC05 : Edition d’une zone de travail

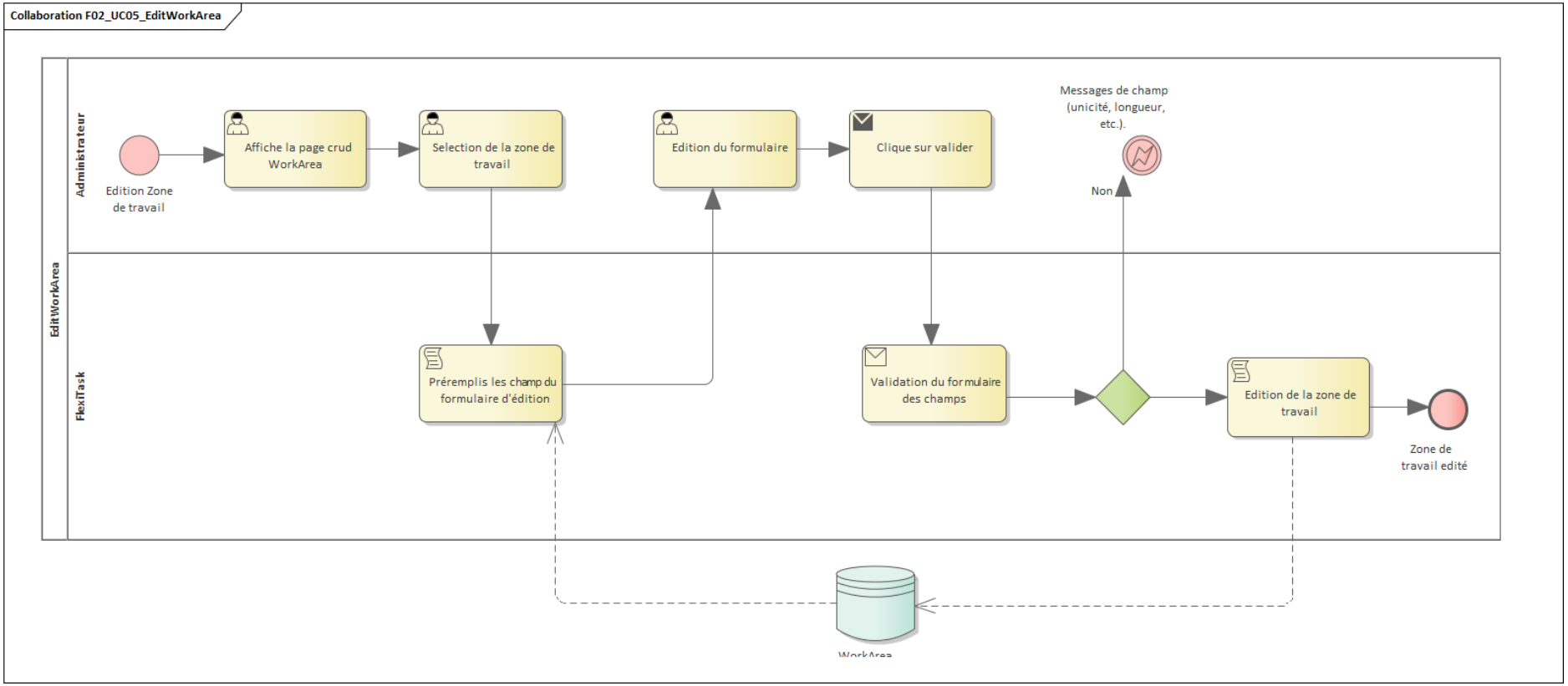


Figure 13 : F02_UC06 : Suppression d'une zone de travail

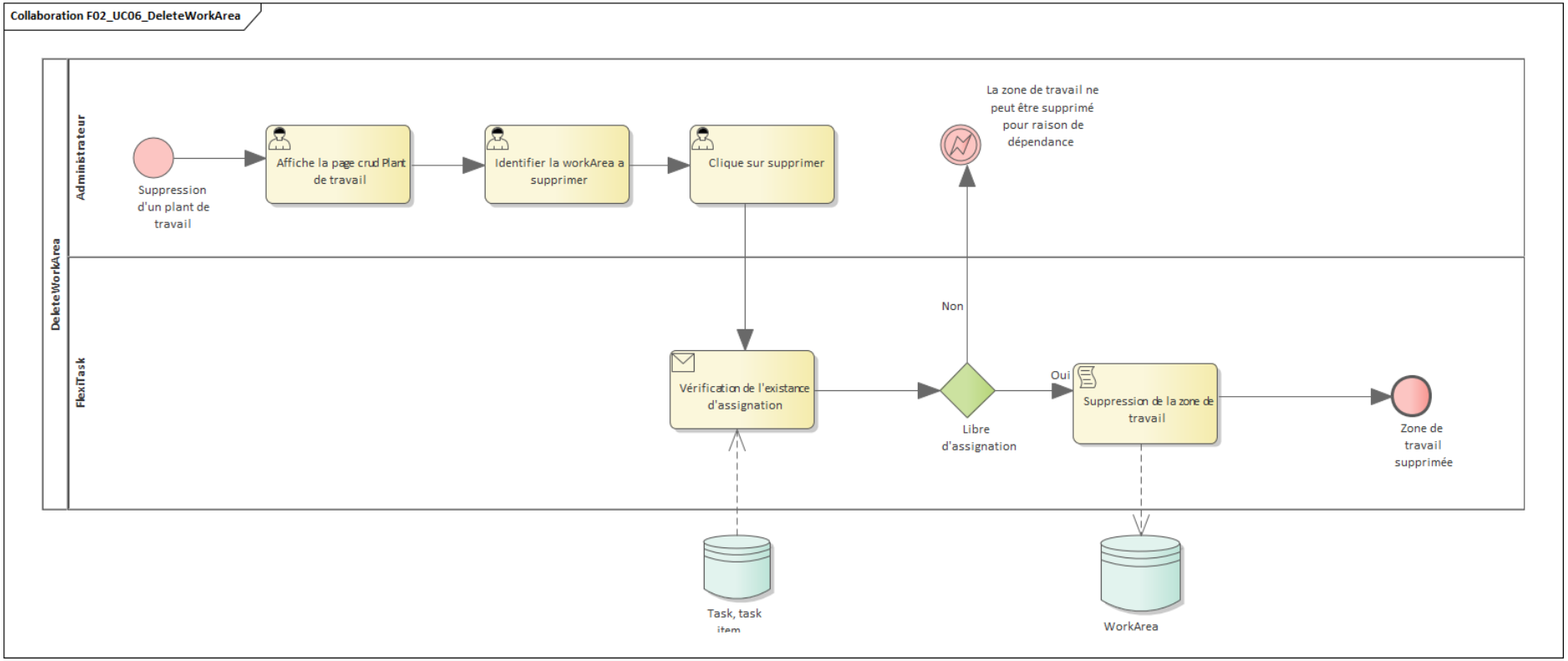


Figure 14 : F03_UC01 : Séquencement d'un groupe de tâches

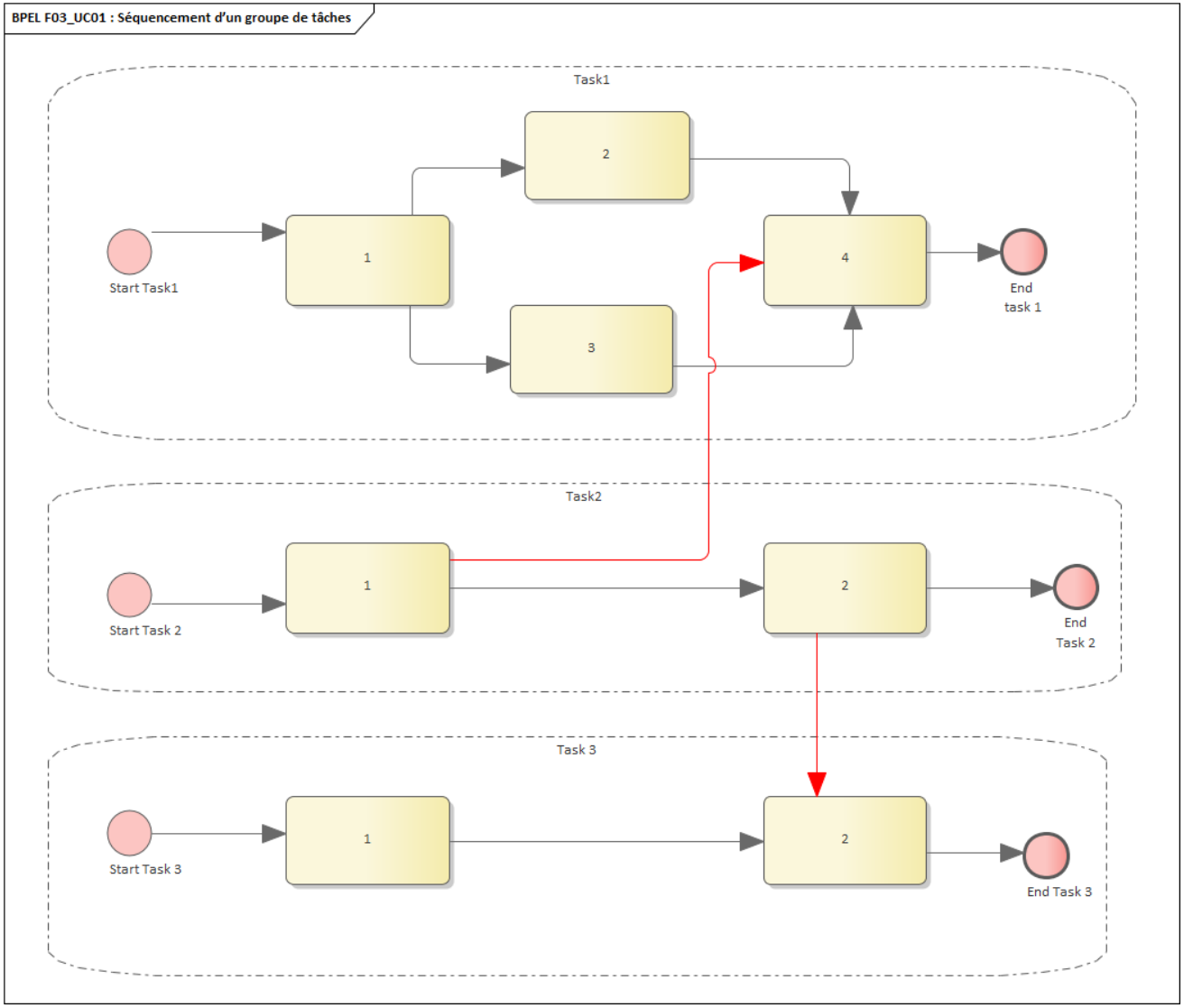


Figure 15: F03_UC02 : Cycle de vie d'une tâche

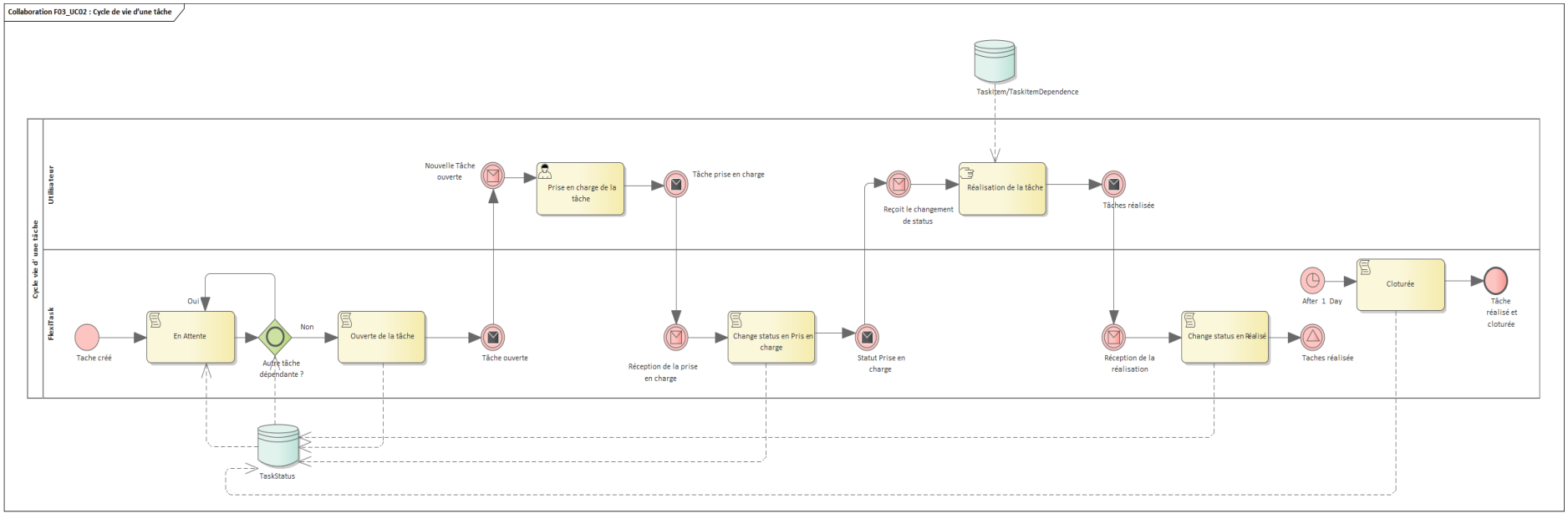


Figure 16 : F03_UC03 : Tâche de transport

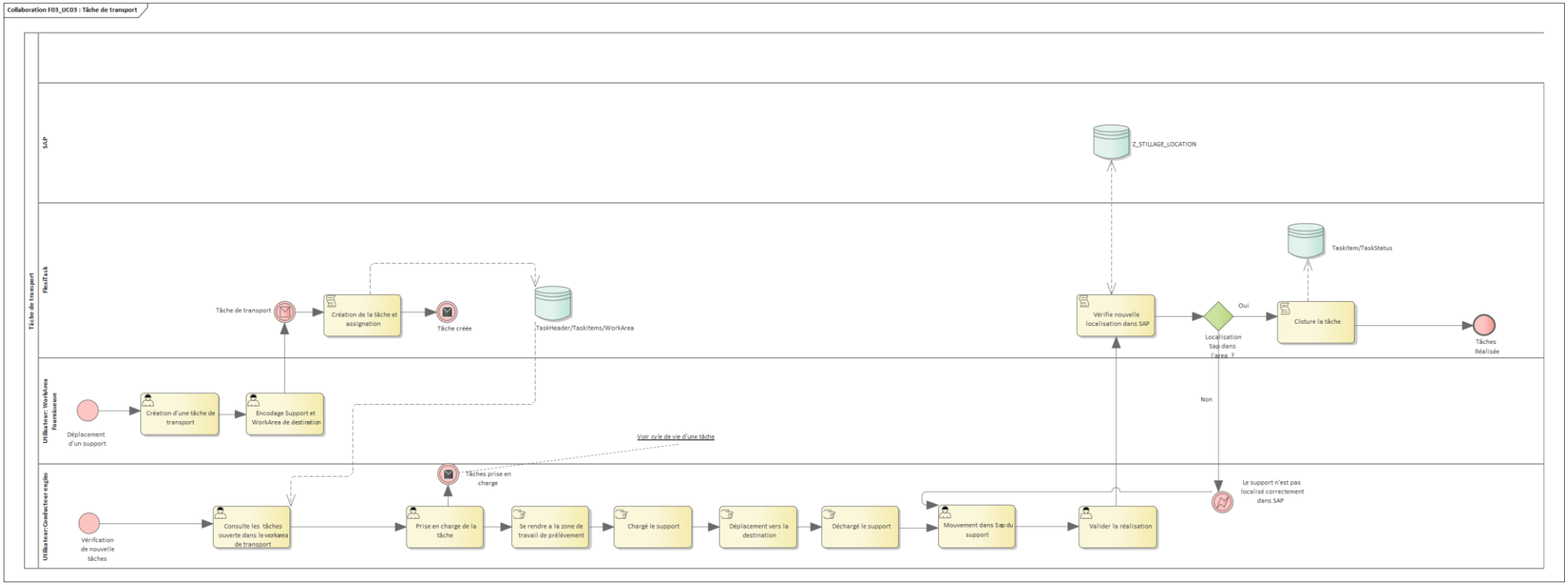


Figure 17 : Modele Entité Asssociation

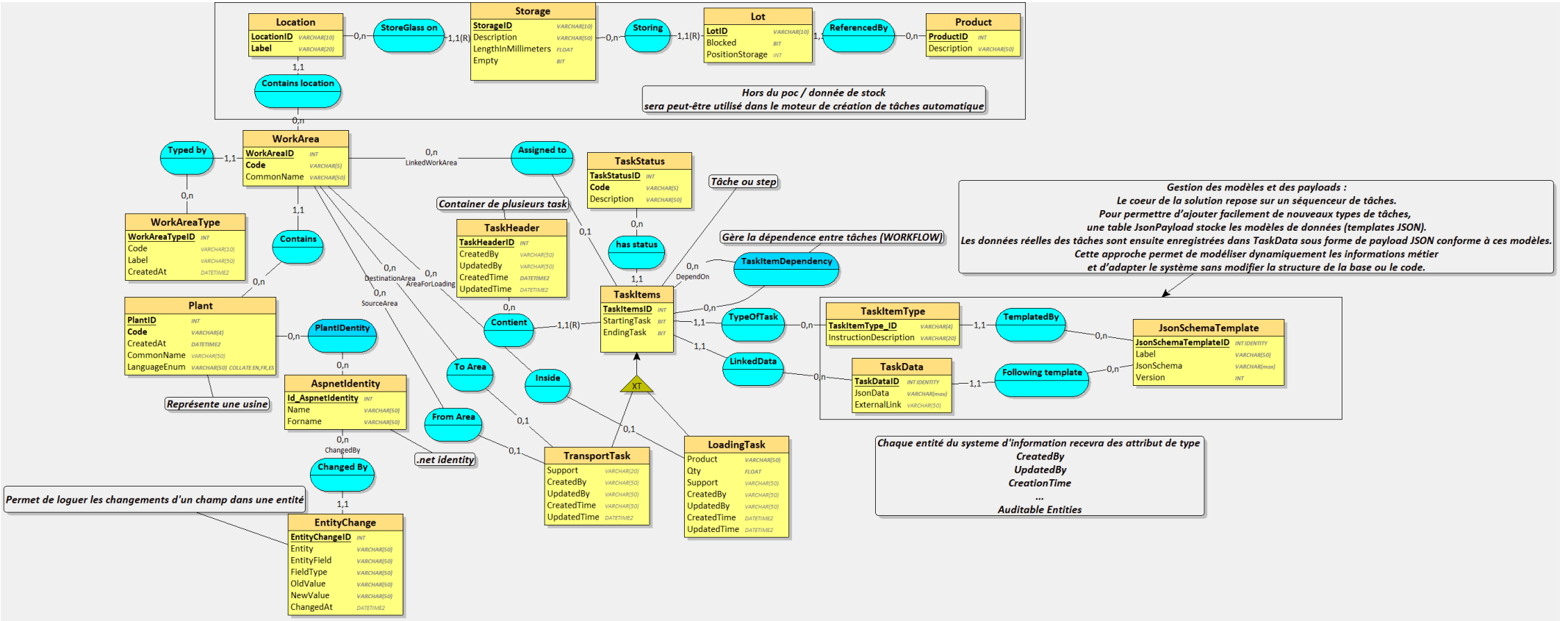


Figure 18 : Solution projet

Figure 19 : Architecture