

Année universitaire 2017/2018

|  |  |  |
| --- | --- | --- |
| **Réalisé par :** |  | **Superviseur :** |
| **Abdelhamid BOULAAJOUL** |  | **M. Tarik MAJID** |
|  |  | **Tuteur :** |
|  |  | **M. Mifdal Lahcen** |

**Cycle Ingénieur de L’ecole Polytechnique d’agadir**

**filiere Genie inforamatique**

**Développement de system de gestion de stock avancer (SGSA)**

**stage TECHNICIEN**

## Dédicaces

Je dédie ce rapport de stage à plusieurs personnes qui ont été d'un soutien inestimable tout au long de mon parcours académique et professionnel.

À mes parents, pour leur amour inconditionnel, leur soutien moral et financier, et pour avoir toujours cru en moi. Leur patience et leur encouragement ont été une source constante de motivation et de détermination.

À mes professeurs et mentors, qui ont partagé leur savoir et leur expérience avec passion et générosité. Leur guidance et leurs conseils ont grandement contribué à mon développement personnel et professionnel.

À mes amis, pour leur amitié sincère, leur soutien et les moments de joie partagés. Leur présence a rendu cette période de ma vie plus agréable et moins stressante.

Enfin, à toute l'équipe de l'entreprise COPAG, et en particulier à mon encadrant professionnel, pour leur accueil chaleureux, leur disponibilité, et leur confiance. Leur expertise et leurs conseils m'ont permis de tirer le meilleur parti de cette expérience de stage.

Merci à vous tous pour votre soutien et votre inspiration.

## Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet de stage.

Tout d'abord, je remercie chaleureusement toute l'équipe de COPAG pour m'avoir accueilli au sein de leur entreprise et pour m'avoir offert cette opportunité enrichissante. Un remerciement particulier à mon encadrant professionnel, M. Tarik MAJID, pour sa guidance, ses conseils avisés, et sa disponibilité tout au long de mon stage. Son expertise et sa bienveillance ont été d'une grande aide pour la réussite de ce projet.

Je souhaite également remercier mes professeurs et tuteurs académiques de l’école polytechnique agadir pour leur soutien continu et leur enseignement de qualité. Leur dévouement et leur passion pour l'enseignement ont largement contribué à mon apprentissage et à ma préparation pour ce stage.

Un grand merci à mes collègues stagiaires et à tous les membres de l'équipe projet pour leur collaboration, leur esprit d'équipe, et les précieux moments partagés. Leur soutien et leur camaraderie ont rendu ce stage non seulement productif mais également agréable.

Je tiens aussi à remercier ma famille et mes amis pour leur soutien moral indéfectible et leurs encouragements constants. Leur présence à mes côtés a été une source de réconfort et de motivation.

Enfin, je remercie tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce projet et à la réussite de mon stage. Votre aide et vos encouragements ont été précieux et je vous en suis profondément reconnaissant.

Merci à vous tous

## TABLE DES MATIERES

## TABLE DES ILLUSTRATIONS

**No table of figures entries found.**

**No table of figures entries found.**

## INTRODUCTION GENERALE

Mon stage, d’une durée de deux mois, s’est déroulé au sein de la coopérative agricole COPAG, spécialisée dans le secteur agroalimentaire. COPAG est une entreprise de grande envergure qui opère principalement dans la production et la distribution de produits laitiers et autres denrées agricoles. Mon stage a eu lieu au siège social de l’entreprise, situé à Taroudant, où j’ai eu l’opportunité d’acquérir de nombreuses compétences dans le domaine de la gestion des stocks et du développement informatique.

Durant cette période, j’ai découvert en profondeur le fonctionnement de l’entreprise ainsi que le métier de développeur dans un environnement industriel. J’ai pu développer des compétences techniques et organisationnelles en participant à des projets concrets, notamment dans le cadre de la gestion avancée des stocks (SGSA). Mon travail a été encadré par un professionnel aguerri qui m’a accompagné dans l’apprentissage des outils et technologies utilisés au sein de l’entreprise.

L’objectif principal de mon stage était de participer à la conception et au développement d’un système de gestion des stocks, un projet d’envergure au sein du département informatique. Mes missions ont principalement porté sur le développement d’une application web et la gestion des bases de données, en lien avec les besoins fonctionnels de l’entreprise. Ce projet a permis de mettre en œuvre différentes technologies modernes comme NestJS, Angular, et TypeORM pour répondre aux exigences spécifiques de la gestion de stock de COPAG.

Ce rapport est structuré en plusieurs chapitres. Le premier chapitre présente le contexte général du projet ainsi que l’environnement de travail. Le deuxième chapitre détaille les spécifications fonctionnelles et techniques du projet. Ensuite, le troisième chapitre aborde l’étude conceptuelle et les différents diagrammes utilisés. Enfin, le dernier chapitre décrit la réalisation du projet et les différentes phases de développement auxquelles j’ai participé, suivie d’une conclusion présentant les perspectives du projet.

# Présentation de l’entreprise

## Présentation générale : Coopérative COPAG

COPAG est une organisation économique constituée de plusieurs services qui se coordonnent entre eux pour former un tissu bien organisé en vue d’avoir une bonne marche de travail par conséquent participer au développement économique. Elle opère dans le secteur agricole au sens le plus large du terme : productions animales (lait, viande) productions végétales (agrumes, primeurs…) agroalimentaire, etc.

COPAG est actuellement le promoteur du développement de l’élevage laitier dans le Souss Massa. Elle a été créée le 7 mai 1987 par un groupe de 39 agriculteurs de la région de Taroudant qui ont éprouvé le besoin et la nécessité de se regrouper afin de maitriser leurs produits agricoles depuis la production jusqu’à un stade avancé de la distribution. Il s’agissait de profiter de la politique de libéralisation des exportations amorcée par l’état marocain.

Le développement de COPAG est le double résultat d’une stratégie d’intégration des stades de la filière agricole et d’une politique de porte ouverte pour augmenter l’effectif des adhérents et faire croître la quantité du lait collecté et transformé.

La COPAG étant une coopérative polyvalente, elle ne se contente pas seulement de la production, la transformation, et la commercialisation du lait et de ses dérivés. Elle s’occupe également de la production et de l’exportation des agrumes et des primeurs. La superficie exploitée atteint 11 100 ha en total, répartis comme suit : 4000 ha d’agrumes, 1100 ha de primeurs, et 6000 ha de cultures fourragères. En ce qui concerne le cheptel bovin, il est constitué à son tour de 80 000 têtes dont 40 000 vaches laitières.

Le caractère économique de la coopérative l’oblige à assurer un ensemble de fonctions pour valoriser la production de ses membres, à travers des actions d’approvisionnement en facteurs de productions, de commercialisation, d’encadrement et de formation en collaboration étroite avec des partenaires nationaux et internationaux. Outre le conditionnement des agrumes et de transformation 11 du lait, la COPAG cherche à améliorer la productivité, la rentabilité de ses activités et par voie de conséquence garantir la croissance économique des secteurs d’activités où elle opère.

Parmi les missions que la COPAG s’est fixée :

• Assurer elle-même ou par l’intermédiaire de ses adhérents le développement socioéconomique du milieu rural de la région du Souss ;

• Offrir des produits agricoles d’origine animale et végétale de plus en plus élaborés qui peuvent satisfaire les attentes actuelles et futures des consommateurs ;

• Améliorer le revenu de la COPAG et de ses adhérents à travers des actions conjuguées à tous les stades de la production, de la transformation et de la commercialisation des produits agricoles (et les dérivés) à forte valeur ajoutée

## RAISON SOCIALE & STATUT JURIDIQUE

### Fiche technique

|  |  |
| --- | --- |
| Raison sociale | Coopérative agricole COPAG Taroudant |
| Date de création | 07 mai 1987 |
| Président | Moulay Mohamed LOULTITY |
| Forme juridique | Coopérative agricole |
| Nombre des adhérents | 14 000 |
| Effectif | Plus de 9 500 des employés directs |
| Secteurs d’activité | Agriculture, industrie agroalimentaire et conditionnement |
| Capacité de production | Pour les agrumes : 8 000 tonnes par an, et pour les primeurs : 10 000 tonnes par an |
| Chiffre d’affaires | 7 MMDH |
| Siège social | BP 1001 FREIJA - 83200 - TAROUDANT MAROC |
| Tél | (05) 28 53 61 71 / 82 / 11 |
| Fax | (05) 28 53 61 39 |
| Email | [mmloultiti@copag.ma](mailto:mmloultiti@copag.ma) |

Tableau 1 Fiche technique de la COPAG

### Activités

COPAG opère principalement dans trois secteurs : la production animale (principalement les produits laitiers), la production végétale (agrumes et autres fruits), et l'agroalimentaire (transformation et commercialisation de produits agricoles). La coopérative est également active dans l'exportation de ses produits, notamment les agrumes, vers des marchés internationaux.

### Organigramme

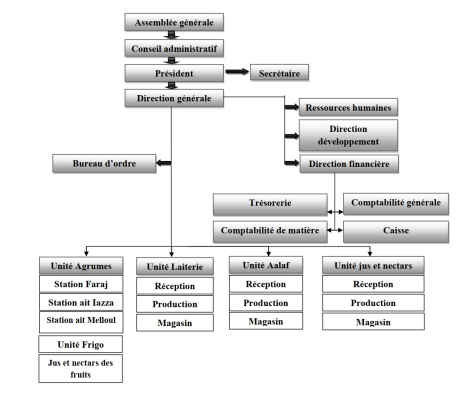


Figure 1 L’organigramme de la COPAG

### Systèmes de production/service

COPAG opère dans différents domaines d’activité liés à la fois à la production végétale et animale et qui s’articulent comme suit :

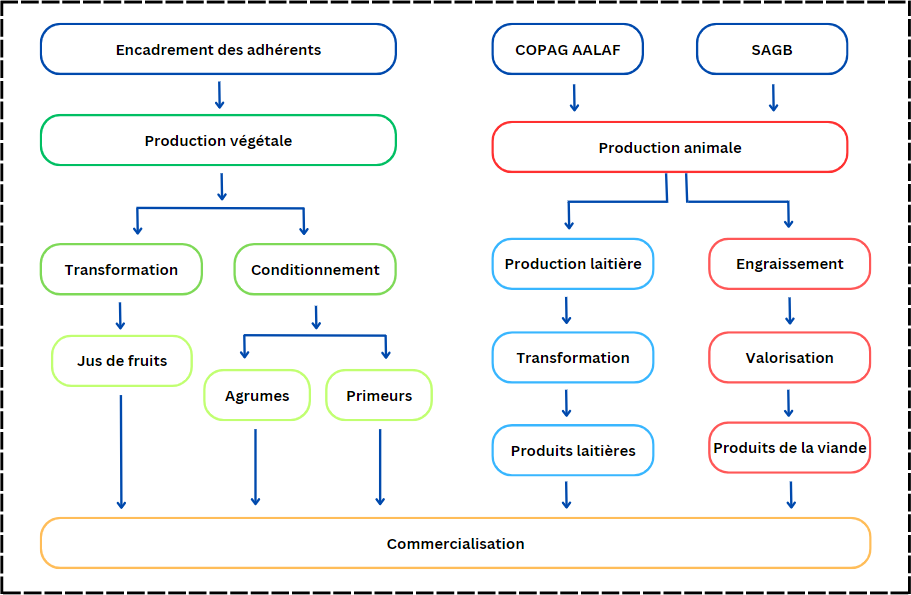


Figure 2 L’organisation de la COPAG

## CONCLUSION

Dans la conclusion de ce chapitre, l’étudiant doit mettre en évidence les points qui ont marqués l’étudiant au sein de l’entreprise d’accueil.

***Remarque****: Ce chapitre doit contenir 3 à 6 pages.*

# Présentation du projet

## Contexte du projet

La réussite de COPAG dépend fortement de l'efficacité et de la précision dans la gestion des stocks. Avec la croissance des volumes de marchandises traitées et la complexité croissante des flux logistiques, il devient crucial d'adopter un système capable de gérer les stocks de manière optimale tout en anticipant les besoins opérationnels. Actuellement, COPAG fait face à des défis majeurs tels que le manque de visibilité en temps réel sur les stocks, l'optimisation limitée des espaces de stockage et des coûts de gestion accrus, menant à des inefficacités et des dépenses supplémentaires. Dans le but de pallier ces obstacles et d'améliorer ses performances logistiques, COPAG a entrepris le développement d’un Système de Gestion de Stock Avancé (SGSA). Ce projet a pour objectif de garantir une traçabilité complète des stocks, d'optimiser l'utilisation de l'espace de stockage, de réduire les coûts de gestion et d'améliorer la précision des opérations logistiques tout en offrant des outils analytiques pour une prise de décision plus éclairée.

### Problématique

COPAG bénéficie d'une vaste expérience en gestion d'entreprise, principalement grâce à son système ERP, qui orchestre les processus, opérations et finances. En tant que coopérative, COPAG collabore avec de nombreux adhérents, effectuant des transactions d’achat et de vente. Afin d’améliorer l’efficacité de ces échanges, un système d’information spécialisé est devenu nécessaire. Ainsi, COPAG a entrepris la mise en place de systèmes d’information spécifiques pour chaque coopérative, permettant des échanges électroniques fluides entre l’ERP de COPAG et ces nouveaux systèmes.

### Solution et Objectif

L'objectif du projet est de concevoir, développer et déployer un Système de Gestion de Stock Avancé (SGSA) répondant aux besoins spécifiques de COPAG en matière de gestion des stocks. Le SGSA sera une solution numérique intégrée, dotée de fonctionnalités avancées visant à optimiser la gestion des stocks, améliorer la traçabilité, et réduire les coûts associés. Plus précisément, l’objectif est de développer une application qui permettra un suivi en temps réel des mouvements de stocks, une optimisation des emplacements de stockage, une estimation précise des coûts, ainsi qu'une gestion efficace des différentes dimensions des stocks. Cette application vise à réduire les coûts de gestion des stocks, à accroître la satisfaction des clients, et à renforcer la compétitivité de COPAG sur le marché. Elle offrira également une meilleure traçabilité grâce à la mise en œuvre de méthodologies adaptées aux spécificités de l’entreprise.

### Livrable final

Le livrable final de ce projet consistera en un Système de Gestion de Stock Avancé (SGSA) entièrement fonctionnel, prêt à être déployé chez COPAG. Ce SGSA sera accompagné d'une documentation complète qui décrira en détail son architecture, son fonctionnement et les procédures nécessaires à son déploiement et à son utilisation.

Avant la livraison, le système sera soumis à des tests rigoureux pour garantir sa fiabilité, sa performance et sa conformité aux besoins spécifiques de COPAG. Une fois ces tests réussis et la validation de l'équipe de projet obtenue, le SGSA sera considéré comme prêt pour une utilisation opérationnelle au sein de l'entreprise

### Périmètre fonctionnelle de projet

Le périmètre fonctionnel du projet englobe les fonctionnalités essentielles d'un Système de Gestion de Stock Avancé (SGSA), notamment la gestion complète des stocks, des commandes, des emplacements, des articles et des inventaires. Le système sera conçu pour offrir une traçabilité précise des stocks en intégrant diverses méthodologies de suivi, telles que la traçabilité par lot, par numéro de série, par date de péremption et par emplacement.

## Spécification générale

### Spécification des besoins fonctionnels

Dans ce chapitre, nous procéderons à une analyse approfondie des exigences du projet. Nous examinerons en détail les besoins fonctionnels et non fonctionnels du système, ainsi que les exigences spécifiques des utilisateurs. Cette analyse nous permettra de définir de manière précise les fonctionnalités et les caractéristiques essentielles de l'application à développer. Nous explorerons également les contraintes et les limitations qui pourraient influencer la conception et le développement du système. En fournissant une compréhension détaillée des exigences du projet, ce chapitre servira de fondement solide pour la phase de conception et de développement ultérieure.

#### Les Dimensions de Stock

##### Analyse de stock

###### Analyse Temporelle :

**Durée de Stockage :** Suivi du temps que les produits passent en stock pour gérer les produits périssables.

**Rotation des Stocks :** Identifier les produits à rotation rapide et lente pour optimiser les niveaux de stock.

###### Analyse Financière :

**Coût de Stockage** : Calcul des frais de stockage, y compris entreposage et assurance.

**Valorisation des Stocks :** Utilisation de méthodes telles que le PMP ou FIFO pour refléter la valeur financière des stocks.

##### Dimensions de Traçabilité

###### Dimension Physique :

**Localisation des Produits :** Gestion des emplacements dans l'entrepôt pour un suivi efficace.

**Volume et Capacité :** Suivi de la capacité physique des zones de stockage et de l'optimisation de l'espace, en tenant compte des dimensions et du volume des produits stockés, ainsi que l'unité de mesure relative au stockage de l’article.

**Numéros de Lots et de Séries :** Attribution de numéros uniques pour une traçabilité précise.

**Numéro de Palette :** Suivi des palettes pour optimiser l'espace de stockage et les mouvements logistiques.

**Référence de Vente au Poids :** Suivi des produits vendus au poids avec des SKU spécifiques pour une gestion précise des variations de poids.

**Référence d’article SKU** : Le SKU (Stock Keeping Unit) est une référence unique pour identifier et suivre facilement les produits, facilitant la gestion des stocks et des commandes grâce aux codes-barres.

###### Axe analytique :

1. Activité de Production

Cet axe analytique permet de suivre les stocks en relation avec les différentes activités de production de l'entreprise. Il offre une vision détaillée des coûts associés à chaque étape du processus de production.

**Objectifs :**

* Comprendre les coûts de production.
* Optimiser les processus de fabrication.
* Identifier les inefficacités et les opportunités d'amélioration.
* Gérer les matières premières, les produits semi-finis et les produits finis de manière efficace.

1. Business Unit.

Cet axe analytique se concentre sur l'analyse des stocks et des coûts par unité commerciale ou secteur d'activité spécifique au sein de l'entreprise. Il permet de mesurer les performances et la rentabilité des différents segments de marché ou lignes de produits.

**Objectifs :**

* Mesurer les performances des différentes unités commerciales.
* Évaluer la rentabilité des segments de marché.
* Prendre des décisions stratégiques basées sur la performance des différentes unités.

1. Canal de Vente

Cet axe analytique permet de suivre les stocks et les coûts en fonction des différents canaux de vente. Il aide à optimiser les niveaux de stock selon les canaux de vente et à identifier les canaux les plus rentables.

**Objectifs :**

* Optimiser les niveaux de stock pour chaque canal de vente.
* Identifier les canaux de vente les plus rentables.
* Ajuster les stratégies de distribution pour maximiser les ventes et la rentabilité.

#### Gestion de Journal de Stock (Consommation et Production de Stock)

La gestion du journal de stock, en particulier les aspects liés à la consommation et à la production de stock, est cruciale pour suivre l'utilisation des matériaux et la création de produits finis. Voici une explication détaillée des différents aspects de cette gestion.

##### Consommation de stock

**Enregistrement** : Matériaux utilisés pour la production, consommation interne, dons ou déchets.

**Suivi des Quantités** : Quantités utilisées et mises à jour en temps réel.

**Analyse** : Rapports et tendances de consommation.

##### Production de Stock

**Enregistrement** : Produits finis et semi-finis.

**Suivi des Quantités** : Quantités produites et mises à jour en temps réel.

**Analyse** : Rapports de production et efficacité.

##### Traçabilité et Contrôle

**Journal des Transactions** : Enregistrement détaillé des transactions.

**Audit et Réconciliation** : Traçabilité pour audits.

##### Contrôles Internes :

**Vérifications Périodiques** : Comparaison des enregistrements avec les quantités physiques.

**Ajustements de Stock** : Rectification des écarts.

#### Gestion d'Ordre de Transfert

La gestion des ordres de transfert est une fonctionnalité clé dans la gestion avancée des stocks pour coordonner et optimiser le déplacement des produits entre différents emplacements ou entrepôts.

##### Process de transfert de stock.

Initiation et Soumission :

* Demande initiée par un employé autorisé, incluant les détails nécessaires.
* Soumission pour validation par le responsable d’entrepôt.

Validation et Mise à Jour :

* Validation de la demande, mise à jour des stocks et confirmation envoyée.
* Création et validation des ordres de transfert par l’expéditeur.

Réception :

* Mise à jour des stocks lors de la réception des articles transférés.

##### Qualité de service de satisfaction des demandes de transfert.

La qualité de service de satisfaction des demandes de transfert est cruciale pour assurer une gestion efficace et fluide des stocks au sein d'une entreprise. Elle repose sur plusieurs dimensions et indicateurs clés qui permettent de mesurer et d'améliorer la performance des processus de transfert de stock. Voici les aspects essentiels à considérer :

**Taux de Satisfaction :** Pourcentage de demandes satisfaites en quantité et qualité.

**Temps de Traitement :** Durée totale pour traiter une demande de transfert.

**Disponibilité des Stocks :** Mesure de la disponibilité des articles pour les transferts.

**Coût des Transferts :** Coût total associé aux transferts de stock.

#### Gestion de Journal de Comptage.

La règle de gestion des journaux de comptage encadre le processus de comptage physique des stocks dans un entrepôt. Elle vise à garantir l'exactitude des enregistrements de stock en comparant les quantités physiques avec les données théoriques et en ajustant les écarts identifiés.

**Initiation et Création :** Création d’un journal pour un entrepôt spécifique, avec spécification des articles à compter.

**Blocage des Articles :** Verrouillage des transactions de stock pour éviter les modifications pendant l’inventaire.

**Saisie Précise des Comptages :** Inclusion des valeurs unitaires, quantités et dimensions de stockage.

**Clôture et Ajustement :** Vérification des Données : Examen des lignes de comptage et des coûts.

**Clôture du Journal :** Verrouillage du journal et ajustement des stocks.

**Libération des Transactions :** Déblocage des transactions pour reprendre les opérations normales.

#### Gestion de Qualité

La gestion de la qualité dans un système de gestion de stock avancé implique deux aspects principaux : les contrôles de qualité et la gestion des non-conformités. Voici une explication détaillée de ces éléments :

##### Contrôles de Qualité

**Inspection à la Réception :** Vérification minutieuse des produits reçus selon des critères définis.

**Inspection en Cours de Stockage :** Contrôles réguliers pour détecter toute dégradation.

##### Gestion des Non-Conformités

**Enregistrement :** Identification et documentation des non-conformités.

**Analyse des Causes :** Enquête sur les causes profondes des problèmes.

**Historique et Rapports :** Traçabilité des non-conformités et génération de rapports pour améliorer les processus.

#### Comptabilité de stock

La gestion des coûts est essentielle pour évaluer le stock (matières premières, produits finis, emballages). Les méthodes comme FIFO, LIFO, et la moyenne pondérée permettent de suivre la valeur des articles et d’optimiser les ressources, aidant à une gestion efficace du coût de revient.

##### Les opérations impactant la valeur de stock.

**Réception de marchandises :** Augmente la valeur du stock selon le coût d'achat.

**Production et assemblage :** Le coût des matières premières diminue, tandis que celui des produits finis augmente.

**Expéditions et ventes :** Réduit la valeur du stock selon la méthode d’évaluation utilisée.

**Inventaire physique :** Ajuste le stock pour les écarts physiques.

**Transferts internes :** Modifie la valeur du stock selon les emplacements.

**Retours de marchandises et ajustements :** Affecte la valeur en fonction des retours ou réévaluations périodiques.

**Destruction :** Retire les articles endommagés ou obsolètes du stock.

##### Les transactions de stock.

**Transactions de consommation :** Réduisent le stock (ventes, prélèvements, pertes).

**Transactions de production :** Augmentent le stock (réception de matières, fabrication).

##### Méthode de calcul de coût des articles en stock.

Les transactions de consommation et de production de stock peuvent avoir des impacts significatifs sur le coût des articles en stock, selon les différentes méthodes de coût utilisées. Voici une explication détaillée pour chaque méthode couramment utilisée :

###### Méthode PMP :

La moyenne pondérée (PMP) est la valeur moyenne de chaque unité de stock, mise à jour à chaque nouvelle réception. Elle résulte de la multiplication de chaque transaction par son prix de revient, reflétant son importance en quantité.

Lorsque les marchandises sont reçues à la suite d'une transaction, le prix de réception est utilisé pour mettre à jour la valeur du prix moyen pondéré. Le prix moyen pondéré est calculé selon la formule suivante :

(Stock physique existant \* PMP existante) + (quantité de réception \* prix de transaction ) / (Stock physique existant + quantité de réception)

Exemple de calcule de PMP :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | Transaction | Qtité | Prix U. | Qtité de stock | Valeur de stock | PMP |
| 01/06/2024 | Stock initial | 10 | 16 | 10 | 160 | 16 |
| 07/06/2024 | Entrée | 100 | 19,5 | (10+100) = 110 | (160+1950) = 2110 | (2110/110) = 19,18 |
| 10/06/2024 | Sortie | 15 | 19,18 | (110-15) = 95 | (95x19,18) = 1822,27 | (1822,27/95) = 19,18 |
| 20/06/2024 | Sortie | 30 | 19,18 | (95-30) = 65 | (65x19,18) =  1246,81 | (1246,81/65) = 19,18 |
| 01/07/2024 | Entrée | 80 | 22 | (80+65) = 145 | (1246,81+1760) = 3006,81 | (3006,81/145) = 20,73 |

Tableau 2 Exemple de calcule de PMP

###### Méthode FIFO :

La méthode FIFO (premier entré, premier sorti) valorise les sorties de stock au coût de l'article le plus ancien dans le stock. Cette méthode est généralement utilisée pour les produits périssables afin d’épuiser les lots arrivés en premier face au risque de perte de valeur par obsolescence. Le système crée des règlements où la première réception correspond à la première sortie, et ainsi de suite.

Exemple de calcule de FIFO :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Transaction** | **Qtité** | **Prix U.** | **Montants** | **Qtité** | **Prix Unitaire** | **Montants** |
| 01/06 | Stock Initial | 80 | 25 | 2000 | 80 | 25 | 2000 |
| 02/06 | Achat | 30 | 32 | 960 | 80 | 25 | 2000 |
| 30 | 32 | 960 |
| 10/06 | Achat | 70 | 28 | 1960 | 80 | 25 | 2000 |
| 30 | 32 | 960 |
| 70 | 28 | 1960 |
| 15/06 | Vente (100) | 80 | 25 | 2000 | 10 | 32 | 320 |
| 20 | 32 | 640 | 70 | 28 | 1960 |
| 22/06 | Achat | 20 | 30 | 600 | 10 | 32 | 320 |
| 70 | 28 | 1960 |
| 20 | 30 | 600 |
| 25/06 | Vente(50) | 10 | 32 | 320 | 30 | 28 | 840 |
| 40 | 28 | 1120 | 20 | 30 | 600 |

Tableau 3 Exemple de calcule de FIFO

##### Gestion de coût physique et financier.

**Augmentations physiques:** Ex: Réceptions de commande, fin d’ordre de fabrication.

**Sorties physiques:** Ex: Bon de livraison, prélèvements d’ordres de fabrication.

##### Dimensionnement de calcul de quantité et coût de stock.

Le stock, de chaque article, peut être évaluer en quantité et en coût suivant deux axes dimensionnels :

* Groupe de dimension de stockage :
  + Site
  + Entrepôt
  + Emplacement

Pour la gestion des dimensions de stockage, l'administrateur système doit définir les dimensions nécessaires pour le calcul de la quantité et de la valeur du stock. Par défaut, le système utilise les dimensions site, entrepôt et emplacement pour suivre la quantité de stock.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Article** | **Objet de suivi** | **Site** | **Entrepôt** | **Emplacement** |
| A1 | Coût | X | X | - |
| Quantité | X | X | X |
| A2 | Coût | X | - | - |
| Quantité | X | X | X |

* Groupe de dimension de suivi :
  + Numéro du lot
  + Numéro de série
  + Propriétaire.
  + Palette.

Pour la gestion des dimensions de suivi, l'administrateur système doit définir les dimensions nécessaires pour le calcul de la quantité et de la valeur du stock.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Article** | **Objet de suivi** | **Lot** | **Série** | **Palette** | **Propriétaire** |
| A1 | Coût | X | X | - | - |
| Quantité | X | X | X | - |
| A2 | Coût | X | - | - | - |
| Quantité | X | X | X | - |
| A3 (Consignation) | Coût | - | - | - | - |
| Quantité | - | - | - | X |

##### Modèle d’article :

Un modèle d'article est une configuration utilisée pour définir les méthodes de calcul de coût des articles et les stratégies de stockage. Voici les éléments clés d'un modèle de prix d'article :

* + Méthode de valorisation de stock (PMP, FIFO)
  + Stock physique négative.
  + Produit stocké.

##### La Clôture de stock.

La clôture de stock permet de finaliser les ajustements et les calculs de coûts en fin de période.

#### Réception de bon de commande :

La gestion de la réception de stock passe par plusieurs étapes essentielles pour s'assurer que les produits sont correctement enregistrés et conformes à la commande. Voici un résumé du processus :

1. **Sélection et vérification des bons de commande** : Le gestionnaire sélectionne le bon de commande dans le système et vérifie les lignes de commande (quantité, dimensions). Les différences entre ce qui a été commandé et ce qui est reçu sont notées.
2. **Création de bons de réception** : Lors de la réception des articles, un bon de réception est généré automatiquement en statut brouillon, permettant des modifications avant validation.
3. **Contrôle de qualité** : Pour les articles nécessitant un contrôle, un ordre de qualité est créé. Les résultats sont enregistrés dans le système, et les lignes non conformes sont annulées.
4. **Validation et mise à jour des stocks** : Après la validation des bons de réception, les stocks sont mis à jour en termes de quantité et de valeur.
5. **Impression des bons de réception** : Une fois validé, le bon de réception est imprimé pour être archivé.
6. **Coordination gestionnaire/équipe qualité** : Collaboration entre les deux parties pour s'assurer que les quantités, dimensions, et obligations de contrôle sont correctement enregistrées.

#### Administration de la Cartographie

Les éléments clés à codifier sont les organisations, sociétés, sites, entrepôts, zones, allées, et emplacements. Chaque entité doit avoir un code unique pour garantir une identification précise et une gestion efficace des stocks.

1. **Organisation** : Un code unique est attribué à chaque organisation avec des informations de contact à jour.
2. **Sociétés, sites, entrepôts** : Chaque niveau hiérarchique doit être clairement codifié, en détaillant les caractéristiques comme la localisation et les conditions de stockage.
3. **Zones, allées, emplacements** : Des codes uniques sont attribués à chaque zone physique de l'entrepôt avec des indications sur la capacité et l'accès.

#### Gestion Master-data

La codification des données de base inclut les comptes, fournisseurs, et articles.

1. **Comptes et fournisseurs** : Chaque compte et fournisseur reçoit un code unique. Les informations doivent être mises à jour régulièrement pour éviter les erreurs.
2. **Fiches articles** : Les articles sont identifiés par un code unique, avec des descriptions complètes et classifiés pour faciliter la gestion des stocks et des achats. Les données doivent être synchronisées en temps réel pour assurer la cohérence dans tous les systèmes.

### Spécification de besoins non fonctionnels

L'analyse des besoins non fonctionnels est essentielle dans le développement d'une application de gestion commerciale et de vente. Elle garantit que le système répond aux attentes des utilisateurs, maintient une performance élevée et respecte les normes et réglementations.

#### Performance

L'application doit pouvoir traiter un grand volume de transactions et supporter de nombreux utilisateurs simultanés sans sacrifier la réactivité ou la vitesse d'exécution.

#### Sécurité

La protection des données sensibles est primordiale. L'application doit garantir la confidentialité, l'intégrité et la disponibilité des informations via des mesures robustes comme l'authentification multi-facteurs, le chiffrement des données, et la détection des activités suspectes.

#### Fiabilité

L'application doit offrir une disponibilité continue, avec un minimum de temps d'arrêt. Des mécanismes de sauvegarde et de récupération des données doivent être en place pour assurer la continuité du service en cas de défaillance.

#### Évolutivité

L'architecture de l'application doit permettre une croissance future, facilitant l'ajout de nouvelles fonctionnalités et l'intégration avec d'autres systèmes au fur et à mesure des besoins de l'entreprise.

#### Facilité d'utilisation

L'interface utilisateur doit être intuitive et accessible à tous les niveaux de compétence. Un support complet, comprenant des tutoriels et une documentation claire, doit être mis à disposition pour guider les utilisateurs.

#### Compatibilité

L'application doit être compatible avec une variété de navigateurs web, d'appareils mobiles, et de systèmes d'exploitation, afin de garantir une expérience fluide et cohérente pour tous les utilisateurs.

#### Disponibilité

L'application doit être disponible 24/7 avec des plans de maintenance bien planifiés pour minimiser les interruptions de service.

#### Conformité

Elle doit respecter les régulations locales et internationales, notamment en ce qui concerne le traitement des données et la protection de la vie privée (comme le RGPD en Europe).

### Spécification technique

#### Architecture logicielle

Figure 5 Architecture logicielle du projet

La partie API se base sur une architecture REST qui fournit l’accès aux ressources identifiées par des URIs, qui sont exploitées par le client. La communication entre l’API et le client s’effectue via les méthodes HTTPs standards (get, post, put, delete) et les données échangées sont formatées en JSON.

L'API permet de lier des modèles à des entités de la base de données. En effet, c’est la partie qui met en œuvre la logique de l'application et sert à :

* Gérer la réception des flux de requêtes, les traiter et les renvoyer au client.
* Persister les données de l’application dans la base de données.

La partie client a pour objectif d’assurer l’échange entre l’utilisateur et l’API qui contient la logique de mon application.

La partie persistance est représentée par le SGBD PostgreSQL pour le traitement des données.

#### Technologies et Outils Utilisés

##### Méthodes de conception/modélisation :

###### UML :

Utilisé pour la modélisation des systèmes à travers des diagrammes tels que le diagramme de cas d'utilisation, le diagramme de classes, etc.

###### BDR (Base de Données Relationnelle) :

Conception de bases de données relationnelles pour structurer et organiser les données du projet.

##### Versioning

###### github

Plateforme de collaboration basée sur Git, utilisée pour héberger le code source du projet. GitHub offre des fonctionnalités telles que les pull requests pour réviser et fusionner le code, les issues pour suivre les bogues et les tâches, ainsi que des actions GitHub pour automatiser les processus de CI/CD (Intégration Continue/Déploiement Continu). Grâce à GitHub, l'équipe de développement a pu travailler de manière collaborative et centralisée, tout en assurant la sécurité et la disponibilité du code.

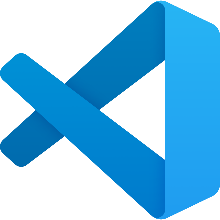
##### Technologies

###### Node.js:

 Un environnement d'exécution JavaScript côté serveur utilisé pour développer le backend du projet. Node.js permet de créer des applications web évolutives et performantes grâce à son architecture non-bloquante et orientée événements.

##### Environnement de développement intégré (EDI)

###### VS Code

Pour le développement de ce projet, j'ai utilisé Visual Studio Code (VSCode) comme environnement de développement intégré (EDI). VSCode est un EDI open-source et très performant, développé par Microsoft. Il offre de nombreuses fonctionnalités telles que le support des extensions, des outils de débogage intégrés, IntelliSense pour les suggestions de code, et une intégration native de Git. Ces fonctionnalités ont grandement facilité mon travail en rendant le développement plus efficace et en assurant une meilleure gestion du code et de la collaboration en équipe.

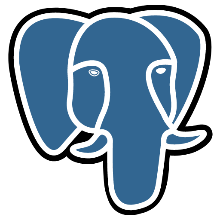
##### Logiciels

###### DBeaver:

 Utilisé pour la gestion et l'administration des bases de données, DBeaver permet de concevoir, interroger, et analyser les bases de données de manière efficace. Il prend en charge de nombreux systèmes de gestion de bases de données, facilitant ainsi l'interaction avec PostgreSQL utilisé dans ce projet.

##### Serveurs de bases de données

###### PostgreSQL:

 Utilisé comme système de gestion de base de données relationnelle (SGBDR) pour stocker et organiser les données du projet. PostgreSQL offre des fonctionnalités avancées telles que la gestion des transactions, les jointures complexes, et une forte conformité ACID, assurant ainsi l'intégrité des données.

##### Framework

Pour la réalisation de ce projet, nous avons utilisé deux frameworks principaux : Angular pour le frontend et NestJS pour le backend.

###### Angular

 Est un framework open-source de développement web créé par Google. Il est conçu pour faciliter la création d'applications web dynamiques et réactives. Angular utilise TypeScript comme langage principal, ce qui permet de bénéficier des avantages de la typage statique et d'une meilleure maintenance du code. Grâce à ses puissants outils et fonctionnalités, Angular nous a permis de développer une interface utilisateur riche et interactive, assurant une expérience utilisateur fluide et réactive. Les capacités de binding bidirectionnel de Angular ont grandement simplifié la synchronisation des données entre le modèle et la vue, tandis que son système de modules a permis une organisation structurée et maintenable du code.

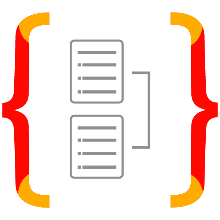
###### NestJS

quant à lui, est un framework backend progressif construit sur Node.js et TypeScript. Il est fortement inspiré par les concepts d'architecture de modules de Angular, ce qui facilite une transition et une intégration harmonieuse entre le frontend et le backend. NestJS adopte une approche modulaire et fournit une architecture solide et évolutive, idéale pour la création d'applications backend robustes et maintenables. Il supporte de nombreux protocoles tels que GraphQL, WebSockets, et bien sûr, REST. NestJS nous a permis de construire une API backend sécurisée et performante, intégrant des fonctionnalités de gestion de la base de données, d'authentification et d'autorisation, ainsi que de communication en temps réel.

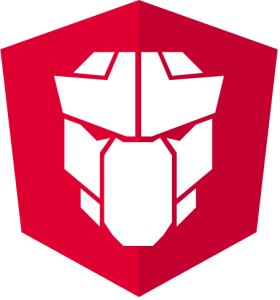
En utilisant Angular et NestJS ensemble, nous avons bénéficié d'une cohérence et d'une synergie accrue entre les couches frontend et backend, facilitant ainsi le développement et la maintenance de l'application globale. Cette combinaison nous a également permis d'assurer une performance optimale, une sécurité renforcée et une évolutivité future pour le Système de Gestion de Stock Avancé (SGSA).

##### Librairies

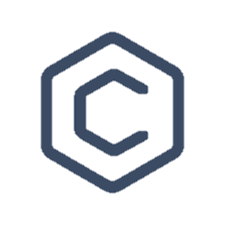
###### TypeORM :

Un ORM (Object-Relational Mapping) pour TypeScript et JavaScript, utilisé avec PostgreSQL pour la gestion des bases de données relationnelles dans le projet. TypeORM simplifie les opérations de base de données en permettant une interaction avec celles-ci via des objets JavaScript/TypeScript.

###### PrimeNG:

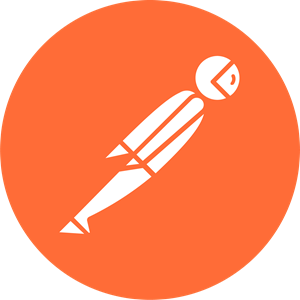
 is a powerful UI component library for Angular that provides a rich set of features and customizable components. It is designed to enhance the development process, enabling developers to create modern, responsive web applications efficiently.

###### CoreUI for Angular:

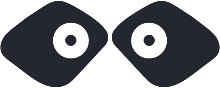
 is a powerful UI component library tailored for Angular applications. With its rich set of features, responsive design, and ease of customization, it enables developers to build modern web applications quickly and efficiently. Whether for admin dashboards or enterprise solutions, CoreUI provides the necessary tools to create a polished user experience.

##### API testing

###### Postman

Postman a été utilisé pour tester et valider les API du projet. Cet outil permet d'envoyer des requêtes HTTP aux différents endpoints du backend afin de vérifier leur bon fonctionnement. Grâce à Postman, nous avons pu tester des opérations telles que les requêtes GET, POST, PUT et DELETE, garantissant ainsi que les réponses du serveur étaient conformes aux spécifications techniques. Postman a également été utilisé pour automatiser les tests de régression afin d'assurer la stabilité des services à travers les différentes versions du développement.

###### Mockoon

Mockoon a été utilisé pour simuler des API en créant des serveurs mock. Cet outil a permis de tester le frontend de manière indépendante, même lorsque le backend n'était pas encore entièrement développé ou disponible. En configurant des réponses personnalisées dans Mockoon, nous avons pu reproduire différents scénarios d'utilisation et s'assurer que le frontend se comportait correctement dans diverses conditions sans avoir besoin d'accéder aux serveurs de production ou de développement. Cela a grandement accéléré les phases de développement et de test.

##### Langages de programmation

###### TypeScript

**Description** : TypeScript est un sur-ensemble de JavaScript développé par Microsoft, apportant des fonctionnalités de typage statique et de programmation orientée objet. Il est compilé en JavaScript, garantissant ainsi la compatibilité avec les navigateurs modernes.

**Utilisation** : Utilisé principalement pour développer la logique métier et les composants de l'application frontend. Il facilite la gestion de grandes bases de code et améliore la qualité et la maintenabilité du code.

**Avantages** : Offre une vérification de type à la compilation, des outils de développement améliorés, et une meilleure structuration du code grâce aux classes et interfaces.

###### HTML

****

**Description** : HTML (HyperText Markup Language) est le langage standard pour la création de pages web. Il définit la structure et le contenu des pages en utilisant des balises.

**Utilisation** : HTML a été utilisé pour créer la structure des pages et des composants de l'application. Il définit les éléments tels que les en-têtes, les paragraphes, les formulaires, et les liens.

**Avantages** : Permet de structurer le contenu de manière hiérarchique et sémantique, facilitant ainsi l'organisation et l'affichage des données sur le web.

###### **SCSS**

**Description** : SCSS (Sassy CSS) est une syntaxe de Sass, un préprocesseur CSS. Il permet d'utiliser des variables, des mixins, des fonctions et des boucles pour écrire du CSS plus maintenable et modulaire.

**Utilisation** : SCSS a été utilisé pour styliser l'application en créant des feuilles de style plus dynamiques et réutilisables. Il facilite la gestion des styles complexes grâce à des fonctionnalités avancées comme l'imbrication et les variables.

**Avantages** : Offre une meilleure organisation et réutilisation des styles, permet de créer des styles complexes de manière plus efficace et améliore la maintenabilité du code CSS.

Ce texte devrait donner une vue d'ensemble complète des langages et technologies que tu as utilisés pendant ton stage.

##### Outils de prototypage

###### Balsamiq

 Pour la création de maquettes et de prototypes, nous avons utilisé Balsamiq. Cet outil permet de concevoir rapidement et efficacement des wireframes pour les interfaces utilisateur, facilitant ainsi la visualisation et la validation des concepts avant de passer à la phase de développement. Balsamiq offre une interface intuitive et des éléments préconçus qui simplifient le processus de création de maquettes, contribuant ainsi à une meilleure communication des idées et des exigences entre les membres de l'équipe.

# Étude Conceptuelle

### Diagramme de flux

Ce diagramme de flux offre une visualisation claire et structurée des étapes nécessaires pour gérer efficacement les réceptions de stock, incluant les contrôles de qualité et les mises à jour des inventaires. Il permet de s'assurer que chaque étape est suivie de manière rigoureuse et que les stocks sont maintenus avec précision et transparence.



Figure 6 Diagramme de flux

### Diagramme de Cas d'Utilisation (Use Case Diagram) :

#### Diagramme de gestion de consultation

Ce diagramme illustre le processus de consultation des stocks disponibles en termes de quantité et de valeur. Le magasinier est l'acteur principal qui accède aux informations de stock. Deux sous-processus sont inclus :

* + Spécifier les dimensions du stock à inclure dans l'image de stock.
  + Spécifier le statut du stock à inclure dans l'image de stock. Ces sous-processus permettent au magasinier de personnaliser la vue des stocks selon les besoins spécifiques de l'entreprise.



Figure 7 use case Diagramme de gestion de consultation

#### Diagramme gestion journaux de transfert

Ce diagramme décrit le processus de création et de gestion des journaux de transfert internes. Le magasinier crée d'abord l'entête du journal de transfert. Ensuite, les lignes de journal sont saisies par articles et dimensions. Une fois le journal de transfert créé, il peut être reçu par le service de gestion des stocks, qui vérifie la disponibilité des stocks par dimension et met à jour les stocks en quantité et en valeur. Ce processus assure une traçabilité et une gestion efficace des mouvements de stock internes.



Figure 8 use case Diagramme gestion journaux de transfert

#### Diagramme gestion ordre de transfert

Ce diagramme représente un processus complet de gestion des ordres de transfert, divisé en trois parties :

1. **Réception d'ordre de transfert :**
   * Le magasinier reçoit l'ordre de transfert et saisit les lignes reçues par article et dimension de stockage.
   * Mise à jour du stock en valeur et quantité.
2. **Demande de transfert :**
   * Le demandeur de transfert saisit les lignes de demande de transfert par article et dimension de stock.
   * L'expéditeur de transfert saisit les lignes d'annulation et de supplément par article et dimension de stock.
   * Validation ou rejet des lignes d'annulation ou de supplément par le demandeur de transfert.
3. **Expédition d'ordre de transfert :**
   * L'expéditeur de transfert saisit les lignes d'ordre de transfert selon les articles et dimensions demandées.
   * Vérification de la disponibilité de stock par dimension de stock.
   * Expédition de l'ordre de transfert.
   * Mise à jour des stocks en qualité et valeur.

.



Figure 9 use case Diagram gestion ordre de transfert

#### Diagramme gestion d’inventaire

Ce diagramme de cas d'utilisation représente les différentes étapes impliquées dans le processus de comptage de stock. Il détaille les interactions entre le Magasinier et le Système pour la gestion des stocks. Voici les principaux cas d'utilisation et leurs relations :

**Création d'un journal de comptage par dimension cartographique** : Le Magasinier crée un journal de comptage en spécifiant des dimensions telles que l'entrepôt, la zone, l'allée et l'emplacement.

**Spécifier les articles à compter** : Le Magasinier sélectionne les articles spécifiques qui doivent être comptés.

**Lancer le comptage** : Une fois les articles spécifiés, le Magasinier lance le processus de comptage.

**Comptage des articles à compter par dimension de stockage** : L'Agent de comptage effectue le comptage physique des articles selon les dimensions spécifiées.

**Validation de journal de comptage** : Le Magasinier valide les résultats du comptage.

**Calcul des écarts de stock par dimension de stock** : Le Magasinier calcule les écarts entre le stock théorique et le stock physique.

**Ajustement de stock en quantité et valeur** : Le Magasinier ajuste les stocks en fonction des écarts constatés.



Figure 10 use case Diagramme gestion d’inventaire

#### Diagramme gestion de réception

Ce diagramme de cas d'utilisation illustre le processus de réception de stock, depuis la sélection du bon de commande jusqu'à la mise à jour des stocks après réception. Les principaux acteurs sont le Gestionnaire des stocks et le Qualiticien. Voici les principaux cas d'utilisation et leurs relations :

**Sélectionner le bon de commande à recevoir** : Le Gestionnaire des stocks choisit le bon de commande à traiter.

**Saisir les lignes de bon de réception en quantité et dimension de stock** : Le Gestionnaire des stocks enregistre les détails des articles reçus.

**Création de bon de réception en statut Brouillon** : Un bon de réception provisoire est créé pour la validation.

**Vérification de quantité restante au bon de commande** : Le Gestionnaire des stocks vérifie les quantités reçues par rapport au bon de commande.

**Vérification d'obligation de contrôle de qualité par ligne de BC** : Le Qualiticien contrôle la qualité des articles reçus.

**Création des ordres de qualité par article** : Des ordres de qualité sont créés pour chaque article nécessitant un contrôle.

**Remplir les résultats de contrôle de qualité** : Le Qualiticien enregistre les résultats du contrôle de qualité.

**Validation de bon de réception** : Le Gestionnaire des stocks valide le bon de réception après vérification.

**Mise à jour de stock en quantité et valeur** : Les stocks sont mis à jour en fonction des réceptions validées.

**Annulation des lignes de BC non conforme** : Les lignes de bon de commande non conformes sont annulées.



Figure 11 use case Diagramme gestion de réception

### Activity diagram

#### Gestion de consultation de stock disponible

Le diagramme de consultation de stock décrit de manière claire et concise le processus permettant de consulter les informations de stock dans un système de gestion des stocks. Ce diagramme est structuré pour illustrer les étapes successives nécessaires à la réalisation d'une demande de consultation de stock, depuis l'initiation de la demande jusqu'à l'affichage des résultats.



Figure 12 Activity Diagram Gestion de consultation de stock disponible

#### Gestion de réception

Le diagramme d'activité ci-dessus illustre le processus de réception des commandes Ce diagramme fournit une vue détaillée des étapes impliquées dans la réception, la vérification, et la mise à jour des stocks suite à une commande.



Figure 13 Activity Diagram Gestion de réception

#### Gestion d’inventaire

Ce diagramme d'activité illustre le processus de comptage de stock dans un entrepôt. Le processus commence par la création d'un journal de comptage, où les détails comme l'entrepôt, la zone et l'allée sont spécifiés. Ensuite, les articles à compter sont sélectionnés. Si le journal est ajusté, les transactions de stock pour ces articles sont bloquées. Les lignes de comptage, incluant la valeur unitaire, la quantité et les dimensions de stock des articles, sont saisies. Après la clôture du journal de comptage, les écarts entre le stock physique et théorique sont ajustés. Enfin, les transactions de stock pour les articles bloqués sont libérées.



Figure 14 Activity diagramme Gestion d’inventaire

#### Gestion journaux de transfert

Ce diagramme d'activité montre le processus de transfert de stock au sein du même entrepôt. Le processus commence par la création d'un journal de transfert. Si la demande de transfert n'est pas encore clôturée, les lignes de journal, y compris les articles, les quantités et les dimensions de stockage, sont saisies. Une fois le journal de transfert clôturé, la disponibilité du stock est vérifiée. Si le stock est disponible, les valeurs de stock et la disponibilité du stock sont mises à jour. Si le stock n'est pas disponible, le processus de transfert doit être réévalué.



Figure 15 Activity Diagram gestion journaux transfert

#### Gestion ordre de transfert

Ce diagramme d'activité décrit le processus de gestion des ordres de transfert de stock entre deux entrepôts. Le processus débute par la création d'une demande de transfert d'un entrepôt demandeur à un entrepôt préparateur. Si la demande de transfert n'est pas encore soumise, les lignes de la demande, incluant les articles et les quantités, sont saisies. Une fois la demande soumise, les lignes de demande sont consultées. Si la ligne de demande est satisfaite à 100% et que le stock est disponible, la demande de transfert est partiellement ou totalement satisfaite selon les articles, les quantités et les dimensions de stock. Enfin, la mise à jour de la valeur du stock et de la disponibilité du stock est effectuée, et l'ordre de transfert est clôturé.

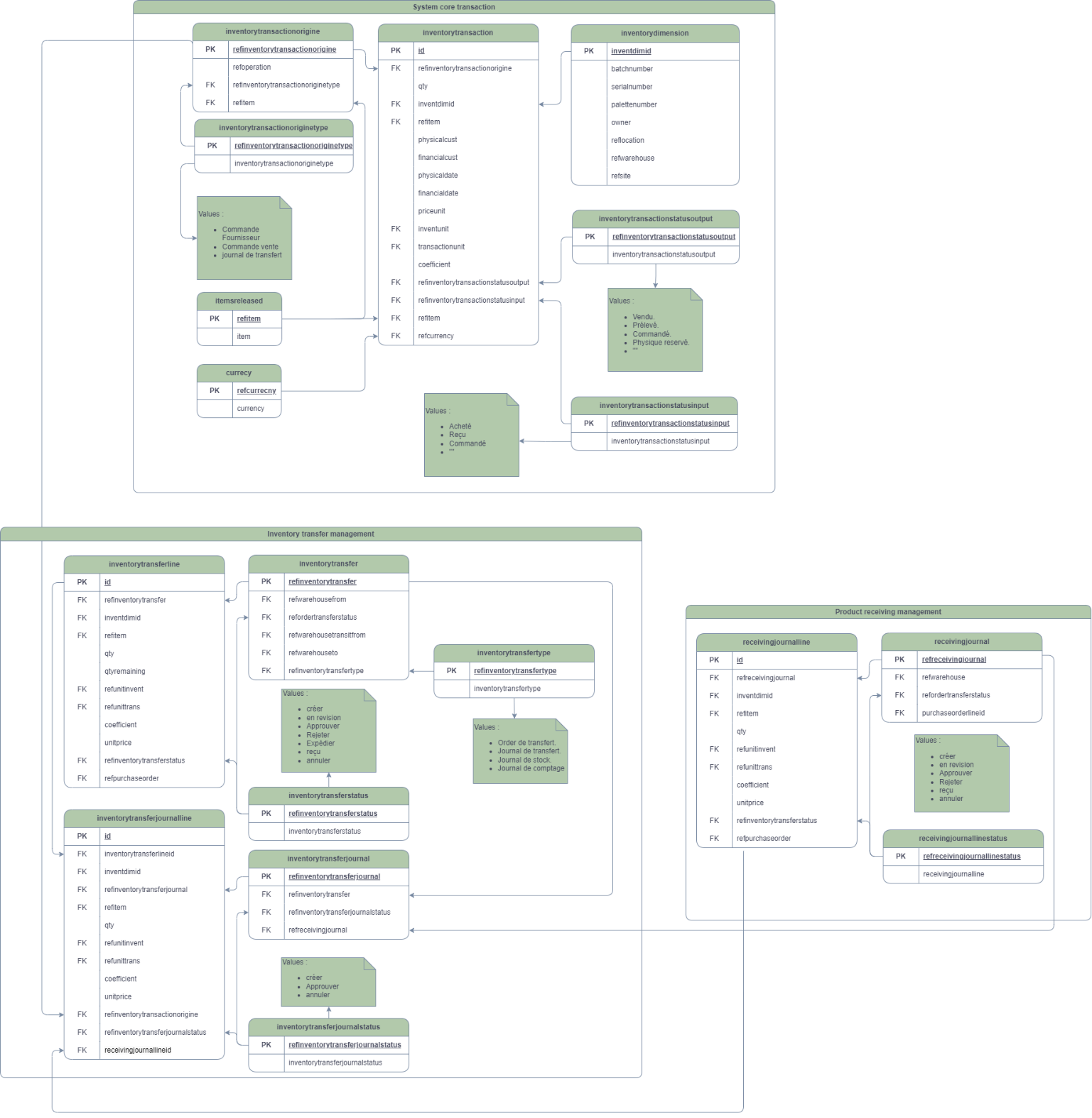


Figure 16 Activity Diagram Gestion ordre transfert

### Diagramme de Classes (Class Diagram) :

### Diagramme de Séquence (Sequence Diagram) :

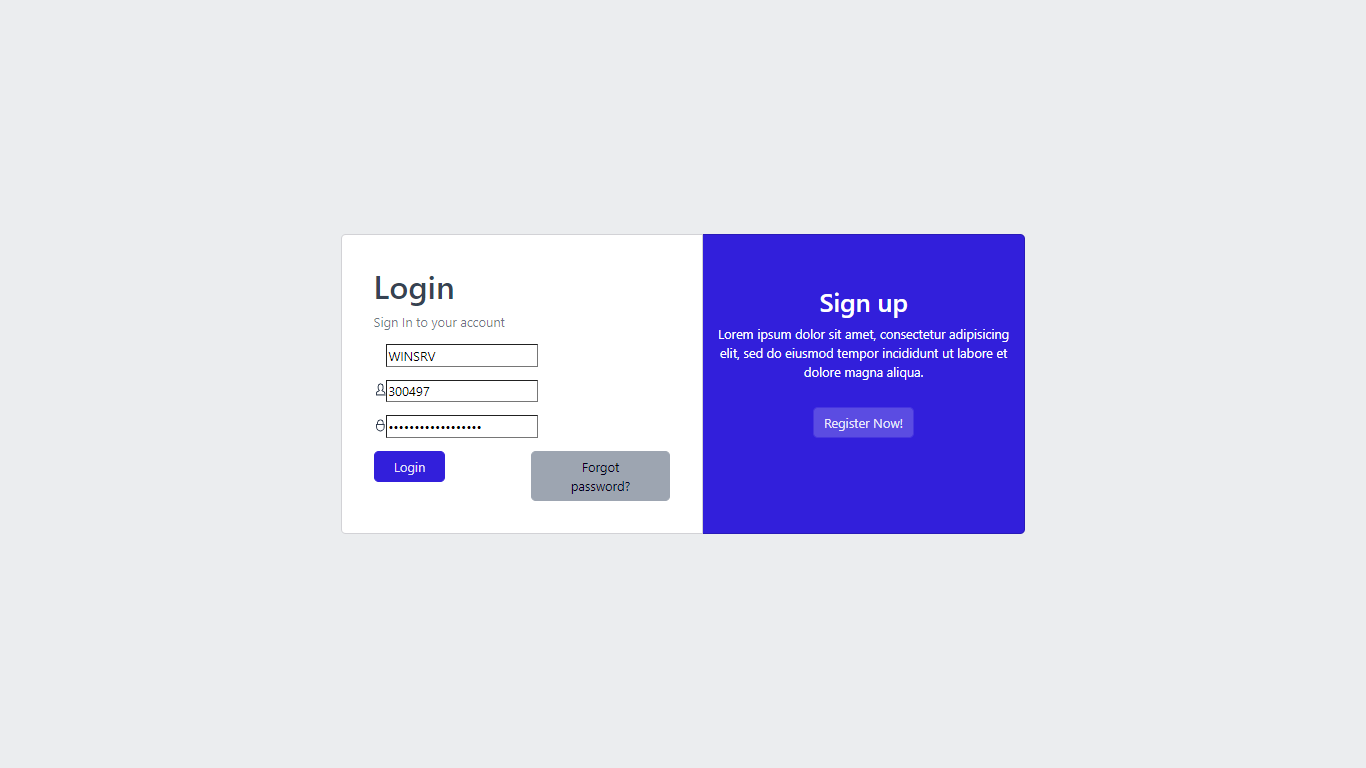
### Diagramme Entité-Relation :



# Mission et travaux réalisés

## Reialisation frontend

### Login page



password

user

Reference de société

Figure 17 page de login

Dans ce page l’utilisateur doit saisir la référence de la société et leur nom d’utilisateur et mot de passe, dans cette étape notre encadrent donne un Token[[1]](#footnote-1) pour le placer dans le logiciel Mockoon pour être connecter à l’application dans la partie de frontend

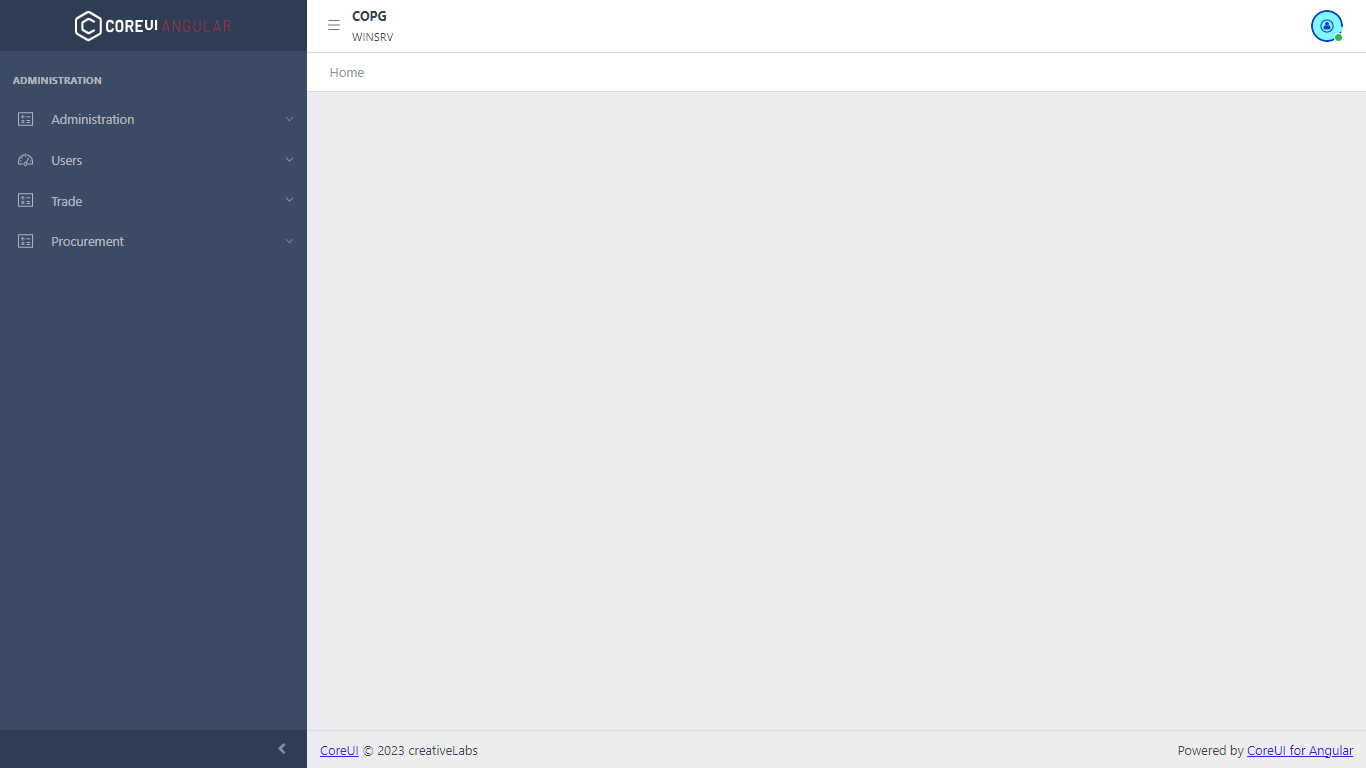


Figure 18 page d'accueil

La page d’accueil consiste juste pour les éléments essentiels dans l’application quelle sont :

* Sidebar : pour organiser les diffèrent linges de navigation d’administrassions
* Header
* footer

### Modele des items

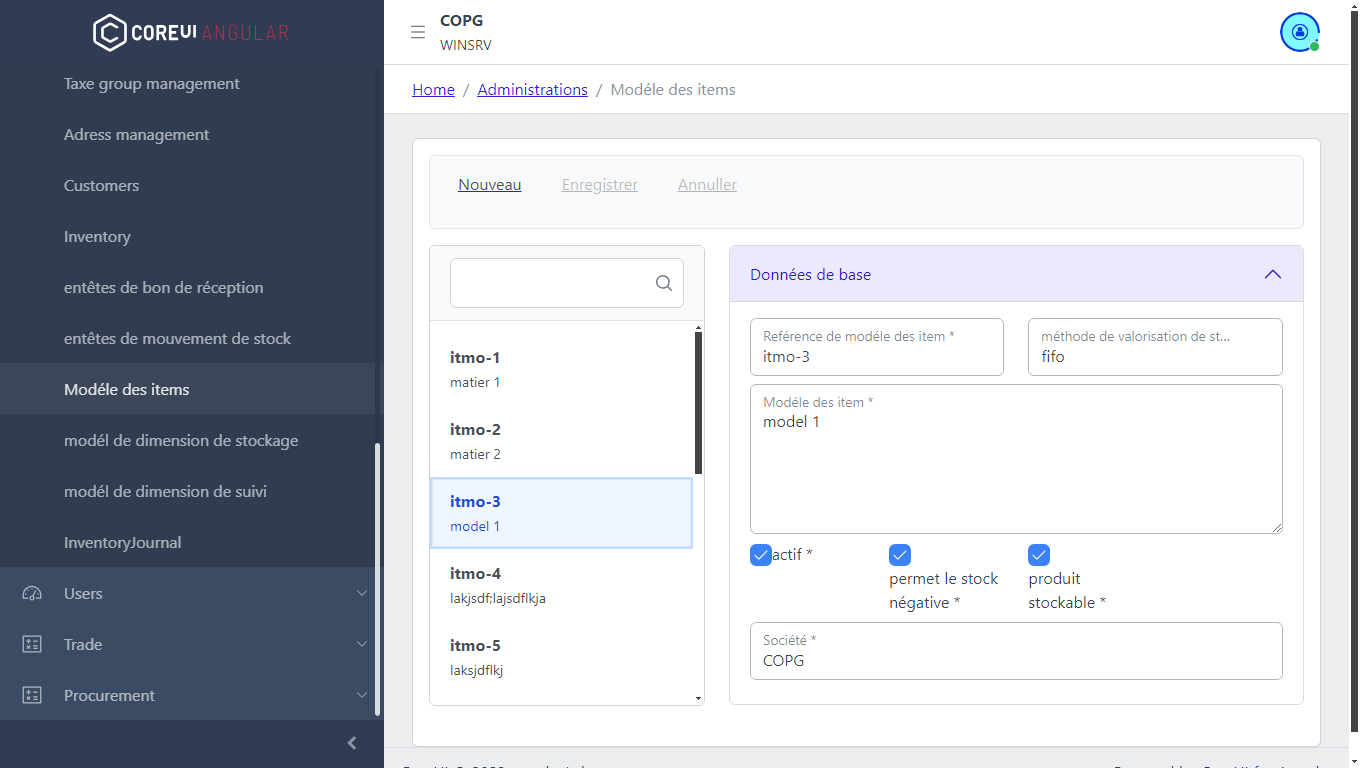
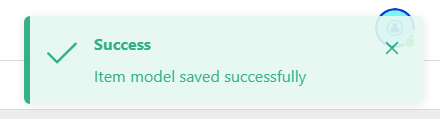
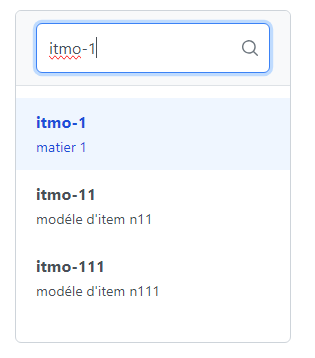


Figure 19 page de model d'item

Dans cette interface en peux gère les model des items en haut de l’en trouve un Megamenu[[2]](#footnote-2) qui consiste trois bottons :

* Nouveau : pour entre dans le mode de crée un nouveau model d’item, dans ce mode en donne le droit de saisir les données de référence de modèle d’item, méthode de valorisation de stock, modèle d’item et de cliquer sur les bottons de choit mems pas de changer la valeur de société car il a une relation avec l’utilisateur actuelle
* Enregistrer : pour soumettre le formulaire pour être enregistre dans la base de données et afficher ci l’enregistrement et fate avec succès ou pas
* Annuler : pour annuler la saisir sortir de mode de création dans ce mode l’utilisateur n’a pas le droit de saisir au modifier les données dans le formulaire la même chose se produit en cliquant sur un modèle d'article dans la liste

A gouache en a une liste qui afficher les model des items qui déjà existe dans la base de données avec la possibilité de rechercher un item grâce a la barre de recherche, la rechercher se déroule en temps réel.

Lorsque l’utilisateur clique sur un modèle d’article, le formulaire à droit sera rempli avec les informations sur ce modèle d’article comme indiqué avec l’itmo-3 dans la figure précédente

### Modél de dimension de stockage

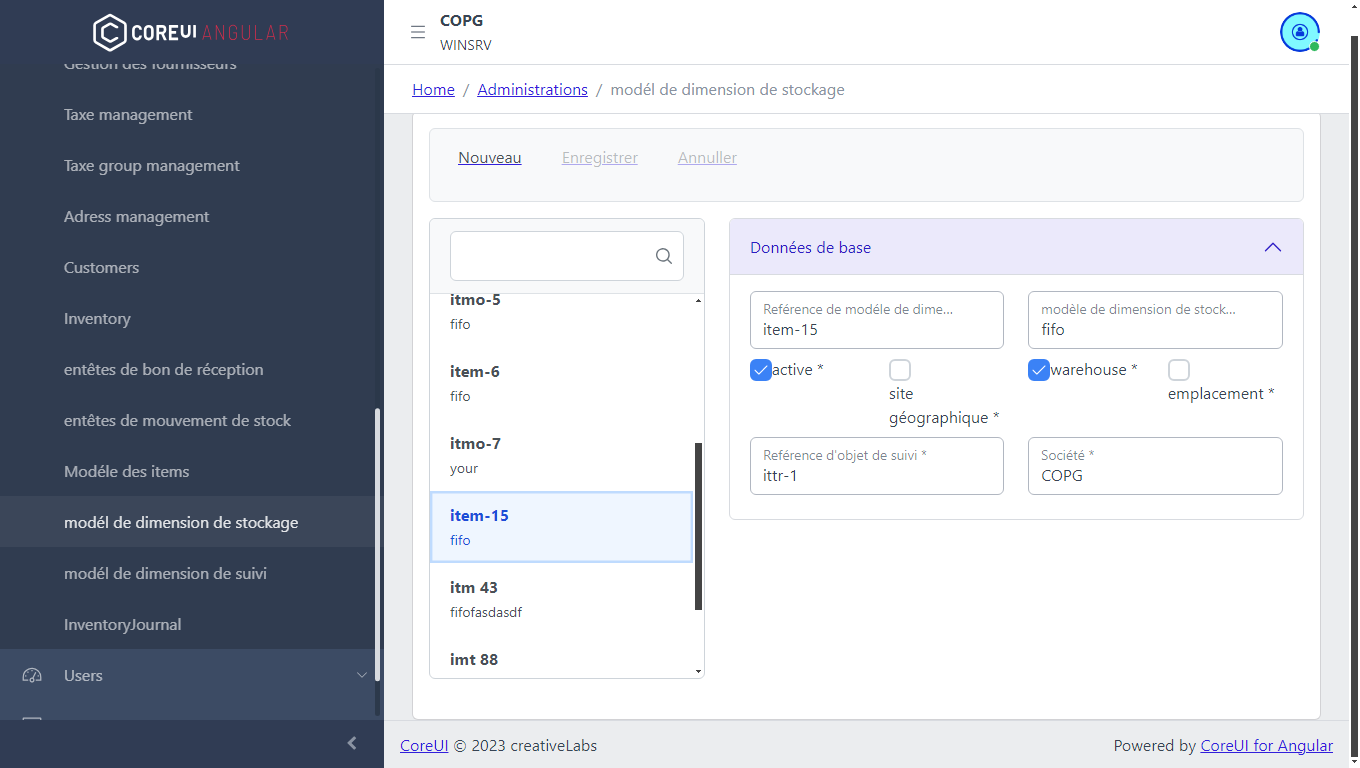


Figure 20 12. Model de dimension de stockage

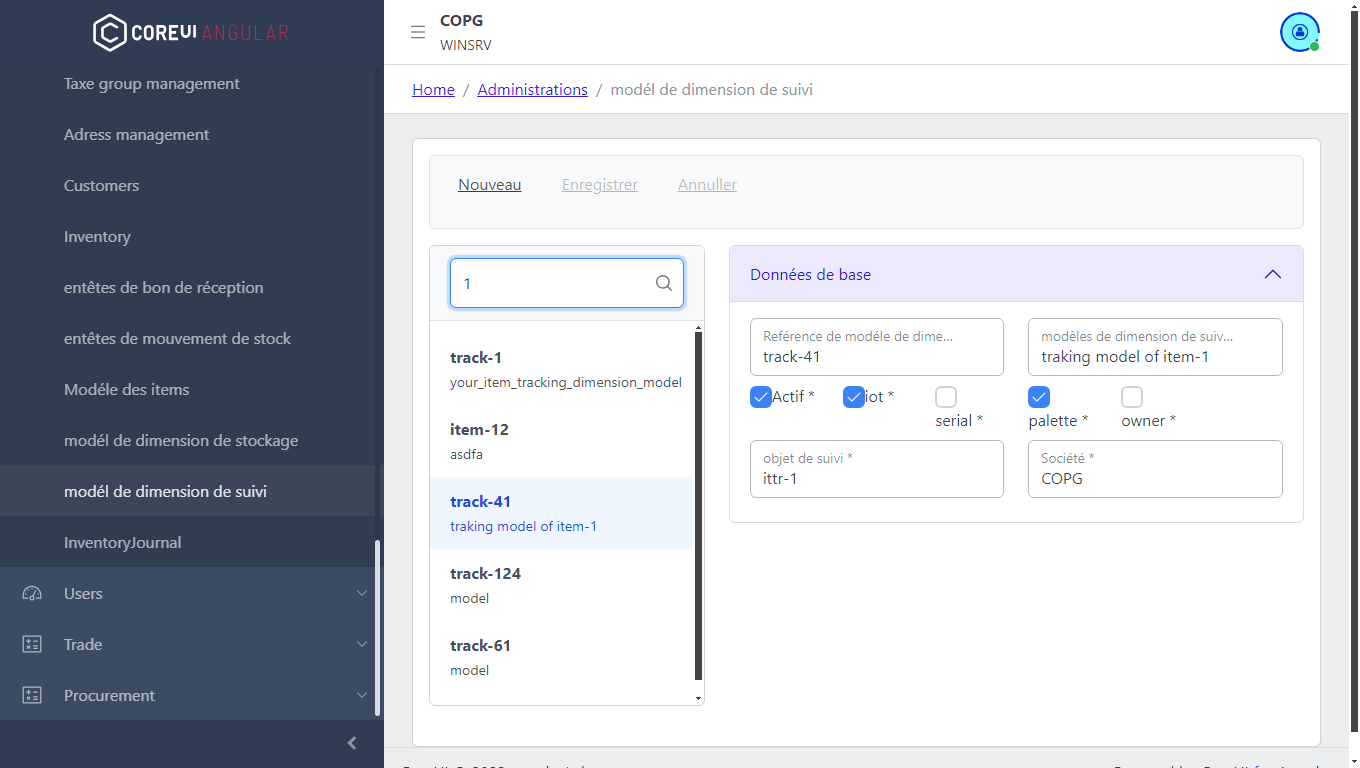
Dans cette interface en a la même chose que modèle d’article : un Megamenu avec les mêmes bottons qui faire le même rôle et une liste des modèles de dimension de stockage qui a aussi la possibilité de la recherche et sélectionner un modèle pour l’afficher. La différence c’est le formulaire de données de base :

Dans ce modèle en a le référence de modèle, le modèle, des choix :

* S’il est actif
* Si a un site géographique
* Si a un entrepôt
* Si a un emplacement

Le référence d’objet de suivi et la société

### Modele de dimension de suivi



Dans ce cas, l'utilisateur recherche un modèle qui porte le numéro 1 dans sa référence et sélectionne le modèle track-41 pour voir ses informations.

Dans cette interface en a la même chose que modèle d’article et model de dimension de stockage : un Megamenu avec les mêmes bottons (nouveau, enregistre, et annuler) qui faire le même rôle et une liste des modèles de dimension de suivi qui a aussi la possibilité de la recherche et sélectionner un modèle pour l’afficher. La différence c’est le formulaire de données de base :

Dans ce modèle en a le référence de modèle, le modèle, des choix :

* S’il est actif
* Si a un lot
* Si a un serial
* Si a un palet
* Si a un propriétaire

Le référence d’objet de suivi et la société

### Bon de réception

#### Liste des entêtes des bon de réception

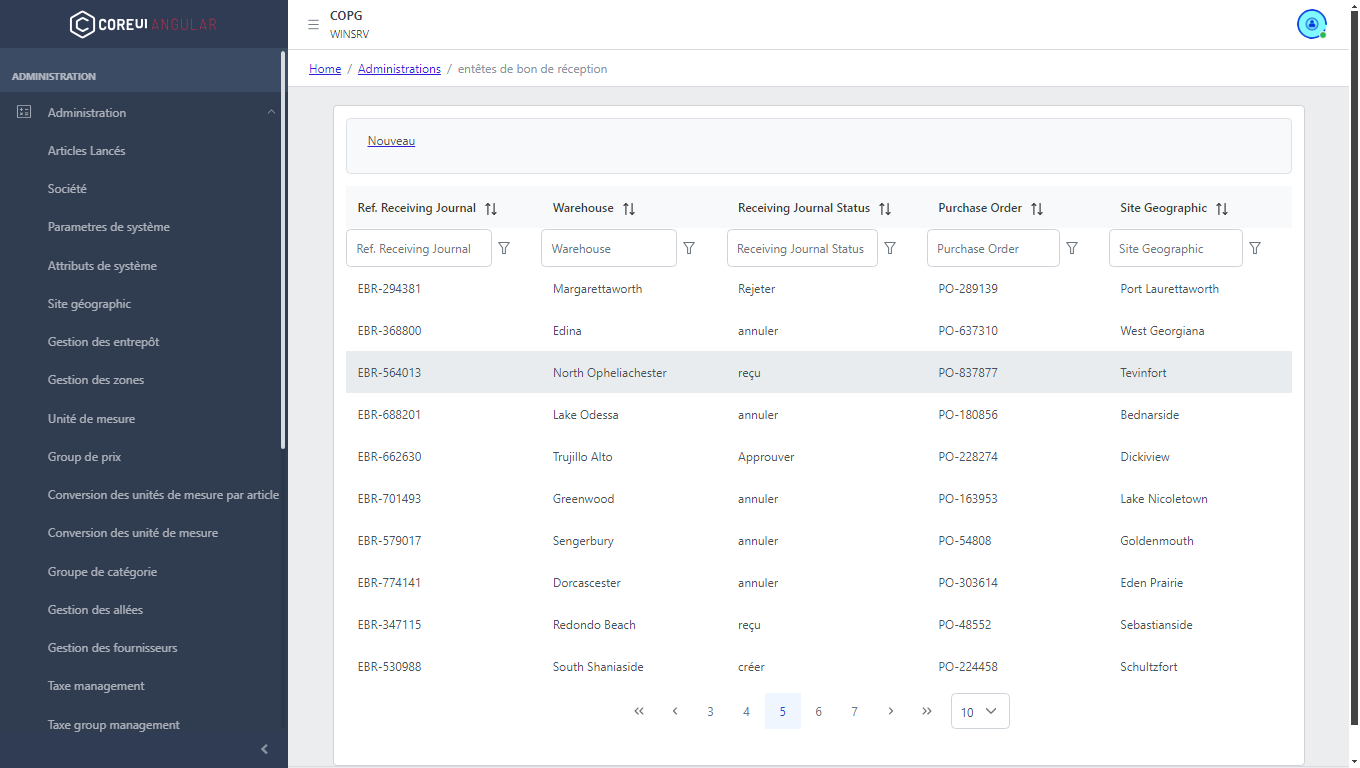


Figure 21 les entêtes des bons de réception

Dans cette interface on a la liste des entêtes des bons de réception.

En haut en a Megamenu avec un seul Botton qui va rediriger ver une interface pour ajouter un nouvel entête de bon de réception en bas on a un tableau qui afficher la liste des entêtes avec des information :

* Référence de l’entête de bon de réception
* L’entrepôt
* Le statut
* Bon de commande
* Le site géographique

Chacune de ces colonnes peut être triée par ordre croissant ou décroissant si l'on clique sur le label d'une colonne

Nous avions également la possibilité de rechercher dans la liste en utilisant différents types de filtrage pour chaque colonne (Commence par, Contient, Ne contient pas, Se termine par, Égal à, et Pas gal) avec le bar de rechercher en haut de chaque colon ne

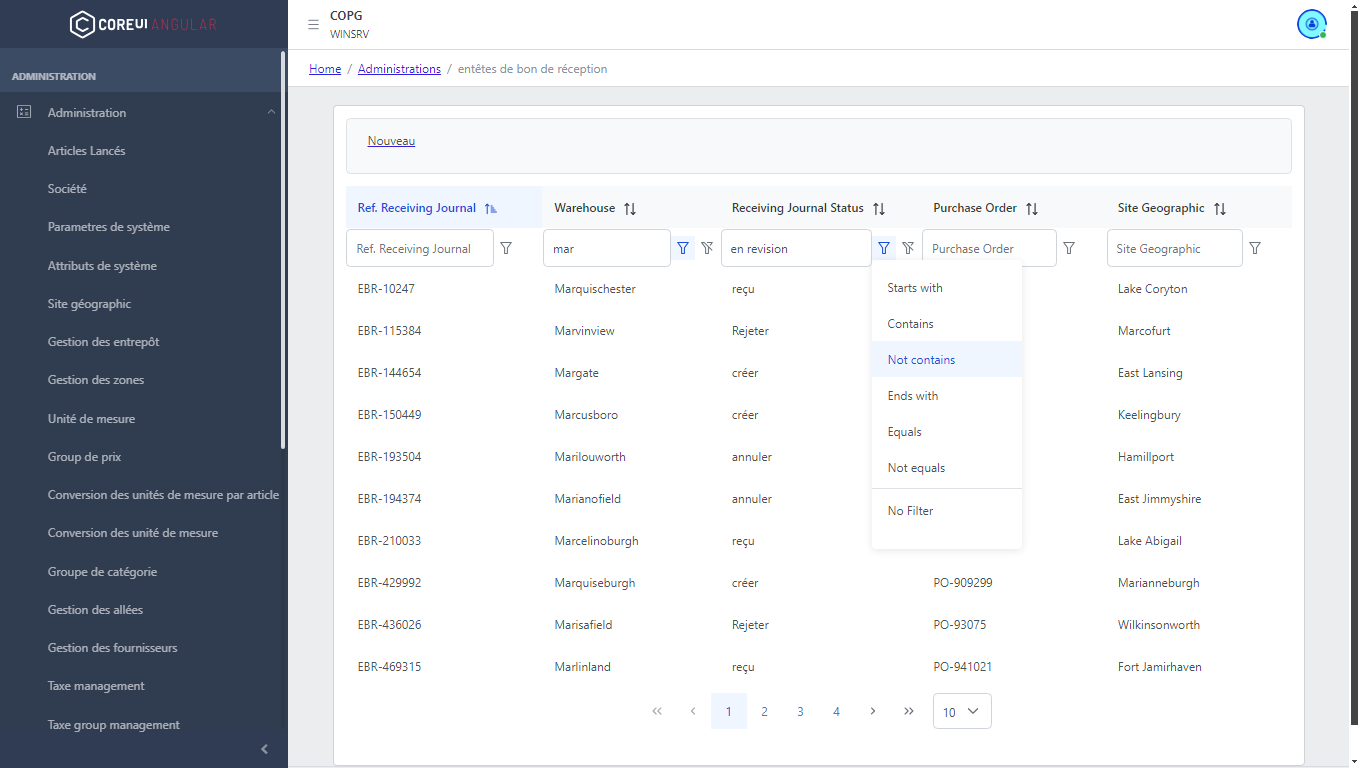


Figure 22 List des entêtes de bon de réception avec filtrage

Dans cet exemple en a trie la liste avec ordre croissant dans les colonnes de référence et en a fait un filtrage pour afficher les entêtes avec un entrepôt qui commence par MAR et un statu qui n’a pas en révision

#### Gère ou crée Nouveau entête

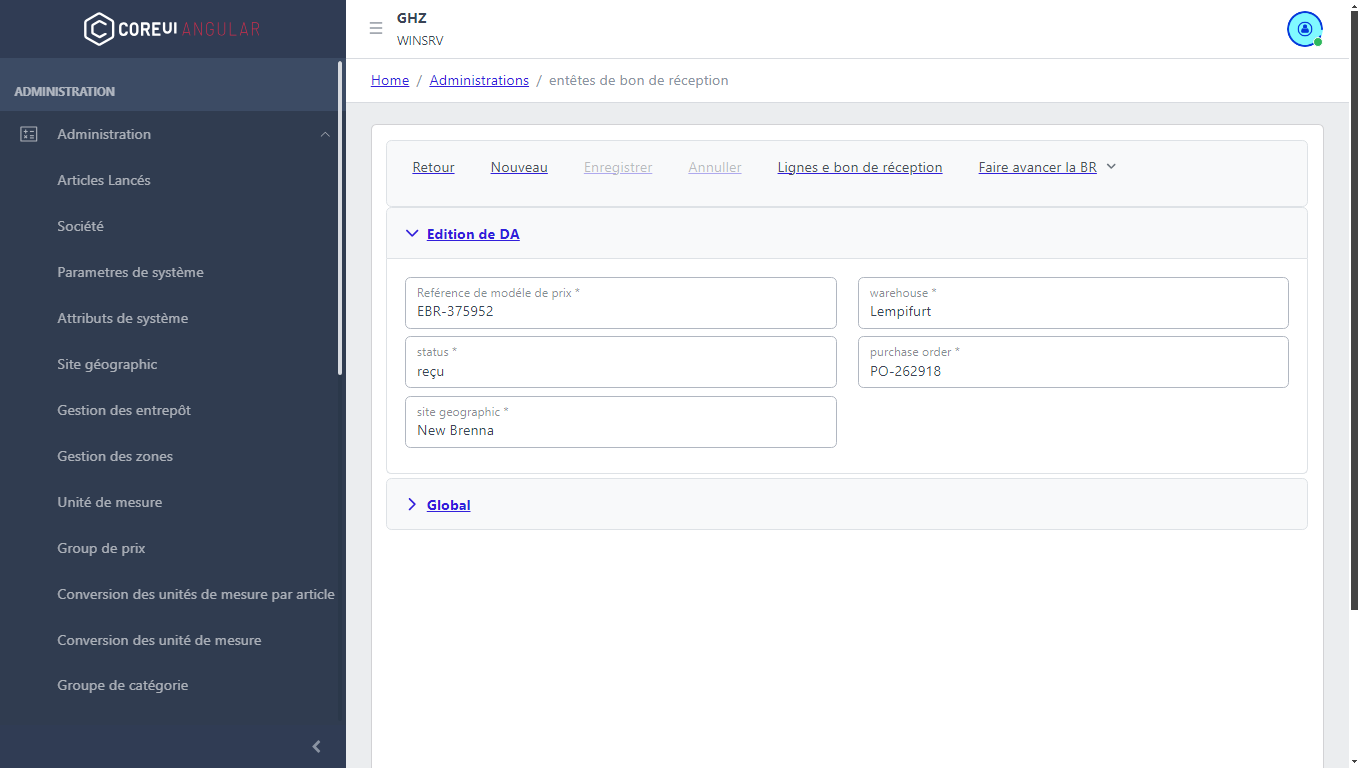
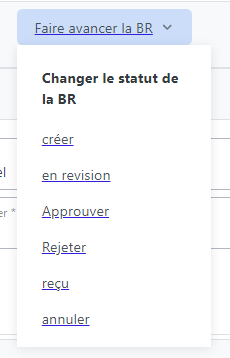


Figure 23 interface d'un entête de bon de réception

Après que l'utilisateur clique sur le bouton nouveau dans la liste des entêtes des bons de réception, il sera redirigé vers cette interface mais le formulaire sera vide. L'alternative pour obtenir cette interface est de cliquer sur un entête d'un BR[[3]](#footnote-3), il sera redirigé vers cette interface avec les informations sur cet entête de BR

Megamenu dans cette interface a les boutons suivants :

* Retour : bouton qui redirige à l’interface précédant (interface des entêtes de BR)
* Nouveau : bouton pour entre dans le mode de création
* Enregistre : pour valider et soumettre le formulaire
* Annuler : pour sortir le mode de création
* Lignes de bon de réception : bouton pour accéder à une interface d’une liste des lignes de ce BR
* Faire avancer le BR : une liste déroulante avec le rôle de changer le statu de ce BR après être traiter par le server de backend

#### Linges de bon de réception

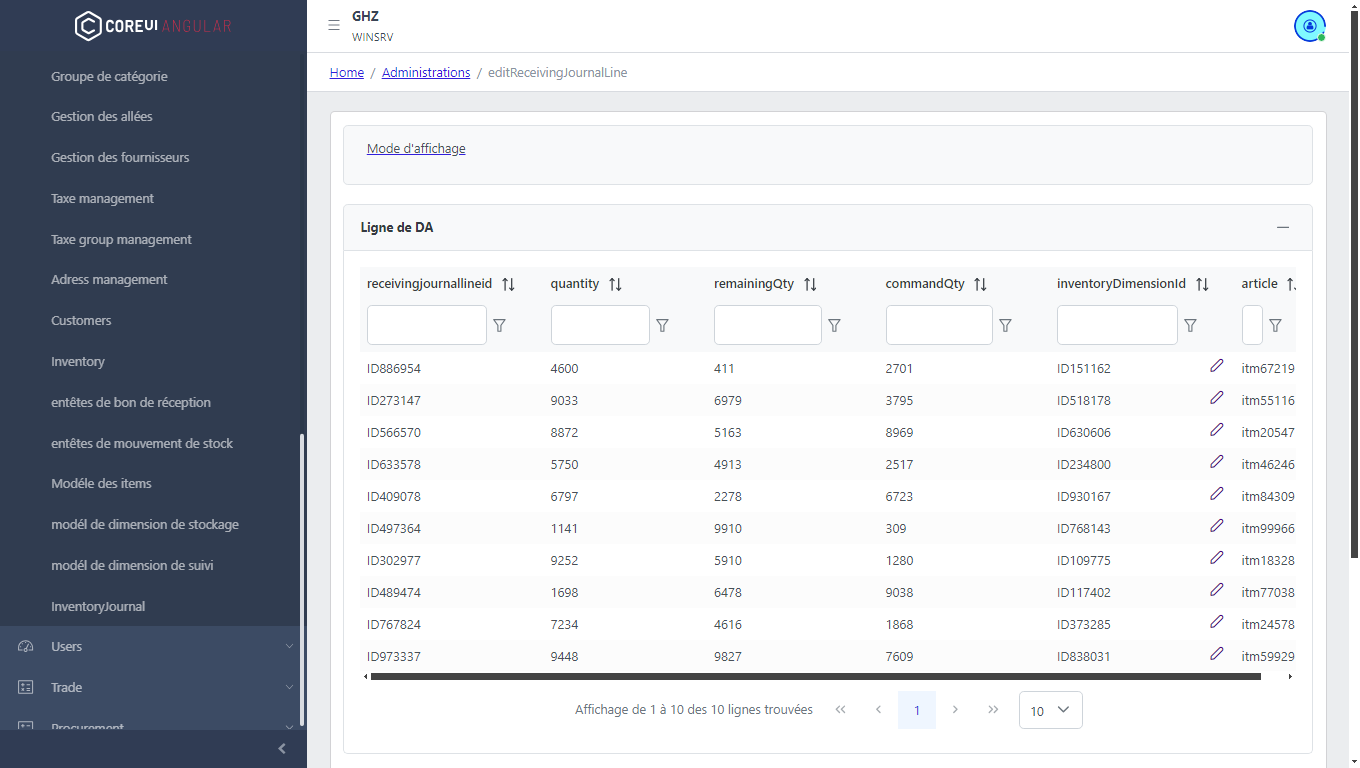


Figure 24 interface des Linges de bon de réception

Après cliquer sur le bouton : Lignes de bon de réception, l’utilisateur va être rediriger vers cette interface avec un seule bouton dans la Megamenu qui et Mode d’affichage pour le moment ne fait rient. Après on a un tableau avec une liste des linges de BR le tableau peut également trier et filtrer comme le tableau précédent des entêtes des BR

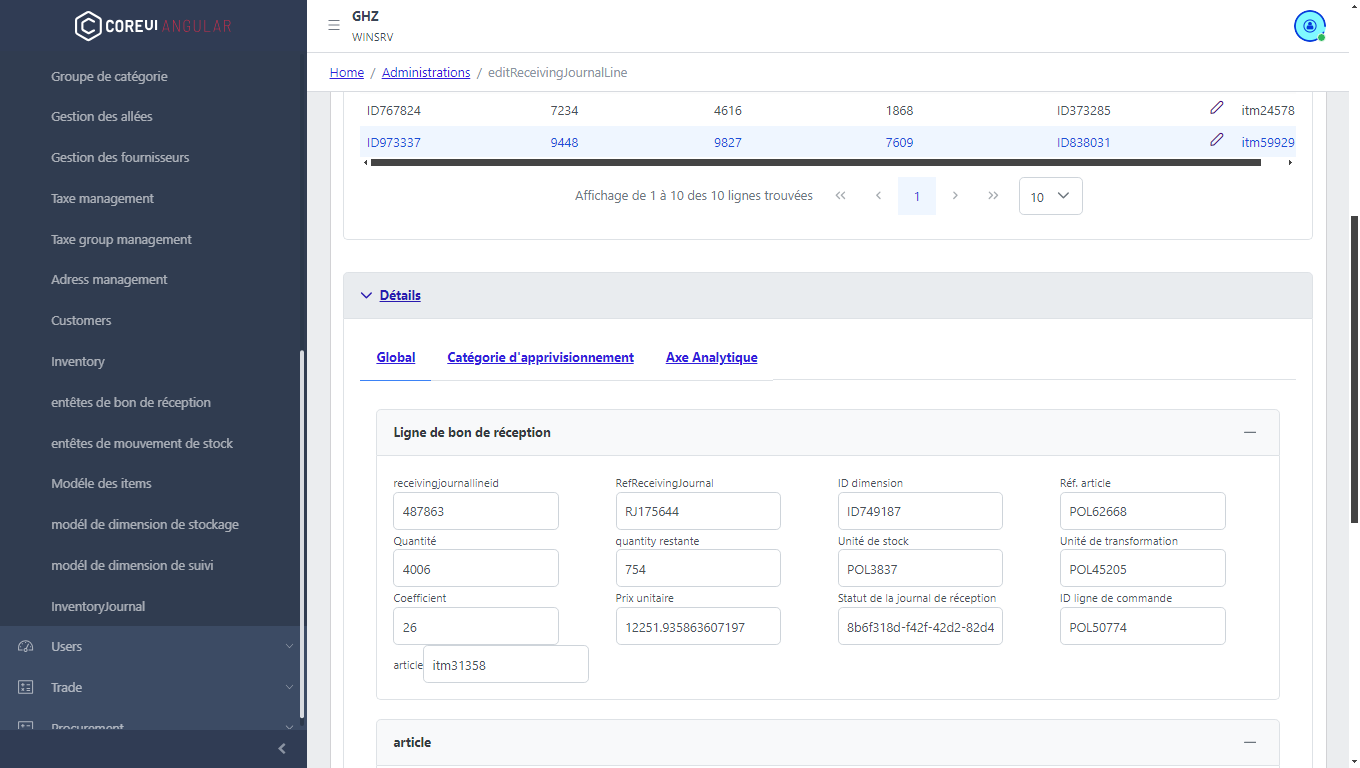


Figure 25 affichage de tous les information un linge de BR

Si l’utilisateur sélectionner un linge de BR un formulaire va être aperçu en bas dans les détails dans catégorie globale avec toutes les informations de ce linge car dans le tableau en n’afficher pas que les informations importantes

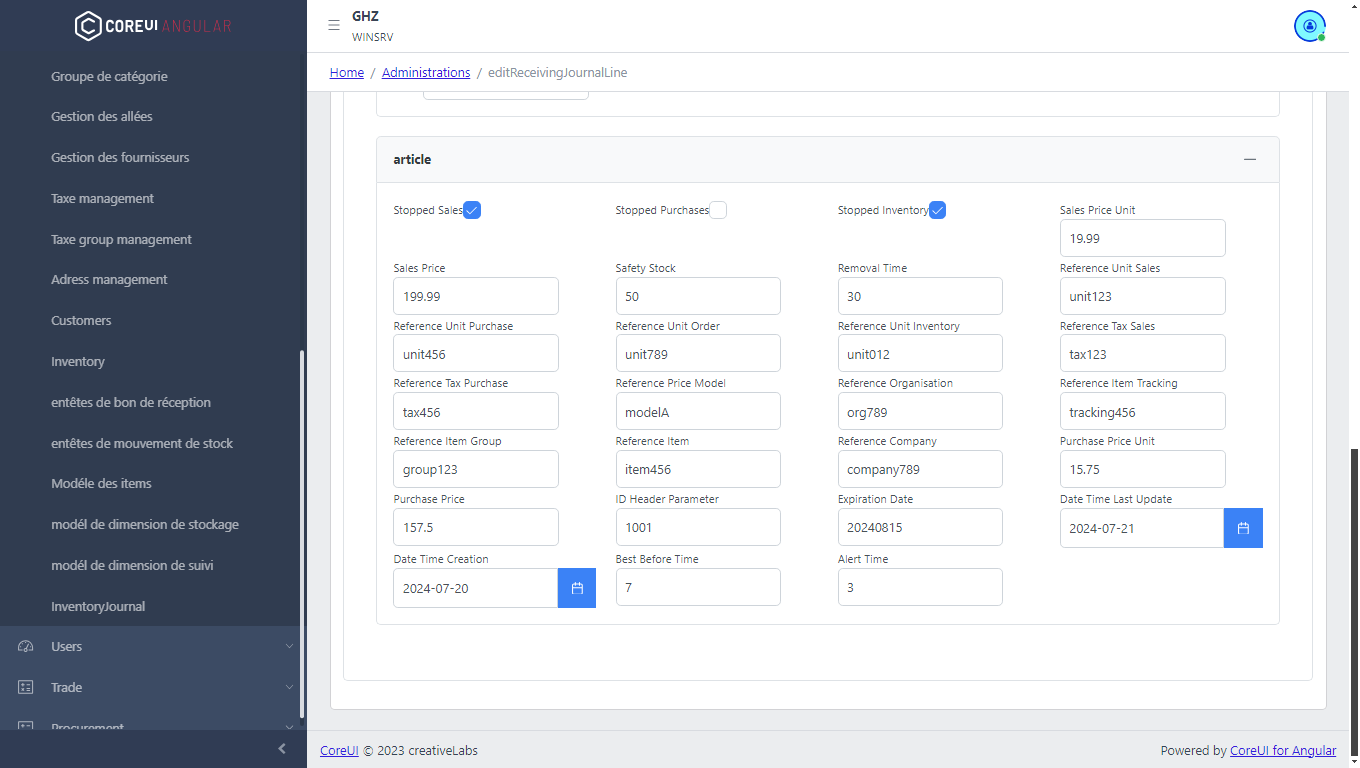
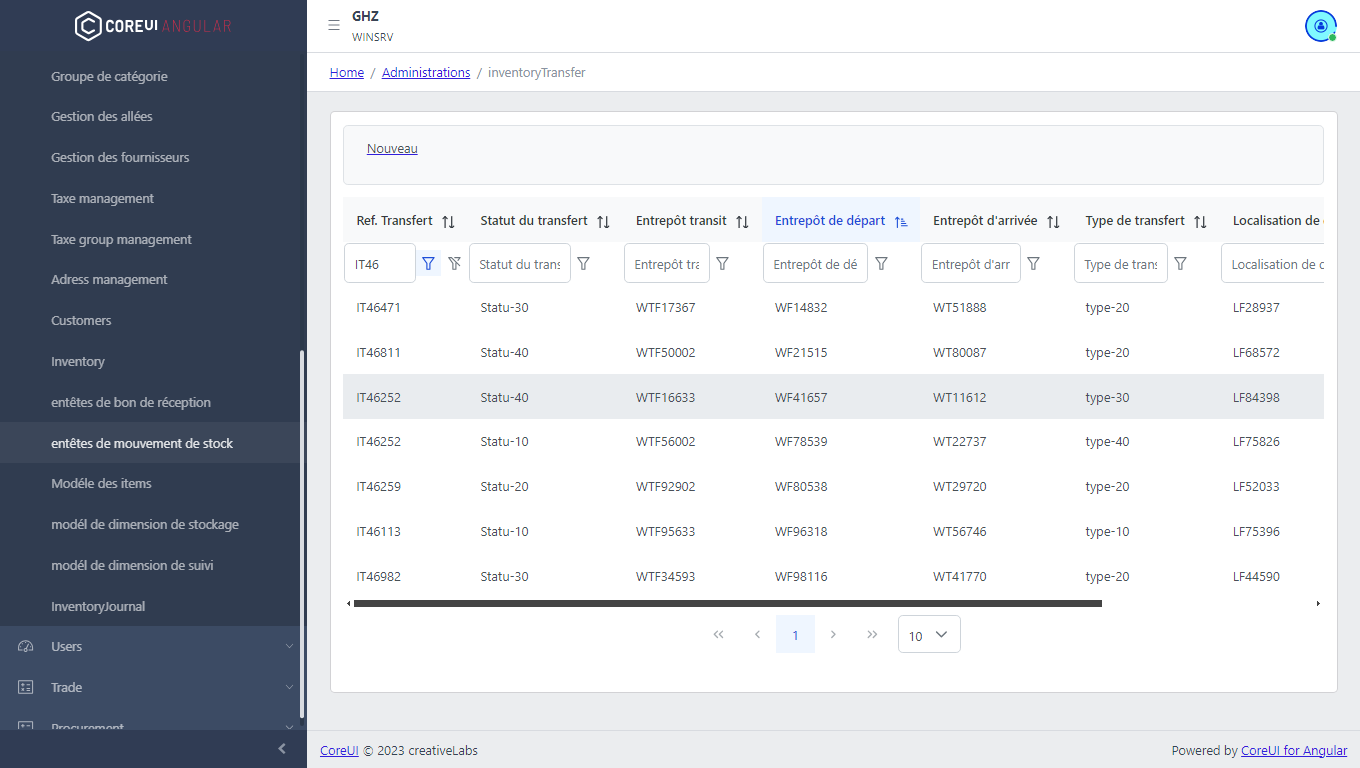


Figure 26 affichage les informations d'article d’un linge de BR

Après la sélectionner un linge de BR aussi un formulaire d’article aussi apparue avec tous les informations d’article

### Mouvement de stock

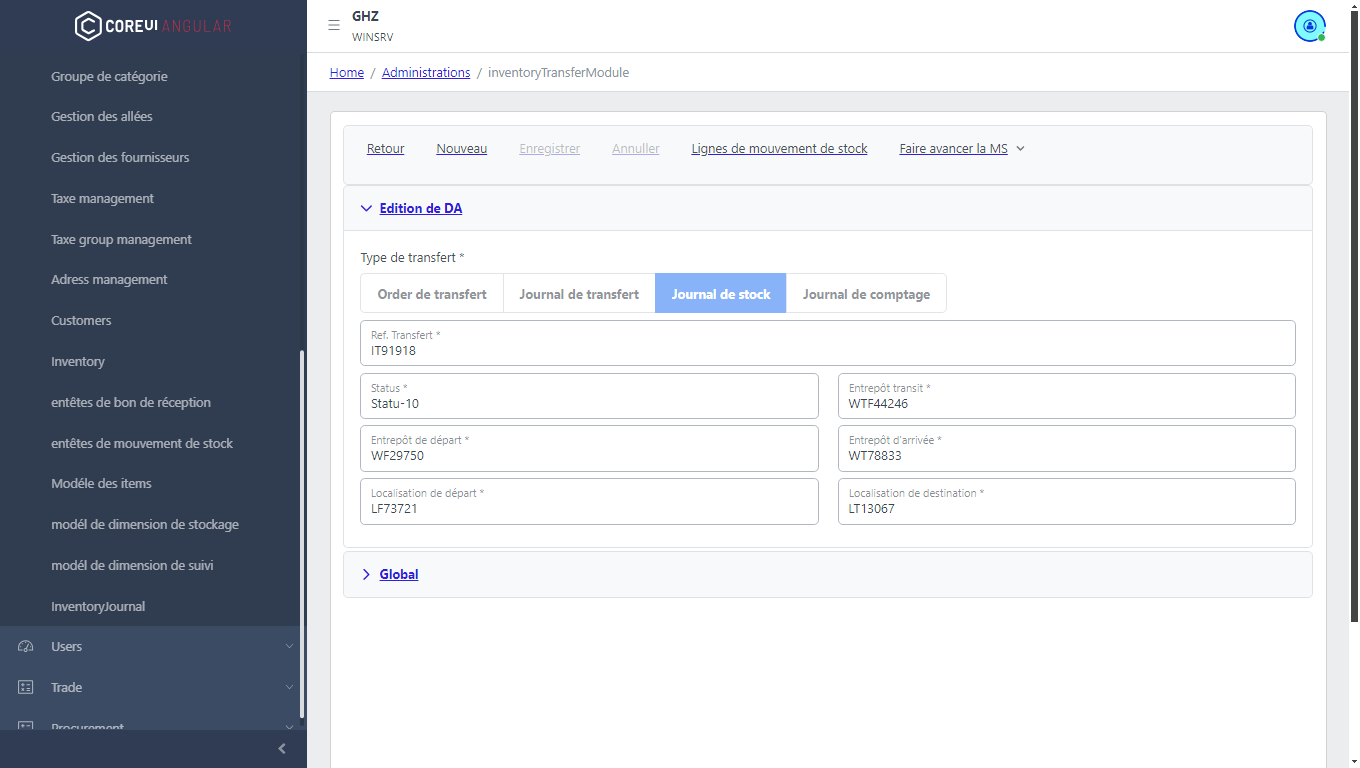
#### List des mouvment de stock



Dans cette interface on a le tableau des entêtes des mouvement de stock comme interface des entêtes des bon s de réception

Dans ce cas on a faire un filtrage par les références qui début par « IT45 » et u triage pour l’entrepôt de départ.

#### Gere ou cree nouveau movement de stock

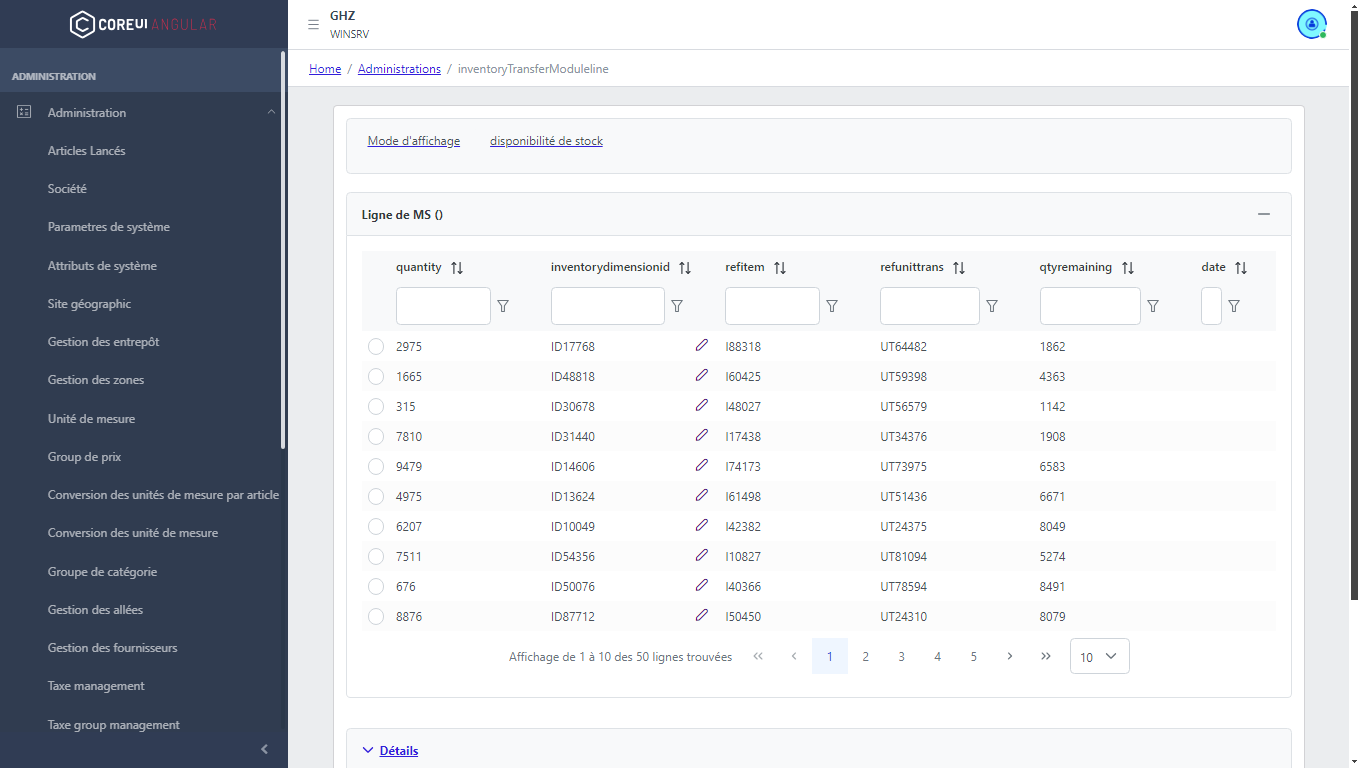


Pour l’interface pour gère l’entête des mouvements de stock on la Megamenu comme l’interface de bon de réception et dans le formulaire l’utilisateur-il que l’utilisateur sélectionner un statut qui va affecter la saisie des informations, en a quarts statuts :

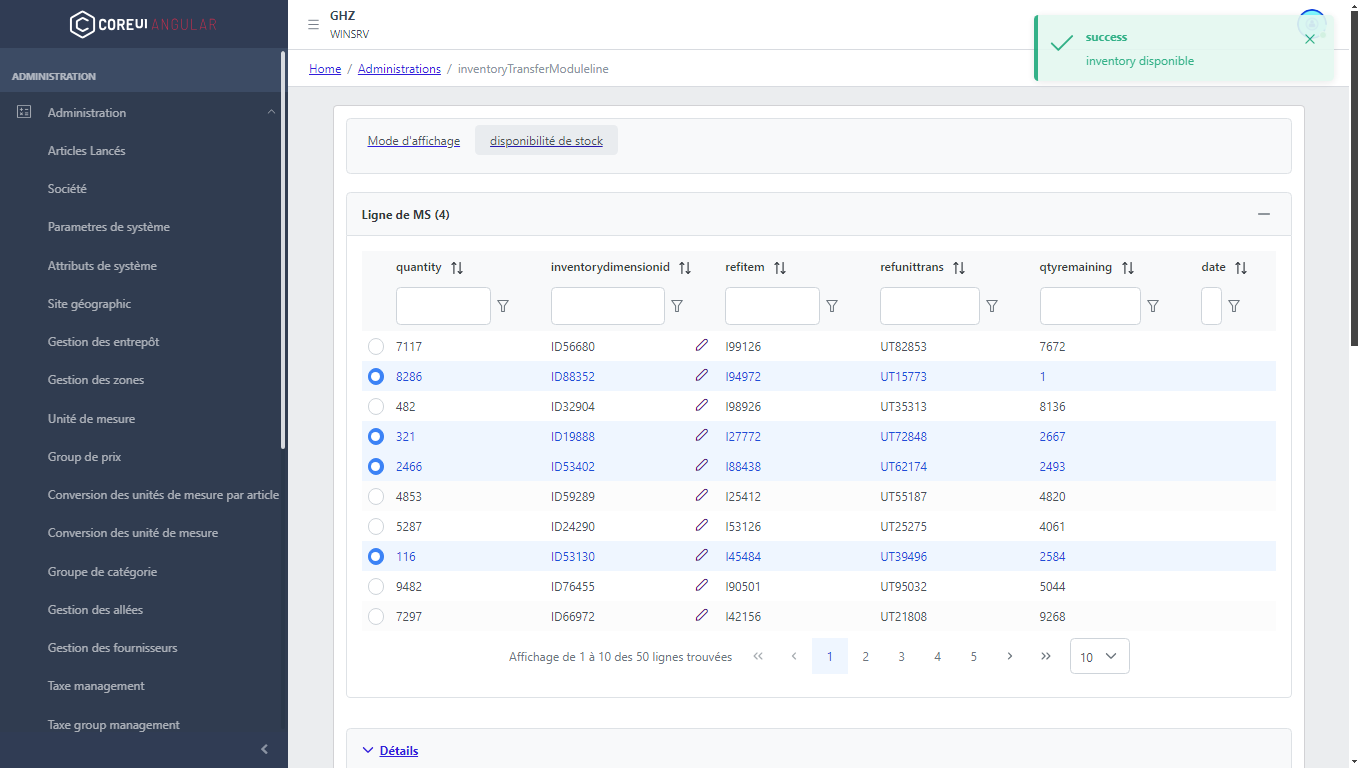
* Ordre de transfert : dans ce cas l’utilisateur no ne doit pas enter les information (entrepôt de départ, entrepôt d’arrivée, localisation de départ, et localisation de destination)
* Journal de transfert : dans ce cas l’utilisateur doit saisir toutes les informations
* Journal de stock : dans ce cas l’utilisateur doit saisir juste l’entrepôt de départ et localisation de départ
* Journal de comptage : ce cas et comme le cas d’ordre de transfert

Pour avancer le MS[[4]](#footnote-4) on a 7 status

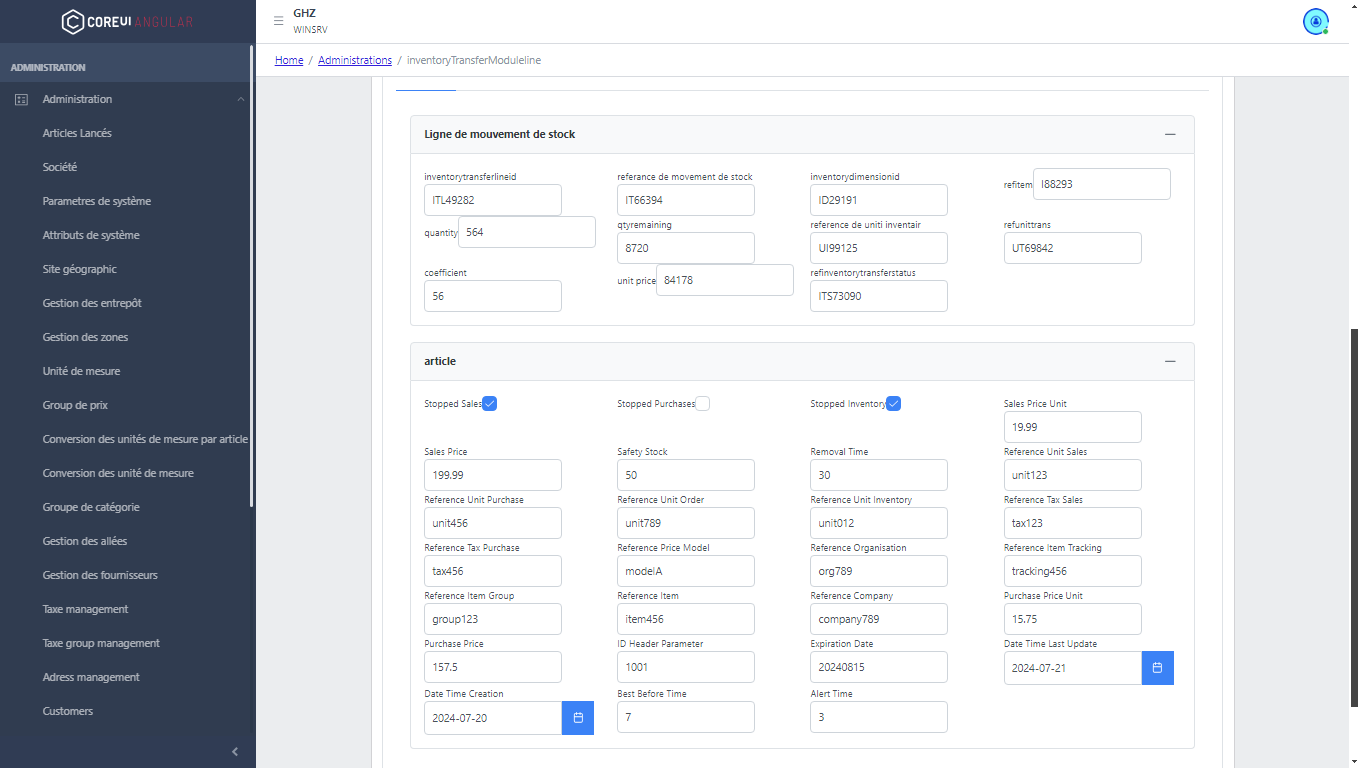
#### Lignes des mouvement de stock



Dans cet interface on a les lignes des movement de stock. Dans la megamenu on a deux button : le mode d’affichage qui fait rient pour le moment et un botton pour vérifier la disponibilite de stock apres la seclectioner des lignes dans la table pour voir si on a une disponibiliter ou non :

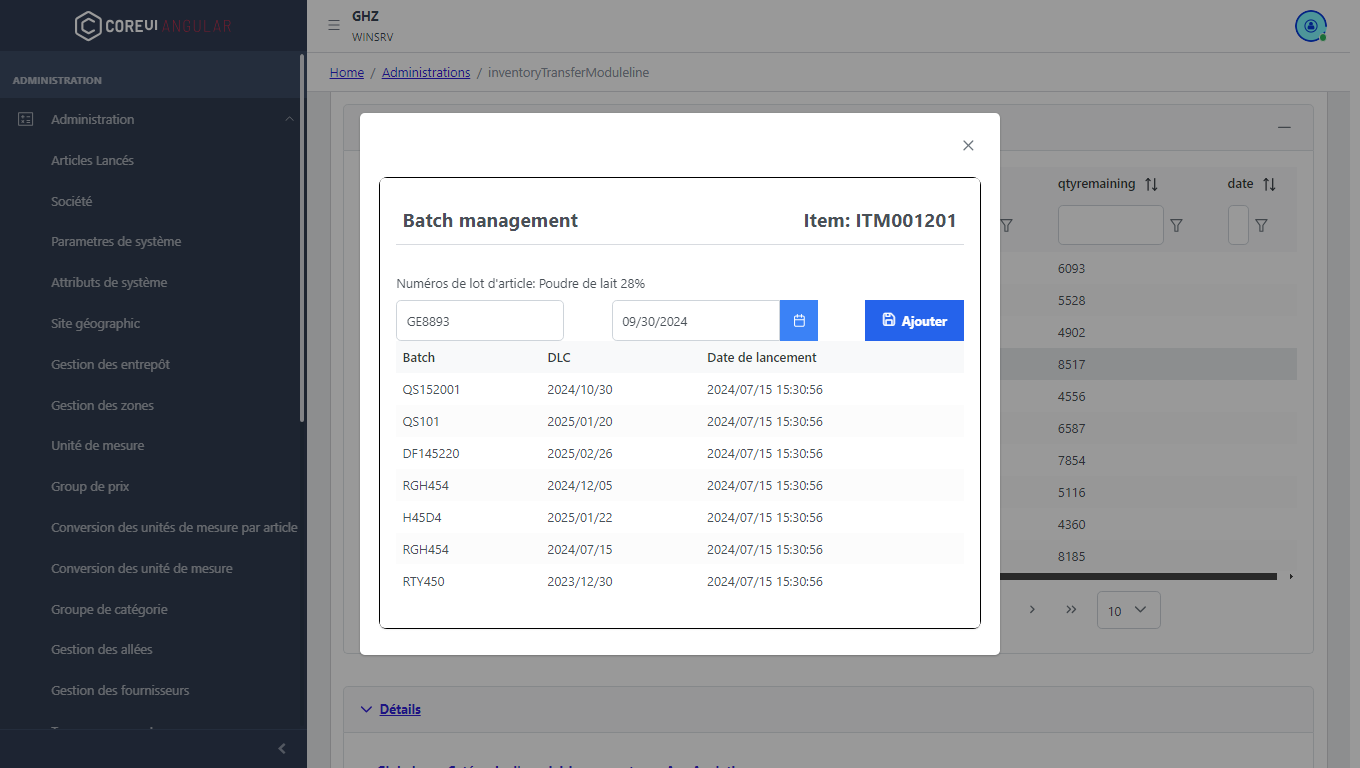


Dans ce cas on a selectioner des lignes pour virifier la disponibiliter de stock et on avoire un reponse de success



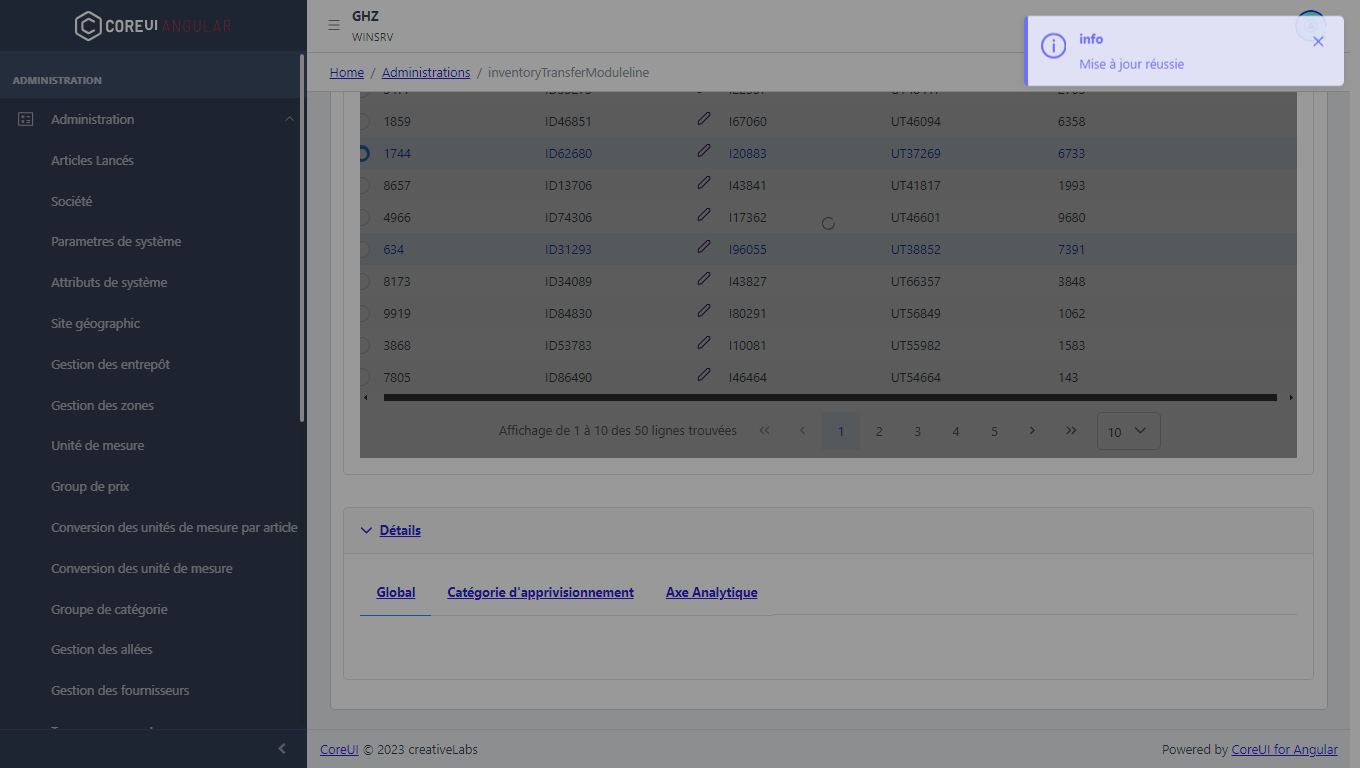
Assui apres la selection d’un ligne dans le tablaux on afficher deux formulare des information relier a la ligne

### Inventory dimention



Pour changer ou cree un novaux batch a un ligne, j'ai créé un nouveau composant que je peux appeler en cliquant sur ce bouton  qui existe-t-il dans l'ID de dimension d'inventaire dans la colonne du tableau des lignes

Dans ce composant je peux selectioner un dimension de suivi ou le cree et donner un DLC[[5]](#footnote-5)



Apres la selection ou la creation d’un lot un message de confermation appare

## Backend

Moderniser la gestion des articles j’ai supremer l’ansien model de prix et le et ajouten l’api pour **model d’article**, **model de dimension de stockage** et **model de dimension de suivi** premiermon ajoutan deux DTO[[6]](#footnote-6) a chaquan un pour afficher et l’autre pour cree. Apres j’ai ajouter troi api’s a le controler d’article pour la creeation de novaux et de avoir les information et de changer le status

### Modele d’article

pour gere un model d’article la method de controler apeler une methode qui fair le traitmen dans le service

#### Cree un model d’article (saveitemmodel)

async createItemModel(*itemmodelDto*: ItemModelCreateDto) {

        // Check if the item model already exists

*const* count = await this.itemmodelRepository.countBy({

                refcompany: *itemmodelDto*.refcompany,

                reforganisation: *itemmodelDto*.reforganisation,

                refitemmodel: *itemmodelDto*.refitemmodel,

        });

        if (count > 0) {

            throw new BadRequestException({

                error: `Le modèle d'article avec refitemmodel : ${*itemmodelDto*.refitemmodel} existe déjà.`

        });

        }

        // If it doesn't exist, create and save the new item model

*const* itemmodel = this.itemmodelRepository.create(*itemmodelDto*);

        return await this.itemmodelRepository.save(itemmodel)

            .then((*res*) *=>* {

                return *res*;

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

            });

    }

Dans cette methode j’ai fair un test si on a déjà un modele d’article avec la meme company, organisation et referance si oui j’ai reterner un erreur et avec un message si non l’article va etre cree et returner le resulta de creation

#### Avoir les information d’un model d’article (getitemmodel)

async getItemmodels(*itemmodelDto*: ItmeModelShowDto){

*const* query = this.itemmodelRepository

        .createQueryBuilder('itemmodel')

        .innerJoinAndSelect(

            'itemmodel.inventoryvaluationmethode',

            'inventoryvaluationmethode'

        )

        .where(

            'itemmodel.refcompany = :refcompany and itemmodel.reforganisation = :reforganisation',

            {refcompany: *itemmodelDto*.refcompany, reforganisation: *itemmodelDto*.reforganisation}

        )

        if(*itemmodelDto*.refitemmodel){

            query.andWhere('itemmodel.refitemmodel = :refitemmodel',

                {refitemmodel: *itemmodelDto*.refitemmodel}

            )

        }

            // Execute the query and return the results

    return await query.getMany()

    .then((*res*) *=>* {

        return *res*;

    })

    .catch((*err*) *=>* {

        throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

    });

    }

Pour cette method j’ai fair une recherche sur les model d’article avec la company et l’organisation et si nous avons la valeur facultative dans le DTO j’ai appere on compt et apres le resulta va etre transmi a le frontend

#### Desactiver le modele d’article (changestateofitemmodel)

async changeStateOfItemmodel(*itemmodelDto*: ItemModelCreateDto) {

        await this.itemmodelRepository

            .createQueryBuilder()

            .update(ItemmodelEntity)

            .set({ actif: *itemmodelDto*.actif })

            .where(

                'refitemmodel = :refitemmodel and refcompany = :refcompany and reforganisation = :reforganisation',

                { refitemmodel: *itemmodelDto*.refitemmodel, refcompany: *itemmodelDto*.refcompany, reforganisation: *itemmodelDto*.reforganisation }

            )

            .execute()

            .then(async (*res*) *=>* {

                return *res*;

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

            });

    }

Dans cette application il ne faut pas suppremer un modele d’article en peux just le disactiver dans j’ai cree cette methode qui changer le statue d’un modele d’article donner

### model de dimension de stockage

#### cree un model de dimension de stockage

async savestoragedimensionmodel(*trackingdimensionmodelDto*: itemstoragedimensionmodelcreateDto){

        // Check if the item model already exists

*const* count = await this.itemstoragedimensionRepository.countBy({

            refcompany: *trackingdimensionmodelDto*.refcompany,

            reforganisation: *trackingdimensionmodelDto*.reforganisation,

            refitemstoragedimensionmodel: *trackingdimensionmodelDto*.refitemstoragedimensionmodel,

        });

        if (count > 0) {

            throw new BadRequestException({

                error: `Le modèle  de dimension de stockage avec refitemstoragedimensionmodel : ${*trackingdimensionmodelDto*.refitemstoragedimensionmodel} existe déjà.`

        });

        }

        // If it doesn't exist, create and save the new item model

*const* itemmodel = this.itemstoragedimensionRepository.create(*trackingdimensionmodelDto*);

        return await this.itemstoragedimensionRepository.save(itemmodel)

        .then((*res*) *=>* {

            return *res*;

        })

        .catch((*err*) *=>* {

            throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

        });

    }

#### Avoir les information d’un model de dimension de stockage

async getItemStorageDimension(*storagedimentionDto*: itemstoragedimensionmodelshowDto){

*const* query = this.itemstoragedimensionRepository

        .createQueryBuilder('itemstoragedimensionmodel')

        .innerJoinAndSelect(

            'itemstoragedimensionmodel.inventorytrackingobject',

            'inventorytrackingobject'

        )

        .where(

            'itemstoragedimensionmodel.refcompany = :refcompany and itemstoragedimensionmodel.reforganisation = :reforganisation',

            {refcompany: *storagedimentionDto*.refcompany, reforganisation: *storagedimentionDto*.reforganisation}

        )

        if(*storagedimentionDto*.refitemstoragedimensionmodel){

            query.andWhere('itemstoragedimensionmodel.refitemstoragedimensionmodel = :refitemstoragedimensionmodel',

                {refitemstoragedimensionmodel: *storagedimentionDto*.refitemstoragedimensionmodel}

            )

        }

            // Execute the query and return the results

    return await query.getMany()

    .then((*res*) *=>* {

        return *res*;

    })

    .catch((*err*) *=>* {

        throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

    });

    }

#### Changer le statuts d’un model de dimension de stockage

async changeitemstoragedimensionstatus(*storageDimensionmodel*: itemstoragedimensionmodelcreateDto) {

        await this.itemstoragedimensionRepository

            .createQueryBuilder()

            .update(ItemstoragedimensionmodelEntity)

            .set({ actif: *storageDimensionmodel*.actif })

            .where(

                `refitemstoragedimensionmodel = :refitemstoragedimensionmodel

                and refcompany = :refcompany

                and reforganisation = :reforganisation`,

                { refitemstoragedimensionmodel: *storageDimensionmodel*.refitemstoragedimensionmodel,

                    refcompany: *storageDimensionmodel*.refcompany,

                    reforganisation: *storageDimensionmodel*.reforganisation }

            )

            .execute()

            .then(async (*res*) *=>* {

                return *res*;

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

            });

    }

### model de dimension de suivi

#### cree un model de dimension de suivi

async savestoragedimensionmodel(*trackingdimensionmodelDto*: itemstoragedimensionmodelcreateDto){

        // Check if the item model already exists

*const* count = await this.itemstoragedimensionRepository.countBy({

            refcompany: *trackingdimensionmodelDto*.refcompany,

            reforganisation: *trackingdimensionmodelDto*.reforganisation,

            refitemstoragedimensionmodel: *trackingdimensionmodelDto*.refitemstoragedimensionmodel,

        });

        if (count > 0) {

            throw new BadRequestException({

                error: `Le modèle  de dimension de stockage avec refitemstoragedimensionmodel : ${*trackingdimensionmodelDto*.refitemstoragedimensionmodel} existe déjà.`

        });

        }

        // If it doesn't exist, create and save the new item model

*const* itemmodel = this.itemstoragedimensionRepository.create(*trackingdimensionmodelDto*);

        return await this.itemstoragedimensionRepository.save(itemmodel)

        .then((*res*) *=>* {

            return *res*;

        })

        .catch((*err*) *=>* {

            throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

        });

    }

#### Avoir les information d’un model de dimension de suivi

async getItemStorageDimension(*storagedimentionDto*: itemstoragedimensionmodelshowDto){

*const* query = this.itemstoragedimensionRepository

        .createQueryBuilder('itemstoragedimensionmodel')

        .innerJoinAndSelect(

            'itemstoragedimensionmodel.inventorytrackingobject',

            'inventorytrackingobject'

        )

        .where(

            'itemstoragedimensionmodel.refcompany = :refcompany and itemstoragedimensionmodel.reforganisation = :reforganisation',

            {refcompany: *storagedimentionDto*.refcompany, reforganisation: *storagedimentionDto*.reforganisation}

        )

        if(*storagedimentionDto*.refitemstoragedimensionmodel){

            query.andWhere('itemstoragedimensionmodel.refitemstoragedimensionmodel = :refitemstoragedimensionmodel',

                {refitemstoragedimensionmodel: *storagedimentionDto*.refitemstoragedimensionmodel}

            )

        }

            // Execute the query and return the results

    return await query.getMany()

    .then((*res*) *=>* {

        return *res*;

    })

    .catch((*err*) *=>* {

        throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query });

    });

    }

#### Changer le statuts d’un model de dimension de suivi

async changeitemstoragedimensionstatus(*storageDimensionmodel*: itemstoragedimensionmodelcreateDto) {

        await this.itemstoragedimensionRepository

            .createQueryBuilder()

            .update(ItemstoragedimensionmodelEntity)

            .set({ actif: *storageDimensionmodel*.actif })

            .where(

                `refitemstoragedimensionmodel = :refitemstoragedimensionmodel

                and refcompany = :refcompany

                and reforganisation = :reforganisation`,

                { refitemstoragedimensionmodel: *storageDimensionmodel*.refitemstoragedimensionmodel,

                    refcompany: *storageDimensionmodel*.refcompany,

                    reforganisation: *storageDimensionmodel*.reforganisation }

            )

            .execute()

            .then(async (*res*) *=>* {

                return *res*;

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

            });

    }

### Purchase requisition

#### Ajout du statut "VALIDER" entre "BROUILLON" et "EN COURS DE RÉVISION"

##### Contexte :

Dans la gestion des workflows de validation, il est souvent nécessaire de définir plusieurs étapes de contrôle pour assurer la qualité et la cohérence des données avant qu'un élément ne soit officiellement accepté ou modifié. Le processus initial comportait les statuts « BROUILLON » et « EN COURS DE RÉVISION », mais il manquait une étape intermédiaire pour valider les données avant révision.

##### Implémentation technique :

* **Mise à jour de l'enum des statuts** : Dans la base de données ou dans le code back-end (selon l'architecture choisie), j'ai ajouté une nouvelle valeur dans l'enum ou la table des statuts.

export *enum* Purchaserequisitionstatuts {

REVS = 'REVS',

VALID = 'VALID',

APRV = 'APRV',

CLTR = 'CLTR',

BRLN = 'BRLN',

RJCT = 'RJCT',

}

* **Déplacement des vérifications métier** : Les vérifications métier, qui étaient auparavant exécutées entre les statuts « BROUILLON » et « EN COURS DE RÉVISION », ont été déplacées entre « BROUILLON » et « VALIDER ». Cela garantit que seules les données validées passeront à l’étape suivante.

else if (*purchaserequisitionChangeStatutDto*.refpurchaserequisitionstatuts ===  Purchaserequisitionstatuts.VALID.toString()){

        // ----------------->>> Valider la DA.

            if (![Purchaserequisitionstatuts.BRLN.toString()].includes(purchreq.refpurchaserequisitionstatuts)) {

                errormessage = 'le statut '+purchreq.refpurchaserequisitionstatuts+'ne permet pas de Valider la DA!';

            } else {

                await this.verifyPurchReqLinesToValider({

                    refpurchaserequisition: *purchaserequisitionChangeStatutDto*.refpurchaserequisition,

                    refcompany: *purchaserequisitionChangeStatutDto*.refcompany,

                    reforganisation: *purchaserequisitionChangeStatutDto*.reforganisation,

                    refvendor: undefined,

                    id: undefined})

                purchreq.refpurchaserequisitionstatuts = *purchaserequisitionChangeStatutDto*.refpurchaserequisitionstatuts;

                purchreq.datesubmittion = new *Date*();

                purchreq.submittedby = *purchaserequisitionChangeStatutDto*.matricule;

            }

        }

async verifyPurchReqLinesToValider(*purchaserequisitionlinesFindDto*: PurchaserequisitionLinesFindDto){

        return await this.purchreqlinesRepository.findBy({

            refcompany: *purchaserequisitionlinesFindDto*.refcompany,

            reforganisation: *purchaserequisitionlinesFindDto*.reforganisation,

            refpurchaserequisition: *purchaserequisitionlinesFindDto*.refpurchaserequisition

        })

            .then(async (*linesPurchReq*) *=>* {

*const* alreadytraiteddata : {refitem: *string*}[] = [];

*let* message = '';

                for(*let* i = 0; i< *linesPurchReq*.length; i++) {

                    if ([undefined, null, ''].includes(*linesPurchReq*[i].refitem)) {

                        message = 'Il faut spécifier un item pour la ligne achat ' + *linesPurchReq*[i].id + ' !'

                        throw new BadRequestException(message, {cause: message, description: message,});

                    }

                    if (*linesPurchReq*[i].quantity <= 0) {

                        message = 'Quantité invalide pour la ligne achat ' + *linesPurchReq*[i].id + ' !'

                        throw new BadRequestException(message, {cause: message, description: message,});

                    }

                    if (alreadytraiteddata.find(*prline* *=>* *prline*.refitem === *linesPurchReq*[i].refitem ) == undefined) {

                        alreadytraiteddata.push({refitem: *linesPurchReq*[i].refitem})

                        await this.itemService.isItemValid({

                            refcompany: *linesPurchReq*[i].refcompany,

                            refitem: *linesPurchReq*[i].refitem,

                            reforganisation: *linesPurchReq*[i].reforganisation,

                        })

                    }

                }

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

            });

    }

* **Logique de changement de statut** : Lors du passage de « VALIDER » à « EN COURS DE REVISION », il n’y a pas de vérification métier. J'ai donc implémenté une logique conditionnelle pour vérifier si un statut est modifié et exécuter ou non des validations en fonction de l'état.

async purchReqStatutsManagementToREVS(*purchaserequisitionChangeStatutDto*: PurchaserequisitionChangeStatutDto){

*const* purchreq = await this.purchreqRepository.findOneBy({

            refpurchaserequisition: *purchaserequisitionChangeStatutDto*.refpurchaserequisition,

            refcompany: *purchaserequisitionChangeStatutDto*.refcompany,

            reforganisation: *purchaserequisitionChangeStatutDto*.reforganisation,

        });

*let* errormessage = '';

        if (*purchaserequisitionChangeStatutDto*.refpurchaserequisitionstatuts ===  Purchaserequisitionstatuts.REVS.toString()) {

        // ----------------->>> lancer la révision pour la DA.

            if (![Purchaserequisitionstatuts.VALID.toString()].includes(purchreq.refpurchaserequisitionstatuts)) {

                errormessage = 'le statut '+purchreq.refpurchaserequisitionstatuts+'ne permet pas de lancer la révision!';

            } else {

                purchreq.refpurchaserequisitionstatuts = *purchaserequisitionChangeStatutDto*.refpurchaserequisitionstatuts;

                purchreq.datesubmittion = new *Date*();

                purchreq.submittedby = *purchaserequisitionChangeStatutDto*.matricule;

            }

        }else{

            errormessage = 'le statut '+*purchaserequisitionChangeStatutDto*.refpurchaserequisitionstatuts+'ne permet pas de lancer la révision!';

        }

        if(!['', null, undefined].includes(errormessage)) {

            throw new BadRequestException(errormessage, { cause: errormessage, description: errormessage,});

        }

        return await this.purchreqRepository

        .save(purchreq)

        .then(async (*res*) *=>* {

            return *res*;

        })

        .catch((*err*) *=>* {

            throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

        });

    }

#### 4. **Clôture des DA avec gestion des lignes**

##### Contexte :

Une DA ne peut être clôturée que si toutes ses lignes sont soit clôturées, soit rejetées. Cela garantit que le processus est complet avant la finalisation.

##### Implémentation technique :

* **Vérification des lignes** : Avant de permettre la clôture d'une DA, j'ai implémenté une vérification qui s’assure que toutes les lignes sont soit en statut « CLÔTURÉ » soit en statut « REJETÉ ».

async verifyPurchReqLinesToApprove(*purchaserequisitionlinesFindDto*: PurchaserequisitionLinesFindDto) {

        await this.purchreqlinesRepository

            .createQueryBuilder('purchaserequisitionlines')

            .where(

                'refpurchaserequisition = :refpurchaserequisition and refcompany = :refcompany and reforganisation = :reforganisation',

                { refcompany: *purchaserequisitionlinesFindDto*.refcompany, refpurchaserequisition :*purchaserequisitionlinesFindDto*.refpurchaserequisition, reforganisation: *purchaserequisitionlinesFindDto*.reforganisation })

            .andWhere('(coalesce(quantity, 0) <= 0 or coalesce(price, 0) <= 0 or refvendor is null or reftaxegroup is null or refcurrency is null or reftaxe is null or refitem is null)')

            .getCount()

            .then(async (*res*) *=>* {

                if ( *res* > 0){

*const* message = 'Merci de saisir les quantités, les prix et les fournisseurs de toutes les lignes de DA'

                    throw new BadRequestException(message, { cause: message, description: message,});

                }

            })

            .catch((*err*) *=>* {

                throw new BadRequestException(*err*.message, { cause: *err*, description: *err*.query,});

            });

    }

## Tests

### Test et Mocking pour le Frontend

Afin de faciliter le développement et le test du frontend de notre application sans dépendre d'un backend en cours de développement, nous avons opté pour l'utilisation de Mockoon, un outil de simulation de serveurs REST API. Mockoon nous a permis de créer des environnements de mock avec des endpoints qui renvoient des données factices (fake data), ce qui s'est avéré essentiel pour valider le bon fonctionnement des appels API dans le frontend sans nécessiter un backend fonctionnel.

**Processus de création de l'environnement Mockoon**

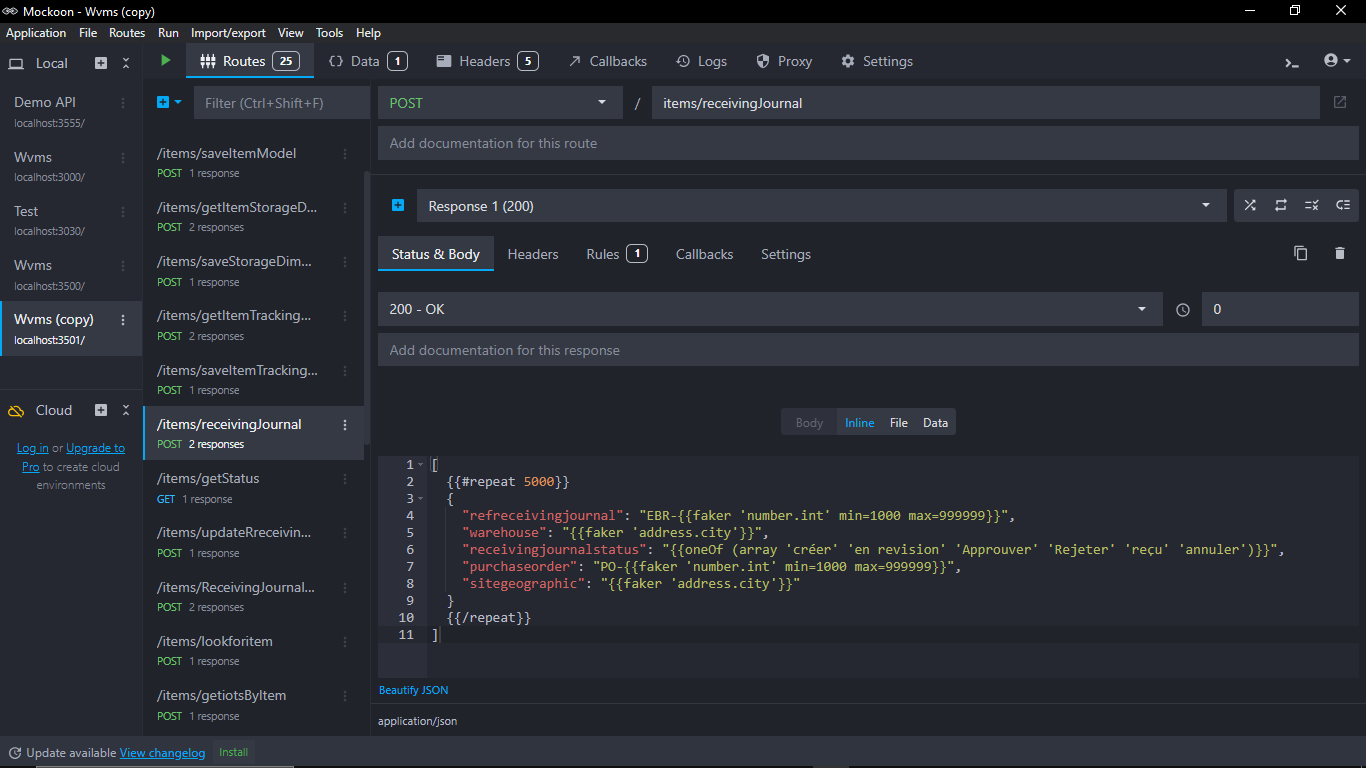
1. **Création des endpoints** : Dans Mockoon, nous avons configuré plusieurs routes simulant les différentes API du backend, telles que l’authentification des utilisateurs, la gestion des stocks, etc. Chaque route renvoyait une réponse JSON pré-configurée correspondant à des scénarios typiques (succès, erreurs).
2. **Configuration des données factices** : Les données retournées par Mockoon ont été générées de manière à représenter des cas réels, par exemple des informations sur des produits, les niveaux de stock, ou encore les détails des utilisateurs. Ces données ont été spécifiquement formatées pour imiter les résultats renvoyés par notre API réelle.
3. **Tests avec le frontend Angular** : Une fois les API simulées configurées, nous avons intégré Mockoon au processus de développement du frontend. Grâce à l’intégration des endpoints mockés, nous avons pu tester l’intégralité des composants du frontend en simulant les requêtes API. Cela a permis de vérifier la gestion des données reçues, le traitement des erreurs, et l'affichage des informations sans attendre que le backend soit complètement implémenté.

**L'utilisation de Mockoon s’est avérée très utile pour :**

* **Accélérer le développement du frontend** : en simulant des API fonctionnelles, il a été possible de développer des fonctionnalités frontales de manière indépendante du backend.
* **Tests unitaires et de bout en bout** : les données mockées nous ont permis de réaliser des tests unitaires et des tests end-to-end sur les composants du frontend.

**Exemple de configuration de Mockoon**

Voici un exemple de configuration d'un endpoint de Mockoon pour simuler l'obtention d'une liste de produits :



Cette capture d'écran montre l'interface de **Mockoon**, un outil utilisé pour simuler des serveurs REST API localement. Le route affichée est **/items/receivingJournal** avec une méthode **POST**.

* **Route /items/receivingJournal :**
* **Statut de réponse :** La réponse HTTP est configurée pour renvoyer un statut 200 OK, ce qui signifie que la requête POST est réussie.
* **Corps de la réponse :** Le corps de la réponse est formaté en JSON et utilise des modèles pour générer des données factices dynamiques.
* La structure utilise la syntaxe de Mockoon avec la fonctionnalité {{repeat 5000}}, qui permet de générer 5000 entrées dans la réponse.
* **Champs générés dynamiquement :**

**refreceivingjournal :** Ce champ représente un identifiant unique pour chaque bon de réception. Il est généré dynamiquement à l'aide de la fonction faker, avec une plage de valeurs numériques allant de 1000 à 999999.

**warehouse :** Ce champ représente l'entrepôt lié au bon de réception. Il utilise la fonction faker.address.city pour générer un nom de ville aléatoire.

**receivingjournalstatus :** Ce champ représente le statut du journal de réception. Il utilise la fonction oneOf pour choisir aléatoirement un statut parmi : créer, en revision, Approuver, Rejeter, reçu, ou annuler.

**purchaseorder :** Il s'agit du numéro de commande d'achat, généré dynamiquement avec la même plage de valeurs que refreceivingjournal.

**sitegeographic :** Il représente la localisation géographique du site, également générée à l'aide de la fonction faker.address.city.

**Simulation de données massives :**

La route retourne 5000 enregistrements, simulant ainsi une large liste de bons de réception. Cette fonctionnalité permet de tester la capacité du frontend à gérer des volumes importants de données.

### Tests des API Backend

Afin de vérifier le bon fonctionnement des API développées avec NestJS, j'ai utilisé **Postman** pour tester et valider les différentes requêtes avant de les passer au frontend. Cette étape a permis de simuler des appels API et de s'assurer que le backend fonctionnait correctement. Voici les principales étapes que j'ai suivies pour réaliser ces tests :

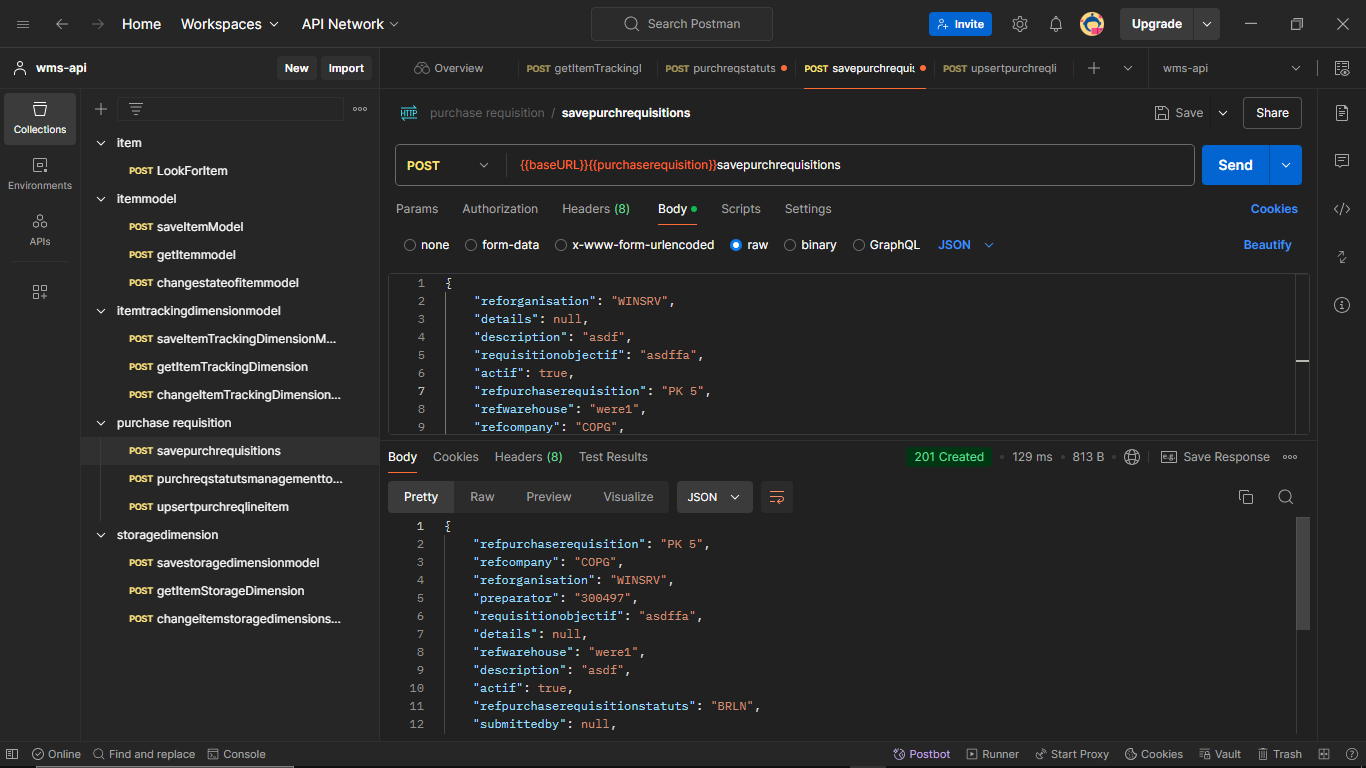
**Configuration des requêtes API** : J'ai défini plusieurs requêtes dans Postman pour interagir avec les endpoints de l'API NestJS. Chaque requête était configurée avec la méthode HTTP appropriée (POST, GET, etc.) et les paramètres requis, par exemple :

* Pour rechercher un article spécifique, j'ai utilisé une requête POST avec les paramètres refcompany, reforganisation, et refitem afin d'obtenir les détails de l'article.
* Pour ajouter un nouveau modèle d'article, j'ai utilisé une autre requête POST en spécifiant des informations telles que refitemmodel, itemmodel, et d'autres attributs dans le corps de la requête.

**Simulation des requêtes et validation des réponses** : Après avoir configuré les requêtes, j'ai exécuté ces dernières dans Postman et analysé les réponses renvoyées par l'API. Cela m'a permis de valider que les données étaient correctement traitées et que les réponses respectaient les spécifications.

**Gestion des erreurs et ajustements** : En cas d'erreurs (par exemple, des réponses avec un code d'état HTTP 400 ou 500), j'ai pu rapidement identifier les problèmes dans la logique de traitement du backend et ajuster le code NestJS en conséquence.

**Passage au Frontend** : Une fois que les API ont été testées et validées, j'ai fourni les détails des endpoints et les résultats attendus à l'équipe frontend. Cela leur a permis d'intégrer facilement les API dans leur logique de développement, en ayant déjà la certitude que le backend fonctionnait correctement.



1. Token : [↑](#footnote-ref-1)
2. Megamenu: [↑](#footnote-ref-2)
3. BR: bon de réception [↑](#footnote-ref-3)
4. MS: movement de stock [↑](#footnote-ref-4)
5. [↑](#footnote-ref-5)
6. DTO: [↑](#footnote-ref-6)