Honoris united universities - Ecole Marocaine des Sciences de l'Ingénieur

Rapport de stage 15Juil-15Sep

Réalisé par

Haitham EL ABDIOUI
Aboubakr KETOUN

LPG IN THE BOX
Module de gestion de production et gestion des stocks assisté par ordinateur (GMAO)

---

sous la direction de :

M. Charafeddine LECHHEB	Encadrant Professionnel
M. Hamza YOUBI	Encadrant Professionnel
Mme Chourouk ELOKRI	Encadrante Académique

Entreprise d'accueil

(OLA ENERGY)

Dédicace

À ma famille aimante et dévouée, qui m'a soutenu tout au long de ce parcours académique. Votre encouragement constant et votre soutien inconditionnel m'ont donné la force de persévérer. Je vous dédie ce mémoire avec tout mon amour et ma gratitude.

À mes encadrant professionels et académique, qui ont été un guide précieux tout au long de ce projet. Votre expertise, votre soutien et vos conseils éclairés ont été essentiels pour mener à bien cette recherche. Je vous remercie sincèrement pour votre engagement et votre mentorat.

À mes amis et camarades de classe, qui ont partagé cette aventure avec moi. Vos encouragements, nos discussions enrichissantes et notre soutien mutuel ont rendu ce parcours plus agréable. Cette dédicace est pour vous, mes chers amis.

À toutes les personnes qui ont participé à cette étude en tant que participants ou répondants, je vous suis reconnaissant de votre contribution précieuse. Votre participation a été essentielle pour la réalisation de ce projet.

Remerciements

Nous tenons à exprimer nos plus sincères remerciements à l'École Marocaine des Sciences de l'Ingénieur (EMSI) pour l'organisation de l'événement "EMSI IT Summer Competition". Cet événement a été une occasion exceptionnelle pour nous, étudiants, de développer nos compétences, d'élargir notre réseau professionnel, et de nous ouvrir à de nouvelles opportunités de stages. Grâce à l'initiative et à l'engagement d'EMSI, cet événement a permis de renforcer notre préparation pour le monde professionnel tout en favorisant l'innovation et l'excellence académique.
Nous souhaitons également adresser notre profonde gratitude à l'entreprise OLA ENERGY pour son précieux soutien à cet événement. Leur engagement en faveur du développement des jeunes talents et leur investissement dans notre future carrière sont à saluer. Leur présence a été un véritable catalyseur de succès, offrant aux participants des perspectives enrichissantes et un lien direct avec le monde de l'entreprise.
Par ailleurs, nous souhaitons exprimer notre profonde reconnaissance à Mr Charafeddine LECHHEB et Mr Hamza YOUBI pour leur encadrement remarquable et leur soutien indéfectible tout au long de notre stage. Leur expertise et leur passion pour leur métier ont été une source constante d'inspiration et de motivation pour nous. Ils ont su créer un environnement d'apprentissage stimulant et bienveillant, où chaque question trouvait une réponse, chaque idée était considérée, et chaque erreur était transformée en une opportunité d'apprendre et de s'améliorer.
Nous avons également énormément apprécié la patience avec laquelle ils ont su répondre à nos nombreuses questions, ainsi que la clarté de leurs explications, qui nous ont permis de comprendre des concepts parfois complexes. Leur approche pédagogique, basée sur l'écoute, l'accompagnement personnalisé, et l'encouragement à se dépasser, a été déterminante pour notre développement. Ils n'ont pas seulement partagé leurs connaissances, mais aussi leur savoir-faire, leurs conseils avisés, et leur vision stratégique, ce qui nous a permis de mieux appréhender les réalités du monde professionnel et de gagner en assurance.
Enfin, leur disponibilité, leur capacité à identifier nos forces et nos axes d'amélioration, et leur engagement à nous aider à progresser ont fait de ce stage une expérience riche et inoubliable. Nous sommes particulièrement reconnaissants pour leur encouragement constant à penser de manière critique, à collaborer de manière efficace au sein de l'équipe, et à explorer de nouvelles idées et approches. Leur confiance en nos capacités a été un véritable moteur pour notre développement personnel et professionnel.
Nous tenons également à remercier sincèrement Mme Chourouk ELOKRI pour son soutien continu et sa disponibilité. Son professionnalisme et son approche bienveillante ont grandement contribué à créer un environnement de travail motivant et enrichissant, où nous avons pu évoluer sereinement.
De plus,Nous tenons également à remercier sincèrement Mme Oukriche Nadia et Mme Melaouen Noussaiba pour leur excellente coordination et leur suivi minutieux. Leur professionnalisme et leur sens de l'organisation ont été essentiels au bon déroulement de l'événement et à sa réussite éclatante.
Pour terminer, nos remerciements vont aux étudiants organisateurs pour leur dévouement, leur esprit

d'initiative, et leur dynamisme tout au long de la préparation et du déroulement de cet événement. Leur travail acharné a assuré la bonne organisation de la compétition, rendant l'expérience mémorable pour tous les participants.

Résumé

Lors de notre stage chez OLA Energy, nous avons conçu et mis en place un système complet de gestion d'inventaire et de stock, composé de deux applications interconnectées, chacune adaptée à des contextes d'utilisation spécifiques, mais partageant une base de données centralisée pour garantir la cohérence des informations.

La première application, développée en Visual Basic, était destinée à une utilisation desktop au siège de l'entreprise. Elle offrait aux gestionnaires une vue globale et détaillée des stocks, leur permettant de suivre et de contrôler efficacement les mouvements de produits. Cette application permettait la gestion des articles et des catégories de produits, avec une classification précise des produits en fonction de leurs caractéristiques. Elle incluait également un module de suivi des mouvements de stock, où les utilisateurs pouvaient enregistrer les entrées et sorties des articles, facilitant la traçabilité des produits. En outre, elle permettait la génération de rapports d'inventaire complets et détaillés, offrant aux gestionnaires une analyse précise des tendances et des niveaux de stock, ce qui facilitait la prise de décisions stratégiques.

En parallèle, nous avons développé une application mobile avec PowerApps, destinée aux employés sur le terrain, notamment dans les différents magasins. Cette application permettait aux agents d'effectuer des opérations d'inventaire directement depuis leur lieu de travail, avec une interface adaptée à une utilisation mobile. Ils pouvaient consulter en temps réel les quantités de produits disponibles, enregistrer les mouvements d'entrée et de sortie lors des ventes ou des réceptions des articles, et synchroniser ces données avec le système central. Grâce à cette synchronisation en temps réel, assurée via une API développée pour relier les deux applications, les informations étaient constamment mises à jour, garantissant une cohérence parfaite entre les données de la version desktop et celles de la version mobile.

Cette intégration fluide entre les deux applications a permis d'améliorer considérablement l'efficacité de la gestion des stocks chez OLA Energy. Le système a optimisé la coordination entre les gestionnaires au siège et les agents sur le terrain, permettant à l'entreprise de mieux anticiper ses besoins en réapprovisionnement et d'améliorer ses processus logistiques. Cette solution a offert une visibilité en temps réel sur les stocks, facilitant ainsi une meilleure réactivité dans la gestion des ressources et une prise de décision plus éclairée.

Table des matières

Table des figures

Introduction

Dans le cadre de notre stage chez OLA ENERGY et en collaboration avec la première édition de l'EMSI IT SUMMER COMPETITION 2024, nous avons entrepris le développement d'un module de gestion de production et de gestion des stocks assisté par ordinateur (GMAO) pour le projet "LPG IN THE BOX". Cette compétition innovante, organisée par l'École Marocaine des Sciences de l'Ingénieur (EMSI), nous a offert une plateforme unique pour mettre en pratique nos compétences techniques et répondre à des problématiques réelles de l'industrie, en partenariat avec des entreprises de premier plan telles que OLA ENERGY,une entreprise de premier plan opérant dans 17 pays à travers l'Afrique.

Le projet "LPG IN THE BOX" a pour objectif de centraliser l'ensemble des flux d'information pour améliorer la gouvernance de l'activité Gaz dans les filiales du groupe à travers l'Afrique. Il s'agit d'un projet ambitieux qui inclut l'analyse et l'amélioration des outils existants de gestion de production et de stocks, tout en intégrant des technologies modernes telles que Vba et Power Apps pour une meilleure visualisation et un suivi optimisé des opérations. En tant que l'un des principaux acteurs du secteur énergétique africain, OLA ENERGY vise à renforcer sa capacité de gestion stratégique en optimisant ses processus et en améliorant l'efficacité de ses systèmes d'information.

Le cadre de l'EMSI IT SUMMER COMPETITION 2024 nous a permis de travailler sur un projet réel tout en bénéficiant d'un encadrement de haut niveau. Grâce au soutien et aux conseils de Mr Charafeddine LECHHEB et Mr Hamza YOUBI, nos encadrants professionnels au sein d'Ola Energy, ainsi qu'à l'accompagnement académique de Mme Chourouk ELOKRI, assurant ainsi que chaque amélioration apportée réponde aux exigences spécifiques du projet et aux standards de qualité d'OLA ENERGY.

Dans ce contexte, nous avons suivi un plan de travail structuré qui inclut plusieurs phases clés : l'analyse des processus actuels de gestion de la production et des stocks, l'identification des besoins des utilisateurs, la conception d'une architecture logicielle adaptée, le développement et l'intégration des nouvelles fonctionnalités, ainsi que des tests rigoureux pour valider les améliorations apportées. Chaque étape a été minutieusement conçue pour garantir une amélioration significative de l'efficacité opérationnelle et de la gouvernance des stocks et de la production. Le projet comprend également l'ajout de sous-modules permettant un suivi détaillé des achats, des consignations, et la mise en place d'une bibliothèque technique pour renforcer la gestion des standards opérationnels.

En participant à l'EMSI IT SUMMER COMPETITION 2024, nous avons non seulement eu l'occasion de mettre en pratique nos compétences théoriques, mais aussi de contribuer à un projet d'envergure qui a un impact direct sur la performance opérationnelle et la compétitivité d'OLA ENERGY en tant que leader du secteur énergétique africain.

Chapitre 1

Présentation de l'entreprise

1.1 OLA Energy : Une Présence Ancrée en Afrique

Les valeurs fondamentales d'OLA Energy sont profondément enracinées dans le continent africain. Présent dans 17 pays, le groupe joue un rôle crucial dans le développement économique et social des communautés où il opère, contribuant ainsi à la prospérité globale de l'Afrique. Fidèle à ses origines, OLA Energy s'engage à être un acteur responsable en adhérant aux valeurs éthiques africaines telles que l'intégrité, l'honnêteté et l'équité.
Au fil des années, OLA Energy s'est affirmé comme un acteur incontournable sur le continent africain. Le groupe emploie plus de 1 500 salariés issus de divers horizons et génère environ 20 000 emplois indirects dans les pays où il est implanté. Chaque jour, jusqu'à 250 000 clients fréquentent ses installations. OLA Energy dispose de plus de 1 200 stations-service, 8 usines de mélange, plus de 60 terminaux de carburant, et opère dans plus de 50 aéroports à travers l'Afrique, témoignant de son engagement à maintenir et à renforcer sa présence sur le continent.

1.2 Vision

Devenir le distributeur d'énergie de référence en Afrique, en façonnant l'avenir énergétique du continent tout en favorisant la prospérité de ses habitants.

1.3 Mission

Satisfaire nos clients en leur fournissant des produits et services de haute qualité grâce à des équipes dynamiques et des partenariats solides, tout en stimulant la croissance et en assurant des retours positifs pour toutes les parties prenantes.

1.4 Valeurs

— Excellence en matière de sécurité et de santé
— Engagement envers la qualité
— Responsabilité sociale
— Recherche constante de l'excellence

1.5 Histoire d'OLA Energy

— **1993 :** Début des investissements en Afrique. Tamoil entame ses activités en aval en Égypte sous la holding Oilinvest basée en Europe.
— **2000 :** Expansion sur le continent africain. Création de Tamoil Tchad SA, Tamoil Niger SA, Tamoil Mali SA et Tamoil Burkina SA, suivie de l'acquisition de Shell Érythrée. Constitution de Tamoil Africa Holdings Limited à Malte pour regrouper toutes les activités africaines d'Oilinvest.
— **2008 :** Expansion stratégique par acquisitions. Acquisition de cinq filiales Shell au Niger, au Tchad, à Djibouti, en Éthiopie et au Soudan, ainsi que de neuf filiales ExxonMobil au Niger, au Sénégal, en Côte d'Ivoire, au Gabon, au Cameroun, au Kenya, à La Réunion, en Tunisie et au Maroc. Constitution de Libya Oil Holdings Ltd.
— **Aujourd'hui :** Diversification des activités avec l'acquisition de Joint Oil et Altube, ainsi qu'une participation dans Circle Oil et AIC.

1.6 Organigramme de OLA Energy

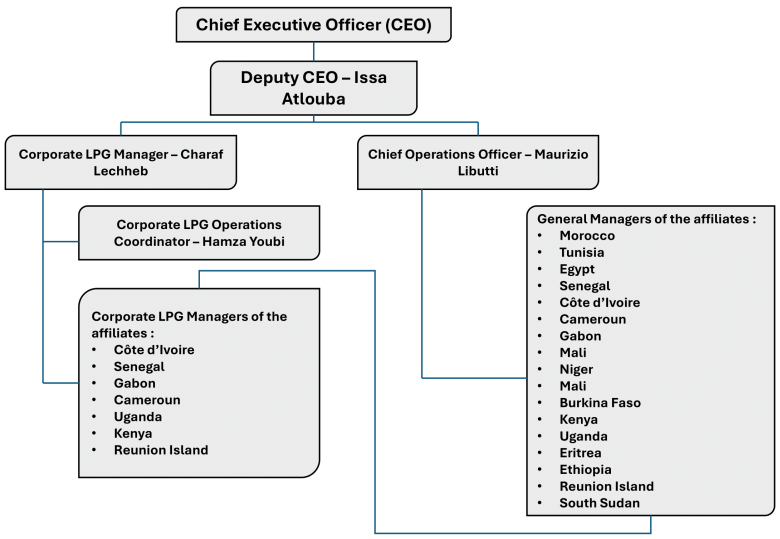

Figure 1.1 – Organigramme OLA Energy

1.7 Déscription de la gestion des inventaires et stocks utiliser par l'entreprise

L'entreprise a mis en place un processus rigoureux de gestion des stocks et des inventaires, structuré en trois phases distinctes, afin d'optimiser la précision et l'efficacité de ses opérations logistiques. Chacune de ces phases joue un rôle crucial dans le contrôle des flux de produits et dans la réduction des risques de disparités de stock.

- **Établissement de l'inventaire journalier :** Cette première phase est cruciale, car elle permet de capturer quotidiennement une image fidèle de l'état des stocks. Chaque jour, les articles sortis de l'inventaire sont enregistrés, tout comme les entrées, qu'il s'agisse de retours ou de nouvelles réceptions. La méthode utilisée pour déterminer le stock disponible à la fin de la journée est la soustraction des sorties aux entrées. Cela donne une vue claire et actualisée de l'état des stocks, essentielle pour les opérations du lendemain. Ce suivi quotidien aide également à prévenir les erreurs d'inventaire et les ruptures de stock en offrant une visibilité immédiate sur les niveaux actuels.
- **Détail des mouvements de stock :** La seconde phase approfondit l'analyse initiée par l'inventaire journalier en se focalisant sur les mouvements de stock spécifiques de la journée. Elle détaille chaque transaction, qu'il s'agisse d'une vente, d'un achat ou d'un transfert entre emplacements. Cette étape est vitale pour tracer la provenance et la destination de chaque article, permettant ainsi de vérifier l'exactitude des enregistrements et de corriger toute incohérence. Le détail des mouvements est aussi un outil précieux pour les audits internes, offrant une traçabilité complète et facilitant la résolution des problèmes liés aux écarts d'inventaire.
- **Tableau de réconciliation :** La dernière phase est la réconciliation, qui se fait sur une base périodique (mensuelle ou annuelle selon les besoins de l'entreprise). Ce tableau de réconciliation compile tous les mouvements enregistrés sur la période concernée, en juxtaposant les entrées et les sorties pour chaque catégorie d'article. Cette analyse est essentielle pour comprendre les tendances de consommation, évaluer l'efficacité des politiques de gestion des stocks, et ajuster les stratégies d'approvisionnement. Elle permet aussi de détecter et d'analyser les écarts significatifs, facilitant ainsi l'identification des problèmes systémiques ou des erreurs opérationnelles.

En intégrant ces trois phases dans son processus de gestion des stocks, l'entreprise s'assure d'une gestion précise et proactive de ses inventaires, contribuant à minimiser les coûts tout en optimisant la disponibilité des produits. Cette approche systématique est non seulement un gage de fiabilité pour les opérations internes mais aussi un facteur de confiance pour les partenaires et clients de l'entreprise.

Chapitre 2

Analyse et définition des besoins

2.1 Détermination des fonctionnalités

La création d'une plateforme pour la gestion des inventaires nécessite une méthode organisée pour répondre aux besoins essentiels des utilisateurs. Le besoin primordial est le suivi précis des inventaires journaliers. Cette fonctionnalité doit permettre aux utilisateurs d'enregistrer les niveaux d'inventaire au début et à la fin de chaque journée d'activité. En offrant une vue claire des quantités disponibles, la plateforme permet de détecter rapidement les écarts et d'ajuster les stocks en conséquence. La récolte quotidienne de données garantit une traçabilité complète et une gestion proactive des stocks, ce qui est décisif pour maintenir l'exactitude des niveaux d'inventaire.

En plus de cette fonctionnalité, il est essentiel de mettre en place un système efficace pour le suivi des mouvements quotidiens des produits. Cette fonctionnalité doit permettre l'enregistrement détaillé de chaque entrée et sortie de stock. Suivre les mouvements en temps réel offre une vision complète des transactions journalières, ce qui aide à gérer les niveaux de stock de manière optimale. Une telle accessibilité facilite également la planification et l'optimisation des processus logistiques, contribuant ainsi à une gestion plus fluide et réactive des inventaires.

Enfin, un tableau de réconciliation robuste est nécessaire pour compléter la gestion des inventaires. Ce tableau permet de mesurer les données d'inventaire avec les mouvements enregistrés sur une période spécifiée, offrant ainsi un contrôle approfondi de la gestion des stocks. La réconciliation aide à identifier les incohérences, à vérifier l'exactitude des informations et à ajuster les enregistrements si nécessaire. En fournissant une analyse détaillée et un mécanisme de contrôle, le tableau de réconciliation assure lexactitude des données et contribue à une gestion stricte des stocks.

En résumé, les besoins pour la plateforme incluent un suivi précis des inventaires journaliers, un enregistrement détaillé des mouvements quotidiens des produits, et un tableau de réconciliation pour le contrôle des stocks. Ces fonctionnalités sont importantes pour garantir une gestion efficace des inventaires et une prise de décision claire.

2.2 Étude du système déja existant

L'analyse du fichier BASSA.xlsm, fourni par l'entreprise, révèle plusieurs difficultées significatives en termes de structure et d'efficacité du code. Ces observations mettent en avant des domaines

nécessitant des améliorations substantielles pour optimiser la gestion des données et garantir la pérennité du système.

2.2.1 Structure du Code et Modularité :

La structure actuelle du code dans BASSA.xlsm présente des défauts majeurs en matière de modularité, ce qui complique la maintenance et l'évolution du fichier. Le code n'est pas convenablement réparti, rendant difficile l'isolation et la résolution des problèmes spécifiques. Une refonte vers une approche modulaire, avec des fonctions et des procédures distinctes, améliorerait en grande partie la lisibilité et la flexibilité du code. Cela faciliterait également les mises à jour et les modifications futures, minimisant les risques de perturbations dans le système global.

```
Sub MovementFile()
' Mise à jour de l'historique des mouvements

' Vérifie la saisie de tous les champs des articles à mouvementer ( test Ok / Non-Ok)

Dim CoherenceFlag As String
CoherenceFlag = Sheets("Movement Sheet").Range("B33").Value

If CoherenceFlag = "Non-Ok" Then

MsgBox ("Prière de renseigner tous les champs requis pour la génération du mouvement , Tous les champs doivent apparaître en vert avec mention - Ok -")

Exit Sub

End If

' Enlever la protection des feuilles

Sheets("Movement Record").Unprotect "OECM-CMT-2021@"
Sheets("Movement Sheet").Unprotect "OECM-CMT-2021@"

'Impression du mouvement en format PDF

Dim moisc As String
moisc = Sheets("Movement Sheet").Range("L12").Value
Dim annee As String
annee = Sheets("Movement Sheet").Range("M12").Value

' concatenation Référence du texte + Numéro du BL

Dim BLRefTxt As String
BLRefTxt = Sheets("Movement Sheet").Range("G12").Value
Sheets("Movement Sheet").Select

Application.PrintCommunication = False
    With ActiveSheet.PageSetup
        .Orientation = xlPortrait
        .Zoom = False
        .PrintArea = Worksheets("Movement Sheet").Range("D3:O40")
        .FitToPagesWide = 1
        .FitToPagesTall = 1
```

FIGURE 2.1 – Problème de modularité

2.2.2 Utilisation des Fonctions Prédéfinies :

Le fichier s'appuie concidérablement sur les fonctions prédéfinies d'Excel, ce qui ne répond pas toujours aux besoins spécifiques de gestion des données de l'entreprise. Bien que ces fonctions soient convenables pour des opérations générales, elles ne sont pas nécessairement conçues pour des exigences de traitement complexes ou personnalisées. Il est recommandé d'intégrer des fonctions personnalisées et adaptées aux besoins spécifiques de l'entreprise pour améliorer la précision et la flexibilité du traitement des données.

Sélectionnez une date début réconciliation	24/03/2025	24 mars 2025
Sélectionnez une date fin réconciliation	01/01/2024	1 janvier 2024
Sélectionnez le magasin	Magasin GPL BASSA Code N°415	
Sélectionnez le nom du magasin LPG	BASSA	

FIGURE 2.2 – Exemple d'un problème logique

2.2.3 Gestion des Erreurs :

La gestion des erreurs dans BASSA.xlsm est insuffisante, ce qui génère des problèmes de cohérence dans la base de données. L'absence de mécanismes robustes pour la gestion des erreurs peut entraîner des entrées incorrectes ou incomplètes, compromettant ainsi l'intégrité des données. Il est obligatoire de mettre en œuvre des contrôles d'erreurs plus rigoureux et des validations systématiques pour assurer que les données sont correctement enregistrées et traitées.

284	05/10/2021	Fin d'activité	18H30	Bouteilles vide en service	Bouteille 35KG	100	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
285	05/10/2021	Fin d'activité	18H30	Bouteille vide en consignation	Bouteille 35KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
286	05/10/2021	Fin d'activité	18H30	Bouteille fuyarde	Bouteille 35KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
287	05/10/2021	Fin d'activité	18H30	Bouteille à ré éprouver	Bouteille 35KG	13	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
288	05/10/2021	Fin d'activité	18H30	Bouteille à entretenir	Bouteille 35KG	981	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
289	05/10/2021	Fin d'activité	18H30	Bouteille à détruire	Bouteille 35KG	3	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
290	05/10/2021	Fin d'activité	18H30	Casier en bon état	Casier 6KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
291	05/10/2021	Fin d'activité	18H30	Casier à entretenir	Casier 6KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
292	05/10/2021	Fin d'activité	18H30	Casier à réformer	Casier 6KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
293	05/10/2021	Fin d'activité	18H30	Casier en bon état	Casier 12.5KG	210	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
294	05/10/2021	Fin d'activité	18H30	Casier à entretenir	Casier 12.5KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
295	05/10/2021	Fin d'activité	18H30	Casier à réformer	Casier 12.5KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
296	05/10/2021	Fin d'activité	18H30	Casier en bon état	Casier 35KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
297	05/10/2021	Fin d'activité	18H30	Casier à entretenir	Casier 35KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
298	05/10/2021	Fin d'activité	18H30	Casier à réformer	Casier 35KG		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
299	05/10/2021	Fin d'activité	18H30	Robinet en bon état	Robinet	7100	Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/
300	05/10/2021	Fin d'activité	18H30	Robinet défectueux	Robinet		Ola Energy	Inventaire Régulier	Maurice Florentin TJOUEN NYEM	10/

FIGURE 2.3 – Problème d'incohérence de la base de donnés

2.2.4 Rigidité du Système :

Le système en place démontre une rigidité qui le rend compliquer à adapter aux évolutions futures. La structure actuelle semble peu flexible et ne permet pas d'ajustements ou de modifications aisés pour répondre aux nouveaux besoins. Il est nécessaire de concevoir un système plus adaptable, doté d'une architecture permettant des extensions et des modifications, afin de garantir sa pérennité et sa capacité à évoluer en fonction des besoins futurs.

2.2.5 Type d'Interface :

Actuellement, le fichier BASSA.xlsm présente une interface statique qui ne s'adapte pas aux sélections de l'utilisateur. Les menus et les options accessibles ne changent pas en fonction des choix effectués par l'utilisateur, ce qui limite la flexibilité et l'intuitivité de l'application. Une interface dynamique est importante pour améliorer l'expérience utilisateur en permettant aux choix et aux options d'être ajustés automatiquement en fonction des sélections faites. Cette fonctionnalité offrirait une interaction plus fluide et personnalisée, répondant mieux aux besoins précis des utilisateurs.

2.2.6 Sécurité :

La sécurité du fichier est actuellement insuffisante, permettant à tout utilisateur d'accéder à la base de données. Il est crucial de fortifier la gestion des accès et de mettre en place des mécanismes de sécurité appropriés pour protéger les informations sensibles et éviter les manipulations non autorisées. La mise en œuvre de contrôles d'accès stricts est indispensable pour sécuriser les données et diminuer les risques de compromission.
En conclusion, le fichier BASSA.xlsm présente plusieurs faiblesses remarquables en matière de structure du code, de modularité, de gestion des erreurs, de flexibilité et de sécurité. Il est fortement recommandé de procéder à une révision détaillée de ces aspects afin d'optimiser la fonctionnalité et la sécurité du système de gestion des inventaires

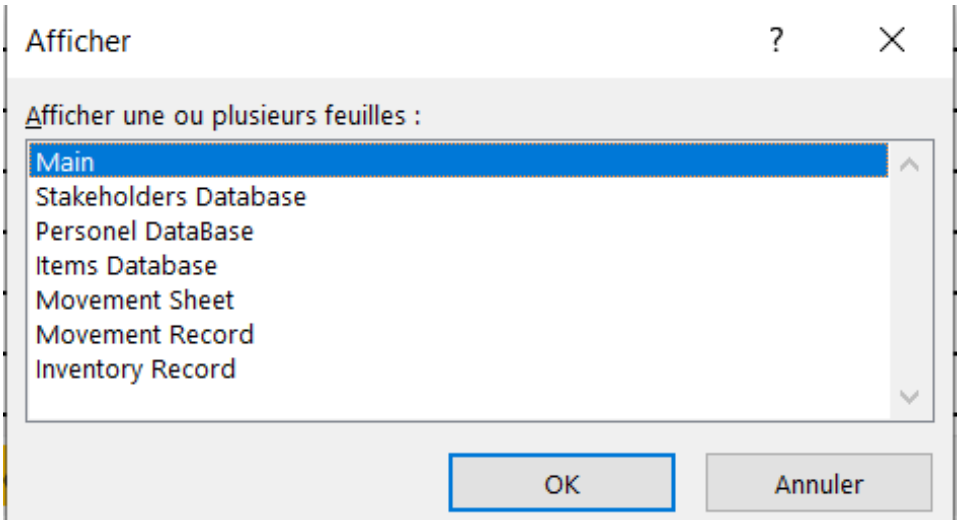

FIGURE 2.4 – Problème de sécurité

2.3 Objectifs principales

L'évaluation du fichier BASSA.xlsm a révélé plusieurs contraintes critiques qui nécessitent une attention particulière pour aligner le système avec les besoins de l'entreprise. Les objectifs primaires doivent être révisés en fonction de ces contraintes afin d'optimiser la gestion des données et d'améliorer l'efficacité globale du système.

2.3.1 Amélioration de la Modularité et de la Maintenance du Code

CONTRAINTES : La structure du code est actuellement peu modulaire, rendant complexe la maintenance et l'évolution du fichier.
ANALYSE : Pour atteindre cet objectif, il est impératif de refondre le code en modules clairement définis, chacun ayant une responsabilité spécifique. Une approche modulaire permet de séparer les préoccupations et facilite la gestion du code en isolant les fonctionnalités distinctes. Cela améliore la lisibilité, clarifie la maintenance et permet des modifications ciblées sans affecter l'ensemble du système. En outre, cette modularité facilite les tests unitaires et le débogage, garantissant ainsi une meilleure qualité du code et une plus grande résilience face aux évolutions futures.

2.3.2 Optimisation de la Flexibilité des Fonctions

CONTRAINTES : Le système utilise des fonctions prédéfinies d'Excel qui ne s'adaptent pas toujours aux besoins spécifiques de l'entreprise.
ANALYSE : Pour répondre à cet objectif, il est important de développer des fonctions personnalisées adaptées aux exigences spécifiques de gestion des données de BASSA. Les fonctions prédéfinies d'Excel, bien que puissantes, peuvent être restreintes dans des scénarios complexes. La création de fonctions sur mesure permettra de traiter des cas particuliers, d'optimiser les performances et d'améliorer la précision des opérations. Par exemple, des fonctions personnalisées peuvent intégrer des logiques métier spécifiques, gérer des formats de données particuliers ou automatiser des tâches complexes non couvertes par les fonctions standard.

2.3.3 Renforcement de la Gestion des Erreurs et de la Cohérence des Données

Contraintes : La gestion des erreurs est limitée, entraînant des incohérences dans la base de données.
Analyse : L'amélioration de la gestion des erreurs doit inclure la mise en œuvre de contrôles et de validations systématiques tout au long du processus de traitement des données. L'objectif est d'introduire des mécanismes de capture des erreurs, des alertes en cas de données incorrectes, et des routines de correction automatique lorsque cela est possible. Cette approche proactive assurera l'harmonie des données en détectant et en rectifiant les erreurs avant qu'elles n'affectent l'intégrité globale du système. Par exemple, des validations de saisie, des vérifications de cohérence et des contrôles de qualité doivent être intégrés pour s'assurer que les données saisies sont correctes et conformes aux exigences.

2.3.4 Amélioration de la Flexibilité et de l'Adaptabilité du Système

Contraintes : Le système actuel est inflexible et difficile à adapter aux évolutions futures.
Analyse : Pour atteindre cet objectif, il est vital de concevoir une architecture système flexible qui permet des ajustements aisés et des extensions futures. La mise en place d'une architecture évolutive, avec des interfaces modulables et des mécanismes d'intégration ouverts, facilitera l'adaptation aux nouvelles rigueures et technologies. Cela pourrait inclure la mise en place de couches d'abstraction, l'utilisation de services web ou d'API pour l'intégration, et la conception de structures de données extensibles. Une telle flexibilité garantit que le système peut évoluer avec les besoins de l'entreprise sans nécessiter de refontes majeures.

2.3.5 Renforcement de la Sécurité des Données

Contraintes : La sécurité du fichier est faible, permettant un accès non autorisé à la base de données.
Analyse : Le renforcement de la sécurité des données implique la mise en œuvre de contrôles d'accès stricts, d'authentifications robustes, et de mécanismes de cryptage des données sensibles. L'objectif est d'assurer que seules les personnes autorisées peuvent obtenir et manipuler les données. Cela exige la configuration de rôles et de permissions pour contrôler l'accès aux différentes sections du système et l'utilisation de protocoles de sécurité pour protéger les informations contre les intrusions. De plus, la mise en place de journaux d'audit et de systèmes de surveillance renforcera la capacité à détecter et à répondre aux tentatives d'accès non autorisées.

2.3.6 Création d'une Interface Dynamique et Intuitive

Contraintes : L'interface préesent est statique et ne s'adapte pas aux choix de l'utilisateur.
Analyse : Pour répondre à cet objectif, il est essentiel de concevoir une interface utilisateur dynamique qui adapte les options et les informations en fonction des actions et des sélections de l'utilisateur. Une interface dynamique améliore l'expérience utilisateur en rendant le système plus dynamique et réactif aux besoins individuels. Cela pourrait inclure des éléments tels que des menus contextuels interactifs, des filtres dynamiques et des champs de saisie qui se mettent à jour en fonction des choix précédents. Cette adaptabilité propose une navigation plus fluide et personnalisée, réduisant ainsi le risque de confusion et d'erreurs, et augmentant l'efficacité de l'utilisation du système.

En conclusion, les objectifs principaux de BASSA doivent être réévalués et améliorés en fonction des contraintes identifiées pour améliorer la fonctionnalité, la sécurité et l'efficacité du système de gestion des inventaires. En adressant ces objectifs de manière approfondie, BASSA pourra développer un système plus résistant, flexible et sécurisé, répondant mieux aux besoins actuels et futurs de l'entreprise.

2.4 Définition des besoins

2.4.1 Gestion des utilisateurs

La définition claire des rôles des utilisateurs est fondamentale pour assurer un fonctionnement optimal, sécurisé et efficace du système de gestion des inventaires. Chaque utilisateur se voit attribuer des permissions précis en fonction de ses responsabilités et de ses besoins opérationnels.

Administrateur

L'administrateur est le gestionnaire suprême du système. Il detient tous les droits nécessaires pour effectuer des opérations de gestion complète sur l'application. Son rôle est central et implique plusieurs responsabilités, notamment :

- **Gestion des utilisateurs** : L'administrateur peut créer, modifier ou supprimer les comptes des autres utilisateurs (manager, magasinier, sales, reconciliation). Il accorde des rôles, met à jour les informations d'accès et ajuste les permissions.
- **Paramétrage du système** : Il est responsable de la configuration et de la personnalisation de l'application selon les exigences de l'organisation, incluant la gestion des accès, des vues, et des fonctions disponibles pour chaque type d'utilisateur.
- **Surveillance et audit** : L'administrateur est en mesure de visualiser les journaux d'audit et d'activité, d'assurer la compatibilité avec les politiques internes et de détecter toute anomalie ou activité suspecte.

L'administrateur est ainsi le garant du bon fonctionnement de l'application et de la sécurité des données, agissant comme un superviseur de haut niveau.

Manager

Le rôle du manager se focalise principalement sur la gestion des opérations au sein de l'application. Contrairement à l'administrateur, ses permissions sont limitées pour des raisons de sécurité et de division des responsabilités :

- **Visualisation et consultation des données** : Le manager peut atteindre toutes les données relatives aux produits, aux mouvements de stock et aux inventaires, lui permettant de suivre en temps réel l'état des stocks.
- **Rapports et analyses** : Il a la capacité de générer des rapports et de réaliser des analyses sur les données disponibles pour la prise de décision stratégique.
- **Coordination des équipes** : Il peut superviser les opérations des autres utilisateurs (magasinier, sales, reconciliation) sans avoir la possibilité de manipuler leurs droits ou leurs accès.

Le manager est donc un acteur clé pour le suivi de l'activité et la prise de décisions, sans pour autant pouvoir modifier les configurations du système.

Magasinier

Le magasinier est le seul utilisateur autorisé à entrer des données dans l'application. Ce rôle est spécialisé et se concentre sur la saisie et la mise à jour des informations liées aux mouvement de stock :

- **Entrée de données** : Il peut ajouter, modifier ou supprimer des données relatives aux produits, aux stocks d'entrée et de sortie, et aux inventaires quotidiens.
- **Gestion des stocks** : En tant que responsable des données d'entrée, le magasinier garantie que les enregistrements sont corrects, à jour et en accord avec la réalité du terrain.

Ce rôle est important pour la fiabilité des données présentes dans l'application, étant donné que le magasinier est le seul utilisateur ayant le droit de manipuler ces données.

Sales

Le rôle de l'utilisateur "Sales" se contente à la consultation des données de la base de données sans aucune capacité de modification. Ce rôle est destiné aux utilisateurs qui ont besoin de se rendre aux informations de l'inventaire pour des analyses, des rapports ou des présentations, sans interférer avec les opérations :

- **Visualisation des données** : L'utilisateur Sales peut se renseigner de toutes les informations de la base de données mais ne peut pas ajouter, modifier ou supprimer les données.
- **Rapports et suivis** : Bien qu'il ne puisse pas changer les données, il peut visualiser des rapports générés et effectuer des suivis sur les ventes et les mouvements de stock.

Le rôle Sales assure ainsi une consultation sécurisée des données pour le personnel impliqué, sans risques de manipulation accidentelle ou malveillante.

Réconciliation

L'utilisateur "Reconciliation" joue un rôle spécifique dans le processus de contrôle

- **Manipulation du tableau de réconciliation** : Cet utilisateur est autorisé à manipuler le tableau de réconciliation, permettant ainsi de vérifier l'adaptabilité des mouvements de stock et de s'assurer que toutes les transactions sont correctement enregistrées.
- **Visualisation des données** : Comme l'utilisateur Sales, l'utilisateur Reconciliation peut également examiner toutes les données de la base de données sans la capacité de les modifier.
- **Contrôle des stocks** : Il a un rôle important dans le contrôle de la gestion des stocks et peut signaler toute anomalie ou erreur de données.

L'utilisateur Reconciliation procure une couche supplémentaire de contrôle et d'assurance qualité, en veillant à ce que les informations stockées soient exactes et conformes aux mouvements physiques des stocks.

La distinction entre les rôles d'administrateur, manager, magasinier, sales, et reconciliation est essentielle pour maintenir une organisation claire et sécurisée dans l'application de gestion des inventaires. Chaque rôle a des responsabilités spécifiques et des permissions limitées en fonction des besoins opérationnels. Cela facilite non seulement d'assurer la sécurité des données et l'intégrité du système, mais aussi de faciliter une utilisation efficace et ciblée de l'application par les différents intervenants. Une telle séparation des tâches est vitale pour une gestion rigoureuse et efficace des inventaires au sein de toute organisation.

2.4.2 Éviter les DEADLOCKS

Introduction

Dans le contexte d'une application de gestion des inventaires, la saisie et la mise à jour des données de stock sont des opérations critiques qui doivent être exécutées de manière fluide et sécurisée. Les transactions concernant la manipulation des données de stock, telles que l'ajout, la modification ou la suppression des enregistrements, doivent être réalisées de manière à éviter les interblocages (deadlocks) qui peuvent paralyser le système et nuire à la continuité des opérations. Ainsi, il est nécessaire de définir des stratégies et des mécanismes efficaces pour prévenir ces situations de blocage et garantir une gestion maximisée des transactions.

Définition du Deadlock

Un deadlock ou interblocage est une situation dans laquelle deux ou plusieurs transactions ou processus se bloquent mutuellement en attendant que l'autre libère une ressource. Dans le contexte d'une application de gestion des stocks, cela peut se produire lorsque plusieurs utilisateurs ou processus tentent d'accéder aux mêmes enregistrements de données, en essayant de les verrouiller pour modification ou mise à jour. Par exemple, si le Magasinier A essaie de mettre à jour l'état de stock d'un produit pendant que le Magasinier B tente de supprimer ce même produit de l'inventaire, un deadlock peut survenir si chaque transaction attend que l'autre délivre le verrou.

Importance de la Prévention des Deadlocks

La prévention des deadlocks est cruciale pour plusieurs raisons :

- **Continuité des Opérations :** Les deadlocks peuvent entraîner des blocages dans les processus de saisie et de mise à jour des données, déstabilisant ainsi la continuité des opérations de gestion des stocks. Éviter ces blocages permet d'assurer un flux constant de transactions et d'opérations critiques, diminuant ainsi le temps d'attente des utilisateurs et les interruptions.
- **Amélioration des Performances du Système :** Lorsque des deadlocks se produisent, les ressources du système sont consommées inutilement par des transactions en attente. Cela peut entraîner une baisse des performances globales de l'application et affecter négativement l'expérience utilisateur. En évitant les deadlocks, on optimise l'utilisation des ressources et on garantit une réactivité accrue du système.
- **Sécurité et Intégrité des Données :** Les deadlocks peuvent également compromettre l'intégrité des données. Lorsque des transactions sont bloquées ou annulées en raison d'interblocages, il y a un risque que certains changements ne soient pas correctement enregistrés ou que les données deviennent confus. La prévention des deadlocks contribue donc à la sécurité des données en garantissant leur exactitude et leur fiabilité.
- **Réduction des Risques de Pertes Financières et Opérationnelles :** Dans un environnement de gestion d'inventaire, où les décisions sont souvent prises en temps réel en fonction des données accessibles, les deadlocks peuvent ralentir des décisions critiques et entraîner des pertes financières et opérationnelles. En éliminant ces blocages, on réduit les risques de pertes et on promet une gestion efficace des stocks.

Stratégies de Prévention des Deadlocks

Pour répondre à ce besoin essentiel, plusieurs stratégies peuvent être mises en œuvre afin d'éviter les deadlocks dans la gestion des transactions de saisie de données :

— **Ordonnancement des Transactions :** Exiger un ordre strict pour l'accès aux ressources peut aider à éviter les deadlocks. Par exemple, les transactions peuvent être conçues de manière à toujours parvenir aux ressources dans un ordre prédéfini. Cela réduit les chances que deux transactions se verrouillent mutuellement en attendant des ressources.
— **Utilisation de Verrouillages Temporels (Timeouts) :** Configurer des mécanismes de temporisation pour les verrous peut empêcher les transactions de rester bloquées indéfiniment. Si une transaction ne parvient pas à obtenir un verrou après un certain délai, elle est annulée ou mise en attente pour être réessayée plus tard, évitant ainsi un interblocage potentiel.
— **Transactions Atomiques et Granulaires :** Diviser les transactions complexes en opérations plus petites et atomiques peut réduire les risques de deadlocks. Les petites transactions sont moins susceptibles de se coincer entre elles et peuvent être traitées plus rapidement.
— **Détection et Reprise après Deadlock :** Mettre en place un mécanisme de détection de deadlock et une stratégie de reprise aide à controler activement les transactions et de prendre des mesures correctives lorsque des deadlocks sont détectés. Cela peut intégrer l'annulation de certaines transactions pour libérer des ressources.
— **Verrouillage Optimiste :** Au lieu d'utiliser un verrouillage pessimiste (qui bloque les ressources dès le début de la transaction), le verrouillage optimiste permet aux transactions de lire les données sans verrouillage, en vérifiant ensuite si des conflits se sont faits avant de finaliser l'opération. Cela baisse la probabilité de deadlocks.

Conclusion

La précaution des deadlocks dans la gestion des transactions de saisie de données est un besoin fondamental pour garantir le bon fonctionnement et la performance d'une application de gestion des stocks. En mettant en œuvre des stratégies efficaces pour éviter ces blocages, on renforce la continuité des opérations, l'intégrité des données, les performances du système, et on réduit les risques de pertes financières. La gestion proactive des deadlocks est donc importante pour toute organisation cherchant à optimiser ses processus de gestion des stocks et à assurer la fiabilité et la sécurité de son système d'information.

2.4.3 Développement d'une application mobile

Introduction

Dans un monde de plus en plus connecté et mobile, les entreprises cherchent à augmenter leur efficacité opérationnelle et à optimiser leurs processus en offrant plus de flexibilité à leurs utilisateurs. Un des besoins majeurs de notre entreprise est de développer une application mobile intégrée capable de réaliser tous les traitements nécessaires liés à la gestion des inventaires, des stocks, des ventes, et des autres opérations. Cette application mobile viendra compléter le système existant sur PC, leur donnant ainsi une solution hybride qui sera plus bénéfique et plus adaptée aux réalités des opérations modernes.

Limites d'un Système Uniquement sur PC

Actuellement, la majorité des opérations de gestion sont effectuées via une application ou un système installé sur des ordinateurs de bureau ou des PC portables. Bien que cette approche ait été bien dans le passé, elle présente plusieurs limitations dans le contexte actuel de la digitalisation et de la mobilité accrue :

— **Manque de Flexibilité et de Mobilité :** Les utilisateurs sont limités à effectuer leurs tâches à partir de leur poste de travail fixe. Cela n'est pas optimal pour les utilisateurs sur le terrain, comme les magasiniers, les agents de vente, ou les responsables de contrôle d'inventaire, qui doivent fréquemment se déplacer dans l'entrepôt, le magasin ou d'autres installations.
— **Accès Limité en Temps Réel :** L'absence de mobilité réduit la capacité à accéder aux informations en temps réel, ce qui est vital pour la prise de décision rapide et la gestion efficace des stocks et des ventes.
— **Adaptabilité Restreinte aux Nouveaux Modèles de Travail :** Avec l'augmentation du télétravail, des déplacements réguliers et des besoins de travail collaboratif, les systèmes exclusivement sur PC deviennent de moins en moins adaptés aux besoins modernes d'adaptabilité et de connectivité.

Avantages d'une Application Mobile Complète

La création d'une application mobile complète, qui gère tous les traitements possibles, présente plusieurs avantages par rapport à un système seulement basé sur PC. Voici les principales raisons pour lesquelles cette approche serait plus bénéfique :

— **Mobilité et Flexibilité Accrues :** Une application mobile permet aux utilisateurs de se rendre au système à tout moment et en tout lieu, offrant ainsi une flexibilité incomparable. Que ce soit pour la saisie des données, le suivi des stocks, la gestion des ventes ou la réconciliation des inventaires, les utilisateurs peuvent réaliser leurs tâches sur le terrain, en déplacement ou à distance, sans être liés à un poste fixe.
— **Mise à Jour des Données en Temps Réel :** Grâce à la connectivité mobile et à la synchronisation via le cloud, les utilisateurs peuvent mettre à jour les informations en temps réel. Cela assure que toutes les données sont toujours à jour, ce qui est important pour la gestion efficace des stocks, l'optimisation des ventes et la prise de décision rapide.
— **Amélioration de l'Efficacité Opérationnelle :** Une application mobile intégrée réduit les temps d'attente et les étapes intermédiaires. Par exemple, un magasinier peut immédiatement scanner des produits, enregistrer les mouvements de stock, et mettre à jour le système sans avoir à retourner à un bureau, ce qui maximise considérablement l'efficacité opérationnelle.
— **Meilleure Expérience Utilisateur :** Les applications mobiles modernes peuvent prroposer une interface utilisateur intuitive, simplifiée et adaptée aux petits écrans, ce qui rend la navigation et l'exécution des tâches plus faciles et plus rapides par rapport aux systèmes PC traditionnels.
— **Intégration de Fonctionnalités Spécifiques au Mobile :** Les applications mobiles peuvent tirer parti de fonctionnalités spécifiques telles que la géolocalisation, les notifications push, la reconnaissance vocale, les caméras pour la lecture de codes-barres ou QR codes, et bien d'autres, qui peuvent optimiser les processus de gestion des stocks et des ventes.

Communication Centralisée via le Cloud et une Base de Données Unique

Pour maximiser l'efficacité et assurer une cohérence parfaite entre les différents systèmes (PC et mobile), il est crucial que les deux applications (mobile et PC) communiquent via un cloud centralisé et partagent une base de données unique. Cette approche offre de nombreux avantages supplémentaires :

— **Synchronisation des Données en Temps Réel :** En centralisant la base de données dans le cloud, toutes les données saisies via l'application mobile ou le système PC sont instantanément synchronisées. Cela garantie une mise à jour instantanée des informations sur tous les dispositifs, éliminant les incohérences et les doublons de données.

— **Accessibilité Globale et Sécurisée :** Un système qui est basé sur le cloud permet aux utilisateurs de se rendre de manière sécurisée aux données et aux fonctionnalités de l'application depuis n'importe où dans le monde, tant qu'ils ont une connexion Internet. Cela est important pour les équipes de vente et les gestionnaires d'inventaire qui opèrent dans différentes régions.
— **Maintenance et Mise à Jour Simplifiées :** En centralisant le système dans le cloud, les mises à jour et la maintenance du logiciel peuvent être réalisées de manière centralisée, ce qui limite le besoin d'interventions sur chaque appareil individuel, que ce soit un PC ou un smartphone.
— **Évolutivité et Adaptabilité :** Une architecture basée sur le cloud donne une grande évolutivité, permettant d'ajuster les ressources et les capacités en fonction de la croissance de l'entreprise et des besoins changeants.
— **Sécurité Renforcée des Données :** Les solutions cloud modernes offrent des niveaux de sécurité élevés, y compris le cryptage des données, les contrôles d'accès granulaires, et les mécanismes de sauvegarde automatique, garantissant que les données de l'entreprise sont préservées contre les pertes ou les accès non autorisés.

Conclusion

Le besoin de développer une application mobile complète qui communique avec le système existant sur PC via un cloud centralisé et une base de données unique est un pas stratégique vers la modernisation des opérations de l'entreprise. Cette solution hybride offre non seulement une flexibilité et une mobilité accrues, mais elle améliore également l'efficacité, la sécurité, et la précision des processus de gestion. En investissant dans cette approche intégrée, l'entreprise pourra répondre de manière plus efficiente aux besoins évolutifs de ses utilisateurs, tout en restant compétitive dans un environnement de plus en plus axé sur la mobilité et la connectivité.

Chapitre 3

Analyse Fonctionelle

3.1 Diagramme de cas d'utlisation

3.1.1 Définition

En langage de modélisation unifié (UML), un diagramme de cas d'utilisation peut servir à résumer les informations des acteurs et leurs actions avec votre système. La création de ce type de diagramme UML utilise un ensemble de symboles et de connecteurs spécifiques. Lorsqu'ils sont bien conçus, les diagrammes de cas d'utilisation peuvent aider à représenter :

- Les scénarios dans lesquels votre système interagit avec des personnes, des organisations ou des systèmes externes ;
- Les objectifs que votre système ou application permet aux entités (appelées acteurs) d'atteindre ;
- La portée de votre système.

Le diagramme de cas d'utilisation utilise trois relations importantes entre les actions qui sont :

- **Generalize :** qui signifie brièvement l'héritage entre les actions
- **Include :** qui signifie la dépendance d'une action à l'autre (il faut qu'une action soit exécuté pour que l'autre soit aussi)
- **Extends :** qui fait référence à un cas spécial pour les actions dérivées du cas normale.

3.1.2 Réalisation

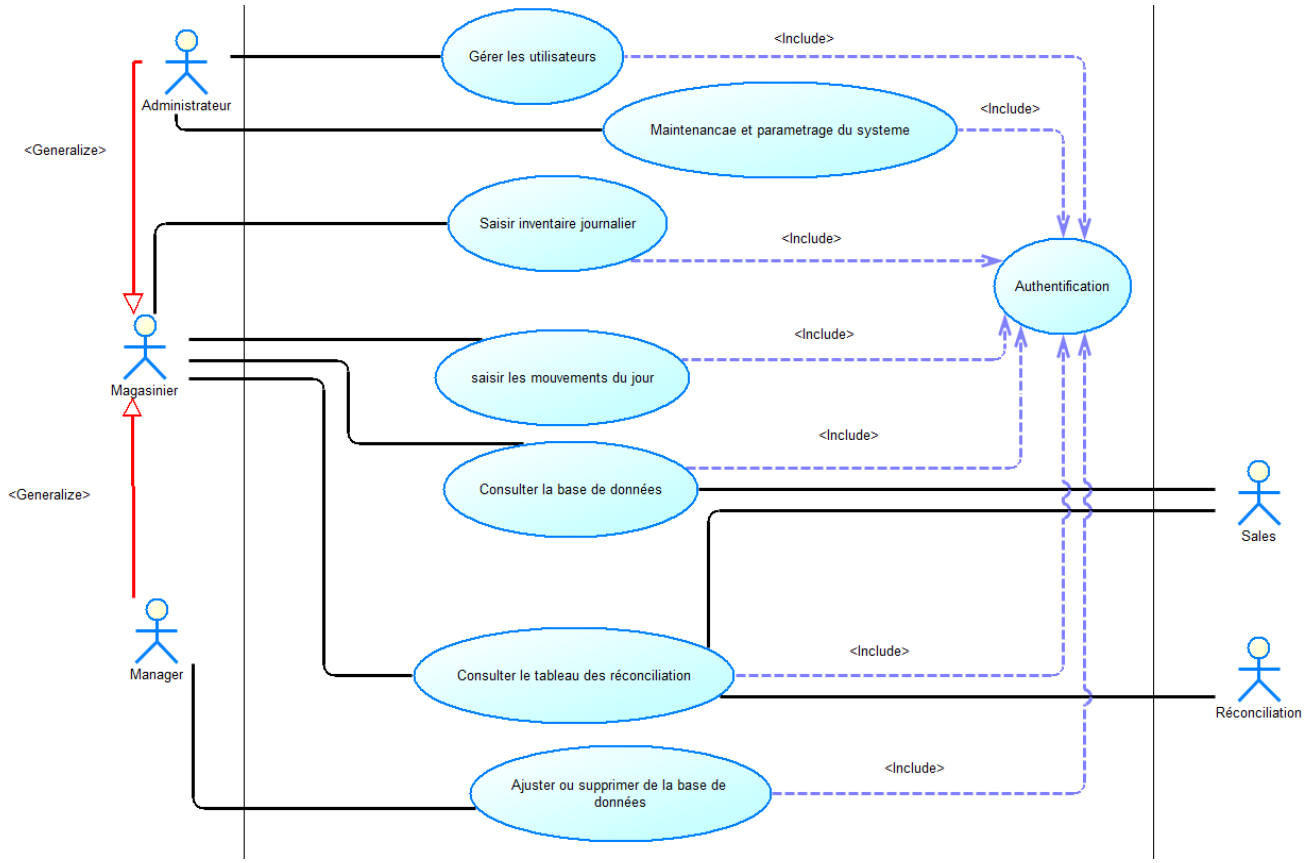

FIGURE 3.1 – Diagramme de cas d'utilisation

3.1.3 Analsye

Introduction :

Ce diagramme de cas d'utilisation représente les fonctionnalités majeures du système, ainsi que les interactions entre les différents types d'utilisateurs et ces fonctionnalités. Les acteurs principaux identifiés sont l'Administrateur, le Magasinier, le Manager, le service des Ventes (Sales), et le service de Réconciliation.

Acteurs :

- **Administrateur :** Responsable de la gestion des utilisateurs et de la maintenance du système. Il dispose des droits les plus élevés et peut réaliser diverses actions critiques comme la gestion des utilisateurs et la maintenance du système.
- **Magasinier :** Un utilisateur du système qui a des responsabilités liées à la saisie des inventaires et des mouvements quotidiens. Ce rôle est généralisé sous celui de l'Administrateur, ce qui suggère que le Magasinier pourrait hériter de certaines fonctionnalités de l'Administrateur.
- **Manager :** Il est aussi généralisé sous le Magasinier, ce qui signifie qu'il peut accomplir les tâches d'un Magasinier en plus de ses propres tâches spécifiques.
- **Sales (Ventes) :** Cet acteur interagit avec le système principalement pour consulter la base de données, probablement pour obtenir des informations pertinentes aux ventes.
- **Réconciliation :** Cet acteur est en charge des activités de réconciliation, c'est-à-dire vérifier et ajuster les enregistrements pour garantir leur exactitude.

Cas d'Utilisation :

Gérer les utilisateurs : Action principalement effectuée par l'Administrateur pour créer, modifier ou supprimer des comptes utilisateurs dans le système.

- **Maintenance et paramétrage du système :** Inclus dans plusieurs autres cas d'utilisation, cette fonctionnalité permet de conserver le bon fonctionnement du système et de configurer ses paramètres pour répondre aux besoins des utilisateurs.
- **Saisir l'inventaire journalier :** Un cas d'utilisation clé pour les Magasiniers, qui repose sur l'enregistrement des stocks journaliers dans le système.
- **Saisir les mouvements du jour :** Impliquant la saisie des transactions quotidiennes, comme les entrées et sorties de stock.
- **Consulter la base de données :** Permet aux utilisateurs de rechercher et d'afficher des informations stockées dans la base de données, vital pour le service des Ventes et pour d'autres utilisateurs cherchant à vérifier des données.
- **Consulter le tableau de réconciliation :** Une fonctionnalité spécifique pour controler les enregistrements, notamment en identifiant et en corrigeant les écarts dans les données.
- **Ajuster ou supprimer de la base de données :** Permet de modifier ou de supprimer des données existantes, une fonctionnalité critique pour la gestion des stocks et la précision des enregistrements.

Relation entre les Cas d'Utilisation :

Le diagramme utilise les relations d'inclusion (« include ») pour indiquer que certaines fonctionnalités dépendent d'autres. Par exemple, l'authentification est un prérequis pour presque toutes les actions, soulignant l'importance de la sécurité et de l'accès contrôlé.

Observations et Recommandations :

- **Sécurité :** Le diagramme montre que l'authentification est un point central, ce qui est essentiel pour garantir que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités critiques.
- **Modularité :** Les rôles d'Administrateur, Magasinier, et Manager sont bien définis, mais il pourrait être utile de clarifier les responsabilités spécifiques de chaque acteur pour éviter toute confusion, surtout avec les généralisations montrées.
- **Évolutivité :** Le système parait conçu pour permettre l'ajout de nouvelles fonctionnalités sans perturber les fonctionnalités existantes, ce qui est un bon indicateur de la flexibilité du système.

Conclusion :

Ce diagramme de cas d'utilisation fournit une vue d'ensemble claire des responsabilités des différents acteurs et des fonctionnalités essentielles du système. Il met en évidence l'importance de la gestion des accès et de la sécurité, tout en permettant une gestion efficace des inventaires et des données critiques. L'analyse met en lumière les aspects fondamentaux du système qui pourraient être améliorés pour renforcer sa robustesse et sa flexibilité.

3.2 Outils utilisés

3.2.1 Microsoft Excel Visual Basic "VBA"

Excel Visual Basic for Applications (VBA) est un langage de programmation événementiel intégré à Microsoft Excel, permettant aux utilisateurs d'automatiser des tâches, de maximiser les flux de travail, et de développer des applications personnalisées directement au sein du tableur. Développé par Microsoft, VBA est une déclinaison spécifique du langage Visual Basic, conçu pour être utilisé dans les applications de la suite Microsoft Office telles qu'Excel, Word, Access, et PowerPoint.

Principales Caractéristiques de VBA dans Excel :

- **Automatisation des Tâches Répétitives :** VBA permet d'enregistrer et de jouer des macros, des séquences d'actions automatisées, pour réaliser des tâches monotones avec un minimum d'intervention manuelle. Par exemple, une macro peut être utilisée pour formater un ensemble de données, générer des rapports quotidiens, ou importer et traiter des fichiers externes.
- **Création de Fonctions Personnalisées :** Outre les fonctions intégrées d'Excel, VBA permet aux utilisateurs de créer des fonctions personnalisées (appelées UDFs - User Defined Functions) qui peuvent être utilisées comme des fonctions natives d'Excel dans les formules de cellules. Cela est spécialement utile pour les calculs complexes ou spécifiques à un domaine particulier.
- **Développement d'Interfaces Utilisateur :** VBA offre la possibilité de créer des formulaires et des interfaces utilisateur personnalisées (UserForms), qui permettent une interaction spontanée avec l'utilisateur final. Ces interfaces peuvent inclure des boutons, des listes déroulantes, des cases à cocher, et d'autres contrôles interactifs, facilitant la saisie et la gestion des données.
- **Accès et Manipulation des Objets Excel :** VBA donne un accès programmatique à tous les objets d'Excel, tels que les cellules, les feuilles de calcul, les classeurs, les graphiques, et les tableaux. Cela permet de manipuler les données et les objets de manière dynamique, par exemple en créant ou modifiant des tableaux croisés dynamiques, en mettant à jour automatiquement des graphiques, ou en consolidant des données provenant de plusieurs feuilles de calcul.
- **Interaction avec d'Autres Applications :** VBA peut également être utilisé pour interagir avec d'autres applications Microsoft Office et même des applications externes, via l'automation OLE (Object Linking and Embedding). Cela permet, par exemple, de transmettre des données d'Excel vers Word pour créer des rapports personnalisés, ou d'envoyer des emails via Outlook basés sur des informations extraites d'une feuille Excel.
- **Gestion des Erreurs et Débogage :** VBA intègre des fonctionnalités de gestion des erreurs qui facilite de capturer et de gérer les erreurs potentielles pendant l'exécution des scripts, assurant ainsi une robustesse accrue des solutions développées. Le débogueur intégré permet également de tester et d'affiner les macros en identifiant les erreurs de code et en visualisant l'exécution en temps réel.

Applications Pratiques de VBA dans Excel :

- **Automatisation de Processus Métier :** Que ce soit pour la génération automatique de factures, la gestion des stocks, ou la planification des ressources, VBA permet d'automatiser des processus complexes, atténuant ainsi le temps et l'effort manuels.

— **Analyse Avancée des Données :** En utilisant VBA, les utilisateurs peuvent écrire des scripts pour analyser de grandes quantités de données, effectuer des modèles statistiques, ou générer des visualisations complexes, tout cela de manière automatisée.
— **Intégration de Données :** VBA peut être utilisé pour extraire, transformer, et charger (ETL) des données à partir de plusieurs sources différentes, qu'elles soient internes à Excel ou provenant de bases de données externes, ce qui en fait un outil puissant pour la gestion et la consolidation des données.

Excel VBA est un outil exceptionnel pour les utilisateurs cherchant à aller au-delà des fonctionnalités standards d'Excel. Il permet non seulement d'automatiser des tâches fastidieuses, mais aussi de développer des solutions sur mesure adaptées aux besoins particuliers de chaque organisation. Grâce à sa flexibilité et à sa puissance, VBA transforme Excel en une plateforme capable de gérer des processus complexes, de renforcer la productivité, et d'apporter une valeur ajoutée significative aux analyses de données et à la gestion de l'information.

FIGURE 3.2 – Logo de Excel VBA

3.2.2 Microsoft PowerAPPS

Power Apps est une suite de produits Microsoft qui permet aux développeurs et aux utilisateurs non techniques de créer des applications personnalisées pour répondre à divers besoins professionnels. La suite Power Apps comprend une variété d'applications, de services et de connecteurs, ainsi qu'une plateforme de données qui, ensemble, créent un environnement de développement instantané d'applications dont l'utilisation ne nécessite que peu ou pas d'expertise en matière de codage.
Power Apps est une suite populaire de services "low-code" utilisés pour créer des applications professionnelles. En plus du développement d'applications à code bas ou sans code, la plateforme combine l'analyse et l'automatisation en libre-service. Le cloud Microsoft Azure héberge la suite d'applications Power Apps qui peut utiliser des données d'entreprise stockées sur une plateforme de données unique telle que Microsoft Dataverse ou dans diverses sources de données telles que SharePoint, Microsoft 365, Dynamics 365 ou Microsoft SQL Server. Des interfaces de programmation d'applications (API) permettent de telles connexions de données.
Les développeurs peuvent utiliser la plateforme Power Apps pour interagir de manière programmatique avec les données et métadonnées sous-jacentes et les intégrer à des données externes. Ils peuvent

également appliquer une logique d'entreprise et créer des connecteurs personnalisés si nécessaire.

Avantages de Power Apps

Microsoft a lancé Power Apps fin 2015 en tant qu'offre de plateforme en tant que service (PaaS). La plateforme et les outils de développement permettent aux utilisateurs non techniques de créer, de gérer et de partager facilement des applications professionnelles. Ils facilitent et accélèrent le processus de création d'applications en permettant à n'importe quel utilisateur de créer une application personnalisée riche en fonctionnalités sans avoir à écrire de code.
L'interface de conception Power Apps permet aux utilisateurs de concevoir et de développer de nouvelles applications sans code. La plateforme Power Apps se caractérise par une conception visuelle intuitive et une fonctionnalité "glisser-déposer". Son interface utilisateur (UI) reproduit des applications telles que Microsoft PowerPoint, que de nombreux utilisateurs connaissent et utilisent avec aisance. La plateforme est extensible et peut interagir avec les données, créer des connecteurs et appliquer une logique métier à toute application créée.
Les applications créées à l'aide de Power Apps offrent des capacités de flux de travail avancées pour convertir les processus manuels en processus automatisés. Ces applications peuvent fonctionner de manière transparente sur les appareils iOS, Android et Windows, offrant ainsi une grande flexibilité et une expérience utilisateur réactive.

Applications créées avec Power Apps

Les utilisateurs professionnels peuvent créer de nombreux types d'applications à l'aide de Power Apps. Deux des plus populaires sont les applications de canevas et les applications basées sur des modèles.
Les applications Canvas peuvent être créées pour le web, les applications mobiles et les applications pour tablettes. Les concepteurs et créateurs d'applications peuvent créer ces applications en utilisant une variété de sources, y compris :

- Sources de données dispersées.
- Une plateforme de données unique telle que Dataverse.
- Une toile entièrement vierge.
- Microsoft AppSource, qui est intégré à la page d'accueil des applications Microsoft 365.

Power Apps offre la flexibilité essentielle pour créer n'importe quel type d'expérience utilisateur et d'interface. Les créateurs d'applications peuvent connecter leurs interfaces à plus de 200 sources de données.
Les développeurs peuvent créer des applications basées sur des modèles à partir des données et des processus de base de l'entreprise dans le Dataverse. Ces applications formalisent des formulaires, des vues, des flux de processus, des règles métier et d'autres composants. Power Apps génère automatiquement des interfaces utilisateur pour les applications basées sur des modèles qui sont réactives sur tous les appareils.
Outre les applications de type Canvas et les applications basées sur des modèles, Power Apps permet aux développeurs de créer des micro-applications appelées "cartes". Ces applications disposent des éléments d'interface utilisateur légers qui peuvent être reproduits dans plusieurs applications sans aucune compétence en matière de codage ou d'informatique. Les cartes peuvent faire apparaître des données d'entreprise par le biais de connecteurs Power Platform. Elles peuvent être personnalisées à l'aide de la logique métier.

FIGURE 3.3 – Logo de PowerApps

3.3 Méthodologie

Dans le cadre de notre projet, la méthodologie de travail adoptée a consisté en deux grandes étapes : la reconstruction du fichier Excel BASSA.xlsm et le développement ultérieur de l'application mobile avec PowerApps. Cette approche structurée a permis de garantir une transition fluide entre les deux plateformes tout en gardant la logique et les fonctionnalités clés.

3.3.1 Reconstruction du Fichier BASSA.xlsm :

La première étape de notre méthodologie a été de reconstruire le fichier BASSA.xlsm **à partir de zéro**. Ce fichier, qui est au cœur de notre système de gestion, a été redéveloppé en tenant compte des objectifs principaux du système, à savoir :

- **Amélioration de la Modularité du Code :** Chaque fonction a été réécrite pour optimiser la réutilisabilité et minimiser les redondances. Cela a permis de structurer le code de manière plus efficace, facilitant ainsi les modifications futures et l'ajout de nouvelles fonctionnalités.
- **Optimisation de la Flexibilité des Fonctions :** Nous avons conçu les fonctions de manière à ce qu'elles puissent être simplement adaptées à différents cas d'utilisation sans nécessiter de modifications essentielles. Par exemple, les formules et les scripts VBA ont été paramétrés pour accepter des variables dynamiques.
- **Renforcement de la Gestion des Erreurs :** La gestion des erreurs a été intégrée dès le début pour s'assurer que le fichier Excel puisse gérer les exceptions de manière robuste. Des mécanismes d'alerte et de log ont été mis en place pour informer les utilisateurs en cas de problème.
- **Amélioration de l'Adaptabilité du Système :** Le fichier a été reconstruit pour être évolutif, capable de gérer l'augmentation des données et des utilisateurs sans compromettre les performances. Cette phase a également inclus l'amélioration des tableaux de bord et des rapports pour fournir une vue d'ensemble plus claire des données.
- **Renforcement de la Sécurité des Données :** Des mesures de sécurité ont été implémentées pour sauvegarder les données sensibles, incluant des contrôles d'accès basés sur les rôles et la protection par mot de passe des macros critiques.
- **Création d'une Interface Utilisateur Dynamique :** Une attention particulière a été portée à la création d'une interface utilisateur conviviale, avec des formulaires interactifs et des contrôles dynamiques permettant une saisie de données plus intuitive et plus rapide.

3.3.2 Développement de l'Application Mobile avec PowerApps :

Une fois le fichier BASSA.xlsm entièrement reconstruit et optimisé, la deuxième étape de notre méthodologie a été de convertir cette logique dans une application mobile développée avec PowerApps. Le choix de PowerApps s'est imposé pour offrir une interface plus moderne et accessible, tout en conservant la logique métier déjà établie dans Excel.

- **Transposition de la Logique Existante :** Nous avons veillé à ce que les fonctionnalités développées dans Excel soient soigneusement reproduites dans PowerApps. Cela a impliqué la réécriture des formules et des scripts dans un environnement adapté aux applications mobiles, tout en conservant la structure de données sous-jacente.
- **Interface Utilisateur Mobile :** L'interface utilisateur a été redessinée pour être cohérent avec les écrans mobiles, en mettant l'accent sur la simplicité et l'efficacité. Les utilisateurs peuvent désormais accéder aux mêmes fonctionnalités depuis leur smartphone, avec une expérience utilisateur optimisée pour les petits écrans.
- **Intégration des Données :** Les données gérées dans l'application Excel ont été synchronisées avec PowerApps, offrant une mise à jour en temps réel des informations entre les deux plateformes. Cette intégration a été rendue possible grâce à l'utilisation de connecteurs et d'API qui assurent une communication fluide entre Excel et PowerApps.
- **Sécurité et Gestion des Accès :** Comme dans le fichier Excel, des mécanismes de sécurité stricts ont été implémentés dans PowerApps pour confirmer que seules les personnes autorisées puissent accéder et manipuler les données critiques.
- **Tests et Validation :** Avant le déploiement final, l'application PowerApps a été précisement testée pour s'assurer qu'elle répondait à toutes les exigences fonctionnelles et qu'elle fonctionnait de manière cohérente avec le fichier Excel. Des ajustements ont été effectués pour garantir que l'expérience utilisateur soit compatible et sans faille.

Cette méthodologie de travail, qui a débuté par la reconstruction minutieuse du fichier Excel BASSA.xlsm, a permis de mettre en place une base solide avant de développer l'application mobile avec PowerApps. En suivant la même logique de travail, nous avons pu controler une continuité fonctionnelle et une cohérence entre les deux plateformes, tout en tirant parti des forces spécifiques de chacune pour améliorer l'efficacité et l'accessibilité du système.

Première partie

Reconstruction du Fichier BASSA.xlsm

Chapitre 4

Conception de l'application desktop

4.1 Répartition du code

Dans l'élaboration de ce projet, nous avons mis un point d'honneur à structurer le code de manière à créer un environnement de travail qui soit non seulement efficace, mais aussi intuitif et évolutif. Cette organisation repose sur une organisation bien pensée entre les feuilles Excel, les modules, et le formulaire de connexion, chacun ayant été conçu pour répondre à des besoins spécifiques tout en s'intégrant harmonieusement dans l'ensemble.

4.1.1 Les Feuilles Excel

Les feuilles Excel sont les fondations visibles de l'application. Chaque feuille a été soigneusement élaborée pour répondre à une fonction spécifique, créant ainsi une interface utilisateur qui est à la fois claire et cohérente. Dans cet espace, les utilisateurs trouvent les outils dont ils ont besoin pour réaliser leurs tâches quotidiennes avec efficacité. Les feuilles ne sont pas seulement des supports de données, mais des espaces de travail dynamiques où les informations sont structurées et accessibles. Elles offrent une vue d'ensemble des différents processus en cours, tout en permettant une navigation fluide entre les différentes sections du projet.

4.1.2 Les Modules

Les modules constituent l'ossature invisible qui soutient toute l'application. Ils ont été conçus pour encapsuler les fonctionnalités de manière à les rendre facilement accessibles et réutilisables à travers l'ensemble du projet. Cette approche modulaire permet de maintenir une grande flexibilité dans le développement et l'évolution de l'application. En isolant les différentes fonctions dans des modules spécifiques, nous avons cherché à créer une architecture de code qui soit à la fois robuste et adaptable, capable de répondre aux besoins actuels tout en étant prête à évoluer avec les exigences futures possible. Chaque module joue un rôle crucial dans le bon fonctionnement de l'application, en assurant que les différentes fonctionnalités s'intègrent parfaitement les unes aux autres, sans redondance ni complexité excessive.

4.1.3 Le Formulaire de Connexion

Le formulaire de connexion représente l'entrée principale de l'application, confirmant que l'accès est contrôlé et sécurisé. Plus qu'un simple écran de connexion, il incarne la première interaction de

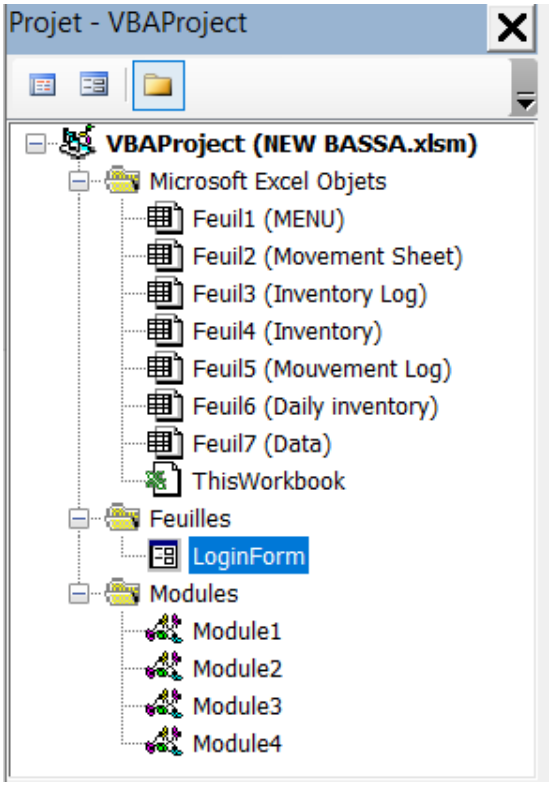

FIGURE 4.1 – Répartition du code

l'utilisateur avec l'application, posant les bases d'une expérience utilisateur sécurisée et fiable. Le formulaire est conçu pour être à la fois simple, facile et efficace, tout en assurant une protection rigoureuse des données. En le plaçant au cœur de l'application, nous avons voulu souligner l'importance de la sécurité et de la confidentialité des informations, tout en offrant une transition fluide vers les autres fonctionnalités une fois l'accès autorisé.

4.2 Mouvement sheet

4.2.1 movementCounters

Le code présenté commence par la déclaration d'une variable nommée movementCounters, qui est définie comme un objet de type Scripting.Dictionary. Cette déclaration est typique dans les langages de programmation VBA (Visual Basic for Applications), utilisés couramment pour automatiser des tâches dans des applications telles qu'Excel.

Définition et rôle de Scripting.Dictionary

Scripting.Dictionary est une collection associative, ou un conteneur de paires clé-valeur. Contrairement aux tableaux traditionnels, où les éléments sont indexés par des entiers, un dictionnaire permet d'utiliser des clés de type chaîne de caractères ou d'autres types, ce qui offre une plus grande souplesse pour accéder et manipuler les données. Dans le contexte d'un projet de gestion d'inventaire, par exemple, un dictionnaire pourrait être utilisé pour stocker les quantités de différents types de produits en utilisant les noms de produits comme clés.

Utilisation de Scripting.Dictionary dans le Projet

Dans le cadre de mon projet de stage, la variable movementCounters est utilisée pour suivre et gérer les mouvements des stocks au quotidien. Chaque type de produit (ou groupe de produits) est associé à une clé spécifique dans le dictionnaire. La valeur associée à chaque clé correspond au nombre de mouvements (entrées ou sorties) pour ce type de produit.
Cette approche permet de :

- Optimiser la gestion des données : Le dictionnaire permet un accès rapide et direct aux données des mouvements de chaque produit, sans avoir à explorer une liste ou un tableau pour les retrouver.
- Faciliter les mises à jour en temps réel : À chaque fois qu'un produit est déplacé (ajout ou retrait du stock), le dictionnaire est mis à jour rapidement, garantissant ainsi que les données sont toujours à jour.
- Améliorer la flexibilité du code : La nature dynamique du dictionnaire permet d'ajouter ou de retirer des produits sans nécessiter de modifications primaires dans la structure du code. Cela est particulièrement utile dans un environnement de travail où les types de produits peuvent changer fréquemment.

Avantages pour le Projet de Gestion des Stocks

Lutilisation d'un Scripting.Dictionary pour suivre les mouvements de stocks présente plusieurs avantages :

- **Efficacité :** Le dictionnaire permet une gestion des mouvements plus rapide et plus efficace que les structures de données traditionnelles, car il offre un accès direct aux données via les clés.
- **Réduction des erreurs :** Grace à la nature associative du dictionnaire, le risque d'erreurs liées à l'accès aux données est réduit ; car il n'y a pas de confusion possible entre les index numériques.
- **Extensibilité :** Le code utilisant un dictionnaire est facilement extensible. Si de nouveaux types de produits sont ajoutés à l'inventaire, il suffit de les ajouter comme nouvelles clés dans le dictionnaire, sans nécessiter de restructuration du code deja existant.

En résumé, la déclaration de la variable movementCounters en tant que Scripting.Dictionary représente une forte approche et flexible pour gérer les mouvements de stocks dans le cadre du projet de stage. Cette technique améliore non seulementt la performance du code, mais aussi sa maintenabilité et sa robustesse, tout en garantissant une gestion précise et à jour des données d'inventaire.

4.2.2 InitializeCounters

La fonction `InitializeCounters`, écrite en VBA (Visual Basic for Applications), joue un rôle clé dans linitialisation et la gestion des compteurs de mouvements de stock au sein de l'application de gestion d'inventaire que je développe dans le cadre de mon stage. Cette fonction est responsable de la création et de la configuration initiale d'un dictionnaire, `movementCounters`, qui est utilisé pour suivre les différents types de mouvements de produits au quotidien.

Structure de la Fonction

La fonction commence par la déclaration de la variable `movementCounters` en tant qu'objet de type `Scripting.Dictionary`. Ensuite, elle initialise ce dictionnaire en le remplissant avec des paires clé-

valeur, où chaque clé représente un type spécifique de mouvement de stock, et la valeur associée est initialisée à zéro.

Étapes d'Initialisation

Voici un aperçu détaillé des étapes effectuées par la fonction :
-Création du Dictionnaire Création d'un dictionnaire pour stocker les compteurs de mouvements, permettant d'associer des clés uniques à des valeurs correspondantes. Cette ligne de code crée une instance du dictionnaire `movementCounters`. Le dictionnaire est un conteneur de données où chaque élément est constitué d'une paire clé-valeur. Ici, `CreateObject("Scripting.Dictionary")` est utilisé pour instancier un nouvel objet de type `Scripting.Dictionary`, qui servira à stocker les compteurs de différents types de mouvements de stock.
-Ajout des Clés et Initialisation des Valeurs Ajout de clés dans le dictionnaire movementCounters, initialisant chaque type de mouvement avec un compteur à 0 pour suivre les occurrences de ces mouvements.
Chaque clé est ajoutée au dictionnaire `movementCounters` avec une valeur initiale de 0.
Chaque appel à la méthode `Add` du dictionnaire ajoute une nouvelle paire clé-valeur. Les clés sont des identifiant pour différents types de mouvement, et les valeurs associées sont démarrées à zéro, indiquant qu'aucun mouvement de ce type na encore été enregistré.

Description des Clés :

- `"TRFEMB"` (Transfert d'emballages) : Compte les mouvements de transfert des emballages entre différentes zones ou entités.
- `"EXPEMB"` (Expédition d'emballages) : Suivi des emballages qui sont expédiés en dehors du site.
- `"LIVPRD"` (Livraison de produits) : Comptabilise les livraisons de produits aux clients ou entre les sites.
- `"RCPPRD"` (Réception de produits) : Enregistre les réceptions de produits entrants sur le site.
- `"RCPEMB"` (Réception d'emballages) : Suivi des emballages reçus sur le site.
- `"RCPACC"` (Réception d'accessoires) : Compte les accessoires reçus en stock.
- `"EXPACC"` (Expédition d'accessoires) : Suivi des accessoires expédiés.
- `"TRFACC"` (Transfert d'accessoires) : Comptabilise les transferts d'accessoires entre les sites ou départements.

-Importance et Utilisation dans le Projet La fonction `InitializeCounters` est essentielle pour garantir que tous les types de mouvement de produits sont justement suivis dès le démarrage de l'application. En initialisant les valeurs à zéro, la fonction garantit que chaque compteur commence à une valeur neutre, prête à être incrémentée au fur et à mesure que des mouvements sont enregistré. Cette méthode d'initialisation assure également que le dictionnaire contient uniquement les types de mouvements pertinents pour l'application, évitant ainsi toute confusion ou erreur potentiel liée à des types de mouvements non pris en charge.

Avantages de l'Approche Utilisée

- **Clarté et lisibilité du code :** L'utilisation d'un dictionnaire avec des clés explicites permet une gestion claire et lisible des différents types de mouvements. Cela rend le code plus facile à comprendre et à maintenir.

— **Facilité de mise à jour :** Si de nouveaux types de mouvements doivent être ajouté à l'avenir, il suffit de les ajouter dans la fonction `InitializeCounters`, sans avoir à restructurer l'ensemble du code.
— **Évolutivité :** Cette approche est évolutive et peut être simplement adapté pour inclure de nouveaux types de mouvements ou pour être utilisée dans d'autres contextes de suivi des données.

En conclusion, la fonction `InitializeCounters` constitue une étape fondamentale dans l'initialisation de l'application de gestion des stocks développée durant mon stage. Elle permet de préparer efficacement le terrain pour la gestion des mouvements de produits ; en s'assurant que chaque type de mouvements est correctement suivi dès le début. Cette fonction contribue de manière significative à la robustesse et à la fiabilité de l'application.

4.2.3 Worksheet Activate

La fonction `Worksheet_Activate` en VBA (Visual Basic for Applications) est un événement qui se déclenche systématiquement à chaque fois que la feuille de calcul associé est activé, cela veut dire que lorsque l'utilisateur clique sur cette feuille ou y navigue depuis une autre feuille. Cette fonction est essentielle pour configurer et mettre à jour dynamiquement les validations de données sur certaines cellules ; garantissant que l'utilisateur dispose d'options précises et pertinentes lors de la saisie de données.

Structure et Rôle de la Fonction

La fonction commence par la déclaration de plusieurs variables, dont `articles`, `categories`, et `FileType`, qui stockent des tableaux de chaîne de caractères. Ces tableaux sont ensuite utilisés pour créer des listes déroulantes dans des cellules spécifiques de la feuille de calcul ; via la fonctionnalité de validation des données d'Excel.

Déclaration des Variables

— `articles` : Contient une liste des articles disponibles pour la sélection.
— `categories` : Stocke les différentes catégories auxquelles ces articles peuvent appartenir.
— `FileType` : Contient une liste des types de fichiers ou formulaires pertinents pour les enregistrements.

Initialisation des Tableaux

Définition de tableaux contenant différentes catégories d'entités, types d'articles, et types de fichiers pour organiser et gérer les données dans le système. Ces tableaux permettent de structurer les informations en groupes spécifiques comme les catégories de clients, les types de bouteilles ou pièces, et les types de documents.
Ces tableaux sont remplis avec des valeurs prédéfinis qui correspondent aux divers options disponibles pour les utilisateurs lors de la saisie de données. Chaque tableau regroupe des éléments similaires pour simplifier l'organisation des options.

Configuration des Listes de Validation de Données

Le cœur de la fonction réside dans la création et la configuration des listes déroulantes pour certaines cellules à l'aide de la validation des données. Cela permet de garantir que seules les valeurs valides peuvent être saisies par lutilisateur dans ces cellules.

- **Validation pour la Cellule L6 (Type de Fichier)**
  Cette partie du code configure la cellule `L6` pour qu'elle affiche une liste déroulante contenant les elements du tableau `FileType`. Avant de créer la validation, toute validation existante est supprimée pour éviter les conflits.
- **Validation pour les Cellules G42 et G43 (Articles)**
  Les cellules `G42` et `G43` sont configurées de manière similaire à `L6`, mais cette fois en utilisant les `articles` comme options disponibles. Ceci est crucial pour garantir que seuls les articles valides peuvent être sélectionnés.
- **Validation pour les Cellules H20 et L20 (Catégories)**
  Les cellules `H20` et `L20` sont également configurées pour afficher une liste déroulante des `catégories` disponibles. Cela permet aux utilisateurs de choisir facilement la catégorie appropriée lors de la saisie des données.

Mise à Jour Dynamique des Colonnes

Boucle à travers les lignes 42 à 80, appelant la fonction UpdateDColumn pour chaque ligne afin de mettre à jour la colonne D avec les données ou calculs nécessaires.
Cette boucle `for` itère sur les lignes `42` à `80`, appelant une fonction (probablement définie ailleurs) nommée `UpdateDColumn`, qui met à jour activement les validations ou autres paramètres des cellules dans la colonne `D` pour les lignes spécifiées. Cela assure que les validations sont actualisées en fonction des données actuelles ou des besoins spécifiques de chaque ligne.

Appel de la Fonction `CreateDropDownListFromColumnM`

Appel de la procédure CreateDropDownListFromColumnM pour créer une liste déroulante en se basant sur les données de la colonne M.
Cette dernière ligne appelle une autre fonction nommée `CreateDropDownListFromColumnM`, qui est chargée de créer une liste déroulante basé sur les valeurs présentes dans la colonne `M`. Cette fonctionnalité est utile pour dynamiser davantage la feuille de calcul, et assurer une meilleure flexibilité dans la gestion des données.

4.2.4 IncrementCounter

La fonction `IncrementCounter` en VBA (Visual Basic for Applications) est une procédure essentielle pour la gestion des compteurs de mouvements dans le cadre d'un projet de suivi d'inventaire. Son rôle principal est d'incrémenter un compteur spécifique associé à un type de mouvement donné. Cette fonction est conçue pour être à la fois robuste et adaptable, en veillant à ce que les compteurs soient correctement initialisés avant d'être modifiés.

Importance et Utilisation dans le Projet

La fonction `IncrementCounter` est cruciale dans le cadre du projet de gestion d'inventaire que nous développons. Chaque fois qu'un mouvement de stock est enregistré, cette fonction est appelée

pour mettre à jour le compteur correspondant. Cela permet de suivre en temps réel le nombre de mouvements pour chaque type de produit ou d'article.

- **Suivi en temps réel** : En permettant l'incrémentation immédiate des compteurs, la fonction contribue à un suivi en temps réel des opérations, ce qui est essentiel pour une gestion efficace des stocks.
- **Flexibilité** : L'usage de la fonction `InitializeCounters` pour l'initialisation permet à la fonction `IncrementCounter` d'être utilisée à tout moment, même si le dictionnaire n'est pas encore prêt. Cela rend le système flexible et capable de s'adapter à différents scénarios d'utilisation.

Avantages de l'Approche Utilisée

- **Sécurité** : La vérification de l'initialisation et de l'existence des clés assure que la fonction fonctionne toujours dans un état valide, réduisant les risques d'erreurs.
- **Modularité** : La fonction est conçue de manière modulaire, ce qui permet une réutilisation simple et efficace dans d'autres parties du code ou dans d'autres projets nécessitant une gestion similaire des compteurs.
- **Simplicité** : L'approche simple et directe de l'incrémentation rend la fonction facile à comprendre et à maintenir.

4.2.5 SetValidationList

La fonction `SetValidationList` en VBA (Visual Basic for Applications) est conçue pour configurer une liste de validation de données sur une feuille Excel spécifique. Cette fonctionnalité est cruciale pour garantir que les utilisateurs sélectionnent des options prédéfinies dans une cellule donnée, ce qui réduit les erreurs d'entrée et améliore l'adaptabilité des données. Dans le cadre de mon projet de stage, cette fonction est utilisée pour garantir une gestion précise et contrôlée des types de mouvements de stock.

Structure de la Fonction

La fonction `SetValidationList` configure une liste déroulante dans une cellule spécifique en utilisant une liste d'options prédéfinies. Elle commence par définir les variables nécessaires, puis passe à la création de la liste de validation de données.

- **Définition des Variables** : Les variables `ws`, `validationArray`, et `nameMapping` sont initialisées pour stocker la feuille de calcul cible, le tableau de validation, et le mappage des noms respectivement.
  - `ws` : Représente la feuille de calcul où la validation sera appliquée.
  - `validationArray` : Contient les options de validation qui seront affichées dans la liste déroulante.
  - `nameMapping` : Un tableau à deux dimensions qui associe les descriptions textuelles des mouvements avec leurs abréviations correspondantes.
- **Mappage des Noms** : Le tableau `nameMapping` associe les descriptions textuelles des mouvements de stock à leurs abréviations respectives. Ce mappage permet de garantir que seules les options valides sont disponibles pour la sélection, facilitant ainsi l'identification rapide des types de mouvements dans la gestion des stocks.

— **Sélection de la Feuille de Calcul** : La feuille de calcul où la validation sera appliquée est définie en utilisant `Set ws`. Ici, la feuille s'appelle "Movement Sheet". Si la feuille spécifiée n'existe pas, un message d'erreur s'affiche et la fonction s'arrête pour éviter des erreurs d'exécution.
— **Création de la Liste de Validation** : L'étape suivante consiste à créer un tableau unidimensionnel, `validationArray`, qui contient les descriptions textuelles des mouvements (la première colonne de `nameMapping`). Ce tableau est ensuite utilisé pour générer la liste de validation.
— **Application de la Validation de Données** : La validation de données est appliquée à la cellule `L7` de la feuille spécifiée. La validation de données est d'abord supprimée (s'il en existe), puis une nouvelle validation de type `xlValidateList` est ajoutée, permettant à l'utilisateur de sélectionner uniquement les options définies dans `validationArray`.
  — **Effacement des validations existantes** : Le code commence par supprimer toute validation déjà présente dans la cellule, ce qui assure que seule la nouvelle validation définie sera en place.
  — **Ajout de la liste de validation** : La liste déroulante est ensuite ajoutée avec les options définies, ce qui facilite la sélection correcte des types de mouvements.

Importance et Utilisation dans le Projet

La fonction `SetValidationList` est essentielle pour la gestion des mouvements de stock dans le cadre de mon projet de stage. En limitant les choix possibles à une liste prédéfinie, cette fonction réduit les risques d'erreurs lors de la saisie des données et assure la cohérence des enregistrements.
— **Réduction des erreurs** : En utilisant des listes déroulantes, la fonction réduit les choix possibles, ce qui diminue considérablement les erreurs de saisie.
— **Cohérence des données** : Les options étant standardisées, toutes les entrées pour un type de mouvement unique seront identiques, ce qui améliore la qualité des données.
— **Facilité d'utilisation** : Les utilisateurs peuvent facilement sélectionner le bon type de mouvement grâce à la liste déroulante, rendant le processus plus rapide et plus intuitif.

Avantages de l'Approche Utilisée

— **Robustesse** : La vérification de l'existence de la feuille de calcul avant d'appliquer la validation confirme que le code ne génère pas d'erreurs inattendues.
— **Flexibilité** : Le mappage des noms dans un tableau permet d'ajouter facilement de nouveaux types de mouvements ou de modifier les existants sans avoir à changer le code principal.
— **Simplicité et Maintenabilité** : Le code est écrit de manière à être facilement compréhensible et modifiable, ce qui est crucial pour la maintenance future.

4.2.6 Worksheet Change

La fonction `Worksheet_Change` est un gestionnaire d'événements dans VBA qui se déclenche automatiquement lorsqu'une modification est apportée à une cellule de la feuille de calcul. Cette fonction est un élément clé dans la gestion active des données de l'inventaire au sein du projet. Voici une explication détaillée de son fonctionnement.

Objectif Principal

La fonction `Worksheet_Change` surveille et gère les modifications apportées à certaines cellules spécifiques de la feuille de calcul. Elle effectue diverses actions basées sur ces changements, telles que la mise à jour des listes de validation, l'incrémentation des compteurs de mouvement, et l'affichage de messages d'avertissement lorsque des conditions spécialisées sont remplies.

Détection des Modifications

La fonction commence par verifier si les modifications se sont produites dans des plages données, telles que les colonnes G, H, et L, et prend des mesures en conséquence :

- **Colonne G42** : une fois que des changements sont détectés dans cette plage, la fonction appelle `UpdateDColumn` pour mettre à jour les données de la colonne D correspondant à la ligne modifiée.
- **Cellule L7** : Si une modification est apportée à cette cellule, la fonction récupère la valeur sélectionnée, la compare à une liste prédéfinie de types de mouvements (nom court) et incrémente un compteur spécifique via la fonction `IncrementCounter`. Le compteur mis à jour est affiché dans la cellule L8.
- **Cellules H19 et L19** : Toute modification dans ces cellules provoque la mise à jour des cellules H20 et L20, respectivement, en appelant des fonctions dédiées `UpdateH20BasedOnL19` et `UpdateL20BasedOnL19`.
- **Colonne Q (à partir de Q21)** : Lorsque des modifications sont apportés dans cette colonne, les cellules correspondantes dans les colonnes R et S sont effacées ; et la fonction `CheckQAndRetrieveEmailAndPost` est appelé pour mettre à jour ces colonnes en fonction des nouvelles données saisies dans Q.

Gestion des Avertissements

La fonction inclut également une vérification pour s'assurer que les cellules L20 et H20 ne contiennent pas les mêmes valeurs, ce qui pourrait entraîner des incohérences dans les données d'inventaire (par exemple, la destination ne peut pas être similaire à la source). Si une telle situation est repérée, un message d'avertissement est affiché, et la cellule L20 est effacée.

4.2.7 UpdateDColumn

La fonction `UpdateDColumn` est un sous-programme en VBA (Visual Basic for Applications) qui a été développée pour gérer la validation dynamique des données dans une feuille de calcul Excel intitulée *Movement Sheet*. Cette fonction est conçue pour mettre à jour la validation de données dans une cellule de la colonne H en fonction de la valeur sélectionnée dans la colonne G pour une ligne donnée. Voici une explication détaillée de chaque étape de la fonction, accompagnée de la logique sous-jacente et de son utilité dans le contexte du projet.

Objectif de la Fonction

L'objectif principal de `UpdateDColumn` est de s'assurer que les options disponibles pour une cellule donnée dans la colonne H (où les choix d'emballage sont effectués) correspondent précisément au type de produit ou d'emballage sélectionné dans la colonne G. Cette validation dynamique optimise l'intégrité des données et réduit le risque d'erreurs de saisie en limitant les choix possibles aux options pertinentes pour le produit ou l'emballage en question.

Explication du Code

- **Initialisation des Variables** : La fonction commence par déclarer et initialiser plusieurs variables :
  - **ws** : Représente la feuille de calcul *Movement Sheet*.
  - **selectedValue** : Stocke la valeur sélectionnée dans la colonne G.
  - **emballageOptions** : Un tableau qui contiendra les options spécifiques de validation pour la cellule de la colonne H.
  - **validationList** : Une chaîne de caractères qui contient les options de validation sous forme de liste séparée par des virgules.
  - **previousValue** : Stocke la valeur précédente de la colonne H pour vérifier si elle doit être réinitialisée.
- **Validation de la Plage de Lignes** : La fonction vérifie si le numéro de ligne (`rowNum`) passé en paramètre est compris entre 42 et 80. Si ce n'est pas le cas, la fonction se termine immédiatement avec `Exit Sub` pour éviter des erreurs de traitement en dehors de la plage prévue.
- **Récupération des Valeurs Précédentes et Sélectionnées** : Le programme récupère la valeur précédente dans la colonne H (`previousValue`) et la valeur sélectionnée dans la colonne G (`selectedValue`) pour la ligne spécifiée. Si la valeur sélectionnée diffère de la valeur précédente, la cellule de la colonne H est réinitialisée (son contenu est effacé).
- **Détermination des Options d'Emballage** : La fonction utilise une structure `Select Case` pour déterminer les options d'emballage spécifiques à afficher en fonction de la valeur sélectionnée dans la colonne G. Chaque cas correspond à un type particulier de produit ou d'emballage, et le tableau `emballageOptions` est rempli avec les choix appropriés. Par exemple :
  - Pour une *Bouteille 6KG*, les options incluent *Bouteille pleine*, *Bouteille vide en service*, etc.
  - Pour un *Robinet*, les options sont *Robinet en bon état* et *Robinet défectueux*.

  Si l'option sélectionnée ne nécessite qu'une seule valeur (par exemple *Ecrou*), cette valeur est instantanément définie dans la colonne H.
- **Création et Application de la Liste de Validation** : Si le tableau `emballageOptions` contient des valeurs, celles-ci sont converties en une liste séparée par des virgules (`validationList`). La validation de données de la cellule de la colonne H est ensuite mise à jour pour refléter cette liste. Toute validation déjà existante est d'abord supprimée avant d'appliquer la nouvelle liste pour éviter les conflits ou les erreurs.
- **Gestion des Cas où la Liste est Vide** : Si la liste de validation est vide, la fonction imprime un message de débogage indiquant qu'aucune option n'a été trouvée pour la valeur sélectionnée. Cela permet de détecter et de corriger des erreurs possibles lors du développement.

4.2.8 UpdateH20BasedOnL19

La fonction `UpdateH20BasedOnL19` est une procédure VBA (Visual Basic for Applications) conçue pour gérer de manière dynamique la validation des données dans une cellule spécifique d'une feuille de calcul Excel, en fonction de la valeur sélectionnée dans une autre cellule. Plus précisément, cette fonction met à jour la validation des données dans la cellule H20 de la feuille de calcul *Movement Sheet* en fonction de la valeur choisie dans la cellule H19, en appliquant des listes nommées appropriées.

Objectif de la Fonction

L'objectif principal de la fonction `UpdateH20BasedOnL19` est d'adapter la liste déroulante des options disponibles dans la cellule H20 selon le type de catégorie sélectionnée dans la cellule H19. Cela garantit que les utilisateurs ont accès uniquement aux options pertinentes et appropriées, ce qui réduit les erreurs de saisie et améliore l'intégrité des données.

Description des Étapes du Code

- **Initialisation des Variables** :
  - La fonction commence par déclarer et initialiser plusieurs variables :
    - **ws** : variable de type `Worksheet` qui fait référence à la feuille de calcul active nommée *Movement Sheet*.
    - **selectedValue** : variable de type `String` qui stocke la valeur sélectionnée dans la cellule H19.
    - **namedRange** : variable de type `String` qui contient le nom de la plage nommée à utiliser pour la validation de la cellule H20.
- **Récupération de la Valeur Sélectionnée** :
  - La fonction récupère la valeur présente de la cellule H19 dans la variable `selectedValue`.
  - Cette valeur détermine quelles options seront disponibles pour la cellule H20.
- **Suppression de la Validation Existante** :
  - Avant de mettre à jour la validation des données dans H20, toute validation existante est supprimée pour éviter les conflits.
  - Cela est accompli avec la méthode `Validation.Delete`.
- **Détermination de la Plage Nommée Basée sur la Valeur Sélectionnée** :
  - Une structure `Select Case` est utilisée pour associer la valeur sélectionnée dans H19 à une plage nommée spécifique :
    - Par exemple, si la valeur est *StationDeService*, la plage nommée *StationSRV* est attribuée à `namedRange`.
    - De même, d'autres valeurs comme *ClientB2B*, *MagasinGPL*, *DépôtGPL*, *Fournisseur*, et *Prestataire* sont mappées à leurs plages nommées respectives.
  - Si la valeur sélectionnée dans H19 ne correspond à aucun des cas spécifiés, `namedRange` reste vide.
- **Application de la Validation de Données** :
  - Si une plage nommée valide est déterminée, la fonction confirme d'abord l'existence de cette plage pour réduire les erreurs.
  - Si la plage existe, une validation de données est ajoutée à la cellule H20.
  - Cette validation utilise la plage nommée déterminée précédemment comme source pour la liste déroulante.
  - La validation est configurée pour permettre la sélection directe dans la cellule (grâce à `InCellDropdown = True`) et pour afficher des messages d'erreur et d'entrée le cas échéant.
- **Prise en Compte des Cas avec une Seule Option** :
  - Si la plage nommée ne contient qu'une seule cellule, cette valeur particulière est automatiquement définie dans H20.
  - Si la plage contient plusieurs valeurs, H20 est laissée vide pour permettre à l'utilisateur de faire un choix.
- **Gestion des Erreurs et Messages** :
  - La fonction inclut une gestion d'erreurs simple pour alerter l'utilisateur si la plage nommée

spécifiée n'existe pas.
- Un message d'avertissement apparaît alors pour informer de cette situation.
- Si `namedRange` est vide, la cellule H20 est simplement réinitialisée à une valeur vide.

4.2.9 UpdateL20BasedOnL19

La fonction `UpdateL20BasedOnL19` est un sous-programme VBA (Visual Basic for Applications) conçu pour gérer automatiquement la validation des données dans une cellule spécifique d'une feuille de calcul Excel, en fonction de la valeur donnée dans une autre cellule. Plus précisément, cette fonction met à jour la validation des données dans la cellule L20 de la feuille de calcul *Movement Sheet* en fonction de la valeur choisie dans la cellule L19, en appliquant des plages nommées appropriées.

Objectif de la Fonction

L'objectif principal de cette fonction est d'adapter la liste déroulante des options disponibles dans la cellule L20 selon le type de catégorie sélectionné dans la cellule L19. Cela permet de garantir que les utilisateurs n'ont accès qu'aux options pertinentes et appropriées, réduisant ainsi les erreurs de saisie et améliorant l'intégrité des données.

Étapes Clés du Code

- **Initialisation des Variables** :
  - La fonction commence par déclarer et initialiser plusieurs variables :
    - **ws** : de type `Worksheet`, elle fait référence à la feuille de calcul active nommée *Movement Sheet*.
    - **selectedValue** : de type `String`, elle stocke la valeur sélectionnée dans la cellule L19.
    - **namedRange** : de type `String`, elle contient le nom de la plage nommée à utiliser pour la validation des données dans la cellule L20.
- **Récupération de la Valeur Sélectionnée** :
  - La fonction récupère la valeur actuelle de la cellule L19 dans la variable `selectedValue`.
  - Cette valeur détermine quelles options seront disponibles pour la cellule L20.
- **Suppression de la Validation Existante** :
  - Avant de mettre à jour la validation des données dans L20, toute validation existante est supprimée à l'aide de la méthode `Validation.Delete`.
  - Cela confirme que la nouvelle validation sera appliquée sans conflit.
- **Détermination de la Plage Nommée Basée sur la Valeur Sélectionnée** :
  - Une structure `Select Case` est utilisée pour associer la valeur sélectionnée dans L19 à une plage nommée spécifique :
    - Par exemple, si la valeur est *StationDeService*, la plage nommée *StationSRV* est attribuée à `namedRange`.
    - D'autres valeurs comme *ClientB2B*, *MagasinGPL*, *DépôtGPL*, *Fournisseur*, et *Prestataire* sont mappées à leurs plages nommées respectives.
  - Si la valeur sélectionnée dans L19 ne correspond à aucun des cas spécifiés, `namedRange` reste vide, ce qui signifie qu'aucune validation spécifique ne sera appliquée.
- **Application de la Validation de Données** :
  - Si une plage nommée valide est déterminée, la fonction vérifie d'abord l'existence de cette plage pour éviter les erreurs.

  - Si la plage existe, une validation de données est insérée à la cellule L20, utilisant la plage nommée déterminée précédemment comme source pour la liste déroulante.
  - La validation est configurée pour permettre la sélection directe dans la cellule, ainsi que pour afficher des messages d'entrée et des erreurs si essentiel.
- **Gestion des Cas avec une Seule Option** :
  - Si la plage nommée ne contient qu'une seule cellule, cette valeur unique est dynamiquement définie dans L20.
  - Si la plage contient plusieurs valeurs, L20 est laissée vide pour permettre à l'utilisateur de faire un choix parmi les options disponibles.
- **Gestion des Erreurs et Messages** :
  - La fonction inclut une gestion d'erreurs simple pour alerter l'utilisateur si la plage nommée spécifiée n'existe pas.
  - Un message d'avertissement s'affiche alors pour informer l'utilisateur de cette situation.
  - Si `namedRange` est vide, la cellule L20 est tout simplement réinitialisée à une valeur vide.

4.3 Inventory

4.3.1 Workbook Open

La fonction `Workbook_Open` est un événement déclenché automatiquement à chaque fois que le classeur Excel est ouvert. Ce sous-programme VBA (Visual Basic for Applications) est conçu pour définir des valeurs par défaut dans certaines cellules précises de la feuille de calcul `Inventory` lorsque le classeur est ouvert. Cette automatisation permet de préparer le classeur avec des données prédéfinies sans nécessiter d'interaction manuelle de la part de l'utilisateur.

Objectif de la Fonction

L'objectif principal de la fonction `Workbook_Open` est de garantir que certaines cellules dans la feuille de calcul `Inventory` sont initialisées avec des valeurs spécifiques à chaque ouverture du classeur. Cela offre une cohérence des données et permet aux utilisateurs de commencer avec des informations de base déjà renseignées, facilitant ainsi la saisie et l'analyse ultérieure.

Description des Étapes du Code

- **Déclaration de l'Événement `Workbook_Open`**
  Le sous-programme est défini comme `Private Sub Workbook_Open()`, ce qui signifie qu'il s'exécute automatiquement lorsque le classeur est ouvert, sans qu'il soit nécessaire de l'exécuter manuellement.
- **Définition de la Feuille de Calcul**
  La fonction utilise l'objet `ThisWorkbook` pour accéder au classeur actif et à la feuille de calcul particulière. La feuille de calcul `Inventory` est ciblée par le code suivant :

  ```
  With ThisWorkbook.Sheets("Inventory")
  ```

  Définit un contexte de travail pour interagir avec la feuille de calcul nommée "Inventory" dans le classeur actuel. Cela permet d'appliquer les modifications seulement à cette feuille de calcul spécifique.
- **Initialisation des Cellules**
  Les valeurs suivantes sont définies pour certaines cellules de la feuille `Inventory` :

— Cellule B8 : La valeur est définie comme une chaîne vide `""`. Cette initialisation peut être utilisée pour réinitialiser ou nettoyer la cellule à chaque ouverture du classeur.
— Cellule E8 : La valeur est également définie comme une chaîne vide `""`, de la même manière que pour la cellule B8.
— Cellule E7 : La valeur est définie sur la date actuelle, obtenue par la fonction `Date`. Cela insère automatiquement la date du jour dans la cellule E7.
— Cellule B7 : La valeur est également définie sur la date actuelle avec la même fonction `Date`, insérant ainsi la date du jour dans la cellule B7.

Ces initialisations sont réalisées dans le bloc `With ... End With` pour simplifier le code et améliorer sa lisibilité.

— **Terminaison de l'Événement**
Le bloc `With ... End With` est fermé pour terminer la section de code qui modifie les cellules de la feuille `Inventory`.

4.3.2 Worksheet_Change

La fonction `Worksheet_Change` est un événement VBA qui se déclenche instantanément chaque fois qu'une modification est apportée à la feuille de calcul. Cette fonction est conçue pour gérer les modifications particulières apportées à certaines cellules et appliquer des règles de validation des données en conséquence. Voici une explication détaillée du fonctionnement et des objectifs de cette fonction.

Objectif de la Fonction

L'objectif principal de la fonction `Worksheet_Change` est de gérer les changements dans les cellules clés de la feuille de calcul et d'appliquer des validations appropriées. Plus précisément, cette fonction :

— Valide les dates lorsqu'elles sont modifiées dans certaines cellules.
— Met à jour la liste de validation des données dans une cellule en fonction de la valeur sélectionnée dans une autre cellule.

Description des Étapes du Code

— **Déclaration des Variables**
La fonction commence par déclarer deux variables principales :
  — `selected_lpg` : une variable de type `String` qui stocke la valeur sélectionnée dans la cellule `B8`.
  — `magasins` : une variable de type `Variant` utilisée pour stocker les options de validation des données à appliquer dans la cellule `E8`.
— **Gestion des Modifications dans les Cellules `B7` et `E7`**
  — Le code utilise la méthode `Intersect` pour déterminer si les cellules modifiées se trouvent dans la plage `B7` ou `E7`.
  — Si les modifications concernent ces cellules, la fonction `ValidateDatesInput` est appelée pour vérifier la validité des dates saisies.
  — Si les dates sont invalides, les modifications sont annulées à l'aide de `Application.Undo`.
— **Gestion des Modifications dans la Cellule `B8`**
  — Lorsque la cellule `B8` est modifiée, la fonction récupère la nouvelle valeur sélectionnée et la stocke dans `selected_lpg`.
  — La cellule `E8` est vidée pour préparer l'application d'une nouvelle liste de validation.

— Selon la valeur sélectionnée dans `B8`, une liste spécifique d'options est assignée à `magasins` à l'aide d'une structure `Select Case`.
— La validation des données pour la cellule `E8` est configurée en fonction des options déterminées. Toute validation existante est d'abord supprimée avec `.Delete`, puis une nouvelle validation de liste est ajoutée.
— La liste déroulante dans `E8` est ainsi mise à jour pour refléter les options appropriées.

4.3.3 Worksheet Activate

La fonction `Worksheet_Activate` est un événement VBA qui se déclenche automatiquement chaque fois que la feuille de calcul devient active, c'est-à-dire lorsque l'utilisateur sélectionne ou passe à cette feuille dans le classeur Excel. Cette fonction est utilisée pour programmer des éléments de la feuille de calcul en fonction de l'activation de la feuille. Voici une explication détaillée du fonctionnement et des objectifs de cette fonction.

Objectif de la Fonction

L'objectif de la fonction `Worksheet_Activate` est de préparer la feuille de calcul en définissant une liste déroulante pour une cellule spécialisée chaque fois que la feuille devient active. Plus précisément, cette fonction montre une liste déroulante dans la cellule `B8`, permettant aux utilisateurs de choisir parmi une liste prédéfinie d'options.

Description des Étapes du Code

— **Déclaration de la Variable**
La fonction commence par déclarer une variable :
— `LPGS` : une variable de type `Variant` qui est utilisée pour stocker un tableau des options disponibles pour la liste déroulante.
— **Définition des Options de Liste**
Le tableau `LPGS` est initialisé avec les valeurs suivantes :

```
LPGS = Array("Option1", "Option2", "Option3")
```

Crée un tableau `LPGS` contenant les noms des différentes localités ou sites associés à "LPGS". Ces valeurs représentent les différentes options disponibles pour la liste déroulante, correspondant à différents sites ou emplacements.
— **Configuration de la Liste Déroulante**
La fonction configure ensuite la liste déroulante pour la cellule `B8` de la feuille de calcul active :
— **Suppression de la Validation Existante** : La méthode `.Delete` est utilisée pour supprimer toute validation des données existante dans la cellule `B8`. Cela garantit que la nouvelle liste déroulante remplace toute configuration précédente.
— **Ajout de la Nouvelle Validation** : La méthode `.Add` est utilisée pour créer une nouvelle validation des données. Le type de validation est défini comme `xlValidateList`, ce qui montre que la cellule contiendra une liste déroulante.
— **Configuration de la Liste Déroulante** : La liste des options est fournie via `Formula1:=Join(LPG ",")`. La fonction `Join` convertit le tableau `LPGS` en une chaîne de texte séparée par des virgules, ce qui est requis pour la validation de liste dans Excel.

— **Paramètres Additionnels** : Les propriétés `.IgnoreBlank`, `.InCellDropdown`, `.ShowInput`, et `.ShowError` sont configurées pour permettre une liste déroulante dans la cellule, ignorer les entrées vides, afficher un message d'entrée et afficher un message d'erreur si nécessaire.

4.4 mouvement log

4.4.1 ExportSheetAsPDF

La fonction `ExportSheetAsPDF` est une macro VBA conçue pour exporter une feuille de calcul Excel en tant que fichier PDF. Cette fonctionnalité est particulièrement utile pour partager ou archiver des données dans un format universel et lisible, tout en préservant la mise en page et la présentation de la feuille de calcul. Voici une explication détaillée du fonctionnement de cette fonction, étape par étape.

Objectif de la Fonction

L'objectif principal de la fonction `ExportSheetAsPDF` est de sauvegarder une feuille de calcul spécifique d'un classeur Excel en tant que fichier PDF sur le bureau de l'utilisateur. La fonction configure la mise en page de la feuille, définit les zones d'impression, et exporte le document au format PDF avec des paramètres de qualité spécifiés.

Description des Étapes du Code

— **Déclaration des Variables**
La fonction commence par déclarer deux variables :
— `ws` : Une variable de type `Worksheet` pour représenter la feuille de calcul à exporter.
— `pdfPath` : Une variable de type `String` pour stocker le chemin du fichier PDF de destination.

— **Définition de la Feuille de Calcul à Exporter**
La variable `ws` est définie pour faire référence à la feuille de calcul nommée `"Mouvement Log"` :

```
Set ws = ThisWorkbook.Sheets("Mouvement Log")
```

Définit l'objet `ws` pour faire référence à la feuille de calcul nommée "Mouvement Log" dans le classeur actuel. Il est important de remplacer `"Mouvement Log"` par le nom réel de la feuille de calcul souhaitée si nécessaire.

— **Définition du Chemin de Sauvegarde du PDF**
Ce code construit le chemin du fichier PDF en combinant le chemin de dossier `C:\Users\hp\Desktop` avec le nom de la feuille de calcul (`ws.Name`) et l'extension .pdf. Cela permet de définir le chemin complet où le fichier PDF sera enregistré :

```
\texttt{pdfPath = "C:\Users\hp\Desktop\" & ws.Name & ".pdf"}
```

— **Configuration de la Zone d'Impression**
La zone d'impression est définie pour couvrir l'ensemble de la plage utilisée dans la feuille de calcul :

```
\texttt{ws.PageSetup.PrintArea = ws.UsedRange.Address}
```

Définit la zone d'impression de la feuille de calcul `ws` pour inclure toute la plage utilisée dans cette feuille. Cela garantit que toute la plage de données visible sera incluse dans le PDF exporté.

— **Configuration de la Mise en Page**
La mise en page de la feuille est configurée pour ajuster la feuille à une seule page :
  — **Désactivation du Zoom** : `.Zoom = False` désactive le zoom automatique.
  — **Ajustement à une Page Large** : `.FitToPagesWide = 1` ajuste la feuille pour qu'elle tienne sur une seule page en largeur.
  — **Ajustement à une Page en Hauteur** : `.FitToPagesTall = 1` ajuste également la feuille pour qu'elle tienne sur une seule page en hauteur.

— **Exportation de la Feuille en tant que PDF**
La méthode `ExportAsFixedFormat` est utilisée pour exporter la feuille de calcul en format PDF :

```
\texttt{ws.ExportAsFixedFormat Type:=xlTypePDF, Filename:=pdfPath, Quality:
    =xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False,
     OpenAfterPublish:=False}
```

  — **Type** : Spécifie que le fichier doit être exporté au format PDF (`xlTypePDF`).
  — **Filename** : Indique le chemin et le nom du fichier PDF à créer.
  — **Quality** : Définit la qualité du PDF comme étant standard (`xlQualityStandard`).
  — **IncludeDocProperties** : Inclut les propriétés du document dans le PDF (`True`).
  — **IgnorePrintAreas** : Ne pas ignorer les zones d'impression définies (`False`).
  — **OpenAfterPublish** : Ne pas ouvrir automatiquement le fichier après l'exportation (`False`).

— **Notification à l'Utilisateur**
Après l'exportation, une boîte de message informe l'utilisateur que la feuille de calcul a été exportée avec succès :

```
\texttt{MsgBox "La feuille de calcul a été exportée en PDF avec succès à l'
    emplacement : " & pdfPath}
```

Affiche un message indiquant que la feuille de calcul a été exportée en PDF, en incluant le chemin du fichier PDF.

4.5 daily inventory

4.5.1 Workbook Open

La fonction `ReinitialiserValeurs` est conçue pour réinitialiser les valeurs dans certaines cellules d'une feuille de calcul Excel, en mettant à jour les données à des valeurs par défaut ou vides. Cela est souvent nécessaire pour garantir que les données entrées dans la feuille sont cohérentes et conformes aux exigences du processus de gestion des données ou de reporting. Cette fonction est typiquement appelée lorsque l'utilisateur souhaite réinitialiser ou nettoyer les champs de saisie dans une feuille de calcul avant d'entrer de nouvelles informations.

Description de la Fonction

La fonction `ReinitialiserValeurs` est utilisée pour définir des valeurs par défaut dans des cellules spécifiques d'une feuille de calcul. Ces valeurs par défaut sont souvent choisies pour garantir que

les cellules sont correctement préparées pour la prochaine utilisation ou pour rétablir un état initial avant de commencer une nouvelle saisie de données.

Exemple de Code

Supposons que la fonction `ReinitialiserValeurs` est définie comme suit :

Explication des Étapes du Code

- **Déclaration de la Fonction**
  La fonction commence par la déclaration de `Sub ReinitialiserValeurs()`, ce qui indique le début de la procédure VBA. Cette fonction ne prend aucun argument et n'a pas de valeur de retour.
- **Définition de la Feuille de Calcul**
  La fonction utilise `ThisWorkbook.Sheets("Inventory")` pour se référer à la feuille de calcul nommée `"Inventory"` dans le classeur actuel. Cela signifie que toutes les opérations de réinitialisation seront appliquées à cette feuille de calcul spécifique.
- **Réinitialisation des Valeurs des Cellules**
  La fonction utilise la méthode `Range` pour accéder à des cellules spécifiques et définir leurs valeurs :
  - `.Range("B8").value = " "` : Cette ligne réinitialise la cellule `B8` à une chaîne vide. Cela efface le contenu existant dans cette cellule.
  - `.Range("E8").value = " "` : De même, cette ligne réinitialise la cellule `E8` à une chaîne vide.
  - `.Range("E7").value = Date` : Cette ligne définit la cellule `E7` à la date actuelle (`Date`). Cela met à jour la cellule avec la date du jour, ce qui est utile pour enregistrer la date actuelle lorsque la feuille est réinitialisée.
  - `.Range("B7").value = Date` : Cette ligne définit également la cellule `B7` à la date actuelle, pour les mêmes raisons que ci-dessus.
- **Utilisation et Importance**
  La fonction `ReinitialiserValeurs` est particulièrement utile dans les scénarios suivants :
  - **Préparation des Formulaires** : Lors de la préparation d'un formulaire pour une nouvelle entrée de données, il est crucial de réinitialiser les champs pour éviter les erreurs ou les incohérences.
  - **Nettoyage des Données** : Avant d'enregistrer ou de traiter de nouvelles informations, il peut être nécessaire de réinitialiser les cellules pour s'assurer que les anciennes valeurs n'affectent pas les nouvelles saisies.
  - **Automatisation des Processus** : La fonction peut être utilisée dans des scripts plus complexes pour automatiser la réinitialisation des valeurs lorsqu'un utilisateur démarre une nouvelle session de saisie ou avant l'impression d'un rapport.

4.5.2 Worksheet Activate

La fonction `Worksheet_Activate` est une procédure événementielle dans VBA (Visual Basic for Applications) qui se déclenche automatiquement lorsque la feuille de calcul sur laquelle elle est définie devient active. Ce type de fonction est utilisé pour exécuter des actions spécifiques dès qu'un utilisateur accède à la feuille de calcul, permettant ainsi de préparer ou d'initialiser des éléments selon les besoins du contexte de la feuille.

Description de la Fonction

La fonction `Worksheet_Activate` appelle une autre procédure, `CreateDropDownListFromColumnM2`, lorsque la feuille de calcul devient active. Cette approche est utilisée pour automatiser certaines tâches ou configurations spécifiques dès que l'utilisateur accède à la feuille de calcul.

Exemple de Code

Voici le code de la fonction `Worksheet_Activate` :
Exécute la procédure CreateDropDownListFromColumnM2 chaque fois que la feuille de calcul est activée.

Explication des Étapes du Code

- **Déclaration de la Fonction**
  La fonction commence par `Private Sub Worksheet_Activate()`, ce qui indique que c'est une procédure événementielle qui s'exécute automatiquement lorsqu'un événement spécifique se produit, dans ce cas, l'activation de la feuille de calcul.
- **Appel de la Procédure `CreateDropDownListFromColumnM2`**
  La ligne `Call CreateDropDownListFromColumnM2` appelle une autre procédure VBA nommée `CreateDropDownListFromColumnM2`. Cela signifie que chaque fois que la feuille de calcul devient active, Excel exécutera cette procédure spécifique.
- **Fonction de la Procédure Appelée**
  La procédure `CreateDropDownListFromColumnM2` est probablement conçue pour créer une liste déroulante dans une cellule de la feuille de calcul, en utilisant les valeurs d'une colonne spécifique (dans ce cas, la colonne M2). Voici un exemple hypothétique de ce que pourrait faire cette procédure :
- **Utilisation et Importance**
  La fonction `Worksheet_Activate` est utilisée pour garantir que certaines actions sont prises en compte chaque fois que la feuille de calcul est activée. Cette approche est particulièrement utile pour :
  - **Automatisation des Tâches** : En appelant `CreateDropDownListFromColumnM2`, on s'assure que les listes déroulantes sont mises à jour ou créées automatiquement à chaque fois que la feuille est activée, facilitant ainsi l'interaction utilisateur.
  - **Préparation Dynamique** : Cette fonction permet d'adapter la feuille de calcul en fonction des données présentes dans une colonne, rendant la feuille plus interactive et réactive aux modifications de données.
  - **Gestion des Données** : Elle simplifie la gestion des données en automatisant la création des listes déroulantes, ce qui peut améliorer la précision des entrées de données et la cohérence des informations saisies.

4.5.3 Worksheet Change

La fonction `Worksheet_Change` est une procédure événementielle dans VBA (Visual Basic for Applications) qui s'exécute automatiquement chaque fois qu'une modification est apportée à une cellule dans une feuille de calcul Excel. Cette procédure est particulièrement utile pour gérer les interactions dynamiques et les validations en fonction des modifications apportées par l'utilisateur.

Description de la Fonction

La fonction `Worksheet_Change` est utilisée pour surveiller et réagir aux changements dans des cellules spécifiques de la feuille de calcul. Elle effectue diverses vérifications et mises à jour en réponse à ces modifications, garantissant que les données restent cohérentes et correctes.

Explication des Étapes du Code

- **Déclaration de la Fonction**
  - La fonction commence par `Private Sub Worksheet_Change(ByVal Target As Range)`. Elle s'active automatiquement lorsque des modifications sont apportées à la feuille de calcul.
- **Vérification des Modifications dans C8** :
  - Vérifie si la cellule modifiée (Target) intersecte la cellule C8 de la feuille de calcul.
  - Si la cellule modifiée est C8, la fonction appelle `ValidateDateInput` pour vérifier la validité de la date.
  - Si la date n'est pas valide, un message d'alerte est affiché et la fonction se termine.
  - Si la date est valide, la fonction `CheckInventoryType` est appelée pour mettre à jour la cellule E8 en fonction de la valeur de C8.
- **Définition de la Plage à Surveiller dans la Colonne J** :
  - Définit `affectedRange` comme la plage allant de la cellule J15 jusqu'à la dernière cellule non vide de la colonne J dans la feuille de calcul.
  - Cette étape permet de spécifier les cellules dans la colonne J qui seront surveillées pour des modifications.
- **Vérification des Modifications dans la Plage Affectée** :
  - Vérifie si la cellule modifiée (Target) se trouve dans la plage définie par `affectedRange`.
  - Si la cellule modifiée se trouve dans cette plage, la fonction parcourt chaque cellule modifiée.
- **Effacement des Valeurs et Appel de la Sous-routine** :
  - Pour chaque cellule modifiée dans la plage, les cellules correspondantes dans les colonnes K et L pour la même ligne sont effacées.
  - La sous-routine `CheckQAndRetrieveEmailAndPost2` est ensuite appelée pour mettre à jour les colonnes K et L en fonction des changements dans la colonne J.

Utilisation et Importance

La fonction `Worksheet_Change` permet de gérer efficacement les changements apportés par les utilisateurs dans une feuille de calcul. Ses principales applications comprennent :

- **Validation Dynamique des Données** : En appelant `ValidateDateInput`, la fonction assure que les dates saisies sont valides et ne sont pas dans le futur, améliorant ainsi la qualité des données.
- **Mise à Jour Automatique** : En réagissant aux modifications dans C8 et dans la colonne J, la fonction met à jour automatiquement d'autres cellules en conséquence, ce qui simplifie la gestion des données complexes.
- **Interaction Utilisateur** : La fonction fournit des retours immédiats à l'utilisateur (comme les messages d'erreur), ce qui améliore l'expérience utilisateur en assurant que les données saisies sont correctes.

4.6 thisworkbook

4.6.1 Workbook Open

La fonction `Workbook_Open` est une procédure événementielle dans VBA (Visual Basic for Applications) qui s'exécute automatiquement chaque fois qu'un classeur Excel est ouvert. Cette fonction est utilisée pour gérer l'affichage des feuilles de calcul et pour contrôler l'accès au contenu du classeur en fonction de l'authentification utilisateur. Voici une explication détaillée de cette fonction.

Description de la Fonction

La fonction `Workbook_Open` est conçue pour garantir que seules certaines feuilles de calcul sont visibles au lancement du classeur et pour afficher un formulaire de connexion afin de protéger l'accès au contenu sensible. La fonction suit les étapes suivantes :

- **Masquage Initial des Feuilles de Calcul**
  - Utilisation d'une boucle `For Each` pour parcourir toutes les feuilles de calcul du classeur (`ThisWorkbook.Worksheets`).
  - Chaque feuille est masquée en utilisant la propriété `xlSheetVeryHidden`, ce qui les rend invisibles dans l'interface utilisateur, même avec l'option *Afficher*.
  - Utilisation de `On Error Resume Next` pour ignorer les erreurs potentielles qui pourraient se produire si une feuille est déjà masquée ou si d'autres erreurs surviennent.
  - Réactivation du traitement normal des erreurs avec `On Error GoTo 0` après avoir terminé la boucle.
- **Affichage de la Feuille "Menu"**
  - La feuille de calcul nommée `"Menu"` est rendue visible en utilisant la propriété `xlSheetVisible`.
  - La feuille `"Menu"` est sélectionnée pour s'assurer qu'elle est la feuille active lorsque le classeur est ouvert.
- **Affichage du Formulaire de Connexion**
  - Affichage du formulaire de connexion (`LoginForm`) en mode modal. Cela empêche l'utilisateur d'interagir avec d'autres parties du classeur tant que le formulaire n'est pas fermé.
- **Vérification de l'Authentification**
  - Après la fermeture du formulaire de connexion, vérification de la valeur de la propriété `Tag` du formulaire.
  - Si la valeur est `"Authenticated"`, cela indique que l'utilisateur a réussi à se connecter, et la fonction continue avec les opérations suivantes.
  - Si la valeur n'est pas `"Authenticated"`, le classeur est fermé sans enregistrer les modifications.
- **Gestion des Feuilles en Fonction de l'Authentification**
  - **Si l'authentification est réussie** :
    - La feuille `"Menu"` est maintenue visible.
    - La feuille `"Data"` reste très cachée (`xlSheetVeryHidden`), donc elle n'est pas accessible via l'interface utilisateur.
    - Les feuilles `"Mouvement Log"` et `"Inventory Log"` sont masquées (`xlSheetHidden`), rendant leur affichage possible uniquement via des opérations spécifiques de l'utilisateur.
  - **Si l'authentification échoue** :
    - Le classeur est fermé sans enregistrer les modifications éventuelles apportées par l'utilisateur :

```
ThisWorkbook.Close False
```

Objectifs et Importance de la Fonction

La fonction `Workbook_Open` joue un rôle crucial dans la gestion de l'accès au contenu du classeur en fonction des autorisations de l'utilisateur. Voici les objectifs principaux :

- **Sécurité des Données** : En masquant toutes les feuilles sauf la feuille `"Menu"` et en affichant un formulaire de connexion, la fonction protège les données sensibles du classeur. Les utilisateurs non authentifiés ne peuvent pas accéder aux informations importantes.
- **Contrôle d'Accès** : La vérification de l'authentification assure que seuls les utilisateurs autorisés peuvent accéder aux feuilles cachées et très cachées du classeur. Si l'authentification échoue, le classeur se ferme automatiquement, empêchant tout accès non autorisé.
- **Gestion de l'Interface Utilisateur** : En sélectionnant la feuille `"Menu"` et en affichant le formulaire de connexion, la fonction améliore l'expérience utilisateur en guidant les utilisateurs vers la page de connexion avant d'accéder à d'autres parties du classeur.
- **Préparation à l'Utilisation** : La fonction garantit que le classeur est dans un état prêt à l'emploi avec les bonnes feuilles visibles et cachées en fonction de l'état de l'authentification.

4.7 loginform

4.7.1 ConnectButton Click

La fonction `ConnectButton_Click` est une procédure événementielle en VBA (Visual Basic for Applications) qui se déclenche lorsqu'un utilisateur clique sur le bouton de connexion d'un formulaire. Cette fonction est responsable de la validation des informations de connexion de l'utilisateur en les comparant aux données stockées dans une feuille de calcul et de gérer l'affichage du contenu du classeur en fonction de l'authentification réussie. Voici une explication détaillée de cette fonction.

Description de la Fonction

- **Récupération des Informations de Connexion**
  - Attribue les valeurs saisies dans les champs de texte `userbox` et `passbox` aux variables `mail` et `password`, respectivement.
  - Les informations de connexion (adresse e-mail et mot de passe) sont récupérées depuis les contrôles `userbox` et `passbox` du formulaire de connexion.
- **Initialisation des Variables**
  - `isValid` est utilisé pour indiquer si les informations de connexion sont valides.
  - `ws` représente la feuille de calcul où les données des utilisateurs sont stockées.
  - `lastRow` et `i` sont utilisés pour déterminer la dernière ligne avec des données et pour parcourir les lignes respectivement.
- **Définition de la Feuille de Calcul et Recherche de la Dernière Ligne**
  - Définit l'objet `ws` pour faire référence à la feuille de calcul nommée `"Data"` dans le classeur actuel.
  - Détermine la dernière ligne non vide dans la colonne M pour savoir jusqu'où parcourir les données.
- **Validation des Informations de Connexion**

- La fonction parcourt les lignes de la feuille `"Data"` pour vérifier si les informations de connexion fournies correspondent à celles stockées dans les colonnes P (e-mail) et R (mot de passe).
- Si une correspondance est trouvée, les informations de l'utilisateur (nom, ID, affectation, e-mail, poste, mot de passe) sont stockées dans des variables et `isValid` est défini sur `True`.
- La boucle est ensuite interrompue.

— **Traitement en Fonction de la Validité des Informations de Connexion**
- **Si les informations de connexion sont valides** :
  - Un message de bienvenue est affiché avec le nom de l'utilisateur.
  - La propriété `Tag` du formulaire est définie sur `"Authenticated"` pour indiquer une connexion réussie.
  - Les informations de l'utilisateur (nom, ID, poste, affectation) sont mises à jour dans divers objets de forme sur les feuilles `"Menu"`, `"Movement Sheet"`, et `"Daily inventory"`.
  - La feuille `"Menu"` est activée et toutes les feuilles de calcul du classeur sont rendues visibles.
  - Le formulaire de connexion est masqué pour que l'utilisateur puisse accéder aux autres feuilles du classeur.
- **Si les informations de connexion ne sont pas valides** :
  - Un message d'erreur est affiché pour indiquer que le nom d'utilisateur ou le mot de passe est incorrect.

Objectifs et Importance de la Fonction

La fonction `ConnectButton_Click` joue un rôle central dans la gestion de l'accès au classeur Excel en fonction des informations de connexion fournies. Voici les principaux objectifs :

- **Validation de l'Authentification** : La fonction vérifie que les informations de connexion correspondent aux données stockées dans la feuille `"Data"`, garantissant que seuls les utilisateurs autorisés peuvent accéder au contenu du classeur.
- **Mise à Jour Dynamique des Informations Utilisateur** : Les informations utilisateur sont mises à jour dans divers objets de forme sur plusieurs feuilles de calcul, assurant que les détails pertinents sont affichés correctement après la connexion.
- **Contrôle d'Accès** : En affichant ou en masquant les feuilles de calcul en fonction de l'authentification réussie, la fonction assure que les utilisateurs ne voient que les informations qui leur sont autorisées.
- **Amélioration de l'Expérience Utilisateur** : En fournissant des messages d'accueil ou des erreurs appropriés, la fonction améliore l'interaction utilisateur et facilite l'accès aux fonctionnalités du classeur.

4.7.2 Image1 BeforeDragOver

La fonction `Image1_BeforeDragOver` est une procédure événementielle en VBA (Visual Basic for Applications) qui se déclenche avant qu'une opération de glisser-déposer ne se produise sur un objet spécifique, en l'occurrence une image dans un formulaire utilisateur (`UserForm`). Voici une explication détaillée de cette fonction.

Description de la Fonction

- **Définition des Paramètres**
  - La fonction ne nécessite pas de paramètres spécifiques pour cet événement, mais elle interagit avec l'objet `Image1` dans le formulaire.
- **Action Réalisée**
  - Lorsque l'événement `BeforeDragOver` est déclenché, la fonction exécute la commande `ThisWorkbook.Close`, ce qui ferme le classeur actuel sans sauvegarder les modifications.
- **Objectifs et Importance de la Fonction**
  - **Prévention des Modifications Accidentelles** : En fermant le classeur lors d'une tentative de glisser-déposer, la fonction empêche les utilisateurs de modifier le contenu du classeur accidentellement ou de faire glisser des éléments sur le formulaire, ce qui pourrait altérer le fonctionnement prévu du classeur.
  - **Gestion des Événements d'Interaction** : L'événement `BeforeDragOver` est utilisé ici pour déclencher une action critique avant que le glisser-déposer ne commence. Cela permet de contrôler et d'intercepter les interactions des utilisateurs avec l'interface graphique du formulaire.
  - **Protection des Données** : En fermant le classeur immédiatement, la fonction peut également servir de mécanisme de protection pour éviter des manipulations non autorisées ou non désirées des données du classeur.
  - **Comportement Non Conventionnel** : Cette fonction introduit un comportement non conventionnel en fermant le classeur lors d'une tentative de glisser-déposer. Cela peut surprendre les utilisateurs et doit être bien documenté pour éviter toute confusion. Il est important de s'assurer que cette action est bien justifiée dans le contexte de l'application et qu'elle ne compromet pas l'expérience utilisateur.

4.8 module 1

4.8.1 GoToMouvement

La fonction `GoToMouvement` est une procédure en VBA (Visual Basic for Applications) utilisée dans les applications Excel pour manipuler les feuilles de calcul. Cette fonction est conçue pour rendre visible une feuille de calcul spécifique et la sélectionner. Voici une explication détaillée de chaque partie de cette fonction.

- **Déclaration de la Subroutine**
  - `Sub GoToMouvement()` : La déclaration `Sub` indique le début d'une subroutine (ou procédure) en VBA.
  - Le nom de la subroutine est `GoToMouvement`, ce qui suggère que sa fonction est de naviguer vers ou de se déplacer à la feuille de calcul nommée `"Movement Sheet"`.
- **Rendre la Feuille Visible**
  - La fonction rend visible la feuille de calcul nommée `"Movement Sheet"`.
  - Par défaut, les feuilles de calcul peuvent être cachées ou visibles.
  - La propriété `Visible` est utilisée ici pour s'assurer que la feuille est affichée à l'utilisateur avant toute autre action.
- **Sélectionner la Feuille**
  - La fonction sélectionne la feuille de calcul `"Movement Sheet"`.
  - En sélectionnant la feuille, elle devient la feuille active dans l'interface utilisateur d'Excel.

— Cela permet à l'utilisateur de la voir immédiatement à l'écran après que la fonction ait été exécutée.

Objectifs et Importance de la Fonction

— **Navigation Utilisateur** : Cette fonction facilite la navigation dans le classeur Excel en rendant visible et en sélectionnant automatiquement une feuille spécifique, améliorant ainsi l'expérience utilisateur.
— **Gestion Dynamique des Feuilles** : Dans des scénarios où certaines feuilles peuvent être cachées pour des raisons de sécurité ou de gestion de l'interface utilisateur, cette fonction assure que la feuille `"Movement Sheet"` est accessible lorsque nécessaire, sans nécessiter d'interaction supplémentaire de l'utilisateur.
— **Simplification des Tâches** : En automatisant la visibilité et la sélection de la feuille, la fonction simplifie les tâches pour l'utilisateur, particulièrement dans des macros ou des scripts nécessitant l'interaction avec plusieurs feuilles de calcul.
— **Utilisation dans des Macros** : La fonction `GoToMouvement` est souvent utilisée dans des macros pour automatiser des flux de travail nécessitant un accès spécifique à certaines feuilles, comme diriger l'utilisateur vers la feuille de calcul où des données doivent être examinées ou traitées.

4.8.2 GoToInventory

La fonction `GoToInventory` est une procédure écrite en VBA (Visual Basic for Applications) pour Excel, conçue pour manipuler l'affichage et la sélection d'une feuille de calcul spécifique dans un classeur. Voici une analyse détaillée de chaque composant de cette fonction.

— **Déclaration de la Subroutine**
  — `Sub GoToInventory()` : Cette ligne indique le début de la procédure en VBA.
  — Le nom de la procédure, `GoToInventory`, suggère que la fonction est destinée à naviguer vers la feuille de calcul nommée `"Inventory"`.
— **Rendre la Feuille Visible**
  — La fonction rend visible la feuille de calcul intitulée `"Inventory"`.
  — Dans Excel, les feuilles de calcul peuvent être cachées pour diverses raisons.
  — Cette ligne de code garantit que la feuille `"Inventory"` est affichée dans l'interface utilisateur avant toute autre action.
— **Sélectionner la Feuille**
  — La fonction sélectionne la feuille de calcul `"Inventory"`, la rendant ainsi active.
  — Lorsque la feuille est sélectionnée, elle devient visible à l'écran, permettant à l'utilisateur de la voir immédiatement après l'exécution de la fonction.

Objectifs et Importance de la Fonction

— **Faciliter la Navigation** : La fonction permet une navigation facile et rapide vers la feuille de calcul `"Inventory"`, utile surtout si le classeur contient de nombreuses feuilles ou si certaines sont cachées.
— **Gestion Dynamique des Feuilles** : Dans des environnements où certaines feuilles sont masquées pour des raisons de sécurité ou de gestion, cette fonction garantit que la feuille `"Inventory"` est visible et sélectionnée lorsque nécessaire.

— **Améliorer l'Expérience Utilisateur** : En automatisant la visibilité et la sélection de la feuille `"Inventory"`, la fonction simplifie l'expérience utilisateur, évitant la nécessité de rechercher ou de démasquer manuellement la feuille.
— **Utilisation dans des Macros et Scripts** : La fonction est souvent intégrée dans des macros ou scripts VBA nécessitant une interaction avec des feuilles spécifiques, comme diriger l'utilisateur vers la feuille où les informations doivent être vérifiées ou analysées.

4.8.3 GoToDailyINV

La fonction `GoToDailyINV` est une procédure écrite en VBA (Visual Basic for Applications) pour Excel, conçue pour manipuler l'affichage et la sélection d'une feuille de calcul spécifique dans un classeur. Voici une analyse détaillée de chaque composant de cette fonction.

— **Déclaration de la Subroutine**
  — `Sub GoToDailyINV()` : Cette ligne indique le début de la procédure en VBA.
  — Le nom de la procédure, `GoToDailyINV`, suggère que la fonction est destinée à naviguer vers la feuille de calcul nommée `"Daily inventory"`.
— **Rendre la Feuille Visible**
  — Cette ligne de code rend visible la feuille de calcul intitulée `"Daily inventory"`.
  — Dans Excel, les feuilles de calcul peuvent être cachées pour diverses raisons.
  — Cette ligne garantit que la feuille `"Daily inventory"` est affichée dans l'interface utilisateur avant toute autre action.
— **Sélectionner la Feuille**
  — Cette ligne sélectionne la feuille de calcul `"Daily inventory"`, la rendant active.
  — Une fois sélectionnée, la feuille devient visible à l'écran, permettant à l'utilisateur de la voir immédiatement après l'exécution de la fonction.

Objectifs et Importance de la Fonction

— **Faciliter la Navigation** : La fonction permet une navigation facile et rapide vers la feuille de calcul `"Daily inventory"`. Cela est particulièrement utile dans des classeurs contenant de nombreuses feuilles ou lorsque certaines feuilles sont cachées pour diverses raisons.
— **Gestion Dynamique des Feuilles** : Dans des environnements où les feuilles sont masquées pour des raisons de sécurité ou d'organisation, cette fonction assure que la feuille `"Daily inventory"` est visible et sélectionnée lorsqu'elle est nécessaire, évitant ainsi à l'utilisateur de rechercher ou de démasquer manuellement la feuille.
— **Améliorer l'Efficacité de Travail** : En automatisant le processus de rendre visible et de sélectionner la feuille `"Daily inventory"`, la fonction simplifie l'expérience utilisateur et augmente l'efficacité. L'utilisateur peut accéder directement à la feuille appropriée sans avoir à naviguer manuellement à travers les feuilles.
— **Utilisation dans des Macros et Scripts** : La fonction est souvent utilisée dans des macros ou des scripts VBA nécessitant une interaction avec des feuilles spécifiques. Par exemple, après la mise à jour des données ou la génération de rapports, la fonction `GoToDailyINV` peut être appelée pour diriger l'utilisateur directement vers la feuille où les informations doivent être vérifiées ou mises à jour.

4.8.4 GoToMENU

La fonction `GoToMENU` est une procédure VBA (Visual Basic for Applications) utilisée dans Microsoft Excel pour faciliter la navigation entre les feuilles de calcul. Cette fonction est conçue pour rendre une feuille spécifique visible et la sélectionner, afin d'assurer que l'utilisateur puisse y accéder immédiatement. Voici une analyse détaillée de chaque composant de cette fonction.

- **Déclaration de la Subroutine**
  - `Sub GoToMENU()` : La déclaration `Sub` marque le début de la procédure VBA.
  - Le nom de la procédure, `GoToMENU`, indique que cette fonction est destinée à naviguer vers la feuille de calcul nommée `"MENU"`.
- **Rendre la Feuille Visible**
  - Cette ligne de code assure que la feuille de calcul intitulée `"MENU"` est visible.
  - Excel permet de masquer des feuilles pour des raisons d'organisation ou de sécurité.
  - Cette ligne rend la feuille `"MENU"` visible avant que toute autre action soit effectuée.
- **Sélectionner la Feuille**
  - Cette ligne de code sélectionne la feuille de calcul `"MENU"`, la rendant active.
  - Une fois la feuille sélectionnée, elle devient la feuille actuellement affichée à l'écran, permettant à l'utilisateur de la voir et d'interagir avec elle immédiatement.

Objectifs et Importance de la Fonction

- **Faciliter l'Accès au Menu Principal** : La fonction permet à l'utilisateur d'accéder rapidement à la feuille `"MENU"`, qui est généralement utilisée comme un tableau de bord ou une page de navigation principale dans un classeur. Cela est particulièrement utile dans les environnements de travail où la feuille `"MENU"` sert de point d'entrée pour diverses fonctionnalités ou sections du classeur.
- **Gestion des Feuilles Masquées** : Dans les classeurs où certaines feuilles peuvent être masquées pour des raisons de sécurité ou d'organisation, cette fonction assure que la feuille `"MENU"` est non seulement visible mais également sélectionnée, facilitant ainsi l'accès direct à cette feuille sans nécessiter une manipulation manuelle.
- **Optimiser l'Expérience Utilisateur** : En automatisant le processus de rendre visible et de sélectionner la feuille `"MENU"`, la fonction améliore l'efficacité et l'expérience utilisateur. Les utilisateurs peuvent naviguer directement vers la feuille principale sans avoir à chercher ou à manipuler les paramètres de visibilité des feuilles.
- **Utilisation dans des Macros et Scripts** : La fonction `GoToMENU` est souvent intégrée dans des macros ou des scripts VBA plus complexes. Par exemple, après l'exécution d'une série d'actions ou de mises à jour de données dans d'autres feuilles, cette fonction peut être appelée pour ramener l'utilisateur à la feuille `"MENU"` afin de récapituler ou de vérifier les résultats.

4.9 module 2

4.9.1 ValidateDatesInput

La fonction `ValidateDatesInput` est une procédure VBA (Visual Basic for Applications) utilisée pour vérifier la validité des dates saisies dans un classeur Excel. Son objectif est de garantir que les dates de début et de fin entrées par l'utilisateur répondent à des critères spécifiques et sont correctement formatées. Voici une explication détaillée de chaque composant de cette fonction.

Analyse des Composants de la Fonction

- **Déclaration de la Fonction** :
  - `Function ValidateDatesInput() As Boolean` : La déclaration de la fonction commence avec `Function`, indiquant que la fonction retourne une valeur de type Boolean (True ou False).
  - Le nom de la fonction, `ValidateDatesInput`, reflète son but : valider les dates saisies.
- **Définition de la Feuille de Calcul** :
  - `Dim ws As Worksheet`
  - `Set ws = ThisWorkbook.Sheets("Inventory")` : La fonction commence par définir une variable `ws` de type `Worksheet` et l'assigne à la feuille de calcul nommée `"Inventory"`.
  - Cette feuille est utilisée pour récupérer les valeurs des cellules contenant les dates à valider.
- **Déclaration des Variables pour les Dates** :
  - `Dim startDate As Date`
  - `Dim endDate As Date`
  - `Dim startDateValue As Variant`
  - `Dim endDateValue As Variant` :
  - Les variables `startDate` et `endDate` sont utilisées pour stocker les dates valides après conversion, tandis que `startDateValue` et `endDateValue` stockent les valeurs brutes récupérées des cellules.
- **Initialisation et Récupération des Valeurs** :
  - `ValidateDatesInput = False` : La fonction est initialisée avec une valeur par défaut de `False`, indiquant que la validation a échoué par défaut jusqu'à ce que toutes les vérifications passent.
  - `startDateValue = ws.Range("B7").Value`
  - `endDateValue = ws.Range("E7").Value` : Les valeurs des cellules B7 et E7 sont récupérées et stockées dans `startDateValue` et `endDateValue`, respectivement.
- **Vérification de la Validité des Dates** :
  - `If IsDate(startDateValue) Then` : La fonction utilise la fonction intégrée `IsDate` pour vérifier si `startDateValue` est une date valide.
  - Si ce n'est pas le cas, un message d'erreur est affiché et la fonction se termine.
  - `startDate = CDate(startDateValue)` : Si la valeur est une date valide, elle est convertie en type `Date` et assignée à `startDate`.
  - `If IsDate(endDateValue) Then` : Un processus similaire est effectué pour `endDateValue`.
- **Validation de la Relation Entre les Dates** :
  - `If endDate < startDate Then` : La fonction vérifie si `endDate` est antérieure à `startDate`.
  - Si c'est le cas, un message d'erreur est affiché, et la fonction se termine.
- **Retour du Résultat de Validation** :
  - `ValidateDatesInput = True` : Si toutes les vérifications passent sans erreur, la fonction retourne `True`, indiquant que les dates sont valides.

Objectifs et Importance de la Fonction

- **Assurer la Validité des Données** : La fonction garantit que les dates saisies dans les cellules B7 et E7 sont des dates valides et que la date de fin est postérieure ou égale à la date de début.
- **Prévenir les Erreurs de Saisie** : En validant les dates avant de procéder à d'autres actions ou calculs, la fonction aide à prévenir les erreurs qui pourraient survenir en raison de dates incorrectes ou mal saisies.

— **Améliorer l'Expérience Utilisateur** : Les messages d'erreur clairs aident les utilisateurs à comprendre les problèmes avec les données saisies et les guident pour corriger les erreurs, ce qui améliore l'expérience globale.
— **Intégration dans des Processus Automatisés** : Cette fonction peut être utilisée dans des macros ou des procédures automatisées pour s'assurer que les données sont correctement validées avant l'exécution de processus ultérieurs, garantissant ainsi l'intégrité des données dans le classeur.

4.10 module 3

4.10.1 AddNewRecord

La fonction `AddNewRecord` est une macro VBA utilisée pour ajouter un nouvel enregistrement à une feuille de calcul Excel contenant une table de mouvements, en utilisant un modèle de ligne pour garantir la cohérence des données. Voici une explication détaillée de chaque composant de cette fonction.

— **Déclaration et Initialisation des Variables**
  — `Dim ws As Worksheet` : Déclare une variable pour stocker l'objet Worksheet.
  — `Dim lastRow As Long`
  `Dim templateRow As Long`
  `Dim tableStartRow As Long`
  `Dim tableEndRow As Long` : Déclare des variables pour stocker les numéros de ligne et les limites de la table de mouvements.
— **Définition de la Feuille de Calcul**
  — `Set ws = ThisWorkbook.Sheets("Movement Sheet")` : La variable `ws` est assignée à la feuille de calcul nommée `"Movement Sheet"`. Cette feuille contient la table où les nouveaux enregistrements seront ajoutés.
— **Définition des Limites de la Table**
  — `tableStartRow = 42` : Définit la première ligne de la table "Details du mouvement".
  — `tableEndRow = 80` : Définit la dernière ligne de la table "Details du mouvement".
— **Identification de la Dernière Ligne Remplie**
  — `lastRow = ws.Cells(tableEndRow, "F").End(xlUp).Row` : Trouve la dernière ligne remplie dans la colonne F, en remontant depuis la fin de la table (ligne 80). Cette ligne est utilisée pour déterminer où insérer le nouvel enregistrement.
— **Vérification des Limites de la Table**
  — `If lastRow < tableStartRow Or lastRow >= tableEndRow Then` : Vérifie si la ligne trouvée est en dehors des limites définies pour la table. Si c'est le cas, un message d'erreur est affiché et la fonction se termine.
— **Détermination de la Ligne Modèle**
  — `If lastRow >= tableStartRow + 1 Then` : Vérifie si la dernière ligne est au-dessus de la première ligne de la table.
  — `templateRow = lastRow - 1` : Si la condition est remplie, détermine la ligne modèle à utiliser pour copier les formats et les formules. La ligne modèle est deux lignes au-dessus de la dernière ligne remplie.
— **Insertion d'une Nouvelle Ligne**
  — `ws.Rows(lastRow + 1).Insert Shift:=xlDown` : Insère une nouvelle ligne après la der-

nière ligne remplie dans la plage définie.

— **Copie du Modèle vers la Nouvelle Ligne**
  — `ws.Rows(templateRow).Copy Destination:=ws.Rows(lastRow + 1)` : Copie les formats et les formules de la ligne spécifiée par `templateRow` et les colle dans la ligne suivante après `lastRow`.
— **Effacement des Valeurs dans la Nouvelle Ligne**
  — `ws.Rows(lastRow + 1).SpecialCells(xlCellTypeConstants).ClearContents` : Efface les valeurs constantes dans la nouvelle ligne pour permettre une saisie propre des nouvelles données.
— **Mise à Jour de la Dernière Ligne**
  — `lastRow = ws.Cells(tableEndRow, "F").End(xlUp).Row` : Met à jour la variable `lastRow` pour refléter la nouvelle ligne ajoutée.
— **Numérotation de la Nouvelle Ligne**
  — `ws.Cells(lastRow + 1, 6).Value = ws.Cells(lastRow, 6).Value + 1` : Numérote la nouvelle ligne en incrémentant la valeur de la cellule de la colonne F de la dernière ligne remplie.

Objectifs et Importance de la Fonction

— **Ajouter des Enregistrements de Manière Structurée** : La fonction facilite l'ajout de nouveaux enregistrements à la table "Details du mouvement" tout en maintenant la structure et le formatage cohérents grâce à l'utilisation d'un modèle de ligne.
— **Maintenir la Cohérence des Données** : En copiant les formats et les formules du modèle, elle assure que les nouvelles lignes sont formatées de manière cohérente avec les lignes existantes.
— **Éviter les Erreurs de Saisie** : En effaçant les valeurs dans la nouvelle ligne, la fonction permet une saisie propre des nouvelles données, réduisant ainsi les risques d'erreurs.
— **Automatiser la Numérotation des Lignes** : La fonction automatise la numérotation des lignes pour garantir que chaque enregistrement est correctement identifié.

4.10.2 InsertDataIntoDestination

La fonction `InsertDataIntoDestination` est une macro VBA conçue pour transférer des données d'une feuille de calcul source vers une feuille de calcul destination, tout en assurant la validation et la gestion des erreurs. Voici une explication détaillée de chaque composant de cette fonction.

Analyse des Composants de la Fonction

— **Déclaration et Initialisation des Variables** :
  — Les variables nécessaires sont déclarées pour stocker les valeurs des cellules, les références de feuille de calcul, et les messages d'erreur. Ces variables sont cruciales pour gérer le transfert des données de manière efficace et en toute sécurité.
— **Gestion des Erreurs** :
  — Une gestion des erreurs est configurée pour diriger le programme vers un gestionnaire d'erreurs (souvent appelé `ErrorHandler`) en cas de problème. Cela permet de capturer et de traiter les erreurs potentielles pendant l'exécution de la macro, assurant ainsi que le programme se termine proprement même en cas de problèmes.
— **Identification des Dernières Lignes** :

— La fonction détermine la dernière ligne non vide dans la colonne Q de la feuille de calcul source (`wsSource`). Cette identification est essentielle pour savoir jusqu'où les données doivent être lues et transférées, évitant ainsi de traiter des cellules vides.

4.10.3 IsLong

La fonction `IsLong` est définie comme suit :

Analyse des Composants de la Fonction

— **Déclaration de la Fonction** :
  — `Function IsLong(value As Variant) As Boolean` :
    — La fonction `IsLong` prend un argument `value` de type `Variant`.
    — Le type `Variant` permet à la fonction de gérer différents types de données.
    — La fonction retourne un résultat de type `Boolean` (Vrai ou Faux) indiquant si la valeur peut être interprétée comme un entier long.
— **Gestion des Erreurs** :
  — `On Error Resume Next` :
    — Indique que, en cas d'erreur lors de l'exécution de la ligne suivante, VBA doit passer à la ligne suivante sans interrompre le programme.
    — Cela permet à la fonction de continuer à s'exécuter même si une erreur survient lors de la conversion de la valeur en entier long.
  — `On Error GoTo 0` :
    — Désactive la gestion des erreurs et restaure le comportement par défaut de VBA, où les erreurs non gérées entraîneraient l'arrêt du programme.
— **Conversion et Vérification de la Valeur** :
  — `IsLong = (CLng(value) = value And IsNumeric(value))` :
    — `CLng(value)` :
      — Tente de convertir la valeur en un entier long.
      — Si la conversion échoue (par exemple, si la valeur n'est pas un nombre valide), elle générera une erreur, mais grâce à `On Error Resume Next`, cette erreur est ignorée.
    — `CLng(value) = value` :
      — Vérifie si la valeur convertie en entier long est égale à la valeur d'origine.
      — Si la conversion est réussie et que la valeur est effectivement un entier long, cette condition sera vraie.
    — `IsNumeric(value)` :
      — Vérifie si la valeur est numérique.
      — Cette vérification est nécessaire pour s'assurer que la valeur peut être traitée comme un nombre, ce qui est un pré-requis pour la conversion en entier long.
— **Fonctionnement de la Fonction**
— La fonction `IsLong` essaie de convertir la valeur passée en argument en un entier long en utilisant la fonction `CLng`.
— Si cette conversion est réussie et que la valeur d'origine est un nombre, la fonction retourne `True`.
— Sinon, si la conversion échoue ou si la valeur n'est pas numérique, la fonction retourne `False`.

Cas d'Utilisation

— Cette fonction est utile dans des situations où il est nécessaire de vérifier que les valeurs entrées par l'utilisateur ou récupérées de sources externes sont des entiers longs valides avant de les utiliser dans des calculs ou de les stocker dans une base de données.
— Par exemple, elle pourrait être utilisée pour valider des entrées dans des formulaires ou pour vérifier des données importées dans des feuilles de calcul Excel.

4.10.4 SaveMovementData

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `Dim wsSource As Worksheet` : Déclare une variable pour représenter la feuille de calcul source.
  — `Dim wsDestination As Worksheet` : Déclare une variable pour représenter la feuille de calcul de destination.
  — `Dim startRow As Long` : Déclare une variable pour la première ligne de la table des détails du mouvement.
  — `Dim endRow As Long` : Déclare une variable pour la dernière ligne de la table des détails du mouvement.
  — `Dim success As Boolean` : Déclare une variable pour stocker le résultat de l'opération d'insertion des données.
— **Définition des Feuilles de Calcul Source et Destination**
  — `Set wsSource = ThisWorkbook.Sheets("Movement Sheet")` : Assigne la feuille de calcul "Movement Sheet" à la variable `wsSource`.
  — `Set wsDestination = ThisWorkbook.Sheets("Mouvement Log")` : Assigne la feuille de calcul "Mouvement Log" à la variable `wsDestination`.
— **Validation des Dates**
  — `If Not ValidateDates() Then Exit Sub` : Appelle la fonction `ValidateDates()` pour vérifier la validité des dates. Si la fonction retourne `False`, la procédure s'arrête (`Exit Sub`), évitant ainsi de continuer avec des dates invalides.
— **Définition des Lignes de Début et de Fin**
  — `startRow = 42` : Définit la ligne de début de la table des détails du mouvement.
  — `endRow = 80` : Définit la ligne de fin de la table des détails du mouvement.
— **Insertion des Données dans la Feuille de Destination**
  — `InsertDataIntoDestination wsSource, wsDestination, startRow, endRow, success` : Appelle la procédure `InsertDataIntoDestination` pour transférer les données de `wsSource` à `wsDestination`, en utilisant les lignes définies et en passant la variable `success` pour obtenir le résultat de l'opération.
— **Traitement Après l'Insertion des Données**
  — `If success Then` : Vérifie si l'insertion des données a réussi (`success` est `True`).
  — `Call ExportRangeAsPDFAndSendEmail2` : Appelle la procédure `ExportRangeAsPDFAndSendEmail2` pour exporter la plage de données en PDF et envoyer par email (prérequis non inclus dans cette fonction).
  — `ResetSourceData wsSource, startRow, endRow` : Appelle la procédure `ResetSourceData` pour réinitialiser les données de la feuille source après l'insertion.
  — `Else` : Si l'insertion des données échoue (`success` est `False`), affiche un message d'erreur

à l'utilisateur.

Cas d'Utilisation

— La fonction `SaveMovementData` est utilisée dans des situations où il est nécessaire de transférer des données de la feuille "Movement Sheet" vers "Mouvement Log" après avoir vérifié la validité des dates.
— Cette fonction automatise le processus de vérification, transfert, et traitement ultérieur des données, en assurant une gestion correcte des erreurs et en facilitant l'intégration avec d'autres fonctionnalités telles que l'exportation et l'envoi par email.
— En résumé, la fonction `SaveMovementData` joue un rôle central dans la gestion des mouvements de données en garantissant la validité des informations, en assurant le transfert précis des données, et en exécutant des tâches complémentaires lorsque le transfert est réussi.

4.10.5 ResetSourceData

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `Dim i As Long` : Déclare une variable pour les indices de lignes utilisés dans la boucle.
  — `Dim lastRowNames As Long` : Déclare une variable pour stocker le numéro de la dernière ligne avec des données dans la colonne "Q".
— **Calcul de la Dernière Ligne avec Données**
  — `lastRowNames = wsSource.Cells(wsSource.Rows.Count, "Q").End(xlUp).Row + 2` : Détermine la dernière ligne non vide dans la colonne "Q" de la feuille source, et ajoute 2 pour définir la fin de la plage à réinitialiser. Le résultat est stocké dans `lastRowNames`.
  — `Debug.Print lastRowNames` : Imprime le numéro de la dernière ligne déterminée pour le débogage.
— **Réinitialisation des Champs Spécifiques**
  — `wsSource.Range("L6").value = ""` : Réinitialise le champ "FA".
  — `wsSource.Range("L8").value = ""` : Réinitialise le champ "Fiche de mouvement".
  — `wsSource.Range("L7").value = ""` : Réinitialise le champ "Type de Mouvement".
  — `wsSource.Range("L9").value = Date` : Met à jour le champ "Date actuelle" avec la date du jour.
  — `wsSource.Range("H19").value = ""` : Réinitialise le champ "Source".
  — `wsSource.Range("L19").value = ""` : Réinitialise le champ "Destination".
  — `wsSource.Range("H20").value = ""` : Réinitialise le champ "Source".
  — `wsSource.Range("L20").value = ""` : Réinitialise le champ "Destination".
  — `wsSource.Range("H21").value = ""` : Réinitialise le champ "Source".
  — `wsSource.Range("L21").value = ""` : Réinitialise le champ "Destination".
— **Réinitialisation des Données Répétitives**
  — `For i = startRow To endRow` : Boucle à travers les lignes définies par `startRow` et `endRow`.
  — `wsSource.Cells(i, 8).value = ""` : Réinitialise le champ "Article" dans chaque ligne.
  — `wsSource.Cells(i, 7).value = ""` : Réinitialise le champ "Emballage".
  — `wsSource.Cells(i, 9).value = ""` : Réinitialise le champ "Quantite".
  — `wsSource.Cells(i, 10).value = ""` : Réinitialise le champ "Marque".
  — `wsSource.Cells(i, 11).value = ""` : Réinitialise le champ "Commentaire".
— **Réinitialisation des Champs Supplémentaires**

  - `wsSource.Range("F27").value = ""` : Réinitialise le champ "Gestionnaire de stock".
  - `wsSource.Range("I27").value = ""` : Réinitialise le champ "Agent de gardiennage".
  - `wsSource.Range("L27").value = ""` : Réinitialise le champ "Transporteur".
- **Réinitialisation des Valeurs dans la Plage de "Q21" à la Dernière Ligne**
  - `For Each cell In wsSource.Range("Q21:Q" & lastRowNames)` : Boucle à travers chaque cellule dans la plage de "Q21" à `lastRowNames`.
  - `wsSource.Cells(cell.Row, "Q").value = ""` : Réinitialise les valeurs dans la colonne "Q".
  - `wsSource.Cells(cell.Row, "R").value = ""` : Réinitialise les valeurs dans la colonne "R".
  - `wsSource.Range(wsSource.Cells(cell.Row, "S"), wsSource.Cells(cell.Row, "T")).valu = ""` : Réinitialise les valeurs dans les colonnes "S" et "T".

Cas d'Utilisation

- La fonction `ResetSourceData` est utilisée après la copie des données de la feuille "Movement Sheet" vers la feuille "Mouvement Log".
- Elle est responsable de la réinitialisation de toutes les informations pertinentes pour permettre une nouvelle saisie de données. Cela inclut la suppression des valeurs existantes dans des cellules spécifiques, la réinitialisation des données répétitives, et la remise à zéro des champs supplémentaires.
- Cette fonction assure ainsi que la feuille source est prête pour une nouvelle utilisation, en nettoyant correctement la feuille après chaque transfert de données.

4.10.6 ValidateDates

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `Dim ws As Worksheet` : Déclare une variable pour stocker la feuille de calcul à valider.
  - `Dim startDate As Date` : Déclare une variable pour stocker la date après conversion.
  - `Dim startDateValue As Variant` : Déclare une variable pour stocker la valeur récupérée des cellules fusionnées.
  - `Dim dateFormat As String` : Déclare une variable pour spécifier le format attendu de la date.
- **Définition du Format de Date Attendu**
  - `dateFormat = "dd/mm/yyyy"` : Spécifie le format de date attendu, ici "jour/mois/année".
- **Initialisation de la Fonction**
  - `ValidateDates = False` : Initialise la fonction avec une valeur par défaut de False, signifiant que la validation n'a pas encore réussi.
- **Récupération de la Valeur de la Date**
  - `startDateValue = ws.Range("L9").MergeArea.Cells(1, 1).value` : Récupère la valeur de la cellule fusionnée située en `L9` de la feuille "Movement Sheet". Cette cellule est supposée contenir la date à valider.
- **Vérification de la Présence d'une Valeur**
  - `If IsEmpty(startDateValue) Then` : Vérifie si la valeur récupérée est vide.
  - `MsgBox "Erreur : La date doit etre rempli.", vbCritical` : Affiche un message d'erreur si la date est manquante et quitte la fonction.

— **Validation du Format de Date**
  — `If IsDate(startDateValue) Then` : Vérifie si la valeur récupérée est une date valide.
  — `startDate = CDate(startDateValue)` : Convertit la valeur en type Date.
  — `If Format(startDate, dateFormat) = Format(startDateValue, dateFormat) Then` : Vérifie si la date formatée correspond au format attendu.
  — `ValidateDates = True` : Si la date est valide et au bon format, la fonction retourne True.
  — `Else` : Si le format ne correspond pas :
  — `MsgBox "La date n'est pas au format correct. Veuillez utiliser le format jj/mm/aa` `vbExclamation` : Affiche un message d'avertissement pour indiquer l'erreur de format.
— **Gestion des Dates Invalides**
  — `Else` : Si la valeur n'est pas une date valide :
  — `MsgBox "Veuillez entrer une date valide.", vbExclamation` : Affiche un message d'avertissement pour demander une entrée de date correcte.

Cas d'Utilisation

— La fonction `ValidateDates` est utilisée pour s'assurer que les dates saisies dans la feuille "Movement Sheet" sont correctement remplies et au format attendu avant de procéder à d'autres opérations.
— Elle joue un rôle crucial dans la validation des données d'entrée pour éviter les erreurs qui pourraient survenir à cause de formats de date incorrects ou de données manquantes.
— En garantissant que les dates sont valides et correctement formatées, cette fonction contribue à la robustesse et à la précision des traitements de données dans l'application VBA.

4.10.7 ExportRangeAsPDFAndSendEmail2

La fonction `ExportRangeAsPDFAndSendEmail2` est conçue pour accomplir les tâches suivantes :

1. Définir la plage de données à exporter en PDF.
2. Configurer les paramètres de mise en page pour l'exportation.
3. Exporter la plage de données en fichier PDF.
4. Envoyer le fichier PDF par email à plusieurs destinataires à partir des adresses stockées dans une plage spécifique de la feuille de calcul.

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `Dim ws As Worksheet` : Variable pour référencer la feuille de calcul contenant les données à exporter.
  — `Dim pdfPath As String` : Variable pour stocker le chemin où le fichier PDF sera enregistré.
  — `Dim pdfFileName As String` : Variable pour le nom complet du fichier PDF.
  — `Dim emailRange As Range` : Variable pour définir la plage contenant les adresses email des destinataires.
  — `Dim cell As Range` : Variable pour itérer sur chaque cellule de la plage des adresses email.
  — `Dim outlookApp As Object` : Variable pour créer une instance de l'application Outlook.
  — `Dim outlookMail As Object` : Variable pour créer et configurer un nouvel email.

- `Dim exportRange As Range` : Variable pour la plage de données à exporter.
- `Dim currentDate As String` : Variable pour stocker la date actuelle au format "aaaa-mm-jj".
- `Dim lastRow As Long` : Variable pour déterminer la dernière ligne utilisée dans la feuille de calcul.

— **Définition de la Plage de Données à Exporter**
- `Set ws = ThisWorkbook.Sheets("Movement Sheet")` : Référence la feuille de calcul "Movement Sheet".
- `lastRow = ws.Cells(ws.Rows.Count, "F").End(xlUp).Row + 30` : Détermine la dernière ligne utilisée dans la colonne F, augmentée de 30 lignes pour inclure des données supplémentaires.
- `Set exportRange = ws.Range("E4:N" & lastRow)` : Définit la plage de cellules à exporter, allant de la cellule E4 jusqu'à la dernière ligne déterminée.

— **Configuration du Nom et du Chemin du Fichier PDF**
- `magasin_text = Sheets("Menu").Shapes("magasin_text").TextFrame.Characters.Text` : Récupère le texte d'une forme nommée "magasin_text" dans la feuille "Menu" pour inclure dans le nom du fichier.
- `currentDate = Format(Date, "yyyy-mm-dd")` : Formate la date actuelle au format "yyyy-mm-dd".
- `pdfFileName = pdfPath & magasin_text & " " & currentDate & " Movement sheet.pdf"` : Construit le nom du fichier PDF en combinant le texte de la forme et la date actuelle.

— **Configuration de la Mise en Page pour l'Exportation**
- `With ws.PageSetup` : Configure les paramètres de mise en page pour s'assurer que la plage est exportée correctement.
- `.PrintArea = exportRange.Address` : Définit la zone d'impression comme étant la plage à exporter.
- `.Orientation = xlPortrait` : Définit l'orientation de la page en portrait.
- `.Zoom = False` : Désactive le zoom automatique.
- `.FitToPagesWide = 1` et `.FitToPagesTall = 1` : Ajuste la plage pour qu'elle tienne sur une seule page en largeur et hauteur.

— **Exportation en PDF**
- `exportRange.ExportAsFixedFormat Type:=xlTypePDF, Filename:=pdfFileName, Quality:=` Exporte la plage de données comme un fichier PDF avec une qualité standard.

— **Envoi du PDF par Email**
- `Set emailRange = ws.Range("S21:S30")` : Définit la plage contenant les adresses email.
- `Set outlookApp = CreateObject("Outlook.Application")` : Crée une instance de l'application Outlook.
- `For Each cell In emailRange` : Parcourt chaque cellule dans la plage des adresses email.
- `If Trim(cell.value) <> "" Then` : Vérifie si la cellule contient une adresse email non vide.
- `Set outlookMail = outlookApp.CreateItem(0)` : Crée un nouvel email.
- `With outlookMail` : Configure les propriétés de l'email.
  - `.To = cell.value` : Définit le destinataire de l'email.
  - `.Subject = "Exported PDF"` : Définit le sujet de l'email.
  - `.Body = "Please find the attached PDF document."` : Définit le corps de l'email.
  - `.Attachments.Add pdfFileName` : Ajoute le fichier PDF en tant que pièce jointe.
  - `.Display` : Affiche l'email pour révision (remplacer par `.Send` pour un envoi direct).

— **Gestion des Erreurs et Nettoyage**
  — `On Error GoTo ErrorHandler` : Redirige vers le gestionnaire d'erreurs en cas de problème.
  — `MsgBox "An error occurred: " & Err.Description` : Affiche un message d'erreur si une exception se produit.
  — `Set outlookMail = Nothing` et `Set outlookApp = Nothing` : Libère les objets Outlook pour éviter les fuites de mémoire.
  — `MsgBox "Emails sent successfully!"` : Affiche un message de confirmation après l'envoi des emails.

4.10.8 CreateDropDownListFromColumnM

La fonction `CreateDropDownListFromColumnM` réalise les tâches suivantes :

1. Définir les feuilles de calcul source et destination.
2. Déterminer la plage de données à partir de laquelle la liste déroulante sera créée.
3. Configurer les cellules de la feuille de calcul de destination pour utiliser les données de la colonne spécifiée comme source pour les listes déroulantes.

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `Dim wsData As Worksheet` : Variable pour référencer la feuille de calcul contenant les données source.
  — `Dim wsDest As Worksheet` : Variable pour référencer la feuille de calcul où les listes déroulantes seront ajoutées.
  — `Dim lastRow As Long` : Variable pour déterminer la dernière ligne contenant des données dans la colonne M de la feuille source.
  — `Dim rngM As Range` : Variable pour définir la plage de données à utiliser pour la liste déroulante.
  — `Dim listRangeAddress As String` : Variable pour stocker l'adresse de la plage de données au format de chaîne.
  — `Dim cell As Range` : Variable pour itérer sur chaque cellule de la plage où la validation des données sera appliquée.
— **Définition des Feuilles de Calcul**
  — `Set wsData = ThisWorkbook.Sheets("Data")` : Référence la feuille `Data` contenant les données source.
  — `Set wsDest = ThisWorkbook.Sheets("Movement Sheet")` : Référence la feuille `Movement Sheet` où les listes déroulantes seront configurées.
— **Détermination de la Plage de Données**
  — `lastRow = wsData.Cells(wsData.Rows.Count, "M").End(xlUp).Row` : Trouve la dernière ligne avec des données dans la colonne M de la feuille source.
  — `Set rngM = wsData.Range("M2:M" & lastRow)` : Définit la plage de données de M2 à la dernière ligne avec des données dans la colonne M.
— **Définition de l'Adresse de la Plage**
  — `listRangeAddress = wsData.Name & "!" & rngM.Address` : Construit l'adresse complète de la plage de données, incluant le nom de la feuille et l'adresse de la plage.
— **Suppression de la Validation Précédente**

- `On Error Resume Next` : Ignore les erreurs pour éviter les interruptions si aucune validation n'existe déjà.
- `wsDest.Range("Q21:Q30").Validation.Delete` : Supprime toute validation de données précédente dans la plage Q21 à Q30 de la feuille de destination.

— **Configuration de la Validation des Données**
- `For Each cell In wsDest.Range("Q21:Q30")` : Itère sur chaque cellule dans la plage Q21 à Q30.
- `With cell.Validation` : Configure la validation des données pour chaque cellule.
  - `.Delete` : Supprime toute validation précédente pour éviter les conflits.
  - `.Add Type:=xlValidateList, AlertStyle:=xlValidAlertStop, Formula1:="=" & listRangeAddress` : Ajoute une validation de liste en utilisant la plage de données définie.
  - `.IgnoreBlank = True` : Permet les cellules vides.
  - `.InCellDropdown = True` : Affiche une liste déroulante dans les cellules.
  - `.ShowInput = True` et `.ShowError = True` : Affiche les messages d'entrée et d'erreur associés à la validation.

— **Gestion des Erreurs**
- `On Error GoTo ErrorHandler` : Active la gestion des erreurs.
- `MsgBox "Error " & Err.Number & ": " & Err.Description` : Affiche un message d'erreur si une erreur se produit pendant l'exécution de la fonction.

Cas d'Utilisation

La fonction `CreateDropDownListFromColumnM` est utilisée pour automatiser la création de listes déroulantes dans une feuille de calcul Excel. En utilisant les données d'une colonne spécifique d'une feuille source, cette fonction permet d'assurer que les utilisateurs peuvent sélectionner des valeurs à partir d'une liste prédéfinie et mise à jour dynamiquement. Cette approche réduit les erreurs de saisie, améliore la cohérence des données, et simplifie l'entrée des données en offrant des options sélectionnables à l'utilisateur.

4.10.9 CheckQAndRetrieveEmailAndPost

Analyse des Composants de la Fonction

— **Déclaration des Variables**
- `Dim wsDest As Worksheet` : Variable pour référencer la feuille de calcul où les données seront mises à jour (`Movement Sheet`).
- `Dim wsData As Worksheet` : Variable pour référencer la feuille de calcul contenant les données source (`Data`).
- `Dim selectedName As String` : Variable pour stocker le nom sélectionné dans la colonne Q de la feuille de destination.
- `Dim rngM As Range` : Variable pour définir la plage de données dans la colonne M de la feuille de données.
- `Dim rowIndex As Variant` : Variable pour stocker l'index de la ligne où le nom sélectionné est trouvé.
- `Dim lastRow As Long` : Variable pour déterminer la dernière ligne avec des données dans la colonne M de la feuille de données.

  - `Dim cell As Range` : Variable pour itérer à travers chaque cellule de la plage Q21 à la dernière ligne de la feuille de destination.
  - `Dim targetRow As Long` : Variable pour stocker la ligne cible dans la feuille de destination où les données seront mises à jour.
  - `Dim foundRow As Long` : Variable pour stocker la ligne trouvée dans la feuille de données correspondant au nom sélectionné.
- **Gestion des Erreurs**
  - `On Error GoTo ErrorHandler` : Définit la gestion des erreurs pour capturer et afficher les messages d'erreur.
- **Définition des Feuilles de Calcul**
  - `Set wsDest = ThisWorkbook.Sheets("Movement Sheet")` : Référence la feuille `Movement Sheet` pour la mise à jour des données.
  - `Set wsData = ThisWorkbook.Sheets("Data")` : Référence la feuille `Data` contenant les données sources.
- **Détermination de la Plage de Données**
  - `lastRow = wsData.Cells(wsData.Rows.Count, "M").End(xlUp).Row` : Trouve la dernière ligne avec des données dans la colonne M de la feuille `Data`.
  - `Set rngM = wsData.Range("M2:M" & lastRow)` : Définit la plage de données dans la colonne M, de M2 à la dernière ligne avec des données.
- **Traitement des Cellules dans la Colonne Q**
  - `For Each cell In wsDest.Range("Q21:Q" & wsDest.Cells(wsDest.Rows.Count, "Q").End(`
    Itère à travers chaque cellule dans la colonne Q de la feuille `Movement Sheet`, de Q21 jusqu'à la dernière ligne remplie.
  - `selectedName = cell.Value` : Récupère la valeur de la cellule actuelle dans la colonne Q.
- **Recherche du Nom Sélectionné**
  - `If selectedName <> "" Then` : Vérifie si un nom a été sélectionné.
  - `rowIndex = Application.Match(selectedName, rngM, 0)` : Recherche le nom sélectionné dans la plage de données de la colonne M. `Application.Match` retourne l'index de la ligne où le nom est trouvé, ou une erreur si non trouvé.
  - `If Not IsError(rowIndex) Then` : Vérifie si la recherche a réussi.
  - `foundRow = rngM.Cells(rowIndex).Row` : Calcule la ligne réelle dans la feuille `Data` où le nom est trouvé.
  - `targetRow = cell.Row` : Définit la ligne cible dans la feuille `Movement Sheet` pour mettre à jour les données.
  - `wsDest.Range("R" & targetRow).Value = wsData.Range("Q" & foundRow).Value` : Récupère le poste de la colonne Q dans la feuille `Data` et le place dans la colonne R de la feuille `Movement Sheet`.
  - `wsDest.Range("S" & targetRow).Value = wsData.Range("P" & foundRow).Value` : Récupère l'adresse e-mail de la colonne P dans la feuille `Data` et le place dans la colonne S de la feuille `Movement Sheet`.
- **Gestion des Erreurs**
  - `MsgBox "Erreur " & Err.Number & ": " & Err.Description, vbCritical` : Affiche un message d'erreur en cas de problème pendant l'exécution de la fonction.

Cas d'Utilisation

La fonction `CheckQAndRetrieveEmailAndPost` est utilisée pour automatiser la recherche et la mise à jour des informations dans une feuille de calcul Excel en fonction des sélections faites dans une autre feuille. En extrayant les données pertinentes basées sur les noms sélectionnés et en les plaçant dans les colonnes appropriées, cette fonction facilite la gestion des informations et améliore l'efficacité des tâches de mise à jour des données. Elle est utile dans des scénarios où il est nécessaire de relier des informations entre plusieurs sources et de mettre à jour dynamiquement des données en fonction des sélections effectuées.

4.11 module 4

4.11.1 SetDefaultValues

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `Dim wsDaily As Worksheet` : Variable pour référencer la feuille de calcul `Daily inventory` où les valeurs par défaut seront définies.
- **Gestion des Erreurs**
  - `On Error GoTo ErrorHandler` : Définit le gestionnaire d'erreurs pour capturer et afficher les messages d'erreur éventuels qui pourraient survenir pendant l'exécution de la fonction.
- **Définition de la Feuille de Calcul**
  - `Set wsDaily = ThisWorkbook.Sheets("Daily inventory")` : Référence la feuille de calcul `Daily inventory` où les valeurs seront mises à jour.
- **Vérification et Définition de la Date Par Défaut**
  - `If Not ValidateDateInput() Then` : Appelle la fonction `ValidateDateInput` pour vérifier si la date fournie est valide.
  - `wsDaily.Range("C8").Value = Date` : Si la date est invalide, définit la cellule C8 à la date actuelle.
  - `wsDaily.Range("C8").NumberFormat = "dd/mm/yyyy"` : Formate la cellule C8 pour afficher la date au format `jj/mm/aaaa`.
  - `MsgBox "La date était invalide. Elle a été définie à aujourd'hui : " & Format(Dat "dd/mm/yyyy"), vbInformation, "Information"` : Affiche un message informatif indiquant que la date a été corrigée et définie à la date actuelle.
- **Vérification et Définition de l'Heure Par Défaut**
  - `If Not ValidateTime() Then` : Appelle la fonction `ValidateTime` pour vérifier si l'heure fournie est valide.
  - `wsDaily.Range("C9").Value = Format(Now, "HH:mm")` : Si l'heure est invalide, définit la cellule C9 à l'heure actuelle au format `HH:mm`.
  - `wsDaily.Range("C9").NumberFormat = "HH:mm"` : Formate la cellule C9 pour afficher l'heure au format `HH:mm`.
- **Gestion des Erreurs**
  - `MsgBox "Erreur Automation: " & Err.Description, vbCritical, "Erreur"` : Affiche un message d'erreur critique en cas de problème durant l'exécution de la fonction.

Cas d'Utilisation

La fonction `SetDefaultValues` est utile pour garantir que certaines cellules dans la feuille de calcul `Daily inventory` contiennent des valeurs valides et actuelles. En vérifiant et en définissant les valeurs par défaut pour la date et l'heure, elle assure que les utilisateurs disposent des informations correctes et à jour, même si des erreurs ou des oublis surviennent lors de l'entrée des données. Cette automatisation aide à maintenir la cohérence des données et à éviter les erreurs potentielles, ce qui est essentiel dans un contexte de gestion d'inventaire ou de suivi quotidien.

4.11.2 ValidateDateInput

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `Dim ws As Worksheet` : Variable pour référencer la feuille de calcul `Daily inventory`.
  - `Dim dateValue As Variant` : Variable pour stocker la valeur extraite de la cellule `C8`.
  - `Dim dateValueInCell As Date` : Variable pour convertir et stocker la valeur de la date en format de date.
- **Définition de la Feuille de Calcul**
  - `Set ws = ThisWorkbook.Sheets("Daily inventory")` : Référence la feuille de calcul `Daily inventory`, où la date sera validée.
- **Initialisation de la Fonction**
  - `ValidateDateInput = False` : Initialise la fonction avec une valeur par défaut de `False`, ce qui indique que la date n'est pas encore validée.
- **Récupération de la Valeur de la Cellule**
  - `dateValue = ws.Range("C8").Value` : Extrait la valeur de la cellule `C8`, qui est censée contenir la date à valider.
- **Vérification de la Validité de la Date**
  - `If IsDate(dateValue) Then` : Vérifie si la valeur extraite est une date valide.
  - `dateValueInCell = CDate(dateValue)` : Convertit la valeur en une date réelle pour une manipulation ultérieure.
  - `If dateValueInCell <= Date Then` : Vérifie si la date est antérieure ou égale à la date actuelle. Si la date est dans le futur, elle est considérée comme invalide.
- **Retour de la Valeur de la Fonction**
  - `ValidateDateInput = True` : Si toutes les conditions sont remplies, la fonction retourne `True`, indiquant que la date est valide et ne dépasse pas la date actuelle.

Cas d'Utilisation

La fonction `ValidateDateInput` est essentielle pour garantir l'intégrité des données de date dans les applications de gestion de données et de formulaires. Elle est utilisée pour :

- **Assurer la Précision des Données** : En vérifiant que les dates ne sont pas dans le futur, elle empêche les erreurs de saisie et garantit que les informations sont actuelles.
- **Prévenir les Erreurs de Saisie** : En validant la date avant qu'elle ne soit utilisée dans d'autres calculs ou processus, elle aide à éviter les erreurs potentielles qui pourraient affecter l'exactitude des rapports et des analyses.
- **Faciliter le Traitement des Données** : En intégrant cette fonction dans des processus plus vastes de gestion de données, les utilisateurs peuvent s'assurer que les entrées de date sont

conformes aux attentes et aux exigences du système.

En résumé, `ValidateDateInput` est une fonction cruciale pour maintenir la qualité et la précision des données dans les feuilles de calcul, en particulier pour les applications qui nécessitent une gestion rigoureuse des dates.

4.11.3 ValidateTime

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `Dim ws As Worksheet` : Variable pour référencer la feuille de calcul `Daily inventory` où l'heure sera validée.
  - `Dim timeValue As Variant` : Variable pour stocker la valeur extraite de la cellule `C9`.
  - `Dim timeValueInCell As Date` : Variable pour convertir et stocker la valeur de l'heure en format de date/heure.
- **Définition de la Feuille de Calcul**
  - `Set ws = ThisWorkbook.Sheets("Daily inventory")` : Associe la feuille de calcul `Daily inventory` à la variable `ws`.
- **Initialisation de la Fonction**
  - `ValidateTime = False` : Initialise la fonction avec une valeur par défaut de `False`, ce qui indique que l'heure n'est pas encore validée.
- **Récupération de la Valeur de la Cellule**
  - `timeValue = ws.Range("C9").Value` : Extrait la valeur de la cellule `C9`, qui est censée contenir l'heure à valider.
- **Vérification de la Validité de l'Heure**
  - `If IsDate(timeValue) Then` : Vérifie si la valeur extraite est une date ou une heure valide. La fonction `IsDate` peut aussi identifier les heures en tant que valeurs de date.
  - `timeValueInCell = CDate(timeValue)` : Convertit la valeur en une date réelle pour une manipulation ultérieure. Cette conversion permet de vérifier si l'entrée est une valeur de date ou d'heure correcte.
- **Retour de la Valeur de la Fonction**
  - `ValidateTime = True` : Si la condition `IsDate` est remplie, cela signifie que la valeur dans `C9` est une heure valide, donc la fonction retourne `True`.

Cas d'Utilisation

La fonction `ValidateTime` est utilisée pour :

- **Assurer la Précision des Données Horaires** : Elle garantit que les heures saisies sont correctement formatées et valides, ce qui est crucial pour les systèmes qui dépendent des entrées horaires pour le calcul ou l'enregistrement des données.
- **Prévenir les Erreurs de Saisie** : En validant l'entrée horaire avant son utilisation dans des processus ou des calculs ultérieurs, cette fonction aide à éviter les erreurs potentielles qui pourraient nuire à l'exactitude des rapports et des analyses.
- **Faciliter le Traitement des Données** : En intégrant cette fonction dans les formulaires et les systèmes de gestion, les utilisateurs peuvent s'assurer que les données horaires saisies sont conformes aux attentes et aux exigences du système.

En résumé, `ValidateTime` est une fonction essentielle pour maintenir la qualité et la précision des données horaires dans les feuilles de calcul, en assurant que les heures sont valides et correctement

formatées pour les besoins de l'application ou du processus en cours.

4.11.4 Stocker

La fonction `Stocker` est une macro VBA utilisée pour transférer des données depuis une feuille de calcul source (*Daily inventory*) vers une feuille de calcul de destination (*Inventory Log*), en les ajoutant à un tableau structuré dans cette feuille de destination. Cette fonction est cruciale pour la gestion des stocks, car elle permet de centraliser et d'enregistrer les informations pertinentes liées aux inventaires. Voici une explication détaillée de cette fonction :

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `wsSource` : Feuille de calcul source (*Daily inventory*).
  - `wsDestination` : Feuille de calcul destination (*Inventory Log*).
  - `tbl` : Table structurée dans la feuille de destination.
  - `lastRowD` : Dernière ligne utilisée dans la colonne E de la feuille source.
  - `newRow` : Nouvelle ligne ajoutée au tableau de la feuille destination.
  - `currentTime` : Heure actuelle.
  - `heureFormatted` : Heure formatée en "HH :mm".
  - `dateValue`, `typeInventaire`, `regAdjustValue`, `marqueValue` : Valeurs extraites des cellules spécifiques.
- **Gestion des Erreurs**
  `On Error GoTo ErrorHandler` : Gestion des erreurs pour afficher un message en cas de problème.
- **Définition des Feuilles de Calcul**
  - `Set wsSource = ThisWorkbook.Sheets("Daily inventory")` : Définit la feuille source.
  - `Set wsDestination = ThisWorkbook.Sheets("Inventory Log")` : Définit la feuille de destination.
- **Vérification de la Présence de Données**
  `If lastRowD < 12 Then` : Vérifie s'il y a des données à traiter à partir de la ligne 12. Si aucune donnée n'est trouvée, un message d'erreur est affiché et la fonction s'arrête.
- **Initialisation des Variables**
  - `currentTime = Format(Now)` : Obtient l'heure actuelle.
  - `heureFormatted = Format(wsSource.Range("C9").value, "HH:mm")` : Formate l'heure de la cellule C9 en "HH :mm".
- **Vérification des Champs Critiques**
  `If IsEmpty(dateValue) Or IsEmpty(typeInventaire) Or IsEmpty(regAdjustValue) Or IsEmpty(marqueValue) Then` : Vérifie que les champs critiques (date, type d'inventaire, régulier/ajustement, marque) ne sont pas vides.
- **Boucle de Traitement des Données**
  `For i = 12 To lastRowD` : Itère à travers chaque ligne de données à partir de la ligne 12.
  - `names = ""` : Concatène les valeurs de la colonne J pour chaque ligne en une seule chaîne.
  - `If cellValue <> 0 And Not IsEmpty(articleValue) And Not IsEmpty(emballageValue) And Not IsEmpty(quantiteValue) Then` : Vérifie que les valeurs ne sont pas nulles avant de les enregistrer.
  - `Set newRow = tbl.ListRows.Add` : Ajoute une nouvelle ligne au tableau.

— `With newRow ... End With` : Remplit la nouvelle ligne avec les valeurs correspondantes.
— **Vérification Finale et Message de Succès**
`If Not dataToSave Then` : Si aucune donnée n'est enregistrée, un message d'erreur s'affiche et la fonction se termine. Sinon, la fonction se termine avec un message de succès.

4.11.5 ReinitialiserValeurs

La fonction `ReinitialiserValeurs` est une macro VBA conçue pour réinitialiser et nettoyer des valeurs spécifiques dans la feuille de calcul *Daily inventory*. Cette fonction est utilisée pour préparer la feuille de calcul pour une nouvelle session de saisie de données en effaçant les anciennes valeurs et en réinitialisant les champs importants à leurs valeurs par défaut. Voici une explication détaillée des différentes étapes de cette fonction :

Analyse des Composants de la Fonction

— **Déclaration et Initialisation des Variables**
  — `wsDaily` : Référence à la feuille de calcul *Daily inventory*.
  — `lastRowNames` : Détermine la dernière ligne utilisée dans la colonne J plus 2. Cela sert à définir la plage de cellules à réinitialiser.
  — `cell` : Variable utilisée pour itérer à travers chaque cellule dans la plage spécifiée.
— **Définition de la Feuille de Calcul**
`Set wsDaily = ThisWorkbook.Sheets("Daily inventory")` : Définit la feuille de calcul sur laquelle les opérations seront effectuées.
— **Détermination de la Dernière Ligne à Réinitialiser**
`lastRowNames = wsDaily.Cells(wsDaily.Rows.Count, "J").End(xlUp).Row + 2` : Trouve la dernière ligne avec des données dans la colonne J et ajoute 2 pour déterminer la fin de la plage à réinitialiser.
— **Réinitialisation des Valeurs dans les Colonnes J, K, et L**
Efface les valeurs des cellules dans les colonnes J, K, et L pour chaque ligne de la plage allant de J15 jusqu'à la dernière ligne spécifiée (lastRowNames) dans la feuille de calcul wsDaily.
— **Réinitialisation de la Date et de l'Heure**
Attribue la date actuelle à la cellule C8 et formate-la au format "jour/mois/année". Attribue l'heure actuelle à la cellule C9 et formate-la au format "heure ".
— **Effacement du Champ de Type d'Inventaire**
`wsDaily.Range("F8").ClearContents` : Efface le contenu de la cellule F8, qui contient le type d'inventaire.
— **Réinitialisation des Quantités** Détermine la dernière ligne non vide dans la colonne B de la feuille de calcul wsDaily et attribue la valeur 0 à toutes les cellules de la colonne E, de la ligne 12 jusqu'à cette dernière ligne.

Objectifs et Utilisation de la Fonction

La fonction `ReinitialiserValeurs` a pour objectif principal de réinitialiser les champs de la feuille de calcul *Daily inventory* afin de préparer un nouvel enregistrement d'inventaire. Voici quelques-unes des utilisations et avantages clés :

— **Préparation pour une Nouvelle Saisie** : La fonction assure que les champs précédemment utilisés sont nettoyés et réinitialisés, permettant ainsi une nouvelle saisie d'inventaire sans interférer avec les données antérieures.

— **Assurance de la Cohérence des Données** : En réinitialisant les valeurs de manière systématique, la fonction aide à maintenir la cohérence et la précision des enregistrements d'inventaire.
— **Gain de Temps et Réduction des Erreurs** : En automatisant le processus de réinitialisation, la fonction réduit le temps nécessaire pour préparer la feuille de calcul et minimise les risques d'erreurs humaines lors de la réinitialisation manuelle des valeurs.

En résumé, la fonction `ReinitialiserValeurs` est une macro essentielle pour la gestion des données d'inventaire, facilitant le nettoyage et la préparation de la feuille de calcul pour de nouvelles entrées tout en garantissant la précision et la cohérence des données.

4.11.6 CheckInventoryType

La fonction `CheckInventoryType` est une macro VBA conçue pour déterminer et mettre à jour le type d'inventaire basé sur les données présentes dans deux feuilles de calcul : *Daily inventory* et *Inventory Log*. Elle compare les valeurs de date et de type d'inventaire dans la feuille de calcul *Daily inventory* avec les enregistrements existants dans *Inventory Log* pour décider si le type d'inventaire doit être marqué comme "Ajustement" ou "Régulier". Voici une explication détaillée de la fonction :

Analyse des Composants de la Fonction

— **Déclaration et Initialisation des Variables**
  — `wsDest` : Référence à la feuille de calcul *Daily inventory*, où les données actuelles de l'inventaire sont stockées.
  — `ws` : Référence à la feuille de calcul *Inventory Log*, qui contient les enregistrements d'inventaire historiques.
  — `lastRow` : Détermine la dernière ligne avec des données dans la colonne A de la feuille *Inventory Log*.
  — `currentDate` : Stocke la date actuelle de la feuille *Daily inventory* (cellule C8).
  — `currentType` : Stocke le type d'inventaire actuel de la feuille *Daily inventory* (cellule E9).
  — `i` : Utilisé comme compteur pour parcourir les lignes de la feuille *Inventory Log*.
  — `isAdjustment` : Booléen utilisé pour déterminer si le type d'inventaire est un ajustement ou non.
— **Définition des Feuilles de Calcul** Définit wsDest comme la feuille de calcul "Daily inventory" pour les données d'inventaire actuelles, et ws comme la feuille de calcul "Inventory Log" pour les enregistrements d'inventaire.
— **Détermination de la Dernière Ligne**
Trouve la dernière ligne contenant des données dans la colonne A de la feuille "Inventory Log", en partant du bas vers le haut.
— **Lecture des Valeurs Actuelles**
Lit la date actuelle depuis la cellule C8 et le type d'inventaire depuis la cellule E9 de la feuille "Daily inventory".
— **Vérification des Enregistrements** Itère à travers chaque ligne de la feuille "Inventory Log" à partir de la ligne 2, et compare les valeurs de la date et du type d'inventaire avec les valeurs actuelles. Si une correspondance est trouvée, marque le type d'inventaire comme "Ajustement" et quitte la boucle pour éviter des itérations supplémentaires.
— **Mise à Jour du Type d'Inventaire**
Vérifie si le type d'inventaire a été marqué comme ajustement. Si c'est le cas, met à jour la

cellule E8 de la feuille "Daily inventory" avec "Ajustement". Sinon, met à jour la cellule E8 avec "Régulier".

Objectifs et Utilisation de la Fonction

La fonction `CheckInventoryType` a pour but de vérifier si le type d'inventaire actuel dans la feuille *Daily inventory* est un ajustement ou un type régulier en fonction des enregistrements passés. Voici les principaux objectifs et avantages de cette fonction :

- **Validation des Données** : La fonction assure que le type d'inventaire est correctement marqué comme “Ajustement” ou “Régulier” en se basant sur des données historiques. Cela aide à maintenir l'exactitude des classifications d'inventaire.
- **Automatisation du Processus** : En automatisant la vérification et la mise à jour du type d'inventaire, la fonction réduit le besoin d'intervention manuelle, ce qui diminue le risque d'erreurs et améliore l'efficacité des opérations de gestion d'inventaire.
- **Amélioration de la Précision des Rapports** : En garantissant que le type d'inventaire est correctement catégorisé, la fonction contribue à la précision des rapports d'inventaire, ce qui est crucial pour la gestion et l'analyse des stocks.

En résumé, la fonction `CheckInventoryType` est une macro importante pour assurer la précision et la cohérence des informations d'inventaire en comparant les données actuelles avec les enregistrements passés. Elle facilite la gestion des stocks en automatisant la vérification et la mise à jour du type d'inventaire, ce qui permet une gestion plus efficace des données d'inventaire.

4.11.7 ExportRangeAsPDFAndSendEmail

La macro VBA `ExportRangeAsPDFAndSendEmail` est conçue pour exporter une plage de données d'une feuille de calcul Excel en tant que fichier PDF et envoyer ce fichier par email à une liste de destinataires spécifiée dans la feuille de calcul. Cette fonction intègre plusieurs étapes cruciales pour garantir la bonne exécution de la tâche. Voici une explication détaillée de chaque composant de la fonction :

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `ws` : Référence à la feuille de calcul contenant les données à exporter (*Daily inventory*).
  - `pdfPath` : Chemin d'accès où le fichier PDF sera enregistré.
  - `pdfFileName` : Nom du fichier PDF, incluant la date actuelle et un texte récupéré d'une forme dans une autre feuille.
  - `emailRange` : Plage de cellules contenant les adresses email des destinataires.
  - `cell` : Cellule dans la plage d'adresses email utilisée pour envoyer les emails.
  - `outlookApp` : Objet représentant l'application Outlook pour envoyer les emails.
  - `outlookMail` : Objet représentant un email à envoyer.
  - `exportRange` : Plage de cellules à exporter en PDF.
  - `magasin text` : Texte récupéré depuis une forme dans la feuille *Menu*.
  - `currentDate` : Date actuelle formatée pour inclure dans le nom du fichier PDF.
- **Configuration de la Feuille de Calcul et de la Plage à Exporter**
  Définit la feuille de calcul source comme "Daily inventory" et spécifie la plage de cellules à exporter, de B2 à E50.

— **Définition du Nom et du Chemin du Fichier PDF**
Récupère le texte d'une forme nommée magasin text dans la feuille "Menu", formate la date actuelle pour l'inclure dans le nom du fichier PDF, et construit le chemin complet et le nom du fichier PDF. **Configuration de la Mise en Page pour l'Export PDF** Définit les paramètres de mise en page pour que la plage d'exportation soit ajustée sur une seule page en mode portrait.
— **Exportation de la Plage en PDF**
Exporte la plage spécifiée en tant que fichier PDF avec une qualité standard.
— **Envoi du PDF par Email**
Définit la plage contenant les adresses email des destinataires, initialise l'application Outlook, puis parcourt chaque adresse email dans la plage. Pour chaque adresse non vide, crée un nouvel email, configure ses paramètres, ajoute le fichier PDF en pièce jointe, et prépare l'email pour l'envoi.
— **Nettoyage et Gestion des Erreurs**
Libère les objets Outlook pour libérer les ressources utilisées, affiche un message de succès pour indiquer que les emails ont été envoyés avec succès, et capture et affiche les éventuelles erreurs survenues pendant l'exécution de la fonction.

Objectifs et Utilisation de la Fonction

La fonction `ExportRangeAsPDFAndSendEmail` est conçue pour automatiser le processus d'exportation d'une plage de données en tant que PDF et l'envoi de ce PDF à une liste de destinataires via email. Voici les principaux objectifs et avantages :

— **Automatisation** : Simplifie l'envoi de rapports PDF par email en automatisant l'exportation et l'envoi des fichiers, ce qui réduit le temps et les erreurs humaines associées à ces tâches.
— **Rapidité** : Permet d'envoyer rapidement des documents à plusieurs destinataires sans intervention manuelle, améliorant ainsi l'efficacité des communications.
— **Personnalisation** : Le nom du fichier PDF inclut des informations personnalisées comme le texte de la forme "magasin_text" et la date actuelle, ce qui facilite l'organisation et l'identification des fichiers envoyés.

En résumé, `ExportRangeAsPDFAndSendEmail` est une fonction essentielle pour automatiser la distribution de rapports en PDF, offrant une solution efficace pour partager des informations critiques avec plusieurs parties prenantes par email.

4.11.8 DailyButton1 Click

La macro VBA `DailyButton1_Click` est conçue pour être déclenchée par un bouton dans une feuille de calcul Excel. Elle exécute une série d'opérations liées à la gestion des données d'inventaire quotidien, telles que la vérification et la mise à jour de la date, le stockage des données dans un autre emplacement, l'exportation de ces données en PDF et l'envoi du PDF par email. Voici une explication détaillée de chaque composant de la fonction :

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `wsDaily` : Référence à la feuille de calcul nommée *Daily inventory* où les données d'inventaire quotidien sont gérées.

  - `success` : Variable booléenne utilisée pour indiquer si les opérations de stockage des données ont réussi.
- **Vérification et Mise à Jour de la Date**
  Vérifie la validité de la date dans la cellule C8 de la feuille "Daily inventory". Si la date est invalide ou dans le futur, elle est remplacée par la date actuelle, et la cellule est formatée en "jour/mois/année". Un message informe l'utilisateur que la date a été modifiée.
- **Stockage des Données** Appelle la procédure Stocker pour transférer les données de la feuille "Daily inventory" à la feuille "Inventory Log", en utilisant le paramètre success pour obtenir un retour sur l'état de l'opération.
- **Gestion des Opérations Basées sur le Résultat du Stockage**
  Vérifie si le stockage des données a réussi. Si oui, appelle la fonction ExportRangeAsPDFAndSendEmail pour exporter les données en PDF et envoyer le fichier par email, puis appelle ReinitialiserValeurs pour préparer la feuille "Daily inventory" pour de nouvelles données. Si le stockage échoue, affiche un message d'erreur pour informer l'utilisateur.

Objectifs et Utilisation de la Fonction

La fonction `DailyButton1_Click` est une macro clé qui orchestre plusieurs tâches liées à la gestion quotidienne des inventaires :

- **Validation et Mise à Jour de la Date** : Assure que la date d'inventaire est correcte et mise à jour si nécessaire, garantissant que les données sont saisies avec la date appropriée.
- **Stockage des Données** : Transfère les données de la feuille *Daily inventory* vers une feuille d'enregistrement, ce qui permet de centraliser les données pour un suivi ultérieur.
- **Exportation et Envoi par Email** : Automatise le processus d'exportation des données en PDF et leur envoi par email, facilitant la distribution rapide des rapports d'inventaire.
- **Réinitialisation des Valeurs** : Nettoie la feuille *Daily inventory* après chaque opération pour la préparer à une nouvelle entrée, assurant ainsi la clarté et la précision des données futures.

En résumé, `DailyButton1_Click` est une macro intégrée qui coordonne plusieurs processus importants pour garantir que les données d'inventaire quotidien sont correctement validées, stockées, exportées et partagées. Cela améliore l'efficacité opérationnelle et assure une gestion efficace des inventaires au quotidien.

4.11.9 CheckQAndRetrieveEmailAndPost2

La macro `CheckQAndRetrieveEmailAndPost2` est conçue pour synchroniser des données entre deux feuilles de calcul dans un classeur Excel. Elle vérifie les noms dans une colonne spécifique d'une feuille de calcul et, en fonction de ces noms, récupère des informations supplémentaires (poste et email) depuis une autre feuille de calcul. Ces informations sont ensuite ajoutées à la feuille de calcul d'origine pour compléter les données disponibles. Voici une explication détaillée de chaque composant de la fonction :

Analyse des Composants de la Fonction

- **Déclaration des Variables**
  - `wsDest` : Référence à la feuille de calcul *Daily inventory*, où les informations doivent être mises à jour.

- `wsData` : Référence à la feuille de calcul *Data*, contenant les informations complémentaires (poste et email) à récupérer.
- `tbl` : Variable non utilisée dans cette version du code, initialement prévue pour les tables de données.
- `selectedName` : Variable pour stocker le nom sélectionné dans la feuille *Daily inventory*.
- `rngM` : Plage de cellules dans la colonne M de la feuille *Data*, où les noms sont recherchés.
- `rowIndex` : Index de la ligne trouvée dans la plage `rngM` qui correspond au nom sélectionné.
- `lastRow` : Numéro de la dernière ligne avec des données dans la colonne M de la feuille *Data*.
- `cell` : Cellule actuelle dans la colonne J de la feuille *Daily inventory* à traiter.
- `targetRow` : Numéro de la ligne dans la feuille *Daily inventory* où les informations doivent être mises à jour.
- `foundRow` : Numéro de la ligne trouvée dans la feuille *Data* qui correspond au nom sélectionné.

— **Gestion des Erreurs**
Active la gestion des erreurs, redirigeant l'exécution vers le gestionnaire d'erreurs en cas d'échec de la macro.

— **Définir les Feuilles de Calcul** Définit les feuilles de calcul source et destination : wsDest pour la feuille "Daily inventory", où les informations seront mises à jour, et wsData pour la feuille "Data", contenant les informations complémentaires.

— **Trouver la Dernière Ligne avec des Données**
Trouve la dernière ligne contenant des données dans la colonne M de la feuille "Data".

— **Définir la Plage de Recherche**
Définit la plage de recherche dans la colonne M de la feuille "Data", allant de la ligne 2 jusqu'à la dernière ligne contenant des données.

— **Boucle à Travers les Cellules de la Colonne J** Parcourt chaque cellule dans la colonne J, depuis la ligne 15 jusqu'à la dernière ligne contenant des données.

— **Vérifier et Rechercher les Noms Sélectionnés**
Obtient le nom de la cellule actuelle et vérifie s'il est non vide. Ensuite, recherche ce nom dans la plage définie et obtient l'index de la ligne correspondante, en vérifiant que la recherche a réussi.

— **Mettre à Jour les Informations** Obtient le numéro de ligne correspondant dans la feuille "Data", puis utilise cette information pour remplir les colonnes K et L de la feuille "Daily inventory" avec les données des colonnes Q et P de la feuille "Data", respectivement.

— **Gestion des Erreurs** En cas d'erreur, affiche un message détaillant le numéro et la description de l'erreur rencontrée.

Objectifs et Utilisation de la Fonction

La macro `CheckQAndRetrieveEmailAndPost2` est utilisée pour mettre à jour des informations spécifiques dans la feuille *Daily inventory* en récupérant des données associées (poste et email) depuis la feuille *Data*. Elle est particulièrement utile pour synchroniser les données lorsque des noms sont sélectionnés dans la feuille *Daily inventory* :

— **Recherche de Données** : La fonction permet de rechercher les noms dans une liste et de trouver les informations associées dans une autre feuille de calcul.

— **Mise à Jour Automatique** : Automatiser la mise à jour des postes et des adresses email pour chaque nom sélectionné, ce qui facilite le suivi et la gestion des informations.

— **Précision et Efficacité** : Réduit les erreurs manuelles et améliore l'efficacité en automatisant le processus de mise à jour des informations.

En résumé, `CheckQAndRetrieveEmailAndPost2` est une macro essentielle pour la gestion des informations dans des environnements où les données doivent être synchronisées entre plusieurs feuilles de calcul, garantissant ainsi l'exactitude et l'efficacité des mises à jour de données.

4.11.10 CreateDropDownListFromColumnM2

La macro `CreateDropDownListFromColumnM2` est conçue pour créer des listes déroulantes dans une feuille de calcul Excel en utilisant les valeurs d'une colonne spécifique d'une autre feuille. Voici une explication détaillée de la fonction, de ses composants et de son fonctionnement :

Analyse des Composants de la Fonction

— **Déclaration des Variables**
  — `wsData` : Référence à la feuille de calcul `"Data"`, qui contient les données source pour les listes déroulantes.
  — `wsDest` : Référence à la feuille de calcul `"Daily inventory"`, où les listes déroulantes seront ajoutées.
  — `lastRow` : Numéro de la dernière ligne avec des données dans la colonne M de la feuille `"Data"`.
  — `rngM` : Plage de cellules dans la colonne M de la feuille `"Data"`, de M2 jusqu'à la dernière ligne avec des données.
  — `listRangeAddress` : Adresse de la plage de données définie pour les listes déroulantes, au format `SheetName!RangeAddress`.
  — `cell` : Cellule actuelle dans la feuille `"Daily inventory"` où la validation des données sera appliquée.
— **Gestion des Erreurs**
  — `On Error GoTo ErrorHandler` : Active la gestion des erreurs pour afficher un message en cas d'échec de la macro.
  — `On Error Resume Next` : Ignore les erreurs pour permettre la suppression de la validation existante.
— **Définir les Feuilles de Calcul**
  — `Set wsData = ThisWorkbook.Sheets("Data")` : Définit la feuille de calcul contenant les données pour les listes déroulantes.
  — `Set wsDest = ThisWorkbook.Sheets("Daily inventory")` : Définit la feuille de calcul où les listes déroulantes seront créées.
— **Trouver la Dernière Ligne avec des Données**
  — `lastRow = wsData.Cells(wsData.Rows.Count, "M").End(xlUp).Row` : Détermine le numéro de la dernière ligne contenant des données dans la colonne M de la feuille `"Data"`.
— **Définir la Plage de Données**
  — `Set rngM = wsData.Range("M2:M" & lastRow)` : Définit la plage de cellules de M2 à la dernière ligne avec des données dans la colonne M.
— **Définir l'Adresse de la Plage de Données**
  — `listRangeAddress = wsData.Name & "!" & rngM.Address` : Crée l'adresse complète de la plage de données à utiliser pour les listes déroulantes.
— **Effacer les Validations Anciens**

  - `On Error Resume Next` : Ignore les erreurs potentielles lors de la suppression de validations existantes.
  - `wsDest.Range("J15:J20").Validation.Delete` : Supprime toute validation de données existante dans la plage J15 de la feuille `"Daily inventory"`.
- **Définir la Validation des Données**
  - `For Each cell In wsDest.Range("J15:J20")` : Parcourt chaque cellule dans la plage J15 pour appliquer la validation des données.
  - `With cell.Validation` : Configure la validation des données pour chaque cellule.
    - `.Delete` : Efface toute validation précédente dans la cellule.
    - `.Add Type:=xlValidateList` : Définit la validation des données comme une liste déroulante.
    - `Formula1:="=" & listRangeAddress` : Utilise l'adresse de la plage de données comme source pour la liste déroulante.
    - `.IgnoreBlank = True` : Permet des cellules vides.
    - `.InCellDropdown = True` : Affiche une liste déroulante dans la cellule.
    - `.ShowInput = True` : Affiche le message d'entrée.
    - `.ShowError = True` : Affiche les messages d'erreur en cas de saisie invalide.
- **Gestion des Erreurs**
  - `ErrorHandler:` : En cas d'erreur, affiche un message contenant le numéro et la description de l'erreur.

Objectifs et Utilisation de la Fonction

La macro `CreateDropDownListFromColumnM2` est utilisée pour créer des listes déroulantes dans une feuille de calcul Excel en utilisant les valeurs d'une colonne d'une autre feuille de calcul. Elle est particulièrement utile pour :

- **Automatisation de la Création de Listes Déroulantes** : Simplifie la gestion des listes déroulantes en automatisant leur création à partir d'une plage de données dynamique.
- **Mise à Jour Dynamique** : Permet aux utilisateurs de sélectionner des valeurs directement à partir d'une liste mise à jour, garantissant que les choix sont toujours basés sur les données les plus récentes.
- **Réduction des Erreurs de Saisie** : En offrant une liste déroulante, elle réduit les erreurs de saisie en limitant les choix disponibles aux valeurs valides.

En résumé, `CreateDropDownListFromColumnM2` est une macro essentielle pour améliorer l'efficacité de la gestion des données dans Excel en facilitant la création et la mise à jour des listes déroulantes basées sur des données dynamiques.

Deuxième partie

Développement de l'Application Mobile avec PowerApps

Chapitre 5

Conception de l'application mobile

5.1 Introduction

La partie PowerApps de ce projet représente une adaptation fidèle de la solution initialement développée dans BASSA.xlsm, cette fois-ci optimisée pour une application mobile. En tirant parti de la plateforme low-code PowerApps, nous avons pu recréer les fonctionnalités essentielles tout en les adaptant à un environnement mobile, rendant l'application plus accessible et intuitive pour les utilisateurs en déplacement. Plutôt que de plonger dans les détails techniques complexes, cette approche nous permet de nous concentrer sur une présentation claire et structurée de chaque page du projet, en mettant en avant les fonctionnalités spécifiques et les processus qu'elles soutiennent. Chaque page de l'application a été conçue pour refléter les flux de travail et les opérations de la version Excel, tout en bénéficiant des avantages offerts par une interface mobile moderne et réactive. Cette transition vers une application mobile vise à maintenir une expérience utilisateur cohérente tout en exploitant pleinement les capacités de PowerApps pour simplifier le développement et la maintenance.

5.2 Page Login

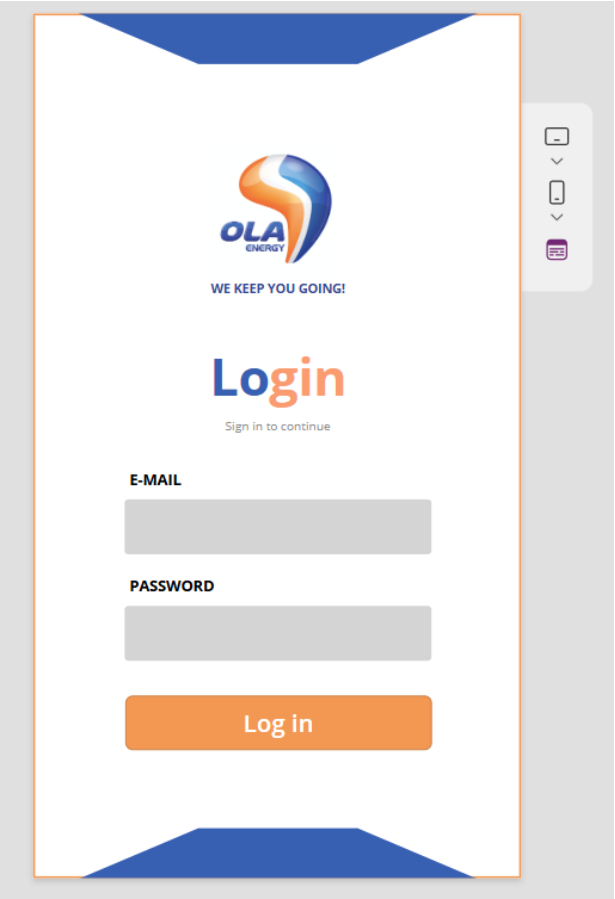

FIGURE 5.1 – Login

5.2.1 Déscription

Cette page représente l'écran de connexion de l'application mobile OLA Energy, conçu pour permettre aux utilisateurs d'accéder de manière sécurisée à l'application. L'utilisateur est invité à entrer son adresse e-mail et son mot de passe pour se connecter. La gestion des utilisateurs est assurée par un système d'authentification robuste, garantissant que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités et données de l'application. Cette interface, simple et intuitive, reflète l'engagement d'OLA Energy à fournir une expérience utilisateur fluide et sécurisée.

5.2.2 Éxplication du code

Le code présente deux parties distinctes, chacune exécutée dans des contextes différents : l'une lors de l'affichage de l'écran (*OnVisible*), et l'autre lors de la soumission d'un formulaire.

Partie OnVisible Lorsqu'un écran devient visible, le code exécute deux actions simultanées grâce à la fonction `Concurrent`. Ces actions consistent à réinitialiser les champs de saisie pour l'adresse e-mail et le mot de passe. La réinitialisation des champs est importante pour garantir que les utilisateurs

commencent avec des champs vides chaque fois qu'ils accèdent à l'écran de connexion, évitant ainsi l'affichage de données précédentes ou erronées.

Partie Soumission du Formulaire Lors de la soumission du formulaire, le code commence par vérifier si l'adresse e-mail ou le mot de passe sont laissés vides. Si l'un de ces champs est vide, une notification d'erreur est affichée, indiquant que l'adresse e-mail ou le mot de passe ne peut pas être laissé en blanc. Cela assure que les utilisateurs fournissent toutes les informations nécessaires avant de tenter de se connecter.
Si les deux champs contiennent des valeurs, le code vérifie ensuite si les informations saisies correspondent à un enregistrement valide dans la base de données des utilisateurs (*USERS*). Si une correspondance est trouvée, l'utilisateur est redirigé vers un autre écran appelé *MENU*, avec une transition visuelle en fondu. Cette étape marque une connexion réussie. Si aucune correspondance n'est trouvée, une notification d'erreur informe l'utilisateur que l'adresse e-mail ou le mot de passe est incorrect, demandant ainsi une nouvelle tentative de connexion.

En résumé, cette logique gère les interactions des utilisateurs avec l'écran de connexion en réinitialisant les champs de saisie à l'affichage de l'écran et en validant les informations lors de la soumission du formulaire, tout en fournissant des retours d'information appropriés pour une meilleure expérience utilisateur.

5.3 Menu

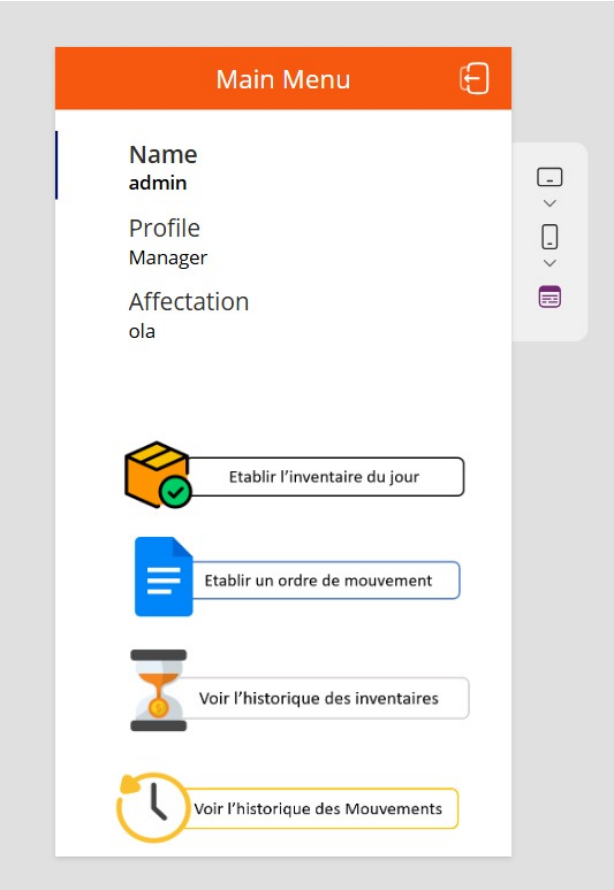

FIGURE 5.2 – Menu

5.3.1 Déscription

L'écran `Menu` ajuste son contenu en fonction du rôle de l'utilisateur connecté. Lors de l'authentification, les informations personnelles de l'utilisateur, telles que le nom et l'affectation, sont automatiquement récupérées et affichées. Selon le rôle de l'utilisateur (comme magasinier, réconciliation, ventes, responsable ou administrateur), l'écran `Menu` propose différentes options et fonctionnalités. Par exemple :

- **Magasinier** : Accès aux outils de gestion des stocks.
- **Réconciliation** : Outils pour vérifier et ajuster les stocks.
- **Ventes** : Fonctions liées aux transactions de vente.
- **Responsable** : Vue étendue avec des options de gestion avancées.
- **Administrateur** : Accès complet à toutes les fonctionnalités et paramètres de l'application.

Cette personnalisation assure que l'utilisateur voit uniquement les options pertinentes à ses responsabilités et facilite une navigation efficace dans l'application.

5.3.2 Éxplication du code

Ce code est utilisé pour gérer l'accès aux fonctionnalités d'une application en fonction du rôle de l'utilisateur et pour afficher certaines informations de l'utilisateur à l'écran.

Gestion des Accès en Fonction du Rôle de l'Utilisateur Lorsque l'écran devient visible ou lorsqu'un certain événement se produit, le code commence par récupérer les informations de l'utilisateur à partir de la base de données en utilisant l'adresse e-mail et le mot de passe fournis. Ces informations sont stockées dans une variable appelée `UserRecord`.
Ensuite, le code vérifie le rôle ou le profil de l'utilisateur en consultant le champ `Profile` dans la base de données. En fonction du rôle détecté, le code configure différents paramètres d'affichage pour contrôler l'accès aux diverses fonctionnalités de l'application :

- Si l'utilisateur est un *Storekeeper* (magasinier), les paramètres associés à l'affichage des informations d'inventaire quotidien et des mouvements de stock sont activés.
- Si l'utilisateur est en *Reconciliation* (réconciliation) ou en *Sales* (ventes), l'accès est accordé aux données d'inventaire, aux mouvements de données, et aux fonctionnalités de réconciliation.
- Pour les rôles de *Manager* (responsable) et *Administrator* (administrateur), tous les paramètres d'accès sont activés, permettant une visibilité complète sur toutes les fonctionnalités de l'application.
- Si le profil de l'utilisateur ne correspond à aucun des rôles définis, tous les paramètres sont désactivés, garantissant que l'utilisateur n'a pas accès à des fonctionnalités non pertinentes pour son rôle.

Affichage des Informations de l'Utilisateur Après avoir configuré les paramètres d'accès, le code crée une table affichant les informations de l'utilisateur. Cette table inclut le nom, le profil, et l'affectation de l'utilisateur, offrant une vue claire et structurée des données pertinentes à l'écran.

En résumé, ce code permet de personnaliser l'accès aux fonctionnalités de l'application en fonction du rôle de l'utilisateur et de fournir des informations sur l'utilisateur de manière organisée. Cela assure que chaque utilisateur a accès aux fonctionnalités appropriées selon son profil, tout en affichant clairement ses informations personnelles.

5.4 Gestion des mouvements

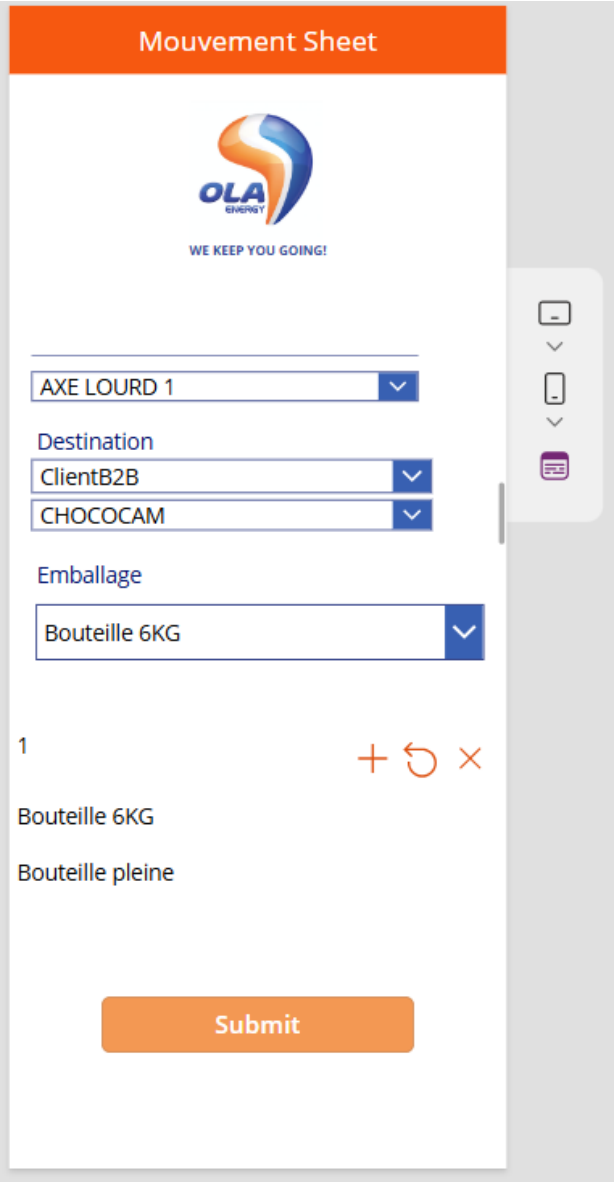

FIGURE 5.3 – formulaire des mouvements

5.4.1 Déscription

Cette version de la page "Mouvement Sheet" intègre des fonctionnalités dynamiques pour faciliter la gestion des mouvements de produits :

— **Sélection Dynamique des Sources et Destinations :** Les menus déroulants pour "Source" (ex : "AXE LOURD 1") et "Destination" (ex : "ClientB2B", "CHOCOCAM") changent dynamiquement en fonction de la catégorie sélectionnée. Cela permet une adaptation automatique des options disponibles en fonction du contexte, améliorant ainsi la précision des mouvements enregistrés.

— **Choix de l'Article et de l'Emballage :** Les options pour l'article (ex : "Bouteille 6KG") et son emballage sont également dynamiques. Les utilisateurs peuvent sélectionner l'article et l'emballage approprié, ce qui permet une flexibilité et une personnalisation accrues du mouvement.
— **Gestion des Articles et des Emballages :** Les utilisateurs peuvent ajouter plusieurs articles et emballages à la liste avant de soumettre le mouvement. Chaque élément peut être ajouté avec sa quantité spécifique, et il est possible de modifier ou de supprimer des articles avant de finaliser l'enregistrement. Les boutons "+" (ajouter), "modifier", et "x" (supprimer) permettent une gestion intuitive de cette liste.
— **Soumission du Mouvement :** Une fois que toutes les informations sont saisies et vérifiées, l'utilisateur peut cliquer sur "Submit" pour enregistrer le mouvement dans la base de données. Cette fonctionnalité assure que toutes les données sont complètes et exactes avant la soumission.
Cette interface améliorée rend le suivi des mouvements de produits plus flexible et adaptable, en tenant compte des différentes catégories de sources, de destinations, d'articles, et d'emballages, tout en offrant la possibilité d'ajuster les entrées avant la soumission finale.

5.4.2 Éxplication du code

Ce code regroupe trois opérations clés dans la gestion des données d'une application : la mise à jour des enregistrements, l'ajout de nouveaux éléments à une collection, et la suppression du dernier enregistrement ajouté.

Mise à Jour des Enregistrements La première partie du code est conçue pour mettre à jour la source de données `Mouvement` avec les informations provenant de la collection `Products`. Tout d'abord, le code vérifie si la collection contient des enregistrements. Si elle en contient, il utilise une boucle pour traiter chaque enregistrement individuellement. Pour chaque enregistrement, les informations telles que le numéro, l'emballage, l'article, le commentaire, la marque, et plusieurs autres détails (comme le type de mouvement, la référence du mouvement, et la date du mouvement) sont envoyées à la source de données `Mouvement`. Cette opération est effectuée en utilisant la fonction `Patch`, qui permet de mettre à jour ou de créer des enregistrements dans la source de données. Une fois toutes les mises à jour réalisées, la source de données `Mouvement` est rafraîchie pour refléter les modifications apportées.

Ajout de Nouveaux Enregistrements La deuxième partie du code gère l'ajout de nouveaux enregistrements à la collection `Products`. Lorsqu'un nouvel enregistrement est ajouté, le code attribue un numéro unique en fonction du nombre actuel d'enregistrements dans la collection, en ajoutant un à ce total pour garantir l'unicité de chaque numéro. Les autres informations ajoutées incluent l'emballage, la quantité, l'article, le commentaire, et la marque, qui sont toutes saisies par l'utilisateur via l'interface de l'application. Cette opération permet de compléter et de gérer la collection des produits en fonction des données fournies.

Suppression du Dernier Enregistrement La troisième partie du code permet de supprimer le dernier enregistrement ajouté à la collection `Products`. Avant d'effectuer la suppression, le code vérifie que la collection contient au moins un enregistrement pour éviter les erreurs potentielles. Si la condition est remplie, le code supprime le dernier élément de la collection. Après la suppression, une

notification est affichée pour informer l'utilisateur que le dernier enregistrement a été supprimé avec succès. Cette fonctionnalité assure que les utilisateurs peuvent retirer facilement les enregistrements indésirables et reçoivent un retour d'information approprié sur l'action effectuée.

supression des articles La fonction `Clear(Mouvement)` serait utilisée pour effacer tous les mouvements enregistrés temporairement, par exemple après avoir terminé un processus de saisie ou de validation de mouvements de stock. Cela garantit que la collection est propre pour la prochaine utilisation, évitant tout risque de confusion avec des enregistrements précédents.

Les affectations des utilisateurs sont enregistrées dans la base de données après l'authentification. Ces informations, telles que le rôle et l'affectation de chaque utilisateur (magasinier, responsable, administrateur, etc.), sont automatiquement récupérées et utilisées pour personnaliser l'interface et les fonctionnalités accessibles dans l'application. Cela garantit que chaque utilisateur voit uniquement les options pertinentes à son rôle, tout en centralisant et sécurisant la gestion des affectations dans la base de données.

En résumé, ce code facilite la gestion dynamique des données en permettant de mettre à jour une source de données, d'ajouter de nouveaux enregistrements à une collection, et de supprimer des éléments, tout en fournissant des retours d'information clairs à l'utilisateur pour une meilleure expérience de gestion des données.

5.5 Gestion de l'inventaire journalière

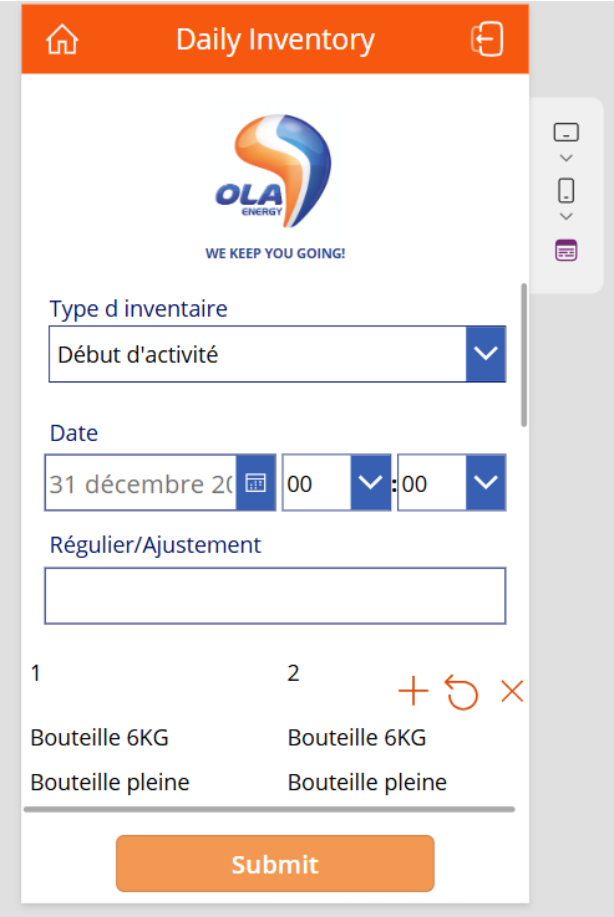

FIGURE 5.4 – formulaire des mouvements

5.5.1 Déscription

Cette version de la page "Daily Inventory" dans Power Apps présente une interface améliorée pour la gestion des inventaires quotidiens, avec des fonctionnalités facilitant l'ajout et la gestion des articles :

- **Type d'Inventaire Dynamique :** Un menu déroulant permet à l'utilisateur de sélectionner le type d'inventaire, comme "Début d'activité". Cette fonctionnalité aide à classifier les mouvements d'inventaire selon le moment ou le type d'opération, facilitant la gestion des différentes catégories.
- **Date et Heure Sélectionnables :** L'interface inclut un sélecteur de date et d'heure personnalisable. Les utilisateurs peuvent choisir la date ainsi que l'heure précise pour enregistrer les mouvements, assurant une précision temporelle dans la documentation des activités d'inventaire.
- **Gestion des Articles :** Les utilisateurs peuvent ajouter plusieurs articles (par exemple, "Bouteille 6KG") avec des quantités spécifiques à la liste des articles inventoriés. Chaque ligne correspond à un article ou un emballage, et plusieurs options sont disponibles pour gérer cette liste.

— **Boutons d'Action Intuitifs :** Les boutons "+" pour ajouter un nouvel article, et "x" pour supprimer un article facilitent la manipulation de la liste avant de finaliser l'enregistrement.
— **Soumission du Mouvement :** Une fois que toutes les informations sont correctement saisies, un bouton "Submit" permet à l'utilisateur de soumettre l'inventaire finalisé. Cela garantit que les données sont bien enregistrées dans la base de données.

Cette interface offre une flexibilité accrue pour gérer les inventaires quotidiens en permettant des ajustements rapides et un suivi précis des stocks, tout en assurant une navigation claire et des actions intuitives pour les utilisateurs.

5.5.2 Éxplication du code

Ce code est composé de trois parties distinctes, chacune gérant une opération différente dans l'application : la mise à jour des données dans une source, l'ajout d'enregistrements à une collection, et la suppression du dernier enregistrement ajouté.

Mise à Jour des Données La première partie du code est responsable de la mise à jour d'une source de données appelée `Inventory`. Elle commence par vérifier si la collection `Items` contient des enregistrements. Si cette condition est remplie, le code utilise une boucle pour parcourir chaque enregistrement dans `Items` et effectuer une opération de mise à jour dans `Inventory`. Pour chaque enregistrement, des informations telles que la date, l'emballage, l'article, la marque, la quantité, le type d'inventaire, le type d'ajustement, et l'expéditeur sont envoyées à la source de données. Après avoir appliqué ces mises à jour, la source de données `Inventory` est rafraîchie pour refléter les modifications.

Ajout d'Enregistrements à la Collection La deuxième partie du code est utilisée pour ajouter un nouvel enregistrement à la collection `Items`. Lors de l'ajout, le code attribue un numéro unique à chaque nouvel enregistrement en comptant les enregistrements déjà présents dans `Items` et en ajoutant un à ce total. Ensuite, il collecte des informations fournies par l'utilisateur via l'interface, telles que l'emballage, la quantité, l'article, et la marque, et les ajoute à la collection sous forme de nouvel enregistrement.

Suppression du Dernier Enregistrement La troisième partie du code gère la suppression du dernier enregistrement ajouté à la collection `Items`. Elle commence par vérifier si la collection contient au moins un enregistrement. Si c'est le cas, le code supprime le dernier élément de la collection. Une fois l'enregistrement supprimé, une notification est affichée à l'utilisateur pour confirmer que la suppression a été effectuée avec succès. Cette fonctionnalité assure que les utilisateurs peuvent retirer facilement les enregistrements indésirables tout en recevant un retour d'information sur l'action réalisée.

supression des articles De la même manière, `Clear(Inventory)` permettrait de réinitialiser la liste des stocks temporairement enregistrés dans l'application. Cela peut être utile après une vérification d'inventaire ou une mise à jour des quantités de stock, garantissant que l'utilisateur peut recommencer avec une liste d'inventaire vide lors de la prochaine opération.

Les affectations des utilisateurs sont enregistrées dans la base de données après l'authentification. Ces informations, telles que le rôle et l'affectation de chaque utilisateur

(magasinier, responsable, administrateur, etc.), sont automatiquement récupérées et utilisées pour personnaliser l'interface et les fonctionnalités accessibles dans l'application. Cela garantit que chaque utilisateur voit uniquement les options pertinentes à son rôle, tout en centralisant et sécurisant la gestion des affectations dans la base de données.

En résumé, ce code permet de gérer les opérations essentielles de mise à jour des données, d'ajout d'enregistrements, et de suppression d'éléments au sein de l'application, facilitant ainsi la gestion dynamique des données et améliorant l'expérience utilisateur.

5.6 Les boutons

5.6.1 Bouton "Home"

Le code `Navigate(MENU; ScreenTransition.Fade)` est utilisé dans une application développée sous PowerApps pour gérer la navigation d'une page à une autre. Ici, lorsqu'un utilisateur clique sur le bouton "Home", l'application le redirige vers l'écran appelé "MENU". L'instruction `ScreenTransition.Fade` précise que la transition entre les deux écrans doit se faire avec un effet de fondu. Cet effet permet d'améliorer l'expérience utilisateur en rendant la navigation plus fluide et visuellement agréable.

5.6.2 Bouton "Logout"

Le code `Navigate(Authentification; ScreenTransition.Fade)` fonctionne de manière similaire, mais dans ce cas, il redirige l'utilisateur vers l'écran d'authentification, nommé "Authentification". Cela est souvent utilisé lors de la déconnexion ou du changement d'utilisateur. Comme pour le bouton "Home", la transition entre les deux écrans se fait avec un effet de fondu, assurant une navigation cohérente et fluide à travers l'application.

Ces deux blocs de code montrent comment gérer la navigation entre les différentes interfaces d'une application en utilisant PowerApps, tout en offrant une expérience visuelle agréable à l'utilisateur grâce à l'effet de transition "Fade".

Conclusion et Perspectives

En conclusion, ce projet de gestion d'inventaire et de stock au sein d'OLA Energy a permis de mettre en œuvre deux applications interconnectées avec succès. Cependant, nous avons rencontré plusieurs défis techniques, notamment avec Visual Basic. L'utilisation de la base de données intégrée à Visual Basic s'est révélée limitée en termes de performance et de gestion des données à grande échelle. Il aurait été plus avantageux d'utiliser un système de gestion de base de données (SGBD) tel que MySQL, qui offre une meilleure robustesse, évolutivité, et gestion des transactions complexes. De plus, Visual Basic a montré des limites dans la gestion des listes déroulantes avec un grand volume de données, rendant difficile l'optimisation de l'expérience utilisateur lorsque de nombreux éléments étaient impliqués. Cela a également posé des contraintes dans différents programmes du projet, limitant notre flexibilité de développement.

En ce qui concerne PowerApps, bien que cette plateforme ait permis une intégration rapide sur mobile, elle présente des restrictions importantes, notamment dans la gestion du code. La limitation du nombre de lignes de code qu'il est possible d'écrire par fonctionnalité a réduit la complexité des traitements que nous pouvions implémenter. De plus, il était impossible de regrouper plusieurs traitements dans une même fonctionnalité, ce qui a fragmenté certaines parties du développement et a nécessité des solutions alternatives peu optimales.

Par manque de temps, nous n'avons pas pu entamer la troisième phase du projet, qui consistait à intégrer Power BI pour la visualisation des données. Cette étape aurait permis de fournir aux gestionnaires une vue d'ensemble des performances en temps réel à travers des tableaux de bord dynamiques, améliorant ainsi la prise de décision. Ce projet, bien que réussi dans sa globalité, met en lumière les limites techniques et les choix de plateformes qui ont influencé la complexité et l'efficacité du développement.

Toutefois, nous restons pleinement disponibles pour tout futur projet ou amélioration possible, notamment pour perfectionner les solutions existantes et intégrer de nouvelles fonctionnalités en réponse aux besoins de l'entreprise.

Bibliographie

[Dophis, 2024] Dophis (2024). *Tutoriels VBA Excel*. Dophis.

[Energy, 2024] Energy, O. (2024). *About Us - Corporate Profile*. OLA Energy.

[Microsoft, 2024] Microsoft (2024). *Présentation de Power Apps - Vue d'ensemble*. Microsoft.