НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп'ютерні системи та мережі”
спеціальності 123 “Комп’ютера інженерія”

на тему: Система автоматизації складського обліку

Виконав : студент __4__ курсу, групи _ІО-91_
(шифр групи)

_____Коноплін Ілля Володимирович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник ____доцент, к.т.н., Долголенко О.М_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) с.в., Виноградов Ю. М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп'ютерні системи та мережі”

спеціальності 123 “Комп’ютера інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Конопліна Іллі Володимировича

1. Тема проєкту Система автоматизації складського обліку
керівник проєкту Долголенко Олександр Миколайович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2101-с

2. Термін здачі студентом закінченого проєкту *8 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Історія розвитку, сучасний стан та актуальність системи автоматизації складського обліку.

Розділ 2. Вибір технологій та середовища розробки.

Розділ 3. Розробка та проеткування системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) взаємодія компонентів системи(структурна схема), діаграма класів(функціональна схема), алгоритм дій програмного забезпечення(принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «9» лютого 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*04.02.2023-9.02.2023*	
2.	*Вивчення та аналіз завдання*	*09.02.2023-10.04.2023*	
3.	*Розробка архітектури та загальної структури системи*	*10.04.2023-25.04.2023*	
4.	*Розробка структур окремих підсистем*	*25.04.2023-05.04.2023*	
5.	*Програмна реалізація системи*	*5.04.2023-15.04.2023*	
6.	*Оформлення пояснювальної записки*	*15.04.2023-20.05.2023*	
7.	*Захист програмного продукту*	*25.04.2023*	
8.	*Передзахист*	*05.06.2023*	
9.	*Захист*	*19.06.2023*	

Студент-дипломник _____ Ілля КОНОПЛІН
(підпис)

Керівник проєкту _____ Олександр ДОЛГОЛЕНКО
(підпис)

АНОТАЦІЯ

Дипломна робота присвячена вивченню та дослідженню системи управління складом Odoo WMS (Warehouse Management System). В роботі детально розглядаються основні принципи роботи цієї системи, її функціональні можливості та переваги. В роботі було розроблено власний практичний приклад використання Odoo WMS для оптимізації процесу управління складом. Результати дослідження та розробки демонструють ефективність та користь використання системи Odoo WMS у пракичних ситуаціях.

ANNOTATION

The thesis is devoted to the study and research of the Odoo WMS (Warehouse Management System) warehouse management system. The paper discusses in detail the basic principles of this system, its functionality and advantages. The paper developed its own practical example of using Odoo WMS to optimise the warehouse management process. The results of the research and development demonstrate the effectiveness and benefits of using the Odoo WMS system in practical situations.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система автоматизації	3		
			складського обліку			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система автоматизації	70		
			складського обліку			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система автоматизації	1		
			складського обліку			
			Взаємодія компонентів			
			системи(структурна схема)			
	А4	***ІАЛЦ.4672008.005 Д2***	Система автоматизації	1		
			складського обліку			
			Діаграма класів			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Система автоматизації	1		
			складського обліку			
			Алгоритм дій програмного			
			забезпечення			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Система автоматизації	15		
			складського обліку			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА	
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*		
Розроб		Коноплін І.В.			*Система автоматизації складського обілку* Опис альбому	Літ. / Аркуш: 1 / Аркушів: 1
Перев		Долголенко О.М				***КПІ ім. Ігоря Сікорського ФІОТ ІО-91***

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система автоматизації складського обліку»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи автоматизації складського обілку

Областю застосування цієї системи є проектування систем скалдського обліку в різних галузях бізнесу

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», яке було затвердженно факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи автоматизації складського обілку.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Оптимизація та автоматизація процесів складського обілку
- Надати можливість користувачам власноруч налаштовувати систему.

- Надати безпеку даних
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- PyCharm 2020 IDE версії або вище.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i5-3470.
- ROM не менше ніж 128 ГБ.
- RAM не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	05.02.2023-10.02.2023
Вивчення та аналіз завдання	10.02.2023-15.03.2023
Розробка архітектури та загальної структури системи	15.03.2023-25.03.2023
Розробка структур окремих частин системи	25.03.2023-5.04.2023
Програмна реалізація системи	5.04.2023-15.04.2023
Виправлення помилок	15.04.2023-20.05.2023
Оформлення пояснювальної записки	15.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система автоматизації складського обілку*»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

WMS	(warehouse management system) Система управління складом
ТСД	Термінал збору даних

Вступ

У сучасному бізнес-середовищі ефективне управління складськими процесами є важливою складовою успішної діяльності підприємств. Збільшення обсягів товарів, постійні зміни вимог споживачів та рост конкуренції ставлять перед підприємствами складського господарства значні виклики. Автоматизація складського обліку є одним із ключових факторів, що допомагає оптимізувати та забезпечити ефективність складських процесів.

Метою даної дипломної роботи є дослідження та розробка системи автоматизації складського обліку на платформі Odoo. Odoo, як інтегрована ERP-платформа, надає широкий функціонал для керування бізнес-процесами, включаючи складський облік. Вибір Odoo як базової платформи для розробки системи дозволяє поєднати функціональні можливості з гнучкістю налаштування та розширення, що відповідає потребам сучасних підприємств.

У ході роботи будуть виконані наступні завдання: огляд існуючих систем та алгоритмів автоматизації складського обліку, вибір найкращих рішень для розробки системи на основі Odoo, детальне вивчення технологій та інструментів для розробки системи, розробка власних алгоритмів та програмних компонентів, а також впровадження та налагодження системи на реальному підприємстві.

Результатом дипломної роботи буде розроблена система автоматизації складського обліку на платформі Odoo, яка забезпечуватиме ефективність та точність управління запасами, замовленнями, отриманням та відправленням товарів. Дана система сприятиме оптимізації роботи складського відділу та покращенню загальної продуктивності підприємства.

Далі у роботі будуть розглянуті теоретичні основи автоматизації складського обліку, аналіз сучасних підходів та розробка практичної реалізації системи на основі платформи Odoo.

РОЗДІЛ 1
ІСТОРІЯ РОЗВИТКУ, СУЧАСНИЙ СТАН ТА АКТУАЛЬНІСТЬ СИСТЕМИ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ

1.1 Основні поняття

1.1.1 ERP

ERP (Enterprise Resource Planning) - це інтегрована система управління бізнес-процесами, що дозволяє будь-якій організації автоматизувати та оптимізувати різні функціональні області, такі як фінанси, ресурси підприємства, виробництво, продажі, логістика, управління проектами та інші.[1]

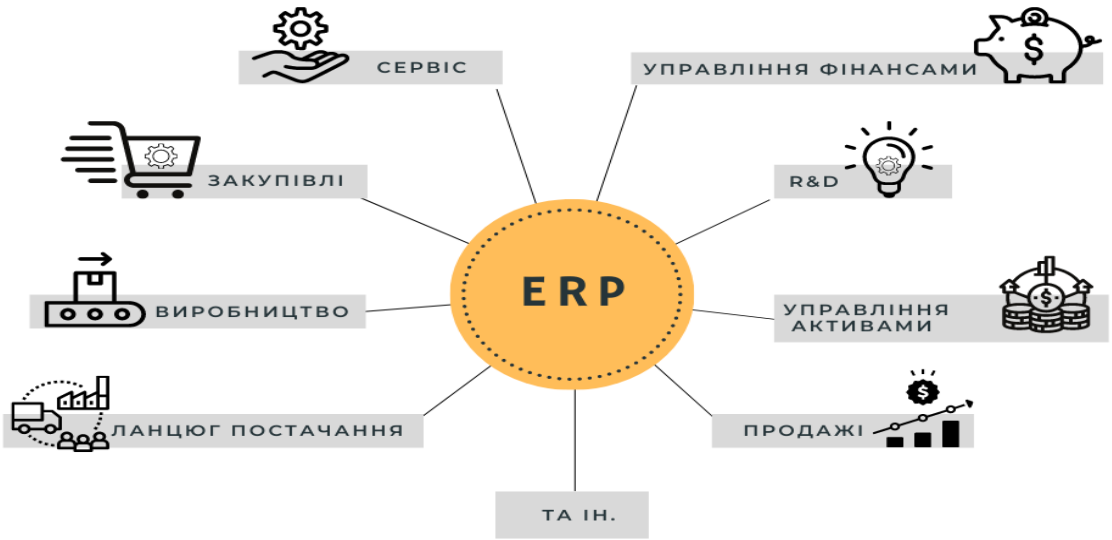


Рисунок 1.1 – Приклад функціональних областей

Перші ERP системи з'явилися у 1960-х роках, коли комп'ютеризація бухгалтерського обліку почала займати перші позиції. Вони включали основні модулі для фінансового обліку, збуту та виробництва, і працювали на основі мейнфреймів.

У 1990-х роках зростання комп'ютерних технологій та популярність клієнт-серверної архітектури призвели до збільшення можливостей ERP систем. Вони стали більш гнучкими, масштабованими та інтегрованими. З'явилися нові модулі для управління взаємовідносинами з клієнтами (CRM), управління ланцюгом постачання (SCM) та інші.

З появою Інтернету та технологій хмарного обчислення в 2000-х роках, ERP системи стали доступнішими та зручнішими для розгортання та використання. Компанії могли використовувати ERP системи в якості послуги SaaS, що дозволило знизити витрати на апаратне забезпечення та інфраструктуру.

Сьогодні ERP системи стали невід'ємною частиною багатьох організацій, незалежно від їх розміру та галузі. Вони пропонують повний спектр функцій для управління фінансами, ресурсами, збутом, виробництвом, логістикою та іншими бізнес-процесами.

ERP система містить в собі базу даних, яка інтегрує різні бізнес-функції та забезпечує обмін даними між ними. ERP системи дозволяють підприємствам працювати на основі єдиного інформаційного простору, що забезпечує більш ефективний і точний аналіз даних, зменшення часу на обробку даних та підвищення якості прийняття рішень.

Основний принцип ERP – централізований збір інформації. Всі дані знаходяться в єдиному сховищі, що дає можливість всім користувачам – від генерального директора до працівників бухгалтерії – зберігати, створювати та використовувати єдині актуальні дані підприємства[3]. Тобто, якщо зараз

компанія користується окремими базами даних для моніторингу складської звітності, обробки замовлень та бухгалтерського обліку – використовуючи при цьому велику кількість ненадійних електронних таблиць, то ERP об’єднує ці процеси в єдиний та зрозумілий потік інформації.

ERP система складається з різних модулів, які охоплюють різні аспекти бізнес-процесів. Наприклад, модуль фінансів включає у себе бухгалтерію, управління кредитами та дебетами, фінансовий аналіз та бюджетування. Модуль продажів охоплює управління продажами, контактами з клієнтами, управління запасами та інші аспекти.

Однією з головних переваг ERP систем є їх масштабованість та гнучкість. Підприємства можуть інтигрувати лише ті модулі, які їм необхідні, і додавати нові модулі у майбутньому. ERP системи також дозволяють інтегрувати різні інформаційні системи, що дозволяє забезпечувати більш широкий обмін даними та співпрацю між різними підрозділами підприємства.[3]

ERP системи є важливим інструментом для підтримки різних видів бізнесів, від маленьких компаній до великих корпорацій. Вони дозволяють автоматизувати та оптимізувати бізнес-процеси, знизити витрати та збільшити продуктивність.

ERP системи дозволяють підприємствам забезпечити більш точний прогнозування витрат та прибутку, зменшити витрати на запаси та оптимізувати їх управління, а також забезпечити кращу координацію між різними підрозділами та зменшити час на прийняття рішень.

Окрім цього, ERP системи дозволяють забезпечити більш точний та швидкий доступ до даних, що полегшує процес прийняття рішень. Вони дозволяють автоматизувати більшість бізнес-процесів, що зменшує ймовірність помилок та збільшує продуктивність працівників.

У зв'язку зі зростаючою кількістю даних, які генеруються та обробляються в компаніях, ERP системи стають все більш важливими для ефективного управління бізнесом. Вони дозволяють забезпечити більш точний

та швидкий доступ до даних, що полегшує процес прийняття рішень та забезпечує більшу конкурентоспроможність підприємств.

1.1.2 WMS системи

WMS (Warehouse Management System) - це програмне забезпечення, призначене для оптимізації та автоматизації процесів управління складом. WMS дозволяє контролювати та відслідковувати всі операції на складі, від приймання та розміщення товарів до підготовки до відвантаження та відправки замовлень[2]. Впровадження WMS системи підвищує ефективність роботи складу, точність усіх процесів, а також підвищує утилізацію обсягу на 10-30%.

Розвиток систем управління складом (WMS) відбувався протягом багатьох років і включав в себе важливі етапи. Перші системи управління складом були простими та ручними, але з появою комп'ютерних технологій і прогресу автоматизації, WMS системи поступово еволюціонували.[15]

З ростом складських операцій та комплексності бізнес-процесів, WMS системи стали більш функціональними та інтегрованими. Розширення функціональності включало управління запасами, контроль за поставками, оптимізацію розміщення товарів, а також прогнозування попиту.

Згодом, інтеграція WMS систем з ERP системами стала ключовим фактором розвитку. Це дозволило автоматизувати та координувати більшість ділових процесів в підприємстві, включаючи фінанси, виробництво, управління запасами та логістику.

Застосування технологій IoT (Internet of Things) внесло нові можливості до WMS систем. Використання датчиків та RFID міток дозволяє відстежувати рух товарів, контролювати умови зберігання та автоматизувати ряд складських операцій.[4]

Останнім трендом в розвитку WMS систем є впровадження штучного інтелекту та машинного навчання. Алгоритми машинного навчання дозволяють

прогнозувати попит, оптимізувати розміщення товарів та приймати ефективні рішення управління запасами.

Узагальнюючи, розвиток WMS систем відбувався від простих ручних процесів до комплексних інтегрованих платформ, що використовують сучасні технології для автоматизації та оптимізації складського обліку та управління запасами.[12]

Рисунок 1.2 - Приклад процесів управління складом

WMS зазвичай включає такі функції, як:

- Моніторинг запасів: WMS дозволяє в режимі реального часу відслідковувати кількість товарів на складі та їх розміщення.
- Планування та оптимізація місць зберігання: WMS допомагає планувати оптимальні місця для зберігання товарів на складі, що дозволяє ефективніше використовувати простір та зменшувати час на пошук товарів.

- Керування замовленнями: WMS автоматизує процеси підготовки до відвантаження та відправки замовлень, забезпечуючи точність та швидкість виконання.
- Інвентаризація та аудит: WMS дозволяє вести точний облік запасів та здійснювати їх регулярну інвентаризацію.
- Персонал: Модуль дає змогу менеджерам контролювати показники кожного співробітника на складі, визначати ефективність їхньої роботи.

1.1.3 ТСД

Термінали збору даних (ТСД) в контексті систем управління складом відіграють важливу роль у поліпшенні ефективності та точності операцій на складі.

ТСД - це портативні пристрої, оснащені сканерами штрих-кодів, клавіатурою та екраном, які дозволяють операторам збирати, обробляти та передавати дані на місці проведення складських операцій. Вони забезпечують безпосереднє зв'язування між фізичними товарами на складі та обліковими записами в системі WMS.[19]

За допомогою ТСД оператори можуть швидко та точно сканувати штрих-коди товарів, збирати дані про кількість та розміщення товарів, виконувати переміщення та інвентаризацію запасів, оформляти поставки та відвантаження, а також здійснювати інші складські операції. Вся ця інформація негайно передається до системи WMS, що дозволяє оновлювати статуси операцій у реальному часі.

Використання ТСД у контексті WMS допомагає покращити продуктивність та точність складських процесів. Вони дозволяють уникнути помилок, пов'язаних з ручним введенням даних, та забезпечують швидкий збір та обробку інформації. Операторам необхідно лише сканувати штрих-коди, а

решта процесу автоматизована, що знижує ризик помилок та збільшує продуктивність роботи.

Також ТСД можуть мати додаткові функціональні можливості, такі як бездротове підключення, підтримка голосових команд, можливість працювати в умовах низького освітлення або екстремальних температур. Все це дозволяє використовувати ТСД в різних умовах та оптимізувати роботу на складі.

Загалом, використання ТСД у контексті WMS є важливим елементом для підвищення ефективності, точності та автоматизації складських операцій. Вони дозволяють операторам збирати дані у реальному часі, оновлювати інформацію у системі WMS та забезпечують ефективну взаємодію між фізичними товарами на складі та цифровими записами у системі управління складом.

1.2 Вибір Odoo як WMS системи

1.2.1 Odoo

Odoo - це відкрите програмне забезпечення для управління бізнес-процесами, яке надає цілісний підхід до управління бізнесом. Odoo містить набір модулів, які покривають більшість функцій, що необхідні для ефективного управління бізнесом, включаючи керування складом, продажі, закупівлі, фінанси, проектний менеджмент, виробництво та інші.[5]

Одним з ключових переваг Odoo є те, що він є повністю відкритим програмним забезпеченням, що дозволяє користувачам вільно редагувати та налаштовувати його функціонал відповідно до власних потреб. Крім того, Odoo забезпечує досить простий та зручний інтерфейс користувача, що дозволяє легко використовувати програмне забезпечення без спеціальної підготовки.

Ще однією перевагою Odoo є те, що він має велику та активну спільноту користувачів та розробників, яка постійно вдосконалює та доповнює його

функціонал. Це забезпечує підтримку та оновлення програмного забезпечення відповідно до останніх технологічних тенденцій та вимог користувачів.

У залежності від потреб бізнесу, Odoo може бути налаштований для виконання різних завдань, включаючи управління складом, виробництвом, продажем, закупівлями, фінансами та іншими. Одним з недоліків Odoo є те, що налаштування та конфігурація системи можуть вимагати певного рівня експертизи та знань з програмування.[13]

Ще однією перевагою Odoo є те, що він підтримує різні мови та локалізацію, що дозволяє використовувати його для управління бізнесом в різних країнах світу з урахуванням місцевих законодавчих та культурних особливостей.

Odoo також має вбудований веб-магазин, що дозволяє легко створювати та управляти електронними магазинами та інтернет-майданчиками для продажу товарів та послуг. Крім того, Odoo підтримує інтеграцію з різними сторонніми програмними засобами та сервісами, такими як платіжні системи, електронні поштові сервіси, сервіси доставки тощо.

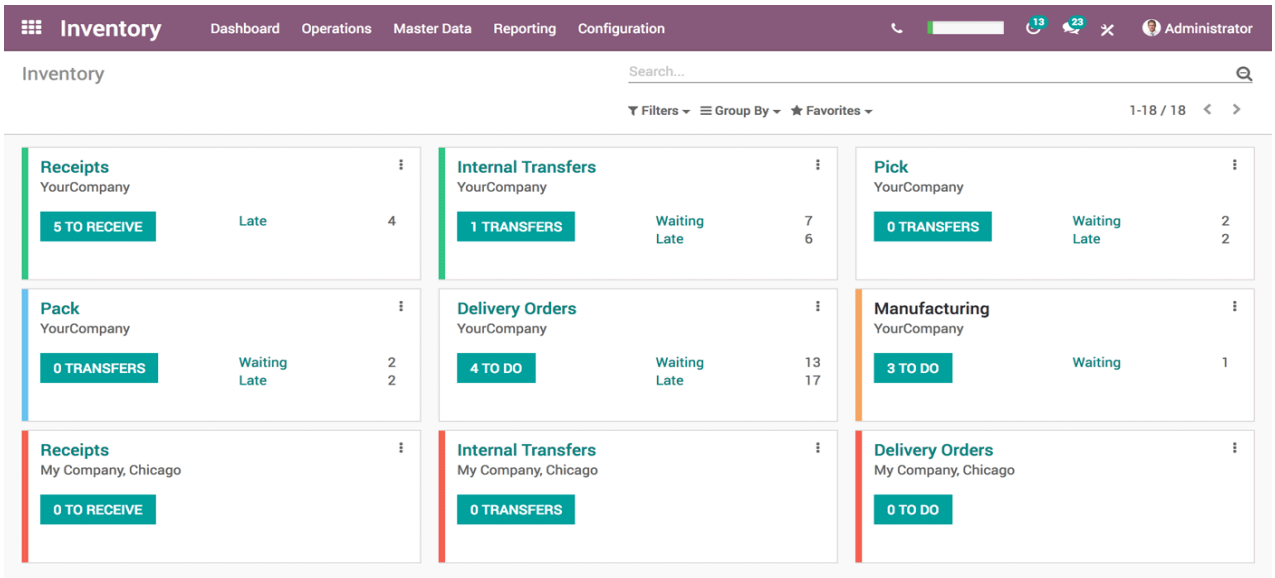


Рисунок 1.3 - Приклад інтерфейсу Odoo

Однак, як і у будь-якого програмного забезпечення, Odoo також має свої недоліки. Наприклад, при налаштуванні системи можуть виникати складнощі

та проблеми з безпекою, особливо якщо користувачі не мають досвіду в роботі з програмним забезпеченням. Крім того, якщо бізнес вимагає великої кількості користувачів, то можуть виникнути проблеми з продуктивністю та швидкістю роботи системи.

Усього враховуючи, Odoo є потужним та гнучким програмним забезпеченням для управління бізнесом, яке надає широкі можливості для налаштування та інтеграції з іншими системами. Залежно від потреб та особливостей конкретного бізнесу, Odoo може бути ефективним рішенням для управління різними аспектами діяльності, включаючи складський облік.

1.2.2 WMS на базі Odoo

Odoo надає інтегровану платформу, що охоплює не тільки складський облік, а й інші функціональні області, такі як продажі, закупівлі, бухгалтерія та керування виробництвом. Це забезпечує взаємодію між різними департаментами підприємства та покращує продуктивність та ефективність процесів. Система надає широкі можливості для управління складськими операціями, включаючи керування запасами, отримання та відправлення товарів, замовлення, пакування та відстеження доставки. Завдяки різноманітним функціям та інструментам, Odoo допомагає автоматизувати багато рутинних задач та покращити ефективність роботи складського відділу.

Odoo забезпечує гнучкість та можливість налаштування системи під конкретні потреби підприємства. Вона дозволяє використовувати готові модулі для складського обліку, але є його функціонал зазвичай є малим та не відповідає всім потребам WMS системи.[15]

Odoo, як універсальна бізнес-система, може мати обмежену функціональність та спеціалізацію в порівнянні зі спеціалізованими WMS системами, які були розроблені спеціально для конкретних галузей або

індустрій. В деяких випадках це може бути обмеженням для компаній з особливими вимогами до складського обліку.

Odoo надає можливість розробляти та інтегрувати власні модулі за необхідністю. Це дозволяє адаптувати систему до унікальних бізнес-процесів та вимог компанії. Система базується на відкритому коді, що дозволяє підприємству мати повний контроль над системою та використовувати її без обмежень ліцензійних витрат. Крім того, відкритий код забезпечує гнучкість та можливість змінювати або розширювати функціонал системи залежно від власних потреб.

Ще одним недоліком є вартість. В порівнянні з деякими іншими WMS системами, використання Odoo може бути вартісним, особливо коли враховувати витрати на налаштування, підтримку та навчання персоналу. Для деяких компаній це може

Загалом, Odoo є потужним інструментом для впровадження системи управління складом (WMS), що надає широкі можливості для автоматизації та оптимізації складських процесів у різних підприємствах.

1.3 Опис існуючих рішень

1.2.1 SAP ERP

SAP є однією з найбільш відомих ERP-систем, що надає рішення для складського обліку. Вона надає функціональні можливості для управління запасами, прогнозування попиту, операційної діяльності, логістики та фінансів. SAP ERP також підтримує багато мов та валют, що робить його корисним для багатонаціональних компаній. Однак, SAP є досить дорогим та складним у

використанні.

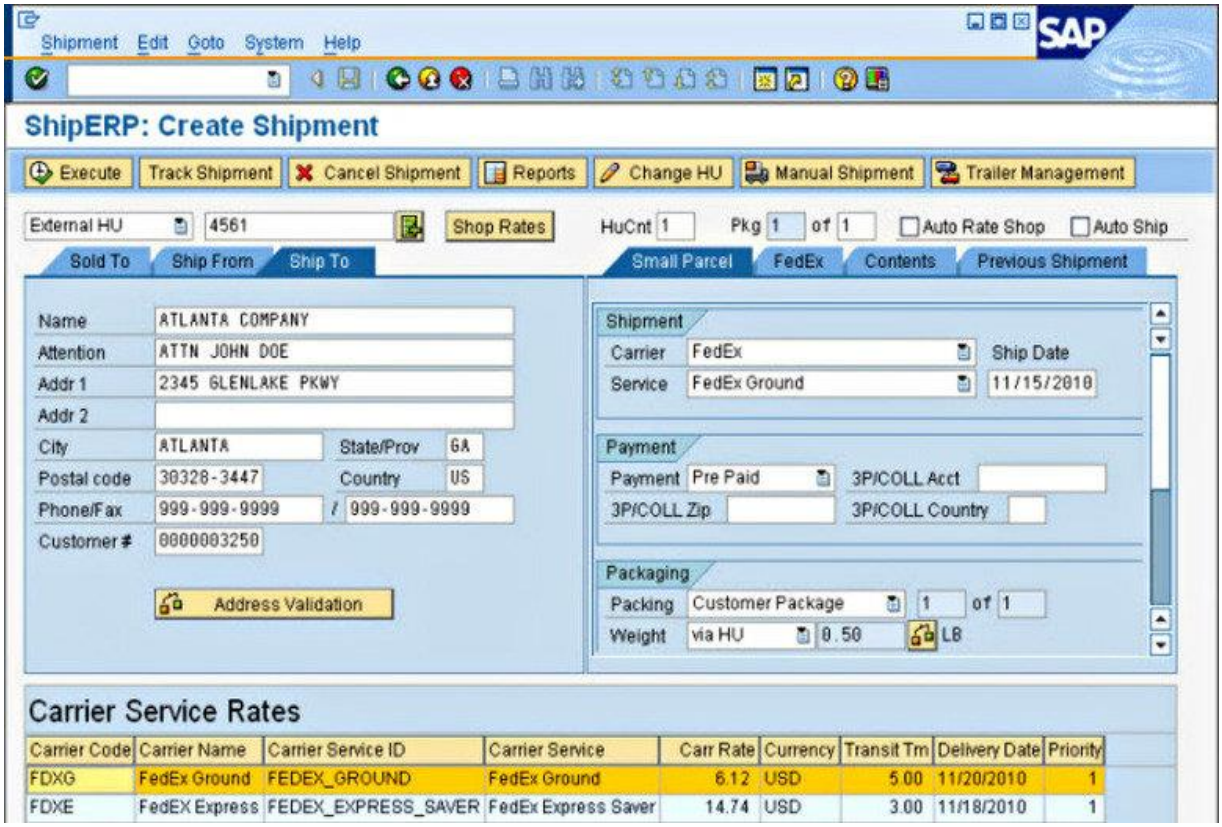


Рисунок 1.4 - Приклад інтерфейсу SAP ERP

SAP ERP використовується компаніями різних розмірів та галузей, надаючи їм інструменти для ефективного управління бізнесом та досягнення стратегічних цілей.

1.2.2 Oracle Warehouse Management

Це ERP-система, що спеціалізується на управлінні складом та запасами. Oracle Warehouse Management надає функціональні можливості для управління

запасами, приймання та відвантаження товарів, а також прогнозування попиту.

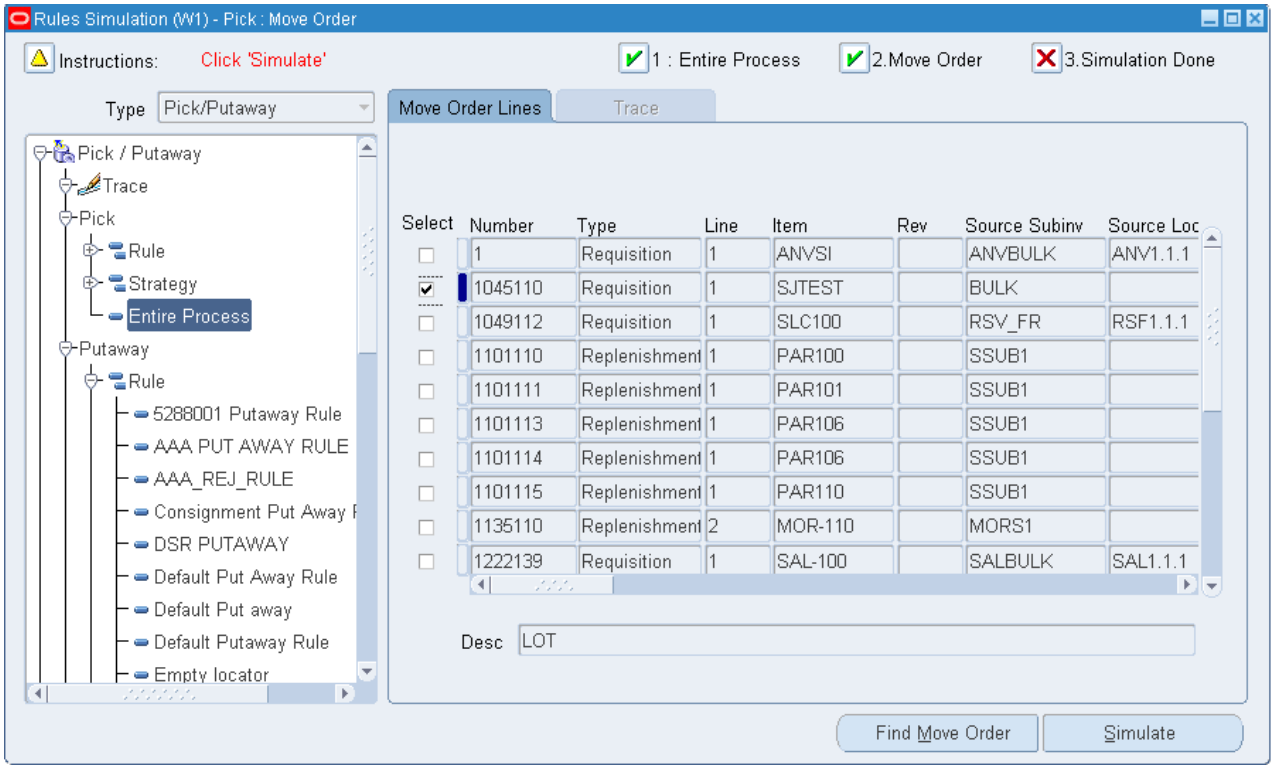


Рисунок 1.5 - Приклад інтерфейсу Oracle Warehouse Management

Однією з переваг Oracle Warehouse Management є те, що вона може інтегруватись з іншими системами управління запасами, такими як системи радіочастотної ідентифікації (RFID). Однак, Oracle Warehouse Management також є дорогим та складним у використанні. Впровадження OWM може бути складним та часоємним процесом. Необхідно провести детальний аналіз бізнес-процесів, налаштувати систему відповідно до потреб підприємства, інтегрувати її з іншими системами та навчити персонал користуватися новою системою.

1.2.3 Microsoft Dynamics 365 Supply Chain Management

Це ERP-система, що надає рішення для управління запасами, логістики та операційної діяльності. Microsoft Dynamics 365 Supply Chain Management також має функціональні можливості для прогнозування попиту та планування

виробництва.

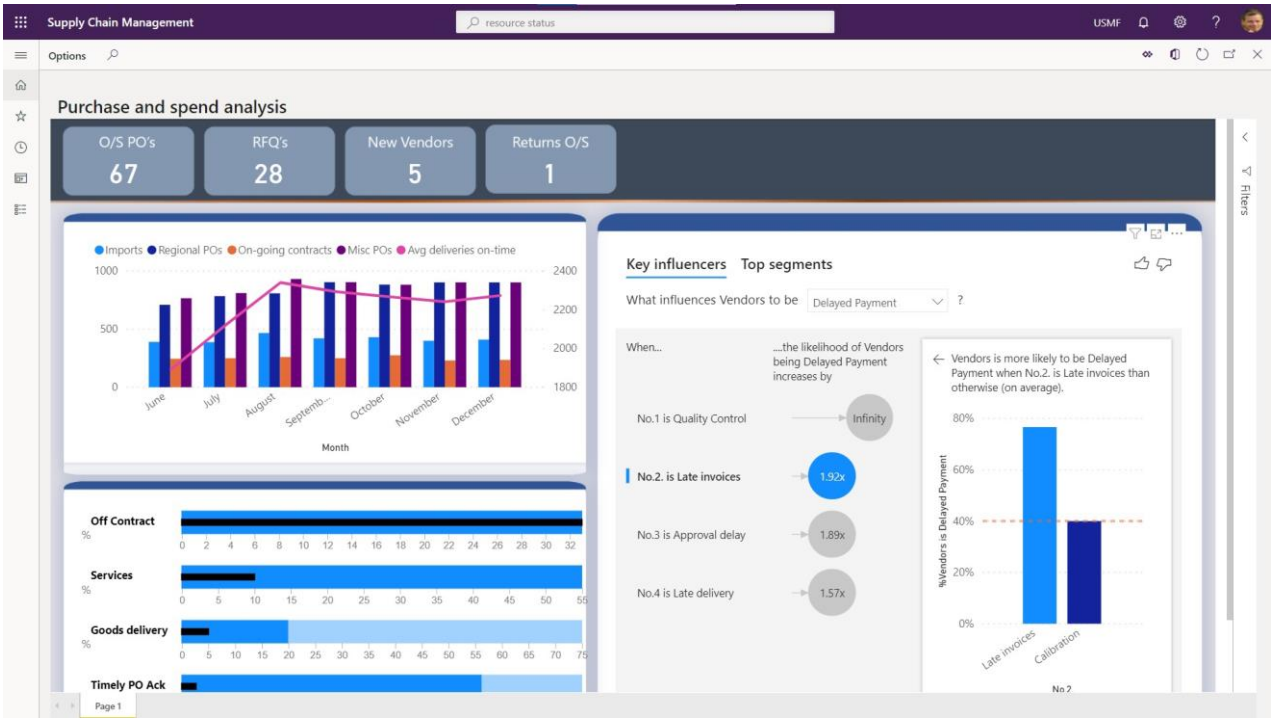


Рисунок 1.6 - Приклад інтерфейсу Microsoft Dynamics 365 Supply Chain Management

Однією з переваг Microsoft Dynamics 365 Supply Chain Management є те, що вона інтегрується з іншими продуктами Microsoft, такими як Microsoft Office та Power BI. Microsoft Dynamics 365 Supply Chain Management також є більш доступною та менш складною у використанні, порівняно з SAP та Oracle.

З недоліків системи можна виділити обмеження в настраювання: Microsoft Dynamics 365 Supply Chain Management має обмежені можливості настройки та модифікації. Це може обмежувати гнучкість системи та можливості впровадження, особливо для підприємств з унікальними бізнес-процесами.

1.3 Порівняння існуючих рішень

Кожна з цих ERP-систем має свої переваги та недоліки. Вибір ERP-системи для складського обліку залежить від потреб компанії та її можливостей щодо фінансів та технічного обладнання. Наприклад, якщо компанія має обмежений бюджет, то Microsoft Dynamics 365 Supply Chain Management може бути кращим варіантом, оскільки вона є більш доступною за ціною. Однак, якщо компанія є багатонаціональною та має потребу у рішенні, яке підтримує багато мов та валют, то SAP може бути кращим варіантом.

Проте, вибір системи управління ланцюгом постачання є індивідуальним для кожного підприємства, і важливо ретельно проаналізувати свої унікальні потреби та вимоги перед прийняттям рішення. Дослідження, порівняння і взваження всіх факторів, таких як функціональність, вартість, впровадження, інтеграція та обслуговування, є необхідними кроками для забезпечення успішного вибору системи, яка найкраще відповідає потребам вашого підприємства.

Крім того, варто звернути увагу на інтерфейс користувача та зручність використання ERP-системи. Це особливо важливо для компаній, що мають багато співробітників, що працюють з ERP-системою.

Отже, при виборі ERP-системи для складського обліку варто ретельно проаналізувати потреби компанії та порівняти різні варіанти, щоб знайти той, який найкраще відповідає її потребам.

Таблиця 1.1 – Порівняння існуючих рішень

Особливості	Odoo	SAP	Microsoft Dynamics 365 Supply Chain Management	Oracle Warehouse Management
Функціональність	Повний набір функцій	Широкий спектр функцій	Розширені можливості управління ланцюгом постачання	Широкий спектр функцій
Вартість	Відкритий код, вартість реалізації та підтримки залежить від постачальника та потреб підприємства	Висока вартість володіння та підтримки	Висока вартість володіння та підтримки	Висока вартість володіння та підтримки
Впровадження	Відносно простий процес впровадження	Складний та часоємний процес впровадження	Складний та часоємний процес впровадження	Складний та часоємний процес впровадження
Гнучкість	Висока гнучкість і настроюваність	Висока гнучкість та можливості настроювання	Обмежені можливості настроювання	Обмежені можливості настроювання

Виходячи з порівняльної таблиці, рекомендується обрати Odoo як WMS систему

ВИСНОВОК ДО РОЗДІЛУ 1

У висновку, моє доопрацювання до системи WMS на базі Odoo має на меті вирішити і преодоліти всі недоліки, які присутні у WMS системах загалом, включаючи Odoo. Шляхом належного аналізу та розробки, мої зусилля спрямовані на створення удосконаленої системи, яка буде ефективно вирішувати проблеми, пов'язані зі складським обліком.

Інтеграція з іншими модулями Odoo, вдосконалення налаштування та забезпечення більшої гнучкості управління складськими процесами дозволить покращити функціональність та забезпечити більш точний та ефективний облік запасів. Крім того, постійне вдосконалення системи на базі Odoo, використання відкритого коду та спільноти розробників дозволить швидко реагувати на зміни та внести необхідні модифікації для вирішення специфічних потреб компаній.

Моє доопрацювання до системи WMS на базі Odoo сприятиме покращенню управління складським обліком, забезпеченню точності та ефективності управління запасами, а також інтеграції з іншими бізнес-процесами. В результаті це дозволить забезпечити більш оптимальне використання ресурсів, зниження витрат та підвищення загальної продуктивності.

РОЗДІЛ 2

ВИБІР ТЕХНОЛОГІЙ ТА СЕРЕДОВИЩА РОЗРОБКИ ПЛАТФОРМИ

2.1 Обрані технології розробки

2.1.1 Python

Python - це потужна інтерпретована, високорівнева мова програмування, яка поєднує простоту і зручність використання з широким спектром можливостей. Вона була розроблена Гвідо ван Россумом і вперше випущена у 1991 році.[6]

Однією з основних переваг Python є його зрозумілість і легкість вивчення. Синтаксис мови простий та інтуїтивно зрозумілий, що дозволяє швидко розпочати програмування навіть новачкам. Python відомий своєю читабельністю і чистотою коду, що сприяє підтримці і розумінню програмного коду з часом.

Python є мовою загального призначення, що означає, що вона підходить для різних типів задач і може бути використана в різних галузях, таких як веб-розробка, наукові дослідження, аналітика даних, штучний інтелект, машинне навчання та багато іншого.

Python є мовою з великою кількістю стандартних бібліотек, які надають багато корисних функцій і інструментів. Крім того, велика спільнота розробників активно підтримує мову і створює багато сторонніх бібліотек та фреймворків, що дозволяють розширити можливості Python ще більше.

Одним з найбільш відомих особливостей Python є його динамічна типізація, що дозволяє змінювати типи змінних під час виконання програми. Це спрощує розробку та збільшує гнучкість мови.

Python також підтримує різні парадигми програмування, включаючи процедурне, об'єктно-орієнтоване та функціональне програмування. Це дозволяє розробникам вибирати найбільш підходящий стиль програмування для своїх потреб.

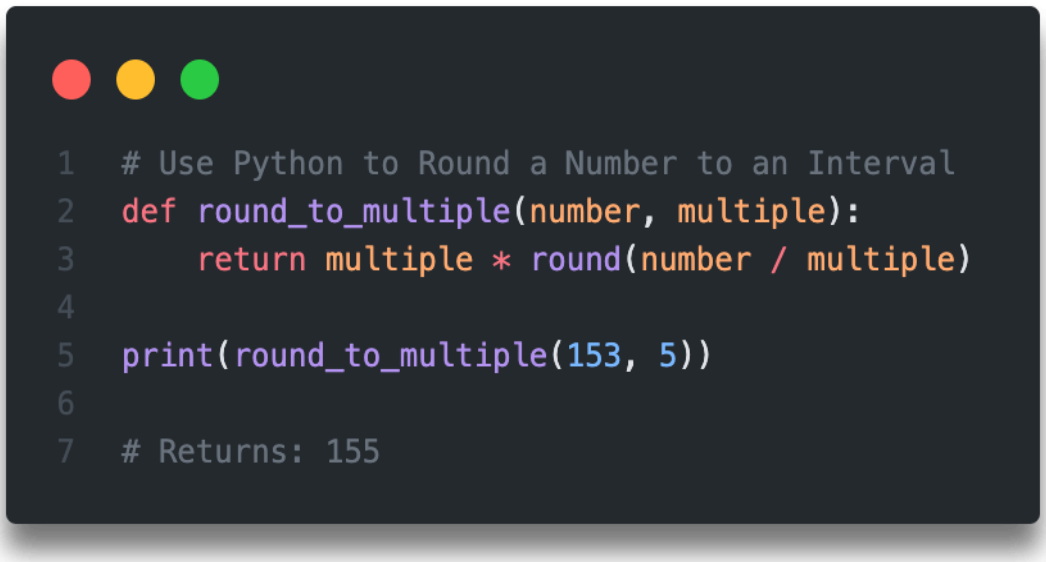


Рисунок 2.1 - приклад коду на мові Python

Узагальнюючи, Python - це потужна та зручна мова програмування, яка надає широкі можливості для розробки різноманітних програм і додатків. Його простий синтаксис, широкі бібліотеки та активна спільнота розробників роблять Python популярним вибором для багатьох проектів. Також вібір Python обумовлений тим, що Odoo побудована за допомогою цієї мови.

2.1.2 OWL

У контексті фреймворку Odoo, термін OWL походить від скорочення "Object Widget Library" (бібліотека об'єктних віджетів). OWL - це набір інтерактивних компонентів і віджетів, які використовуються для створення користувацького інтерфейсу в Odoo.[5]

OWL надає розробникам широкі можливості для створення різноманітних елементів у користувацькому інтерфейсі, таких як кнопки, поля введення, списки, таблиці, фільтри і багато інших. Ці компоненти дозволяють створювати інтерактивні та зручні для використання елементи у програмах Odoo.

```
odoo.define("slide_local_video.form_render", function (require) {
    "use strict";


    var FormRenderer = require('web.FormRenderer');
    var FormController = require('web.FormController');
    var FormView = require('web.FormView');
    FormRenderer.include({
        _renderView: function () {
        var self = this;
        var res = this._super.apply(this, arguments).then(function () {
            $(self.$('#input_file')).on('change', function (el) {
                self.readFile(el.currentTarget.files[0], self)
            })
        });

            return res
    },
```

Рисунок 2.2 - приклад коду на мові JS з використанням OWL

Особливість OWL полягає в його інтеграції з серверною частиною фреймворку. OWL використовує AJAX (асинхронний JavaScript і XML) для взаємодії з сервером без перезавантаження сторінок. Це дозволяє розробникам

створювати динамічні та швидкодіючі користувацькі інтерфейси, які реагують на дії користувача в реальному часі.[13]

Крім того, OWL підтримує розширення та модифікацію компонентів за допомогою класів на основі JavaScript. Розробники можуть використовувати ці класи для налаштування поведінки віджетів, додавання нових функціональних можливостей та забезпечення відповідності інтерфейсу вимогам бізнес-логіки додатків.

Узагальнюючи, OWL відіграє важливу роль в створенні інтерактивного та зручного для використання користувацького інтерфейсу в Odoo. Використовуючи компоненти OWL, розробники можуть створювати розширені та інтерактивні елементи у програмах Odoo, що сприяє поліпшенню користувацького досвіду та ефективності роботи з системою.

2.1.3 XML

XML (Extensible Markup Language) - це розширювана мова розмітки, яка використовується для опису структурованої інформації в електронному вигляді. XML використовується для зберігання і передачі даних між різними системами, а також для розробки веб-сторінок, веб-сервісів та інших програмних рішень.[16]

Основними принципами XML є:

1. Розширюваність: XML дозволяє розробникам визначати свої власні теги та структури даних, що дозволяє створювати спеціалізовані формати для конкретних потреб.
2. Читабельність: XML використовує текстовий формат, що робить дані легкими для читання та редагування як людиною, так і комп'ютером.

3. Роздільність даних і вигляду: XML окремо зберігає дані та їх вигляд. Дані описуються в структурованому форматі, а вигляд визначається за допомогою CSS або XSLT.
4. Підтримка міжплатформенності: XML є платформо незалежним форматом, що дозволяє передавати дані між різними операційними системами та програмами.

В контексті фреймворку Odoo, XML використовується для опису структури та поведінки модулів. Використання XML дозволяє визначити модель даних, вигляд користувацького інтерфейсу, правила доступу до даних, діаграми робочих процесів та багато іншого. XML файлами описуються конфігураційні параметри, що дозволяє зручно налаштовувати та розширювати функціональні можливості системи.

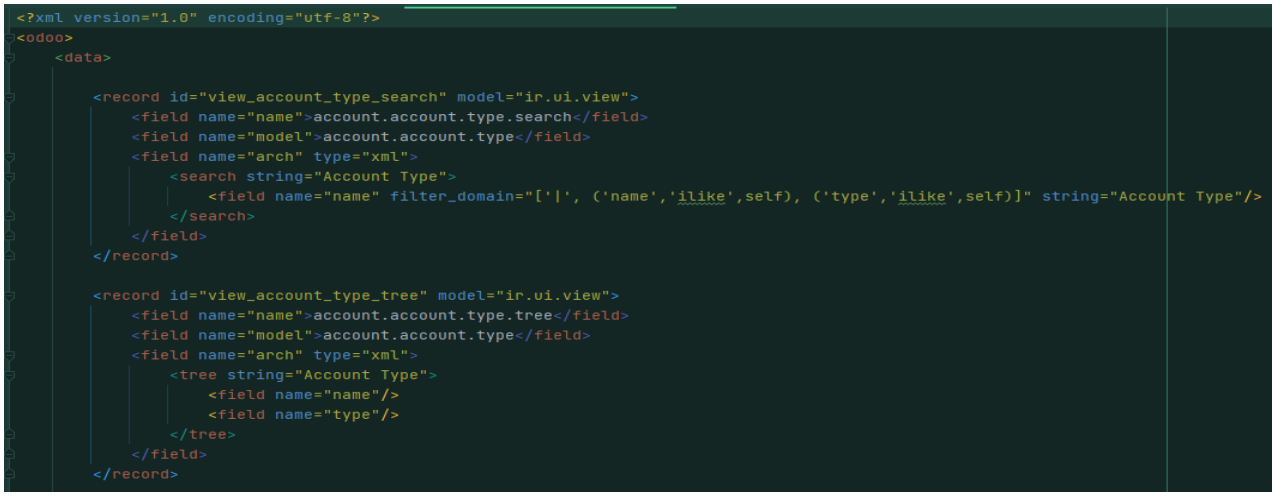


Рисунок 2.3 - приклад коду на мові XML для створення вигляду користувацького інтерфейсу

Узагальнюючи, XML є мовою розмітки, яка використовується для структурування даних та опису їх вигляду. В контексті Odoo, XML використовується для опису модулів та їх функціональності, що дозволяє розробникам легко визначати та налаштовувати систему.

2.1.4 Vue.js

Vue.js - це прогресивний JavaScript фреймворк для розробки користувацьких інтерфейсів веб-додатків. Він зосереджується на створенні інтерактивних односторінкових додатків (SPA) та реактивних компонентів.[14]

Основні особливості Vue.js:

1. Легкість вивчення: Vue.js має простий і зрозумілий синтаксис, що робить його легким для вивчення навіть для початківців. Він також має детальну документацію та активну спільноту, яка готова надати підтримку та відповіді на питання.
2. Реактивність: Vue.js використовує реактивний підхід до програмування, що дозволяє автоматично оновлювати відображення сторінки при зміні даних. Завдяки цьому, розробники можуть легко відслідковувати та керувати станом додатку.
3. Компонентна архітектура: Vue.js побудований на основі компонентної архітектури, що дозволяє розбити додаток на незалежні компоненти. Це спрощує розробку, підтримку та перевикористання коду.
4. Широкі можливості: Vue.js має багатий екосистема плагінів та доповнень, які дозволяють розширити його функціональність. Також, Vue.js можна поєднувати з іншими бібліотеками та фреймворками, що дає велику свободу вибору технологій для розробки.
5. Висока продуктивність: Vue.js має оптимізовану систему рендерингу, що забезпечує швидку відповідь інтерфейсу користувача. Він також має можливість лінивої загрузки компонентів, що дозволяє завантажувати лише необхідний код для певного компонента.

Узагальнюючи, Vue.js є потужним та легким у використанні JavaScript фреймворком, який дозволяє розробникам ефективно створювати реактивні

веб-додатки та користувацькі інтерфейси. Його простота, реактивність та компонентна архітектура роблять його популярним вибором для розробників.

2.1.5 PostgreSQL

PostgreSQL - це потужна та розширювана система управління базами даних (СУБД), яка надає надійне та ефективне зберігання та керування даними. Основні особливості PostgreSQL включають:

1. Реляційна модель даних: PostgreSQL заснована на реляційній моделі даних, що дозволяє зберігати дані у вигляді таблиць зі зв'язками між ними. Це забезпечує структурованість та організованість даних, що полегшує їх обробку та аналіз.
2. Розширена підтримка SQL: PostgreSQL повністю підтримує мову SQL (Structured Query Language) і надає широкий набір функцій та операцій для роботи з даними. Вона дозволяє виконувати різноманітні запити, створювати складні запити, здійснювати об'єднання та агрегування даних.
3. Розширені можливості: PostgreSQL має багатий набір функціональностей та можливостей, які дозволяють ефективно керувати даними. Серед них - підтримка транзакцій, оптимізація запитів, індексація даних, реплікація, безпека даних та інше. Також PostgreSQL підтримує різні типи даних, включаючи текстові, числові, дати, географічні та багато інших.
4. Розширюваність: PostgreSQL надає можливість розширювати функціональність шляхом створення власних функцій, типів даних та мов програмування. Це дозволяє розробникам адаптувати систему під свої потреби та використовувати специфічні рішення.

5. Висока надійність та масштабованість: PostgreSQL підтримує механізми забезпечення надійності даних, такі як транзакції, контроль цілісності та механізми відновлення. Вона також може працювати з великими обсягами даних та підтримує розподілені системи зберігання даних.

У контексті розробки на Odoo, PostgreSQL використовується як база даних для зберігання і керування даними, які використовуються в системі. Його розширені можливості та надійність роблять PostgreSQL частим вибором для підприємств, які використовують Odoo для автоматизації своїх бізнес-процесів.

2.2 Ознайомлення з середовищем для розробки програмного додатку

2.2.1 PyCharm

Рисунок 2.5 - PyCharm

PyCharm - це інтегроване середовище розробки (IDE) для мови програмування Python, розроблена компанією JetBrains. Воно надає

розширений набір інструментів та функціональностей, що полегшують процес розробки Python-додатків.[9]

Основні особливості PyCharm:

1. Редактор коду: PyCharm має потужний редактор коду з висвітленням синтаксису, автозаповненням, перевіркою на помилки та іншими корисними функціями. Він також підтримує роботу з різними типами файлів, такими як Python-сценарії, HTML-сторінки, CSS-файли тощо.
2. Управління проектами: PyCharm дозволяє легко створювати та налаштовувати проекти Python. Він надає можливість організовувати файли та пакети, керувати залежностями, встановлювати віртуальні середовища та налаштовувати інші параметри проекту.
3. Відладка: PyCharm надає потужний інструментарій для відлагодження Python-коду. Він підтримує точки зупинки, відстеження значень змінних, крок за кроком виконання коду та інші функції, які допомагають знайти та виправити помилки в програмі.
4. Автоматичне завершення коду: PyCharm пропонує автодоповнення коду, що значно прискорює процес розробки. Він виявляє доступні методи, функції, класи та інші елементи коду та пропонує їх у вигляді підказок.
5. Інтеграція з інструментами: PyCharm підтримує інтеграцію з різними засобами розробки, такими як системи контролю версій (наприклад, Git), інструменти для управління пакетами (наприклад, pip), системи тестування, бази даних та інші. Це дозволяє розробникам зручно використовувати різноманітні інструменти в межах одного середовища.
6. Розширення функціональності: PyCharm підтримує розширення за допомогою плагінів. Розробники можуть встановлювати розширення, які додають нові функції, підтримку інших мов програмування, інтеграцію з іншими інструментами та інші можливості.

Узагальнюючи, PyCharm є потужним інструментом для розробки Python-додатків, який надає широкий спектр функціональностей та інструментів для зручного та продуктивного програмування.

2.2.2 PgAdmin 4

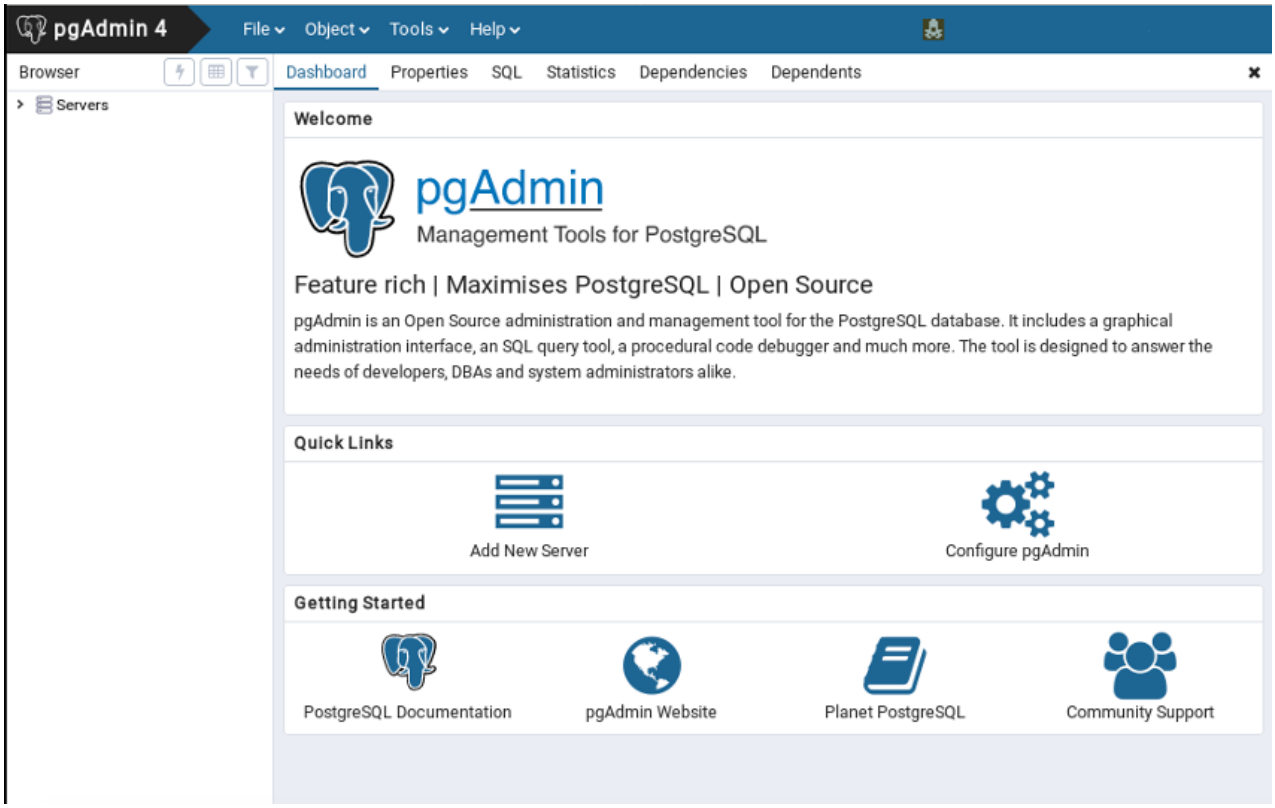


Рисунок 2.6 - Інтерфейс PgAdmin 4

PgAdmin 4 є потужним графічним інструментом для управління базами даних PostgreSQL. Він надає зручне та інтуїтивно зрозуміле середовище для адміністрування та роботи з базами даних PostgreSQL. Основні особливості pgAdmin 4 включають:

1. Керування базами даних: pgAdmin 4 дозволяє створювати, видаляти та змінювати бази даних PostgreSQL. Ви можете виконувати різні операції, такі як створення схем, таблиць, індексів та інших об'єктів бази даних.

2. Виконання SQL-запитів: За допомогою pgAdmin 4 ви можете виконувати SQL-запити безпосередньо до бази даних. Ви можете виконувати запити на вибірку, вставку, оновлення та видалення даних, а також виконувати складні запити з використанням функцій та операцій PostgreSQL.[18]
3. Візуалізація структури бази даних: pgAdmin 4 надає зручний інтерфейс для перегляду структури бази даних, схем, таблиць, стовпців та зв'язків між ними. Ви можете використовувати деревовидний перегляд для навігації по об'єктах бази даних та отримання детальної інформації про них.
4. Моніторинг та оптимізація: pgAdmin 4 надає інструменти для моніторингу та оптимізації продуктивності бази даних PostgreSQL. Ви можете переглядати статистику запитів, виконувати пошук неефективних запитів, аналізувати плани виконання та знаходити шляхи для покращення продуктивності.
5. Безпека та управління користувачами: pgAdmin 4 дозволяє керувати безпекою бази даних PostgreSQL. Ви можете створювати та керувати ролями та користувачами, встановлювати дозволи доступу до об'єктів бази даних, а також виконувати резервне копіювання та відновлення даних.

Загалом, pgAdmin 4 є незамінним інструментом для адміністрування та роботи з базами даних PostgreSQL у зручному та потужному графічному інтерфейсі.

2.3 Опис архітектури програмного забезпечення

Трирівнева архітектура в Odoo використовується для розробки багатофункціональних бізнес-систем і включає наступні рівні:[17]

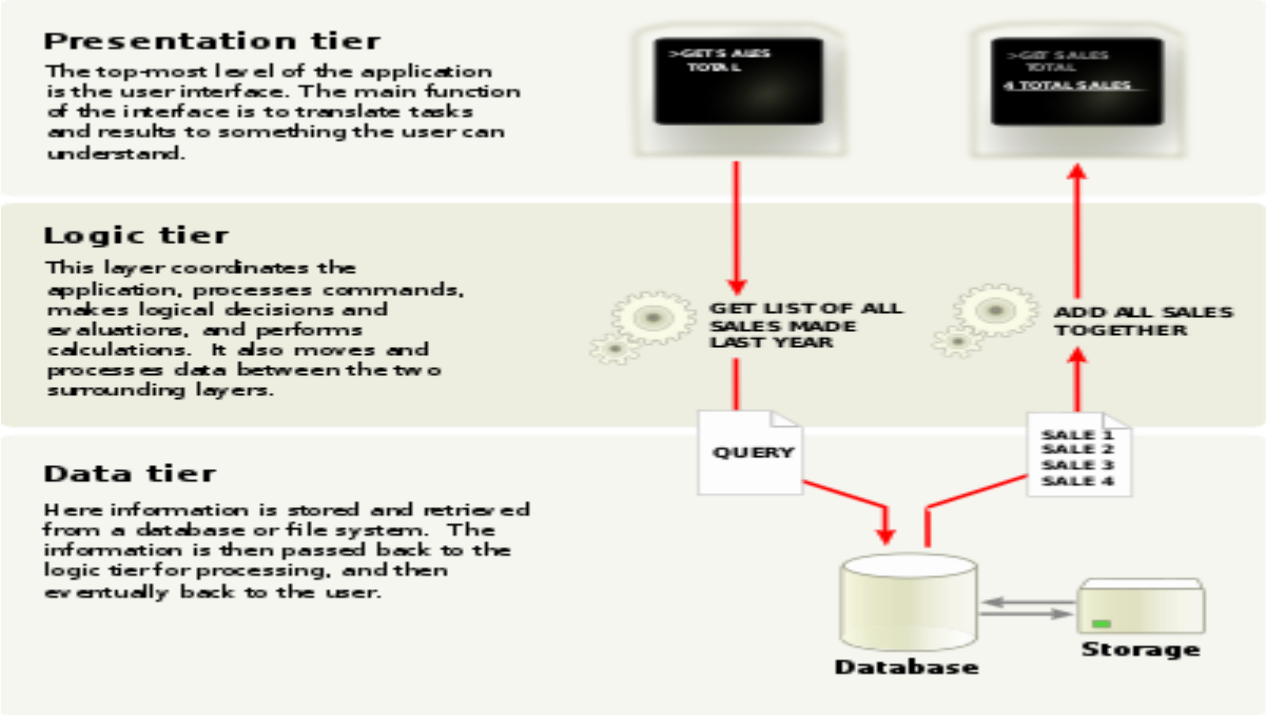


Рисунок 2.7 - Огляд трирівневої програми.

1. Рівень бази даних:

- Використовується база даних PostgreSQL для збереження даних системи.

- База даних містить структуру та відношення між різними об'єктами даних, такими як клієнти, продукти, замовлення тощо.

2. Бізнес-логіка:

- У цьому рівні виконується бізнес-логіка системи, така як обробка замовлень, управління складом, фінансовий облік та інше.

- Використовуються модулі Odoo, які включають функціональність для реалізації різних бізнес-процесів.

- Розробники можуть створювати власні модулі або налаштовувати існуючі модулі для відповідності потребам компанії.

3. Веб-інтерфейс:

- Користувачі взаємодіють з системою за допомогою веб-інтерфейсу.

- Веб-інтерфейс забезпечує зручну навігацію, введення та виведення даних, а також доступ до різних функцій системи.

- Використовується фреймворк веб-розробки, такий як Vue.js або JavaScript, для створення динамічного інтерфейсу.

Трирівнева архітектура дозволяє відокремити логіку бізнес-процесів від бази даних та представлення, що забезпечує більшу гнучкість і можливість масштабування системи. Крім того, це дозволяє розробникам працювати окремо над кожним рівнем і забезпечує легку модифікацію та розширення системи.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі дипломної роботи були представлені інформація та аналіз обраних технологій для автоматизації складського обліку на платформі Odoo. Обрані технології включають Odoo як основну платформу, базу даних PostgreSQL та мову програмування Python та JavaScript, фреймворк OWL та Vue.js, мову розмітки XML.

Odoo є високо функціональною платформою для управління бізнес-процесами, що надає широкі можливості для автоматизації складського обліку. Вона забезпечує інтеграцію різних модулів, що дозволяє ефективно керувати запасами, замовленнями, поставками та іншими аспектами складського обліку. Вибір Odoo в якості основної платформи був обґрунтований її функціональністю, гнучкістю та можливістю розширення.

PostgreSQL обрано як базу даних для зберігання та управління інформацією, пов'язаною зі складським обліком. Ця база даних відома своєю надійністю, масштабованістю та широким набором функціональних можливостей. Використання PostgreSQL дозволяє ефективно зберігати та обробляти великі обсяги даних, що є важливим для точного та швидкого складського обліку.

Мова програмування Python була обрана для розробки та налаштування функціональності на платформі Odoo. Python є простою, зрозумілою та потужною мовою програмування, що дозволяє реалізувати різноманітні функції та забезпечити інтеграцію з іншими системами. Використання Python спрощує процес розробки та підтримки рішень для складського обліку на платформі Odoo.

Загалом, обрані технології - Odoo, PostgreSQL та Python - відповідають вимогам та цілям автоматизації складського обліку. Вони забезпечують широкі можливості для ефективного керування запасами, оптимізації процесів та забезпечення точності та надійності у складському обліку.

РОЗДІЛ 3

РОЗРОБКА ТА ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Ініціалізація та налаштування проекту

3.1.1 Ініціалізація та налаштування Odoo

На перших етапах розробки нам потрібно встановити всі необхідні компоненти. Спочатку треба скачати та встановити IDE PyCharm, яка надасть зручне середовище для розробки на Python. Це можна зробити з офіційного сайту компанії розробника JetBrains, де доступна безкоштовна Community Edition.

Далі нам необхідно скачати та встановити базу даних Postgresql SQL, яка використовується як основне сховище даних для Odoo. Це потрібно зробити з офіційного веб-сайту Postgresql, де можна знайти відповідну версію для своєї операційної системи.

Щоб почати працювати з Odoo, необхідно скопіювати проект з офіційного репозиторію GitHub Odoo. В вашому випадку ви будете використовувати версію 14.0 Community Edition. Це можна зробити шляхом клонування репозиторію за допомогою Git або завантажити архівний файл з GitHub і розпакувати його.

Також необхідно встановити всі залежності виконавши команду

pip install -r requirements.txt

Для запуску локального сервера у вікні конфігурації проекта потрібно вказати параметр *-c*, та вказати путь до файлу конфігурації

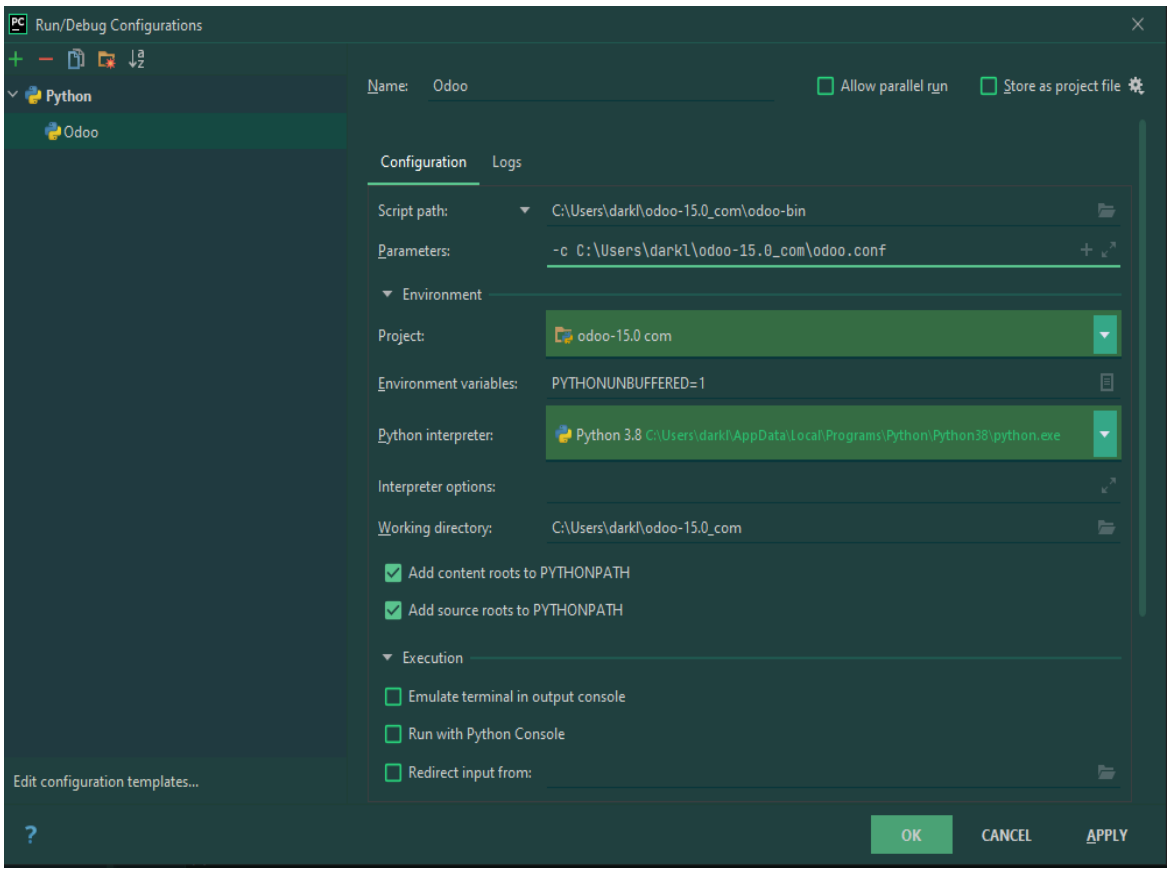


Рисунок 3.1- Приклад конфігурації проекту

Якщо все встановлено та налаштовано правильно у консолі ви побачите наступне

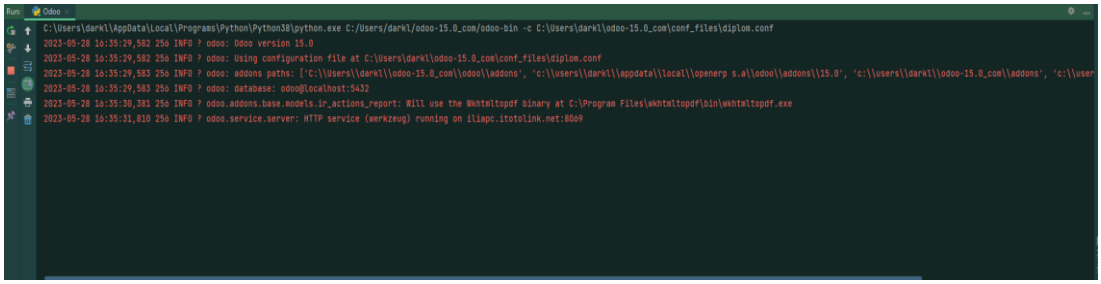


Рисунок 3.2- Приклад консолі при правильному налаштуванні

Далі потрібно перейти на http://localhost:8069/web після чого буде доступне вікно налаштування бази даних

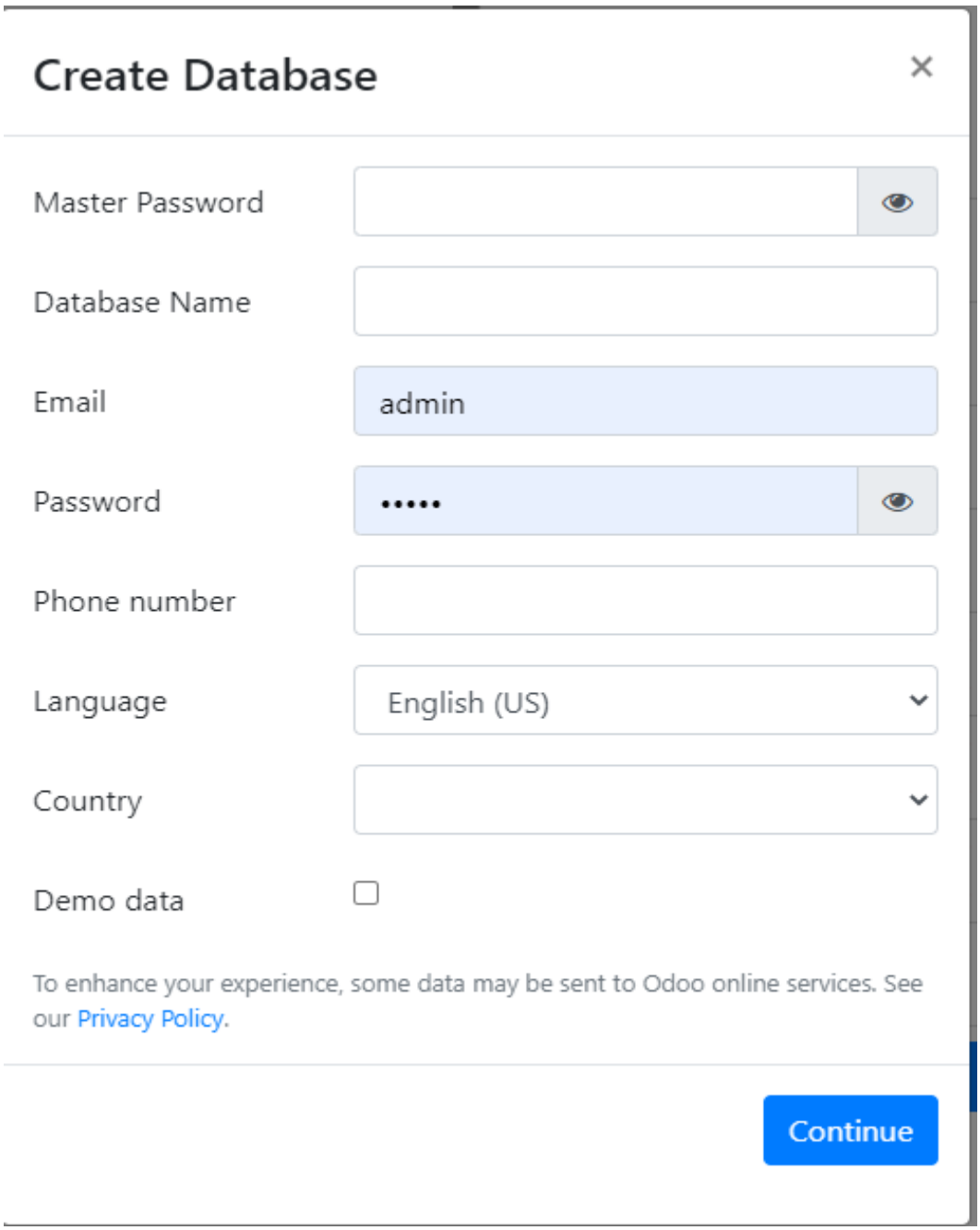


Рисунок 3.3 - Приклад створення нової бази даних

Після створення нової бази даних та входу в неї ми побачимо вікно доступних модулів. Встановлюємо будь-який модуль, наприклад Inventory(склад).

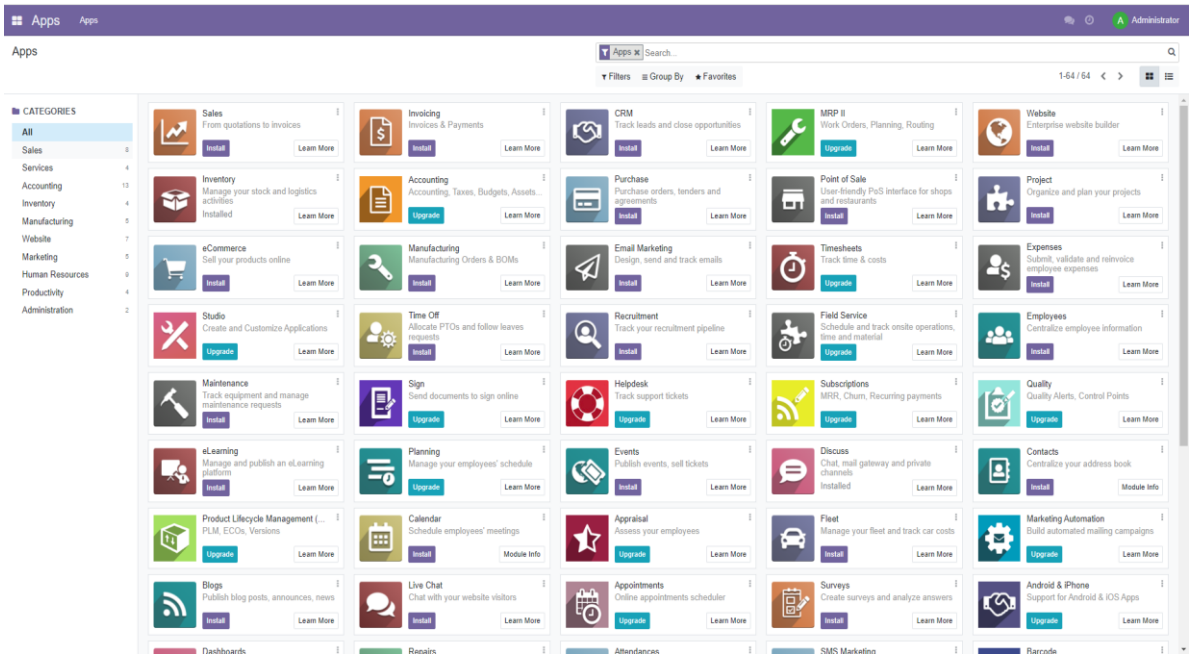


Рисунок 3.4 - Приклад сторінки модулів

Після цього налаштування та встановлення базового варіанту Odoo є завершеним.

3.1.2 Ініціалізація та налаштування власного модуля

Наступним етапом буде ініціалізація власного модуля, який буде доповнювати функціонал базового модулю Inventory (склад) в системі Odoo. Для цього вам потрібно створити директорію з назвою модулю. У нашому випадку вона буде мати назву "diplom_wms". В цій директорії потрібно створити основні файли та додаткові директорії. Основними файлами, які потрібно створити, є:

1. *__init__.py*: Цей файл має порожній вміст і використовується для ініціалізації модуля як пакету Python.
2. *__manifest__.py*: Це основний файл опису вашого модулю. Ви повинні вказати основну інформацію про модуль, таку як його назву, версію, автора, опис та залежності від інших модулів. Також ви можете вказати

необхідні ресурси, такі як файли перекладу, зображення, CSS-стилі та інші.

3. *models*: Це директорія, де ви створюєте файл (або файли) з визначенням моделей даних вашого модулю. Ви можете створити нові моделі або розширити існуючі моделі з базового модулю Inventory. Визначте поля, методи, спостерігачі та інші атрибути, необхідні для вашої функціональності.
4. *views*: Ця директорія використовується для зберігання файлів опису вигляду вашого модулю. Ви можете створити XML-файли, що визначають форми, списки, звіти та інші елементи інтерфейсу користувача, пов'язані з вашим модулем.

За необхідності можна створити додаткові директорії для зберігання файлів безпеки системи, JS, CSS та файлів з перекладом

Файли безпеки в оду - це важний аспект, який необхідно враховувати при роботі з інформацією та даними в університетському середовищі. Ці файли містять конфіденційну, особисту або важливу інформацію, яка повинна бути захищено від несанкціонованого доступу, втрати або пошкодження. Загальною метою файлів безпеки в оду є забезпечення конфіденційності, цілісності та доступності інформації. Застосування вищезазначених заходів допоможе зменшити ризик несанкціонованого доступу до файлів

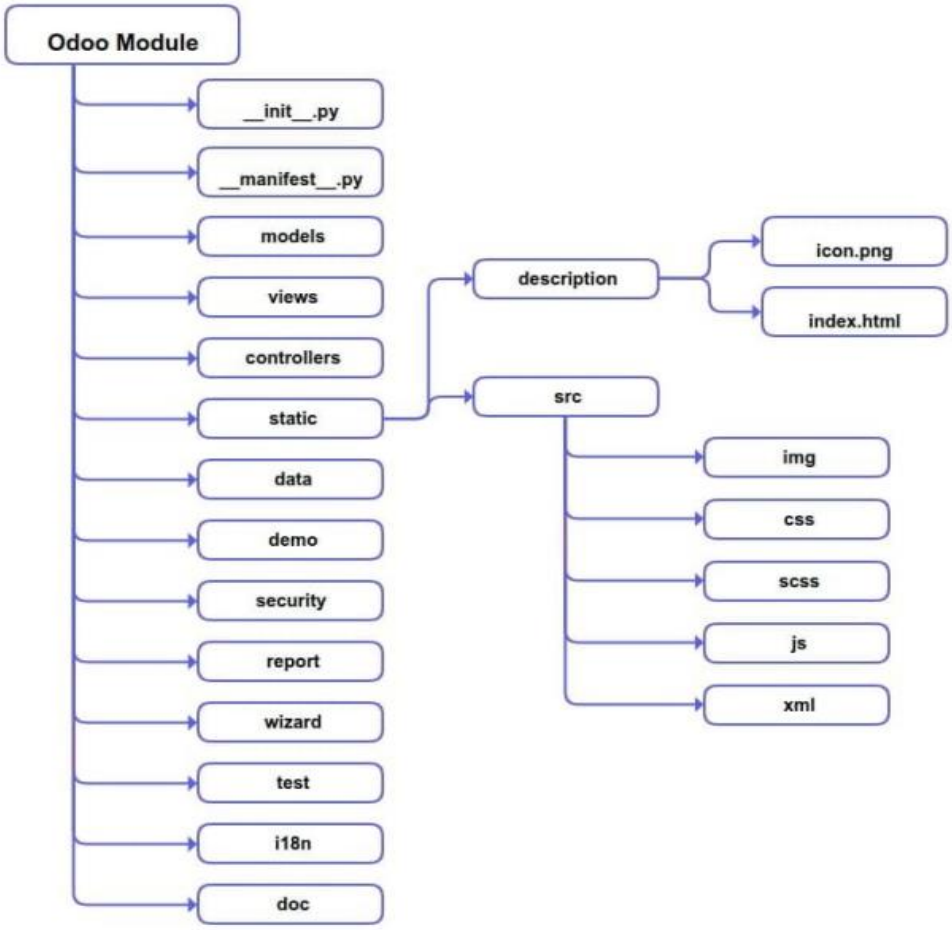


Рисунок 3.5 - Структура модулю в Odoo

Далі потрібно оформити файл “__manifest__.py”, який є основним файлом опису модулю в системі Odoo. У цьому файлі потрібно вказати основну інформацію про модуль, таку як його назву, версію, автора, опис та залежності від інших модулів. Опис полів які варто заповнювати у файлі “__manifest__.py”:

1. *name*: Назва модулю.
2. *version*: Версія модулю.
3. *author*: Автор модулю.
4. *summary*: Короткий опис модулю.
5. *description*: Детальний опис модулю, включаючи його функціональність та особливості.
6. *category*: Категорія, до якої належить модуль. Наприклад, "Warehouse" або "Inventory".

7. *depends*: Перелік модулів, від яких залежить модуль. Це дозволяє забезпечити правильну установку та роботу модулів, від яких розширюється функціональність.
8. *data*: Перелік шляхів до файлів опису вигляду(XML-файлів), перекладів, даних для заповнення тощо.
9. installable: Показник, що вказує, чи може модуль бути встановлений.
10. application: Показник, що вказує, чи є модуль додатком.

Це загальний опис ключових полів, які можна включити у файл __manifest__.py. За необхідності можна додати інші поля або налаштувати їх залежно від конкретної потреби та вимог модулю.

Рисунок 3.6 - Приклад налаштування “__manifest__.py”

Щоб ваш новостворений модуль був видимим в системі Odoo, потрібно вказати шлях до директорії, де він зберігається, у файлі конфігурації Odoo.

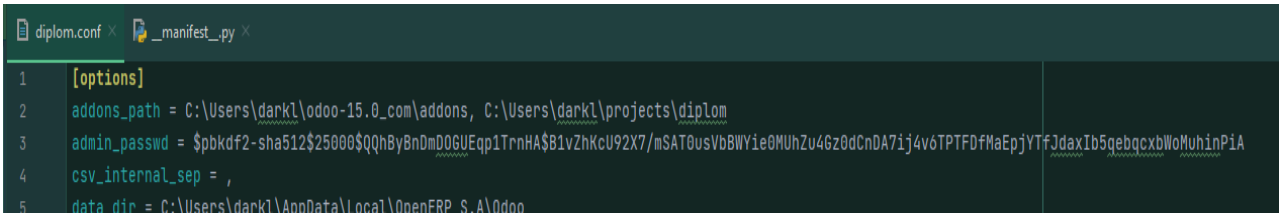


Рисунок 3.7 - Приклад файлу конфігурації

Після додавання шляху до директорії модуля у файлі конфігурації Odoo і перезапуску сервера, потрібно виконати декілька кроків, щоб модуль з'явився у списку доступних для встановлення модулів в системі Odoo:

1. Перейти на сторінку "Модулі" або "Apps".
2. На сторінці модулів натиснути кнопку "Оновити список програм" або "Update Apps list". Це дозволить системі оновити список доступних модулів на основі нових змін у файлі конфігурації.
3. У полі пошуку на сторінці модулів ввести назву створення модуля
4. Натиснути кнопку “Install” або “Встановити”

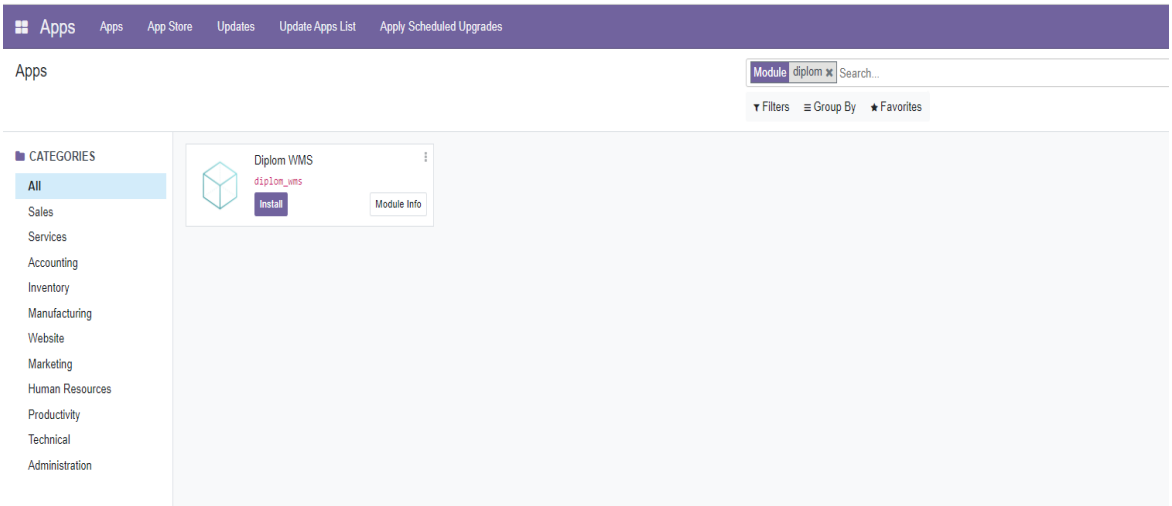


Рисунок 3.8 - Приклад відображення власного модуля

Після цих кроків модуль успішно встановлено в системі Odoo і можна почати використовувати його функціональність.

3.2 Розробка back-end частини

3.2.1 Клас StockPicking

Клас StockPicking в Odoo WMS (Warehouse Management System) є ключовим компонентом для управління процесом переміщення товарів на складі. StockPicking відповідає за реєстрацію та керування всіма вхідними та вихідними операціями, пов'язаними зі складськими документами.

StockPicking включає в себе різні типи операцій, такі як прихід товарів, відправлення товарів, переміщення товарів між різними розташуваннями на складі, а також інші процеси, пов'язані зі складським управлінням. Цей клас забезпечує централізовану точку керування та контролю за всіма складськими операціями.

StockPicking має вбудовані функціональні можливості для автоматизації процесу підготовки товарів до відправлення, включаючи формування списку комплектації замовлень (picking list), резервування товарів, виділення товарів з резерву та підтвердження відправлення. Крім того, StockPicking забезпечує можливість відстежування статусу операцій, включаючи підтвердження отримання товарів та виконання доставки.

Клас StockPicking також інтегрується з іншими модулями та компонентами системи Odoo, що дозволяє забезпечити повну автоматизацію складського обліку. Наприклад, він співпрацює з модулем складського управління (Warehouse Management) для визначення параметрів складу, зберігання товарів на різних розташуваннях, керування структурою складу та іншими важливими функціями.

Узагальнюючи, клас StockPicking в Odoo WMS є незамінним елементом для автоматизації та ефективного керування складськими операціями. Він забезпечує централізоване керування рухом товарів на складі, спрощує процеси зберігання, переміщення та відправлення товарів, а також забезпечує точний облік та відстеження товарних операцій.

Для масштабування його функціоналу потрібно доповнити цей клас новими полями та функціями, які забезпечують коректну роботу WMS системи згідно з її потребами.

З нових полів, які будуть додані до класу StockPicking, можна виділити декілька важливих. Поле “wms_state” буде відповідати за статуси документи при його роботі з ТЗД(Терминал сбора данных).

```
divergences_ids = fields.One2many('mo.divergence', 'picking_id')
volume = fields.Float('Volume', compute='_cal_volume', digits='Volume')
volume_actual = fields.Float('Volume Actual', compute='_cal_val_actual', digits='Volume')
weight_actual = fields.Float('Weight Actual', compute='_cal_val_actual', digits='Stock Weight')
```

Рисунок 3.9 - Приклад додавання нових полів

Поле зв'язку “divergences_ids” між класом StockPicking та Divergence. Поля для внесення даних про розміри вантажу “volume”, “volume_actual”, “weight_actual”.

3.2.2 Клас Divergence

Клас Divergence, який прив'язаний до класу StockMove, відповідає за виявлення розбіжностей між попитом на товари та фактично отриманим товаром. Його основна мета полягає в контролі та управлінні цими розбіжностями для забезпечення точного та ефективного управління запасами.

Клас Divergence використовується для порівняння запланованого попиту на товари з фактично отриманими товарами. Він аналізує дані, що пов'язані з

класом StockMove, такі як замовлена кількість товарів, вартість, дати доставки та отримання, і порівнює їх з фактично отриманими кількостями товарів. Якщо виявляються розбіжності, клас Divergence створю відповідний запис в документі складського переміщення.

```python
class Divergence(models.Model):
    _name = 'mo.divergence'
    _description = 'Divergence in stock picking'

    product_id = fields.Many2one('product.product', string='Product')
    deviation = fields.Float('Deviation')
    demand = fields.Float('Demand')
    picking_id = fields.Many2one('stock.picking', ondelete='cascade')
    move_id = fields.Many2one('stock.move', ondelete='cascade')
```

Рисунок 3.11 - Клас Divergence

Після виявлення розбіжностей, клас Divergence може виконувати різні дії. Наприклад, він може створювати повідомлення або логи з виявленими розбіжностями, сповіщати відповідних осіб для подальшого вирішення проблем, оновлювати дані у системі або автоматично запускати процеси для усунення розбіжностей.

```python
def update_divergences(self):
    # create deviation logs
    if self.divergences_ids:
        self.divergences_ids.unlink()
    for line in self.move_ids_without_package:
        if line.picking_id.picking_type_code == 'internal':
            if line.reserved_availability != line.quantity_done:
                self.divergences_ids.create({
                    'product_id': line.product_id.id,
                    'deviation': line.deviation,
                    'picking_id': line.picking_id.id,
                    'demand': line.product_uom_qty,
                    'move_id': line.id
                })
        else:
            if line.deviation > 0 or line.deviation < 0:
                self.divergences_ids.create({
                    'product_id': line.product_id.id,
                    'deviation': line.deviation,
                    'picking_id': line.picking_id.id,
                    'demand': line.product_uom_qty,
                    'move_id': line.id
                })
```

Рисунок 3.10 - Приклад функція для створення логів при виявлені розбіжності

Клас Divergence відіграє важливу роль у забезпеченні точності та надійності управління запасами та складськими процесами. Він допомагає виявляти та вирішувати розбіжності між попитом на товари та фактично отриманими товарами, що забезпечує ефективне управління запасами, зменшення втрат та підвищення задоволеності клієнтів.

3.2.3 Клас StockMove

Клас StockMove в Odoo є важливим компонентом, відповідальним за управління рухом товарів на складі. Цей клас відображає окремі переміщення товарів між різними розташуваннями на складі і містить інформацію про деталі цих переміщень.

StockMove зберігає в собі важливі атрибути та характеристики, такі як ідентифікатор переміщення, код товару, кількість, розташування відправлення та отримання, статус переміщення та інші відомості. Цей клас забезпечує точний облік та контроль за рухом товарів на складі, що дозволяє ефективно управляти запасами і виконувати різні операції з ними.

StockMove дозволяє виконувати різноманітні дії, пов'язані з переміщенням товарів, такі як прихід товарів на склад, відправлення товарів клієнтам, переміщення товарів між різними розташуваннями на складі, повернення товарів та інші операції. Цей клас також забезпечує відстеження статусу кожного переміщення, що дозволяє контролювати процес та забезпечувати точність виконання операцій.

StockMove взаємодіє з іншими модулями та компонентами Odoo, такими як клас StockPicking, для забезпечення повної автоматизації складського управління. Цей клас інтегрується з іншими модулями, щоб забезпечити поток даних між різними складськими операціями, включаючи формування замовлень, резервування товарів, відстеження запасів, підтвердження

виконання операцій та багато іншого.

```
def _action_done(self, cancel_backorder=False):
    res = super()._action_done(cancel_backorder)
    if all(move.state == 'done' for move in self.picking_id):
        self.picking_id.write({
            'wms_state': 'done'
        })
    return res
```

Рисунок 3.12 - Приклад оновлення функції “_action_done”

В цьому класі перероблено деякі функції, наприклад “_action_done”, для коректної роботи класу у поєднанні з ТЗД та принципами WMS систем.

3.2.4 Клас StockMoveLine

Клас StockMoveLine в Odoo WMS (Warehouse Management System) є ключовим компонентом, відповідальним за керування лініями переміщень товарів на складі.

StockMoveLine представляє окремі лінії переміщення товарів. Кожен екземпляр класу StockMoveLine містить інформацію про конкретну лінію переміщення, включаючи ідентифікатор переміщення, товар, кількість одиниць товару, місцезнаходження початку та кінця переміщення, статус та інші атрибути, необхідні для відстеження переміщення товарів.

StockMoveLine дозволяє відстежувати кожну лінію переміщення товарів на складі. Цей клас забезпечує контроль за кількістю переміщуваних одиниць товару, визначає місцезнаходження початку та кінця переміщення, контролює статус переміщення та забезпечує точність виконання складських операцій.

В класі StockMoveLine в Odoo WMS були внесені зміни для поліпшеної взаємодії з ТСД.

Ці зміни включали перероблення основних функцій, пов'язаних зі збором даних на складі за допомогою ТСД. Було додано нові методи та розширено функціональність класу StockMoveLine, щоб забезпечити безперервний та точний збір даних з ТСД.

```python
@api.model_create_multi
def create(self, vals_list):
    line_picking = []
    for vals in vals_list:
        if 'qty_done' in vals and vals.get("picking_id", False):
            line_picking.append(vals['picking_id'])
            break

    if line_picking:
        self.env['stock.picking'].browse(line_picking).write({
            'wms_state': 'in_work'
        })

    res = super().create(vals_list)
    return res
```

Рисунок 3.13 - Приклад переробки функції “create”

В результаті цих змін, клас StockMoveLine міг взаємодіяти з ТСД для отримання інформації про переміщення товарів, сканування штрих-кодів, введення кількості товару та оновлення статусу переміщення. Це дозволило забезпечити швидкий та точний збір даних на складі, покращити ефективність роботи та знизити можливість помилок.

3.2.5 Клас ProductProduct

Клас ProductProductє ключовим компонентом, відповідальним за управління та інформацію про товари на складі.

ProductProduct відображає окремі товари або продукти, які знаходяться на складі. Кожен екземпляр класу ProductProduct представляє конкретний товар і

містить інформацію про його атрибути та характеристики, такі як назва товару, код, опис, ціна, одиниці виміру, категорія, доступність та інші важливі дані.

ProductProduct забезпечує повну інформацію про кожен товар на складі. Цей клас дозволяє відстежувати запаси товарів, включаючи кількість доступних одиниць, місцезнаходження товару на складі, статус (наприклад, в наявності, заблокований, замовлений тощо) та інші важливі дані, необхідні для ефективного управління запасами.

Клас ProductProduct взаємодіє з різними модулями та компонентами Odoo, зокрема з класом StockMove і StockMoveLine, для забезпечення повної автоматизації процесу переміщення та управління товарами на складі. Він дозволяє виконувати операції, пов'язані з товаром, такі як створення замовлень, отримання, відправлення, переміщення, резервування, повернення товарів та інші складські операції.

В класі ProductProduct в Odoo WMS були додані нові поля: “height”, “width”, “length”. Ці поля були введені для доповнення стандартів WMS та надання більш детальної інформації про товари на складі.

```
class Product(models.Model):
    _inherit = "product.product"

    height = fields.Float('Height', digits='Size')
    width = fields.Float('Width', digits='Size')
    depth = fields.Float('Depth', digits='Size')
```

Рисунок 3.14 - Нові поля у класі ProductProduct

Додавання полів висоти, ширини і довжини дозволяє більш точно визначати фізичні розміри товару. Ця інформація є важливою для оптимізації використання простору на складі, розташування товарів у палетах або контейнерах, а також для визначення вартості доставки на основі об'єму товару.

3.2.6 Клас StockQuant

Клас StockQuant в Odoo відіграє важливу роль у відстеженні та контролі за запасами товарів на складі.

StockQuant представляє окремі кількісні показники або кількості товарів, які знаходяться на складі. Кожен екземпляр класу StockQuant містить інформацію про конкретний товар, його місцезнаходження на складі, кількість одиниць товару та інші атрибути, необхідні для відстеження запасів.

StockQuant дозволяє точно відстежувати кількість товарів на складі. Цей клас забезпечує оновлення кількості товару при різних складських операціях, таких як отримання товарів, відправлення, переміщення, повернення та інші. Він допомагає визначати доступну кількість товару, забезпечує відстеження змін у запасах, контроль за виконанням операцій та забезпечує точність управління запасами.

В класі StockQuant було додано нове поле та функція для розрахунку зайнятого об'єму в ячейці “quant_volume”.

```
quant_volume = fields.Float('Quant Volume', compute='_compute_volume', store=True, digits='Volume')

@api.depends('available_quantity')
def _compute_volume(self):
    for item in self:
        item.quant_volume = item.product_id.volume * item.available_quantity
```

Рисунок 3.15 - Поле та функція для розрахунку зайнятого об'єму в ячейці

Це поле було введено для забезпечення точного виміру та контролю об'єму, який займає кожна позиція товару на складі.

3.2.7 Клас StockInventory

Клас StockInventory є важливим компонентом, відповідальним за інвентаризацію складських запасів та підтримку точності даних на складі.

StockInventory дозволяє здійснювати періодичні інвентаризації запасів, підраховувати фактичну кількість наявних товарів на складі та порівнювати її зі збереженою в системі кількістю. Це допомагає виявляти та виправляти розбіжності, що можуть виникати між фактичними запасами та обліковими даними.

Клас StockInventory надає можливості для створення нових інвентаризаційних операцій, включаючи вибір складських позицій, підрахунок наявної кількості, внесення корекцій та оновлення даних запасів. Він також надає засоби для аналізу розбіжностей, генерації звітів та забезпечення зручного інтерфейсу для користувачів, які здійснюють інвентаризацію.

Застосування класу StockInventory дозволяє підвищити точність та надійність даних про запаси на складі, сприяє виявленню та вирішенню розбіжностей, покращує контроль за складськими запасами та допомагає забезпечити належний рівень обліку запасів в системі.

В класі StockInventory були внесені зміни, спрямовані на поліпшення взаємодії з ТСД (Терміналами збору даних) під час інвентаризації складських запасів.

```python
class Inventory(models.Model):
    _inherit = "stock.inventory"

    wms_state = fields.Selection([('draft', 'Draft'),
                                  ('in_work', 'In Work'),
                                  ('done', 'Done'),
                                  ('cancel', 'Canceled')], string="WMS State", default='draft', tracking=True,
                                 copy=False)

    def action_cancel_draft(self):
        super().action_cancel_draft()
        self.write({
            "wms_state": 'draft'
        })
```

Рисунок 3.16 - Приклад нового поля та перероблення функції “action_cancel_draft”

Ці зміни включали перероблення основних функцій класу StockInventory та додавання нових полів для забезпечення сумісності та інтеграції з ТСД.

3.3 Розробка front-end частини

3.3.1 Створення нових відображень

Розробка нових відображень у Odoo за допомогою XML є потужним інструментом, який дозволяє створювати кастомні інтерфейси та змінювати вигляд і поведінку системи. Використання XML дозволяє зручно та ефективно описувати структуру елементів інтерфейсу, їх властивості та взаємодію з користувачем.

При створенні нових відображень в Odoo з використанням XML, ви можете визначити різні компоненти і їх поведінку. Наприклад, ви можете визначити нові форми, списки, кнопки, поля введення та інші елементи інтерфейсу, використовуючи відповідні теги та атрибути.

Для кастомізації вигляду елементів ви можете використовувати CSS-стилі, які вбудовані в XML-файли. Це дозволяє задавати кольори, розміри, шрифти та інші властивості для створених вами елементів.

Крім того, ви можете використовувати Python-код у ваших XML-файлах для реалізації бізнес-логіки та динамічної зміни відображення. Це дає можливість здійснювати розрахунки, валідацію даних, виконувати запити до бази даних та багато іншого.

Щоб використовувати ваші нові відображення в Odoo, вам потрібно буде встановити їх як модуль і активувати в системі. Після цього ви зможете

використовувати створені вами елементи інтерфейсу у різних модулях та взаємодіяти з ними відповідно до вашої бізнес-логіки.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <record id="mo_set_arrive_zone_form_view" model="ir.ui.view">
        <field name="name">mo.set.arrive.zone.form</field>
        <field name="model">mo.set.arrive.zone</field>
        <field name="arch" type="xml">
            <form>
                <group>
                    <field name="parent_location_id_comp" invisible="1"/>
                    <field name="arrive_zone" required="1"/>
                    <field name="car_and_trailer_number"/>
                    <field name="ttn"/>
                    <field name="order"/>
                    <field name="parent_location_id" invisible="1"/>
                    <field name="euro_pallet_count_arrive"/>
                    <field name="other_pallet_count_arrive"/>
                </group>
                <footer>
                    <button name="set_arrive_zone" string="Apply" type="object" class="btn-primary"/>
                    <button string="Cancel" class="btn-secondary" special="cancel"/>
                </footer>
            </form>
        </field>
    </record>

    <record id="mo_set_arrive_zone_form_wizard" model="ir.actions.act_window">
        <field name="name">Set Arrive Zone</field>
        <field name="type">ir.actions.act_window</field>
        <field name="res_model">mo.set.arrive.zone</field>
        <field name="view_mode">form</field>
        <field name="view_id" ref="mo_set_arrive_zone_form_view"/>
        <field name="target">new</field>
    </record>

</odoo>
```

Рисунок 3.17 - Приклад створення нового відображення

Загалом, розробка нових відображень у Odoo за допомогою XML є гнучким та потужним засобом для створення кастомних інтерфейсів та налаштування системи під ваші потреби. Ви можете змінювати та доповнювати функціональність системи, забезпечуючи зручне та ефективне використання Odoo у вашому бізнесі.

3.3.2 Доповнення існуючих елементів за допомогою XPath

Доповнення існуючих елементів за допомогою XPath в Odoo дозволяє вибрати та модифікувати конкретні елементи інтерфейсу або їх властивості, що вже присутні у системі.

За допомогою XPath можна точно визначити шлях до потрібного елемента у структурі XML-документа Odoo. Використовуючи XPath-вирази, можна вибрати елементи за їх ім'ям, атрибутами, значеннями або певними умовами. Це дозволяє знайти потрібний елемент або групу елементів для подальшої модифікації.

Після вибору потрібного елемента за допомогою XPath, можна змінювати його властивості, додавати нові атрибути, змінювати текстові значення або навіть видаляти елементи. Також можна використовувати XPath для вибору батьківських елементів та змінювати їх властивості або структуру.

```
<xpath expr="//page[@name='note']" position="before">
    <page string="Divergences" name="divergences">
        <button name="update_divergences" position="inside" type="object">Update</button>
        <field name="divergences_ids">
            <tree decoration-info="deviation &gt; 0" decoration-danger="deviation &lt; 0">
                <field name="product_id"/>
                <field name="demand"/>
                <field name="deviation"/>
                <field name="picking_id" invisible="1"/>
                <field name="move_id" invisible="1"/>
            </tree>
        </field>
    </page>
</xpath>
```

Рисунок 3.18 - Приклад використання Xpath

Це дозволяє розширювати та налаштовувати існуючі елементи інтерфейсу у системі Odoo, забезпечуючи більш точне та гнучке керування їх виглядом і поведінкою. Можна змінювати розміщення елементів, додавати нові

кнопки, поля введення, таблиці або будь-які інші елементи, які відповідають вашим потребам.

Використання XPath для доповнення існуючих елементів в Odoo дозволяє швидко та зручно налаштовувати систему під вимоги без потреби великих змін в основному коді. Це робить розширення та налаштування Odoo більш простими та ефективними.

3.4 Розробка інтерфейсу для ТСД

3.4.1 Інтерфейс ТСД

Створення інтерфейсу для ТСД на Vue.js в Odoo відкриває безліч можливостей для налаштування та оптимізації роботи зі складськими операціями.

Щоб почати розробку інтерфейсу на Vue.js в Odoo, вам потрібно створити новий модуль. У цих файлах буде описано компоненти, шаблони та логіку, які будуть використовуватися для створення інтерфейсу ТСД.

Використовуючи Vue.js, ви можете створювати компоненти, такі як форми, таблиці, кнопки та інші елементи інтерфейсу. Можна використовувати шаблони Vue.js для відображення даних та зв'язування їх зі змінними та функціями в JavaScript.

Одна з ключових переваг Vue.js - це двостороннє зв'язування даних (two-way data binding), яке дозволяє автоматично оновлювати дані на інтерфейсі при їх зміні, а також змінювати дані, коли користувач взаємодіє з елементами інтерфейсу. Це робить процес розробки інтерфейсу для ТСД більш зручним та ефективним.

Для взаємодії зі складськими операціями та виконання різних дій на ТСД, використовуються API-запити до серверної частини Odoo. Це дозволяє

отримувати та надсилати дані з/на сервер, оновлювати стан інтерфейсу та виконувати різні операції безпосередньо на ТСД.

Щоб додати новий компонент в Odoo з використанням Vue.js, спочатку вам потрібно створити відповідний файл компонента.

У шаблоні потрібно визначити HTML-структуру компонента, використовуючи синтаксис Vue.js. У скрипті компонента описана логіка компонента, включаючи дані, методи, обробники подій та обчислювальні властивості.

```
Vue.component("pbox-check-actions", {
    props: ["product_template", "products", "moves", "box_qty", "picking_id", "batch_id"],
    data() {
        return {
            dialog: false,
        };
    },
    methods: {
        handle_action(action) {
            const box_qty = this.$children[0].$children[1].$children[0].$children[0].value
            this.$emit(action, this._props.product_template.id, this._props.products, this._props.moves, this._props.picking_id, this._props.batch_id, box
            this.dialog = false;
        },
    },
    template: `
  <div class="batch-picking-line-actions">
    <v-dialog v-model="dialog"  tile class="actions text-center">
      <template v-slot:activator="{ on }">
        <div class="button-list button-vertical-list full">
          <v-row class="actions bottom-actions">
            <v-col class="text-center" cols="12">
              <btn-action v-on="on">{{ $t('btn.mo_create_box.title') }}</btn-action>
            </v-col>
          </v-row>
        </div>
      </template>
      <v-card>
        <div class="button-list button-vertical-list full">
          <v-row align="center">
            <v-col class="text-center" cols="12">
                  {{ $t("screen.pbox_check.box_qty") }}
            </v-col>
          </v-row>
          <v-row align="center">
            <v-col class="text-center" cols="12">
                  <input-number-spinner
                    :class="'number-spinner'"
                    :mode="'s1'"
```

Рисунок 3.19 - Приклад додавання нового компонента в Vue.js

Виклик функції Python при скануванні на ТСД в Odoo з використанням Vue.js реалізований через використання API-запитів та механізму взаємодії між фронтендом (Vue.js) та бекендом (Python).

Основна ідея полягає в тому, що можна налаштувати реакцію на події сканування на ТСД, наприклад, при натисканні кнопки або введенні даних у

відповідне поле. При спрацюванні події, викликається функція в JavaScript, яка буде відправляти запит на серверну частину Odoo.

На серверній стороні Odoo визначено відповідний контролер Python, який оброблятиме отримані запити. У цьому контролері описано функції, які будуть виконуватися при отриманні запиту від фронтенду. Ці функції можуть виконувати необхідні дії на сервері, наприклад, обробляти скановані дані, здійснювати пошук у базі даних, оновлювати статуси тощо.

```
on_scan: (scanned) => {
    this.wait_call(
        this.odoo.call("scan_line", {
            picking_batch_id: this.current_batch().id,
            move_line_id: this.state.data.id,
            barcode: scanned.text,
        })
    );
},
```

Рисунок 3.20 - Приклад виклику функції при скануванні на ТСД

Після виконання дій на сервері до фронтенд частини повиртається відповідь зазвичай у вигляді JSON-об'єкта. У JavaScript отримані відповідь обробляється та оновлює дані на екрані ТСД.

3.4.2 Обробка даних від ТСД в бекенд

В бекенд-частині ТСД в Odoo на Python реалізовано логіку обробки подій та взаємодії з ТСД за допомогою контролерів та моделей.

Принцип роботи бекенду побудовано на контролерах Python, які відповідають за обробку запитів з ТСД. Описано та змодельовані різні методи, що відповідають на різні події на ТСД, такі як сканування штрих-коду, введення

даних або натискання кнопок.

```
def scan_line(self, picking_batch_id, move_line_id, barcode):
    batch = self.env["stock.picking.batch"].browse(picking_batch_id)
    if not batch.exists():
        return self._response_batch_does_not_exist()
    move_line = self.env["stock.move.line"].browse(move_line_id)
    if not move_line.exists():
        return self._pick_next_line(
            batch, message=self.msg_store.operation_not_found()
        )

    search = self._actions_for("search")

    picking = move_line.picking_id

    package = search.package_from_scan(barcode)
    if package and move_line.package_id == package:
        return self._scan_line_by_package(picking, move_line, package)

    # use the common search method so we search by packaging too
    product = search.product_from_scan(barcode)
    if product and move_line.product_id == product:
        return self._scan_line_by_product(picking, move_line, product)
```

Рисунок 3.21 - Приклад функції для обробки даних ТСД при скануванні

У цих методах можна виконати різні дії відповідно до потреб WMS системи. Наприклад, можна виконати пошук товару за сканованим штрих-кодом, перевірити доступність товару на складі, оновити статуси складських операцій або зберегти дані в базі даних.

3.4.3 Валідація даних від ТСД в бекенд

Валідація даних, отриманих від ТСД є важливим етапом у процесі обробки та перевірки інформації, що надходить зі сканера або інших пристроїв на серверній стороні системи управління складом.

Валідація даних від ТСД допомагає забезпечити, що інформація, яка надходить у систему WMS, відповідає заданій схемі або формату, що покращує якість та цілісність даних, а також унеможливлює помилкові або некоректні дані.

```
class ShopfloorSchemaAction(Component):

    _inherit = "shopfloor.schema.action"

    def package(self, with_packaging=False):
        schema = super().package(with_packaging=with_packaging)
        schema['shopfloor_closed'] = {"type": "boolean", "required": False, "nullable": True}
        return schema

    def move_lines_counters(self):
        return {
            "lines_count": {"type": "float", "required": False, "nullable": True},
            "packages_count": {"type": "float", "required": False, "nullable": True},
            "use_count_packages": {"type": "boolean", "required": False, "nullable": True},
            "picking_count": {"type": "float", "required": False, "nullable": True},
            "priority_lines_count": {
                "type": "float",
                "required": False,
                "nullable": True,
            },
            "priority_picking_count": {
                "type": "float",
                "required": False,
                "nullable": True,
            },
        }

    def printed_product_box_for_list(self):
```

Рисунок 3.22 - Приклад схем валідації даних

Підхід який було використано за допомогою методу схем JSON. Цей метод дозволяє перевірити, чи відповідають отримані дані заданій схемі або формату

При отриманні даних від ТСД на серверній стороні WMS перевірка проводитися за допомогою вбудованих можливостей Python таких, як формат даних dict(словник) або бібліотеки json

3.5 Огляд системи

3.5.1 Огляд інтерфейсу ТСД

Інтрфейс ТСД придставляє собої меню яке складається з основних пунктів прием товару, відвантаження, комплектація та інших операцій пов'язаних с перемещением товару в рамках WMS системи.

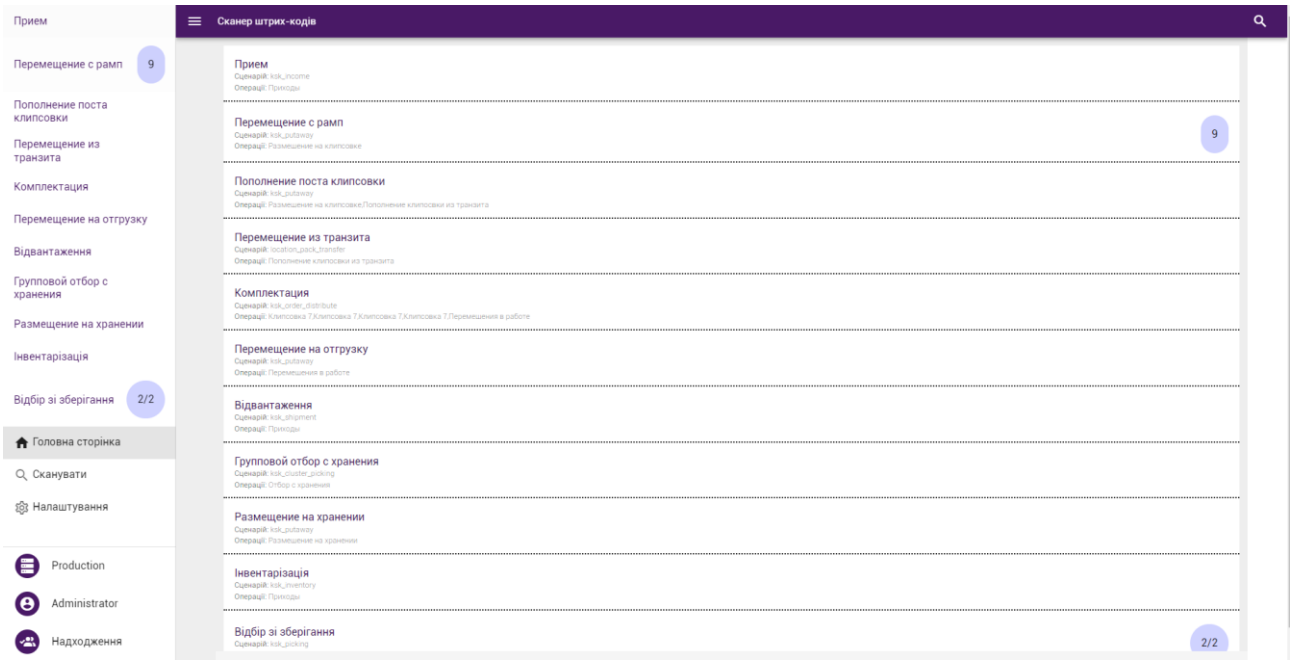


Рисунок 3.23 - Інтерфейс сканування

Обравши будь-яку операцію, наступним кроком є сканування документа, товара або пакунка залежно від обраного типу операції

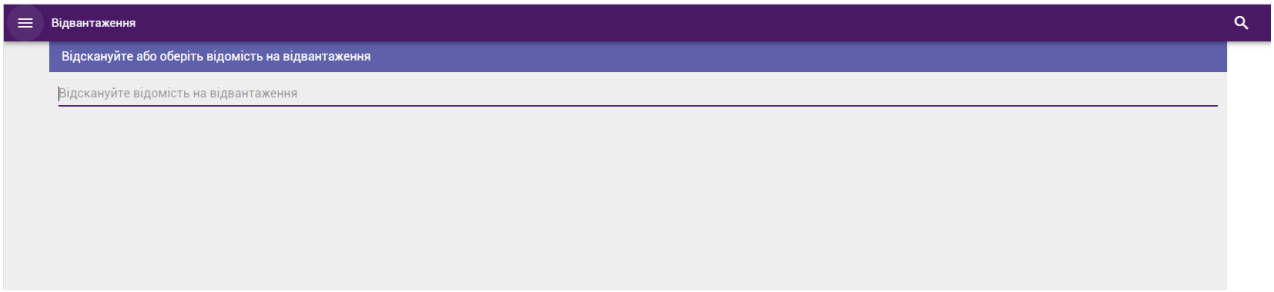


Рисунок 3.24 - Інтерфейс сканування

Після сканування відповідного коду, наприклад документу прийому товару, відкривається наступний пункт в якому є можливість виконати всі доступні дії в рамках обраної операції. Можна подивитись товар, який буде прийнято, обробити кількості та підтвердити документ.

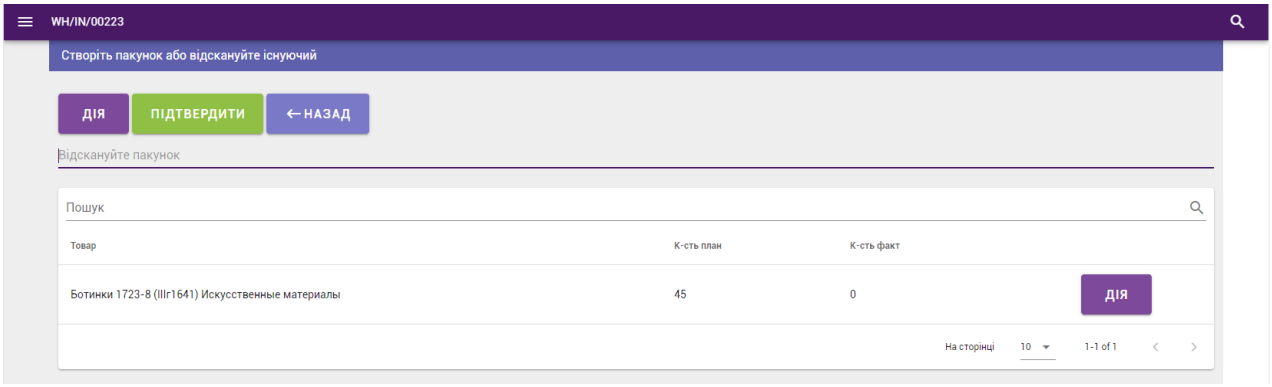


Рисунок 3.23 - Приклад успішного сканування документа

Інтерфейс ТСД також надає можливість переглядати інформацію про товари, їх характеристики, кількість на складі та інші важливі дані.

3.5.2 Огляд інтерфейсу системи

В WMS системі основним документом є документ переміщення. Цей документ використовується для фіксації переміщення товарів з одного місця на складі до іншого. Він є ключовим елементом управління логістичними операціями на складі і дозволяє контролювати рух товарів та оновлювати інформацію про запаси Документ переміщення може бути створений вручну оператором складу або автоматично системою WMS на підставі різних подій, таких як отримання нових поставок, видача замовлень або переміщення товарів для оптимізації розташування на складі. Після створення документу переміщення, він може бути направлений на виконання до робочих місць або пристроїв, які забезпечують фізичне переміщення товарів на складі.

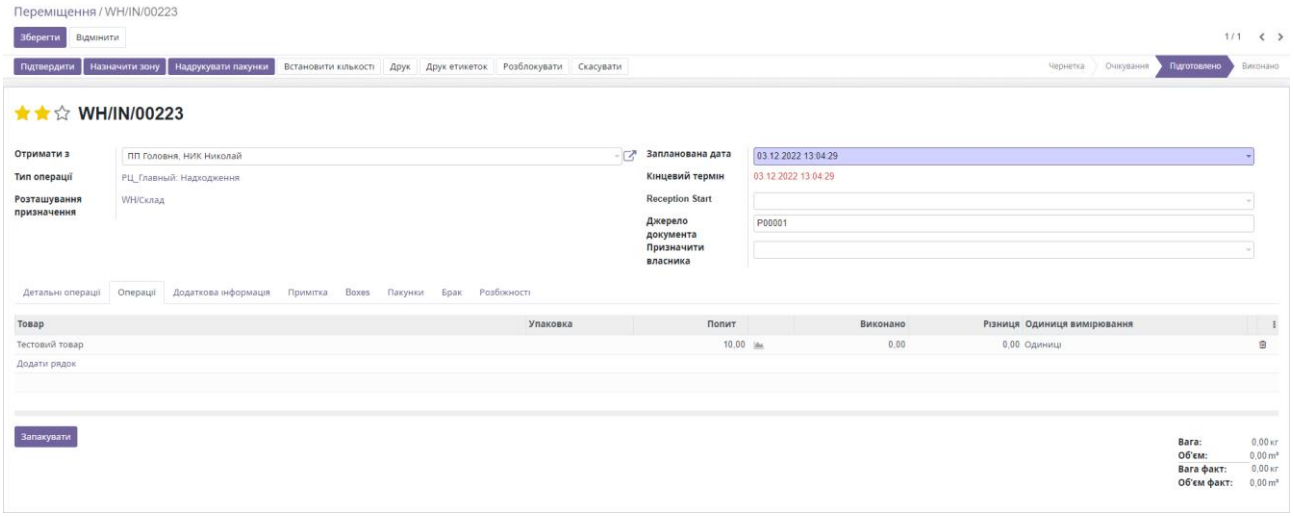


Рисунок 3.23 - Інтерфейс документу перемещения

Карточка товару є важливою складовою WMS системи і дозволяє зберігати та управляти інформацією про кожен конкретний товар на складі. Вона містить докладні дані про товар, що допомагають в управлінні запасами та оптимізації логістичних процесів. Основна мета карточки товару - забезпечити точну та повну інформацію про товар для ефективного управління ним на складі.

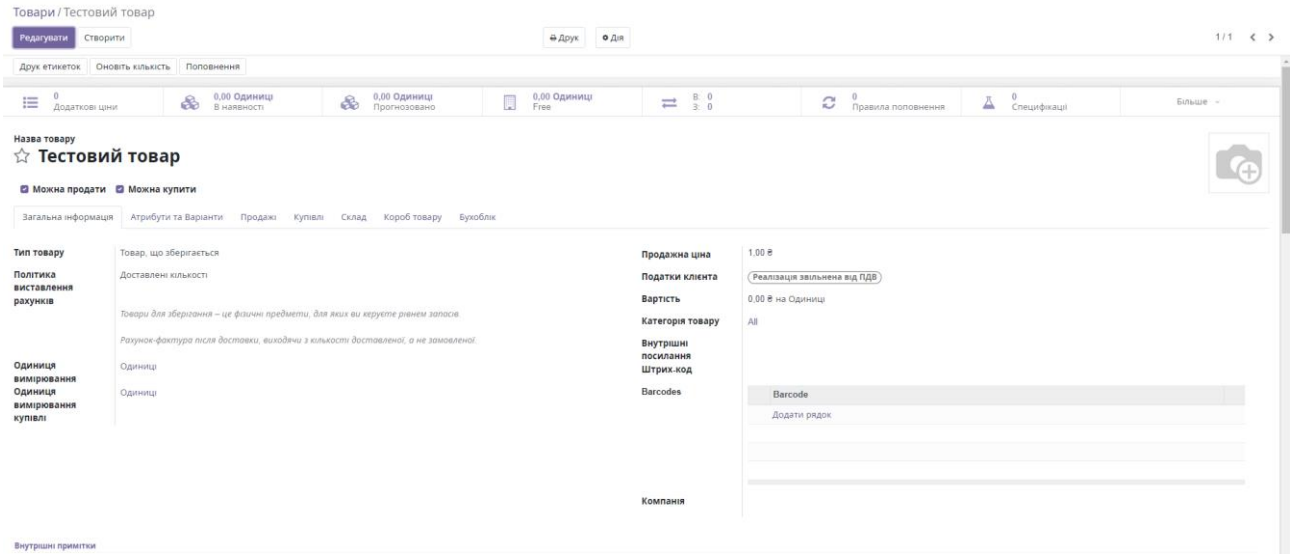


Рисунок 3.23 - Інтерфейс карточки товара

Щоб побачити повну інформацію об товарі, потрібно перейти в карту відповідного товару. Інтерфейс дозволяє побачити всю необхідну інформацію та за необхідності налаштувати її.

3.6 Тестування системи

В Odoo є власний інструмент для проведення тестів, який називається Odoo Testing Framework. Цей фреймворк надає розширені можливості для автоматизованого тестування функціональності, інтеграції та продуктивності.

Odoo Testing Framework дозволяє створювати тести, які перевіряють правильність роботи модулів, включаючи функціональність, правила бізнес-логіки, інтерфейс користувача та інші аспекти системи. Він підтримує автоматичну генерацію тестових даних, запуск тестів у режимі "одного кліку" та зручне управління тестовими наборами.

Odoo Testing Framework забезпечує широкий спектр можливостей для написання тестів, включаючи створення тестових сценаріїв, перевірку очікуваних результатів, маніпулювання даними, перехоплення помилок та багато іншого. Він також надає можливість проводити тести як на рівні модулів, так і на рівні системи в цілому.

Використання Odoo Testing Framework допомагає забезпечити якість та стабільність системи, зменшити кількість помилок та ризик неправильної роботи функціональності під час розгортання нових модулів або оновлення системи.

Загальне використання цього інструменту сприяє поліпшенню розробки та підтримки Odoo, допомагає забезпечити високу якість програмного забезпечення і покращує довіру користувачів до системи.

```python
class TestActionsChangePackageLot(CommonCase):
    """Tests covering changing a package on a move line"""

    @classmethod
    def setUpClass(cls):
        super().setUpClass()
        with cls.work_on_actions(cls) as work:
            cls.change_package_lot = work.component(usage="change.package.lot")

    @classmethod
    def setUpClassVars(cls):
        super().setUpClassVars()
        cls.wh = cls.env.ref("stock.warehouse0")
        cls.picking_type = cls.wh.out_type_id

    def _create_picking_with_package_level(self, packages):
        picking_form = Form(self.env["stock.picking"])
        picking_form.partner_id = self.customer
        picking_form.origin = "test"
        picking_form.picking_type_id = self.picking_type
        picking_form.location_id = self.stock_location
        picking_form.location_dest_id = self.packing_location
        for package in packages:
            with picking_form.package_level_ids_details.new() as move:
                move.package_id = package
        picking = picking_form.save()
        picking.action_confirm()
        picking.action_assign()
        return picking
```

Рисунок 3.23 - Приклад тест-кейсу

Для запуску тестування потрібно запустити локальний сервер додавши нові параметри. Приклад таких праметрів: `-d {назва_бази_даних} -i {назва_модулю} --test-enable`

```
2023-06-04 15:09:48,982 1636 INFO diplom unittest.suite: ======================================================================
2023-06-04 15:09:48,982 1636 ERROR diplom unittest.suite: ERROR: setUpClass (odoo.addons.shopfloor_packing_info.tests.test_checkout_scan_line.CheckoutScanLineCase)
Traceback (most recent call last):
```

Рисунок 3.23 - Невдалий результат тестування тест-кейсу

Для тестування інтерфейсу ТСД, введемо свідомо неправильні дані, щоб перевірити що система не знайде такий документ і не спинить процес роботи

системи

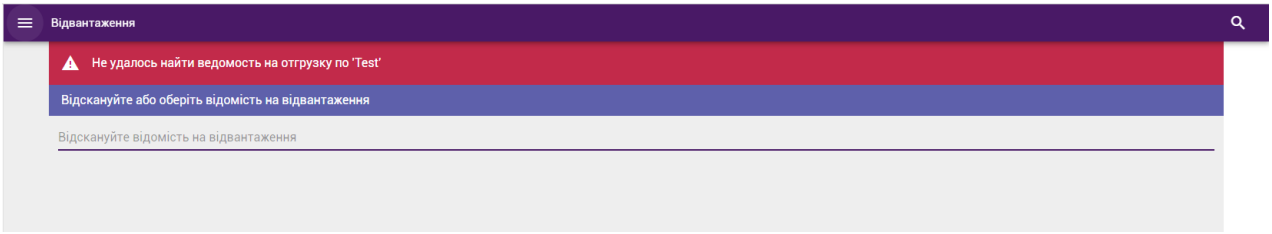


Рисунок 3.23 - Невдалий результат сканування

Вдалий результат тестування в WMS системи Odoo означає, що система працює без помилок і відповідає вимогам та очікуванням стандартів WMS систем.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі було розглянуто інтерфейс терміналів збору даних (ТСД) в контексті системи управління складом (WMS) на платформі Odoo. Були оглянуті класи та модулі, що використовуються для реалізації інтерфейсу ТСД, а також бекенд-частини, яка взаємодіє з ТСД та обробляє події.

При розробці бекенд-частини для ТСД на платформі Odoo було використано мову програмування Python. Були створені контролери та моделі, які відповідають за обробку подій, отримання даних з ТСД, виконання необхідних операцій на сервері та взаємодію з базою даних.

Тестування розробленої функціональності є важливим етапом в процесі розробки. Під час тестування було перевірено правильність роботи інтерфейсу ТСД, коректність обробки подій та взаємодії з серверною частиною. Також важливо впевнитись, що дані, які надходять з ТСД, правильно обробляються і відображаються в системі.

Також було розглянуто та доповнено основі класи та моделі WMS системи. Ці класи включають StockPicking, StockMove, StockMoveLine, ProductProduct і StockQuant. Кожен з цих класів виконує певну роль у процесі автоматизації складського обліку і допомагає забезпечити ефективну роботу на складі. Доповнення основних класів новим полями, перероблення основних функцій та додавання нових дозволяє підняти рівень системи до стандартів WMS.

ВИСНОВОК

У даній дипломній роботі було проведено дослідження та розроблено систему автоматизації складського обліку на платформі Odoo. Основною метою роботи було покращення ефективності та точності управління запасами, замовленнями, отриманням та відправленням товарів в підприємстві та створення інтерфейсу для ТСД.

У ході роботи було проведено огляд існуючих систем та алгоритмів автоматизації складського обліку, а також вивчено технології та інструменти для розробки на платформі Odoo.

Результатом роботи є розроблена система автоматизації складського обліку, яка інтегрується з платформою Odoo та забезпечує оптимізацію роботи складського відділу та підвищення загальної продуктивності підприємства. Система дозволяє ефективно керувати запасами, виконувати замовлення, контролювати отримання та відправлення товарів, що сприяє підвищенню якості обліку та зниженню помилок.

Для розробки системи були використані такі технології, як Python, Odoo Framework, Vue.js, XML та QWeb. Крім того, проведено тестування системи з використанням різних методик, що дозволило перевірити правильність функцій та взаємодії компонентів системи.

Загальний висновок полягає в тому, що розроблена система автоматизації складського обліку на платформі Odoo відповідає поставленим завданням та сприяє покращенню ефективності складських процесів у підприємстві. Її впровадження може призвести до позитивних результатів, таких як зниження часу та витрат на облік, підвищення точності та надійності, а також забезпечення більш ефективного управління запасами та покращення загальної продуктивності підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лапчук А. Що таке ERP-система та як вона допоможе вашому бізнесу? [Електронний ресурс] / Анна Лапчук // Дія Бізнес. – 2021. – Режим доступу до ресурсу: https://business.diia.gov.ua/cases/sistematizacia-biznes-procesiv/so-take-erp-sistema-ta-ak-vona-dopomoze-vasomu-biznesu.
2. WMS система: як це працює? [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://ssk.ua/blog/wms-sistema-kak-eto-rabotaet-501
3. О'Лири Д. ERP-системи: вибір, впровадження, експлуатація. Сучасне планування і управління ресурсами підприємства / Дэниел О'Лири., 2004. – 271 с.
4. Ландватер Д. World Class Production and Inventory Management / Дэррил Ландватер., 1993.
5. Odoo docs [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/documentation/15.0/
6. Python documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/
7. Matt Harrison. Effective PyCharm: Learn the PyCharm IDE with a Handson Approach / Matt Harrison., 2019. – 222 с. – (Metasnake)
8. O. Obe R. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database / Regina O. Obe., 2017. – 314 с.
9. JetBrains [Електронний ресурс] - – Режим доступу до ресурсу: https://www.jetbrains.com/
10. About JavaScript standards [Електронний ресурс] — Режим доступу: https://www.w3schools.com/js/js_es6.asp
11. Moss G. Working with Odoo / Greg Moss. – Birmingham: Packt Publishing Limited, 2015. – 432 с.

12. Ричардс Г. Warehouse Management : A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse / Гвинн Ричардс., 2011.
13. Рейс Д. Odoo Development Essentials: Fast Track Your Development Skills to Build Powerful Odoo Business Applications / Дэниел Рейс., 2015.
14. vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/
15. O'Donnell J. warehouse management system (WMS) [Електронний ресурс] / Jim O'Donnell – Режим доступу до ресурсу: https://www.techtarget.com/searcherp/definition/warehouse-management-system-WMS
16. XML introduction [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction
17. Що таке трирівнева архітектура? - визначення з техопедії [Електронний ресурс] – Режим доступу до ресурсу: https://uk.theastrologypage.com/three-tier-architecture.
18. pgAdmin Docs [Електронний ресурс] – Режим доступу до ресурсу: https://www.pgadmin.org/docs/.
19. ТСД [Електронний ресурс] – Режим доступу до ресурсу: https://www.iterator.com.ua/ru/poleznye-materialy/212-tsd-terminaly-sbora-dannykh-v-voprosakh-i-otvetakh.

ДОДАТОК 1

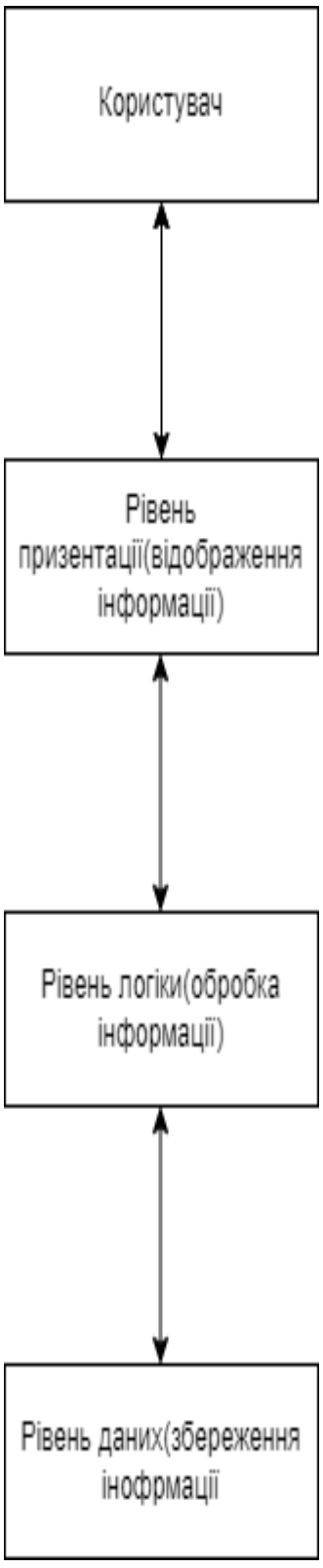
Система автоматизації складського обліку

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д1		
		№ докум.	Підпис	Дата			
Розробив		Коноплін І.В			**Система автоматизація складського обліку** Взаємодія компонентів системи(структурна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Долголенко О.М					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91	
Н. Контр.		Виноградов Ю.М					
Затвердив							

ДОДАТОК 2

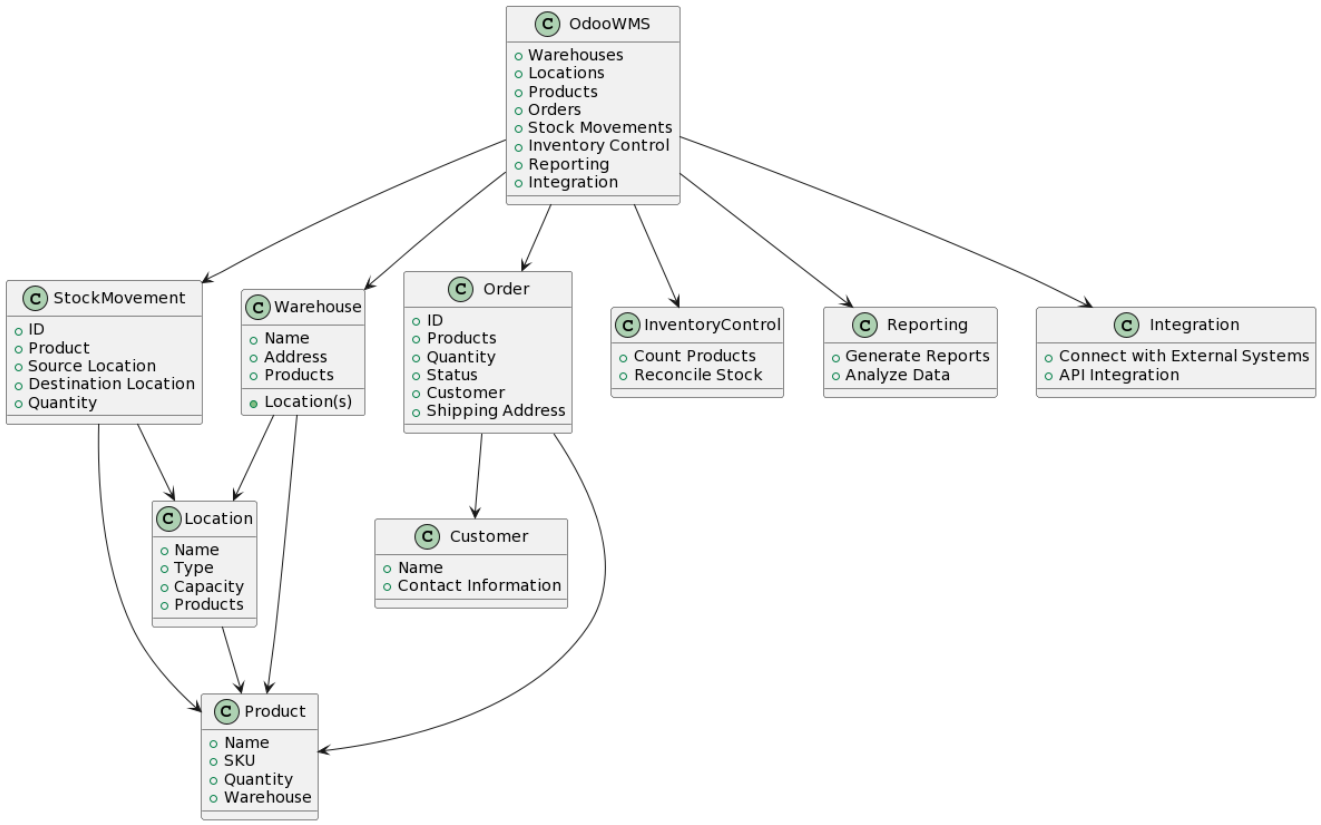
Система автоматизації складського обліку

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Коноплін І.В			**Система автоматизація складського обліку** Діаграма класів(функціональна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Долголенко О.М					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91	
Н. Контр.		Виноградов Ю.М					
Затвердив							

ДОДАТОК 3

Система автоматизації складського обліку

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2023 р

					ІАЛЦ.467200.006 Д3
		№ докум.	Підпис	Дата	
Розробив		Коноплін І.В			**Систма автоматизації складського обліку** Алгоритм дій програм-ного забезпечення(принципова схема	Літ.	Аркуш 1	Аркушів 1
Перевірив		Долголенко О.М						
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91		
Н. Контр.		Виноградов Ю.М						
Затвердив								

ДОДАТОК 4

Система автоматизації складського обліку

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 15

Київ 2023 р

Defect_divegence.py

```
from odoo import api, fields, models, _
from odoo.exceptions import ValidationError
class Defect(models.Model):
  _name = 'mo.defect'
  _description = 'Defect in stock picking'

  product_id = fields.Many2one('product.product', string='Product')
  quality_status = fields.Many2one('quality.status', string='Quality Status')
  package_id = fields.Many2one('stock.quant.package', string='Package')
  defect_count = fields.Float('Defect Number')
  picking_id = fields.Many2one('stock.picking', ondelete='cascade')
  move_id = fields.Many2one('stock.move.line', ondelete='cascade')
class Divergence(models.Model):
  _name = 'mo.divergence'
  _description = 'Divergence in stock picking'

  product_id = fields.Many2one('product.product', string='Product')
  deviation = fields.Float('Deviation')
  demand = fields.Float('demand')
  picking_id = fields.Many2one('stock.picking', ondelete='cascade')
  move_id = fields.Many2one('stock.move', ondelete='cascade')
```

Stock_move.py

```
# -*- coding: utf-8 -*-
import logging
import math
from itertools import groupby
from collections import defaultdict
from odoo import api, fields, models, _
from odoo.exceptions import UserError
from odoo.tools.float_utils import float_compare, float_is_zero, float_round
from odoo.osv import expression
from operator import itemgetter
from odoo.tools.misc import clean_context, OrderedSet
_logger = logging.getLogger(__name__)
class StockMove(models.Model):
  _inherit = "stock.move"

  glob_count = 0
  volume = fields.Float(compute='_cal_move_weight', digits='Stock Volume', store=True, compute_sudo=True)

  deviation = fields.Float(compute='_on_deviation')

  def _on_deviation(self):
    for i in self:
      i.deviation = i.quantity_done - i.product_uom_qty

  @api.depends('product_id', 'product_uom_qty', 'product_uom')
  def _cal_move_weight(self):
    # добавлен расчет объема
    super(StockMove, self)._cal_move_weight()
   moves_with_volume = self.filtered(lambda moves: moves.product_id.volume > 0)
    for move in moves_with_volume:
      move.volume = (move.product_qty * move.product_id.volume)
    (self - moves_with_volume).volume = 0

  @api.model_create_multi
  def create(self, vals_list):
```

```
    res = super().create(vals_list)
    if res.picking_id and res.picking_id.group_id:
        res.write({
            'group_id': res.picking_id.group_id.id,
            'origin': res.picking_id.origin
        })
    return res


  def _push_apply(self):
    for move in self:
        # if the move is already chained, there is no need to
check push rules
        if move.move_dest_ids:
            continue
        # if the move is a returned move, we don't want to
check push rules, as returning a returned move is the only
decent way
        # to receive goods without triggering the push rules
again (which would duplicate chained operations)
        domain = [('location_src_id', '=',
move.location_dest_id.id), ('action', 'in', ('push',
'pull_push'))]
        # first priority goes to the preferred routes defined
on the move itself (e.g. coming from a SO line)
        warehouse_id = move.warehouse_id or
move.picking_id.picking_type_id.warehouse_id
        if move.location_dest_id.company_id ==
self.env.company:
            rules =
self.env['procurement.group']._search_rule(move.route_id
s, False, move.product_id, warehouse_id,
                                                domain)
        else:
            rules =
self.sudo().env['procurement.group']._search_rule(move.r
oute_ids, False, move.product_id,
                                                   warehouse_i
d,
                                                   domain)
        # Make sure it is not returning the return


        # Make sure it is not returning the return
    moves_todo.mapped('move_line_ids').sorted()._actio
n_done()
    # Check the consistency of the result packages; there
should be an unique location across
    # the contained quants.
    for result_package in moves_todo \
            .mapped('move_line_ids.result_package_id') \
            .filtered(lambda p: p.quant_ids and
len(p.quant_ids) > 1):
        if len(result_package.quant_ids.filtered(
                lambda q: not float_is_zero(abs(q.quantity) +
abs(q.reserved_quantity),
                                    precision_rounding=q.produ
ct_uom_id.rounding)).mapped(
            'location_id')) > 1:
            raise UserError(
                _('You cannot move the same package content
more than once in the same transfer or split the same
package into two location.'))
    picking = moves_todo.mapped('picking_id')
    moves_todo.write({'state': 'done', 'date':
fields.Datetime.now()})
```

```
    move_dests_per_company = defaultdict(lambda:
self.env['stock.move'])
    for move_dest in moves_todo.move_dest_ids:
        move_dests_per_company[move_dest.company_id
.id] |= move_dest
    for company_id, move_dests in
move_dests_per_company.items():
        move_dests.sudo().with_company(company_id)._a
ction_assign()


    # We don't want to create back order for scrap moves
    # Replace by a kwarg in master
    if self.env.context.get('is_scrap'):
        return moves_todo


    # add manual cancel backorder
    if not self.env.context.get('cancel_backorder', False):
        if picking and not cancel_backorder:
            picking._create_backorder()
    return moves_todo
Stock_picking.py
from odoo import api, fields, models, SUPERUSER_ID, _
from odoo.exceptions import AccessError, UserError,
ValidationError



class StockPiking(models.Model):
  _inherit = "stock.picking"


  clips_package = fields.Many2one('stock.quant.package',
string='Clips Package',
compute='_compute_clips_package')


  def _compute_clips_package(self):
    for pick in self:
        if pick.picking_type_id.is_clips:
            if pick.move_line_ids_without_package:
                pick.clips_package =
pick.move_line_ids_without_package.mapped('package_i
d')[0]
            else:
                pick.clips_package = False
        else:
            pick.clips_package = False


  state = fields.Selection(selection_add=[('in_transit', 'In
Transit'),
                                          ('proceed', 'Proceed'),
                                          ('placing', 'Placing'),
                                          ('placed', 'Placed'),
                                          ('picking', 'Picking Transfer'),
                                          ('picked', 'Picked'),
                                          ('printed', 'Printed'),
                                          ('complecting', 'Complecting'),
                                          ('on_clarification', 'On
Clarification'),
                                          ('packing', 'Packing'),
                                          ('wrong_qty', 'Wrong
Quantities'),
                                          ('packed', 'Packed'),
                                          ('loading', 'Loading'),
                                          ('loaded', 'Loaded in
Transport'),
                                          ])
```

```
  @api.depends('move_type', 'immediate_transfer',
'move_lines.state', 'move_lines.picking_id')
  def _compute_state(self):
      old_states = {}
      for p in self:
          old_states.update({
              p: p.state
          })
          if self._context.get('avoid_recompute_pick_state',
False):
              # picking.state = old_states.get(picking)
              return
      super()._compute_state()
      for picking in self:
          if all(move.state == 'in_transit' for move in
picking.move_lines) and picking.move_lines:
              picking.state = 'in_transit'

          relevant_move_state =
picking.move_lines._get_relevant_state_among_moves()
          if relevant_move_state == 'partially_available':  #
если не хватает хоть одной штуки
              picking.state = 'confirmed'

  @api.model
  def fields_view_get(self, view_id=None,
view_type='form', toolbar=False, submenu=False):
      r = super().fields_view_get(view_id, view_type,
toolbar, submenu)
      if view_type == 'form' and 'state' in r['fields']:

          def find_item(val):
              for s in r['fields']['state']['selection']:
                  if s[0] == val:
                      return s[1]

          r['fields']['state']['selection'] = [
              ('draft', find_item('draft')),
              ('in_transit', find_item('in_transit')),
              ('waiting', find_item('waiting')),
              ('confirmed', find_item('confirmed')),
              ('assigned', find_item('assigned')),
              ('proceed', find_item('proceed')),
              ('placing', find_item('placing')),
              ('placed', find_item('placed')),
              ('picking', find_item('picking')),
              ('picked', find_item('picked')),
              ('printed', find_item('printed')),
              ('complecting', find_item('complecting')),
              ('on_clarification', find_item('on_clarification')),
              ('packing', find_item('packing')),
              ('wrong_qty', find_item('wrong_qty')),
              ('packed', find_item('packed')),
              ('loading', find_item('loading')),
              ('loaded', find_item('loaded')),
              ('done', find_item('done')),
              ('cancel', find_item('cancel')),
          ]

      return r

  group_picking_id =
fields.Many2one('mo.group.picking', 'Group Picking')
  start_date = fields.Datetime(string='Reception Start')
  car_and_trailer_number = fields.Char(string='Number
of the Car/Trailer')
  ttn = fields.Char(string='TTN')
  order = fields.Char(string='Order')
  euro_pallet_count_arrive = fields.Integer('Euro')
  other_pallet_count_arrive = fields.Integer('Other')
  noncof_act_number = fields.Char('Nonconformity Act
Number')
  volume = fields.Float('Volume',
compute='_cal_volume', digits='Volume')
  volume_uom_name = fields.Char(string='Weight unit of
measure label',
                                default=lambda self:
self.env.ref('uom.product_uom_cubic_meter').name)
  sequence_code =
fields.Char(related='picking_type_id.sequence_code',
store=True)

  divergences_ids = fields.One2many('mo.divergence',
'picking_id')
  defect_ids = fields.One2many('mo.defect', 'picking_id')

  volume_actual = fields.Float('Volume Actual',
compute='_cal_val_actual', digits='Volume')
  weight_actual = fields.Float('Weight Actual',
compute='_cal_val_actual', digits='Stock Weight')

  @api.depends('move_lines')
  def _cal_val_actual(self):
      vols = []
      weights = []
      for move in self.move_lines.filtered(lambda x: x.state
!= 'cancel'):
          vols.append(move.product_id.volume *
move.quantity_done)
          weights.append(move.product_id.weight *
move.quantity_done)
      self.volume_actual = sum(vols)
      self.weight_actual = sum(weights)

  @api.depends('move_lines')
  def _cal_volume(self):
      for picking in self:
          picking.volume = sum(move.volume for move in
picking.move_lines if move.state != 'cancel')


  def create_product_table(self, package_id=False):
      if not package_id:
          package_count =
self.move_line_nosuggest_ids.mapped('result_package_id'
).mapped('id')
      else:
          package_count = package_id
      res = []
      for i in package_count:
          package =
self.env['stock.quant.package'].browse(i).name
          by_package =
self.move_line_ids_without_package.filtered(lambda x:
x.result_package_id.id == i)
          if by_package:
              done_qty =
sum(by_package.mapped('qty_done'))
```

```
            total_weight =
sum(by_package.mapped('product_id').mapped('weight'))
            res.append([package, by_package, done_qty,
total_weight])

        return res

    def create_transport_order(self, package_id=False):
        if not package_id:
            packages =
self.move_line_ids_without_package.mapped('package_id
')
        else:
            packages = package_id
        if packages:
            for pack in packages:
                lines =
self.move_line_ids_without_package.filtered(lambda x:
x.result_package_id == pack)
                transport_order =
self.env['mo.transport.order'].search(
                    # [('package_id', '=', pack.id), ('picking_ids',
'in', self.ids)])
                    [('package_id', '=', pack.id), ('state', 'in',
['new'])])
                scrap =
self.env['stock.scrap'].search([('package_id', '=', pack.id),
('state', '=', 'draft')])
                # transport_order = False
                if scrap:
                    transport_order =
self.env['mo.transport.order'].search(
                        [('package_id', '=', pack.id), ('scrap_ids', 'in',
scrap.ids)])
                    processed_lines = []
                    for scrp in scrap:
                        if scrp.product_id.tracking not in ['none']:
                            if scrp.lot_id:
                                continue
                            ln = lines.filtered(lambda x:
x.product_id.id == scrp.product_id.id)
                            if ln:
                                for l in ln:
                                    if l.id not in processed_lines:
                                        scrp.write({
                                            'lot_id': l.lot_id.id
                                        })
                                        processed_lines.append(l.id)
                                        break


                if transport_order:
                    transport_order.write({
                        'mov_line_ids': [(4, i.id) for i in lines] if not
scrap else False,
                        'scrap_move_ids': [(4, scrp.move_id.id) for
scrp in scrap if scrp.move_id] if scrap else False,
                    })
                    if not scrap:
                        transport_order.write({
                            'picking_ids': [(4, self.id)],
                        })
                        self.write({
                            'transfer_order_ids': [(4,
transport_order.id)]
                        })
                else:
                    transport_order =
self.env['mo.transport.order'].create({
                        'mov_line_ids': [(4, i.id) for i in lines] if not
scrap else False,
                        'scrap_move_ids': [(4, scrp.move_id.id) for
scrp in scrap if scrp.move_id if
                                           scrp.move_id] if scrap else
False,
                        'package_id': pack.id,
                        'picking_ids': [(4, self.id)] if not scrap else
False,
                        'scrap_ids': [(4, scrp.id) for scrp in scrap] if
scrap else False,
                        'for_scrap': True if scrap else False
                    })
                    self.write({
                        # 'state': 'placing',
                        'transfer_order_ids': [(4, transport_order.id)]
                    })

                if self._context.get('assign_locations', False):
                    lines.reassign_location()


    def reassing_locations(self):
        if self.package_ids:
            for pack in self.package_ids:
                lines_to_reassign =
self.move_line_ids_without_package.filtered(lambda x:
x.result_package_id == pack)
                if lines_to_reassign:
                    lines_to_reassign.reassign_location()


    def button_validate(self):
        res = super(StockPiking, self).button_validate()
        if self.batch_id:
            if all(i.state == 'done' for i in
self.batch_id.picking_ids):
                self.batch_id.write({
                    'state': 'done'
                })
            else:
                if all(i.state in ['done', 'cancel'] for i in
self.group_picking_id.picking_ids):
                    self.batch_id.write({
                        'state': 'done'
                    })
        # if self.transfer_order_ids:
        #     self.transfer_order_ids.filtered(lambda x: x.state
not in ['done', 'cancelled']).validate_transport()
        return res

    def action_cancel(self):
        res = super().action_cancel()
        cancel_transport =
self.transfer_order_ids.filtered(lambda x: x.state in ['draft',
'active', 'new'])
        if cancel_transport:
            cancel_transport.cancel_transport()
        if self.batch_id:
            if all(i.state == 'cancel' for i in
self.batch_id.picking_ids):
                self.batch_id.write({
                    'state': 'cancel'
                })
```

```
        else:
            if all(i.state in ['done', 'cancel'] for i in
self.batch_id.picking_ids):
                self.batch_id.write({
                    'state': 'cancel'
                })
    # if self.group_id and not
self._context.get('auto_cancel', False):
    #     pickings =
self.group_id.stock_move_ids.mapped('picking_id').filtere
d(lambda x: x.state not in ['done', 'cancel'])
    #     pickings.with_context(auto_cancel=True).action_
cancel()

    if self.transfer_order_ids:
        self.transfer_order_ids.filtered(lambda x: x.state
not in ['done', 'cancelled']).cancel_transport()
    return res

  def action_assign(self):
    if not self._context.get('confirm_from_sale', False):
        if self.picking_type_code == 'internal' and
self.picking_type_id.sequence_code in ['CPR', 'INT']:
            self.env.context = dict(self.env.context)
            self.env.context.update({'get_all_locations':
True})

    if self._context.get('confirm_from_sale', False) or not
self._context.get('confirm_from_sale', False):
        if any(i.picking_type_id.sequence_code in ['PICK']
for i in self):
            self.env.context = dict(self.env.context)
            self.env.context.update({'picking_location':
True})

    res = super().action_assign()
    return res

  def update_defect(self, create_scrap=False):
    # create defect
    if self.defect_ids:
        self.defect_ids.unlink()
    for line in self.move_line_nosuggest_ids:
        if line.quality_status:
            self.defect_ids.create({
                'product_id': line.product_id.id,
                'quality_status': line.quality_status.id,
                'defect_count': line.defect_count,
                'picking_id': self.id,
                'move_id': line.id
            })
            scraps = self.env['stock.scrap']
            if create_scrap:
                find_lot = False
                if line.lot_name:
                    find_lot =
self.env['stock.production.lot'].search(
                        [('name', '=', line.lot_name), ('product_id',
'=', line.product_id.id)])
                    if not find_lot:
                        find_lot =
self.env['stock.production.lot'].create({
                            'product_id': line.product_id.id,
                            'name': line.lot_name,
                            'company_id': self.company_id.id,
                            'expiration_date':
str(line.expiration_date) if line.expiration_date else False
                        })
                scrap = self.env['stock.scrap'].create({
                    'product_id': line.product_id.id,
                    'scrap_qty': line.qty_done,
                    'location_id': line.location_dest_id.id,
                    'scrap_location_id':
line.quality_status.defect_zone.id,
                    'picking_id': self.id,
                    # 'move_id': line.move_id.id,
                    'product_uom_id':
line.product_id.uom_id.id,
                    'origin': line.picking_id.origin,
                    'package_id': line.result_package_id.id,
                    'owner_id': self.owner_id.id,
                    'lot_id': find_lot.id if find_lot else False
                })
                scraps |= scrap
                # move =
self.env['stock.move'].create(scrap._prepare_move_values
())
                # scrap.write({
                #     'move_id': move.id
                # })

                transport_order =
self.env['mo.transport.order'].search(
                    [('package_id', '=',
line.result_package_id.id), ('for_scrap', '=', True)])
                if transport_order:
                    transport_order.write({
                        # 'scrap_move_ids': [(4, scrp.move_id.id)
for scrp in scrap] if scrap else False,
                        'scrap_ids': [(4, scrap.id)],
                    })
                else:
                    transport_order =
self.env['mo.transport.order'].create({
                        # 'scrap_move_ids': [(4,
scrap.move_id.id)],
                        'package_id': line.result_package_id.id,
                        'picking_ids': [(4,
line.move_id.move_dest_ids.picking_id.id)],
                        'scrap_ids': [(4, scrap.id)],
                        'for_scrap': True
                    })
                    line.move_id.move_dest_ids.picking_id.wri
te({
                        'state': 'placing',
                        'transfer_order_ids': [(4,
transport_order.id)]
                    })
                # srp.action_validate()
            if scraps:
                self.env.context = dict(self.env.context)
                self.env.context.update({'scrap_create':
scraps})

  def update_divergences(self):
    # create deviation
    if self.divergences_ids:
        self.divergences_ids.unlink()
    for line in self.move_ids_without_package:
        if line.picking_id.picking_type_code == 'internal':
```

```
            if line.reserved_availability !=
line.quantity_done:
                self.divergences_ids.create({
                    'product_id': line.product_id.id,
                    'deviation': line.deviation,
                    'picking_id': line.picking_id.id,
                    'demand': line.product_uom_qty,
                    'move_id': line.id
                })
            else:
                if line.deviation > 0 or line.deviation < 0:
                    self.divergences_ids.create({
                        'product_id': line.product_id.id,
                        'deviation': line.deviation,
                        'picking_id': line.picking_id.id,
                        'demand': line.product_uom_qty,
                        'move_id': line.id
                    })

    def _check_entire_pack(self):
        """ This function check if entire packs are moved in
the picking"""
        for picking in self:
            origin_packages =
picking.move_line_ids.mapped("package_id")
            not_package_id = False
            if not origin_packages:
                if picking.picking_type_id.show_entire_packs:
                    not_package_id = True
                    origin_packages =
picking.move_line_ids.mapped("result_package_id")
            for pack in origin_packages:
                if
picking._check_move_lines_map_quant_package(pack):
                    package_level_ids =
picking.package_level_ids.filtered(lambda pl:
pl.package_id == pack)
                    if not not_package_id:
                        move_lines_to_pack =
picking.move_line_ids.filtered(
                            lambda ml: ml.package_id == pack and
not ml.result_package_id)
                    else:
                        move_lines_to_pack =
picking.move_line_ids.filtered(
                            lambda ml: ml.result_package_id ==
pack and ml.result_package_id)
                    if not package_level_ids:
                        self.env['stock.package_level'].create({
                            'picking_id': picking.id,
                            'package_id': pack.id,
                            'location_id': pack.location_id.id if not
not_package_id else
picking.picking_type_id.default_location_src_id,
                            'location_dest_id':
self._get_entire_pack_location_dest(
                                move_lines_to_pack) or
picking.location_dest_id.id,
                            'move_line_ids': [(6, 0,
move_lines_to_pack.ids)],
                            'company_id': picking.company_id.id,
                        })
                    # TODO: in master, move package field in
`stock` and clean code.
                    if pack.package_use == 'disposable':
                        move_lines_to_pack.write({
                            'result_package_id': pack.id,
                        })
                    else:
                        move_lines_in_package_level =
move_lines_to_pack.filtered(
                            lambda ml:
ml.move_id.package_level_id)
                        move_lines_without_package_level =
move_lines_to_pack - move_lines_in_package_level
                        for ml in move_lines_in_package_level:
                            ml.write({
                                'result_package_id': pack.id,
                                'package_level_id':
ml.move_id.package_level_id.id,
                            })
                        if move_lines_without_package_level:
                            move_lines_without_package_level.write
({
                                'result_package_id': pack.id,
                                'package_level_id':
package_level_ids[0].id,
                            })
                        for pl in package_level_ids:
                            pl.location_dest_id =
self._get_entire_pack_location_dest(
                                pl.move_line_ids) or
picking.location_dest_id.id
                else:
                    # try to reserve all quantities of all products in
this package
                    return
                    if picking.picking_type_id.is_clips:
                        return

                    rules =
self.env["stock.reserve.rule"]._rules_for_location(picking.
location_id)
                    if not rules:
                        return
                    if all(rule == 'default' for rule in
rules.rule_removal_ids.mapped('removal_strategy')):
                        return

                    package_level_ids =
picking.package_level_ids.filtered(lambda pl:
pl.package_id == pack)
                    move_lines_to_pack =
picking.move_line_ids.filtered(
                        lambda ml: ml.package_id == pack and not
ml.result_package_id)
                    move_lines_to_pack_prod =
move_lines_to_pack.mapped('product_id')

                    pack_products =
self.env['stock.quant'].search([('package_id', '=', pack.id),
                                                    # ('product_id',
'not in',
                                                    #  move_lines_t
o_pack_prod.ids),
                                                    ('quantity', '>',
0)])
                    if not pack_products:
                        return
```

```
                package_level =
self.env["stock.package_level"].search(
                    [
                        ("package_id", "=", pack.id),
                        ('state', '!=', 'cancel'),
                        ("picking_id", "=", picking.id),
                    ]
                )
                #
                for pack_product in pack_products:

                    find_prod =
self.env['stock.move'].sudo().search(
                        [('product_id', '=',
pack_product.product_id.id),
                         ('picking_id', '=', picking.id)])
                    if not find_prod:
                        mv =
self.env['stock.move'].sudo().create({
                            'name':
pack_product.product_id.display_name,
                            'product_uom':
pack_product.product_id.uom_id.id,
                            'picking_id': picking.id,
                            'picking_type_id':
picking.picking_type_id.id,
                            'product_id':
pack_product.product_id.id,
                            'product_uom_qty':
abs(pack_product.available_quantity),
                            # 'state': 'confirmed',
                            'location_id': picking.location_id.id,
                            'location_dest_id':
picking.location_dest_id.id,
                        })

                        mv._action_confirm()
                        mv._action_assign_custom()
                    else:
                        if pack_product.available_quantity:
                            ml =
find_prod.move_line_ids.filtered(lambda x: x.package_id
== pack)
                            if ml:

                                find_prod.write({
                                    'product_uom_qty':
abs(pack_product.available_quantity) +
find_prod.product_uom_qty
                                })
                                # find_prod._action_confirm()
                                find_prod.with_context(force_packa
ge=pack)._action_assign_custom()
                            else:
                                mv =
self.env['stock.move'].sudo().create({
                                    'name':
pack_product.product_id.display_name,
                                    'product_uom':
pack_product.product_id.uom_id.id,
                                    'picking_id': picking.id,
                                    'picking_type_id':
picking.picking_type_id.id,
                                    'product_id':
pack_product.product_id.id,
                                    'product_uom_qty':
abs(pack_product.available_quantity),
                                    # 'state': 'confirmed',
                                    'location_id':
picking.location_id.id,
                                    'location_dest_id':
picking.location_dest_id.id,
                                })

                                mv._action_confirm()
                                mv.with_context(force_package=pa
ck)._action_assign_custom()

                # if picking.move_line_ids_without_package:



    def action_confirm(self):
        super().action_confirm()
        if any(i.sale_id for i in self):
            return True
        if not self._context.get('extra_line', False) or not
any(line.additional for line in self.move_lines):
            if self.picking_type_code == 'incoming':
                self.move_lines.write({
                    'state': 'in_transit'
                })
        for line in self.move_lines:
            if line.additional:
                line.write({
                    'location_dest_id':
line.picking_id.location_dest_id
                })
        return True


    @api.model
    def create(self, vals_list):
        res = super(StockPiking, self).create(vals_list)
        return res


    def write(self, vals_list):
        res = super(StockPiking, self).write(vals_list)
        return res



    def _create_backorder(self):
        """ This method is called when the user chose to
create a backorder. It will create a new
        picking, the backorder, and move the stock.moves
that are not `done` or `cancel` into it.
        """
        backorders = self.env['stock.picking']
        for picking in self:
            moves_to_backorder =
picking.move_lines.filtered(lambda x: x.state not in
('done', 'cancel'))
            if moves_to_backorder:
                backorder_picking = picking.copy({
                    'name': '/',
                    'move_lines': [],
                    'move_line_ids': [],
                    'backorder_id': picking.id
                })
                if backorder_picking.move_lines:
                    backorder_picking.action_assign()
```

```
            picking.message_post(
                body=_(
                    'The backorder <a href=# data-oe-
model=stock.picking data-oe-id=%d>%s</a> has been
created.') % (
                        backorder_picking.id,
backorder_picking.name))
            moves_to_backorder.write({'picking_id':
backorder_picking.id})
            moves_to_backorder.mapped('package_level_id'
).write({'picking_id': backorder_picking.id})
            moves_to_backorder.mapped('move_line_ids').w
rite({'picking_id': backorder_picking.id})
            backorders |= backorder_picking
        return backorders
stock_quant.pyfrom odoo import api, fields, models,
SUPERUSER_ID, _
from collections import OrderedDict


def report_vals(self, docids, data):
    package_name = False
    try:
        if isinstance(data['package_id'], list):
            package_id = data['package_id']
        else:
            if isinstance(eval(data['package_id']), list):
                package_id = eval(data['package_id'])
            else:
                package_id = [int(data['package_id'])]

        picking_id = int(data['picking_id'])
    except Exception as err:
        if 'package_id' in data:
            package_id = [int(data['package_id'])]
        else:
            package_id = False
        if 'from_app' not in data:
            picking_id = False
        else:
            picking_id = docids

    if package_id and 'from_app' in data:
        package_name =
self.env['stock.quant.package'].browse(package_id).name

    if self._context.get('group', False) or data.get('group',
False):
        docs =
self.env['stock.picking.batch'].browse(picking_id)
        batch = True
    else:
        docs = self.env['stock.picking'].browse(picking_id)
        batch = False
    return {
        'package_id': package_id,
        'package_name': package_name,
        'docs': docs,
        'batch': batch
    }


class PickPackageReport(models.AbstractModel):
    _name = 'report.mo_wms.report_collection_template'

    @api.model
    def _get_report_values(self, docids, data=None):
        return report_vals(self, docids, data)


class
PickPackageFormSmallReport(models.AbstractModel):
    _name =
'report.mo_wms.incoming_package_template_58x30'

    @api.model
    def _get_report_values(self, docids, data=None):
        return report_vals(self, docids, data)


class
PickPackageForLargemReport(models.AbstractModel):
    _name =
'report.mo_wms.incoming_package_template_100x100'

    @api.model
    def _get_report_values(self, docids, data=None):
        return report_vals(self, docids, data)


class PackageReport(models.AbstractModel):
    _name = 'report.mo_wms.package_template_58x30_'

    @api.model
    def _get_report_values(self, docids, data=None):
        picking_name = ''
        if 'picking_id' in data:
            if isinstance(data['picking_id'], list):
                picking_id = data['picking_id']
            else:
                if isinstance(eval(data['picking_id']), list):
                    picking_id = eval(data['picking_id'])
                else:
                    picking_id = [int(data['picking_id'])]
            if self._context.get('group', False) or
data.get('group', False):
                picking_name =
self.env['stock.picking.batch'].browse(picking_id).name
            else:
                picking_name =
self.env['stock.picking'].browse(picking_id).name
        dt = {
            'picking_name': picking_name,
            'docs':
self.env['stock.quant.package'].browse(docids),
        }
        return dt


class PackageReport100(models.AbstractModel):
    _name = 'report.mo_wms.package_template_100x100_'

    @api.model
    def _get_report_values(self, docids, data=None):
        picking_name = ''
        if 'picking_id' in data:
```

```
        if isinstance(data['picking_id'], list):
            picking_id = data['picking_id']
        else:
            if isinstance(eval(data['picking_id']), list):
                picking_id = eval(data['picking_id'])
            else:
                picking_id = [int(data['picking_id'])]
        picking_name =
self.env['stock.picking'].browse(picking_id).name
        dt = {
            'picking_name': picking_name,
            'docs':
self.env['stock.quant.package'].browse(docids),
        }
        return dt


class QuantPackage(models.Model):
    _inherit = "stock.quant.package"

    def print_package_action(self):
        data = {
            'picking_id': self._context['picking_id'],
            'package_id': self._context['package_id']
        }
        if self._context.get('paper_format', False):
            if self._context['paper_format'] == 'small':
                return
self.env.ref('mo_wms.action_report_stock_picking_incom
ing_package_tmp58x38').report_action(
                    self._context['picking_id'],
                    data=data)
            else:
                return
self.env.ref('mo_wms.action_report_stock_picking_incom
ing_package_tmp100x100').report_action(
                    self._context['picking_id'],
                    data=data)
        else:
            return
self.env.ref('mo_wms.action_report_stock_picking_collec
t_tmp').report_action(self._context['picking_id'],
                                                    dat
a=data)


class StockQuant(models.Model):
    _inherit = 'stock.quant'

    location_id_vol = fields.Float('Location Volume',
related='location_id.volume', store=True)
    recycle_percent = fields.Integer('Recycling Percentage',
related='location_id.recycle_percent', store=True)
    quant_volume = fields.Float('Quant Volume',
compute='_compute_volume', store=True,
digits='Volume')
    exp_date =
fields.Datetime(related='lot_id.expiration_date',
string='Exp. Date')

    @api.depends('available_quantity')
    def _compute_volume(self):
        for item in self:
            item.quant_volume = item.product_id.volume *
item.available_quantity

    pick_location =
fields.Boolean(related='location_id.for_picking',
store=True, string='For Picking')
    storage_location =
fields.Boolean(related='location_id.for_storage',
store=True, string='For Storage')

    def _gather(self, product_id, location_id, lot_id=None,
package_id=None, owner_id=None, strict=False,
avoid_location=False):
        res = super(StockQuant, self)._gather(product_id,
location_id, lot_id, package_id, owner_id, strict)
        if not strict:
            # don't check scrap locations
            quants = self
            if self._context.get('get_all_locations', False):
                return res
            for r in res:
                if not r.location_id.scrap_location:
                    if avoid_location and r.location_id ==
avoid_location:
                        continue
                    quants |= r

            if self._context.get('storage_location', False):
                quants = quants.filtered(lambda x:
x.location_id.for_storage)
                res = quants
                return res

            # use only picking locations
            # if not self._context.get('picking_location', False):
            # if self._context.get('picking_location', False):
            #     quants = quants.filtered(lambda x:
x.location_id.for_picking)
            #     res = quants
        return res

    def _group_by_package(self):
        """Return quants grouped by packages

        Group by package, but keeping the order of the
quants (if we have more
        than one quant per package, the order is based on the
first quant seen
        in the package). Thus, it can be used on a recordset
returned by
        _gather.

        The returned format is: [(package, quants)]

        """
        seen = OrderedDict()
        for quant in self:
            package = quant.package_id
            if package in seen:
                seen[package] = seen[package] | quant
            else:
                seen[package] = quant
        return [(loc, quants) for loc, quants in seen.items()]

Inventory.py
from odoo import _
```

```
from odoo.addons.component.core import Component


class InventoryAction(Component):

  _inherit = "shopfloor.inventory.action"

  @property
  def inventory_model(self):
     # the _sf_inventory key bypass groups checks,
     # see comment in models/stock_inventory.py
     return
self.env["mo.stock.inventory"].with_context(_sf_inventor
y=True)

  def create_draft_inventory_location(self, location,
prefill_quantity='zero'):
     vals = {
        "name": _("Stock Correction in Location
{}").format(location.name),
        "location_ids": [(6, 0, location.ids)],
        "prefill_counted_quantity": prefill_quantity
     }
     return self.inventory_model.sudo().create(vals)
Schema.py
from odoo.addons.component.core import Component


class ShopfloorSchemaAction(Component):

  _inherit = "shopfloor.schema.action"

  def package(self, with_packaging=False):
     schema =
super().package(with_packaging=with_packaging)
     schema['shopfloor_closed'] = {"type": "boolean",
"required": False, "nullable": True}
     return schema

  def move_lines_counters(self):
     return {
        "lines_count": {"type": "float", "required": False,
"nullable": True},
        "packages_count": {"type": "float", "required":
False, "nullable": True},
        "use_count_packages": {"type": "boolean",
"required": False, "nullable": True},
        "picking_count": {"type": "float", "required":
False, "nullable": True},
        "priority_lines_count": {
          "type": "float",
          "required": False,
          "nullable": True,
        },
        "priority_picking_count": {
          "type": "float",
          "required": False,
          "nullable": True,
        },
     }

  def printed_product_box_for_list(self):
     schema = {
        "id": {"required": True, "type": "integer"},
        "qty": {"required": True, "type": "float"},
        "picking_id":
self._schema_dict_of(self._simple_record(),
required=True),
        "box_id":
self._schema_dict_of(self.product_box_for_list(),
required=True),
     }
     return schema

  def printed_product_box_for_list_inventory(self):
     schema = {
        "id": {"required": True, "type": "integer"},
        "qty": {"required": True, "type": "float"},
        "inventory_id":
self._schema_dict_of(self._simple_record(),
required=True),
        "box_id":
self._schema_dict_of(self.product_box_for_list(),
required=True),
     }
     return schema

  def product_box_for_list(self):
     schema = {
        "id": {"required": True, "type": "integer"},
        "barcode": {"type": "string", "nullable": False,
"required": True},
        "product_template":
self._schema_dict_of(self.product_template_simple(),
required=True),
     }
     return schema
  def picking_order_distribute(self,
with_move_lines=False, no_packaging=False):
     schema = {
        "id": {"required": True, "type": "integer"},
        "name": {"type": "string", "nullable": False,
"required": True},
        "origin": {"type": "string", "nullable": True,
"required": False},
        "note": {"type": "string", "nullable": True,
"required": False},
        "move_line_count": {"type": "integer", "nullable":
True, "required": True},
        "package_level_count": {
          "type": "integer",
          "nullable": True,
          "required": True,
        },
        "bulk_line_count": {"type": "integer", "nullable":
True, "required": True},
        "weight": {"required": True, "nullable": True,
"type": "float"},
        "partner":
self._schema_dict_of(self._simple_record()),
        "carrier":
self._schema_dict_of(self._simple_record(),
required=False),
        "ship_carrier":
self._schema_dict_of(self._simple_record(),
required=False),
        "inter_trans_id":  self._schema_dict_of(self.inter_tr
ans(), required=False, nullable=True),
```

```
        "scheduled_date": {"type": "string", "nullable":
False, "required": True},
    }

    if with_move_lines:
        schema["move_lines"] = self._schema_list_of(
            self.move_line(with_packaging=not
no_packaging)
        )

    return schema
```

income.js

```
const Income = {
  mixins: [ScenarioBaseMixin],
  template: `
    <Screen :screen_info="screen_info">
        <template v-slot:header>
            <state-display-info
info="state.display_info"/>
        </template>
        <choosing-ksk-income-contact
            v-if="state_is('start')"
            @select-contact="state.onSelectContact"
            :partners="state.data.partners"
            :fields="state.fields"
            @found="state.on_scan"
            />
        <choosing-reception-picking-batch
            v-if="state_is('choose_picking_or_batch')"
            @select-batch="state.onSelectBatch"
            @select-picking="state.onSelectPicking"
            :raw_pickings="state.data.raw_pickings"
            :batches="state.data.batches"
            :fields="state.fields"
            @found="state.on_scan"
            :screen_info="state.screen_info"
            />


        <div class="button-list button-vertical-list full"
v-if="state_is('choose_picking_or_batch')">
            <v-row align="center">
                <v-col class="text-center" cols="12">
                    <v-btn @click="$router.push({'name':
'ksk_income'})" class="secondary">
                        <v-icon>mdi-keyboard-
backspace</v-icon>
                        {{$t('btn.mo_back.title')}}
                    </v-btn>
                </v-col>
            </v-row>
        </div>


        <div v-if="state_is('pbox_check')">
            <product-box-check
                @create_box="state.create_box"
                :products="state.data.products"
                :moves="state.data.moves"
                :box_id="state.data.box_id"
                :product_template="state.data.product_t
emplate"
                :batch_id="state.data.batch_id"
                :picking_id="state.data.picking_id"
                :screen_info="state.screen_info"
                :package_id="state.data.package_id"
                />
            <div class="button-list button-vertical-list
full">
                <v-row align="center">
                    <v-col class="text-center" cols="12">
                        <btn-back />
                    </v-col>
                </v-row>
            </div>
        </div>


        <div v-if="state_is('list_pbox')">
            <!--<pbox-select
                :record="state.data.picking"
                :select_records="state.data.boxes"
                :select_options="select_box_manual_sel
ect_opts()"
                :key="make_state_component_key(['pbo
x-select'])"
                />-->


            <manual-select-print
                :records="state.data.boxes"
                :key="make_state_component_key(['ma
nual-select'])"
                :options="select_box_manual_select_opt
s()"
                v-on:print_manual="on_print_manual"
                v-on:print="on_print"
                :print_model="'stock.picking.pbox'"
                />


            <div class="button-list button-vertical-list
full">
                <!--<v-col class="text-center" cols="12">
                    <btn-action
                        :disabled="_.isEmpty(selected_li
nes())"
                        @click="state.printLabelBox">
                        <v-icon>mdi-printer</v-icon>
                    {{$t('btn.mo_print.title')}}
                    </btn-action>
                </v-col>
                <v-col class="text-center" cols="12">
                    <btn-action
                        :disabled="_.isEmpty(selected_li
nes())"
                        @click="state.printLabelBoxMan
ual">
                        <v-icon>mdi-printer-off</v-icon>
                     {{$t('btn.mo_print_manual.title')}}
                    </btn-action>
                </v-col>-->
                <v-row align="center">
                    <v-col class="text-center" cols="12">
                        <btn-back />
                    </v-col>
                </v-row>
            </div>
        </div>

        <div v-if="state_is('pbox_print')">
            <detail-product_box
                :record="state.data.box"
            />
```

```
            <div class="button-list button-vertical-list
full">
                <v-row align="center">
                    <v-col class="text-center" cols="12">
                        <btn-action
@click="printLabelBox">
                        <v-icon>mdi-printer</v-icon>
                        {{ $t("btn.mo_print.title") }}
                        </btn-action>
                    </v-col>
                    <v-col class="text-center" cols="12">
                        <btn-action
@click="printLabelBoxManual">
                        <v-icon>mdi-printer-off</v-icon>
                        {{ $t("btn.mo_print_manual.title") }}
                        </btn-action>
                    </v-col>
                     <v-col class="text-center" cols="12">
                        <btn-back />
                    </v-col>
                </v-row>
            </div>
        </div>

        <div v-if="state_is('create_package')">

            <manual-select-print
                :records="state.data.package_types"
                :key="make_state_component_key(['ma
nual-select'])"
                :options="manual_selection_package_ty
pes_options()"
                v-on:print_manual="on_print_manual"
                v-on:print="on_print"
                :print_model="'stock.quant.package'"
                />

            <div class="button-list button-vertical-list
full">
                <v-row align="center">
                    <!--<v-col class="text-center"
cols="12">
                        <btn-action
                            @click="printLabelBox"
                        >
                        <v-icon>mdi-printer</v-icon>
                         {{ $t('btn.mo_print.title') }}
                        </btn-action>
                    </v-col>
                    <v-col class="text-center" cols="12">
                        <btn-action
@click="printLabelBoxManual">
                        <v-icon>mdi-printer-off</v-icon>
                        {{ $t('btn.mo_print_manual.title') }}
                        </btn-action>
                    </v-col> -->
                     <v-col class="text-center" cols="12">
                        <btn-back />
                    </v-col>
                </v-row>
            </div>
        </div>

        <div
            v-if="state_is('scan_products') ||
state_is('scan_products_step_2')"
        >
            <v-card v-if="packageId ||
state.data.package_id" class="income-package">
                <v-card-title>
                    <div class="main-info">
                        <div class="package">
                            <span
class="label">{{$t("screen.scan_products.label_pack
age")}}:</span>
                            <span>{{
state.data.package_id.name }}</span>
                        </div>
                    </div>
                    <v-dialog v-model="dialog" tile
class="actions text-center">
                        <template v-slot:activator="{ on }">
                            <div class="button-list button-
vertical-list income-package-action">
                                <v-row class="actions bottom-
actions">
                                    <v-col class="text-center"
cols="12">
                                        <btn-action v-on="on">{{
$t('btn.mo_action.title') }}</btn-action>
                                    </v-col>
                                </v-row>
                            </div>
                        </template>
                        <v-card>
                            <div class="button-list button-
vertical-list full">
                                <v-row align="center">
                                    <v-col class="text-center"
cols="12">
                                        {{
state.data.package_id.name }}
                                    </v-col>
                                </v-row>
                                <input-number-spinner
                                    :class="'number-spinner print'"
                                    :mode="'s1'"
                                    :init_value="1"
                                />
                                <v-row align="center">
                                    <v-col class="text-center"
cols="12">
                                        <btn-action
@click="on_print(state.data.package_id,
'stock.quant.package', 1, true)">
                                            <v-icon>mdi-printer</v-icon>
                                            {{ $t('btn.mo_print.title') }}
                                        </btn-action>
                                    </v-col>
                                </v-row>
                                <v-row align="center">
                                    <v-col class="text-center"
cols="12">
                                        <btn-action
@click="on_print_manual(state.data.package_id,
'stock.quant.package', 1, true)">
                                            <v-icon>mdi-printer-off</v-
icon>
                                            {{
$t('btn.mo_print_manual.title') }}
```

```
                        </btn-action>
                      </v-col>
                    </v-row>
                    <v-row align="center">
                      <v-col class="text-center"
cols="12">

                        <v-dialog v-
model="dialog_confirm_close_package" tile
class="actions text-center">
                          <template v-slot:activator="{
on }">
                            <div class="button-list
button-vertical-list income-package-action">
                              <v-row class="actions
bottom-actions">
                                <v-col class="text-center"
cols="12">
                                  <btn-action v-on="on"
action="warn">
                                    <v-icon>mdi-check-
bold</v-icon>
                                    {{
$t('btn.mo_close_package.title') }}
                                  </btn-action>
                                </v-col>
                              </v-row>
                            </div>
                          </template>
                          <!--<user-confirmation-mo
                            v-
bind:confirm_action="'close_package'"
                            v-on:user-
confirmation="on_user_confirm"
                            v-
bind:question="$t('screen.scan_products_step_2.con
firm_close_pack_question')">
                          </user-confirmation-mo>-->
                          <mo-confirm-btn
                            v-
on:action="state.on_action"
                            :confirm_action="'close_pa
ckage'"
                            :question="$t('screen.scan
_products_step_2.confirm_close_pack_question')"
                          />
                        </v-dialog>


                      </v-col>
                    </v-row>
                    <v-row align="center">
                      <v-col class="text-center"
cols="12">
                        <v-btn x-large @click="dialog =
false">{{ $t('btn.mo_back.title') }}</v-btn>
                      </v-col>
                    </v-row>
                  </div>
                </v-card>
              </v-dialog>
            </v-card-title>
          </v-card>


          <scan-product-check-actions
            @list_box="state.on_action('show_printe
d_boxes', false, false, state.data.picking_id,
state.data.batch_id)"
            @create_package="state.on_action('cre
ate_package_start')"
            @user_confirm="on_user_confirm"
            :picking_id="state.data.picking_id"
            :batch_id="state.data.batch_id"
            :inline="'true'"
          >
            <template v-slot:add_button_1>
              <v-dialog v-
model="dialog_confirm_finish_picking" tile
class="actions text-center">
                <template v-slot:activator="{ on }">
                  <btn-action v-on="on"
action="complete" class="income-add-buttons-btn">
                    {{ $t("btn.mo_confirm.title") }}
                  </btn-action>
                </template>
                <!--<user-confirmation-mo
                  v-
bind:confirm_action="'finish_reception'"
                  v-on:user-
confirmation="on_user_confirm"
                  v-
bind:question="$t('screen.scan_products_step_2.con
firm_finish_reception_question')">
                </user-confirmation-mo>-->
                <mo-confirm-btn
                  v-on:action="state.on_action"
                  :confirm_action="'finish_receptio
n'"
                  :question="$t('screen.scan_produ
cts_step_2.confirm_finish_reception_question')"
                />
              </v-dialog>


              <!--<btn-action
@click='stop_reception(true)' action="complete"
class="income-add-buttons-btn">
                {{ $t("btn.mo_confirm.title") }}
              </btn-action>-->
            </template>
            <template v-slot:add_button_2>
              <btn-action
@click='stop_reception()' :action="'back'"
class="income-add-buttons-btn">
                <v-icon>mdi-keyboard-
backspace</v-icon>{{ $t("btn.back.title") }}
              </btn-action>
            </template>
          </scan-product-check-actions>


          <searchbar
            v-on:found="on_scan"
            :input_placeholder="search_input_place
holder"
          />
          <v-data-table
            :search="search"
            @click:row="state.row_click"
            :headers="select_line_table_headers()"
            :items="select_line_table_items()"
            :key="make_state_component_key(['dat
a-table'])"
```

```
            :item-class="itemRowBackground"
            multi-sort
            :footer-props="{
                'items-per-page-text':
$t('screen.scan_products.table_per_page'),
                <!--'items-per-page-options': [15, 30,
50, 100]-->
            }"
            class="elevation-1">

            <template v-slot:top>
              <v-text-field
               v-model="search"
               append-icon="mdi-magnify"
               :label="$t('screen.scan_products.table
_search')"
               single-line
               hide-details
               :class="'search-table-ksk-income'"
              ></v-text-field>
            </template>

            <template v-slot:item.actions="{ item }">
              <ksk-income-line-actions
                  :key="make_state_component_key
(['product-line', item['_origin'].id])"
                  v-on:action="state.on_action"
                  :line="item"
                  :batch_id="state.data.batch_id"
                  :picking_id="state.data.picking_id"
                  :moves="item['_origin'].moves"
                  :package_id="packageId"
              />
            </template>
          </v-data-table>

          <!--
          <scan-product-check-actions
             @list_box="state.on_action('show_printe
d_boxes', false, false, state.data.picking_id,
state.data.batch_id)"
             @create_package="state.on_action('cre
ate_package_start')"
             @user_confirm="on_user_confirm"
             :picking_id="state.data.picking_id"
             :batch_id="state.data.batch_id"
          />

          <div class="button-list button-vertical-list
full">
             <v-row align="center">
                <v-col class="text-center" cols="12">
                   <btn-action
@click='stop_reception()' :action="'back'">
                      <v-icon>mdi-keyboard-
backspace</v-icon>{{ $t("btn.back.title") }}
                   </btn-action>
                </v-col>
             </v-row>
          </div> -->
       </div>
    </Screen>
  `,
methods: {
    stop_reception: function(finish=false) {
        this.wait_call(
          this.odoo.call("stop_reception", {
              partner_id: this.partnerId,
              picking_id: this.state.data.picking_id,
              batch_id: this.state.data.batch_id,
              package_id: this.packageId,
              finish: finish
          })
        );
    },
    screen_title: function () {
        if (this.state.data.name){
            let title = this.state.data.name;
            return title;
        } else if (this.state.data.product_template){
            let title =
this.state.data.product_template.name;
            return title;
        } else if (this.state.data.box){
            let title = this.state.data.doc_name + ' > ' +
this.state.data.box.barcode;
            return title;
        } else if (this.state.key == 'create_package'){
            let title = this.state.data.doc_name + ' > ' +
this.$t('screen.create_package.title')
            return title;
        } else if (this.state.key == 'list_pbox'){
            let title = this.state.data.doc_name + ' > ' +
this.$t('screen.list_pbox.title')
            return title;
        } else if (this.state.key == 'pbox_print'){
            let title = this.$t("screen.screen_title.title")
            return title;
        } else {
            let title = this.menu_item().name;
            return title;
        }
    },
    screen_klass: function () {
        return (
            this.$super(ScenarioBaseMixin).screen_kla
ss() +
            " device-mode-" +
            this.device_mode
        );
    },
    select_line_table_headers: function () {
        // Convert to v-data-table keys
        const headers =
_.map(this.product_line_list_fields(true), function
(field) {
            return {
              text: field.label,
              value: field.path,
              // Sorting delegated to button
//              sortable: false,
            };
        });
        headers.push({
            sortable: false,
            text: "",
            value: 'actions'
        })
        return headers;
    },
<?xml version="1.0" encoding="utf-8"?>
```

```
<odoo>
  <data>
      <record model="ir.ui.view"
id="mo_wms_view_stock_quant_tree_editable">
        <field
name="name">stock.quant.tree.editable</field>
        <field name="model">stock.quant</field>
        <field name="inherit_id"
ref="stock.view_stock_quant_tree_editable"/>
        <field name="arch" type="xml">
           <field name="lot_id" position="after">
              <field name="exp_date"
optional="hide"/>
           </field>
           <field name='company_id' position="after">
              <field name="pick_location"
optional="hide"/>
              <field name="storage_location"
optional="hide"/>
           </field>
           <field name='available_quantity'
position="after">
              <field name="reserved_quantity"
optional="hide"/>
              <field name="quant_volume"
optional="hide"/>
              <field name="location_id_vol"
optional="hide"/>
              <field name="recycle_percent"
optional="hide"/>
           </field>
           <xpath expr="//field[@name='value']"
position="attributes">
              <attribute
name="optional">show</attribute>
           </xpath>
           <xpath
expr="//field[@name='product_uom_id']"
position="attributes">
              <attribute
name="optional">show</attribute>
           </xpath>
        </field>
     </record>
  </data>
</odoo>
```