

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

**Пояснювальна записка**

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

(назва теми)

ПІДТРИМКИ ПЛАНУВАННЯ ДІЯЛЬНОСТІ МЕРЕЖІ

НЕПРОДУКТОВИХ МАГАЗИНІВ. RESEARCH AND SOFTWARE

IMPLEMENTATION OF METHODS FOR PLANNING THE

ACTIVITY OF A NON-GROCERY STORE CHAIN

Виконав(ла): студент(ка) 2 курсу, групи КНТ-142м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

МОРОЗ В.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет \_\_\_\_\_ ФКНТ \_\_\_\_\_  
Кафедра \_\_\_\_\_ програмних засобів \_\_\_\_\_  
Ступінь вищої освіти \_\_\_\_\_ магістр \_\_\_\_\_  
Спеціальність \_\_\_\_\_ 121 Інженерія програмного забезпечення \_\_\_\_\_  
(код і найменування)  
Освітня програма (спеціалізація) \_\_\_\_\_ Інженерія програмного забезпечення \_\_\_\_\_  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
Сергій СУББОТІН  
« \_\_\_\_\_ » \_\_\_\_\_ 2023 року

**З А В Д А Н Н Я**  
**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

МОРОЗА Віталія Вікторовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів підтримки планування діяльності мережі непродуктових магазинів. Research and Software Implementation of Methods for Planning the Activity of a Non-Grocery Store Chain

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович \_\_\_\_\_,  
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 14 » листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року \_\_\_\_\_  
3. Вихідні дані до проєкту (роботи) \_\_\_\_\_ рекомендована література, технічне завдання \_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів, плакатів)  
Слайди презентації \_\_\_\_\_

## 6. Консультанти розділів проєкту (роботи)

| Розділ              | ПРІЗВИЩЕ, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------------|-------------------------------------------|----------------|---------------------------|
|                     |                                           | завдання видав | прийняв виконане завдання |
| 1-5 Основна частина | ЛЬОВКІН В.М., доцент                      |                |                           |
| Нормоконтроль       | КАМІНСЬКА Ж.К., асистент                  |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |

7. Дата видачі завдання « 04 » вересня 2023 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи)                                             | Срок виконання етапів проєкту ( роботи ) | Примітка     |
|-------|--------------------------------------------------------------------------------------|------------------------------------------|--------------|
| 1     | Постановка завдання роботи                                                           | 1 тиждень                                | Завдання, ТЗ |
| 2     | Аналіз предметної області                                                            | 2 тиждень                                | Розділ 1     |
| 3     | Дослідження проблеми                                                                 | 3-5 тижні                                | Розділ 2     |
| 4     | Розробка архітектури програми                                                        | 6 тиждень                                | Розділ 3     |
| 5     | Розробка програми                                                                    | 7-8 тижні                                | Розділ 4     |
| 6     | Тестування та експериментальне дослідження програми                                  | 9-10 тижні                               | Розділ 5     |
| 7     | Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування | 11 тиждень                               | Додатки      |
| 8     | Захист роботи                                                                        | 12 тиждень                               |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |
|       |                                                                                      |                                          |              |

Студент(ка)

  
( підпис )

Віталій МОРОЗ

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

  
( підпис )

Валерій ЛЬОВКІН

(Ім'я ПРІЗВИЩЕ)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:  
116 с., 29 рис., 6 табл., 3 дод., 14 джерел.

НЕПРОДУКТОВИЙ ТОВАР, МЕРЕЖА МАГАЗИНІВ, СКЛАД,  
ДЕРЕВА РІШЕНЬ, ПРОБЛЕМА ПРОГНОЗУВАННЯ, ПЛАНУВАННЯ  
ДІЯЛЬНОСТІ, RANDOM FOREST.

Об'єкт роботи – процеси підтримки планування і підтримки самої діяльності мережі непродуктових магазинів. Предмет роботи – методи планування діяльності мережі непродуктових магазинів. Мета роботи – розробити програмне забезпечення підтримки діяльності мережі непродуктових магазинів для забезпечення планування діяльності на майбутні періоди.

Матеріали, методи та технічні засоби: метод Random Forest, мова python, фреймворк Django, система MySQL.

Результати. Створено програму, яка надає основні можливості, необхідні для операторів і менеджерів складів і магазинів мережі непродуктових магазинів, клієнтів цієї мережі, забезпечуючи засоби моніторингу, аналізу діяльності мережі, прогнозування часу продажу партії товару, що є одним з базових показників для планування діяльності.

Висновки. Наукова новизна роботи полягає в методі прогнозування часу продажу партії непродуктового товару зі складу, який виконує прогнозування кількості днів, протягом яких буде розпродано весь товар з партії на складі, використовуючи ансамблі дерев рішень, створені за методом Random Forest, і визначаючи обчислення ряду вхідних ознак для створення моделі прогнозування.

Галузь використання. Мережі магазинів або окремі магазини, які продають непродуктові товари.

## ABSTRACT

Explanatory note to the diploma qualifying work of the master: 116 pages, 29 figures, 6 tables, 3 appendixes, 14 sources.

NON-GROCERY, STORE CHAIN, WAREHOUSE, DECISION TREES, FORECASTING PROBLEM, ACTIVITY PLANNING, RANDOM FOREST.

The object of the work is the processes of supporting the planning and supporting the activities of the chain of non-grocery stores. The subject of the work is methods of planning the activity of a chain of non-grocery stores. The purpose of the work is to develop software to support the activities of a chain of non-grocery stores to ensure activity planning for future periods.

Materials, methods and technical means: Random Forest method, python language, Django framework, MySQL system.

The results. A program has been created that provides the basic capabilities necessary for operators and managers of warehouses and stores of a chain of non-grocery stores, customers of this chain, providing means of monitoring, analysis of chain activity, forecasting the time of sale of a batch of goods, which is one of the basic indicators for activity planning.

Conclusions. The scientific novelty of the work consists in the method of forecasting the time of sale of a batch of non-grocery goods from a warehouse, which performs a forecast of the number of days during which all goods from a batch in a warehouse will be sold out, using ensembles of decision trees created by the Random Forest method, and determining the calculation of a number of input features for creating a forecasting model.

Field of use. Chain stores or individual stores that sell non-grocery goods.

## ЗМІСТ

|                                                                                                                                      |    |
|--------------------------------------------------------------------------------------------------------------------------------------|----|
|                                                                                                                                      | С. |
| Перелік скорочень та умовних познач.....                                                                                             | 8  |
| Вступ.....                                                                                                                           | 9  |
| 1 Аналіз проблеми та постановка завдань дослідження .....                                                                            | 10 |
| 1.1 Огляд програмного забезпечення підтримки роботи мережі<br>непродуктових магазинів .....                                          | 10 |
| 1.2 Порівняння аналогів .....                                                                                                        | 12 |
| 1.3 Висновки за розділом 1 .....                                                                                                     | 13 |
| 2 Матеріали і методи.....                                                                                                            | 15 |
| 2.1 Задача прогнозування часу продажу партії непродуктового товару зі<br>складу і використання дерев рішень для її розв’язання ..... | 15 |
| 2.2 Метод прогнозування часу продажу партії непродуктового товару зі<br>складу .....                                                 | 17 |
| 2.3 Висновки за розділом 2 .....                                                                                                     | 20 |
| 3 Проектування програмного забезпечення .....                                                                                        | 21 |
| 3.1 Вибір рішень для веброзробки.....                                                                                                | 21 |
| 3.2 Моделювання програмного забезпечення на основі функціональних<br>вимог .....                                                     | 22 |
| 3.3 Проектування структури бази даних.....                                                                                           | 30 |
| 3.4 Висновки за розділом 3 .....                                                                                                     | 36 |
| 4 Основні рішення щодо реалізації програмного забезпечення .....                                                                     | 38 |
| 4.1 Порядок роботи програмного забезпечення .....                                                                                    | 38 |
| 4.2 Рішення з реалізації вебзастосунку .....                                                                                         | 40 |
| 4.3 Висновки за розділом 4 .....                                                                                                     | 62 |
| 5 Експлуатація, тестування та експериментальне дослідження програми .....                                                            | 63 |
| 5.1 Опис програми.....                                                                                                               | 63 |
| 5.2 Експлуатація програми.....                                                                                                       | 63 |
| 5.3 Експериментальне дослідження .....                                                                                               | 69 |
| 5.4 Висновки за розділом 5 .....                                                                                                     | 72 |

|                                    |     |
|------------------------------------|-----|
| Висновки .....                     | 73  |
| Перелік джерел посилання .....     | 74  |
| Додаток А Технічне завдання .....  | 76  |
| Додаток Б Текст програми .....     | 83  |
| Додаток В Слайди презентації ..... | 108 |

## **ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК**

CART – Classification and Regression Tree;

QR – quick response;

БД – база даних.

## ВСТУП

Мережа непродуктових магазинів є достатньо складним механізмом, у роботі якого задіяно багато користувачів, що мають загалом велику кількість ролей. Робота зі всіма екземплярами товарів вимагає контролю на великій кількості етапів, але і аналіз даних про продажі загалом також потребує великої кількості інструментів. Тільки в такому випадку можна створити програмне забезпечення, яке дозволить цілісно підтримувати роботу мережі непродуктових магазинів.

Створення програми підтримки діяльності мережі непродуктових магазинів передбачає реалізацію ряду сценаріїв, які мають забезпечити роботі окремих складів, окремих магазинів, а також і менеджменту мережі магазинів через програму.

Впровадження інструментів прогнозування до такого програмного забезпечення дозволить підвищити ймовірність її застосування на практиці. Тому в програмі потрібно реалізувати обов'язково прогнозування часу продажу партії товару.

Виконуючи мету роботи, потрібно розробити програмне забезпечення підтримки діяльності мережі непродуктових магазинів для забезпечення планування діяльності на майбутні періоди.

## **1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ**

### **1.1 Огляд програмного забезпечення підтримки роботи мережі непродуктових магазинів**

Система планування ресурсів підприємства NetSuite [1] не направлена виключно на мережі магазинів непродуктових магазинів, але надає широкі можливості, для чого надається цілий ряд модулів. В залежності від потреб обирається той чи інший набір модулів. Можливості системи на основі модулів включають зокрема наступні:

- управління фінансами, включаючи виконання бухгалтерських завдань, створення щомісячних фінансових звітів для регулятивної та законодавчої звітності;
- глобальне управління бізнесом: надається можливість керувати підрозділами транснаціональних компаній, підтримуючи різноманітні валюти, мови, податкове законодавство, вимоги до звітності;
- планування та бюджетування для відділів або компаній загалом, включаючи статистичні моделі, за допомогою яких можна прогнозувати доходи, витрати тощо;
- підтримка платежів, що дозволяє керувати виставленням рахунків і оплатою, включаючи відповідні транзакції, підписку, інші моделі виставлення рахунків;
- управління запасами забезпечує перегляд запасів у всіх каналах і місцях, окремий модуль при цьому використовує дані про продажі для визначення повторних замовлень та сповіщення менеджерів із закупівель, коли потрібно поповнити записи і визначити відповідне замовлення;
- управління замовленнями з управлінням усім життєвим циклом замовлення;
- організація закупівель дозволяє вести список схвалених постачальників, автоматизувати замовлення, відстежувати ефективність і витрати;

- керування складом включає функції приймання, комплектування замовлень, сканування штрих-кодів і повернення;
- управління і моніторинг виробництва для компаній, які виготовляють товари;
- планування попиту виконується на основі історичних даних, використовуючи методи ковзної середньої, лінійної регресії або сезонної середньої;
- управління взаємовідносинами з клієнтами;
- розрахунок заробітної плати, автоматизуючи розрахунки прибутків, відрахувань, податків, бонусів і оплачуваних відпусток;
- керування людськими ресурсами з відстеженням змін в просуванні;
- аналітика [2].

Рішення NetSuite для керування відносинами з клієнтами охоплює керування контактами і потенційними клієнтами, замовленнями на продаж, розбиваючи модуль на взаємовідносини з клієнтами, транзакції з продажу (продажі за готівку та замовлення на продаж), звіти про продажі [3].

У Prime Penguin [4] таблиці прогнозування представляють дані для кожного варіанту продукту стосовно:

- поточний запас відповідає загальному рівню запасів на всіх складах;
- поточні продажі містять продажі за останні тижні в абсолютному вираженні;
- середній обсяг продажів містить значення ковзного, зваженого середнього, яке використовується для визначення майбутніх продажів;
- тенденція продажів містить зважену тенденцію, яка використовується для визначення майбутніх продажів;
- кількість тижнів в наявності відповідає прогнозованій кількості тижнів, протягом якої очікується товар у наявності [5].

Можливості Prime Penguin включають:

- контроль запасів: можна встановити точки повторного замовлення та визначити оптимальну кількість замовлень, щоб забезпечити задоволення попиту;
- відстеження запасів: ця функція дозволяє моніторити фізичний рух запасів, коли вони надходять і залишають склад або сховище;
- прогнозування: ця функція дозволяє оцінити майбутній попит на продукти, використовуючи дані про продажі, тенденції та іншу необхідну інформацію;
- переміщення запасів: надається можливість керувати процесом закупівлі необхідних запасів у постачальників, створювати замовлення на купівлю, відстежувати доставку та контролювати процес затвердження нових замовлень;
- звітність: звіти створюються про рівні запасів, показники обороту та інші показники, які дають змогу приймати керовані даними рішення щодо управління запасами [6].

## **1.2 Порівняння аналогів**

При порівнянні програми, що створюється, з аналогами використано NetSuite і Prime Penguin (табл. 1.1). Головною відмінністю програми, яка створюється, є її безкоштовність. Аналоги є достатньо вимогливими щодо фінансових ресурсів, зокрема NetSuite потребує і щорічної сплати, і сплати за кожного користувача, при цьому не має можливості попереднього тестування, тому якщо програмне забезпечення користувачам не підійде, то повернути витрачені кошти неможливо. Окрім того програма використовує моделі прогнозування, які фактично тільки продовжують тенденції з накопичених даних. До того ж налаштування NetSuite є достатньо складним процесом, який потребує навчання, що є окремою послугою, що надається компанією.

Таблиця 1.1 – Порівняння програмного забезпечення, яке може використовуватися для підтримки роботи мережі непродуктових магазинів

| Фактор порівняння      | NetSuite                                                                                         | Prime Penguin                                           | Програма в розробці                                                    |
|------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------|------------------------------------------------------------------------|
| Основні функції        | Керування взаємодією з клієнтами, замовленнями, складом                                          | Керування взаємодією з клієнтами, замовленнями, складом | Керування взаємодією з клієнтами, замовленнями, складом                |
| Засоби аналітики       | Так                                                                                              | Так                                                     | Так                                                                    |
| Налаштування роботи    | Потребує окремого встановлення і налаштування для різних ролей, що потребує додаткового навчання | Менш складний процес ніж у NetSuite                     | Потребує встановлення на сервері, додаткового налаштування не потребує |
| Тип доступу            | Платний                                                                                          | Платний                                                 | Безкоштовний                                                           |
| Показник прогнозування | Попит                                                                                            | Попит, час продажу максимум на 10 тижнів                | Час продажу партії за складом                                          |
| Моделі прогнозування   | Рухоме середнє, лінійна регресія, сезонне середнє                                                | Рухоме, зважене середнє                                 | Random Forest                                                          |

### 1.3 Висновки за розділом 1

Розділ 1 містить огляд програмного забезпечення, яке може використовуватися для підтримки роботи мережі непродуктових магазинів, включаючи NetSuite і Prime Penguin.

До визначених завдань кваліфікаційної дипломної роботи належать:

- проаналізувати програмне забезпечення підтримки роботи мережі непродуктових магазинів;
- розробити метод прогнозування часу продажу партії непродуктового товару зі складу і виконати експериментальне дослідження;
- виконати проєктування програмного забезпечення підтримки роботи мережі непродуктових магазинів;
- реалізувати програмне забезпечення підтримки роботи мережі непродуктових магазинів.

## 2 МАТЕРІАЛИ І МЕТОДИ

### 2.1 Задача прогнозування часу продажу партії непродуктового товару зі складу і використання дерев рішень для її розв'язання

У дипломній кваліфікаційній роботі необхідно було розглянути і дослідити методи підтримки планування діяльності мережі непродуктових магазинів. Планування діяльності мережі непродуктових магазинів є достатньо складним процесом, який вимагає отримання інформації про ряд показників. До таких показників входять:

- кількість наявних ресурсів (зокрема доступних фінансових ресурсів, тобто яку кількість товару можна закупити, яку кількість приміщень можна орендувати або створити для складів мережі);
- час збуту наявного товару або такого, який буде закуплений протягом певного періоду;
- прибуток, отриманий від діяльності за певний період;
- витрати, які було понесено на організацію діяльності за певний період тощо.

З виділених показників, якщо їх аналізувати, можна визначити, що вони мають взаємний вплив. Так час збуту товарів залежить зокрема і від кількості закуплених товарів, а відповідно і кількості наявних ресурсів, а також в свою чергу впливає на те, які витрати та який прибуток понесе мережа за деякий період. Але фактично це дозволяє визначити, що час, за який буде розпродано закуплену партію непродуктового товару, є критичним для визначення значень інших показників. Чим раніше буде розпроданий товар, тим швидше можна буде закупити наступний або принаймні спробувати це зробити (адже не завжди будь-який товар доступний для замовлення в потрібний момент), а відповідно збільшити прибуток у підсумку. Враховуючи це, саме прогнозування часу продажу (збуту) партії непродуктового товару зі складу стало цільовим показником у цій роботі.

Задача прогнозування часу продажу партії непродуктового товару зі складу полягає у тому, щоб встановити кількість днів, протягом якої буде продано партію непродуктового товару, яка характеризується зокрема певною кількістю одиниць товару та значеннями інших показників, якими вона може характеризуватися, повністю. Тобто фактично це різниця між датою, коли цю партію товару було отримано на складі, та датою, коли було продано останній екземпляр товару з партії, виражена саме у кількості днів, які минули між цими двома датами. Те, що товари самі непродуктові, означає, що в процесі продажу мережа магазинів не обмежена термінами придатності. Тобто щонайменше в меншій мірі така мережа обмежена терміновістю споживання, це не впливає на рішення, хоча терміни можуть бути обмежені також і виходом нових моделей, появою нових технологій, але загалом це менший вплив, адже терміни тоді менші ніж при продажі продуктових товарів. При продажі продуктів строки можуть бути дуже короткими (дні, тижні, місяці), що точно не характерно для непродуктових товарів. Тож розв'язання задачі передбачає визначення методу, який після виконання має визначити модель прогнозування, яка як раз за подачі вхідних даних має надати на виході кількості днів продажу партії непродуктового товару.

Для розв'язання такої задачі як задачі прогнозування можуть використовуватися дерева рішень. Це алгоритм машинного навчання з учителем, тобто який є контрольованим (для навчання використовуються дані, за якими відомі значення вихідної ознаки). Використовується як для класифікації, так і для розв'язання задач регресії, якою за своєю суттю і є задача прогнозування часу продажу партії непродуктового товару зі складу.

Дерева рішень по суті представляють собою певні блок-схеми (як візуалізації), які є кореневими деревами: в корені розташовується конкретне питання щодо даних, що веде до гілок, які містять потенційні відповіді, однак перед отриманням остаточних відповідей вводяться додаткові запитання, які розташовані у внутрішніх вузлах, внизу знаходяться листові вузли з

конкретними варіантами відповідей, на чому і закінчується структура дерева [7]-[9].

Розв'язувати одночасно завдання і класифікації, і регресії можуть не всі дерева рішень, такий алгоритм названо Classification and Regression Tree (CART) [7]-[9]. Цей алгоритм необхідний для побудови дерева рішень на основі індексу Джині [7].

Random Forest є окремим алгоритмом для навчання з учителем, який працює не з одним деревом, а з ансамблем, тобто набором дерев, при цьому навчання всіх дерев виконується не на всіх наявних даних, а розбиваючи ці дані на частини, навчаючи дерева з ансамбля на різних частинах, а результат узгоджується з результатів усіх дерев [10].

## **2.2 Метод прогнозування часу продажу партії непродуктового товару зі складу**

Метод прогнозування часу продажу партії непродуктового товару зі складу передбачає виконання ряду етапів, частина з яких охоплює підготовку даних, оброблення даних та вже створення моделі в підсумку, яку потім можна використовувати для прогнозування на нових даних. Однак, створити одразу модель неможливо, адже спершу потрібно підготувати дані, оскільки нерідко навіть основні показники, які характеризують саме окрему партію товару, не відслідковуються в мережі (а відповідно і в сховищі даних).

Враховуючи це, на першому етапі необхідно визначити за кожною партією непродуктових товарів значення характеристик цієї партії. Якщо партію не виділено окремо, то спочатку потрібно виділити зі сховища даних всі записи, які стосуються екземплярів непродуктових товарів (одно й того самого товару), які були поставлені на склад у одну й ту саму дату або мають один і той же номер поставки. У результаті виконання цих дій мають бути сформовані екземпляри даних, кожен з яких характеризує унікальну трійку, в яку входить номер товару, дата (або номер партії) постачання та номер

складу, куди постачання було виконано. Ці трійки не можуть повторюватися, тобто кожен екземпляр повинен характеризуватися різною трійкою. Окрім виділення трійки для кожного екземпляра має бути встановлено:

- кількість одиниць (екземплярів) товару в партії;
- категорію непродуктового товару, до якої належать всі товари в партії;
- середня вартість одиниці товару для споживача;
- середній дисконт, який було надано мережею магазинів, на одиницю товару споживачу;
- дата, коли було отримано цю партію товару на склад;
- кількість днів, які пройшли до моменту продажу останнього екземпляру непродуктового товару з партії від моменту постачання цієї партії на склад (цільовий показник).

Для обчислення середніх показників має бути використано дані з замовлення або окремо наведені дані за кожною одиницею товару, а потім усереднено отримані значення. Якщо значення на одиницю не приведено, то потрібно вартість замовлення розділити на кількість одиниць у замовленні для визначення вартості одиниці, аналогічно обчислити дисконт у замовленні та розділити його на кількість одиниць товару в замовленні.

Через те, що далі ці дані мають оброблені, то дату має бути перетворено в ціле число. Для цього має бути визначено горизонт планування (фактично горизонт використання моделі без оновлення). Це остання дата, у яку має бути отримано партію товару і для якої буде виконано прогнозування. Тоді виконується обчислення на основі формули (2.1):

$$d_i = month_i - month^{min} + 12(year_i - year^{min}), \quad (2.1)$$

де  $d_i$  – перетворене значення для поставки з номером  $i$ ;

$month_i$  – номер місяця, що відповідає даті поставки з номером  $i$ , для якої виконується перетворення;

$month^{min}$  – номер місяця, що відповідає даті, коли найраніше було поставлено одну з партій непродуктового товару серед всіх непродуктових товарів;

$year_i$  – номер року, що відповідає даті поставки з номером  $i$ , для якої виконується перетворення;

$year^{min}$  – номер року, що відповідає даті, коли найраніше було поставлено одну з партій непродуктового товару серед всіх непродуктових товарів.

Зазвичай постачання на склад виконуються не частіше раз на місяць, тому таких перетворень достатньо. У випадку, якщо за один місяць на один і той же самий склад може завозитися декілька партій одного й того ж самого товару при перетворенні потрібно враховувати і номер дня в даті. Тоді при обчисленні потрібно рахувати кількість днів, яка пройшла до поставки з номером  $i$  від дати, коли найраніше було поставлено одну з партій непродуктового товару.

У підсумку має бути сформовано навчальну вибірку. Для проведення дослідження сформовану вибірку було розділено на навчальну та тестову. У навчальній вибірці (та тестовій за наявності) має бути відділено показники, як є вхідними ознаками, та цільовий показник.

Далі має бути виконано нормування значень всіх екземплярів даних. Виконання цих дій має реалізовуватися через визначення максимуму для кожного атрибута екземпляра вибірки, які було описано вище. При цьому як раз для визначення максимуму дати, коли було отримано партію товару на склад, і визначався горизонт планування. Дата, визначена як горизонт планування, використовується в якості максимального значення, для якого за формулою (2.1) має бути розраховано відповідне значення. Тоді мінімальне значення дорівнює 0, а максимальне – значенню, що відповідає горизонту

планування. Нормування виконується на навчальній вибірці, тому наявності тестової такі дані не мають враховуватися в процесі нормування.

Наступний етап передбачає створення і навчання моделі прогнозування часу продажу партії непродуктового товару зі складу. Модель створюється за допомогою ансамблю дерев рішень за методом Random Forest, а для навчання використовується навчальна вибірка, в якості критерія навчання використовується середньоквадратична похибка.

Після цього модель готова для подальшого використання на відведеному горизонті планування. Після того, як такий горизонт завершено, доцільно створити і навчити модель на розширеній вибірці даних заново, хоча ці дії можуть бути виконані раніше. Період оновлення моделі відповідає наявним ресурсам для виконання цих обчислень та періодам планування, тобто коли взагалі в наступний раз буде виконуватися замовлення партій непродуктових товарів. Якщо модель навчати заново надто часто, то нових даних у вибірці не буде отримано. Тобто для того щоб з'явився новий екземпляр у вибірці (або екземпляри), необхідно, щоб партія товару була повністю розпродана, а це вимагає певного періоду часу.

## **2.3 Висновки за розділом 2**

Розділ 2 містить задачу прогнозування часу продажу партії непродуктового товару зі складу, опис дерев рішень, які можуть використовуватися для прогнозування, та метод прогнозування часу продажу партії непродуктового товару зі складу.

Наукова новизна роботи полягає в методі прогнозування часу продажу партії непродуктового товару зі складу, який виконує прогнозування кількості днів, протягом яких буде розпродано весь товар з партії на складі, використовуючи ансамблі дерев рішень, створені за методом Random Forest, і визначаючи обчислення ряду вхідних ознак для створення моделі прогнозування.

### 3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

#### 3.1 Вибір рішень для веброзробки

При виборі рішень для веброзробки для реалізації програми розглянуто вебфреймворк Django [11] і систему керування вмістом Wordpress [12] (табл. 3.1). Обрано Django для реалізації, адже даний проєкт має нестандартні функціональні вимоги та потребує для реалізації засобів машинного навчання.

Таблиця 3.1 – Порівняння рішень для веброзробки

| Фактор порівняння                      | Django                                                                                     | Wordpress                                                                                                   |
|----------------------------------------|--------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Тип рішення                            | Фреймворк веброзробки                                                                      | Система керування вмістом                                                                                   |
| Мова програмування                     | python                                                                                     | PHP                                                                                                         |
| Витрати на розробку                    | Більші, коли рішення типове, однак менші за нетипових вимог, характерних для цього проєкту | Менші, коли рішення типове, однак більші за нетипових вимог, характерних для цього проєкту                  |
| Доступність засобів машинного навчання | Велика кількість готових і потужних бібліотек машинного навчання                           | Стандартні плагіни не реалізують створений метод, обмежені засоби машинного навчання для реалізації плагіна |
| Підтримка масштабованості              | Більш проста, менше обмежень                                                               | Можлива за рахунок використання плагінів, які найчастіше потрібно додатково розробляти для цього            |

### **3.2 Моделювання програмного забезпечення на основі функціональних вимог**

Список функціональних вимог і відповідних сценаріїв, які має забезпечувати програма, включає:

- формування запиту на замовлення непродуктових товарів у виробника полягає у об'єднанні даних про всі товари, які були попередньо замовлені магазинами, але відсутні на складі, одночасно відбувається прогнозування часу продажу такої партії непродуктового товару;
- внесення даних замовлення непродуктових товарів у виробника фактично є результатом виконання попереднього сценарію, коли запропоновану базову кількість менеджер уточнює, вносячи відповідні зміни, для яких відбувається прогнозування часу продажу такої партії непродуктового товару, та зберігає всі ці результати як замовлення;
- відображення замовлень непродуктових товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- деталізація замовлення непродуктових товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- скасування запиту на замовлення непродуктових товарів у виробника;
- формування замовлення непродуктових товарів зі складу в магазин, в тому числі якщо товари відсутні на складі;
- відображення замовлень непродуктових товарів зі складу у відповідності з рівнем доступу та обов'язками відповідного користувача;
- призначення виконання замовлення непродуктових товарів зі складу працівнику;
- деталізація замовлення непродуктових товарів зі складу мережі;
- скасування замовлення непродуктових товарів зі складу мережі;

- реєстрація прибуття непродуктових товарів за запитом на склад від виробника, включаючи реєстрацію конкретних екземплярів товару, визначення розташування цих екземплярів на складі;
- реєстрація відправлення замовлення зі складу до магазину, включаючи реєстрацію конкретних екземплярів товару;
- реєстрація прибуття замовлення зі складу до магазину, включаючи реєстрацію конкретних екземплярів товару;
- реєстрація продажу замовлення клієнту з фіксацією конкретних екземплярів товарів;
- реєстрація повернення товару з замовлення у відведений законодавством для цього строк;
- реєстрація стану гарантійного обслуговування екземпляру товару;
- відображення замовлень у магазині в розрізі станів;
- редагування інформації про непродуктовий товар, що був внесений до каталогу;
- редагування створеного замовлення товарів зі складу мережі у відповідності з рівнем доступу та обов'язками відповідного користувача;
- редагування створеного замовлення товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- визначення складу мережі, непродуктові товари з якого доступні магазину;
- відображення магазинів мережі, що працюють з заданим складом;
- закриття доступу магазину до складу мережі непродуктових магазинів;
- створення нового непродуктового товару, доступного для замовлення;
- закриття непродуктового товару для нових замовлень;
- повернення непродуктового товару як доступного для замовлення;
- відображення замовлень клієнта в розрізі станів, включаючи нові, виконані, скасовані, усі;

- відображення списку непродуктових товарів, які за весь час замовляв клієнт;
- відображення повернень екземплярів непродуктових товарів клієнтом;
- відображення екземплярів непродуктових товарів, що знаходяться у гарантійному обслуговуванні на дану мить;
- пошук непродуктових товарів за визначеними параметрами;
- відображення даних непродуктового товару;
- відображення історії зміни станів екземпляру товару;
- переміщення непродуктового товару на складі мережі;
- замовлення непродуктових товарів;
- скасування невиконаного замовлення непродуктових товарів;
- підтвердження створеного клієнтом замовлення для доставки зі складу;
- скасування створеного клієнтом замовлення для доставки зі складу;
- авторизація користувачів у всіх можливих ролях;
- реєстрація нового менеджера складу мережі непродуктових магазинів;
- реєстрація нового оператора складу мережі непродуктових магазинів;
- реєстрація нового менеджера магазину мережі непродуктових магазинів;
- реєстрація нового оператора магазину мережі непродуктових магазинів;
- реєстрація нового менеджера мережі непродуктових магазинів;
- змінювання місця роботи користувача, в тому числі переведення між складом, магазином, компанією загалом або ролями оператора або менеджера всередині цих відділень;
- відображення списку працівників складу за заданим складом або за складом, де працює користувач;

- відображення списку працівників магазину за заданим магазином або за магазином, де працює користувач;
- відображення списку працівників компанії за можливості такого доступу;
- реєстрація загублення екземпляру непродуктового товару;
- реєстрація повернення екземпляру непродуктового товару на склад;
- пошук працівника в компанії;
- оновлення персональних даних;
- оформлення продажу непродуктових товарів у магазині без замовлення онлайн;
- перегляд включених підкатегорій непродуктових товарів;
- перегляд непродуктових товарів, що продаються у мережі, за підкатегорією;
- моніторинг стану продажу партій за непродуктовим товаром;
- моніторинг повернень товарів за магазином мережі непродуктових магазинів;
- моніторинг продажів непродуктових товарів у розрізі за магазином мережі;
- моніторинг товарів на складі мережі непродуктових магазинів;
- моніторинг товарів у магазині мережі непродуктових магазинів;
- моніторинг непродуктових товарів магазину за категоріями;
- аналіз розподілу наявних у мережі непродуктових товарів за складами;
- аналіз продажу непродуктових товарів за всією мережею за каналами продажу та загалом;
- аналіз роботи магазинів стосовно наданого прибутку;
- аналіз замовлення заданого товару в розрізі магазинів мережі непродуктових магазинів;
- аналіз часу продажу партій непродуктових товарів за заданий період;

- аналіз зміни вартості одиниці непродуктового товару за вказаний період часу;
- аналіз зміни прибутку, отриманого від продажу заданого непродуктового товару;
- прогнозування часу продажу партії непродуктового товару.

Не всі сценарії мають бути доступними всім користувачам, тому вони були розподілені за тим, яку саме роль користувач виконує і відповідно які саме сценарії йому доступні, зокрема для неавторизованого клієнта (рис. 3.1), клієнта (рис. 3.2), оператора (рис. 3.3) і менеджера магазину (рис. 3.4).



Рисунок 3.1 – Діаграма прецедентів неавторизованого клієнта

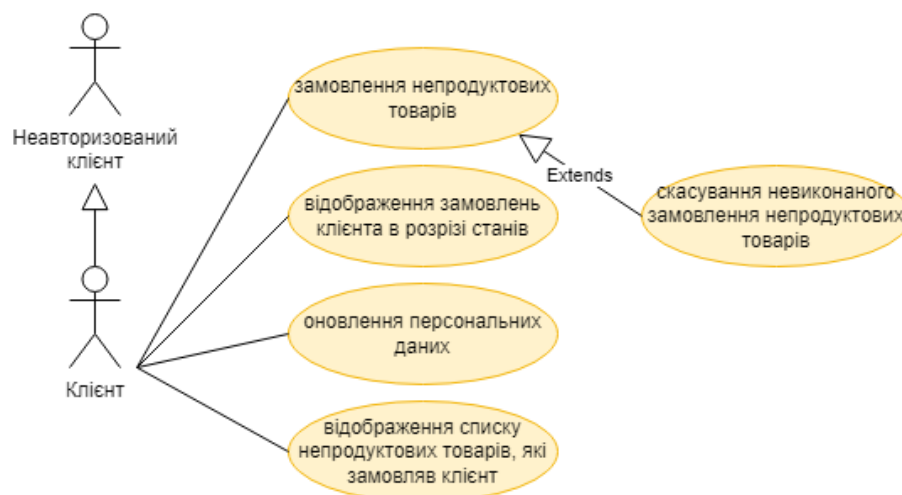


Рисунок 3.2 – Діаграма прецедентів клієнта

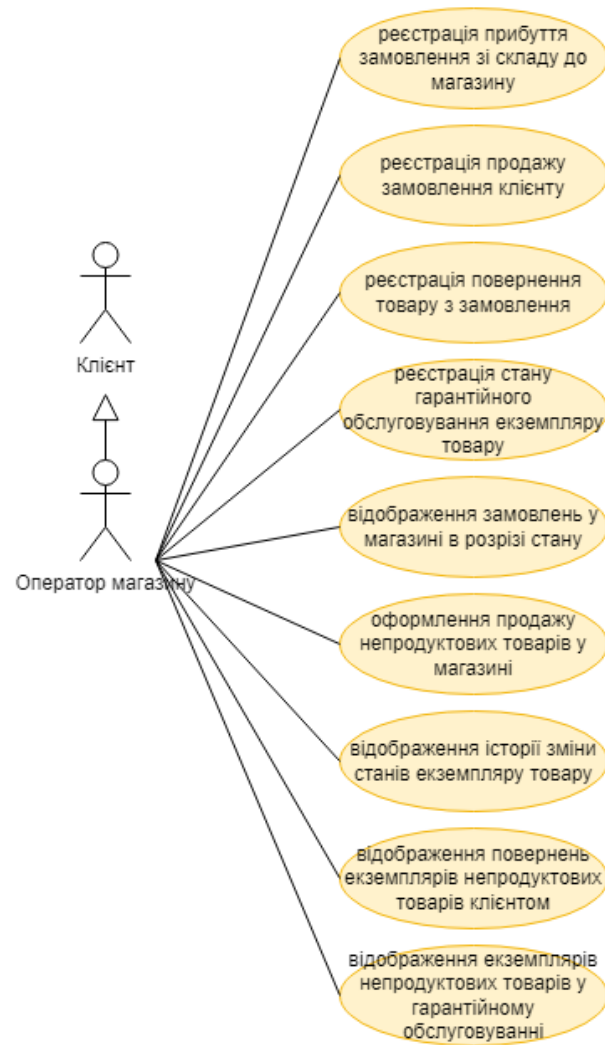


Рисунок 3.3 – Діаграма прецедентів оператора магазину

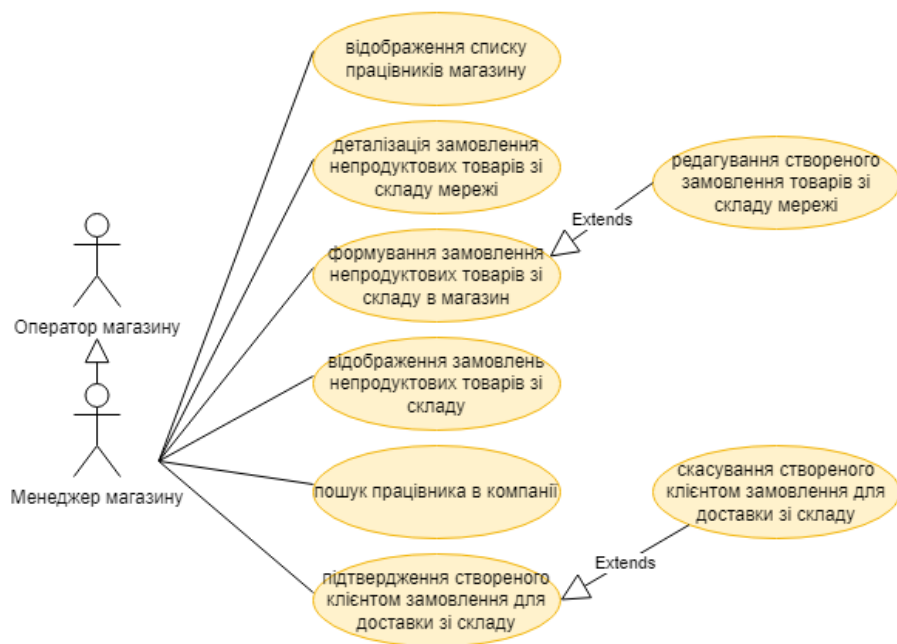


Рисунок 3.4 – Діаграма прецедентів менеджера магазину

Визначені сценарії менеджера мережі (рис. 3.5), оператора (рис. 3.6), менеджера (рис. 3.7) складу.

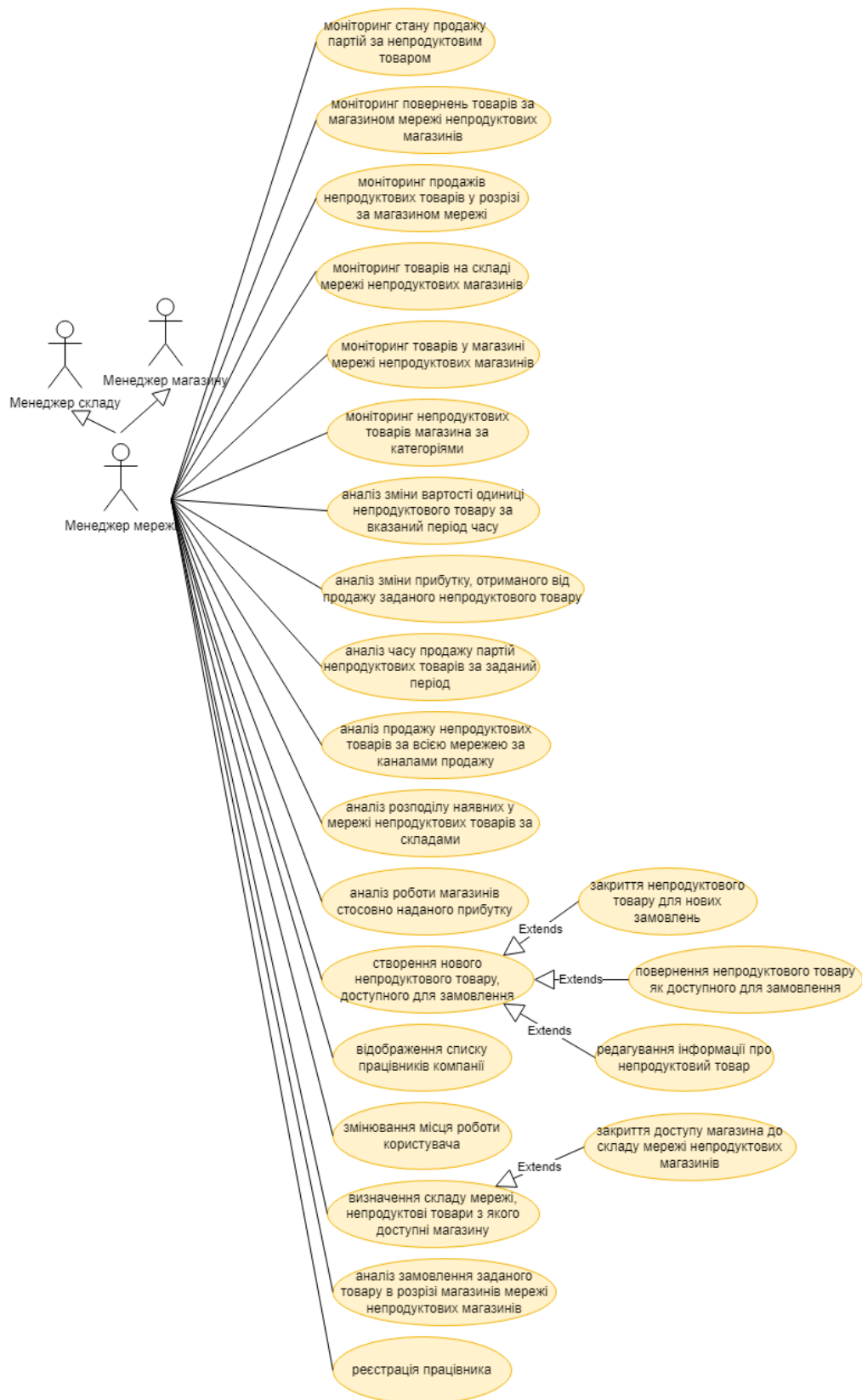


Рисунок 3.5 – Діаграма прецедентів менеджера мережі

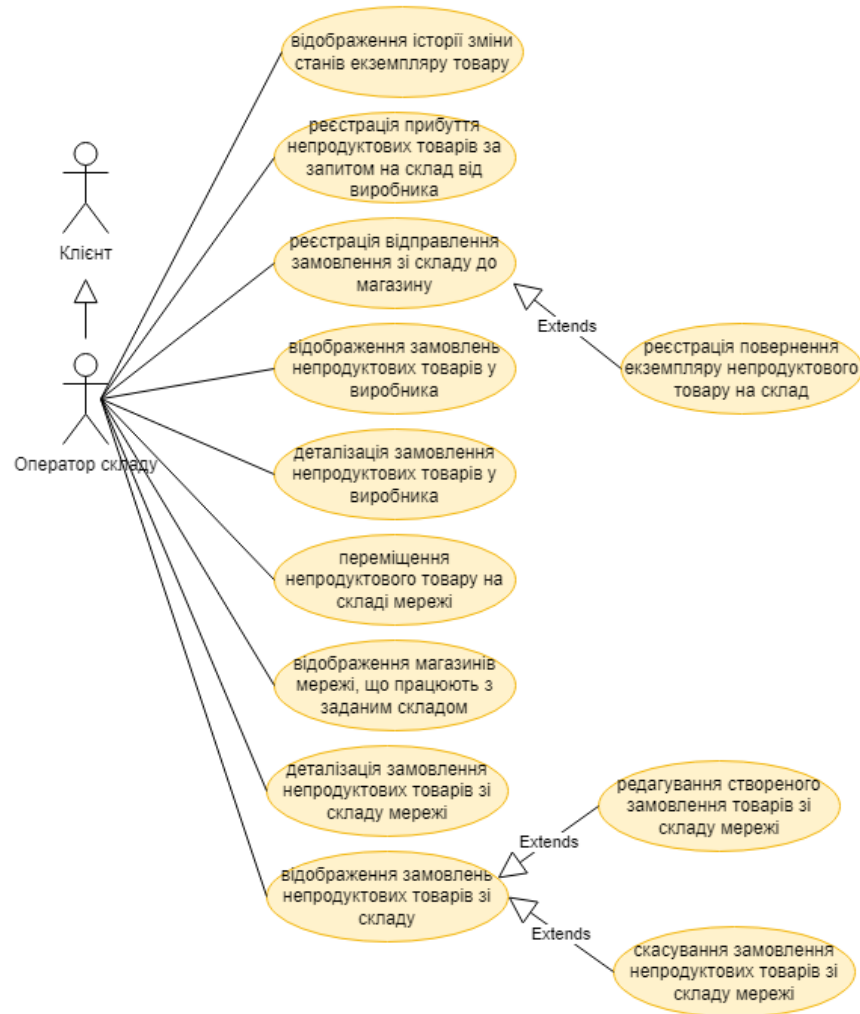


Рисунок 3.6 – Діаграма прецедентів оператора складу

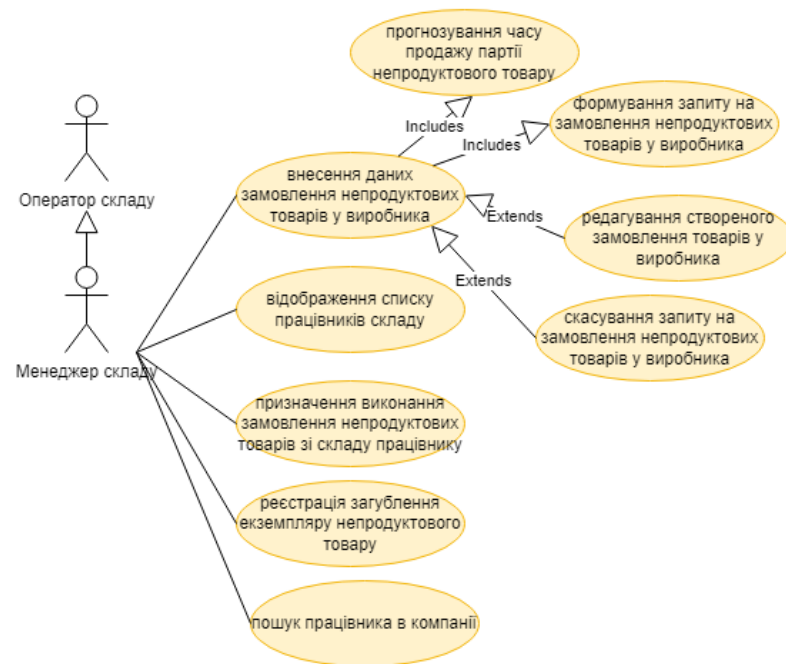


Рисунок 3.7 – Діаграма прецедентів менеджера складу

### 3.3 Проектування структури бази даних

Підрозділ містить опис створеної структури бази даних (БД).

Таблиця `nogrossystem_category` представляє категорії непродуктових товарів, що необхідно, щоб структурувати доступ до товарів, розділивши їх на певні групи, якими на верхньому рівні і є категорії. Таблиця `nogrossystem_subcategory` представляє підкатегорії непродуктових товарів, які вже є другим рівнем структуризації доступу до даних товарів.

Категорії та підкатегорії необхідно тільки для групування товарів, а вже рівнем нижче розпочинаються безпосередньо самі товари, які у свою чергу також було структуровано. Найвищим рівнем є таблиця `nogrossystem_commodity`, яка представляє сам непродуктовий товар загалом. Тобто це не екземпляр цього товару, а товар (фактично загальний опис товару, який продається в мережі). Середнім рівнем є таблиця `nogrossystem_commodityvariant`, яка відповідає деякому варіанту непродуктового товару. Варіант – це певна варіація товару (наприклад, певного кольору, розміру тощо). Найнижчим рівнем є таблиця `nogrossystem_commodityinstance`, яка відповідає конкретному екземпляру непродуктового товару. Тобто в самому товарі зберігається узагальнене представлення товару, у варіанті товару – певна варіація цього товару фактично за додатковими характеристиками, які у різних варіацій можуть відрізнятися, а у екземплярі – конкретний фізичний об'єкт, який може бути продано клієнту в підсумку.

Оскільки непродуктові товари – це фактично узагальнююча сутність, а такими товарами є різні товари, які є достатньо різними між собою, тобто мають дуже різні властивості, то недоцільно вносити властивості товару як поля таблиці, а варто було виносити їх окремо. Для цього і була створена таблиця `nogrossystem_property`, яка використовується для збереження можливих характеристик непродуктових товарів, з яких для конкретного товару або його варіанту далі вибираються ті, які його характеризують.

Відповідно таблиця `nogrossystem_commodityproperty` представляє властивості конкретного непродуктового товару і значення цих властивостей (матеріал, країна виробництва, колір тощо). Однак частина властивостей має описові значення, а частина – числові (цілі числа або числа з дрібною частиною). У таблиці `nogrossystem_commodityproperty` представлені описові значення, які якщо мають порівнюватися, то порівнюються як рядки. Якщо деяка властивість має числове значення (висота, ширина, вага, кількість сходинок), то значення таких властивостей для непродуктового товару знаходяться в таблиці `nogrossystem_commodityfloatproperty` (властивості конкретного непродуктового товару у випадку числових значень).

Не всі властивості можуть стосуватися загалом товару, а частина може відноситися до варіанта товару, через що взагалі і виділяється варіант товару, а не використовується лише товар. Збереження значень цих властивостей влаштовано аналогічно збереженню властивостей непродуктового товару. Таблиця `nogrossystem_commodityvariantproperty` визначає властивості конкретного варіанта непродуктового товару, тобто додаткових властивостей, які характеризують не товар загалом, а особливості його варіанта. Таблиця `nogrossystem_commodityvariantfloatproperty` визначає властивості конкретного варіанта непродуктового товару, тобто додаткових властивостей, які характеризують не товар загалом, а особливості його варіанта, але у випадку числових значень.

Товари продаються кінцевим клієнтам через магазини. Дані магазинів мережі непродуктових магазинів знаходяться в таблиці БД `nogrossystem_store`. Окрім магазинів ще одним варіантом продажу товарів є дистриб'ютори. У випадку дистриб'ютора далі не контролюється вартість продажу товарів, кінцевий клієнт, а фактично віддаються товари дистриб'ютору для подальшого розповсюдження. Однак дані дистриб'юторів зберігаються разом з магазинами, але в таблиці визначається, чи це магазин, чи це дистриб'ютор.

Товари не одразу потрапляють до магазинів від виробників, а спочатку зберігаються на складах, вже закуплені мережею. Таблиця

`nogrossystem_warehouse` відповідає за збереження даних складів у мережі непродуктових магазинів. Товари зберігаються не на складі загалом, адже потім потрібно для швидкості виконання операцій визначити, де саме товар розташований, щоб швидко його отримати і відправити. Таблиця `nogrossystem_position` для цього визначає місця на складі мережі непродуктових магазинів.

Зі складів товари потрапляють у магазини, але не з кожного складу у кожний магазин може потрапити товар у будь-якій ситуації. Можливо логістика налагоджена таким чином, що товари з деяких складів можуть потрапити у певний магазин. Зв'язки між складами і магазинами зберігаються в таблиці `nogrossystem_warehousestore`.

Кінцевими користувачами програми є клієнти і працівники мережі непродуктових магазинів. Дані клієнтів знаходяться в таблиці `nogrossystem_customer`. Дані працівників відповідно розташовуються у таблиці `nogrossystem_employee`.

Працівник представляє з одного боку мережу загалом, але при цьому він має і конкретне місце роботи, яким виходячи з описаної структури мережі, може бути магазин або склад, якого він представляє. Кожен з них може мати різних працівників, при цьому і працівник через певний час, залишаючись користувачем програми, може перейти на роботу до іншого підрозділу, тому зв'язки між цими підрозділами та працівниками винесено в окремі таблиці.

Таблиця `nogrossystem_employeeestore` визначає задіяність працівника в роботі магазину мережі, тобто він є або оператором, або менеджером цього магазину. Таблиця `nogrossystem_employeeewarehouse` визначає задіяність працівника в роботі складу мережі, тобто він є або оператором, або менеджером цього складу.

Визначаючи відповідального за виконання замовлень, не завжди фактично той самий працівник і виконує цю роботу. Адже працівник може захворіти, знаходитися в іншій зміні тимчасово або постійно. Тому для таких

випадків працівники об'єднуються в команду, щоб у разі необхідності інший учасник такої команди міг замінити іншого працівника. Оскільки команда фактично складається з працівників та не потребує певного опису, назви, то всі дані зберігаються безпосередньо в таблиці `nogrosystem_employeeeteam`, яка забезпечує визначення команди працівників, що підтримують разом роботу мережі непродуктових магазинів.

Замовлення непродуктових товарів можуть бути 3 різних типів: замовлення, які виконуються клієнтами онлайн, замовлення, які виконуються магазинами напряму зі складів для продажу товарів безпосередньо в самих магазинах без доступу онлайн, та замовлення, які здійснюються безпосередньо дистриб'юторами. Останні також реалізуються зі складів, але відмінністю від замовлень магазинів є те, що товари в магазинах потім продаються клієнтам, які не мають або не використовують в цей момент свій обліковий запис у системі, проте цей продаж фіксується в системі. У варіанті ж замовлення дистриб'ютора після виконання такого замовлення більше ніякої інформації про товар не фіксується, тобто в подальшому дистриб'ютор самостійно продає цей товар кінцевим споживачам тоді, коли вважає це за потрібне, підтримки цієї діяльності через систему ніякої не виконується. Відповідно ці замовлення відрізняються не тим, як представляються у БД, а тим, як реалізується взаємодія з ними. Тобто це питання функціональності, а не БД, тому в результаті замовлення було зведено до 2 типів: замовлення, які виконуються клієнтами онлайн, замовлення, які виконуються магазинами напряму зі складів, а серед магазинів, як це було визначено раніше, може бути дистриб'ютор, для якого подальша участь конкретних товарів вже не контролюється.

Таблиця БД `nogrosystem_customerorder` визначає відповідно описані замовлення непродуктових товарів, які виконуються клієнтами онлайн. Саме по собі замовлення є тільки певної узагальнюючою точкою, адже замовлення складається з конкретних непродуктових товарів (фактично варіантів непродуктових товарів), які є позиціями замовлення. Таблиця

`nogrossystem_customerordercommodity` представляє варіант непродуктового товару, який є позицією в замовленні.

Оскільки замовлення доставляються зі складів, то важливо зберігати інформацію про те, з якого саме складу це замовлення має бути доставлено. Однак може так трапитися, що доставку потрібно буде виконувати з декількох складів для формування єдиного замовлення. З цією метою створена таблиця `nogrossystem_customerordercommoditywarehouse`, яка визначає джерело постачання для замовлення клієнта, яким є склад.

У замовленні фактично клієнту видаються не загальні варіанти непродуктових товарів, а його конкретні екземпляри. Ці екземпляри важливо зберігати, щоби потім мати інформацію про історію екземпляра, розуміти, кому і коли він фактично був проданий. Таблиця `nogrossystem_customerordercommodityinstance` містить дані про конкретний екземпляр товару, який стосується позиції в замовленні клієнта.

Другий тип замовлень, яким є замовлення магазином зі складу непродуктових товарів, знаходиться в таблиці `nogrossystem_storeorder`, що визначає замовлення непродуктових товарів, яке виконано самим магазином без участі клієнта. Аналогічно попередньому типу замовлень до нього потрібно також зберігати конкретні позиції, з яких замовлення і складається. Для цього таблиця `nogrossystem_storeordercommodity` зберігає варіант непродуктового товару, який є позицією в замовленні непродуктових товарів, яке виконано самим магазином без участі клієнта. Для того щоб також зафіксувати факт продажу конкретного екземпляра, а ця інформація зокрема необхідна для розуміння меж гарантійного обслуговування, створена таблиця `nogrossystem_storeordercommodityinstance`, яка визначає конкретний екземпляр товару, який стосується позиції в замовленні магазину.

Оскільки замовлення магазину не завершується в момент його створення, доставки у магазин, а після цього товари продаються у фізичному магазині, а сам продаж відбувається пізніше, то продажі непродуктових товарів магазином фізично у самому магазині винесені в таблицю

nogrossystem\_storesale. Позиції такого продажу (фактично позиції чеку) внесені в таблицю nogrossystem\_storesalecommodity. Збереження екземплярів товару, проданого у замовленні, у відповідності з позицією замовлення виконується через таблицю nogrossystem\_storeordercommodityinstance.

Товари потрапляють на склади мережі непродуктових магазинів від виробників. Безпосередньо дані виробників, у яких такі запити на придбання товарів оформлюються та які потрібні мережі непродуктові товари виготовляють, зберігаються в таблиці nogrossystem\_manufacturer.

Від виробників визначаються запити на постачання, оформлені самою мережею і які мають дістатися в підсумку конкретного складу мережі. Таблиця nogrossystem\_application зберігає запити від мережі до виробника непродуктових товарів на постачання. Запит може включати декілька варіантів товарів, тому фактично він представляється окремими позиціями. Таблиця nogrossystem\_applicationcommodity визначає позиції запиту від мережі до виробника непродуктових товарів на постачання. Запит фактично є замовленням, однак таким, яке створюється мережею, забезпечується виробником (або його представниками), а постачається на склад мережі.

Після отримання товарів за запитом від мережі до виробника вони розміщуються на складі. Після прибуття товарів відбувається реєстрація екземплярів товарів, а тоді розміщення на певних місцях складу, відповідно в таблицю nogrossystem\_applicationcommodityinstance вноситься розташування екземпляра непродуктового товару на деякому місці на складі мережі непродуктових магазинів.

Таблиця nogrossystem\_warranty зберігає результати гарантійного обслуговування екземпляру непродуктового товару.

Для реалізації роботи з БД у програмі та створення описаної вище структури БД (рис. 3.8) з врахуванням виявлених зв'язків між цими таблицями було застосовано систему керування БД MySQL.

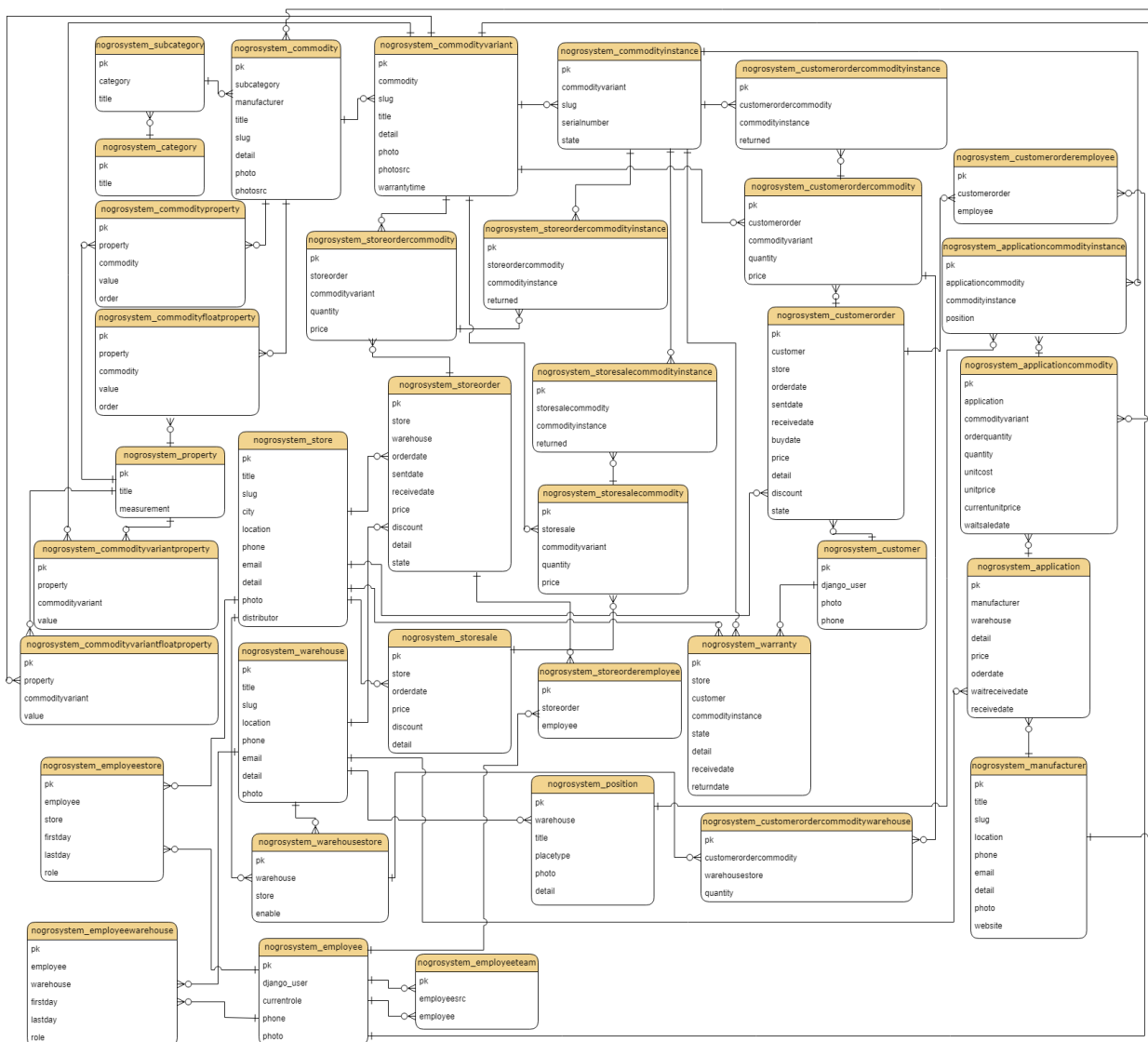


Рисунок 3.8 – Схема БД

### 3.4 Висновки за розділом 3

У розділі 3 виконано повністю завдання проектування програмного забезпечення підтримки роботи мережі непродуктових магазинів:

- виконано порівняння вебфреймворку Django і системи керування вмістом Wordpress, в підсумку Django обрано для реалізації програми;
- реалізовано моделювання сценаріїв роботи оператора, менеджера складу, оператора, менеджера магазину, менеджера мережі, клієнта, клієнта без авторизації;

– визначено сутності, таблиці БД, зв'язки між таблицями, описано, як відповідні рішення було прийнято та пояснено логіку цих рішень, створено схему БД, що відповідає вимогам до програми, зокрема сценаріям роботи з програмою.

# 4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Порядок роботи програмного забезпечення

Екземпляр непродуктового товару за весь час роботи переходить з одного стану до іншого, зокрема входячи до замовлення в певний момент (рис. 4.1).

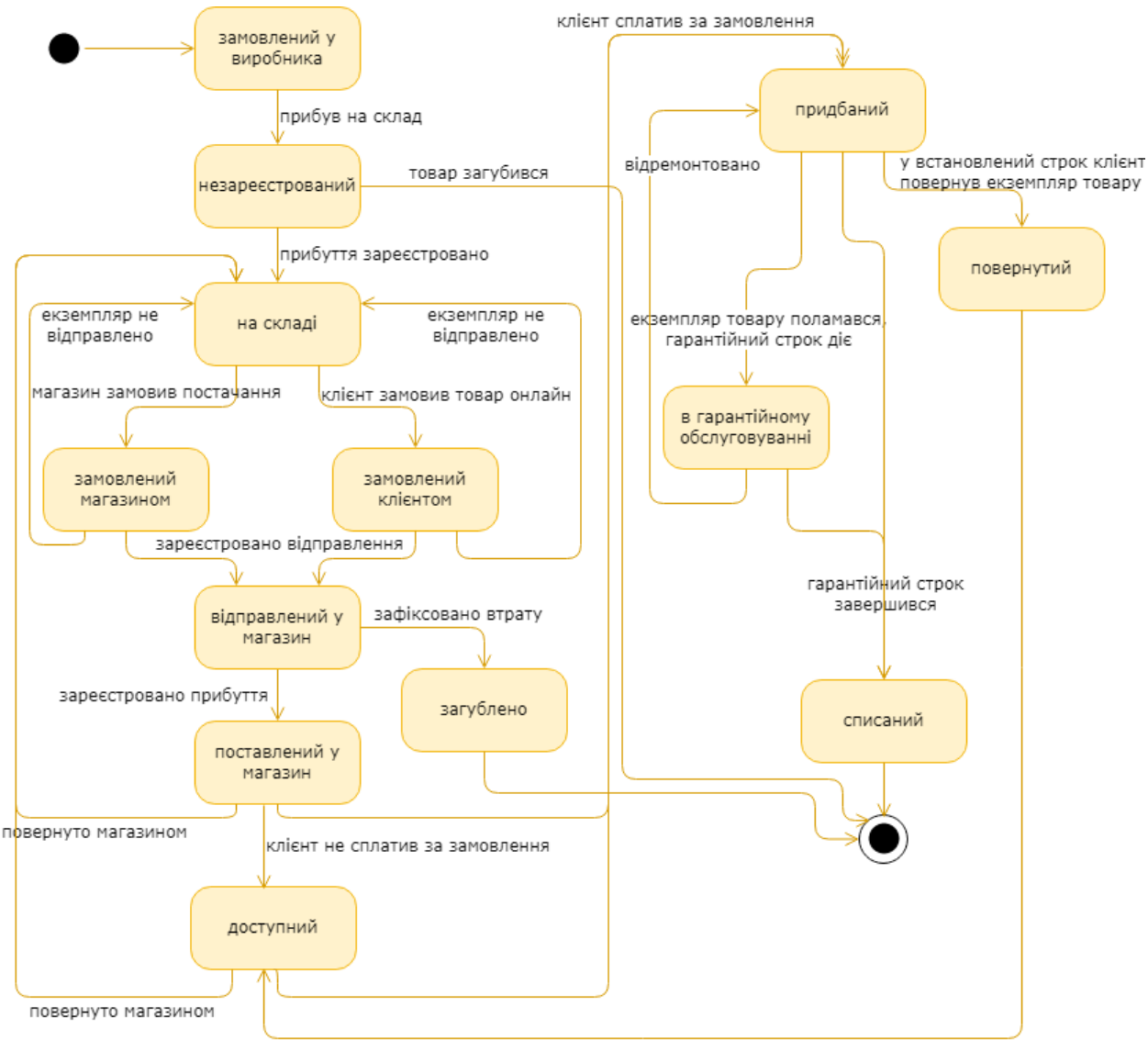


Рисунок 4.1 – Діаграма станів екземпляру непродуктового товару

Об’єкт екземпляру непродуктового товару розпочинає своє існування вже в той момент, коли товар замовлений у виробника. Оскільки

замовляється деяка кількість, то одразу створюється екземпляр товару. Коли товар прибуває на склад, то одразу всі екземпляри товарів з замовлення стають незареєстрованими. Далі зміна стану відбувається тоді, коли буде зареєстровано в програмі конкретний екземпляр товару на складі. Тоді екземпляр стає у стані на складі. Якщо ж один з товарів з партії загублено, то цей екземпляр не буде зареєстровано на складі, а екземпляр завершить своє існування після остаточного підтвердження факту втрати.

Якщо екземпляр товару зареєстрований на складі, то далі він може бути замовлений самим магазином. Тоді один з екземплярів визначається як замовлений магазином. Далі може бути поставлений саме цей екземпляр або інший. Якщо буде відправлено інший за певним замовленням, то цей екземпляр повернеться до стану на складі. Це необхідно для того, щоб не продати товарів більше, ніж наявно. Якщо відправлено саме цей екземпляр, то він переходить до стану відправленого у магазин.

Також екземпляр товару може бути замовлений клієнтом, тоді визначається саме такий стан, а подальші дії аналогічні процедурі, описаній для замовленого магазином.

Далі стандартно екземпляр товару після реєстрації прибуття переходить до стану поставленого у магазин. Якщо ж під час постачання екземпляр товару втрачений і це остаточно зафіксовано, то він переходить до стану загубленого, після цього він перестає використовуватись у програмі.

Після того, як екземпляр поставлено у магазин, далі він може перейти до стану придбаного, якщо клієнт забрав замовлення (якщо товар був саме замовлений). До цього стану екземпляр може також перейти, якщо екземпляр замовлений магазином, але далі був придбаний вже у магазині одним з клієнтів. Якщо екземпляр не було продано тривалий час, то магазин може повернути його на склад. Якщо клієнт не сплатив за замовлення, яке було ним оформлено, то екземпляр може стати знов доступним. Після цього він може або стати придбаним іншим клієнтом, або бути повернутим на склад.

Після придбання екземпляру товару клієнтом він може бути повернутий клієнтом, якщо це зроблено у встановлений строк. Якщо цей строк вийшов, а товар зламався, то екземпляр може перейти до гарантійного обслуговування. Після того, як його буде відремонтовано, екземпляр повертається власнику та стає знов придбаним. Якщо відремонтувати його не вдалося або якщо гарантійний строк завершився, то екземпляр переходить до стану списаного і більше не може змінювати власний стан.

#### **4.2 Рішення з реалізації вебзастосунку**

Враховуючи велику кількість ролей користувачів, які було виділено, різні сценарії роботи, вебзастосунок визначає необхідні шаблони вебсторінок, визначені на основі шаблону [13], але розділяє їх на ряд груп, включаючи:

- шаблони і представлення для формування цих шаблонів, що формують користувацьку підсистему керування мережею магазинів;
- шаблони і представлення для формування цих шаблонів, що формують користувацьку підсистему керування складом;
- шаблони і представлення для формування цих шаблонів, що формують користувацьку підсистему керування магазином;
- шаблони і представлення для формування цих шаблонів, що формують користувацьку підсистему взаємодії з клієнтами.

Усі ці підсистеми реалізовані в єдиному застосунку, розробленому за допомогою вебфреймворку Django, з відділенням моделей, представлень, шаблонів сторінок.

Клас Category забезпечив створення моделі категорії непродуктових товарів. Клас має декілька полів, включаючи pk для визначення первинного ключа з автоматичним збільшенням номера, який у Django створюється для кожної моделі автоматично, та title як поле CharField з назвою самої цієї категорії.

Клас Subcategory забезпечив створення моделі підкатегорії непродуктових товарів.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле category використовується як поле ForeignKey для визначення через зовнішній ключ до попередньої моделі категорії непродуктових товарів, до якої належить ця підкатегорія.

Поле title використовується для визначення на основі поля CharField назви підкатегорії

Клас Commodity забезпечив створення моделі самого непродуктового товару.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле subcategory використовується як поле ForeignKey для визначення через зовнішній ключ до відповідної моделі підкатегорії, до якого належить цей непродуктовий товар.

Поле manufacturer використовується як поле ForeignKey для визначення через зовнішній ключ до відповідної моделі виробника даного непродуктового товару.

Поле title використовується для визначення на основі поля CharField назви товару.

Поле slug використовується для визначення на основі поля SlugField текстової ідентифікації непродуктового товару.

Поле detail використовується для визначення на основі поля TextField текстового опису з розміткою цього непродуктового товару.

Поле photo використовується для визначення основної фотографії непродуктового товару на основі поля ImageField.

Поле photosrc використовується для визначення шляху до теки з іншими фотографіями цього непродуктового товару на основі поля URLField.

Клас `Property` забезпечив створення моделі властивості, яка використовується для характеристики непродуктових товарів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `title` використовується для визначення на основі поля `CharField` назви властивості.

Поле `measurement` використовується для визначення одиниці виміру цієї характеристики непродуктового товару на основі поля `CharField`.

Клас `Commodityvariant` забезпечив створення моделі варіанту непродуктового товару.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `commodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі непродуктового товару, який представляє цей варіант.

Поле `slug` використовується для визначення на основі поля `SlugField` текстової ідентифікації цього варіанта непродуктового товару.

Поле `title` використовується для визначення на основі поля `CharField` назви, що конкретизує цей варіант (якщо назву пропущено, то використовується назва самого непродуктового товару).

Поле `detail` використовується для визначення на основі поля `TextField` текстового опису з розміткою цього варіанта непродуктового товару (якщо значення пропущено, то використовується опис самого непродуктового товару).

Поле `photo` використовується для визначення основної фотографії цього варіанта непродуктового товару на основі поля `ImageField`.

Поле `photosrc` використовується для визначення шляху до теки з іншими фотографіями цього варіанта непродуктового товару на основі поля `URLField`.

Поле `warrantytime` використовується для визначення на основі поля `SmallIntegerField` кількості місяців, протягом якої діє гарантія на цей варіант непродуктового товару.

Клас `Commodityinstance` забезпечив створення моделі конкретного екземпляру непродуктового товару.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого варіанта непродуктового товару, який представляє цей екземпляр.

Поле `slug` використовується для визначення на основі поля `SlugField` текстової ідентифікації конкретного екземпляра непродуктового товару.

Поле `serialnumber` використовується для визначення на основі поля `CharField` серійного номера цього екземпляра непродуктового товару.

Поле `state` використовується для визначення стану, в якому знаходиться екземпляр товару, що вибирається з заданого набору значень, а кодування його реалізується на основі поля `CharField`.

Клас `Commodityproperty` забезпечив створення моделі властивості конкретного непродуктового товару.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `property` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самої властивості цього непродуктового товару.

Поле `commodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого непродуктового товару, до якого відноситься ця властивість.

Поле `value` використовується для визначення значення, яке має ця властивість у цьому непродуктовому товарі, на основі поля `CharField`.

Поле `order` використовується для визначення порядку відображення властивості конкретного непродуктового товару серед всіх властивостей, включаючи числові, на основі поля `PositiveSmallIntegerField`.

Клас `Commodityfloatproperty` забезпечив створення моделі властивості конкретного непродуктового товару у випадку числових значень.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `property` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самої властивості цього непродуктового товару.

Поле `commodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого непродуктового товару, до якого відноситься ця властивість.

Поле `value` використовується для визначення значення, яке має ця властивість у цьому непродуктовому товарі, на основі поля `FloatField`.

Поле `order` використовується для визначення порядку відображення властивості конкретного непродуктового товару серед всіх властивостей, включаючи нечислові, на основі поля `PositiveSmallIntegerField`.

Клас `Commodityvariantproperty` забезпечив створення моделі властивості конкретного варіанта непродуктового товару, тобто додаткових властивостей, які характеризують не товар загалом, а особливості його варіанта.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `property` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самої властивості цього варіанта непродуктового товару.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого варіанта непродуктового товару, до якого відноситься ця властивість.

Поле `value` використовується для визначення значення, яке має ця властивість у цьому варіанті непродуктового товару, на основі поля `CharField`.

Клас `Commodityvariantfloatproperty` забезпечив створення моделі властивості конкретного варіанта непродуктового товару, тобто додаткових властивостей, які характеризують не товар загалом, а особливості його варіанта, але у випадку числових значень.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `property` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самої властивості цього варіанта непродуктового товару.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого варіанта непродуктового товару, до якого відноситься ця властивість.

Поле `value` використовується для визначення значення, яке має ця властивість у цьому варіанті непродуктового товару, на основі поля `FloatField`.

Клас `Warehouse` забезпечив створення моделі складу в мережі непродуктових магазинів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `title` використовується для визначення на основі поля `CharField` назви цього складу.

Поле `slug` використовується для визначення на основі поля `SlugField` текстової ідентифікації складу мережі.

Поле `location` використовується для визначення місця розташування складу мережі на основі поля `CharField`. Оскільки складів умовно небагато у мережі (не тисячі, а швидше і не сотні значень), то і подальшої структуризації адреси непотрібно.

Поле phone використовується для визначення номеру телефону складу мережі на основі поля CharField.

Поле email використовується для визначення поштової скриньки для контактування зі складом на основі поля EmailField.

Поле detail використовується для визначення на основі поля CharField текстового опису цього складу мережі.

Поле photo використовується для визначення фотографії, що візуалізує цей склад мережі, на основі поля ImageField.

Клас Store забезпечив створення моделі магазину мережі непродуктових товарів.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле title використовується для визначення назви магазину на основі поля CharField.

Поле slug використовується для визначення на основі поля SlugField текстової ідентифікації магазину мережі непродуктових товарів.

Поле city використовується для визначення міста розташування магазину мережі на основі поля CharField.

Поле location використовується для визначення місця розташування магазину мережі на основі поля CharField.

Поле phone використовується для визначення номеру телефону магазину мережі на основі поля CharField.

Поле email використовується для визначення поштової скриньки для контактування з магазином на основі поля EmailField.

Поле detail використовується для визначення на основі поля CharField текстового опису цього магазину мережі.

Поле photo використовується для визначення фотографії, що візуалізує цей магазин мережі, на основі поля ImageField.

Поле distributor використовується для визначення статусу магазину (звичайний магазин або дистриб'ютор) на основі поля BooleanField.

Клас Warehousestore забезпечив створення моделі взаємодії між складом і магазином мережі непродуктових магазинів.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле warehouse використовується як поле ForeignKey для визначення через зовнішній ключ до моделі складу, з яким працює напряму магазин мережі непродуктових магазинів.

Поле store використовується як поле ForeignKey для визначення через зовнішній ключ до моделі магазину мережі непродуктових магазинів, з яким працює цей склад.

Поле enable використовується для визначення факту наявності поточної логістики між цим складом і цим магазином або відсутності на основі поля BooleanField.

Клас Customer забезпечив створення моделі клієнта.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле django\_user використовується як поле ForeignKey для визначення через зовнішній ключ до моделі користувача Django, на основі облікового запису якого працює з програмою цей клієнт.

Поле photo використовується для визначення фотографії клієнта на основі поля ImageField.

Поле phone використовується для визначення номеру телефону клієнта на основі поля CharField.

Клас Customerorder забезпечив створення моделі замовлення непродуктових товарів, яке виконано клієнтом.

Поле pk використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле customer використовується як поле ForeignKey для визначення через зовнішній ключ до моделі клієнта, який оформив онлайн для себе це замовлення непродуктових товарів.

Поле `store` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі магазину мережі, на який оформлено доставку цього замовлення для клієнта.

Поле `orderdate` використовується для визначення часу оформлення замовлення непродуктових товарів клієнтом онлайн на основі поля `DateTimeField`.

Поле `sentdate` використовується для визначення часу відправлення замовлення непродуктових товарів зі складу мережі на основі поля `DateTimeField`.

Поле `receivedate` використовується для визначення часу доставки замовлення непродуктових товарів на магазин мережі на основі поля `DateTimeField`.

Поле `buysdate` використовується для визначення часу, коли клієнт забрав своє замовлення непродуктових товарів у магазині, на основі поля `DateTimeField`.

Поле `price` використовується для визначення суми, яку в підсумку коштує замовлення, на основі поля `DecimalField`.

Поле `discount` використовується для визначення знижки, яку було сумарно застосовано до замовлення, на основі поля `DecimalField`.

Поле `detail` використовується для визначення деталей замовлення, сформульованих текстовим описом на основі поля `CharField`.

Поле `state` використовується для визначення стану замовлення непродуктових товарів, яке виконано клієнтом, за вибором з переліку, значення якого кодуються в полі `CharField`.

Поле `employees` використовується для визначення працівників, які відповідальні за реалізацію замовлення на основі поля `ManyToManyField`.

Клас `Customerordercommodity` забезпечив створення моделі непродуктового товару (фактично варіанта товару), який є позицією в замовленні.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `customerorder` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого замовлення, до якого входить цей непродуктовий товар.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого варіанта непродуктового товару, який є позицією у замовленні.

Поле `quantity` використовується для визначення необхідної кількості цього варіанта товару в замовленні.

Поле `price` використовується для визначення суми, яку в підсумку коштує ця позиція замовлення, на основі поля `DecimalField`.

Клас `Customerordercommoditywarehouse` забезпечив створення моделі джерела постачання для замовлення клієнта.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `customerordercommodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі варіанта непродуктового товару, який є позицією у замовленні клієнта.

Поле `warehousestore` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі складу, з якого цей варіант непродуктового товару має бути поставлений.

Поле `quantity` використовується для визначення кількості одиниць товару, які мають бути поставлені з цього складу, на основі поля `PositiveIntegerField`.

Клас `Customerordercommodityinstance` забезпечив створення моделі конкретного екземпляру товару, який стосується позиції в замовленні клієнта.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `customerordercommodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі варіанта непродуктового товару, який є позицією у замовленні клієнта.

Поле `commodityinstance` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі конкретного екземпляра непродуктового товару, який відноситься до цієї позиції у замовленні клієнта.

Поле `returned` використовується для визначення того, чи не було повернуто цей екземпляр товару назад, на основі поля `BooleanField`.

Клас `Employee` забезпечив створення моделі працівника мережі непродуктових магазинів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `django_user` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі користувача Django, на основі облікового запису якого працює з програмою цей працівник.

Поле `currentrole` використовується для визначення посади, яку на даний момент займає цей працівник у мережі непродуктових магазинів, на основі поля `PositiveSmallIntegerField`.

Поле `photo` використовується для визначення фотографії працівника на основі поля `ImageField`.

Поле `phone` використовується для визначення номеру телефону працівника на основі поля `CharField`.

Клас `Employeeteam` забезпечив створення моделі команди працівників, що підтримують разом роботу мережі непродуктових магазинів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `employeesrc` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі працівника, від якого

встановлюється зв'язок у команді (може бути головний працівник, наприклад, менеджер).

Поле `employee` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі працівника, який працює в цій команді.

Клас `EmployeeStore` забезпечив створення моделі задіяності працівника в роботі магазину мережі, тобто він є або оператором, або менеджером цього магазину.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `employee` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі працівника, який працює в магазині мережі непродуктових магазинів.

Поле `store` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі магазину, в якому працює цей працівник.

Поле `firstday` використовується для визначення дати початку роботи працівника в цьому магазині на основі поля `DateField`.

Поле `lastday` використовується для визначення дати закінчення роботи працівника в цьому магазині на основі поля `DateField`.

Поле `role` використовується для визначення посади, на якій працює в цьому магазині працівник (оператор, менеджер).

Клас `EmployeeWarehouse` забезпечив створення моделі задіяності працівника в роботі складу мережі, тобто він є або оператором, або менеджером цього складу.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `employee` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі працівника, який працює на складі мережі непродуктових магазинів.

Поле `warehouse` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі складу, на якому працює цей працівник.

Поле `firstday` використовується для визначення дати початку роботи працівника на цьому складі на основі поля `DateField`.

Поле `lastday` використовується для визначення дати закінчення роботи працівника на цьому складі на основі поля `DateField`.

Поле `role` використовується для визначення посади, на якій працює на цьому складі працівник (оператор, менеджер).

Клас `Storeorder` забезпечив створення моделі замовлення непродуктових товарів, яке виконано самим магазином без участі клієнта.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `store` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі магазину мережі, який створив це замовлення.

Поле `warehouse` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі складу мережі, з якого магазином оформлено доставку цього замовлення.

Поле `orderdate` використовується для визначення часу оформлення замовлення непродуктових товарів магазином на основі поля `DateTimeField`.

Поле `sentdate` використовується для визначення часу відправлення замовлення непродуктових товарів складом на магазин на основі поля `DateTimeField`.

Поле `receivedate` використовується для визначення часу доставки замовлення непродуктових товарів на магазин мережі на основі поля `DateTimeField`.

Поле `price` використовується для визначення суми, яку в підсумку коштує замовлення, на основі поля `DecimalField`.

Поле `discount` використовується для визначення знижки, яку було сумарно застосовано до замовлення, на основі поля `DecimalField`.

Поле `detail` використовується для визначення деталей замовлення, сформульованих текстовим описом на основі поля `CharField`.

Поле `state` використовується для визначення стану замовлення непродуктових товарів, яке виконано магазином, за вибором з переліку, значення якого кодуються в полі `CharField`.

Поле `employees` використовується для визначення працівників, які відповідальні за реалізацію замовлення на основі поля `ManyToManyField`.

Клас `Storeordercommodity` забезпечив створення моделі непродуктового товару (фактично варіанта товару), який є позицією в замовленні непродуктових товарів, яке виконано самим магазином без участі клієнта.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `storeorder` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі замовлення непродуктових товарів, яке виконано самим магазином без участі клієнта і до якого відноситься ця позиція.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі самого варіанта непродуктового товару, який є позицією у замовленні.

Поле `quantity` використовується для визначення необхідної кількості цього варіанта товару в замовленні.

Поле `price` використовується для визначення суми, яку в підсумку коштує ця позиція замовлення, на основі поля `DecimalField`.

Клас `Storeordercommodityinstance` забезпечив створення моделі конкретного екземпляру товару, який стосується позиції в замовленні магазину.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `commodityinstance` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі конкретного екземпляра

непродуктового товару, який відноситься до цієї позиції у замовленні магазину.

Поле `storeordercommodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі варіанта непродуктового товару, який є позицією у замовленні магазину.

Поле `returned` використовується для визначення того, чи не було повернуто цей екземпляр товару назад, на основі поля `BooleanField`.

Клас `Storesale` забезпечив створення моделі продажу непродуктових товарів магазином фізично у самому магазині.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `store` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі магазину, у якому відбувся цей продаж.

Поле `orderdate` використовується для визначення часу продажу непродуктових товарів магазином на основі поля `DateTimeField`.

Поле `price` використовується для визначення вартості продажу на основі поля `DecimalField`.

Поле `discount` використовується для визначення знижки, яку було сумарно застосовано до продажу, на основі поля `DecimalField`.

Поле `detail` використовується для визначення деталей продажу, сформульованих текстовим описом на основі поля `CharField`.

Клас `Storesalecommodity` забезпечив створення моделі позиції непродуктового товару в замовленні, яким оформлено продаж у магазині.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `storesale` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі продажу непродуктових товарів магазином фізично у самому магазині, до якого належить ця позиція.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі варіанта непродуктового товару, який формує цю позицію продажу.

Поле `quantity` використовується для визначення необхідної кількості цього варіанта товару в замовленні.

Поле `price` використовується для визначення суми, яку в підсумку коштує ця позиція замовлення, на основі поля `DecimalField`.

Клас `Storesalecommodityinstance` забезпечив створення моделі екземпляра непродуктового товару у замовленні, реалізованому під час продажу фізично у магазині мережі.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `storesalecommodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі позиції непродуктового товару в замовленні, яким оформлено продаж у магазині і до позиція в якому відноситься цей екземпляр.

Поле `commodityinstance` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі екземпляра непродуктового товару.

Поле `returned` використовується для визначення того, чи не було повернуто цей екземпляр товару назад, на основі поля `BooleanField`.

Клас `Manufacturer` забезпечив створення моделі виробника непродуктових товарів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `title` використовується для визначення назви виробника непродуктових товарів на основі поля `CharField`.

Поле `slug` використовується для визначення на основі поля `SlugField` текстової ідентифікації виробника непродуктових товарів.

Поле `photo` використовується для визначення фотографії, що візуалізує виробника непродуктових товарів, на основі поля `ImageField`.

Поле `phone` використовується для визначення номеру телефону виробника непродуктових товарів на основі поля `CharField`.

Поле `location` використовується для визначення місця розташування виробництва на основі поля `CharField`.

Поле `website` використовується для визначення адреси вебсайту виробника непродуктових товарів на основі поля `URLField`.

Поле `email` використовується для визначення поштової скриньки для контактування з виробником непродуктових товарів на основі поля `EmailField`.

Поле `detail` використовується для визначення на основі поля `CharField` текстового опису цього виробника непродуктових товарів.

Клас `Application` забезпечив створення моделі запиту від мережі до виробника непродуктових товарів на постачання.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `manufacturer` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі виробника, якому адресований цей запит.

Поле `warehouse` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі складу, на який має бути поставлено товари.

Поле `detail` використовується для визначення на основі поля `CharField` текстового опису цього запиту.

Поле `price` використовується для визначення сумарної вартості запиту на основі поля `DecimalField`.

Поле `oderdate` використовується для визначення часу оформлення запиту на основі поля `DateTimeField`.

Поле `waitreceivedate` використовується для визначення очікуваного часу отримання товарів за запитом від виробника на основі поля `DateTimeField`.

Поле `receivedate` використовується для визначення часу отримання товарів за запитом від виробника на основі поля `DateTimeField`.

Клас `Position` забезпечив створення моделі місця на складі мережі непродуктових магазинів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `warehouse` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі складу, на якому знаходиться це місце.

Поле `title` використовується для визначення назви місця, якщо його потрібно якось спеціалізувати (може бути і окремий номер місця на складі), на основі поля `CharField`.

Поле `placetype` використовується для визначення типу цього місця на складі на основі поля `CharField`, що дозволяє зберегти вибір з наданого набору варіантів.

Поле `photo` використовується для визначення фотографії, яка може вказувати це місце за наявності у нього особливостей і можливості проілюструвати фото, на основі поля `ImageField`.

Поле `detail` використовується для визначення текстового опису місця на складі на основі поля `CharField`.

Клас `Warranty` забезпечив створення моделі гарантійного обслуговування екземпляру непродуктового товару.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `customer` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі клієнта, який замовив це обслуговування.

Поле `store` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі магазину через який було подано заявку на це гарантійне обслуговування.

Поле `commodityinstance` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі екземпляра непродуктового товару, який подано на гарантійне обслуговування.

Поле `state` використовується для визначення стану гарантійного обслуговування на основі поля `CharField`, адже стани включають обмежений набір варіантів.

Поле `detail` використовується для визначення текстових деталей стосовно виконання гарантійного обслуговування на основі поля `CharField`.

Поле `receivedate` використовується для визначення часу, коли було розпочато обслуговування, на основі поля `DateTimeField`.

Поле `returndate` використовується для визначення часу, коли було закінчено обслуговування, на основі поля `DateTimeField`.

Клас `Applicationcommodity` забезпечив створення моделі позиції запиту від мережі до виробника непродуктових товарів на постачання.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `application` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі запиту від мережі до виробника непродуктових товарів на постачання.

Поле `commodityvariant` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі конкретного варіанта непродуктового товару як позиції у запиті.

Поле `orderquantity` використовується для визначення замовленої кількості цього варіанта товару на основі поля `PositiveIntegerField`.

Поле `quantity` використовується для визначення поставленої у результаті кількості цього варіанта товару на основі поля `PositiveIntegerField`.

Поле `unitcost` використовується для визначення витрат на одиницю цього варіанта непродуктового товару на основі поля `DecimalField`.

Поле `unitprice` використовується для визначення вартості одиниці цього варіанта непродуктового товару на основі поля `DecimalField`.

Поле `currentunitprice` використовується для визначення вартості одиниці цього варіанта непродуктового товару, яка застосовується на поточний момент (може визначати знижку), на основі поля `DecimalField`.

Поле `waitsaledate` використовується для визначення прогнозованої кількості днів, через яку ця позиція непродуктових товарів з запиту буде продана повністю, на основі поля `PositiveSmallIntegerField`.

Клас `Applicationcommodityinstance` забезпечив створення моделі визначення розташування екземпляра непродуктового товару на деякому місці на складі мережі непродуктових магазинів.

Поле `pk` використовується для визначення первинного ключа з автоматичним збільшенням номера.

Поле `applicationcommodity` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі позиції запиту від мережі до виробника непродуктових товарів на постачання, за якою було отримано цей екземпляр непродуктового товару.

Поле `commodityinstance` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі екземпляру непродуктового товару, який розташований на цьому місці на складі.

Поле `position` використовується як поле `ForeignKey` для визначення через зовнішній ключ до моделі місця на складі мережі непродуктових магазинів, де розташовано цей екземпляр непродуктового товару.

Зважаючи на велику кількість сценаріїв, відповідно велику кількість представлень і шаблонів вебсторінок, тільки деякі були виділені нижче.

Моніторинг стану продажу партій за непродуктовим товаром виконано шаблоном вебсторінки `monitor_applications.html` та представленням `monitor_commodity_applications_state`. Усі параметри передаються через GET-

запит. Це назва непродуктового товару та період часу, для якого ці партії мають бути розглянуті, включаючи місяць початку та місяць закінчення. За встановленим інтервалом виділяються замовлення від виробників, які були отримані за відведений період часу і в які входить заданий товар. Тоді визначаються замовлення всіх типів, тобто створені самим клієнтом, виконані магазином, продані магазином, за допомогою чого перевіряється, що вся кількість товару в замовленні продана на основі цих всіх 3 типів замовлень. Якщо це так, то за кожним замовленням у виробника встановлюється кількість товару, кількість товару, продана через замовлення клієнта, замовлення магазину, продажі в магазині.

Аналіз розподілу наявних у мережі непродуктових товарів за складами виконано шаблоном вебсторінки `analyse_warehouse_commodity.html` та представленням `analyse_warehouse_commodity`. Параметри включають тільки період часу, за який ці товари мають бути виділені, тобто місяць початку і кінця. Розподіл визначається за кожен місяць у встановленому інтервалі. Подібна логіка використана для всіх подібних функцій, що передбачають аналіз. За ними дані об'єднані за кожен місяць. За кожним складом визначається кожен товар у замовленнях від виробника за цей період, а далі за кожним складом за кожен місяць для кожного товару визначається кількість замовлених одиниць. Так само на місячній основі об'єднано дані за результатами аналізу продажу непродуктових товарів за всією мережею за каналами продажу `analyse_channel_sales`, аналізу роботи магазинів стосовно наданого прибутку `analyse_store_profit`, аналізу замовлення заданого товару в розрізі магазинів мережі непродуктових магазинів `analyse_commodity_orders`, аналізу часу продажу партій непродуктових товарів за заданий період `analyse_application_time`, аналізу зміни вартості одиниці непродуктового товару за вказаний період часу `analyse_commodity_profit`.

Пошук непродуктових товарів за визначеними параметрами виконано через створення шаблону вебсторінки `search_commodity.html` та відповідного

представлення `search_commodity`. Роботу також організовано через передачу параметрів у запиті, включаючи зокрема назву моделі, виробника, вартість, необхідну кількість, місто пошуку, категорію, підкатегорію непродуктових товарів, до яких цей товар має належати, а також додаткову перевірку у всіх інших полях, включаючи пошук введеного тексту в значеннях властивостей товару, описі товару.

Замовлення непродуктових товарів реалізовано окремо для клієнта через шаблон вебсторінки `order.html` і представлення `create_customer_order` та окремо для магазинів через шаблон `store_order.html` та представлення `create_store_order`, що надається менеджеру магазину. Формування самого кошику, в який додаються товари виконується на стороні клієнта, а представлення реалізують фактично збереження складу замовлення зі вказаними товарами та їх кількістю. При збереженні замовлення відповідно перевіряється можливість замовити таку кількість непродуктового товару зі складу, який доступний для вказано у замовленні магазину. Замовлення оформляється тільки, якщо така кількість товару доступна для цього магазину. Якщо вона доступна, то екземпляри одразу резервуються в системі. Під час реєстрації виконується уточнення цих даних, відповідно початково внесений екземпляр може бути змінений на інший. Реалізація реєстрації прибуття непродуктових товарів за запитом на склад від виробника реалізується на основі представлення `application_register`, реєстрація відправлення замовлення зі складу до магазину реалізується на основі представлення `send_warehouse_register`, реєстрація прибуття замовлення зі складу до магазину реалізується на основі представлення `receiive_store_register`, реєстрація продажу замовлення клієнту реалізується на основі представлення `sale_register`, реєстрація повернення товару з замовлення у відведений законодавством для цього строк виконується на основі представлення `return_register`, реєстрація стану гарантійного обслуговування екземпляру товару виконується на основі представлення `service_register`.

Відображення всіх замовлень від виробника на складі виконується на основі представлення `manufacturer_warehouse_applicaition_orders`, відображення замовлень від виробника, призначених для виконання працівником складу, виконується на основі представлення `employee_warehoouse_application_orders` з додатковою перевіркою в полі `employees` кожного замовлення, яке включає товари з цього запису, наявності авторизованого користувача, відображення всіх нових замовлень на складі реалізується через виділення недоставлених замовлень від виробника на склад, де працює авторизований працівник, на основі представлення `new_application_orders`, відображення всіх отриманих замовлень на основі представлення `received_application_orders`, відображення всіх очікуваних замовлень від виробника (очікувана дата настала, але замовлення не доставлено) на основі представлення `wait_application_orders`. Подібно влаштована робота з іншими типами замовлень у різних станах.

Для реалізації роботи з `quick response (QR)`-кодами використано пакет `qrcode` з допоміжними пакетами для роботи зокрема з зображеннями, включаючи пакет `pillow`.

Реалізацію створення, навчання моделі забезпечено використанням для цього пакету `scikit-learn` з підпакетом `sklearn.ensemble` і класом `RandomForestRegressor` для реалізації методу `Random Forest`. Підпакет `sklearn.tree` містить реалізацію методу `CART`.

### **4.3 Висновки за розділом 4**

Розділ 4 містить послідовність зміни станів екземпляру непродуктового товару у вигляді діаграми станів, охоплюючи таким чином процеси, пов'язані з замовленням непродуктових товарів від клієнта, магазину, гарантійним обслуговуванням, запитуванням товарів від виробника, тобто всіма основними процесами предметної області, містить представлення створених класів розробленого програмного забезпечення.

## **5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ**

### **5.1 Опис програми**

Програма призначена для підтримки роботи мережі непродуктових магазинів у розрізі компанії загалом, окремих складів, окремих магазинів, працівників цих відділень, покупців. Дана функціональність підтримується через замовлення товарів зі складу в магазини, онлайн-замовлення самими покупцями, реєстрацію замовлень від виробників або їх представників та прогнозування часу продажу партії товару.

### **5.2 Експлуатація програми**

Експлуатація програми може виконуватися різними користувачами з початковим доступом до інтерфейсу роботи звичайного клієнта, який має додаткові можливості після авторизації, проте з авторизації є посилання на різні інтерфейси, які включають склад, магазин і менеджмент. Під час переходу до відповідного режиму роботи одразу запитується авторизація, тобто зі сторінки авторизації можна перейти до спеціалізованої сторінки авторизації. Після успішної авторизації розпочинається робота у відповідному режимі. Тож тільки з базовим режимом може працювати звичайний користувач без облікового запису, при спробі доступу до інших режимів він не зможе отримати доступ до функціональності. При цьому без авторизації клієнт має розділ меню «Категорії» для отримання переліку категорій, вибору з обраної категорії підкатегорії та переходу до списку товарів у категорії, розділ меню «Пошук» для визначення параметрів пошуку непродуктових товарів і далі отримання і сортування результатів (рис. 5.1).

Клієнт після авторизації (рис. 5.1) додатково має також розділ «Замовлення» для перегляду власних замовлень з вибором стану замовлення

або всіх одночасно та перегляду замовлених раніше клієнтом товарів. Розділ «Дані» дозволяє змінити власні облікові дані.

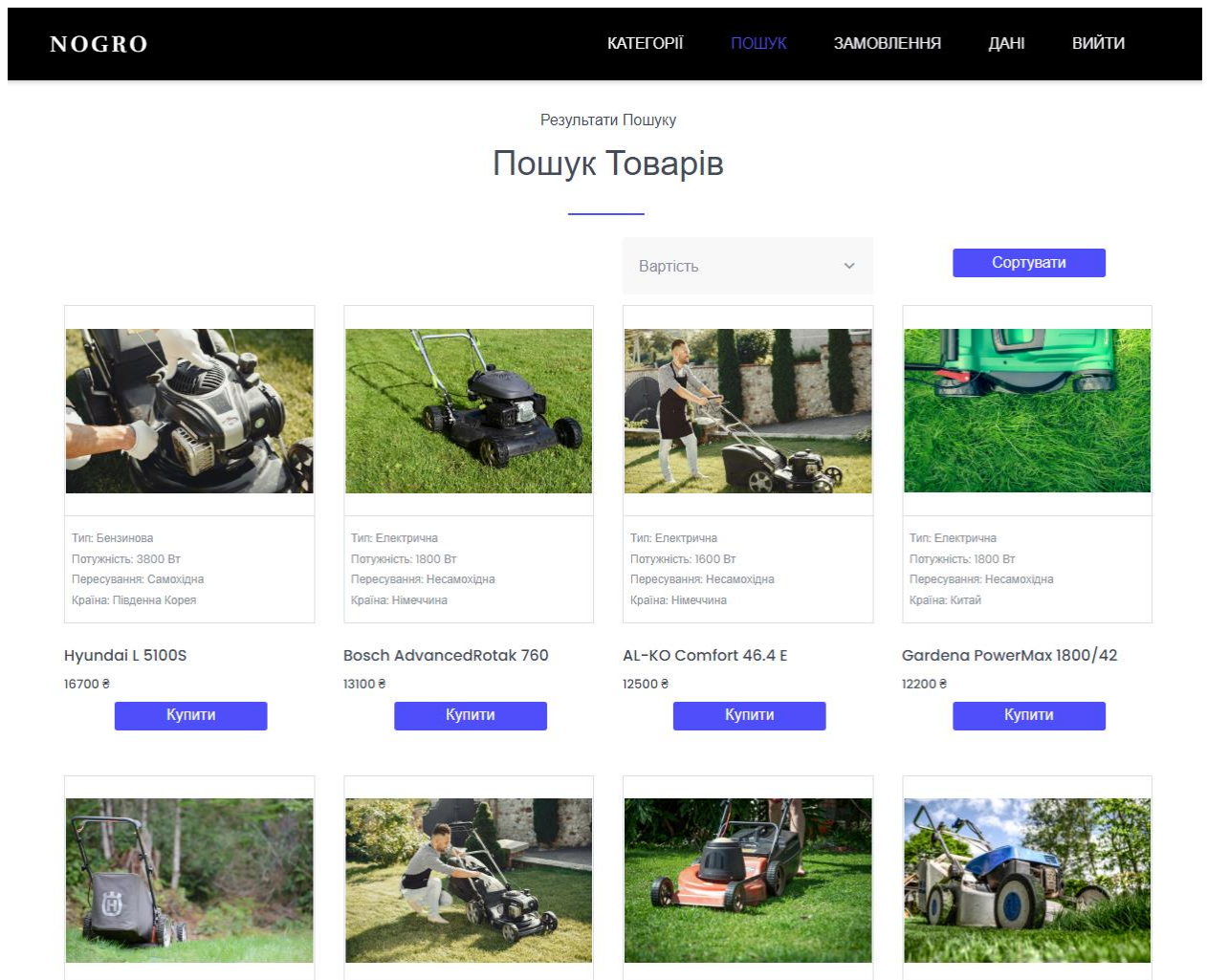


Рисунок 5.1 – Результати пошуку непродуктових товарів

Наступним розглянуто процес роботи з інтерфейсом магазину програми. У цьому інтерфейсі як і в подальших пошук і категорії об'єднано в розділ «Товари», де можна обрати далі один з цих 2 варіантів. Власні замовлення винесено в розділ «Дані» додатковим вибором. Додатковий розділ «Реєстр» направлений на реєстрацію екземплярів товарів. Він присутній в наступних інтерфейсах. Для оператора магазину він дозволяє виконати реєстрацію прибуття товару, продаж у магазині, повернення товару, гарантійне обслуговування. Реєстрація може відбуватися введенням даних або скануванням QR-коду. Розділ «Замовлення» дозволяє працювати з

різними станами, але замовлень, які виконані або для цього магазину, або призначені саме для цього оператора.

Уже оператор магазину може переглянути після реєстрації товару або за вже наявними у замовленні екземплярами товарів історію зміни стану конкретного екземпляру (рис. 5.2). З цієї сторінки можна також роздрукувати сформований QR-код. Історія включає всі дії, пов'язані з замовленнями ще від виробника, далі до магазину, продажу клієнту або замовленнями для клієнта на магазин. Усі дії фіксуються, а потім є можливість так їх відслідкувати. Звідси за номером замовлення можна перейти до сторінки самого замовлення, звідки є перелік товарів та інша інформація.

NOGRO
РЕЄСТР
ТОВАРИ
ЗАМОВЛЕННЯ
ДАНІ
ВИЙТИ

Історія

Ванна Акрилова Knief Nice



Роздрукувати

Категорія

Сантехніка

Підкатегорія

Ванни

Виробник

Knief & Co.

Стан

Придбаний

Реєструвати

| Тип Події              | Дата       | Тип Замовлення | Замовлення              |
|------------------------|------------|----------------|-------------------------|
| Прибуття На Склад      | 24.11.2023 | У Виробника    | Номер 4 Від 23.11.2023  |
| Відправлення Зі Складу | 27.11.2023 | Магазин        | Номер 16 Від 27.11.2023 |
| Отримано Магазином     | 28.11.2023 | Магазин        | Номер 16 Від 27.11.2023 |

Рисунок 5.2 – Відображення історії зміни станів екземпляру товару

Для менеджера магазину доступний також розділ «Працівники», звідки можна виконати пошук по всім працівникам мережі або переглянути

список працівників цього магазину.

У режимі складу замовлення вже охоплюють і замовлення у виробника (рис. 5.3), і замовлення магазину, і замовлення клієнта. Саме ці варіанти вказані у верхньому рядку вибору. Для менеджера складу додатково доступний варіант «Замовити», що дозволяє створити нове замовлення у виробника. Це фактично внесення даних замовлення, адже самі виробники у програмі не присутні.

Рисунок 5.3 – Відображення замовлень непродуктових товарів у виробника

Розділ «Склад» надає можливість переглянути кожне місце на складі і отримати інформацію про товар, який має знаходитись за цим місцем. Розділ «Магазини» містить перелік магазинів, з яким працює цей склад.

У режимі менеджменту з програмою взаємодіє тільки менеджер мережі магазинів непродуктових товарів. У менеджера додаються розділи «Монітор», що об'єднує функції моніторингу, «Аналіз», що об'єднує функції аналізу (рис. 5.4).

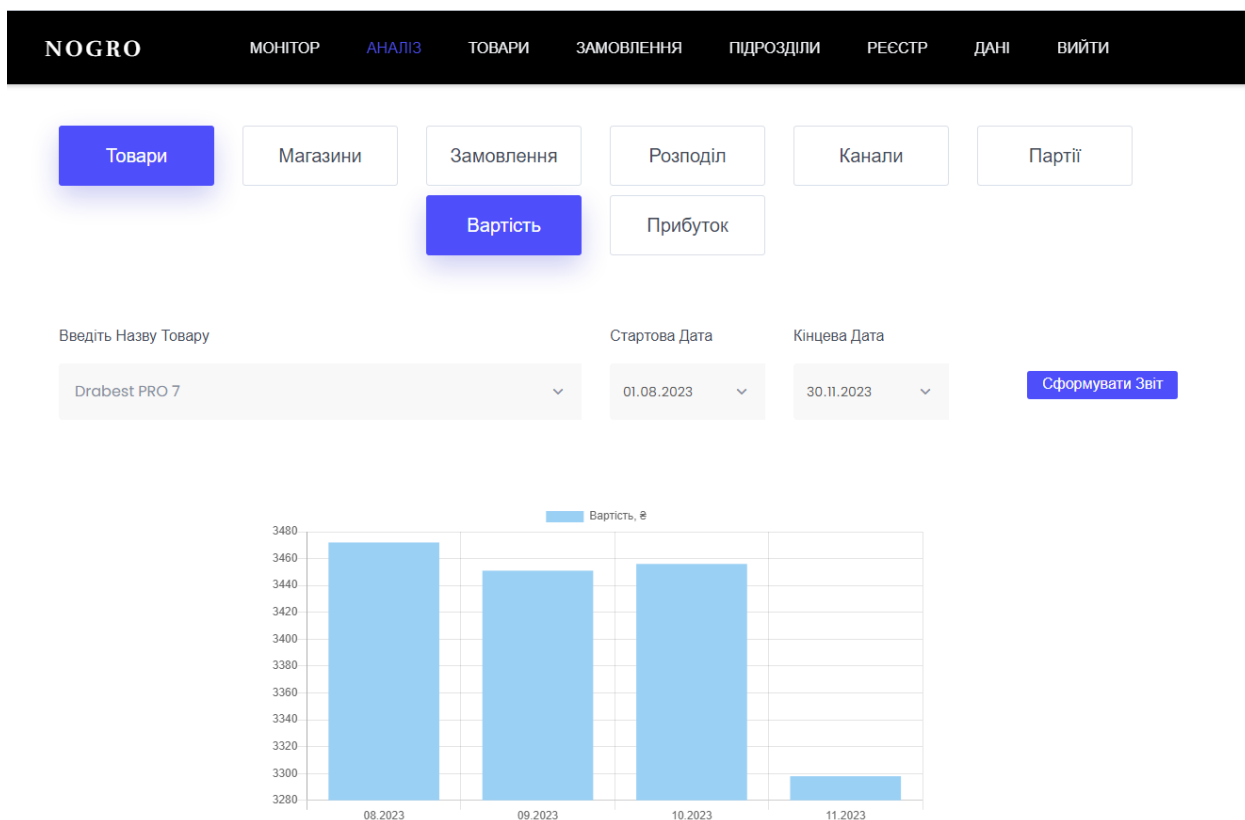


Рисунок 5.4 – Аналіз зміни вартості непродуктового товару

За функціями аналізу або моніторингу доступний набір пунктів, виділених у меню, а за обраним пунктом може відкриватися додатковий вибір у другому рядку. Обираючи варіант у першому рядку пунктів, потім потрібно обрати варіант у другому, а тоді відображається варіант аналізу або моніторингу.

Для менеджера також визначено розділ «Підрозділи», що об'єднує засоби, які доступні для магазину і складу в попередньому меню, але для зменшення кількості пунктів для менеджера їх об'єднано. Окрім того зі сторінок менеджера доступні можливості редагування даних, внесення нових даних. Це ж стосується, наприклад, перегляду товарів за категоріями, пошуку товарів. Зі сторінок товарів доступні саме можливості редагування, зняття з продажу.

При формуванні замовлення до виробника спочатку потрібно обрати виробника, дату поставки, а тоді буде автоматично визначено всі замовлення на товар від магазинів, які стосуються цього виробника і які не закріплені за

товаром на складах, тобто ті, що потребуються магазинами, але відсутні на складі. Далі кількість таких товарів можна змінити, також потрібно встановити дисконт, який планується в середньому надавати, вартість продажу товару клієнтам, вартість придбання у виробника (це значення використовується тільки для обчислення сумарної вартості замовлення). З будь-якими змінами обчислюється прогнозована на основі створеної моделі кількість днів, за яку буде продано все замовлення. З цієї кількості визначається дата, яка виводиться знизу (рис. 5.5).

NOGRO

РЕЄСТР

ТОВАРИ

ЗАМОВЛЕННЯ

СКЛАД

МАГАЗИНИ

ДАНІ

ВИЙТИ

Виробник

Магазини

Клієнти

Замовити

Нове Замовлення У Виробника

Виробник

Потрібна Дата

Всі

01.12.2023

Формування Запиту

| Варіант Товару                             | Вартість, €                   | Кількість | Дисконт, % | Операція                               |
|--------------------------------------------|-------------------------------|-----------|------------|----------------------------------------|
| Turister Bogema 3 Задньою LED Підсвіткою   | Продаж: 3200<br>Закупка: 2500 | 2         | 15         | <div>Змінити</div> <div>Вилучити</div> |
| Turister Bogema 3 Передньою LED Підсвіткою | Продаж: 2800<br>Закупка: 2100 | 2         | 15         | <div>Змінити</div> <div>Вилучити</div> |

Сумарна Вартість Партії: 9200 €

Очікувана Дата Реалізації Партії: 12.04.2024

Додати Товар

Кількість

Продаж

Вибрати

введіть

введіть

Закупка

Дисконт

введіть

введіть

Додати

Рисунок 5.5 – Формування запиту і замовлення у виробника

### 5.3 Експериментальне дослідження

Вибірка даних [14], що представляє дані продажів у мережі магазинів непродуктових товарів, використана для експериментального дослідження. Спочатку виконано дослідження впливу набору вхідних ознак на результати прогнозування на основі методу CART і методу прогнозування часу продажу партії непродуктового товару зі складу (табл. 5.1).

Таблиця 5.1 – Дослідження впливу вхідних ознак на результати

| Вхідні ознаки                                                   | CART       | Метод<br>прогнозування часу<br>продажу партії<br>непродуктового<br>товару зі складу |
|-----------------------------------------------------------------|------------|-------------------------------------------------------------------------------------|
| Початок, кількість, товар, склад,<br>витрати, вартість, дисконт | 0,03559024 | 0,03383645                                                                          |
| Початок, кількість, товар, витрати,<br>вартість, дисконт        | 0,03500126 | 0,03260553                                                                          |
| Початок, кількість                                              | 0,03405903 | 0,03989508                                                                          |
| Початок, кількість, товар                                       | 0,03457473 | 0,03309797                                                                          |
| Початок, кількість, товар, витрати,<br>вартість                 | 0,03500126 | 0,03295672                                                                          |
| Початок, кількість, товар, витрати                              | 0,03457473 | 0,03321438                                                                          |
| Початок, кількість, товар, вартість                             | 0,03500126 | 0,03243525                                                                          |
| Початок, кількість, товар, витрати,<br>дисконт                  | 0,03457473 | 0,03269814                                                                          |
| Початок, кількість, товар, вартість,<br>дисконт                 | 0,03500126 | 0,03221691                                                                          |
| Початок, кількість, товар, дисконт                              | 0,03457473 | 0,03250969                                                                          |

Найкращі результати для методу CART, тобто найменшу середньоквадратичну похибку, було отримано при використанні тільки 2 вхідних ознак. Для методу прогнозування часу продажу партії непродуктового товару зі складу найкращі результати продемонстрував набір вхідних ознак, що включає час початку, кількість товару в партії, категорію товару, вартість продажу одиниці, дисконт продажу. При цьому саме метод мав меншу середньоквадратичну похибку.

У тестові дані в підсумку були віднесені 36 %, відповідно 64 % віднесені до навчальних даних. Дані обирались за кожною індивідуальною парою складу та товару. З усіх дат частина відводилась на навчання, а останні дати – на тестування.

Для методу Random Forest далі досліджено вплив максимальної глибини дерева на середньоквадратичну похибку (табл. 5.2). Найкращий результат отримано при глибині рівній 8 і найкращому наборі вхідних ознак.

Таблиця 5.2 – Дослідження впливу максимальної глибини на результати

| Максимальна глибина<br>дерева | Метод прогнозування часу продажу партії<br>непродуктового товару зі складу |
|-------------------------------|----------------------------------------------------------------------------|
| 2                             | 0,03462187                                                                 |
| 3                             | 0,03361279                                                                 |
| 4                             | 0,03294665                                                                 |
| 5                             | 0,03256456                                                                 |
| 6                             | 0,03232888                                                                 |
| 7                             | 0,03221691                                                                 |
| 8                             | 0,03220835                                                                 |
| 9                             | 0,03227352                                                                 |
| 10                            | 0,03236691                                                                 |

Окремо перевірено вплив кількості дерев на середньоквадратичну похибку (табл. 5.3). Найменше значення похибки при використанні 3000 дерев.

Таблиця 5.3 – Дослідження впливу кількості дерев у ансамблі на результати

| Кількість дерев у ансамблі | Метод прогнозування часу продажу партії непродуктового товару зі складу |
|----------------------------|-------------------------------------------------------------------------|
| 10                         | 0,03389379                                                              |
| 100                        | 0,03220833                                                              |
| 500                        | 0,03228161                                                              |
| 1000                       | 0,03225975                                                              |
| 2000                       | 0,03221671                                                              |
| 3000                       | 0,03216069                                                              |
| 4000                       | 0,03220782                                                              |
| 5000                       | 0,03219613                                                              |
| 6000                       | 0,03221086                                                              |
| 7000                       | 0,03220835                                                              |
| 8000                       | 0,03222869                                                              |
| 9000                       | 0,03221922                                                              |
| 10000                      | 0,03222403                                                              |

У підсумку розглянуто декілька результатів використання методу CART за різних вхідних ознак та використання Random Forest за різних вхідних ознак, різної кількості дерев та різної максимальної глибини (табл. 5.4). Запропонований метод прогнозування часу продажу партії непродуктового товару зі складу дозволив покращити похибку на 5,6 % порівняно з найкращим результатом CART.

Таблиця 5.4 – Підсумкові результати

| Характеристики моделі                                                         | Результат  |
|-------------------------------------------------------------------------------|------------|
| CART (2 вхідні ознаки)                                                        | 0,03405903 |
| CART (3 вхідні ознаки)                                                        | 0,03457473 |
| CART (всі доступні вхідні ознаки)                                             | 0,03559024 |
| CART (вхідні ознаки аналогічно методу)                                        | 0,03500126 |
| Random Forest (3 вхідні ознаки, 3000 дерев, 6 максимальна глибина)            | 0,03251367 |
| Random Forest (всі доступні вхідні ознаки, 7000 дерев, 6 максимальна глибина) | 0,03377447 |
| Метод прогнозування часу продажу партії непродуктового товару зі складу       | 0,03216069 |

#### 5.4 Висновки за розділом 5

Розділ 5 містить опис того, як можна використати визначені для реалізації сценарії роботи оператора, менеджера магазину, складу, менеджера мережі, клієнтів у самій програмі, отримані результати експериментального дослідження. Результати дослідження на вибірці даних продемонстрували ефективність застосування методу прогнозування часу продажу партії непродуктового товару зі складу.

## ВИСНОВКИ

За підсумками роботи розроблено програмне забезпечення підтримки діяльності мережі непродуктових магазинів для забезпечення планування діяльності на майбутні періоди. Основні функції програми визначені з врахуванням наявних аналогів, які є платними, та додаванням тих функцій, які дозволять підвищити ефективність роботи різних користувачів та забезпечити планування діяльності на майбутнє.

Створено програму, яка надає основні можливості, необхідні для операторів і менеджерів складів і магазинів мережі непродуктових магазинів, клієнтів цієї мережі, забезпечуючи засоби моніторингу, аналізу діяльності мережі, прогнозування часу продажу партії товару, що є одним з базових показників для планування діяльності. Реалізовано програму як вебзастосунок, що потребувало використання мови python, фреймворку Django, системи MySQL та інших базових вебзасобів і технологій.

Наукова новизна роботи полягає в методі прогнозування часу продажу партії непродуктового товару зі складу, який виконує прогнозування кількості днів, протягом яких буде розпродано весь товар з партії на складі, використовуючи ансамблі дерев рішень, створені за методом Random Forest, і визначаючи обчислення ряду вхідних ознак для створення моделі прогнозування. Визначено задачу прогнозування часу продажу партії непродуктового товару зі складу, виконано опис дерев рішень, які можуть використовуватися для прогнозування, та представлено сам метод прогнозування часу продажу партії непродуктового товару зі складу. Перевірку роботи методу виконано за допомогою проведення експериментального дослідження на основі вибірки даних, отриману з відкритого доступу, застосовуючи метод, який запропоновано, та логіку даного методу на основі дерев рішень за алгоритмом CART. Виконано порівняння точності прогнозування: більш точний варіант з використанням Random Forest представлений у підсумковому методі.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NetSuite Supply Chain Management (CSMS) [Electronic resource]. – Access mode : <https://www.netsuite.com/portal/products/erp/supply-chain-management.shtml>.
2. McCue, I. What is NetSuite ERP & How Does It Work? [Electronic resource] / I. McCue. – Access mode : <https://www.netsuite.com/portal/resource/articles/erp/what-is-netsuite.shtml>.
3. Needleman, T. Oracle NetSuite OneWorld Review [Electronic resource] / T. Needleman, R. Marvin. – Access mode : <https://www.pcmag.com/reviews/oracle-netsuite-oneworld>.
4. eCommerce Logistics Made Easy. Prime Penguin [Electronic resource]. – Access mode : <https://www.primepenguin.com/>.
5. Inventory forecasting – Prime Penguin [Electronic resource]. – Access mode : <https://www.primepenguin.com/knowledge-base/product-management/inventory-forecasting/>.
6. eCommerce Integration Platform. Prime Penguin [Electronic resource]. – Access mode : <https://www.primepenguin.com/product/>.
7. CART (Classification And Regression Tree) in Machine Learning [Electronic resource]. – Access mode : <https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/>.
8. Decision Trees in Machine Learning [Electronic resource] : Two Types. – Access mode : <https://www.coursera.org/articles/decision-tree-machine-learning>.
9. Waraich, K. CART (Classification And Regression Tree) in Machine Learning [Electronic resource] / K. Waraich. – Access mode : <https://www.codingninjas.com/studio/library/classification-and-regression-tree-algorithm>.
10. Sruthi, E. R. Understand Random Forest Algorithms With Examples [Electronic resource] / E. R. Sruthi. – Access mode :

<https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/#:~:text=A.%20Random%20Forest%20is%20a,of%20all%20the%20different%20models>.

11. Martelli, A. Python in a Nutshell. A Desktop Quick Reference [Text] / Alex Martelli, Anna Martelli, Steve Holden. – Sebastopol : O'Reilly Media, 2017. – 772 p.

12. Blog Tool, Publishing Platform, and CMS – WordPress.org [Electronic resource]. – Access mode : <https://wordpress.org/>.

13. Carvilla Free Website Template [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page296/carvilla>.

14. US Regional Sales Data [Electronic resource]. – Access mode : <https://data.world/dataman-udit/us-regional-sales-data>.

**ДОДАТОК А**  
**Технічне завдання**

## **Вступ**

Мережа непродуктових магазинів є достатньо складним механізмом, у роботі якого задіяно багато користувачів, що мають загалом велику кількість ролей. Робота зі всіма екземплярами товарів вимагає контролю на великій кількості етапів, але і аналіз даних про продажі загалом також потребує великої кількості інструментів. Тільки в такому випадку можна створити програмне забезпечення, яке дозволить цілісно підтримувати роботу мережі непродуктових магазинів. Впровадження інструментів прогнозування до такого програмного забезпечення дозволить підвищити ймовірність її застосування на практиці. Тому в програмі потрібно реалізувати обов'язково прогнозування часу продажу партії товару.

### **A.1 Підстави для розробки**

Наказ № 437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка» з зазначенням теми роботи «Дослідження та програмна реалізація методів підтримки планування діяльності мережі непродуктових магазинів» та завдання на роботу є підставами для розробки.

### **A.2 Призначення розробки**

Розробка призначена для підтримки роботи мережі непродуктових магазинів у розрізі компанії загалом, окремих складів, окремих магазинів, працівників цих відділень, покупців. Дана функціональність має підтримуватися через замовлення товарів зі складу в магазини, онлайн-замовлення самими покупцями, реєстрацію замовлень від виробників або їх представників та прогнозування часу продажу партії товару.

### **А.3 Основні вимоги до програми**

#### **А.3.1 Вимоги до функціональних характеристик**

Забезпечення функціональних характеристик повинно вкладатися в наступний набір функцій:

- формування запиту на замовлення непродуктових товарів у виробника;
- внесення даних замовлення непродуктових товарів у виробника;
- відображення замовлень непродуктових товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- деталізація замовлення непродуктових товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- скасування запиту на замовлення непродуктових товарів у виробника;
- формування замовлення непродуктових товарів зі складу в магазин, в тому числі якщо товари відсутні на складі;
- відображення замовлень непродуктових товарів зі складу у відповідності з рівнем доступу та обов'язками відповідного користувача;
- призначення виконання замовлення непродуктових товарів зі складу працівнику;
- деталізація замовлення непродуктових товарів зі складу мережі;
- скасування замовлення непродуктових товарів зі складу мережі;
- реєстрація прибуття непродуктових товарів за запитом на склад від виробника, включаючи реєстрацію конкретних екземплярів товару, визначення розташування цих екземплярів на складі;
- реєстрація відправлення замовлення зі складу до магазину, включаючи реєстрацію конкретних екземплярів товару;
- реєстрація прибуття замовлення зі складу до магазину, включаючи реєстрацію конкретних екземплярів товару;

- реєстрація продажу замовлення клієнту з фіксацією конкретних екземплярів товарів;
- реєстрація повернення товару з замовлення у відведений законодавством для цього строк;
- реєстрація стану гарантійного обслуговування екземпляру товару;
- відображення замовлень у магазині в розрізі станів;
- редагування інформації про непродуктовий товар, що був внесений до каталогу;
- редагування створеного замовлення товарів зі складу мережі у відповідності з рівнем доступу та обов'язками відповідного користувача;
- редагування створеного замовлення товарів у виробника у відповідності з рівнем доступу та обов'язками відповідного користувача;
- визначення складу мережі, непродуктові товари з якого доступні магазину;
- відображення магазинів мережі, що працюють з заданим складом;
- закриття доступу магазину до складу мережі непродуктових магазинів;
- створення нового непродуктового товару, доступного для замовлення;
- закриття непродуктового товару для нових замовлень;
- повернення непродуктового товару як доступного для замовлення;
- відображення замовлень клієнта в розрізі станів, включаючи нові, виконані, скасовані, усі;
- відображення списку непродуктових товарів, які за весь час замовляв клієнт;
- відображення повернень екземплярів непродуктових товарів клієнтом;
- відображення екземплярів непродуктових товарів, що знаходяться у гарантійному обслуговуванні на дану мить;
- пошук непродуктових товарів за визначеними параметрами;

- відображення даних непродуктового товару;
- відображення історії зміни станів екземпляру товару;
- переміщення непродуктового товару на складі мережі;
- замовлення непродуктових товарів;
- скасування невиконаного замовлення непродуктових товарів;
- підтвердження створеного клієнтом замовлення для доставки зі складу;
- скасування створеного клієнтом замовлення для доставки зі складу;
- авторизація користувачів у всіх можливих ролях;
- реєстрація нового менеджера складу мережі непродуктових магазинів;
- реєстрація нового оператора складу мережі непродуктових магазинів;
- реєстрація нового менеджера магазину мережі непродуктових магазинів;
- реєстрація нового оператора магазину мережі непродуктових магазинів;
- реєстрація нового менеджера мережі непродуктових магазинів;
- змінювання місця роботи користувача, в тому числі переведення між складом, магазином, компанією загалом або ролями оператора або менеджера всередині цих відділень;
- відображення списку працівників складу за заданим складом або за складом, де працює користувач;
- відображення списку працівників магазину за заданим магазином або за магазином, де працює користувач;
- відображення списку працівників компанії за можливості такого доступу;
- реєстрація загублення екземпляру непродуктового товару;
- реєстрація повернення екземпляру непродуктового товару на склад;
- пошук працівника в компанії;

- оновлення персональних даних;
- оформлення продажу непродуктових товарів у магазині без замовлення онлайн;
- відображення власних замовлень;
- перегляд включених підкатегорій непродуктових товарів;
- перегляд непродуктових товарів, що продаються у мережі, за підкатегорією;
- моніторинг стану продажу партій за непродуктовим товаром;
- моніторинг повернень товарів за магазином мережі непродуктових магазинів;
- моніторинг продажів непродуктових товарів у розрізі за магазином мережі;
- моніторинг товарів на складі мережі непродуктових магазинів;
- моніторинг товарів у магазині мережі непродуктових магазинів;
- моніторинг непродуктових товарів магазину за категоріями;
- аналіз розподілу наявних у мережі непродуктових товарів за складами;
- аналіз продажу непродуктових товарів за всією мережею за каналами продажу та загалом;
- аналіз роботи магазинів стосовно наданого прибутку;
- аналіз замовлення заданого товару в розрізі магазинів мережі непродуктових магазинів;
- аналіз часу продажу партій непродуктових товарів за заданий період;
- аналіз зміни вартості одиниці непродуктового товару за вказаний період часу;
- аналіз зміни прибутку, отриманого від продажу заданого непродуктового товару;
- прогнозування часу продажу партії непродуктового товару.

### **A.3.2 Умови експлуатації**

Програмне забезпечення в підсумку має підтримуватись під час експлуатації через:

- організацію хостингу для роботи з сервером;
- розміщення на сервері вебзастосунку з базовим налаштуванням Django;
- наявність інтерпретатора python 3.

Або у складі цього серверу, або як окремий сервер БД має бути використано і налаштовано систему MySQL.

### **A.3.3 Вимоги до складу та параметрів технічних засобів**

Вимоги до організації технічних засобів сервера включають мінімально процесор Dual Intel Xeon, який має частоту 3,2 ГГц, 4 Гб доступної оперативної пам'яті, 10 Гб доступної пам'яті на жорсткому диску.

### **A.4 Порядок контролю та приймання**

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами, результатами та строками. Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно графіком роботи екзаменаційної комісії.

**ДОДАТОК Б**  
**Текст програми**

```

    from .models import Commodity, Applicationcommodity,
    Applicationcommodityinstance, Positioncommodity,
    Storesalecommodityinstance, Storeordercommodityinstance,
    Customerordercommodityinstance, Store, Storesalecommodity,
    Customerordercommodity, Subcategory, Category, Warehouse,
    Customerordercommoditywarehouse
    from django.shortcuts import render
    from django.shortcuts import get_object_or_404
    from datetime import date
    from datetime import timedelta
    from datetime import datetime
    from dateutil.relativedelta import relativedelta

    def monitor_commodity_applications_state(request):
        commoditytitle = self.request.GET.get('commodity')
        commodity_sought = get_object_or_404(Commodity, title =
commoditytitle)
        commodity_applications =
    Applicationcommodity.objects.filter(commodityvariant__commodity =
commodity_sought)
        commodity_applications =
    commodity_applications.filter(application__receivedate__isnull = True)
        restriction = self.request.GET.get('restriction')
        if restriction:
            monitor_start_month = self.request.GET.get('startmonth')
            monitor_start_year = self.request.GET.get('startyear')
            monitor_start_year = int(monitor_start_year)
            monitor_start_month = int(monitor_start_month)
            monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
            monitor_end_month = self.request.GET.get('endmonth')
            monitor_end_year = self.request.GET.get('endyear')
            monitor_end_month = int(monitor_end_month)
            monitor_end_year = int(monitor_end_year)
            monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
            monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
            commodity_applications =
    commodity_applications.filter(application__receivedate__gte =
monitor_start)
            commodity_applications =
    commodity_applications.filter(application__receivedate__lte =
monitor_end)
            commodity_applications = commodity_applications.order_by('-
receivedate')
            monitored = []
            for application_commodity in commodity_applications:

```

```

        instances =
Applicationcommodityinstance.objects.filter(applicationcommodity=
application_commodity).values_list('commodityinstance')
        sale_instances =
Storesalecommodityinstance.objects.filter(commodityinstance__in =
instances, returned = False).count()
        store_instances =
Storeordercommodityinstance.objects.filter(commodityinstance__in =
instance, returned = False).count()
        customer_instances =
Customerordercommodityinstance.objects.filter(commodityinstance__in =
instance, returned = False).count()
        if restriction or ((customer_instances + sale_instances)
!= instances.count()):
            application_commodity.instances = instances.count()
            application_commodity.sale_instances =
sale_instances
            application_commodity.store_instances =
store_instances
            application_commodity.customer_instances =
customer_instances
            monitored.append(application_commodity)
        return render(request,
"nogrossystem/monitor_applications.html", {"applications": monitored})

def monitor_commodity_returns(request):
    commoditytitle = self.request.GET.get('commodity')
    commodity_sought = get_object_or_404(Commodity, title =
commoditytitle)
    storeslug = self.request.GET.get('store')
    store_sought = get_object_or_404(Store, slug = storeslug)
    if commoditytitle:
        commodity_sought = get_object_or_404(Commodity, slug =
commodityslug)
        commodity_sales =
Storesalecommodityinstance.objects.filter(commodityinstance__commodity
variant__commodity = commodity_sought,
storesalecommodity__storesale__store = store_sought, returned = True)
        commodity_orders =
Customerordercommodityinstance.objects.filter(commodityinstance__commo
dityvariant__commodity = commodity_sought,
customerordercommodity__customerorder__store = store_sought, returned
= True)
    else:
        commodity_sales =
Storesalecommodityinstance.objects.filter(commodityinstance__commodity
variant__commodity = commodity_sought,
storesalecommodity__storesale__store = store_sought, returned = True)

```

```

        commodity_orders =
Customerordercommodityinstance.objects.filter(customerordercommodity__
customerorder__store = store_soked, returned = True)
        restriction = self.request.GET.get('restriction')
        if not restriction:
            monitor_end = datetime.now()
            monitor_start = monitor_end - relativedelta(months = 12)
            monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
        else:
            monitor_start_month = self.request.GET.get('startmonth')
            monitor_start_year = self.request.GET.get('startyear')
            monitor_start_year = int(monitor_start_year)
            monitor_start_month = int(monitor_start_month)
            monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
            monitor_end_month = self.request.GET.get('endmonth')
            monitor_end_year = self.request.GET.get('endyear')
            monitor_end_month = int(monitor_end_month)
            monitor_end_year = int(monitor_end_year)
            monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
            monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        commodity_orders =
commodity_orders.filter(customerordercommodity__customerorder__receive
date__gte = monitor_start)
        commodity_orders =
commodity_orders.filter(customerordercommodity__customerorder__receive
date__lte = monitor_end)
        commodity_orders_counts = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            commodity_orders_count =
commodity_orders.filter(customerordercommodity__customerorder__receive
date__lte = monitor_start_end,
customerordercommodity__customerorder__receivedate__gte =
monitor_start_start).count()
            commodity_orders_counts.append(commodity_orders_count)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)
        commodity_sales =
commodity_sales.filter(storesalecommodity__storesale__orderdate__gte =
monitor_start)

```

```

        commodity_sales =
commodity_sales.filter(storesalecommodity__storesale__orderdate__lte =
monitor_end)
        commodity_sales_counts = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = date(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = date(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            commodity_sales_count =
commodity_sales.filter(storesalecommodity__storesale__orderdate__lte =
monitor_start_end, storesalecommodity__storesale__orderdate__gte =
monitor_start_start).count()
            commodity_sales_counts.append(commodity_sales_count)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)

        return render(request, "nogrossystem/monitor_returns.html",
{"orders": commodity_orders_count, "sales": commodity_sales_count})

def monitor_commodity_sales(request):
    commoditytitle = self.request.GET.get('commodity')
    storeslug = self.request.GET.get('store')
    store_sought = get_object_or_404(Store, slug = storeslug)
    if commoditytitle:
        commodity_sought = get_object_or_404(Commodity, title =
commoditytitle)
        commodity_sales =
Storesalecommodity.objects.filter(commodityvariant__commodity =
commodity_sought, storesale__store = store_sought, returned = False)
        commodity_orders =
Customerordercommodity.objects.filter(commodityvariant__commodity =
commodity_sought, customerorder__store = store_sought, returned =
False)
    else:
        commodity_sales =
Storesalecommodity.objects.filter(storesale__store = store_sought,
returned = True)
        commodity_orders =
Customerordercommodity.objects.filter(customerorder__store =
store_sought, returned = True)
        restriction = self.request.GET.get('restriction')
        if not restriction:
            monitor_end = datetime.now()
            monitor_start = monitor_end - relativedelta(months = 12)
            monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
        else:
            monitor_start_month = self.request.GET.get('startmonth')

```

```

        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        commodity_orders =
commodity_orders.filter(customerorder__receivedate__gte =
monitor_start)
        commodity_orders =
commodity_orders.filter(customerorder__receivedate__lte = monitor_end)
        commodity_orders_counts = []
        commodity_orders_quantity = []
        commodity_orders_price = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            commodity_orders_set =
commodity_orders.filter(customerorder__receivedate__lte =
monitor_start_end, customerorder__receivedate__gte =
monitor_start_start)
            commodity_orders_count = commodity_orders_set.count()
            quantity = 0
            price_sum = 0
            for commodity_order in commodity_orders_set:
                quantity = quantity + commodity_order.quantity
                price_sum = price_sum + commodity_order.price
            commodity_orders_counts.append(commodity_orders_count)
            commodity_orders_quantity.append(quantity)
            commodity_orders_price.append(price_sum)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)
        commodity_sales =
commodity_sales.filter(storesale__orderdate__gte = monitor_start)
        commodity_sales =
commodity_sales.filter(storesale__orderdate__lte = monitor_end)
        commodity_sales_counts = []
        commodity_sales_quantity = []
        commodity_sales_price = []

```

```

        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = date(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = date(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            commodity_sales_set =
commodity_sales.filter(storesale__orderdate__lte = monitor_start_end,
storesalecommodity__storesale__orderdate__gte = monitor_start_start)
            commodity_sales_count = commodity_sales_set.count()
            quantity = 0
            price_sum = 0
            for commodity_sale in commodity_sales_set:
                quantity = quantity + commodity_sale.quantity
                price_sum = price_sum + commodity_sale.price
            commodity_sales_counts.append(commodity_sales_count)
            commodity_sales_quantity.append(quantity)
            commodity_sales_price.append(price_sum)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)

        return render(request, "nogrossystem/monitor_sales.html",
            {"orders_count": commodity_orders_counts,
             "orders_quantity": commodity_orders_quantity,
             "orders_price": commodity_orders_price,
             "sales_count": commodity_sales_counts,
             "sales_quantity": commodity_sales_quantity,
             "sales_price": commodity_sales_price,

            })

    def monitor_category_commodities(request):
        categorytype = self.request.GET.get('type')
        if int(categorytype) == 1:
            subcategorypk = self.request.GET.get('subcategory')
            subcategorypk = int(subcategorypk)
            commodity_subcategory = Subcategory.objects.get(pk =
subcategorypk)
        else:
            categorypk = self.request.GET.get('category')
            categorypk = int(categorypk)
            commodity_category = Category.objects.get(pk =
categorypk)
        storeslug = self.request.GET.get('store')
        store_sought = get_object_or_404(Store, slug = storeslug)
        restriction = self.request.GET.get('restriction')
        if not restriction:
            monitor_end = datetime.now()
            monitor_start = monitor_end - relativedelta(months = 12)

```

```

        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        leftinstore = []
        ordered_commodities_list = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            if int(categorytype) == 1:
                storeordercommodities =
Storeordercommodity.objects.filter(commodityvariant__commodity__subcat
egory = commodity_subcategory, storeorder__receivedate__lte =
monitor_start_end)
            else:
                storeordercommodities =
Storeordercommodity.objects.filter(commodityvariant__commodity__subcat
egory__category = commodity_category, storeorder__receivedate__lte =
monitor_start_end)
            totalquantity = 0
            for store_order in storeordercommodities:
                totalquantity = totalquantity + store_order.quantity
            if int(categorytype) == 1:
                sales_set =
Storesalecommodityinstance.objects.filter(storesalecommodity__commodit
yvariant__commodity__subcategory = commodity_subcategory,
storesalecommodity__storesale__orderdate__lte = monitor_start_end,
returned = False)
                commodities_sold =
sales_set.values_list("commodityinstance")
                salescount = sales_set.count()
                ordered_commodities =
Storeordercommodityinstance.objects.filter(storeordercommodity__commod

```

```

ityvariant__commodity__subcategory = commodity_subcategory,
storeordercommodity__storeorder__receivedate__lte =
monitor_start_end).select_related("commodityinstance__commodity").excl
ude(commodityinstance__in =
commodities_sold).values_list("commodityinstance__commodity").distinct
()

        else:
            sales_set =
Storesalecommodityinstance.objects.filter(storesalecommodity__commodit
yvariant__commodity__subcategory__category = commodity_category,
storesalecommodity__storesale__orderdate__lte = monitor_start_end,
returned = False)
            salescount = sales_set.count()
            commodities_sold =
sales_set.values_list("commodityinstance")
            ordered_commodities =
Storeordercommodityinstance.objects.filter(storeordercommodity__commod
ityvariant__commodity__subcategory__category = commodity_category,
storeordercommodity__storeorder__receivedate__lte =
monitor_start_end).select_related("commodityinstance__commodity").excl
ude(commodityinstance__in =
commodities_sold).values_list("commodityinstance__commodity").distinct
()

            instore = totalquantity - salescount
            leftinstore.append(instore)
            ordered_commodities_list.append(ordered_commodities)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)

        return render(request,
"nogrossystem/monitor_category_commodities.html",
            {"commodities_insale": leftinstore,
            "commodities": ordered_commodities_list
            })

def monitor_store_commodities(request):
    storeslug = self.request.GET.get('store')
    store_soked = get_object_or_404(Store, slug = storeslug)
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)

```

```

        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        ordered_commodities_list = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            sales_set =
Storesalecommodityinstance.objects.filter(storesalecommodity__storesal
e__store = store_sought, storesale__orderdate__lte =
monitor_start_end, returned = False)
            commodities_sold =
sales_set.values_list("commodityinstance")
            ordered_commodities =
Storeordercommodityinstance.objects.filter(storeordercommodity__storeo
rder__store = store_sought, storeorder__receivedate__lte =
monitor_start_end).select_related("commodityinstance__commodity").excl
ude(commodityinstance__in =
commodities_sold).values_list("commodityinstance__commodity").distinct
()
            ordered_commodities_list.append(ordered_commodities)
            monitor_start_date = monitor_start_date - timedelta(days
= 1)
        return render(request,
"nogrosystem/monitor_category_commodities.html",
{"commodities": ordered_commodities_list})

def monitor_store_commodities(request):
    warehouseslug = self.request.GET.get('warehouse')
    warehouse_sought = get_object_or_404(Warehouse, slug =
storeslug)
    commoditytitle = self.request.GET.get('commodity')
    if commoditytitle:
        commodity_sought = get_object_or_404(Commodity, title =
commoditytitle)
        restriction = self.request.GET.get('restriction')
        if not restriction:
            monitor_end = datetime.now()
            monitor_start = monitor_end - relativedelta(months = 12)

```

```

        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        soldstore = []
        soldcustomer = []
        notsold = []
        notsoldtotal = []
        commodities_results = []
        monitor_start_date = monitor_start
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 23, 59)
            if commodityslugin:
                application_store =
Storeordercommodityinstance.objects.filter(storeordercommodity_storeor
der__warehouse = warehouse_sought,
storeordercommodity__commodityvariant__commodity = commodity_sought,
returned = False)
                applicationsoldcount = application_store.count()
                soldstore.append(applicationsoldcount)
                application_sold_store =
application_store.values_list("commodityinstance")
                customercommodities =
Customerordercommoditywarehouse.objects.filter(customerordercommodity_
__commodityvariant__commodity = commodity_sought, warehousestore =
warehouse_sought).values_list("customerordercommodity").distinct()
                application_sold_customer =
Customerordercommodityinstance.objects.filter(customerordercommodity__
in = customercommodities, returned =
False).values_list("commodityinstance")
                applicationsoldcustomercount =
application_sold_customer.count()
                soldcustomer.append(applicationsoldcustomercount)

```

```

        applicationnot =
Applicationcommodityinstance.objects.filter(applicationcommodity__comm
odityvariant__commodity = commodity_sought,
applicationcommodity__application__receivedate__lte =
monitor_start_end, applicationcommodity__application__warehouse =
warehouse_sought).exclude(commodityinstance__in =
application_sold_store)
        applicationnotcount =
applicationnot.values_list("commodityinstance__commodity").distinct().
count()
        notsold.append(applicationnotcount)
        applicationnotcounttotal = applicationnot.count()
        notsoldtotal.append(applicationnotcounttotal)
    else:
        application_sold_store =
Storeordercommodityinstance.objects.filter(storeordercommodity__storeo
rder__warehouse = warehouse_sought, returned =
False).values_list("commodityinstance")
        applicationsoldcount =
application_sold_store.count()
        soldstore.append(applicationsoldcount)
        customercommodities =
Customerordercommoditywarehouse.objects.filter(warehousestore =
warehouse_sought).values_list("customerordercommodity").distinct()
        application_sold_customer =
Customerordercommodityinstance.objects.filter(customerordercommodity__
in = customercommodities, returned =
False).values_list("commodityinstance")
        applicationsoldcustomercount =
application_sold_customer.count()
        soldcustomer.append(applicationsoldcustomercount)
        application_set =
Applicationcommodityinstance.objects.filter(applicationcommodity__appl
ication__receivedate__lte = monitor_start_end,
applicationcommodity__application__warehouse =
warehouse_sought).exclude(commodityinstance__in =
application_sold_store)
        application_set =
application_set.exclude(commodityinstance__in =
application_sold_customer)
        applicationnotcount = application_set.count()
        notsold.append(applicationnotcount)
        application_set =
application_set.values_list("commodityinstance__commodity").distinct()
        applicationnotcounttotal = applicationnot.count()
        notsoldtotal.append(applicationnotcounttotal)
        for commodity in application_set:
            commodity_application_sold_store_count =
Storeordercommodityinstance.objects.filter(storeordercommodity__storeo

```

```

rder__warehouse = warehouse_sought, returned = False,
storeordercommodity__commodityvariant__commodity = commodity).count()
        commodity_customercommodities =
Customerordercommoditywarehouse.objects.filter(warehousestore =
warehouse_sought, customerordercommodity__commodityvariant__commodity
= commodity).values_list("customerordercommodity").distinct()
        commodity_application_sold_customer =
Customerordercommodityinstance.objects.filter(customerordercommodity__
in = commodity_customercommodities, returned =
False).values_list("commodityinstance")
        commodity_applicationsoldcustomercount =
commodity_application_sold_customer.count()
        commodity_application_set =
Applicationcommodityinstance.objects.filter(applicationcommodity__comm
odityvariant__commodity = commodity,
applicationcommodity__application__receivedate__lte =
monitor_start_end, applicationcommodity__application__warehouse =
warehouse_sought).exclude(commodityinstance__in =
application_sold_store)
        commodity_application_set_count =
commodity_application_set.exclude(commodityinstance__in =
commodity_application_sold_customer)
        commodity_application_set_count =
commodity_application_set_count.count()
        commodity_data = {}
        commodity_data["commodity"] = commodity
        commodity_data["store_sold"] =
commodity_application_sold_store_count
        commodity_data["customer_sold"] =
commodity_applicationsoldcustomercount
        commodity_data["notsold"] =
commodity_application_set_count
        commodities_results.append(commodity_data)
        monitor_start_date = monitor_start_date - timedelta(days
= 1)
        return render(request,
"nogrossystem/monitor_store_commodities.html",
        {"store_sold": soldstore,
         "customer_sold": soldcustomer,
         "notsold": notsold,
         "notsolddtotal": notsolddtotal,
         "bycommodities": commodities_results
        })

def analyse_warehouse_commodity(request):
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)

```

```

        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        warehouses = Warehouse.objects.all()
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = monitor_start_start +
datetime.relativedelta(months = 1) - timedelta(days = 1)
            monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
            warehouse_results = []
            for warehouse in warehouses:
                commodities =
Applicationcommodity.objects.filter(application__warehouse =
warehouse, receivedate__gte = monitor_start_start, receivedate__lte =
monitor_start_end).values_list('commodityvariant__commodity')
                commodity_results = []
                for commodity in commodities:
                    application_commodities =
Applicationcommodity.objects.filter(application__warehouse =
warehouse, receivedate__gte = monitor_start_start, receivedate__lte =
monitor_start_end, commodityvariant__commodity = commodity)
                    quantity_result = 0
                    for position in application_commodities:
                        quantity_result = quantity_result +
position.quantity

                data = {}
                data['commodity'] = commodity
                data['quantity'] = quantity_result
                commodity_results.append(data)
            warehouse_data = {}
            data['warehouse'] = warehouse
            data['commodities'] = commodity_results

```

```

        warehouse_results.append(warehouse_data)
        date_data = {}
        date_data['date'] = monitor_start_start
        date_data['warehouses'] = warehouse_results
        results.append(date_data)
        monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime.relativedelta(months = 1)
        return render(request,
            "nogrossystem/analyse_warehouse_commodity.html", {"results": results})

def analyse_channel_sales(request):
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = monitor_start_start +
datetime.relativedelta(months = 1) - timedelta(days = 1)
            monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
            commodities =
Customerordercommodityinstance.objects.filter(returned = False,
customerordercommodity__receivedate__gte = monitor_start_start,
customerordercommodity__receivedate__lte =
monitor_start_end).values_list('commodityinstance__commodityvariant__c
ommodity').distinct()
            commodity_results = []
            for commodity in commodities:

```

```

        quantity_result =
Customerordercommodityinstance.objects.filter(commodityinstance__commo
dityvariant__commodity = commodity, returned = False,
customerordercommodity__receivedate__gte = monitor_start_start,
customerordercommodity__receivedate__lte =
monitor_start_end).values_list('commodityinstance').count()
        data = {}
        data['commodity'] = commodity
        data['quantity'] = quantity_result
    date_data = {}
    date_data['date'] = monitor_start_start
    date_data['customer'] = customer_results
    commodities =
Storesalecommodityinstance.objects.filter(returned = False,
storesalecommodity__storesale__orderdate__gte = monitor_start_start,
storesalecommodity__storesale__orderdate__lte =
monitor_start_end).values_list('commodityinstance__commodityvariant__c
ommodity').distinct()
        commodity_results = []
        for commodity in commodities:
            quantity_result =
Storesalecommodityinstance.objects.filter(commodityinstance__commodity
variant__commodity = commodity, returned = False,
storesalecommodity__storesale__orderdate__gte = monitor_start_start,
storesalecommodity__storesale__orderdate__lte =
monitor_start_end).values_list('commodityinstance').count()
            data = {}
            data['commodity'] = commodity
            data['quantity'] = quantity_result
            commodity_results.append(data)
        date_data['store'] = commodity_results
        commodities =
Storeordercommodityinstance.objects.filter(returned = False,
storeordercommodity__storeorder__orderdate__gte = monitor_start_start,
storeordercommodity__storeorde__orderdate__lte = monitor_start_end,
storeordercommodity__storeorder__store__distributor =
True).values_list('commodityinstance__commodityvariant__commodity').di
stinct()
        commodity_results = []
        for commodity in commodities:
            quantity_result =
Storeordercommodityinstance.objects.filter(returned = False,
storeordercommodity__storeorder__orderdate__gte = monitor_start_start,
storeordercommodity__storeorde__orderdate__lte = monitor_start_end,
storeordercommodity__storeorder__store__distributor = True,
commodityinstance__commodityvariant__commodity =
commodity).values_list('commodityinstance').count()
            data = {}
            data['commodity'] = commodity

```

```

        data['quantity'] = quantity_result
        commodity_results.append(data)
        date_data['distributor'] = commodity_results
        results.append(date_data)
        monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime relativedelta(months = 1)
        return render(request,
"nogrossystem/analyse_channel_sales.html", {"results": results})

def analyse_commodity_orders(request):
    commoditytitle = self.request.GET.get('commodity')
    commodity_soked = get_object_or_404(Commodity, title =
commoditytitle)
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        stores = Store.objects.all()
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = monitor_start_start +
datetime relativedelta(months = 1) - timedelta(days = 1)
            monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
            date_data = {}
            date_data['date'] = monitor_start_start
            store_results = []
            for store is stores:
                total_quantity = 0

```

```

        store_commodities =
Storeordercommodity.objects.filter(storeorder__orderdate__gte =
monitor_start_start, storeorder__orderdate__lte = monitor_start_end,
storeorder__store = store, commodityvariant__commodity =
commodity_sought)
        for store_commodity in store_commodities:
            total_quantity = total_quantity +
commodities.quantity
            data = {}
            data['commodity'] = store
            data['quantity'] = total_quantity
            store_results.append(data)
            date_data['stores'] = store_results
            results.append(date_data)
            monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime.relativedelta(months = 1)
            return render(request,
"nogrossystem/analyse_commodity_orders.html", {"results": results})

def analyse_application_time(request):
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        commodities =
Applicationcommodity.objects.filter(application__receivedate__gte =
monitor_start, application__receivedate__lte =
monitor_end).values_list('commodityvariant')
        for commodity in commodities:
            applications =
Applicationcommodity.objects.filter(application__receivedate__gte =

```

```

monitor_start, application__receivedate__lte = monitor_end,
commodityvariant__commodity = commodity).values_list('application')
    for application in applications:
        application_commodity =
Applicationcommodity.objects.get(application = application,
commodityvariant =
commodity).select_related('application').select_related('commodityvari
ant').select_related('commodityvariant__commodity')
        data = {}
        bought = application_commodity.quantity
        instances =
Applicationcommodityinstance.objects.filter(applicationcommodity =
application_commodity)
        distributor_instances =
Storeordercommodityinstance.objects.filter(returned = False,
commodityinstance__in = instances,
storeordercommodity__storeorder__store__distributor = True)
        total = distributor_instances.count()
        sold_instances =
Storesalecommodityinstance.objects.filter(returned = False,
commodityinstance__in = instances)
        total = total + sold_instances.count()
        customer_instances =
Customerordercommodityinstance.objects.filter(returned = False,
commodityinstance__in = instances)
        total = total + customer_instances.count()
        if bought == total:
            data['commodity'] = application_commodity
            sold_instances =
sold_instances.select_related('storesalecommodity__storesale').order_b
y('-storesalecommodity__storesale__orderdate')
            final_date = None
            last_sold = sold_instances.values()
            if last_sold:
                last_sold = last_sold[0]
                last_sold =
last_sold.storesalecommodity.storesale.orderdate
                final_date = last_sold
                distributor_instances =
distributor_instances.select_related('storeordercommodity__storeorder'
).order_by('- storeordercommodity__storeorder__orderdate')
                last_distributor =
distributor_instances.values()
                if last_distributor:
                    last_distributor = last_distributor[0]
                    last_distributor =
last_distributor.storeordercommodity.storeorder.orderdate
                    if final_date and (last_distributor >
final_date):

```

```

        final_date = last_distributor
    elif not final_date:
        final_date = last_distributor
    customer_instances =
customer_instances.select_related('customerordercommodity__customerord
er').order_by('- customerordercommodity__customerorder__receivedate')
    last_customer = customer_instances.values()
    if last_customer:
        last_customer = last_customer[0]
        last_customer =
last_customer.customerordercommodity.customerorder.receivedate
        if final_date and (last_customer >
final_date):
            final_date = last_customer
        elif not final_date:
            final_date = last_customer
        data['commodity'] = final_date
        results.append(data)
    return render(request,
"nogrossystem/analyse_application_time.html", {"results": results})

def analyse_store_profit(request):
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        stores = Store.objects.all()
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)

```

```

        monitor_start_end = monitor_start_start +
datetime.relative_delta(months = 1) - timedelta(days = 1)
        monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
        date_data = {}
        date_data['date'] = monitor_start_start
        store_results = []
        for store in stores:
            instances =
Storesalecommodityinstance.objects.filter(storesalecommodity__storesal
e__orderdate__gte = monitor_start_start,
storesalecommodity__storesale__orderdate__lte = monitor_start_end,
storesalecommodity__storesale__store = store, returned =
False).prefetch_related('storesalecommodity')
            total_profit = 0
            for store_instance in instances:
                total_profit = total_profit +
store_instance.storesalecommodity.price /
store_instance.storesalecommodity.quantity
                appl =
Applicationcommodityinstance.objects.get(commodityinstance =
store_instance.commodityinstance).select_related('applicationcommodity
')
                total_profit = total_profit - appl.unitcost
            instances =
Customerordercommodityinstance.objects.filter(customerordercommodity__
customerorder__orderdate__gte = monitor_start_start,
customerordercommodity__customerorder__orderdate__lte =
monitor_start_end, customerordercommodity__customerorder__store =
store, returned = False).prefetch_related('customerordercommodity')
            for customer_instance in instances:
                total_profit = total_profit +
customer_instance.customerordercommodity.price /
customer_instance.customerordercommodity.quantity
                appl =
Applicationcommodityinstance.objects.get(commodityinstance =
customer_instance.commodityinstance).select_related('applicationcommod
ity')
                total_profit = total_profit - appl.unitcost
            data = {}
            data['store'] = store
            data['profit'] = total_profit
            store_results.append(data)
        date_data['stores'] = store_results
        results.append(date_data)
        monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime.relative_delta(months = 1)
        return render(request,
"nogrossystem/analyse_store_profit.html", {"results": results})

```

```

def analyse_commodity_profit(request):
    commoditytitle = self.request.GET.get('commodity')
    commodity_sought = get_object_or_404(Commodity, title =
commoditytitle)
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')
        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        commodities = Commodity.objects.all()
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = monitor_start_start +
datetime.relativedelta(months = 1) - timedelta(days = 1)
            monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
            date_data = {}
            date_data['date'] = monitor_start_start
            commodity_results = []
            for commodity in commodities:
                instances =
Storesalecommodityinstance.objects.filter(commodityinstance__commodity
variant__commodity = commodity,
storesalecommodity__storesale__orderdate__gte = monitor_start_start,
storesalecommodity__storesale__orderdate__lte = monitor_start_end,
storesalecommodity__storesale__store = store, returned =
False).prefetch_related('storesalecommodity')
                total_profit = 0
                for store_instance in instances:

```

```

        total_profit = total_profit +
store_instance.storesalecommodity.price /
store_instance.storesalecommodity.quantity
        appl =
Applicationcommodityinstance.objects.get(commodityinstance =
store_instance.commodityinstance).select_related('applicationcommodity
')
        total_profit = total_profit - appl.unitcost
        instances =
Customerordercommodityinstance.objects.filter(commodityinstance__commo
dityvariant__commodity = commodity,
customerordercommodity__customerorder__orderdate__gte =
monitor_start_start,
customerordercommodity__customerorder__orderdate__lte =
monitor_start_end, customerordercommodity__customerorder__store =
store, returned = False).prefetch_related('customerordercommodity')
        for customer_instance in instances:
            total_profit = total_profit +
customer_instance.customerordercommodity.price /
customer_instance.customerordercommodity.quantity
            appl =
Applicationcommodityinstance.objects.get(commodityinstance =
customer_instance.commodityinstance).select_related('applicationcommod
ity')
            total_profit = total_profit - appl.unitcost
            data = {}
            data['commodity'] = commodity
            data['profit'] = total_profit
            commodity_results.append(data)
            date_data['commodities'] = commodity_results
            results.append(date_data)
            monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime.relativedelta(months = 1)
        return render(request,
"nogrossystem/analyse_commodity_profit.html", {"results": results})

def analyse_commodity_price(request):
    commoditytitle = self.request.GET.get('commodity')
    commodity_soked = get_object_or_404(Commodity, title =
commoditytitle)
    restriction = self.request.GET.get('restriction')
    if not restriction:
        monitor_end = datetime.now()
        monitor_start = monitor_end - relativedelta(months = 12)
        monitor_start = datetime(monitor_start.year,
monitor_start.month, monitor_start.day, 0, 0)
    else:
        monitor_start_month = self.request.GET.get('startmonth')
        monitor_start_year = self.request.GET.get('startyear')

```

```

        monitor_start_year = int(monitor_start_year)
        monitor_start_month = int(monitor_start_month)
        monitor_start = datetime(monitor_start_year,
monitor_start_month, 1, 0, 0)
        monitor_end_month = self.request.GET.get('endmonth')
        monitor_end_year = self.request.GET.get('endyear')
        monitor_end_month = int(monitor_end_month)
        monitor_end_year = int(monitor_end_year)
        monitor_end = date(monitor_end_year, monitor_end_month +
1, 1) - timedelta(days = 1)
        monitor_end = datetime(monitor_end.year,
monitor_end.month, monitor_end.day, 23, 59)
        results = []
        commodities = Commodity.objects.all()
        while monitor_start_date <= monitor_end:
            monitor_start_start = datetime(monitor_start_date.year,
monitor_start_date.month, monitor_start_date.day, 0, 0)
            monitor_start_end = monitor_start_start +
datetime.relativedelta(months = 1) - timedelta(days = 1)
            monitor_start_end = datetime(monitor_start_end.year,
monitor_start_end.month, monitor_start_end.day, 23, 59)
            date_data = {}
            date_data['date'] = monitor_start_start
            commodity_results = []
            for commodity in commodities:
                instances =
Storesalecommodityinstance.objects.filter(commodityinstance__commodity
variant__commodity = commodity,
storesalecommodity__storesale__orderdate__gte = monitor_start_start,
storesalecommodity__storesale__orderdate__lte = monitor_start_end,
storesalecommodity__storesale__store = store, returned =
False).prefetch_related('storesalecommodity')
                average_price = 0
                cases = 0
                for store_instance in instances:
                    average_price = average_price +
store_instance.storesalecommodity.price
                    cases = cases + 1
                instances =
Customerordercommodityinstance.objects.filter(commodityinstance__commo
dityvariant__commodity = commodity,
customerordercommodity__customerorder__orderdate__gte =
monitor_start_start,
customerordercommodity__customerorder__orderdate__lte =
monitor_start_end, customerordercommodity__customerorder__store =
store, returned = False).prefetch_related('customerordercommodity')
                for customer_instance in instances:
                    average_price = average_price +
customer_instance.customerordercommodity.price

```

```
        cases = cases + 1
    data = {}
    data['commodity'] = commodity
    data['price'] = average_price / cases
    commodity_results.append(data)
    date_data['commodities'] = commodity_results
    results.append(date_data)
    monitor_start_date = monitor_start_date -
datetime.datetime.now() - datetime relativedelta(months = 1)
    return render(request,
        "nogrossystem/analyse_commodity_price.html", {"results": results})
```

**ДОДАТОК В**  
**Слайди презентації**

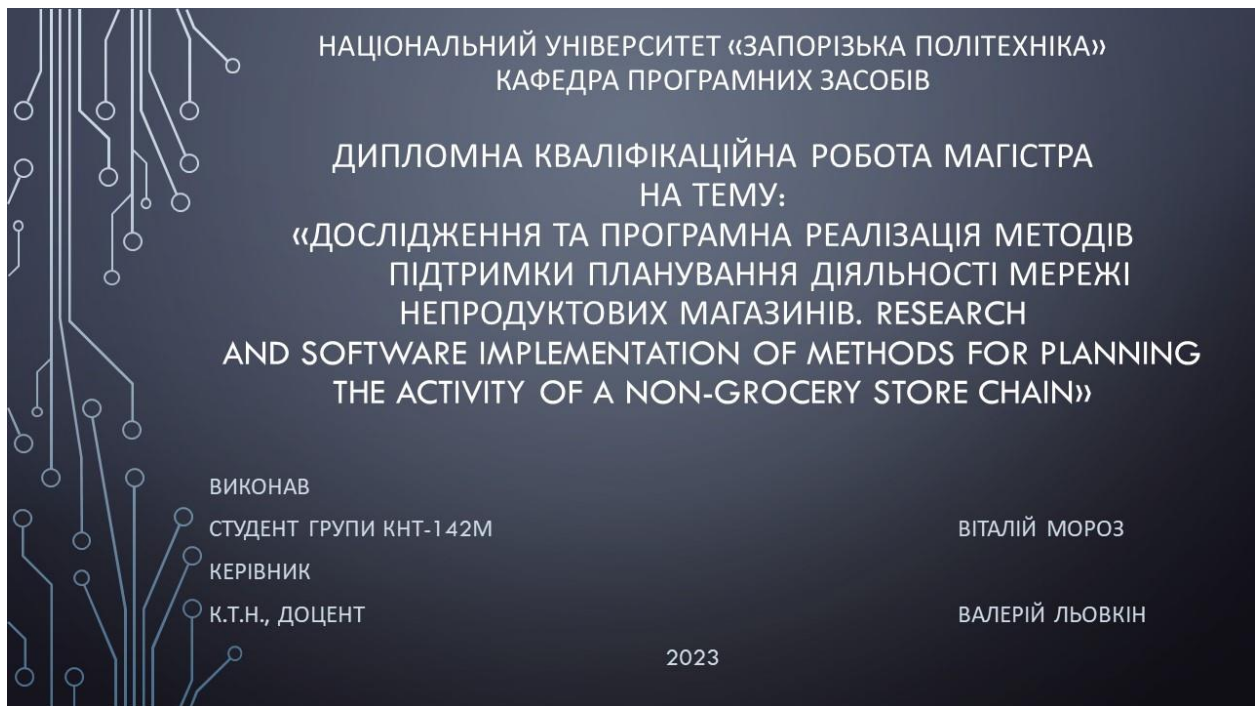


Рисунок В.1 – Слайд № 1 (Титульний слайд)

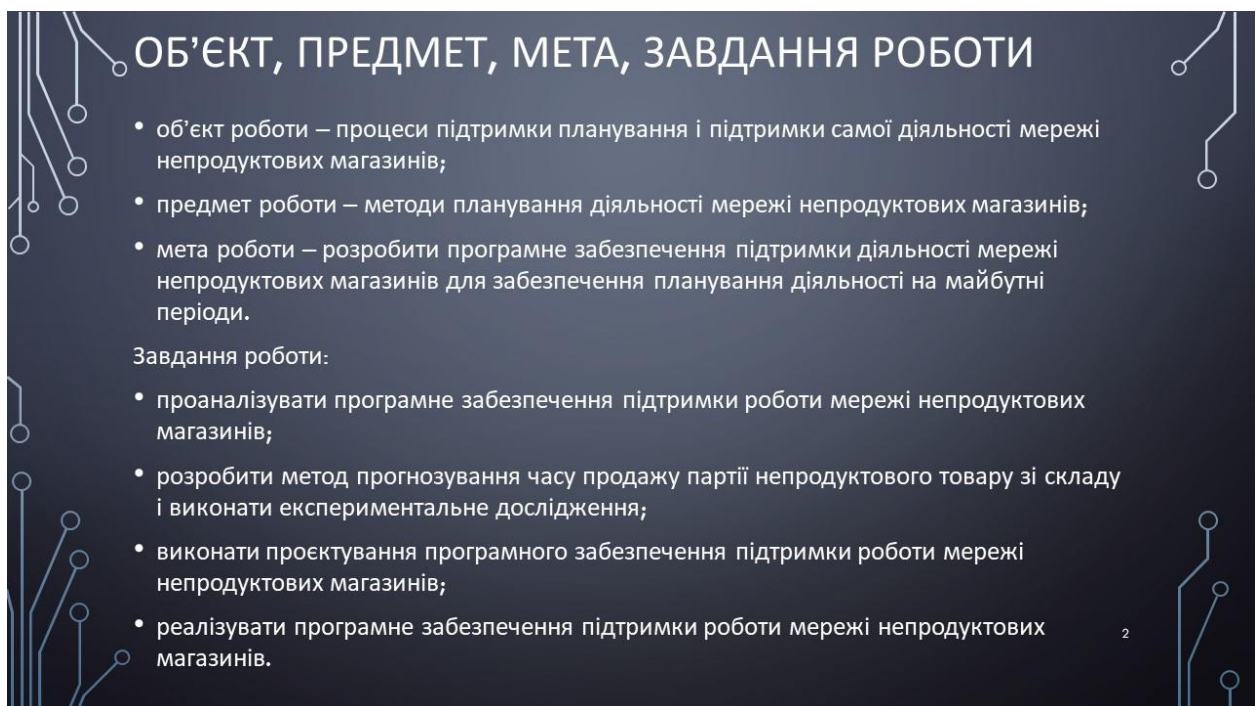


Рисунок В.2 – Слайд № 2 (Об'єкт, предмет, мета, завдання роботи)

### ПОРІВНЯННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЯКЕ МОЖЕ ВИКОРИСТОВУВАТИСЯ ДЛЯ ПІДТРИМКИ РОБОТИ МЕРЕЖІ НЕПРОДУКТОВИХ МАГАЗИНІВ

| Фактор порівняння      | NetSuite                                                                                         | Prime Penguin                                           | Програма в розробці                                                    |
|------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------|------------------------------------------------------------------------|
| Основні функції        | Керування взаємодією з клієнтами, замовленнями, складом                                          | Керування взаємодією з клієнтами, замовленнями, складом | Керування взаємодією з клієнтами, замовленнями, складом                |
| Засоби аналітики       | Так                                                                                              | Так                                                     | Так                                                                    |
| Налаштування роботи    | Потребує окремого встановлення і налаштування для різних ролей, що потребує додаткового навчання | Менш складний процес ніж у NetSuite                     | Потребує встановлення на сервері, додаткового налаштування не потребує |
| Тип доступу            | Платний                                                                                          | Платний                                                 | Безкоштовний                                                           |
| Показник прогнозування | Попит                                                                                            | Попит, час продажу максимум на 10 тижнів                | Час продажу партії за складом                                          |
| Моделі прогнозування   | Рухоме середнє, лінійна регресія, сезонне середнє                                                | Рухоме, зважене середнє                                 | Random Forest                                                          |

Рисунок В.3 – Слайд № 3 (Порівняння програмного забезпечення, яке може використовуватися для підтримки роботи мережі непродуктових магазинів)

### МЕТОД ПРОГНОЗУВАННЯ ЧАСУ ПРОДАЖУ ПАРТІЇ НЕПРОДУКТОВОГО ТОВАРУ ЗІ СКЛАДУ

- визначення за кожною партією непродуктових товарів значень характеристик цієї партії, включаючи кількість одиниць товару, категорію товару, середню вартість одиниці товару для споживача, середній дисконт на одиницю товару, кількість днів збуту партії, дату, коли було отримано цю партію товару на склад, за формулою

$$d_i = month_i - month^{min} + 12(year_i - year^{min}),$$

- формування вибірки з нормуванням значень, виділенням цільової ознаки;
- створення і навчання моделі прогнозування часу продажу партії непродуктового товару зі складу за методом Random Forest.

Рисунок В.4 – Слайд № 4 (Метод прогнозування часу продажу партії непродуктового товару зі складу)

# ДІАГРАМИ ПРЕЦЕДЕНТІВ

The diagrams illustrate the following use cases and actor interactions:

- Неавторизований клієнт** (Unauthenticated User):
  - Відображення даних непродутового товару (View product data)
  - авторизація користувача (User authentication) - *Extends* реєстрація користувачів (User registration)
  - пошук непродутових товарів за вказаними параметрами (Search products by parameters)
  - перегляд непродутових товарів, що продаються у мережі, за підкатегорією (View products for sale by category)
  - перегляд вилучених підкатегорій непродутових товарів (View excluded product categories)
- Клієнт** (Client):
  - замовлення непродутових товарів (Order products) - *Extends* скасування невиконаного замовлення непродутових товарів (Cancellation of unexecuted product order)
  - відображення замовлень клієнта в розрізі станів (View client orders by status)
  - оновлення персональних даних (Update personal data)
  - відображення списку непродутових товарів, які замовляв клієнт (View list of products ordered by client)
- Оператор магазину** (Store Operator):
  - реєстрація прибуття замовлення зі складу до магазину (Register order arrival from warehouse to store)
  - реєстрація продажу замовлення клієнту (Register order sale to client)
  - реєстрація повернення товару з замовлення (Register return of goods from order)
  - реєстрація стану гарантійного обслуговування екземпляру товару (Register warranty service status of a product sample)
  - відображення замовлень у магазині в розрізі стану (View orders in the store by status)
  - оформлення продажу непродутових товарів у магазині (Formalize sale of products in the store)
  - відображення історії зміни станів екземплярів товару (View history of status changes of product samples)
  - відображення повернень екземплярів непродутових товарів клієнтом (View returns of product samples by client)
  - відображення екземплярів непродутових товарів у гарантійному обслуговуванні (View product samples in warranty service)
- Менеджер магазину** (Store Manager):
  - відображення списку працівників магазину (View list of store employees)
  - деталізація замовлення непродутових товарів зі складу мережі (Detailing of product order from network warehouse) - *Extends* відслідковування створеного замовлення товарів зі складу мережі (Tracking of created network warehouse goods order)
  - формування замовлення непродутових товарів зі складу в магазині (Forming product order from store warehouse)
  - відображення замовлень непродутових товарів зі складу (View product orders from warehouse)
  - пошук працівника в компанії (Search employee in company)
  - підтвердження створеного клієнтом замовлення для доставки зі складу (Confirmation of client order for delivery from warehouse) - *Extends* скасування створеного клієнтом замовлення для доставки зі складу (Cancellation of client order for delivery from warehouse)

[illegible]

Рисунок В.8 – Слайд № 8 (Схема БД)



Рисунок В.9 – Слайд № 9 (Діаграма станів екземпляру непродуктового товару)

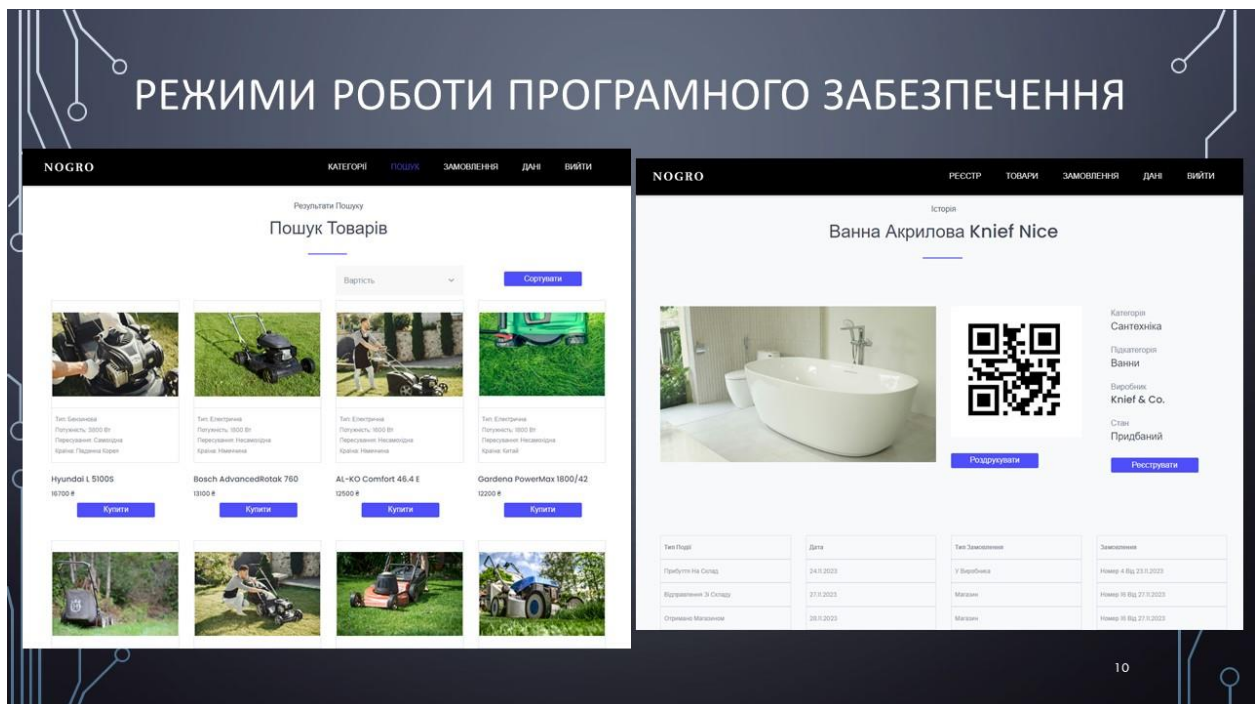


Рисунок В.10 – Слайд № 10 (Режими роботи програмного забезпечення)

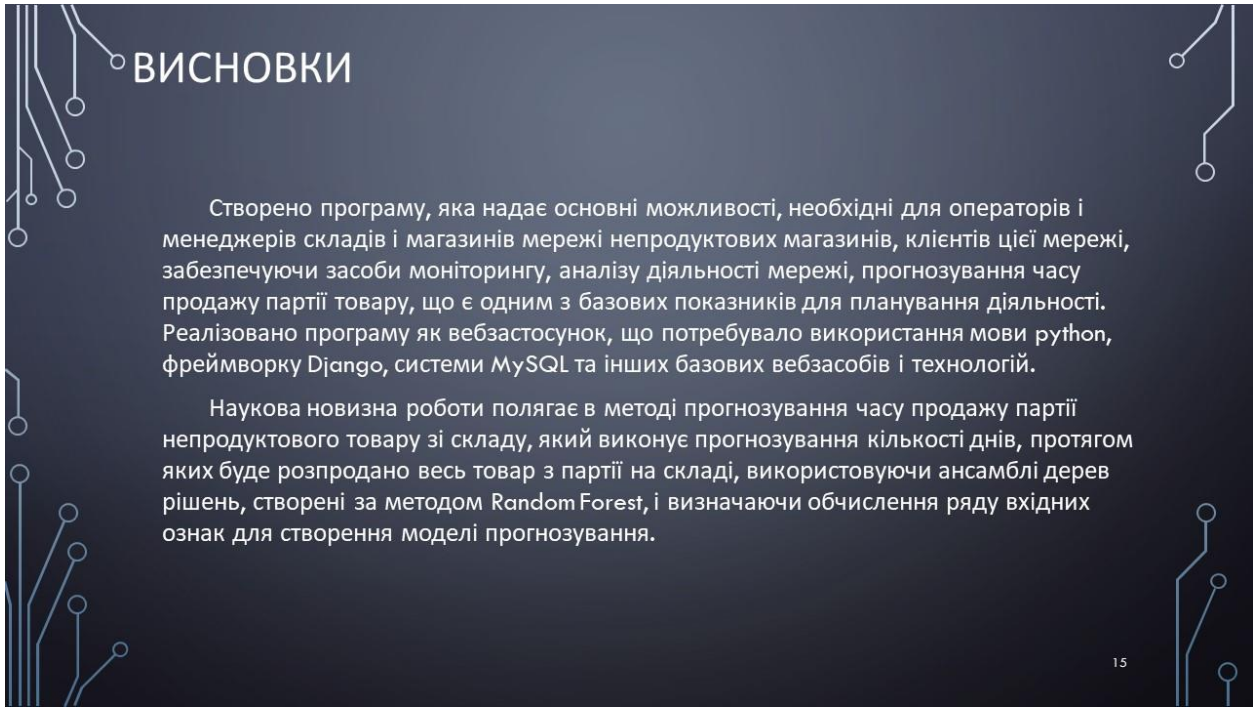
# РЕЖИМИ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

| ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ                                 |            |                      |                            |                                                                         |
|--------------------------------------------------------------|------------|----------------------|----------------------------|-------------------------------------------------------------------------|
| Вхідні ознаки                                                | CART       | Запропонований метод | Максимальна глибина дерева | Метод прогнозування часу продажу партії непродуктового товару зі складу |
| Початок, кількість, товар, склад, витрати, вартість, дисконт | 0,03559024 | 0,03383645           | 2                          | 0,03462187                                                              |
|                                                              |            |                      | 3                          | 0,03361279                                                              |
|                                                              |            |                      | 4                          | 0,03294665                                                              |
|                                                              |            |                      | 5                          | 0,03256456                                                              |
| Початок, кількість, товар, витрати, вартість, дисконт        | 0,03500126 | 0,03260553           | 6                          | 0,03232888                                                              |
|                                                              |            |                      | 7                          | 0,03221691                                                              |
| Початок, кількість                                           | 0,03405903 | 0,03989508           | 8                          | 0,03220835                                                              |
|                                                              |            |                      | 9                          | 0,03227352                                                              |
| Початок, кількість, товар                                    | 0,03457473 | 0,03309797           | 10                         | 0,03236691                                                              |
| Початок, кількість, товар, витрати, вартість                 | 0,03500126 | 0,03295672           | Кількість дерев у ансамблі | Метод прогнозування часу продажу партії непродуктового товару зі складу |
| Початок, кількість, товар, витрати                           | 0,03457473 | 0,03321438           | 500                        | 0,03228161                                                              |
| Початок, кількість, товар, вартість                          | 0,03500126 | 0,03243525           | 1000                       | 0,03225975                                                              |
|                                                              |            |                      | 2000                       | 0,03221671                                                              |
| Початок, кількість, товар, витрати, дисконт                  | 0,03457473 | 0,03269814           | 3000                       | 0,03216069                                                              |
|                                                              |            |                      | 4000                       | 0,03220782                                                              |
| Початок, кількість, товар, вартість, дисконт                 | 0,03500126 | 0,03221691           | 5000                       | 0,03219613                                                              |
|                                                              |            |                      | 6000                       | 0,03221086                                                              |
| Початок, кількість, товар, дисконт                           | 0,03457473 | 0,03250969           | 7000                       | 0,03220835                                                              |
|                                                              |            |                      | 8000                       | 0,03222869                                                              |
|                                                              |            |                      | 9000                       | 0,03221922                                                              |
|                                                              |            |                      | 10000                      | 0,03222403                                                              |

Рисунок В.13 – Слайд № 13 (Експериментальне дослідження)

| ПІДСУМКОВІ РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ                          |            |
|-------------------------------------------------------------------------------|------------|
| Характеристики моделі                                                         | Результат  |
| CART (2 вхідні ознаки)                                                        | 0,03405903 |
| CART (3 вхідні ознаки)                                                        | 0,03457473 |
| CART (всі доступні вхідні ознаки)                                             | 0,03559024 |
| CART (вхідні ознаки аналогічно методу)                                        | 0,03500126 |
| Random Forest (3 вхідні ознаки, 3000 дерев, 6 максимальна глибина)            | 0,03251367 |
| Random Forest (всі доступні вхідні ознаки, 7000 дерев, 6 максимальна глибина) | 0,03377447 |
| Метод прогнозування часу продажу партії непродуктового товару зі складу       | 0,03216069 |

Рисунок В.14 – Слайд № 14 (Підсумкові результати експериментального дослідження)



## ВИСНОВКИ

Створено програму, яка надає основні можливості, необхідні для операторів і менеджерів складів і магазинів мережі непродуктових магазинів, клієнтів цієї мережі, забезпечуючи засоби моніторингу, аналізу діяльності мережі, прогнозування часу продажу партії товару, що є одним з базових показників для планування діяльності. Реалізовано програму як вебзастосунок, що потребувало використання мови `python`, фреймворку `Django`, системи `MySQL` та інших базових вебзасобів і технологій.

Наукова новизна роботи полягає в методі прогнозування часу продажу партії непродуктового товару зі складу, який виконує прогнозування кількості днів, протягом яких буде розпродано весь товар з партії на складі, використовуючи ансамблі дерев рішень, створені за методом `Random Forest`, і визначаючи обчислення ряду вхідних ознак для створення моделі прогнозування.

15

Рисунок В.15 – Слайд № 15 (Висновки)