МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З. ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **“ Інвентаризація ІТ-інфраструктури підприємства реалізованої в хмарному сховищі ”**

Виконав: ст. гр. Іт-62_____

Спеціальності 126 – «Інформаційні системи та технології»_____
(шифр і назва)

_____Багрич Володимир Миколайович_____
(прізвище та ініціали)

Керівник: __к.т.н., доц. Луб П.М._____
(прізвище та ініціали)

Рецензент: _____
(прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З. ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

другий (магістерський) рівень вищої освіти
126 – «Інформаційні системи та технології»

"ЗАТВЕРДЖУЮ"
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
"____"_____2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Багрич Володимир Миколайович

1. Тема роботи: «Інвентаризація ІТ-інфраструктури підприємства реалізованої в хмарному сховищі»

Керівник роботи Луб Павло Миронович, к.т.н., доцент
Затверджені наказом по університету №140/к-с від 28.02.2025.

2. Строк подання студентом роботи 01.12.2025 р.
3. Початкові дані до роботи: 1. Архітектура та принципи функціонування хмарних сервісів. 2. Технічні вимоги до розроблення клієнтської (Front-end) та серверної (Back-end) частин системи. 3. Основні вимоги до проєктування клієнт-серверної архітектури. 4. Хмарний сервіс зберігання та синхронізації даних "File Linker".
4. Зміст розрахунково-пояснювальної записки:

Головні поняття та аналіз потреб інвентаризації даних підприємства
Системи управління інвентаризацією даних
Методика проектування інформаційної системи.
Реалізація інформаційної системи інвентаризації даних
Висновки та пропозиції.
Бібліографічний список.
Додатки.

5. Перелік графічного матеріалу 1 та 2 – Тема, мета, завдання роботи; 3 – Аналіз головних понять з інвентаризації ІТ-активів; 4 – Аналіз хмарних сховищ та їх типів; 5 – Аналіз популярних систем та інвентаризації даних; 6 – Структуризація цілей; 7 – Моделювання потоків даних; 8 – Вибір інструментів для реалізації ІС; 9 – Структура бази даних; 10 – Особливості програмної реалізації; 11 – Результати застосування ІС; 12 – Результати дослідження продуктивності ІС; 14 – Головні висновки.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 4	*Луб П.М., доцент кафедри інформаційних технологій*		
5	*Городецький І.М., доцент кафедри інженерної механіки*		

7. Дата видачі завдання 28 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів роботи	Приміт-ка
1.	*Написання першого розділу та означення головних завдань роботи*	*01.03-01.05.25*	
2.	*Виконання другого розділу та опис інформаційних технологій для виконання завдань роботи*	*01.03-01.05.25*	
3.	*Виконання третього розділу, методика вирішення завдань та елементи наукових досліджень*	*01.05-01.07.25*	
4.	*Виконання четвертого розділу, представлення результатів та їх узагальнення*	*01.05-01.07.25*	
5.	*Написання розділу: «Охорона праці та безпека в надзвичайних ситуаціях»*	*01.07-01.09.25*	
6.	*Завершення оформлення розрахунково-пояснювальної записки та презентаційних матеріалів*	*01.09-01.11.25*	
7.	*Завершення роботи в цілому*	*01.11-01.12.25*	

Студент _____ Багрич В.М.
(підпис)

Керівник роботи _____ Луб П.М.
(підпис)

УДК: 004.738.5:004.6:004.451.3

Кваліфікаційна робота: 75 с. текст. част., 34 рис., 1 табл., 13 слайдів, 21 джерело.

Інвентаризація ІТ-інфраструктури підприємства реалізованої в хмарному сховищі. Багрич В.М. Кафедра ІТ. – Дубляни, Львівський НУВМБ, 2025.

Розглянуто основні теоретичні та практичні аспекти створення інформаційної системи управління інвентаризацією даних підприємства. Проаналізовано сутність процесу інвентаризації, визначено його роль у забезпеченні достовірності та системності обліку ІТ-активів, а також сформульовано вимоги до сучасних засобів управління інвентаризаційними процесами. Проведено порівняльний аналіз найпоширеніших програмних систем для інвентаризації даних з метою виявлення їхніх переваг та недоліків.

Розроблено структуру системи управління інвентаризацією, визначенню її цілей і задач, побудові моделей потоків даних та діаграм декомпозиції, що дозволяють описати логіку взаємодії компонентів системи. Запропоновано методику проектування інформаційної системи, зокрема вибір інструментів реалізації, визначення структури бази даних і зв’язків між її елементами, а також створення узгодженої архітектури ERP-рівня.

Описано програмну частину розробленого застосунку, наведено основні функціональні можливості, інтерфейс користувача та приклади використання. Виконано оцінку стабільності роботи системи, її продуктивності та ефективності при зростанні обсягів даних і кількості запитів. Наведено результати дослідження, які підтверджують доцільність впровадження запропонованої інформаційної системи як інструмента інвентаризаційних даних.

Запропоновано заходи з охорони праці за безпеки в надзвичайних ситуаціях.

Ключові слова: інвентаризація, дані, інформаційна система, база даних, ERP-система, обліку ресурсів, стабільність роботи, аналіз ефективності.

ЗМІСТ

ВСТУП

Стрімкий розвиток інтернет-сервісів спричинив різке зростання товарообігу, що, у свою чергу, ускладнило процеси контролю, відстеження та управління запасами продукції. Для більшості сучасних підприємств використання інформаційних технологій стало фінансово доступним і доцільним інструментом ведення бізнес-процесів. Значення інформаційних систем істотно зросло, що призвело до появи нових завдань, пов’язаних із впорядкуванням, обробкою та обліком даних [5, 7].

Сучасні інформаційні системи інвентаризації дедалі частіше використовують хмарні технології, штучний інтелект і машинне навчання для підвищення точності аналітики. Інтеграція IoT-пристроїв (сканерів, RFID-міток, сенсорів) дозволяє автоматично збирати дані про переміщення та стан товарів у реальному часі. Такі рішення забезпечують не лише актуальність баз даних, але й можливість прогнозувати нестачу ресурсів, оптимізувати логістику та попереджати ризики людських помилок. У перспективі це сприятиме переходу бізнесу до повністю цифрових моделей управління запасами, де ключову роль відіграватиме аналітика великих даних і адаптивні системи підтримки прийняття рішень.

Водночас останні роки характеризуються стрімким зростанням індустрії мобільних додатків. Нині понад 2,3 мільйона розробників по всьому світу створюють мобільні рішення для задоволення потреб різних сфер [3].

Такі тенденції свідчать, що створення спеціалізованих інформаційних систем для цифрової інвентаризації є важливою складовою успішного бізнесу. Раціональним рішенням є інтеграція маркетингових процесів із системами обліку та контролю матеріальних ресурсів. Завдяки цьому користувачі програмного забезпечення можуть проводити інвентаризацію з мобільних пристроїв, оперувати цифровими документами обліку та швидко ідентифікувати товари за допомогою QR-кодів або тегів.

Відповідно до цих положень сформовано мету кваліфікаційної роботи.

Мета роботи – створення інформаційної системи, що підвищує рівень цифровізації управлінських процесів підприємства, забезпечує прозорість інвентаризаційних даних і сприяє оптимізації ресурсів.

Завдання роботи:

- проаналізувати предметну область, визначити основні об’єкти інвентаризації, їх характеристики та взаємозв’язки з метою формування структури бази даних для майбутньої інформаційної системи;

- розробити архітектуру веб-додатку, що включає фронт-енд, бек-енд та базу даних, із використанням сучасних технологій (Angular, TypeScript, ASP.NET Core, Entity Framework Core, Microsoft SQL Server) в хмарі, для забезпечення стабільної та масштабованої роботи системи;

- реалізувати додаток в хмарі із постійним доступом, функціонал авторизації, управління користувачами, додавання, редагування та пошуку товарно-матеріальних цінностей, а також формування звітів про стан запасів у системі;

- встановити закономірність показників продуктивності роботи інформаційної системи.

Об’єктом розробки є веб-додаток інформаційної системи інвентаризації даних, який забезпечує облік, зберігання, пошук та оновлення інформації про товарно-матеріальні цінності, а також взаємодію користувачів із системою через клієнтський інтерфейс.

Предметом розробки є засоби та технології, методи та інструменти, що використовуються для створення веб-додатку, зокрема архітектура front-end та back-end, база даних Microsoft SQL Server, стек Angular і TypeScript для клієнтської частини, ASP.NET Core та Entity Framework Core для серверної логіки, а також алгоритми обробки, валідації і захисту даних у процесі функціонування системи.

Новизна запропонованого підходу полягає у:

- комплексній інтеграції сучасних веб-технологій, які забезпечують високу узгодженість між клієнтською та серверною частинами застосунку;

- поєднання Angular, TypeScript і ASP.NET Core з використанням Entity Framework Core створює гібридне середовище, де застосовуються як концепції об’єктно-орієнтованого програмування, так і сучасні принципи реактивної архітектури;

- побудова бази даних, орієнтованої на багаторівневу структуру взаємопов’язаних об’єктів (користувачі, локації, товари, теги), що забезпечує гнучкість моделювання та можливість розширення без зміни загальної архітектури.

Практична цінність розробленої системи полягає у створенні універсального рішення для обліку та інвентаризації товарно-матеріальних цінностей, яке може бути адаптоване до потреб різних організацій чи підприємств. Завдяки веб-архітектурі застосунок забезпечує доступність з будь-якого пристрою та операційної системи, а використання SQL Server гарантує надійність зберігання даних. Крім того, впроваджена структура бази даних дозволяє швидко інтегрувати додаткові функціональні модулі, наприклад аналітику або модулі автоматичного формування звітності. Таким чином, система має як технологічну, так і прикладну цінність, поєднуючи сучасні IT-підходи з реальними потребами управління даними.

РОЗДІЛ 1
ГОЛОВНІ ПОНЯТТЯ ТА АНАЛІЗ ПОТРЕБ ІНВЕНТАРИЗАЦІЇ ДАНИХ ПІДПРИЄМСТВА

1.1. Інвентаризація даних та її призначення

Інвентаризація даних (*data inventory)* – це систематичний процес виявлення, каталогізації, класифікації та документування всіх або значущих даних (активів даних) організації: де вони зберігаються, хто ними володіє, як вони використовуються, їхній стан, формат, рівень чутливості тощо [21].

У сучасному бізнес-середовищі, керованому даними, організації накопичують величезні обсяги даних з різних джерел. Однак, без чіткого розуміння того, якими даними вони володіють і як вони структуровані, підприємствам може бути важко розкрити їхній справжній потенціал. Саме тут виникає потреба в інвентаризації даних.

Інвентаризація даних – це практика збору детальної інформації про активи даних, якими володіє організація, що дозволяє отримати структуроване уявлення про ландшафт її даних. Створюючи комплексну інвентаризацію, підприємства можуть краще використовувати свої дані для прийняття рішень, підвищення операційної ефективності та забезпечення відповідності вимогам.

Призначення інвентаризації даних:

1) Забезпечити видимість того, які дані має організація, де вони знаходяться та як ними керують.

2) Підтримувати управління даними і політиками (*data governance*) — сприяти дотриманню нормативів, безпеці даних, прозорості.

3) Підвищити ефективність роботи з даними: зменшити дублювання, виявити застарілі чи нерелевантні дані, оптимізувати зберігання і використання.

4) Сприяти прийняттю обґрунтованих рішень: коли відомо, які дані є, їх якість і доступність — легше будувати аналітику.

5) Управляти ризиками: включно з ризиками втрати, витоку, некоректного використання даних.

Ключовими причинами, що лежать в основі формування пріоритету інвентаризації даних у підприємствах є [3, 4]:

Покращене прийняття рішень: чітке розуміння доступних активів даних дозволяє підприємствам приймати рішення на основі даних. Легко отримуючи доступ до відповідних даних, організації можуть виявляти тенденції, закономірності та аналітичні дані, які можуть стимулювати інновації та зростання.

Підвищена ефективність: інвентаризація даних допомагає організаціям оптимізувати свої процеси управління даними. Вона надає інформацію про надлишковість даних, прогалини та невідповідності, що дозволяє краще інтегрувати дані, стандартизувати та керувати якістю даних.

Ефективне управління ризиками: Шляхом категоризації та документування активів даних відповідно до їхньої чутливості та критичності, Інвентаризація даних допомагає організаціям виявляти потенційні ризики для безпеки даних. Це дозволяє підприємствам впроваджувати відповідні заходи безпеки та гарантії для захисту своїх цінних даних.

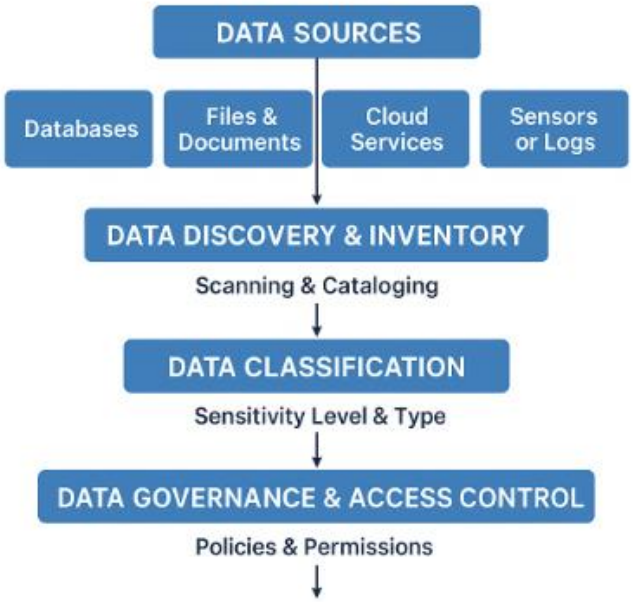

Рисунок 1.1 – Етапи роботи з даними (від збору до класифікації та контролю доступу) [21]

Інвентаризація даних може бути класифікована за різними критеріями:

- *за типом даних:* 1) особисті/персональні дані (PII) — ім’я, контакт, фінансова інформація і т.д.; 2) операційні дані — виробничі, логістичні, облікові процеси; 3) фінансові дані; 4) інші категорії: геолокаційні, веб-треки, соціальні мережі тощо.

- *за метою/контекстом:* 1) інвентаризація для відповідності нормативам (compliance) — зосереджена на тому, щоб показати, які дані організація має, як вони керуються; 2) інвентаризація як частина управління даними (data governance) – більш стратегічно, не лише для відповідності, а для оптимізації використання даних.

- *за форматами або джерелами:* 1) структуровані дані (бази даних); 2) напівструктуровані/неструктуровані дані (документи, файли, лог-файли, хмарні сховища); 3) дані сторонніх джерел або хмарних сервісів.

1.2. Вимоги до управління інвентаризацією IT-активів

Ефективне керування інвентаризацією IT-інфраструктури є ключовим чинником стабільної роботи будь-якої організації. Воно сприяє підтримці високої продуктивності, безпеки та загальної ефективності мережі. Водночас, без належного інструменту для виконання цієї задачі процес інвентаризації може стати складним і трудомістким. Тому вибір оптимального програмного забезпечення для моніторингу й управління активами має вирішальне значення.

Під час вибору системи для «мережевої» інвентаризації підприємства варто звернути увагу на кілька ключових характеристик. До них належать можливість автоматичного виявлення активів, моніторинг стану мережі та оповіщення в режимі реального часу, формування детальної звітності та аналітики, а також інтеграція з іншими інструментами IT-управління. Важливими залишаються також масштабованість рішення та співвідношення його вартості до функціональних можливостей [3].

Автоматизоване виявлення пристроїв у мережі дозволяє зменшити обсяг ручної роботи та мінімізувати ризик пропуску активів, які можуть залишатися поза контролем. Такий підхід забезпечує повну прозорість структури мережі й дає змогу швидко реагувати на будь-які зміни в інфраструктурі.

На сучасному етапі розвитку IT-технологій існує велика кількість програмних рішень, призначених для інвентаризації та моніторингу мережевої інфраструктури. Серед них варто окремо відзначити систему PRTG від компанії Paessler, яка на практиці довела свою ефективність, зручність і надійність у сфері управління IT-активами. Її головною перевагою є висока точність збору даних, гнучкість конфігурації та інтуїтивність у користуванні [7].

Система **Paessler PRTG** є багатофункціональним інструментом для комплексного моніторингу IT-інфраструктури, розробленим німецькою компанією Paessler AG. Її основне завдання полягає у відстеженні стану мереж, серверів, додатків, сервісів і різноманітних пристроїв у режимі реального часу. Головна мета використання цього рішення — забезпечити стабільну та безперебійну роботу корпоративних систем завдяки своєчасному виявленню несправностей і попередженню можливих збоїв у їх роботі.

Архітектура PRTG побудована на використанні так званих сенсорів (sensors) — спеціалізованих модулів, кожен із яких відповідає за контроль певного параметра, наприклад рівня завантаження процесора, швидкості обміну даними, доступності веб-сервісів або стану баз даних. Такий підхід забезпечує детальне спостереження за всіма компонентами інфраструктури, даючи змогу швидко локалізувати й усунути проблему.

PRTG підтримує широкий спектр протоколів, серед яких **SNMP, WMI, SSH, Ping, HTTP, NetFlow, sFlow, jFlow** та інші, що робить її сумісною майже з усіма типами обладнання та операційних систем. Зручний веб-інтерфейс системи дозволяє створювати інтерактивні карти мережі, аналітичні панелі й дашборди з актуальними даними, які оновлюються в режимі реального часу.

Користувачі мають можливість налаштовувати систему сповіщень — про перевищення допустимих навантажень, відмови серверів чи проблеми з

доступом до ресурсів. Повідомлення можуть надходити через електронну пошту, push-сповіщення або інші комунікаційні канали, що забезпечує оперативне реагування на інциденти.

Завдяки своїй масштабованості, простоті впровадження та надійності, PRTG отримала широке розповсюдження в IT-компаніях, державних структурах, освітніх установах і бізнес-секторі. Вона суттєво підвищує ефективність управління IT-ресурсами, сприяє оптимізації роботи мережевих систем і забезпечує стабільність цифрових сервісів навіть у складних інфраструктурних умовах.

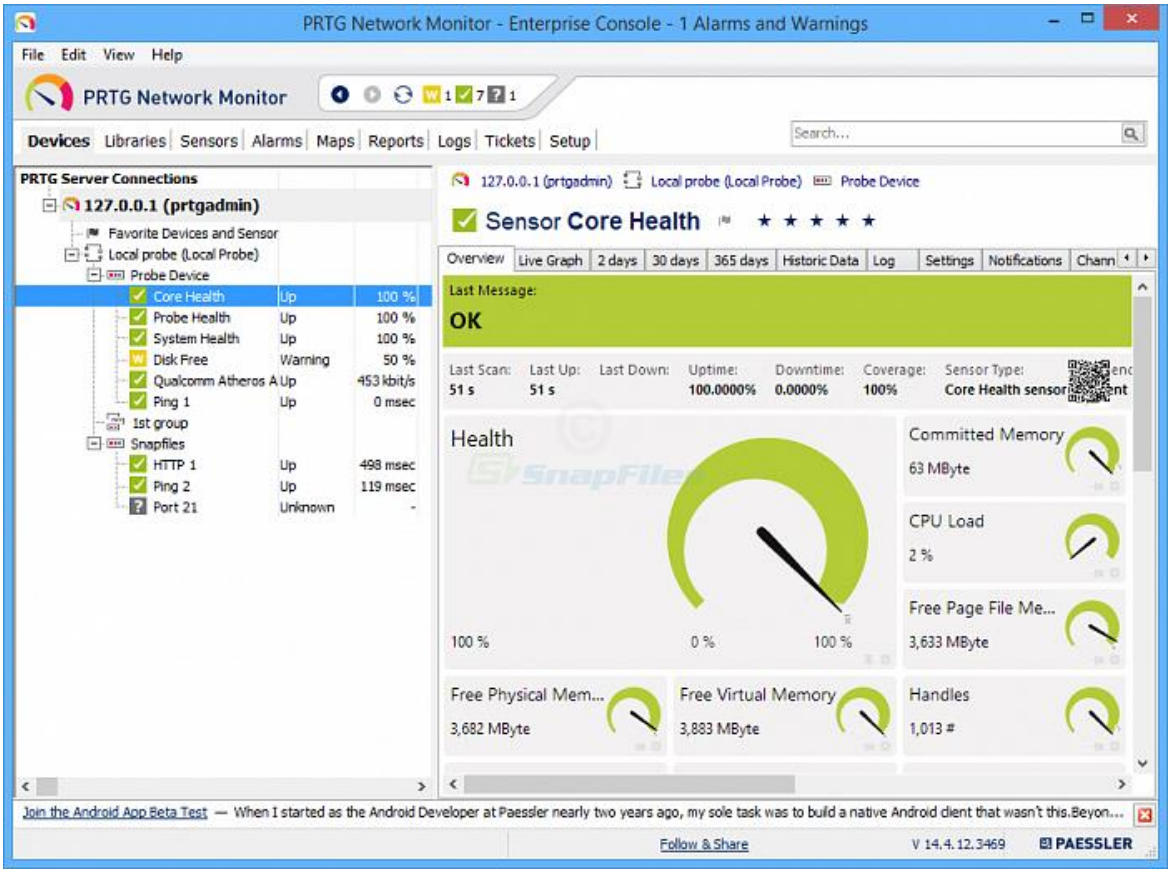

Рисунок 1.2 – Вигляд робочого вікна Paessler PRTG Network Monitor

Функціональні можливості системи PRTG забезпечують комплексний контроль за станом мережевої інфраструктури, дозволяючи здійснювати детальний моніторинг усіх основних параметрів її функціонування. Програмне рішення контролює роботу серверів, мережевих пристроїв, програмних додатків, веб-сервісів і трафіку, що проходить через мережу. Завдяки цьому користувач

отримує повну картину діяльності як локальних, так і глобальних сегментів IT-системи, включаючи доступність ресурсів, швидкість обміну даними та стабільність роботи окремих елементів.

Однією з ключових переваг PRTG є підтримка великої кількості сучасних технологій і протоколів, що робить її універсальним інструментом для адміністраторів різних рівнів. Система може працювати з такими протоколами, як Ping, SNMP, WMI, SSH, HTTP, а також аналізувати трафік за допомогою NetFlow, sFlow, jFlow, IPFIX. Це дозволяє детально вивчати структуру потоків даних, визначати потенційні перевантаження та своєчасно реагувати на загрози або збої.

Важливою складовою функціональності є автоматичне виявлення мережі, під час якого система самостійно ідентифікує підключені пристрої, збирає про них технічну інформацію та автоматично налаштовує відповідні сенсори моніторингу. Такий підхід значно скорочує час первинного налаштування, знижує ризик помилок і полегшує процес впровадження системи навіть у великих корпоративних середовищах.

Для зручності аналізу PRTG пропонує розвинені інструменти візуалізації даних. Користувач може створювати інтерактивні карти мережі, інформаційні панелі та графічні дашборди, які відображають поточний стан системи в реальному часі. Це спрощує моніторинг, полегшує аналіз тенденцій і сприяє прийняттю технічно обґрунтованих управлінських рішень.

Окрему увагу в системі приділено сповіщенням і повідомленням про події. PRTG дозволяє налаштовувати миттєві нотифікації про збої чи відхилення параметрів через електронну пошту, push-повідомлення або HTTP-запити, а також задавати індивідуальні порогові значення для кожного сенсора. Це забезпечує точний контроль стану мережі, оперативне реагування на проблеми та запобігання критичним ситуаціям.

Завдяки своїй масштабованості, автоматизації та широким можливостям інтеграції, PRTG є ефективним рішенням для проведення мережевої інвентаризації. Його функціонал органічно поєднується з підходами IT Asset

Management (ITAM) та IT Service Management (ITSM), що робить систему цінним інструментом для комплексного управління IT-інфраструктурою [3].

Рисунок 1.3 – Візуалізація моніторингової панелі Paessler PRTG Network Monitor

Головні IT-системи для інвентаризації даних та їх особливості:

ITAM (IT Asset Management) – це комплекс заходів, спрямованих на управління життєвим циклом IT-активів. Це включає в себе інвентаризацію, облік, оцінку вартості, управління конфігурацією, технічне обслуговування та утилізацію. Мета ITAM – оптимізувати використання IT-ресурсів, знизити витрати та забезпечити відповідність нормативним вимогам.

ITSM (IT Service Management) – це набір процесів, які забезпечують надання IT-послуг кінцевим користувачам. ITSM охоплює такі процеси, як управління інцидентами, управління проблемами, управління змінами, управління рівнем послуг тощо. Мета ITSM – забезпечити безперебійну роботу IT-систем та максимальну задоволеність користувачів.

Використання системи PRTG у межах концепцій ITAM (IT Asset Management) та ITSM (IT Service Management) забезпечує низку стратегічних

переваг для управління IT-інфраструктурою. Передусім вона значно підвищує рівень прозорості IT-середовища, надаючи адміністраторам постійний доступ до актуальних даних про стан обладнання, програмного забезпечення та мережевих компонентів. Завдяки цьому керівники IT-підрозділів мають змогу оперативно відстежувати зміни, своєчасно реагувати на збої та запобігати можливим інцидентам. Важливою перевагою є також здатність системи виявляти потенційні ризики на ранніх етапах, що мінімізує ймовірність серйозних технічних або фінансових втрат.

Точна інвентаризація ресурсів, яку забезпечує PRTG, сприяє оптимізації витрат, оскільки дозволяє уникати дублювання ліцензій, зменшувати надлишкові закупівлі обладнання та своєчасно планувати оновлення. Завдяки автоматизованому збору даних про всі пристрої, сервери, програми й ліцензії система формує структуровані звіти, які можуть використовуватись для аналітики або внутрішнього аудиту.

Система легко інтегрується з популярними рішеннями ITAM і ITSM — зокрема, з ServiceNow та Jira, — створюючи єдиний інформаційний простір для управління IT-ресурсами. Така інтеграція підвищує рівень узгодженості між технічними процесами та сервісним обслуговуванням, що сприяє безперервній, стабільній і безпечній роботі всієї IT-інфраструктури підприємства.

1.3. Аналіз популярних систем для інвентаризації даних

У сучасному бізнес-середовищі, де швидкість та точність відіграють ключову роль, застосунки для обліку та інвентаризації товарних цінностей стають незамінними інструментами для підтримки ефективного управління запасами та оптимізації ланцюжка постачання. В даному розділі розглядаються системи-аналоги, які надають різноманітні функції та можливості для контролю за товарними запасами. Кожна з цих систем має свої особливості, унікальний набір функцій та інтеграцію з іншими бізнес-інструментами [17].

InFlow Inventory – це програмне забезпечення для управління запасами та інвентаризації, яке дозволяє підприємствам ефективно керувати своїми запасами товарів. Даний застосунок надає широкий спектр функцій: відправка сповіщень про рівень запасів, керування поставками та замовленнями, використання штрих-кодів для ідентифікації товарів. Для забезпечення злагодженої роботи підприємства, даний застосунок може інтегруватися з іншими бізнес-програмами, такими як системи обліку фінансів, електронна комерція та зовнішні системи поставок. На рис. 1.4. зображено головну сторінку веб-сайту InFlow Inventory.

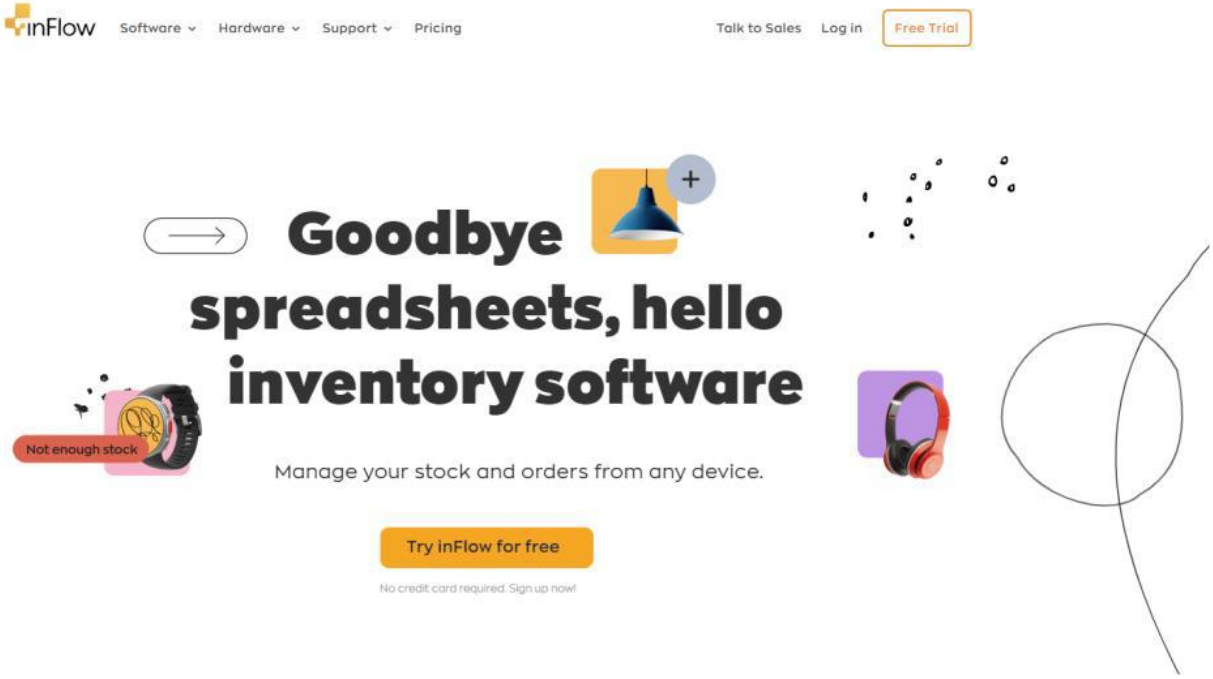

Рисунок 1.4 – Головна сторінка веб-сайту InFlow Inventory

Для ознайомлення з базовими можливостями програмного забезпечення передбачено безкоштовний тестовий період, який не потребує введення платіжних реквізитів. Після проходження процедури авторизації користувач отримує доступ до основного функціоналу системи, що дозволяє оцінити її можливості на практиці. Ключові функції застосунку наведено на рисунку 1.5.

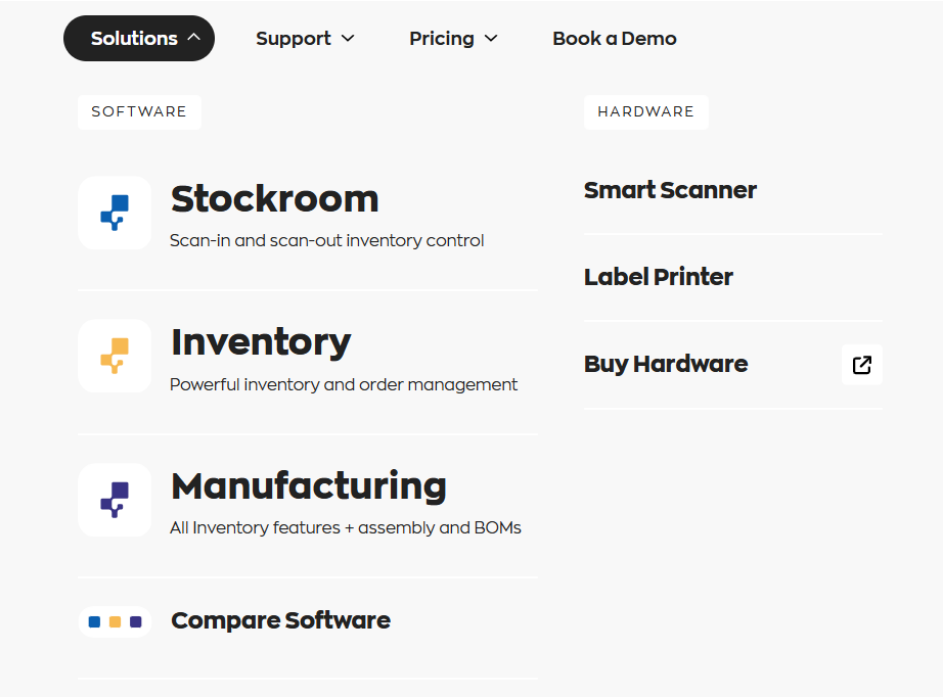

Рисунок 1.5 – Основний функціонал веб-сайту InFlow Inventory

Процес створення нового товару в системі є інтуїтивно зрозумілим і не потребує складних дій — достатньо ввести назву та визначити тип продукту (рис. 1.6) [17].

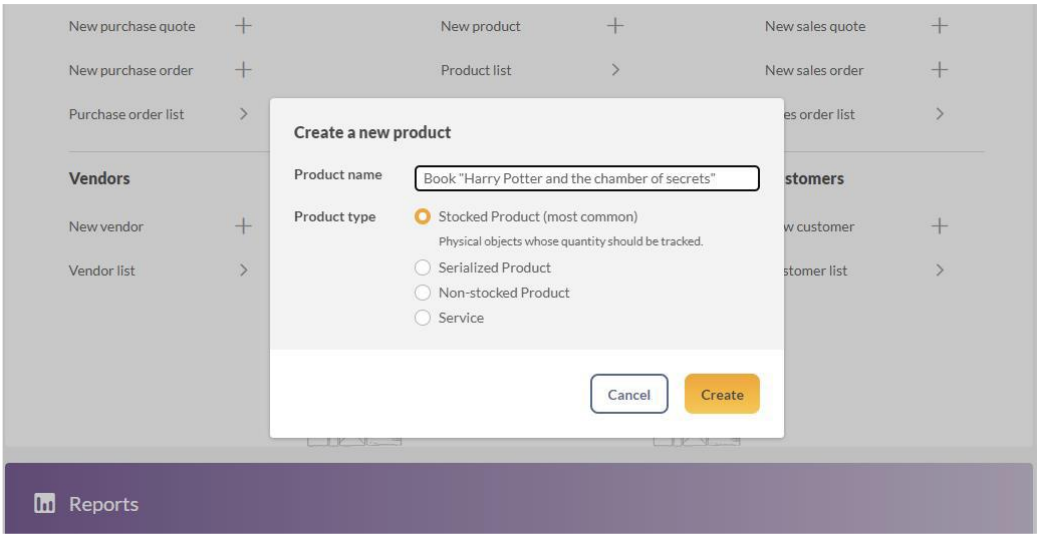

Рисунок 1.6 – Декларування товару веб-сайту InFlow Inventory

Після створення об’єкта користувачу відкривається детальна інформація про товар, яку можна за потреби редагувати. У відповідних полях передбачено можливість додавання опису, зазначення кількості, визначення

місцезнаходження та встановлення ціни. Приклад відображення детальної інформації про товар наведено на рисунку 1.6.

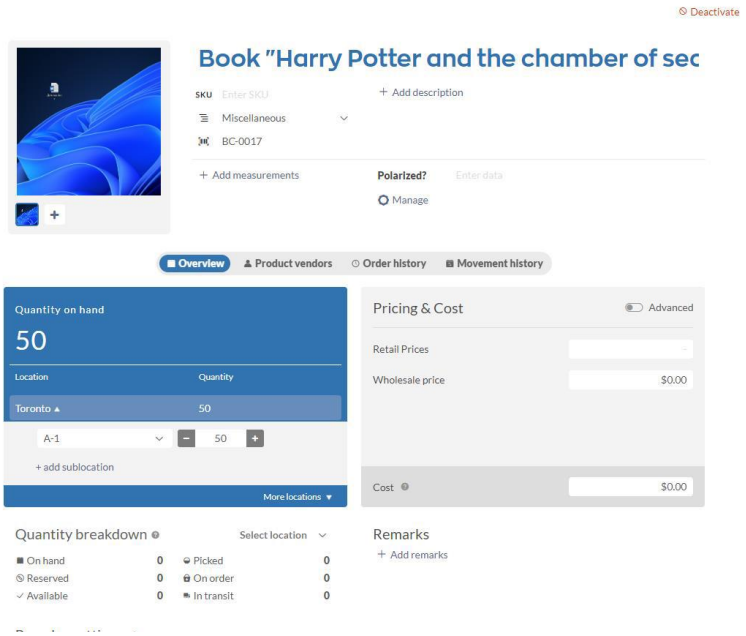

Рисунок 1.7 – Детальна інформація товару веб-сайту InFlow Inventory

Cin7 — це хмарна система управління запасами, призначена для комплексного обліку товарів, постачань і процесів продажу. Вона забезпечує централізоване керування всіма етапами товарообігу, починаючи від закупівлі та зберігання продукції й завершуючи її реалізацією кінцевому споживачеві. Система має широкий функціональний набір, що охоплює управління запасами, замовленнями, виробництвом, продажами, електронною комерцією, а також формування звітів і аналітику діяльності.

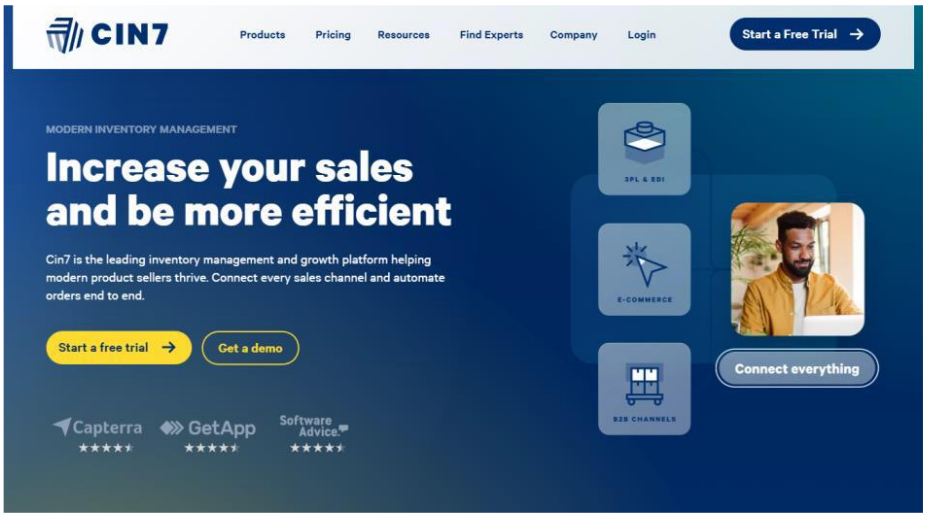

Рисунок 1.8 – Головна сторінка веб-сайту Cin7

Це дозволяє автоматично синхронізувати запаси та замовлення між різними каналами продажу, що спрощує процес обробки замовлень та уникнення надлишкових запасів. Головна сторінка веб-сайту Cin7 зображена на рис. 1.8.

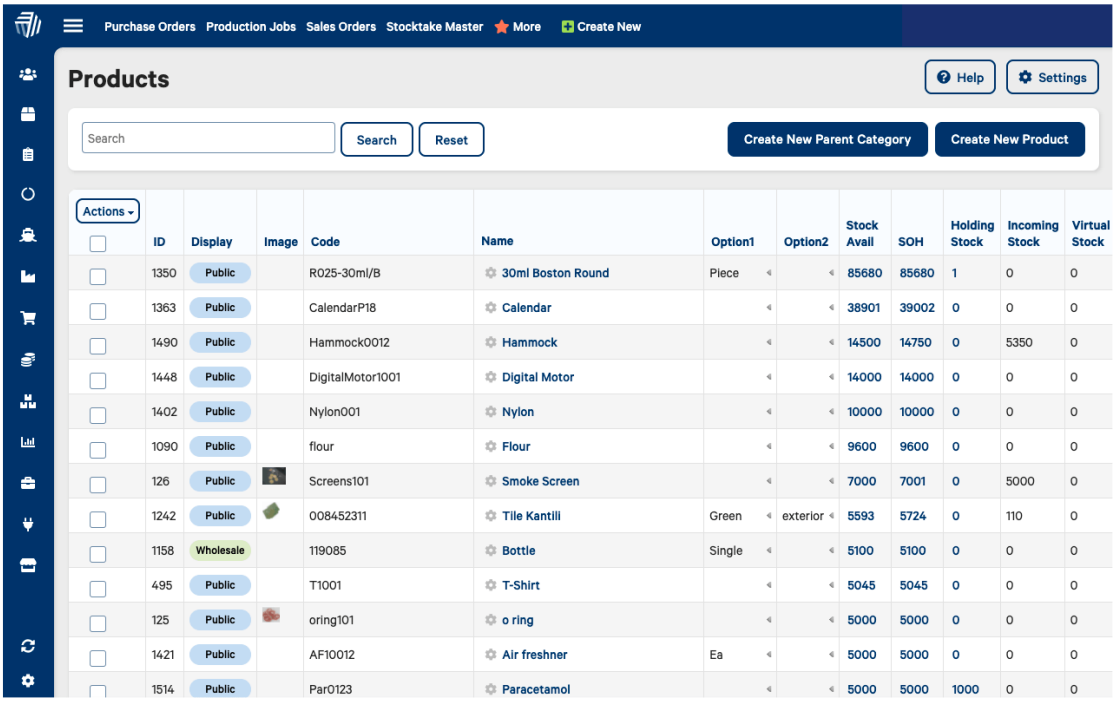

Рисунок 1.9 – Сторінка товарів веб-сайту Cin7

Варто зауважити, що застосунок Cin7 надає можливість ознайомлення з основними функціями системи у межах випробувального періоду, однак для проходження реєстрації необхідно використовувати корпоративну електронну адресу. У разі її відсутності система не дозволяє завершити процес створення облікового запису. На рисунку 1.9 подано інтерфейс сторінки, на якій відображено перелік товарів.

Процес створення нового товару в Cin7 є більш комплексним порівняно з inFlow Inventory, оскільки система вимагає введення розширеного набору параметрів. Користувачеві пропонується зазначити детальну інформацію про продукт, включаючи його характеристики, категорію, ціну, постачальників та інші дані. Візуальне представлення процесу створення товару в системі наведено на рисунку 1.10.

Після створення об’єкта користувач може редагувати інформацію про товар, вносити зміни та оновлювати дані безпосередньо через інтерфейс системи. Крім базових можливостей, Cin7 підтримує управління запасами та продажами на міжнародному рівні. Система дозволяє працювати з мультивалютними операціями, враховує міжнародні стандарти кодування товарів і забезпечує коректне застосування податкових правил різних країн, що робить її ефективним інструментом для глобального бізнесу.

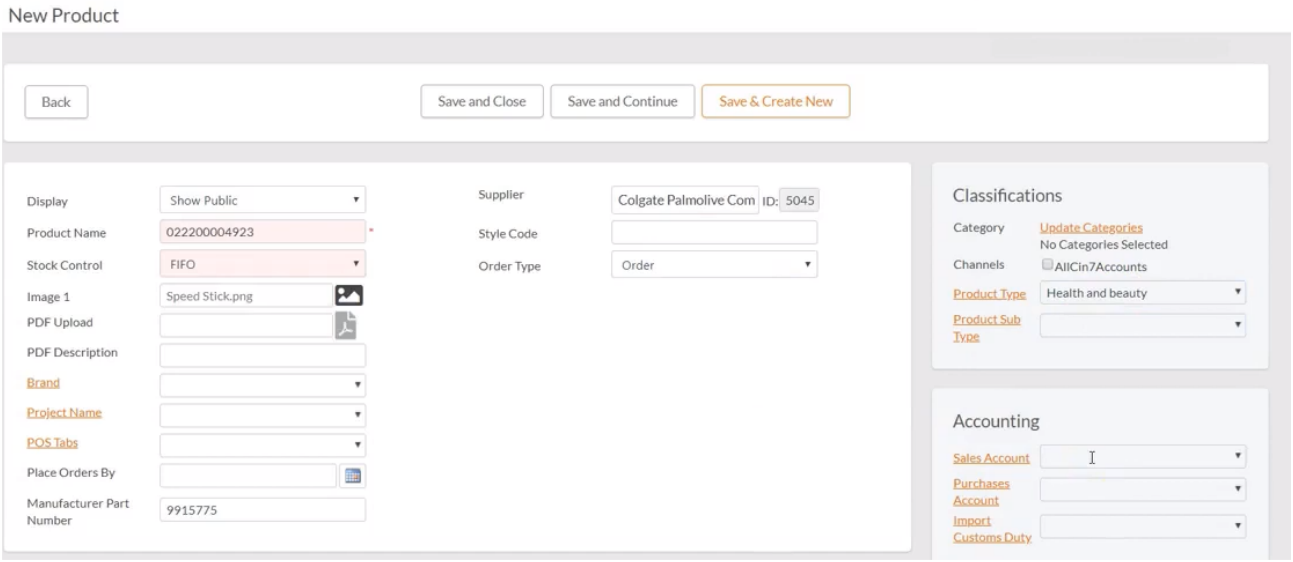

Рисунок 1.10 – Процес створення товару веб-сайту Cin7

Сервіс «Data Asset Inventory» від C&F SA дозволяє створити централізований каталог даних організації: збір даних про джерела, типи, місця зберігання, їх використання, а також супровід метаданих і аналіз.

Компанія C&F S.A. – це провідний постачальник рішень з управління даними та аналітики. Основною метою компанії є допомога бізнесам перетворити складну інформацію на якісні аналітичні інсайти та побудувати надійну й масштабовану IT-інфраструктуру.

Основні завдання, які вирішує C&F S.A.:

Компанія C&F надає послугу «Data Asset Inventory», яка дозволяє організаціям здійснити повну інвентаризацію своїх даних. Завдяки цій послузі можна отримати детальний огляд того, де зберігаються дані, хто має до них доступ та яким чином вони використовуються. Це є особливо актуальним для

підприємств, які прагнуть підвищити прозорість даних, ефективніше ними керувати та підвищити якість аналітичних висновків.

У межах управління метаданими створюються централізовані каталоги даних (data catalogs) та сховища метаданих, що значно полегшує пошук, розуміння та повторне використання даних у межах організації. Такий підхід дозволяє стандартизувати інформацію та оптимізувати процеси роботи з даними.

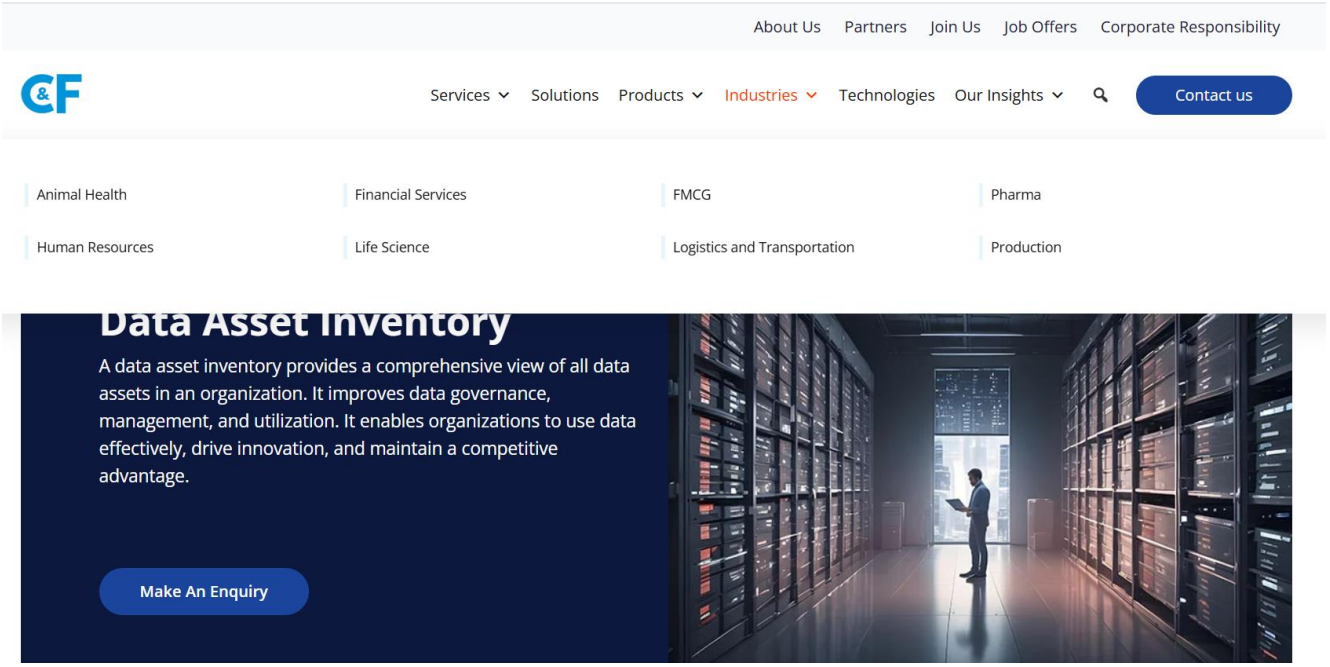

Рисунок 1.11 – Головне вікно веб-сайту C&F

C&F S.A. пропонує комплексні рішення для управління даними, що охоплюють не лише каталогізацію, а й архітектуру, інтеграцію, візуалізацію та автоматизацію процесів. Це дозволяє організаціям будувати цілісну стратегію роботи з інформацією, а не окремі фрагментарні інструменти. Завдяки понад 20-р. досвіду, команді з більш ніж 420 фахівців і реалізації проєктів у понад 60 країнах, компанія адаптує свої рішення до потреб різних галузей і масштабів бізнесу.

PIM (Product Information Management) – це платформа, яка допомагає централізувати та ефективно керувати всіма даними про товари у вашому e-commerce бізнесі. Вона забезпечує точність, узгодженість та актуальність інформації, спрощуючи її поширення на всі канали продажів.

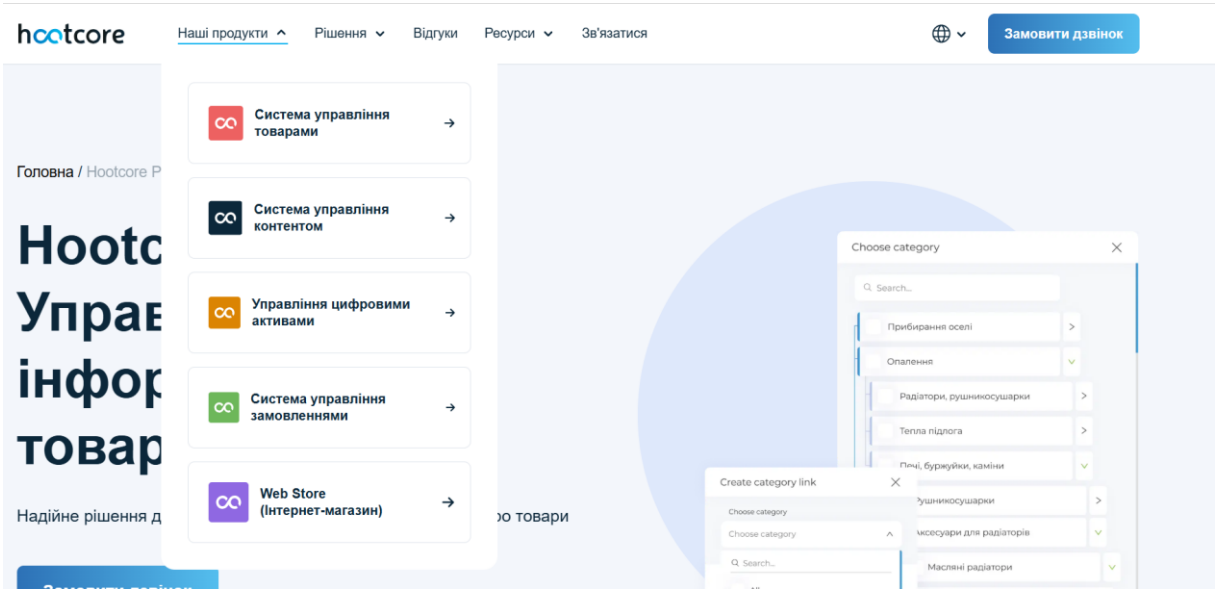

Рисунок 1.12 – Головне вікно веб-сайту C&F

Якщо працювати з великим асортиментом або складною номенклатурою товарів, PIM доцільно використовувати для оптимізації процесів та підвищення ефективності інтернет-магазину.

Система HootCore PIM дає змогу імпортувати товарні дані з різних зовнішніх джерел, оновлювати інформацію автоматично, створювати нові товари з попередньо налаштованими атрибутами, а також задавати і додавати характеристики продуктам — усе це допомагає підтримувати точність, узгодженість і актуальність даних про товари [3].

Щодо переваг застосування HootCore PIM, вони також добре виражені. Використання цієї системи допомагає значно прискорити вихід товарів на ринок, скоротити час і ресурси, витрачені на ручне налаштування, підвищити якість клієнтського досвіду за рахунок узгодженості даних і створити більш ефективну роботу бізнесу.

HootCore PIM побудована на сучасному стеку веб-технологій, орієнтованих на масштабованість і взаємодію з іншими інформаційними системами. Основу платформи становлять веб-серверна архітектура REST API, використання мови програмування Python (у поєднанні з Django або FastAPI для бек-енд-розробки) і технології баз даних PostgreSQL чи MongoDB для зберігання великих обсягів структурованих даних. На клієнтській стороні застосовується

JavaScript з React або Vue.js для побудови гнучкого й інтерактивного інтерфейсу користувача.

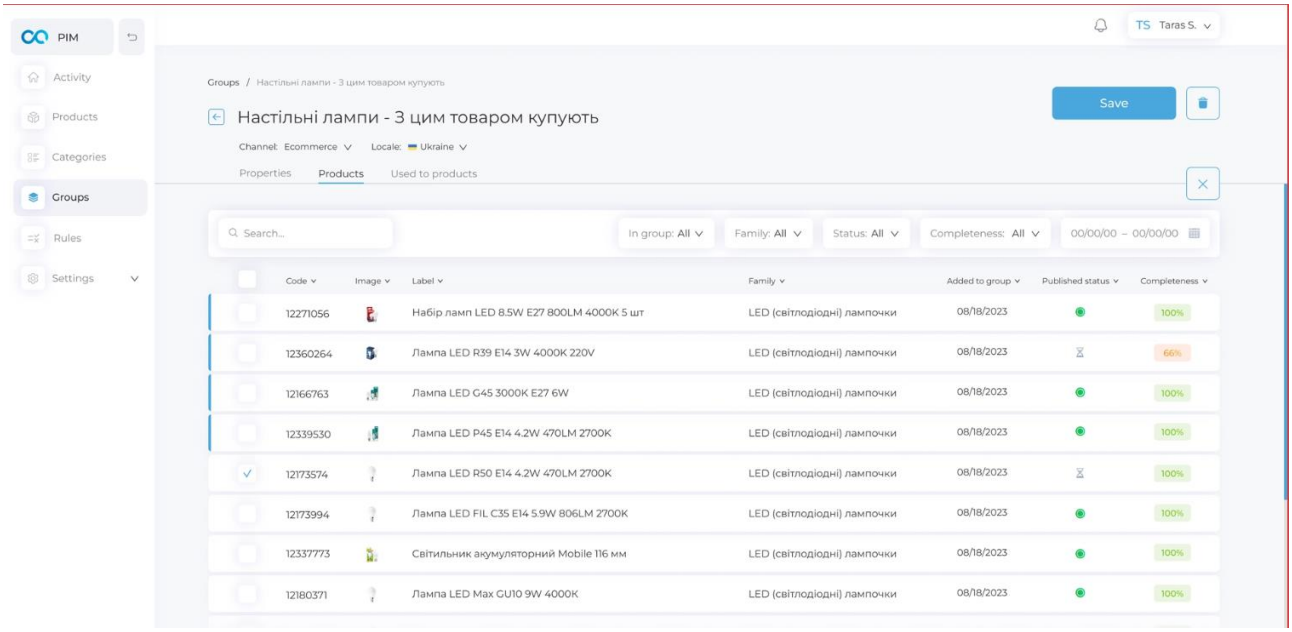

Рисунок 1.13 – Інвентаризація товарів в системі HootCore PIM

Ключовим елементом роботи є API-взаємодія – дані про товари надходять із зовнішніх джерел, проходять нормалізацію та записуються до центрального сховища. Наприклад, запит для отримання інформації про товар має такий вигляд (*python*) [20]:

```
import requests

url = "https://api.hootcore.io/pim/products/12345"
headers = {"Authorization": "Bearer <your_token>"}

response = requests.get(url, headers=headers)
product = response.json()

print(product["name"], product["price"])
```

У процесі імпорту даних система використовує ETL-модулі (Extract-Transform-Load), які очищують і структурують інформацію. Застосування таких технологій дозволяє інтегрувати HootCore PIM із ERP-, CRM- та CMS-системами, забезпечуючи єдине джерело правдивих даних.

РОЗДІЛ 2
СИСТЕМИ УПРАВЛІННЯ ІНВЕНТАРИЗАЦІЄЮ ДАНИХ

2.1. Переваги запровадження системи управління інвентаризацією

Впровадження інформаційної системи управління інвентаризацією на підприємстві надає відчутні переваги, які проявляються не лише в підвищенні точності обліку, а й у трансформації підходів до управління ресурсами загалом. Застосування автоматизованих рішень дозволяє підприємству перейти на якісно новий рівень ведення обліку, де кожна одиниця товару або матеріалу враховується, відстежується, аналізується та підлягає контролю в реальному часі.

Одним із ключових здобутків є істотне підвищення точності облікових операцій. На практиці це означає, що підприємство перестає залежати від людського чинника, який часто є джерелом неточностей, помилок і неузгодженостей.

Автоматизація виключає ризики, пов’язані з ручним введенням даних, дублюванням записів або втратою документів. Як наслідок – значно знижується кількість помилок у залишках, а також вірогідність виникнення ситуацій із нестачами або надлишками, які можуть завдати шкоди фінансовому стану компанії.

Завдяки точному й оперативному обліку запасів підприємство отримує можливість ефективно планувати закупівлі, уникати надлишкових замовлень, зменшувати обсяги зберігання, а отже - знижувати витрати.

Це особливо важливо для підприємств, які працюють із товарами з обмеженим терміном придатності або швидкозмінною номенклатурою, де будь-яке затримання або помилка у плануванні може спричинити списання великих обсягів продукції [7].

Іншою важливою перевагою є оперативність прийняття рішень. Усі дані, які зберігаються в системі, доступні в режимі реального часу, що дозволяє

керівникам на різних рівнях швидко отримувати зведення, аналітичні дані, порівняльні таблиці й прогнози. Це забезпечує більш гнучку й адаптивну модель управління, де реакція на зміни зовнішнього або внутрішнього середовища не затягується, а відбувається своєчасно і з урахуванням повної картини поточної ситуації.

Крім того, автоматизація інвентаризаційних процесів створює основу для глибокої аналітики. Зокрема, система накопичує історичні дані, які можна використовувати для виявлення сезонних тенденцій, аналізу ефективності постачальників, оцінки рентабельності окремих груп товарів. Це відкриває нові можливості для прогнозування, стратегічного планування та оптимізації асортименту.

Важливо також зазначити, що запровадження інформаційної системи підвищує рівень прозорості бізнес-процесів. Кожна дія в системі фіксується, кожна зміна має цифровий слід - це значно спрощує проведення внутрішніх аудитів, перевірок, звірок, а також дозволяє швидко виявляти джерела розбіжностей чи порушень. Така прозорість підвищує відповідальність працівників, зменшує кількість зловживань та сприяє формуванню дисциплінованішого корпоративного середовища.

Покращення прозорості процесів також сприяє підвищенню довіри з боку зовнішніх партнерів, інвесторів і контролюючих органів. Наявність чіткої системи контролю над ресурсами і точна звітність є сигналом про стабільність і надійність компанії, що, у свою чергу, сприяє зміцненню ділової репутації.

Інформаційна система також полегшує масштабування бізнесу. У разі відкриття нових складів, розширення торгової мережі або зміни логістичної структури підприємство не потребує повного переосмислення облікової політики - достатньо налаштувати параметри в межах вже функціонуючої системи. Це дозволяє уникнути додаткових витрат і зберегти цілісність усіх бізнес-процесів [4].

Значною перевагою впровадження сучасних ІТ-рішень є підтримка мобільних додатків, вебінтерфейсів, хмарних сервісів і інтеграцій з іншими

системами. Це надає можливість працювати з даними не лише з робочого місця, а й із будь-якої точки, де є інтернет. Таким чином, керівники або логісти можуть оперативно реагувати навіть у відрядженні чи в дорозі.

Не менш важливою є й довгострокова перспектива. Впровадження інформаційної системи інвентаризації є не просто відповіддю на поточні потреби підприємства, а кроком до його повноцінної цифрової трансформації. Такий перехід дозволяє закласти фундамент для побудови більш ефективної, адаптивної, технологічної бізнес-моделі, яка здатна витримувати конкуренцію навіть у дуже динамічному середовищі.

Загалом, впровадження інформаційної системи управління інвентаризацією формує нову якість управління підприємством, забезпечуючи контроль, прозорість, стабільність і готовність до масштабування. Це стратегічне рішення, яке має реальний фінансовий ефект, зміцнює позиції компанії на ринку та створює передумови для її сталого розвитку у майбутньому.

2.2. Структуризація цілей розробки ІС

Сутність процесу структуризації цілей розробки ІС полягає у поступовому розчленуванні загальної мети на підцілі та завдання нижчих рівнів, щоб забезпечити логічну послідовність, вимірюваність і досяжність кожного елемента. Таким чином, формується основа для подальшого проектування структури, функцій та компонентів системи.

Отже, дерево цілей розробляють під час проектування інформаційних систем (ІС) для того, щоб структуровано відобразити взаємозв’язки між загальною метою системи та підцілями, які забезпечують її досягнення. Воно дозволяє чітко сформулювати головну мету створення інформаційної системи, наприклад підвищення ефективності управління підприємством або автоматизацію окремих процесів.

У процесі побудови дерева цілей загальна мета послідовно розкладається на підцілі нижчих рівнів, що є більш конкретними, вимірюваними й досяжними (рис. 2.1). Такий підхід дає змогу визначити логічні зв’язки між окремими цілями, з’ясувати, які завдання потрібно виконати для досягнення кожного рівня, а також забезпечити системний підхід до проектування.

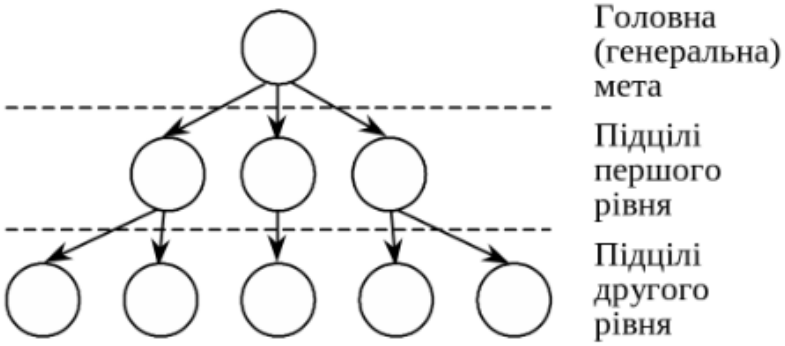

Рисунок 2.1 – Граф дерева цілей [5]

Дерево цілей виступає методичною основою для декомпозиції та узгодження цілей інформаційної системи на всіх етапах її розроблення, сприяє збереженню цілісності проекту, забезпечує узгодженість між його складовими та підвищує ефективність майбутньої системи.

Дерево цілей – це візуальне зображення ієрархії між цілями та засобами їх досягнення. Коренем дерева цілей є генеральна мета – основна мета системи, яка в деталізується поділяючись на окремі підцілі.

Під час побудови дерева цілей необхідно дотримуватися низки вимог, що забезпечують його логічність і практичну ефективність. Усі цілі мають бути взаємопов’язаними та спрямованими на досягнення головної, генеральної мети системи. Кожна з них повинна бути реальною, чітко визначеною, несуперечливою та зрозумілою всім учасникам процесу, що гарантує її досяжність. Важливо, щоб дерево цілей було прозорим і зрозумілим для всіх залучених сторін, адже це запобігає неправильному тлумаченню та організаційним непорозумінням. Структура дерева повинна включати цілі різних рівнів, що дозволяє встановити пріоритети, визначити послідовність дій і

забезпечити раціональне планування. Крім того, дерево цілей має залишатися гнучким і здатним до адаптації в разі зміни умов або завдань проекту.

Отже *основною метою є* – створення системи обліку та інвентаризації товарно-матеріальних цінностей підприємства. Мета ділиться на три підцілі першого рівня, які забезпечують основну структуру. Дерево цілей для системи зображене на рис. 2.2.

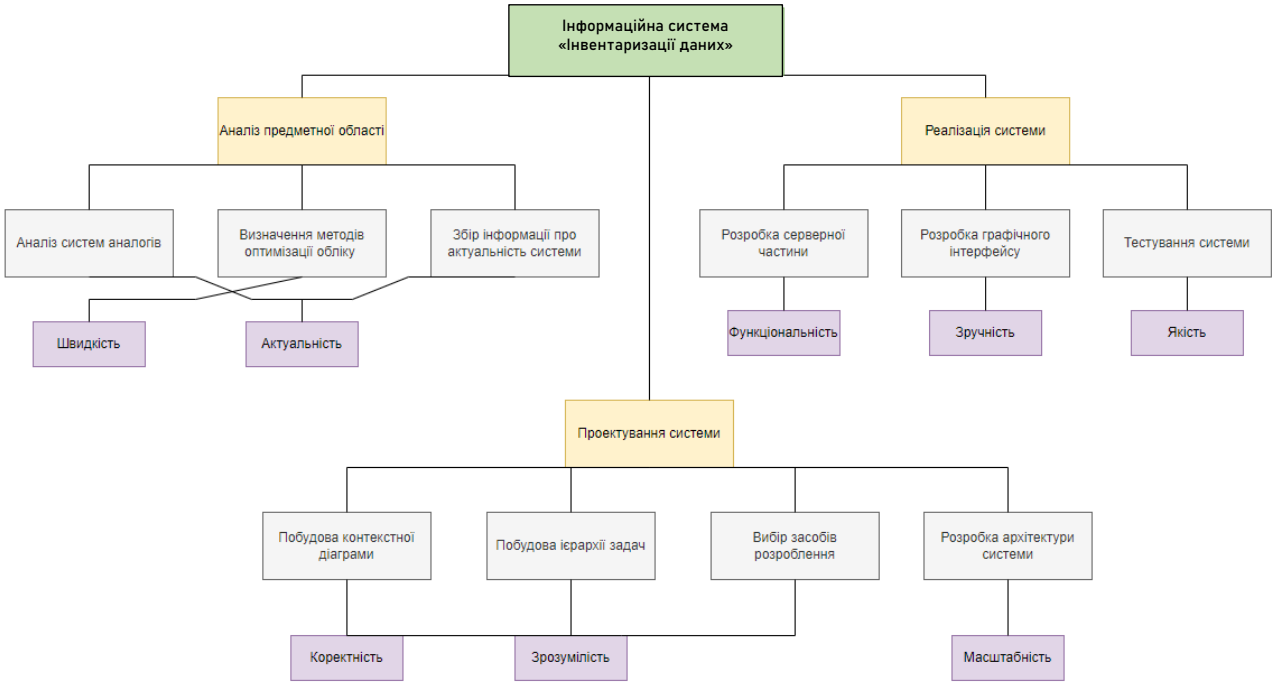

Рисунок 2.2 – Дерево цілей створення ІС

Підцілями *першого рівня* є:

1) аналіз предметної області – огляд літературних та інтернет-джерел, збір та аналіз отриманої інформації;

2) проектування системи – процес формулювання основних вимог системи, розробка діаграм;

3) реалізація системи – процес створення системи, розробка інтерфейсу та програмної частини.

Другий рівень є декомпозицією підцілей першого рівня. Перша підціль «Аналіз предметної області» поділяється на наступні підцілі: 1) аналіз систем аналогів – дослідження систем аналогів, які вирішують проблеми запасів товарів;

2) визначення методів оптимізації обліку – визначення основних проблем, які можна вирішити, введенням даної системи; 3) збір інформації про актуальність системи – оцінювання можливих перешкод, потрібність введення даної системи.

Друга підціль «Проектування системи» поділяється на наступні підцілі: 1) побудова контекстної діаграми – створення діаграми, яка описує структуру системи з переліком сутностей; 2) побудова ієрархії задач – обумовлення основних задач, які повинні виконуватись сисстемою та процесів, за допомогою яких система виконує свою роботу; 3) вибір засобів розроблення – вибір основних засобів для реалізації системи, мови програмування; 4) розробка архітектури системи – розробка та проектування архітектури серверної частини проекту.

Третя підціль «Реалізація системи» поділяється на наступні підцілі: 1) розробка серверної частини – розробка програмного забезпечення системи, основного функціоналу, взаємодії з базою даних для отримання необхідних даних; 2) розробка графічного інтерфейсу – розробка візуальної складової системи для покращення користувацького досвіду; 3) тестування системи – перевірка всього функціоналу системи з метою виявлення помилок, і подальшого їх виправлення.

Після цього, можемо визначити критерії, які допоможуть створити якісну характеристику підцілей другого рівня декомпозиції, було сформовано 8 критеріїв:

Масштабність – забезпечує широке дослідження предметної області для виявлення основних проблем, на основі яких можна зрозуміти потреби користувачів.

Актуальність – забезпечує доцільність проектованої системи, та оптимізує облік товарних запасів.

Швидкість – забезпечує швидкий доступ до даних при взаємодії користувачів із системою, забезпечує миттєвий зворотній зв’язок.

Зручність – забезпечує зручний та інтуїтивно зрозумілий користувацький інтерфейс, який не потребує значного витрачання часу для того, щоб почати з ним працювати.

Зрозумілість – забезпечує чіткі та послідовні дії в описі контекстних діаграм та ієрархії задач для реалізації системи.

Якість – забезпечує конфіденційність системи, захищеність користувацьких даних, а також стабільність та стійкість системи, завдяки надійним алгоритмам функціонування.

Функціональність – забезпечує систему якісним та повним функціоналом.

Коректність – забезпечує правильність досліджень систем для найкращого розуміння потреб користувача.

Для вибору типу розроблюваної системи застосовують низку критеріїв, що дозволяють об’єктивно оцінити її відповідність поставленим завданням. До основних критеріїв належать масштабність (*К1*), яка визначає здатність системи ефективно функціонувати за різних обсягів даних та кількості користувачів; актуальність (*К2*), що відображає ступінь відповідності сучасним потребам і тенденціям; швидкість (*К3*), яка характеризує продуктивність та оперативність оброблення інформації; зручність (*К4*), що оцінює простоту використання системи для кінцевого користувача; зрозумілість (*К5*), яка забезпечує доступність інтерфейсу та логіку взаємодії; якість (*К6*), що відображає надійність і стабільність роботи; функціональність (*К7*), яка визначає повноту реалізації необхідних можливостей, і коректність (*К8*), що гарантує правильність виконання всіх операцій системою.

Також, для визначення типу розроблювальної системи, використаємо метод аналізу ієрархій [7]. Згідно його концепції, слід розглянути 4 інформаційні системи та вибрати одну із них: 1) інформаційно-пошукова система (А1) – система, з допомогою якої відбувається пошук інформації відповідно заданих системою або користувачем параметрів; 2) інформаційно-довідкова система (А2) – система, задачею якої є зберігання та обробка інформації, а також надання цієї інформації за запитом користувача; 3) інформаційно-аналітична система (А3) –

система, задачею якої є обробка інформації та аналіз; 4) інформаційна система прийняття рішень (А4) – система, яка відповідає за підтримку у виборі рішень на основі результатів аналізів, проведених нею.

Для проектованої ІС побудовано структуру за методом аналізу ієрархій, яка представлена на рис. 2.3.

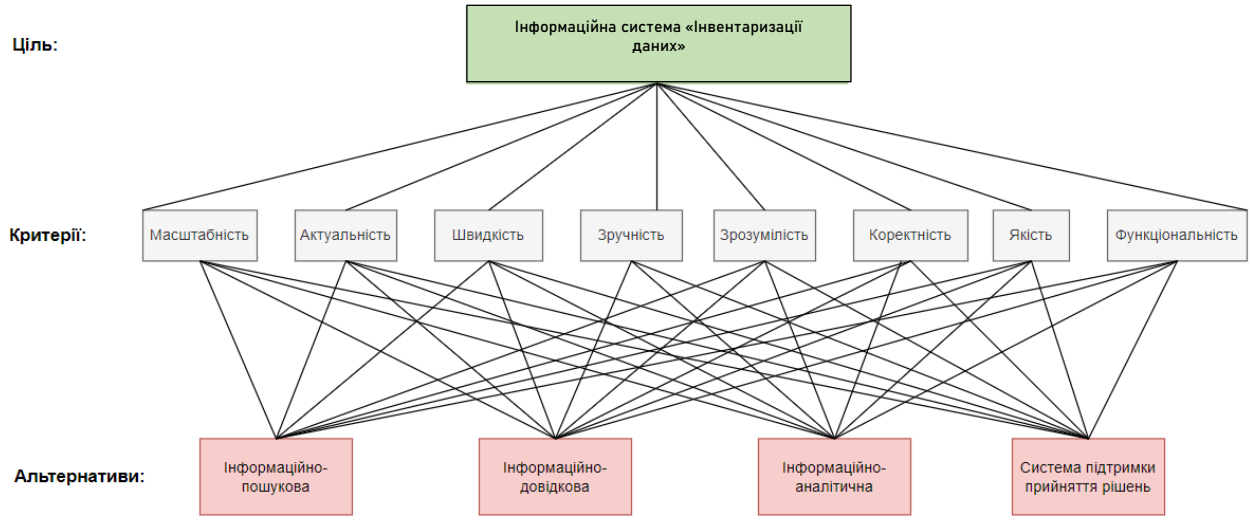

Рисунок 2.3 – Аналіз ієрархій побудови ІС

За результатами застосування методу аналізу ієрархій можна зробити висновок, що найбільше нам підходить інформаційно-довідкова система, яка має пріоритет.

2.3. Моделювання потоків даних та побудова діаграми декомпозицій

Діаграма потоку даних (DFD) – це графічний метод моделювання потоків даних та процесів в інформаційній системі. DFD є ефективним засобом аналізу та проектування бізнес-процесів, оскільки вона дозволяє зображувати як дані проходять через систему та як вони обробляються різними процесами.

У DFD дані зображуються як потоки, а процеси - як блоки, що обробляють ці потоки. Такі блоки можуть бути різних типів, включаючи вхідні, вихідні, опрацювання даних та контрольні. Вона допомагає виявляти недоліки та

можливості для оптимізації процесів в системі. Також допомагає в узгодженні різних зацікавлених сторін, які можуть мати різні уявлення про процеси в системі, та допомагає зрозуміти, які дані потрібні та як вони повинні бути оброблені [5].

Діаграма DFD містить наступні елементи: 1) процес – виконує дії над даними, наприклад створює, оновлює, видаляє; 2) сховище даних – використовується для збереження даних; 3) зовнішня сутність – є джерелом або адресатом потоку даних; 4) потік даних – зображений стрілкою, відображає потік даних між процесами, сховищами, зовнішніми сутностями.

Для побудови контекстних діаграм із використанням структурного підходу та моделі проектування Data Flow використовуємо середовище Allfusion Process Modeler r7.

В контекстній діаграмі передбачено зовнішню сутність користувача та основний процес (рис. 2.4)

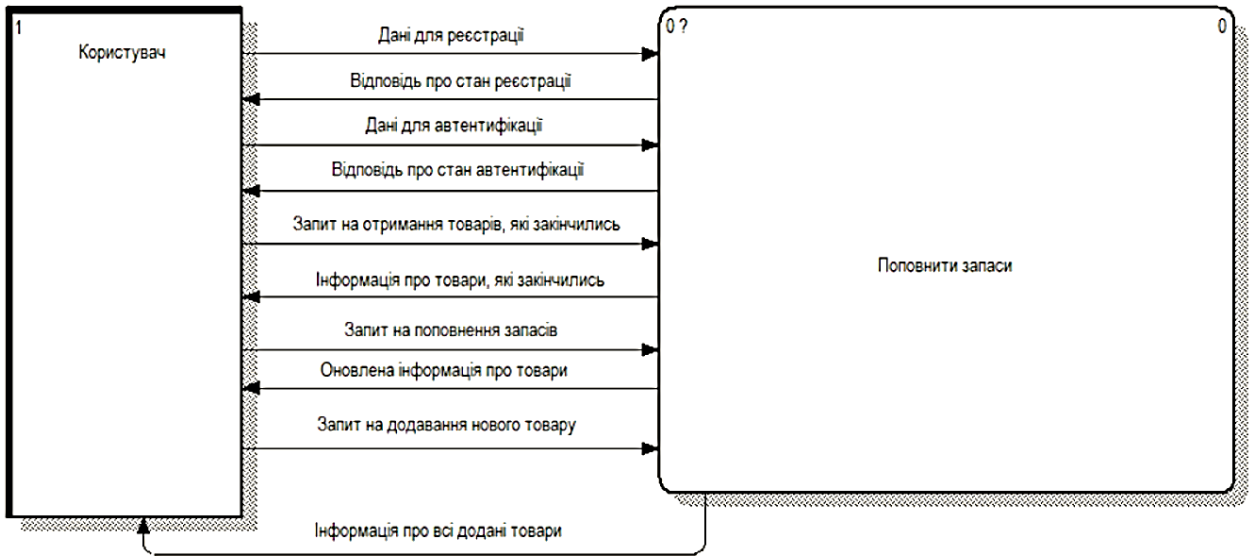

Рисунок 2.4 – Контекстна діаграма ІС

Для додавання товару, користувач повинен авторизуватись в систему для отримання доступу, або ж зареєструватись, якщо його немає в базі даних. Після успішної авторизації, користувач отримує доступ до додавання товару.

Наступним кроком буде декомпозиція головного процесу. Для правильності декомпозиції, потрібно зберігати ієрархію процесів. На діаграмі відображені головні процеси: зареєструвати користувача, автентифікувати користувача, додати товар до сховища даних, змінити категорію/локацію, зберегти товар (рис. 2.5).

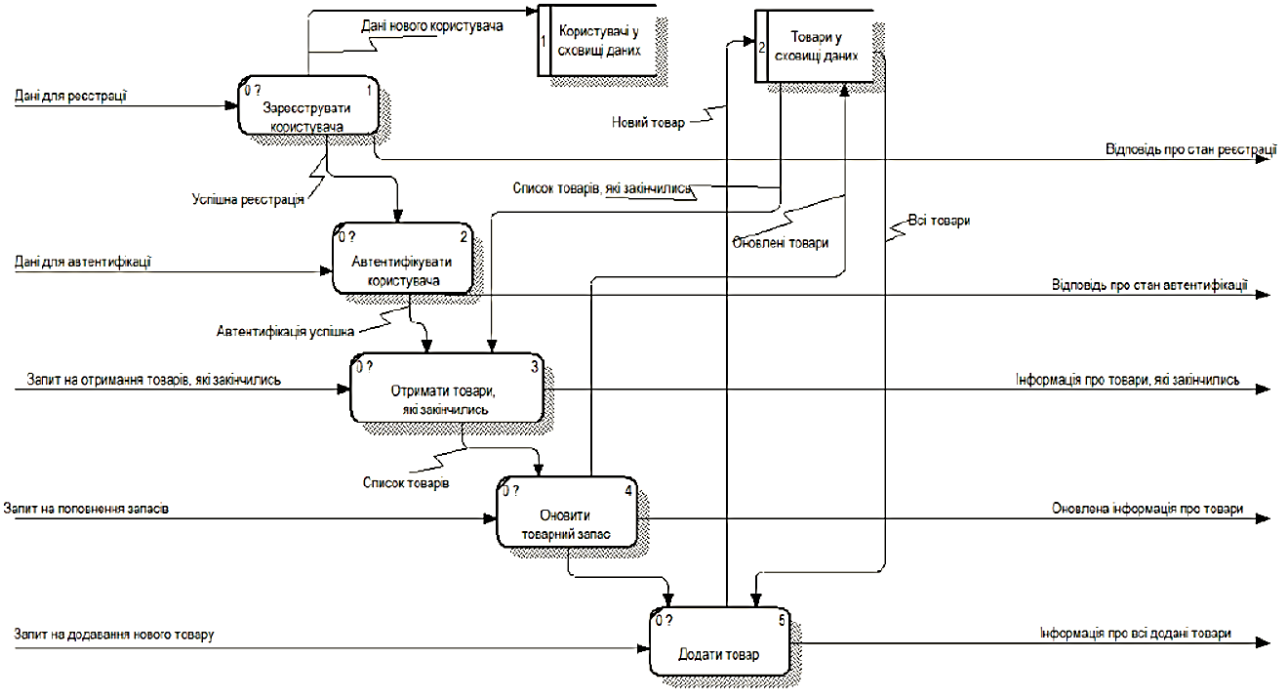

Рисунок 2.5 – Діаграма декомпозиції першого рівня

Процес «Зареєструвати користувача» відповідає власне за реєстрацію нового користувача, після чого запускається процес «Автентифікувати користувача» який відповідає за автентифікацію, після успішною автентифікації користувач отримає доступ до управління запасами системи. Процес «Отримати товари, які закінчились» дозволяє отримати товари, які закінчились із сховища товарів, після чого користувач може запустити процес «Оновити товарний запас», який оновлює інформацію про товари в сховищі даних. Процес «Додати товар» відповідає за додавання нового товару в сховище даних.

Наступним етапом є декомпозиція другого рівня для процесу «Зареєструвати користувача». Діаграма наведена на рис 2.6.

Рисунок 2.6 – Діаграма декомпозиції другого рівня процесу «Зареєструвати користувача»

Зазначений процес перевіряє дані, введені користувачем, на коректність, після цього додає користувача у базу даних. Після успішної верифікації користувача система сповістить його.

Наступні процеси щодо «Автентифікувати користувача», «Отримати товари, які закінчились», «Отримати товари, які закінчились», «Оновити товарний запас», «Додати товар» зображений в дод. А.1.

РОЗДІЛ 3
МЕТОДИКА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Ієрархія процесів та ERP діаграма ІС

Ієрархія процесів або ієрархія задач – це структурований підхід до опису функціональної поведінки системи, що складається зі списку задач та їх взаємозв'язків. Використовується дана ієрархія для опису та управління процесами, які відбуваються в системі, а також для організації компонентів системи в логічній послідовності. При правильному підході до розробки діаграм, в середовищі AllFusion Process modeler стає доступною можливість побудувати ієрархію процесів у вигляді дерева вузлів. На вершині цього дерева знаходиться основна задача системи, після цього від неї походить декомпозиція першого рівня, і від неї, відповідно, декомпозиція другого рівня.

На рис. 3.1 зображена діаграма ієрархії процесів.

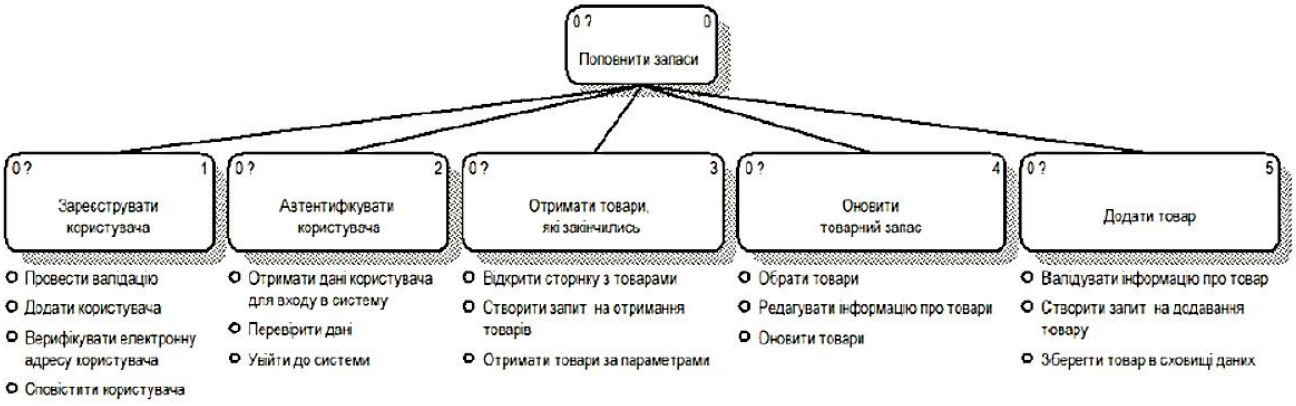

Рисунок 3.1 – Діаграма ієрархії процесів

Основна ціль знаходиться на вершині дерева вузлів, та має назву «Поповнити запас». Основна ціль, розбивається на 5 підцілей, які мають бути виконані для досягнення мети: 1) зареєструвати користувача; 2) автентифікувати користувача; 3) отримати товари, які закінчились; 4) оновити товарний запас; 5) додати товар.

У свою чергу, кожна з підцілей поділяється на власні підцілі. Перша підціль «Зареєструвати користувача» містить наступні задачі: 1) провести валідацію даних; 2) додати користувача до системи; 3) верифікувати електронну адресу користувача; 4) сповістити користувача про успішну реєстрацію.

Підціль «Автентифікувати користувача» містить такі задачі: 1) отримати дані користувача для входу в систему; 2) перевірити коректність введених даних; 3) надати користувачу доступ до системи.

Підціль «Отримати товари, які закінчились» включає: 1) відкрити сторінку з товарами; 2) створити запит на отримання даних про товари; 3) отримати список товарів за заданими параметрами.

Підціль «Оновити товарний запас» охоплює такі задачі: 1) обрати товари для оновлення; 2) редагувати інформацію про вибрані товари; 3) оновити дані про товари в системі.

Підціль «Додати товар» передбачає: 1) провести валідацію інформації про товар; 2) створити запит на додавання нового товару; 3) зберегти товар у сховищі даних.

ER діаграма (*Entity-Relationship diagram*) – являє собою модель даних, що використовується для наочного відображення взаємозв’язків між різними об’єктами в базі даних. Вона складається із сутностей, їхніх атрибутів та взаємозв’язків. Сутність у цьому контексті – це об’єкт або концепція, що зберігається в базі даних, наприклад, товар чи користувач. Взаємозв’язок відображає логічні зв’язки між сутностями та демонструє, як вони взаємодіють між собою. Атрибути характеризують сутність і визначають, які дані про неї зберігаються; наприклад, для товару це можуть бути код, назва, категорія та місцезнаходження.

Використання ER-діаграми дозволяє розробнику бази даних чітко визначити, які саме дані буде обробляти інформаційна система та яким чином вони взаємопов’язані, що забезпечує правильне та структуроване проектування бази даних.

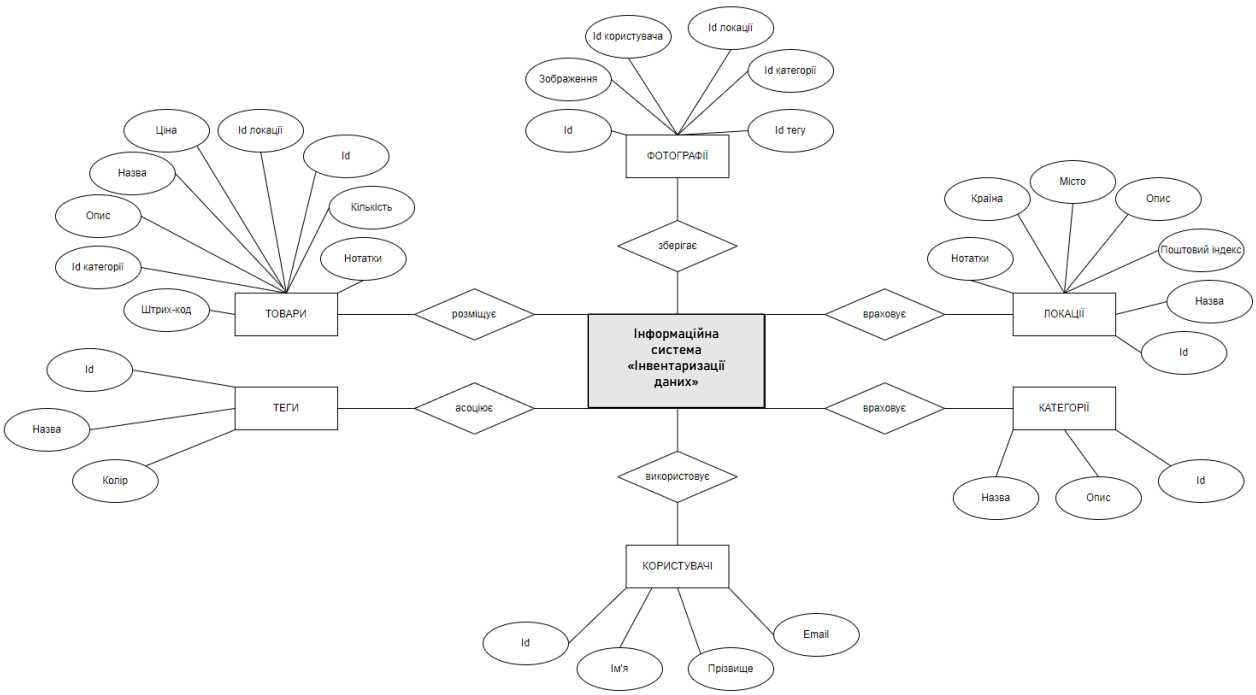

Рисунок 3.2 – ER-діаграма інформаційної системи ініентаризації даних підприємства

Отже, для ефективного функціонування інформаційних систем використовуються різні моделі та інструменти, зокрема дерева цілей, DFD-діаграми та ER-діаграми. Дерево цілей дозволяє чітко структурувати генеральну мету та підцілі, визначити порядок виконання завдань і забезпечити наочне відображення взаємозв'язків між ними. DFD-діаграми демонструють потоки даних у системі та процеси їх обробки, а ER-діаграми забезпечують візуалізацію зв'язків між сутностями бази даних, що спрощує проектування та управління даними. Використання таких підходів сприяє оптимізації роботи IT-систем, підвищенню продуктивності, безпеки та якості обробки інформації, а також дозволяє забезпечити чітку структуру даних і ефективне управління IT-активами.

3.2. Вибір інструментів для реалізації ІС

У сучасних умовах, коли доступ до мережі Інтернет є практично

повсюдним, найбільш доцільним рішенням для створення інформаційної системи обліку та інвентаризації товарно-матеріальних цінностей виступає веб-платформа. Такий підхід є оптимальним, оскільки забезпечує універсальність і зручність користування системою з будь-якого пристрою — персонального комп'ютера, ноутбука, смартфона чи планшета. Розроблення подібної системи передбачає два основні етапи: front-end та back-end.

Front-end являє собою клієнтську частину веб-додатку, тобто інтерфейс, через який користувач безпосередньо взаємодіє із системою. Саме цей рівень відповідає за візуальне відображення даних, зручність навігації та інтерактивність роботи. Для реалізації front-end частини було обрано відповідний стек технологій, що забезпечує стабільність, адаптивність і швидкодію веб-додатку: *HTML, CSS, SCSS, JavaScript, TypeScript, Angular, Ng-zorro (Ant design), QuaggaJS, JetBrains WebStorm Professional Edition*.

Back-end — це серверна частина системи, яка відповідає за реалізацію бізнес-логіки та обробку запитів, приховану від користувача. Коли користувач здійснює певну дію у графічному інтерфейсі, відповідний контролер на сервері приймає цей запит, обробляє його та надсилає результат назад до клієнтської частини. Для реалізації back-end компонента системи обрано відповідний технологічний стек, який забезпечує стабільність, масштабованість і безпечну взаємодію між клієнтом та сервером: *C#, ASP .NET Core, Microsoft SQL Server, SQL Server Management Studio, Entity Framework Core, Microsoft Visual Studio 2022*.

Вичерпну інформацію щодо властивостей та переваг застосування цих інструментів можна знайти на сайтах-сервісах розробників. Однак, розглянемо деякі із них.

JavaScript – це мультипарадигменна мова програмування, яка надає життя веб-сторінці. Завдяки стилям і розмітці ми створюємо картинку і структуру сайту, а завдяки JavaScript веб-сайт отримує змогу виконувати функції [16]. *TypeScript* – це мова програмування, яка розширює стандартний JavaScript, вона замінює динамічну типізацію на статичну, додає класи, інтерфейси, що дозволяє

розробникам писати більш надійний, зрозумілий, та структурований код [2]. Дана мова програмування повністю базується на синтаксисі JavaScript, і може виконуватись на будь-якому драйвері, такому як Node.js або ж в браузері. Зразок коду на TypeScript для Node.js, який демонструє використання класу та інтерфейсу – для товару та класу, який реалізує цей інтерфейс і виконує прості операції з даними:


```
// Інтерфейс описує структуру об'єкта "товар"
interface Product {
  id: number;
  name: string;
  price: number;
  category?: string; // необов’язкове поле
}

// Клас реалізує інтерфейс Product і містить методи для роботи з товарами
class InventoryItem implements Product {
  id: number;
  name: string;
  price: number;
  category?: string;

  constructor(id: number, name: string, price: number, category?: string) {
    this.id = id;
    this.name = name;
    this.price = price;
    this.category = category;
  }

  // Метод для зміни ціни
  updatePrice(newPrice: number): void {
    if (newPrice <= 0) {
      throw new Error("Ціна повинна бути більшою за нуль.");
    }
    this.price = newPrice;
    console.log(`Оновлено ціну товару "${this.name}" до ${this.price} грн.`);
  }

  // Метод для отримання короткої інформації про товар
  getInfo(): string {
    return `Товар: ${this.name}, Категорія: ${this.category ?? "Невизначена"}, Ціна: ${this.price} грн.`;
  }
}

// Приклад використання класу
```

```
const laptop = new InventoryItem(1, "Ноутбук Lenovo", 32000, "Електроніка");
console.log(laptop.getInfo());

laptop.updatePrice(29999);
console.log(laptop.getInfo());
```

Тут інтерфейс Product визначає структуру даних, яку має реалізовувати клас. Клас InventoryItem не лише реалізує інтерфейс, а й додає власну логіку — зміну ціни та виведення інформації про товар.

Angular – це веб-фреймворк, який був розроблений всесвітньо відомою компанією Google [14]. Цей фреймворк дозволяє розробляти великі, складні, продуктивні та масштабовані веб-додатки. Основною мовою даного фреймворку є TypeScript.

Головною перевагою даного фреймворку є модульність та використання компонентів, які можна використовувати в програмі. Angular забезпечує патерн MVC (Model, View, Controller). Model знаходиться на самому низькому рівні шаблону, який відповідає за управління даними застосунку.

Ng-zorro (Ant design) – це колекція компонентів графічного інтерфейсу, яка працює на базі Angular. Містить набір готових колекцій, які можна використовувати для швидкої та легкої розробки користувацького інтерфейсу. Перевагою серед інших бібліотек є підтримка інтерналізації, тобто підтримка багатьох мов. Це дозволяє створювати веб додатки для різних мов та культур, не засмічуючи при цьому базу даних та розмітку веб додатка.

QuaggaJS – це JavaScript-бібліотека для розпізнавання штрих-кодів зображень на стороні клієнта [18]. Основна мета QuaggaJS – надати простий інтерфейс для розпізнавання штрих-кодів, який може бути використаний в браузері або на мобільних пристроях з використанням веб-камери. Дана бібліотека використовує технології комп'ютерного зору, такі як алгоритми розпізнавання шаблонів та машинного навчання, для виявлення та розпізнавання

штрих-кодів. Бібліотека підтримує розпізнавання різних типів штрих-кодів, таких як EAN, UPC, Code 128, Code 39 та QR-коди [19]:

```
import Quagga from 'quagga';

Quagga.init({
  inputStream: {
    name: "Live",
    type: "LiveStream",
    target: document.querySelector('#camera') // елемент відео
  },
  decoder: {
    readers: ["ean_reader", "code_128_reader"] // типи штрих-кодів
  }
}, function(err) {
  if (err) {
    console.error(err);
    return;
  }
  Quagga.start();
});

// Отримання результату сканування
Quagga.onDetected(result => {
  console.log("Зчитаний штрих-код:", result.codeResult.code);
});
```

Наведений код добре показує, що QuaggaJS дає змогу виконувати сканування безпосередньо з браузера, що робить її особливо корисною для веб-додатків у сфері торгівлі, інвентаризації, логістики чи складського обліку.

JetBrains WebStorm – це інтегроване середовище розробки, розроблене компанією JetBrains. Дане середовище надає засоби для розробки веб додатків, з використанням різних технологій: HTML, CSS, JavaScript, TypeScript, Node.js, Angular, PHP та багато інших [17]. Дане середовище розробки містить багато корисних функцій та засобів, які полегшують веб розробку, серед основних можна виділити авто-доповнення коду, перевірка синтаксису, дебагінг, підтримка систем контролю версій.

C# – це об’єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона була розроблена з метою поєднання високої продуктивності C++ і простоти Visual Basic. *ASP .NET Core* – це відкрита та безкоштовна платформа для веб розробки, забезпечує засоби для створення веб додатків, які можуть

запускатись на різних платформах. Фреймворк дозволяє розробникам використовувати різні мови програмування, такі як C#, F# та Visual Basic, а також підтримує різні технології, включаючи MVC, Web API.

Microsoft SQL Server – це реляційна система управління базами даних, розроблена компанією Microsoft, вона забезпечує можливості для зберігання, керування і маніпулювання даними. Дана СУБД використовує мову запитів SQL (Structured Query Language) для взаємодії з даними. MS SQL Server підтримує песимістичне блокування, що є важливим аспектом при виборі СУБД, адже розроблюваний застосунок «InventoEase» передбачає одночасну роботу багатьох користувачів [19].

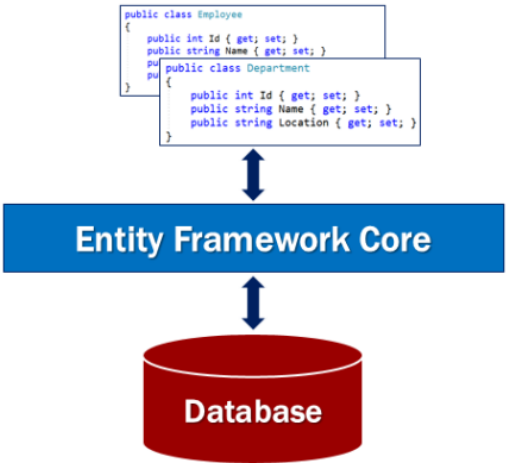

Рисунок 3.3 – Взаємодія мапера Entity Framework Core з базами даних

Entity Framework Core – це об'єктно реляційний мапер для .NET платформи, який дозволяє легко взаємодіяти з базами даних за допомогою .NET об'єктів. Дана бібліотека надає можливості використовувати різні провайдери баз даних, такі як SQL Server, MySQL, PostgreSQL, і забезпечує легкий доступ до даних через мову інтегрованих запитів .NET.

Microsoft Visual Studio – це інтегроване середовище розробки, розроблене компанією Microsoft, яке дозволяє розробникам створювати десктопні, мобільні та веб додатки. Дане середовище містить набір інструментів для розробки, тестування, відлагодження та розгортання програмного забезпеченням [9].

3.3. Структура та звязки елементів бази даних додатку

Інформаційна система «Інвентаризація даних матеріальних ресурсів» створена з метою підвищення ефективності управління товарними запасами підприємства. Ця система надає можливість додавати нові товари, керувати

ними, сканувати штрих-код для швидкого доступу до товару. Відслідковування покупок або продажів, переглядати стан інвентаря всього підприємства, розбивати товар на категорії, керувати місцезнаходженням товарів.

Ця система розроблена з використанням мов TypeScript, JavaScript та фреймворку Angular для фронт-енд частини, на серверній частині використовується ASP .NET Core фреймворк в поєднанні з Entity Framework Core та мовою програмування C#.

При створенні застосунку, виникла потреба зберігання інформації, тому спроектовано базу даних, яка працює на основі Microsoft SQL Server. Запропонована база даних містить наступні таблиці: AspNetUsers, Photos, UsersLocation, Locations, Categories, Tags, Items, ItemTags. Структура бази даних застосунку зображена на рис. 4.1.

Проведемо детальніший огляд кожної таблиці, перша таблиця AspNetUsers зображена на рис. 4.2.

Column Name	Data Type	Allow Nulls
Id	int	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
UserName	nvarchar(256)	☑
Email	nvarchar(256)	☑
NormalizedEmail	nvarchar(256)	☑
PasswordHash	nvarchar(MAX)	☑
SecurityStamp	nvarchar(MAX)	☑
ConcurrencyStamp	nvarchar(MAX)	☑
PhoneNumber	nvarchar(MAX)	☑
PhoneNumberConfirmed	bit	☐
TwoFactorEnabled	bit	☐
LockoutEnd	datetimeoffset(7)	☑
LockoutEnabled	bit	☐
AccessFailedCount	int	☐
RefreshToken	varchar(50)	☑
EmailConfirmed	bit	☐

Рисунок 3.6 – Елементи таблиці AspNetUsers

Інші складові бази даних та їх елементи наведені в Додатку Б.

Таблиця ItemTags це з’єднуюча таблиця для Items та Tags, оскільки один товар може містити багато тегів, і один тег може містити багато товарів, застосовується відношення багато до багатьох і створюється проміжна таблиця.

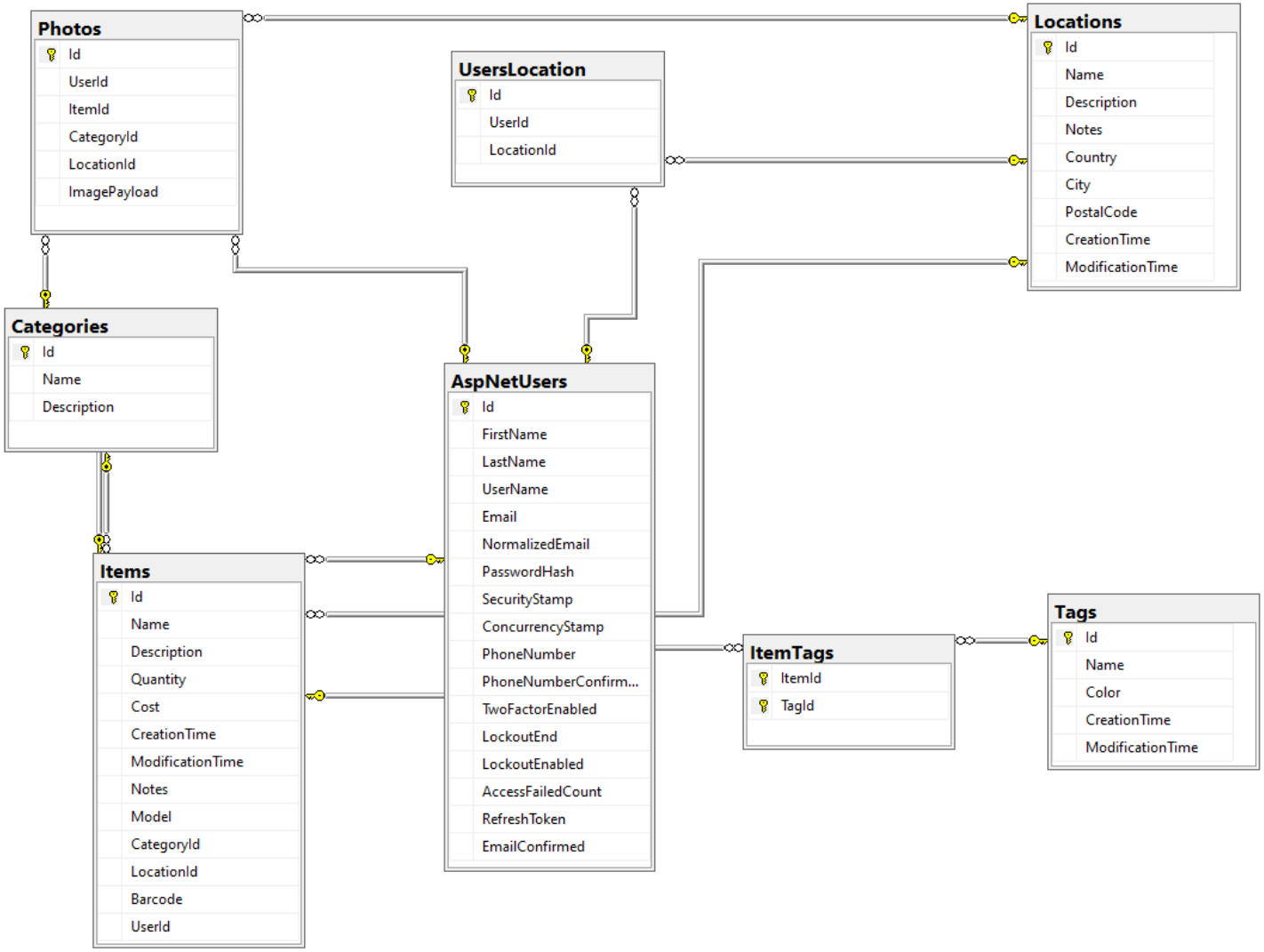

Рисунок 3.6. – Структура бази даних веб-додатку

Елементи таблиці Photos зображені на рис. 4.3.

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	UserId	int	☑
	ItemId	int	☑
	CategoryId	int	☑
	LocationId	int	☑
	ImagePayload	nvarchar(MAX)	☐

Рисунок 3.7 – Елементи *таблиці Photos*

Такий підхід є доцільним і ефективним, оскільки забезпечує збалансовану архітектуру між клієнтською та серверною частинами системи. Використання мов TypeScript і JavaScript у поєднанні з Angular створює гнучке та динамічне середовище для розробки інтерфейсу користувача, що дозволяє реалізувати сучасні принципи реактивності, модульності та повторного використання компонентів. Застосування ASP.NET Core на серверному боці гарантує високу продуктивність, масштабованість і безпеку обробки даних, а інтеграція з Entity Framework Core спрощує роботу з базою даних, забезпечуючи ефективне відображення об’єктів у реляційній моделі.

Використання Microsoft SQL Server як системи керування базами даних підсилює загальну надійність рішення, адже вона підтримує транзакційність, оптимізовані запити та механізми контролю доступу. Структура бази даних, що включає таблиці AspNetUsers, Photos, UsersLocation, Locations, Categories, Tags, Items та ItemTags, демонструє логічну організацію даних і забезпечує гнучкість у зберіганні, пошуку та обробці інформації.

Отже, поєднання зазначених технологій формує комплексний, технологічно зрілий підхід до створення веб-застосунку, який одночасно задовольняє вимоги зручності користувача, стабільності функціонування та ефективності управління даними.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ІНВЕНТАРИЗАЦІЇ ДАНИХ

4.1. Головні функціональні кроки із використання застосунку та особливості програмної реалізаці

На головній сторонці веб-застосунку, користувач може переглянути дані аналітики підприємства: товари, які закінчились, товари, які закінчуються, загальна кількість товарів, сума всіх товарів. На головній сторінці також відображаються найпопулярніші категорії товарів, їх рівень запасів.

Окрім того, користувач може перейти до сторінки зі всіма товарами, де в нього буде змога фільтрувати товари за категоріями, за локацією та за всіма товарними даними. Натиснувши на кнопку фільтру, користувач може створювати свої критерії за якими буде здійснюватися фільтрування та сортування товарів.

Натиснувши на товар, користувач може змінити інформацію про нього, наприклад змінити назву, кількість, місцезнаходження і т.п. Перейшовши на сторінку сканеру, використовуючи камеру, користувач з легкістю може відсканувати штрих-код товару, і отримати всю інформацію, якщо вона наявна в базі даних, якщо ні – система запропонує додати цей товар, заповнивши форму створення товару.

Коли підприємство отримує нову поставку товару, працівник з легкістю може оновити інформацію про кількість товарів в застосунку, відсканувавши штрих-код, і оновивши кількість.

Графічний інтерфейс системи, забезпечує користувачу декілька режимів перегляду товарів, що дає змогу швидше отримати потрібну інформацію. Для зручності редагування або видалення товарів, є можливість вибрати декілька товарів за допомогою функції мульти-вибору.

Щодо програмного коду веб-додатку то слід зазанчити, що його побудовано за типовою архітектурою Angular + TypeScript (Front-end) та

ASP.NET Core (Back-end) із взаємодією через HTTP API. Основну логіку взаємодії даних реалізовано у файлах DataService.ts та CommonService.ts, які відповідають за завантаження, обробку, фільтрацію й оновлення інформації у локальній базі даних (IndexedDB або локальний кеш) та синхронізацію з сервером.

Лістинг коду наведений в додатку В.

Наведемо деякі його особливості. Щодо DataService.ts то це централізоване сховище даних для веб-додатку, яке дозволяє швидко зберігати, читати, фільтрувати і підраховувати різні об’єкти, а також надає статистику та керує станом UI. Фактично, це “серце” додатку для роботи з моделями даних і колекціями:

DataService.ts

```
export class DataService {
public items: Map<string, Item> = new Map();
public categories: Map<string, Category> = new Map();
public photos: Map<string, Photo> = new Map();
public locations: Map<string, Location> = new Map();
public sales: Map<string, Sale> = new Map();
public purchases: Map<string, Purchase> = new Map();
public contacts: Map<string, Contact> = new Map();
public tags: Map<string, Tag> = new Map();
public images: Map<string, any> = new Map();
public filters: Filter[] = [];
public criteria: Criteria[] = [];
public groups: Map<string, GroupInterface> = new Map();
public actions: Map<string, ActionInterface> = new Map();
public purchasesTotal = 0;
public onlineTotal = 0;
public replacementTotal = 0;
public itemsTotal = 0;
public lowStockTotal = 0;
public outOfStockTotal = 0;
public locationData: any;
public categoriesData: Category[];
public purchasesData: any;
public salesData: any;
public populateDate = true;
constructor(
private commonService: CommonService,
private uiService: UiService,
private groupsService: GroupsService,
) {}
```

Зокрема, клас DataService містить колекції об’єктів у пам’яті, представлені структурами типу Map, які зберігають різні типи даних з ключами-рядками: 1) *items* – товари або основні елементи; 2) *categories* – категорії товарів; 3) *photos* та *images* – фотографії або медіафайли; 4) *locations* – місця або склади; 5) *sales* і *purchases* – продажі та покупки; 6) *contacts* – контакти користувачів або клієнтів; 7) *tags* – мітки, які можна прив’язувати до товарів чи інших об’єктів; 8) *groups* і *actions* – логічні групи та дії, які можна виконувати над об’єктами. Використання колекцій Map замість масивів забезпечує швидкий доступ до елементів за ключем (O(1)), що особливо ефективно при великій кількості об’єктів.

Клас також містить масиви, числові показники та допоміжні дані, необхідні для роботи застосунку: 1) *filters* і *criteria* – масиви, призначені для зберігання критеріїв фільтрації або пошуку, які застосовуються до даних; 2) *purchasesTotal, onlineTotal, replacementTotal, itemsTotal, lowStockTotal, outOfStockTotal* – числові змінні, що відображають статистичні показники, зокрема кількість товарів, покупок або позицій із низьким залишком; 3) *locationData, categoriesData, purchasesData, salesData* – дані, отримані із сервера або локальних джерел, що містять агреговану інформацію про об’єкти; 4) *populateDate* – логічний прапорець, який контролює автоматичне заповнення дат у записах; 5) через конструктор клас приймає три зовнішні сервіси: *commonService* – для виконання загальних утиліт або API-запитів, *uiService* – для керування інтерфейсом користувача, та *groupsService* – для роботи з групами об’єктів.

4.2. Результати застосування інформаційної системи інвентаризації даних

Для початку роботи з системою, користувачу потрібно авторизуватись, або зареєструватись, якщо він не зробив цього раніше. Перший етап заповнення

форми реєстрації наведено в рис. 4.1.

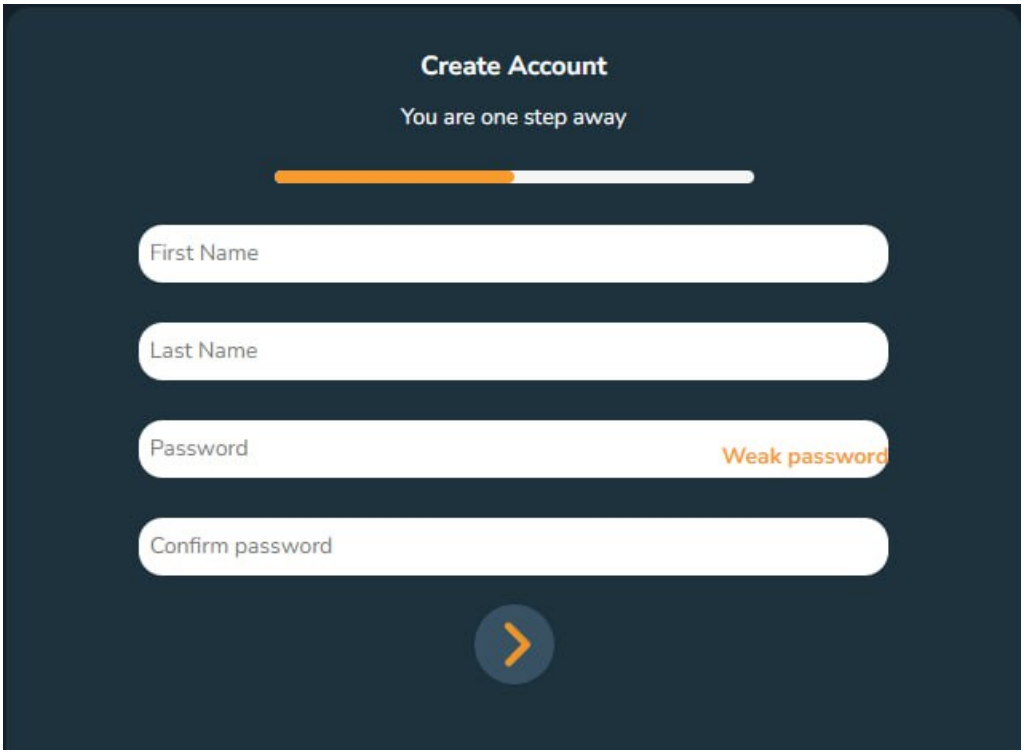

Рисунок 4.1 – Реєстрація у веб-застосунку та створення акаунту

Зазвичай процес реєстрації передбачає введення імені та прізвища користувача, після чого потрібно створити пароль. Він має складатися щонайменше з восьми символів і містити цифри, спеціальні знаки, а також літери у верхньому та нижньому регістрах.

Після введення й повторного підтвердження пароля система перевіряє коректність даних спершу на клієнтській стороні, а потім — на сервері. Такий підхід забезпечує надійність перед збереженням інформації й мінімізує ризик її підробки шляхом перехоплення або зміни мережевого трафіку.

Якщо всі умови дотримані, користувач переходить до наступного етапу — внесення додаткових даних, таких як країна, місто та електронна адреса. Після підтвердження форми реєстрації відомості про користувача зберігаються у базі даних, і система автоматично перенаправляє його на сторінку входу в систему. Якщо користувач вірно авторизується то надалі він потрапляє на головну сторінку застосунку (рис. 4.2).

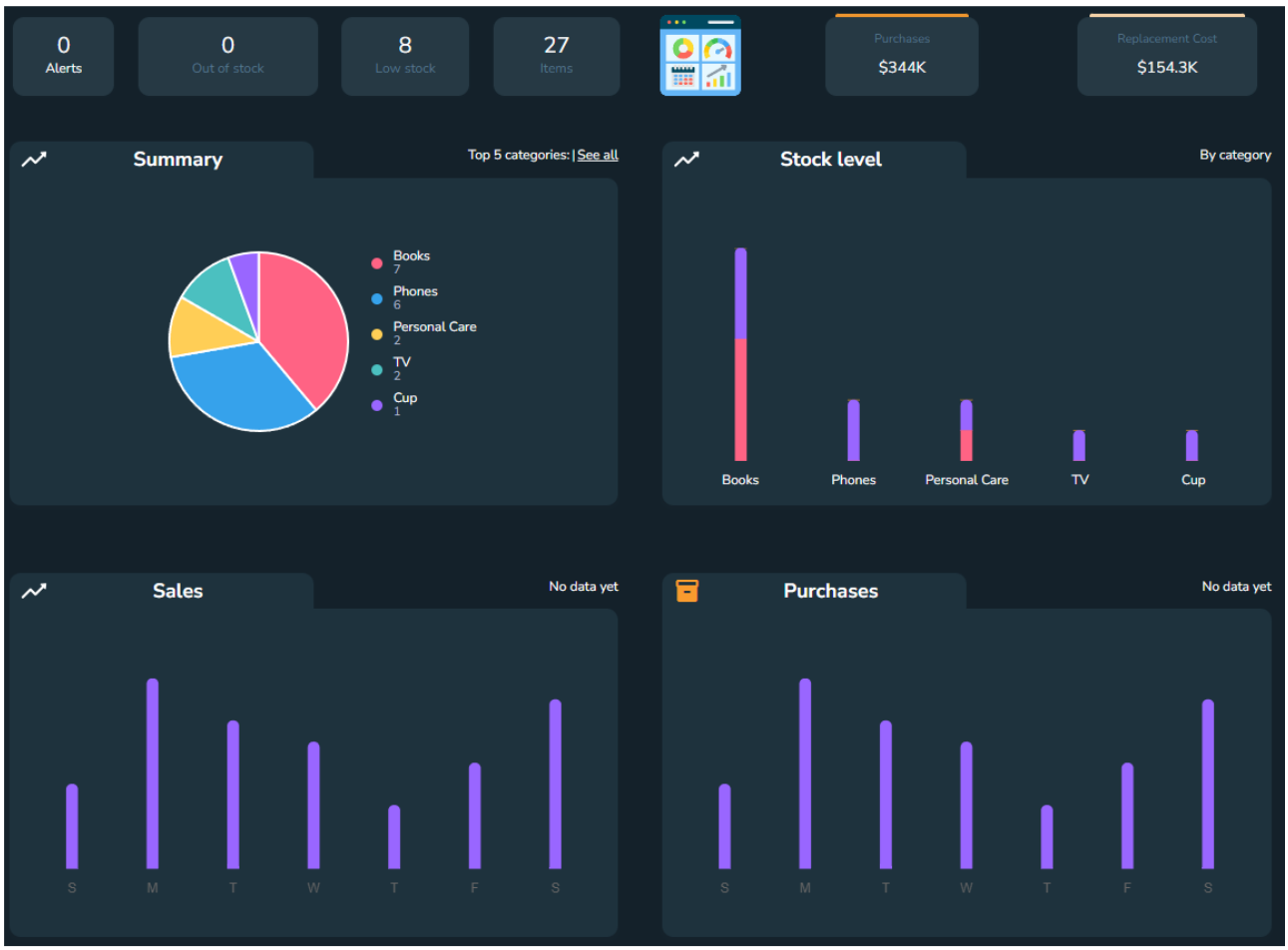

Рисунок 4.2 – Головна сторінка застосунку з даними

На головній сторінці вебзастосунку відображається зведена інформація про стан товарних запасів: кількість позицій, що вже закінчилися, кількість товарів із низьким залишком, а також загальна кількість усіх товарів і їх сумарна вартість. У центральній частині сторінки розміщено аналітичний блок із діаграмами різних типів. Перша діаграма демонструє п’ять найпопулярніших категорій товарів, наступна — кількість запасів у межах кожної з цих категорій. Окрема діаграма продажів ілюструє щоденну статистику реалізації протягом тижня, а діаграма покупок відображає аналогічні дані щодо надходжень. Після натискання на показники, що стосуються товарів із малими залишками або тих, що повністю відсутні, система автоматично формує перелік відповідних позицій, які відповідають вибраному критерію (рис. 4.3).

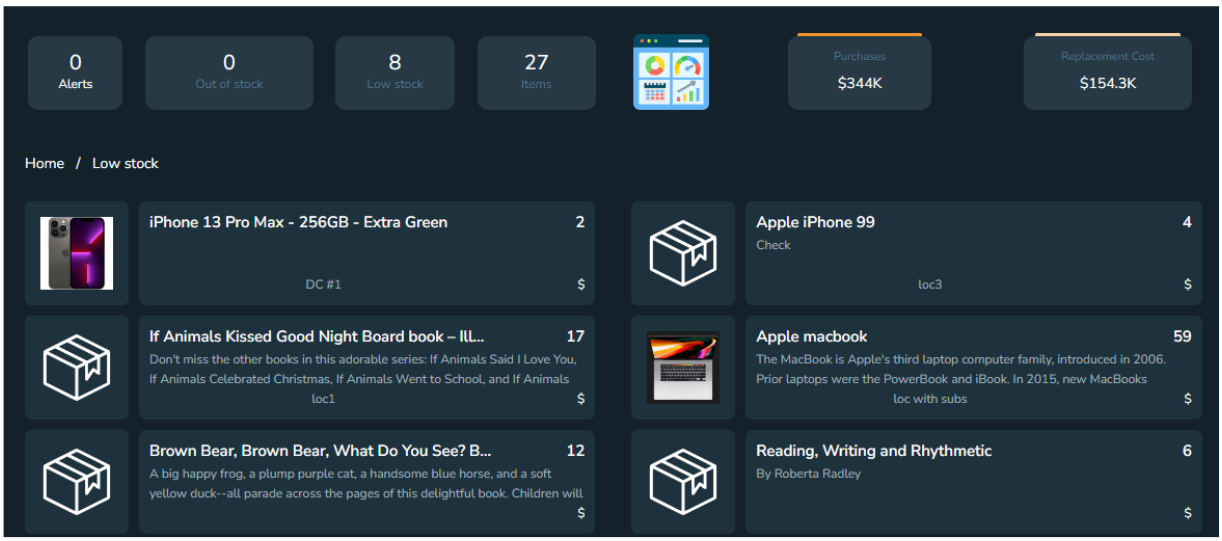

Рисунок 4.3 – Рух товарів та їх наявність на складі

Наступною, після домашньої сторінки є загальна сторінка всіх колекцій: товари, категорії, локації, теги (рис. 4.4).

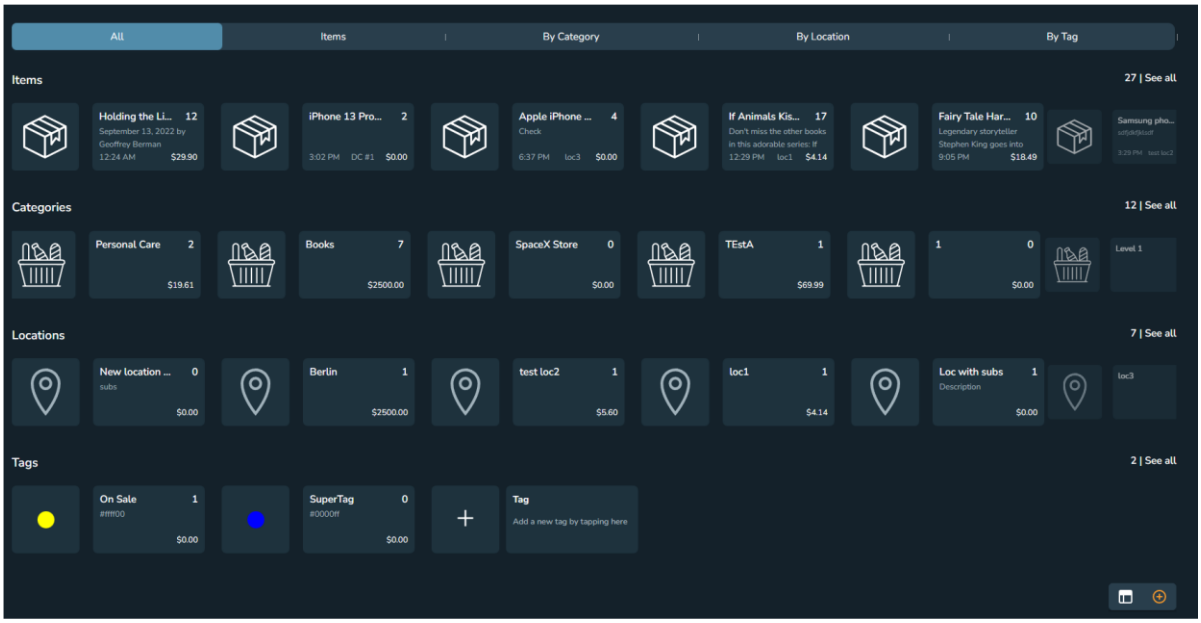

Рисунок 4.4 – Сторінка списків усіх колекцій товарів підприємства

Як можна побачити, на сторінці представлено повний перелік товарів, категорій, локацій і тегів, причому у правій частині інтерфейсу вказано їхню кількість. За допомогою панелі навігації користувач може перейти безпосередньо до сторінки, де відображаються лише товари.

Після вибору кількох позицій набір кнопок керування змінюється, надаючи можливість редагування або видалення обраних елементів. Для

зручності пошуку реалізовано функцію фільтрації за назвою товару: у поле пошуку можна ввести потрібне слово, і система автоматично відобразить лише ті товари, у назві яких воно присутнє.

Для розширених варіантів пошуку передбачене додаткове меню, що відкривається натисканням другої кнопки керування. Однією з ключових можливостей сторінки є створення нових товарів — ця дія виконується через останню кнопку керування, натискання якої відкриває форму для додавання нового запису (рис. 4.5).

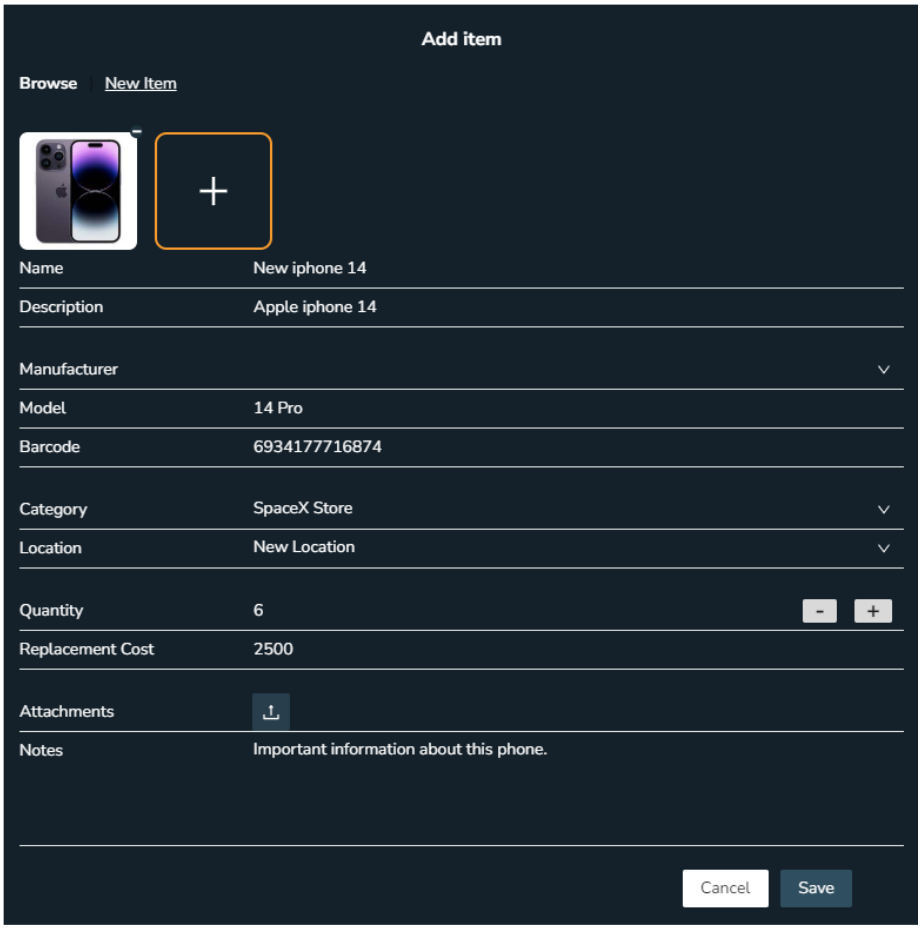

Рисунок 4.5 – Форма для заповлення даних про товар

За допомогою цієї форми користувач має можливість додати кілька фотографій для одного товару. Для того щоб товар з’явився у системі, достатньо вказати його назву. Якщо необхідно надати більш детальну інформацію, передбачено поля для моделі товару, штрих-коду (який пізніше

використовується сканером), категорії, локації та кількості одиниць тощо. Після збереження всі введені дані додаються до системи.

Для зручного маркування товарів реалізовано окрему сторінку з тегами, яка структурно схожа на сторінки категорій і локацій, але виконує іншу функцію. При створенні нового тегу користувач може вручну вибрати його колір або обрати із списку найчастіше використовуваних варіантів.

Ще однією зручністю для пошуку товарів є використання сканера. На відповідній сторінці користувач може відсканувати штрих-код товару за допомогою камери смартфона, щоб знайти його у базі даних. Якщо камера недоступна, передбачена можливість ручного введення штрих-коду, після чого система в режимі реального часу відображає всі товари, які відповідають введеному значенню (рис. 4.6).

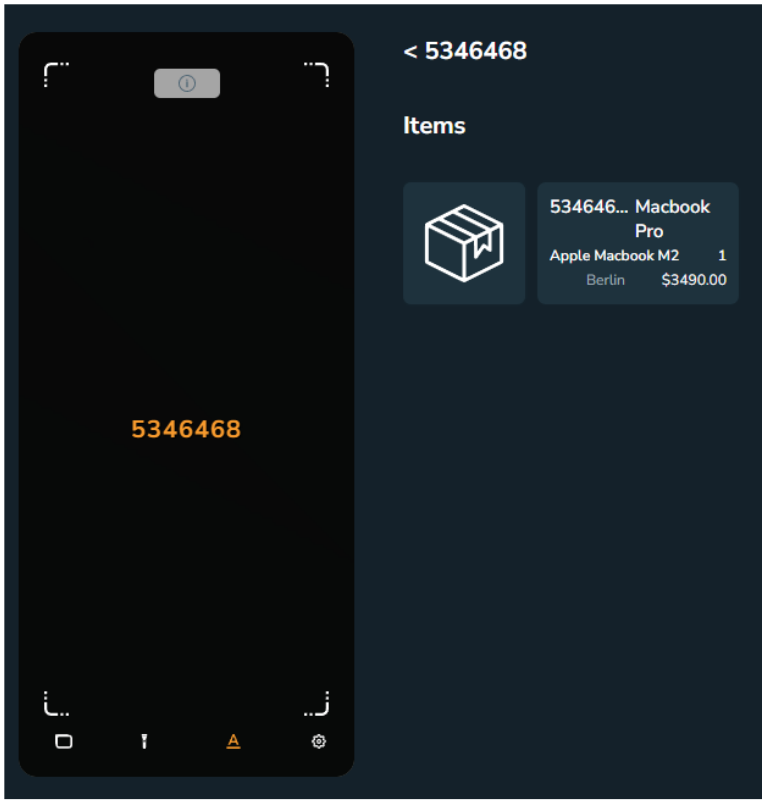

Рисунок 4.6 – Спосіб ручного введення штрих-коду в сканері

Після вибору конкретного товару відкривається вікно його перегляду, звідки користувач може перейти до редагування даних. Після сканування товару повторне відкриття сканера дозволяє переглянути останні відсканові позиції.

Таким чином, реалізовано інформаційну систему для управління інвентаризацією малого підприємства. Було спроектовано базу даних, описано класи для вирішення поставлених завдань та наведено контрольний приклад, який підтверджує коректну роботу системи.

4.3. Дослідження продуктивності роботи інформаційної системи інвентаризації даних

Дослідження та оптимізацію застосунку проведено через аналіз продуктивності (профілювання запитів, кешування даних, оптимізацію індексів у базі), а також мінімізацію обсягу переданих даних між клієнтом і сервером. Для зручності користування застосовано UX-тестування з реальними користувачами. Безпеку перевірено через аудит автентифікації, шифрування даних, захист від SQL-ін’єкцій, XSS та CSRF-атак.

Отже, для дослідження продуктивність системи у реальних умовах експлуатації під час тестування використано такі підходи: 1) тестування часу відгуку (Response Time); 2) вимірювання часу запису та читання з IndexedDB; 3) тестування коректності обробки даних; 4) оцінка зручності використання (Usability Testing); 5) безпекові аспекти.

Зокрема, нами виконано дослідження продуктивності та тестування часу відгуку (Response Time). Яке включало навантажувальне тестування – основна ідея полягає у симулюванні одночасні запитів до веб-додатку і вимірюванні часу відповіді.

Використано інструменти: 1) *JMeter* — класичний інструмент для тестування навантаження, дозволяє задавати кількість паралельних користувачів, повтори та зберігати час відповіді; 2) *Gatling* — сучасний, зручний для сценаріїв з різною інтенсивністю запитів; 3) *Locust (Python*) — дозволяє писати скрипти на Python, легко масштабувати тестування, відстежувати час відповіді у реальному часі.

Сценарій тестування передбачав: 1) задати поступове збільшення одночасних запитів: 1, 2, 3 … 10-20; 2) для кожного рівня паралельних запитів виміряти середній час відповіді; 3) зібрати дані: кількість запитів – середній час відповіді.

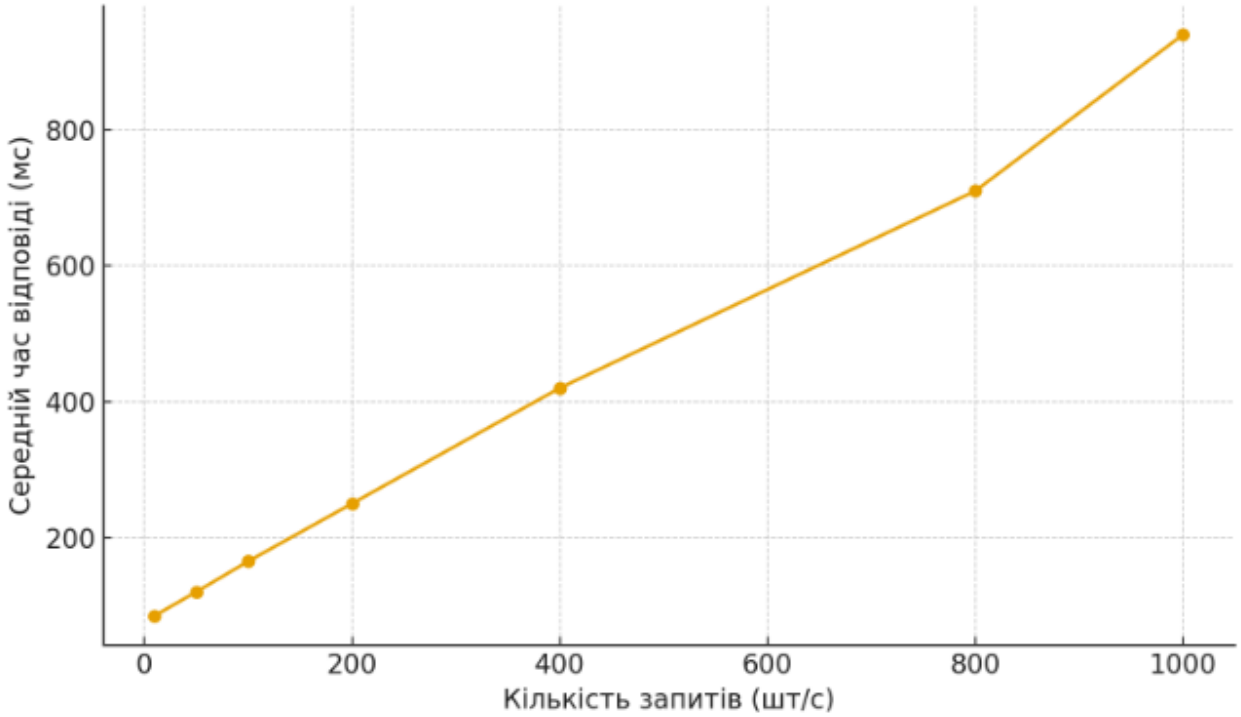

Рисунок 4.7 – Залежність часу відповіді системи від кількості запитів

Відповідно до отриманх даних рис. 4.7, залежність є нелінійною: до ≈200 запитів/с час росте повільно, система зберігає високу ефективність. Далі – різке зростання затримки, що свідчить про вичерпання можливостей асинхронної обробки потоків у RxJS (особливо при використанні forkJoin), а також про перевантаження браузерної пам’яті при розгортанні великих колекцій Map. Типова поведінка для SPA-застосунків з клієнтським кешуванням у IndexedDB.

Для встановлення звязку між використанням пам’яті та кількістю об’єктів виконано моніторинг пам’яті *JVM* / *Node.js* / іншого середовища: 1) для *Java* (*Spring*, *Tomcat*, інші) - *VisualVM* або *JProfiler* для аналізу об’єктів у пам’яті; jconsole або *Java Mission Control* для відстеження *heap usage*; 2) для *Node.js* - *Node.js --inspect*, *Chrome DevTools Memory tab*; бібліотеки heapdump або *memwatch*-next для збору статистики об’єктів.

Сценарій збору даних: 1) створити скрипт або тест, який поступово додає об’єкти у Map або інші колекції; 2) для кожного кроку вимірювати: кількість створених об’єктів, загальне використання пам’яті (heap).

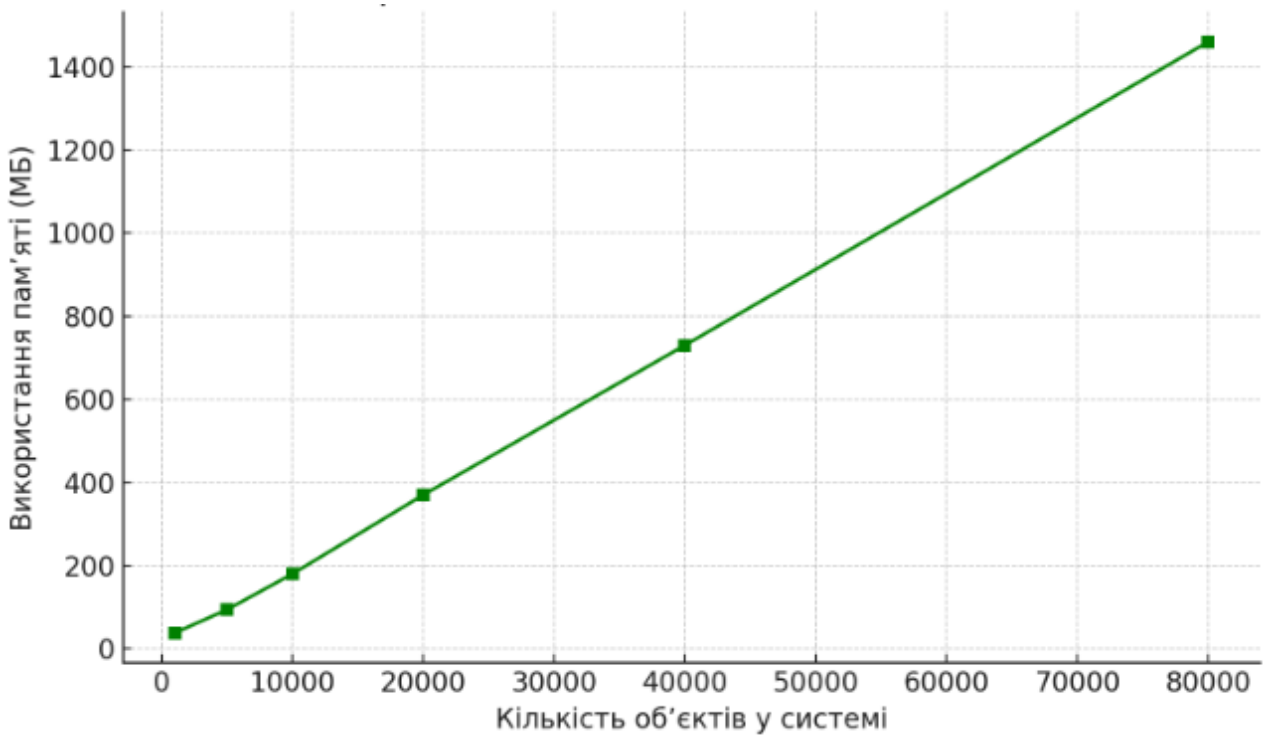

Рисунок 4.8 – Залежність використання пам’яті від кількісті об’єктів

Відповідно до отриманх даних рис. 4.8, пам’ять зростає майже лінійно, оскільки кожен об’єкт Item, Sale або Category має вкладені властивості (включно з посиланнями на інші сутності). Особливу роль відіграє функція setGroups(), де через підписку (subscribe) утворюються замикання, що можуть утримувати посилання на об’єкти після завершення ініціалізації, збільшуючи споживання пам’яті (ефект “витоків”).

РОЗДІЛ 5
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1. Розробка логіко-імітаційної моделі виникнення травм і аварій

Методикою оцінки рівня небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію рівня небезпеки для конкретного об'єкта [6]. Таким показником вибрана ймовірність виникнення аварії, травми залежно від явища, що досліджується.

Для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми в процесі створення мікрокліматичних умов у приміщенні оцінюють відповідні небезпечні події. Кожній із них присвоїмо ймовірність виникнення:

Шифр	Назва події	Ймовірність
P_1	Відсутність захисного заземлення	0,02
P_2	Пошкодження захисного заземлення	0,04
P_3	Спрацювання складових захисту	0,1
P_4	Неправильна експлуатація захисту	0,02
P_5	Відсутність профілактичних заходів	0,2
P_6	Відсутність захисного щита	0,12
P_7	Недотримання правил вибору взуття	0,15
P_8	Незнання правил техніки безпеки	0,1
P_9	Відсутність засобів індивідуального захисту	0,2
P_{10}	Легковажність	0,08

На основі наведених подій будуємо матрицю логічних взаємозв'язків між окремими пунктами, графічна інтерпретація якої зображено на рис. 5.1.

Розрахуємо ймовірності виникнення подій, що формують логіко-імітаційну модель процесів створення мікрокліматичних умов. Розглянемо травмонебезпечну ситуацію, що виникає за умови роботи працівників із

електронебезпекою.

Підставивши дані ймовірностей базових подій у формулу, отримаємо ймовірність події 13: $P_{13} = 0{,}2 + 0{,}4 - 0{,}2 \cdot 0{,}4 = 0{,}0592$.

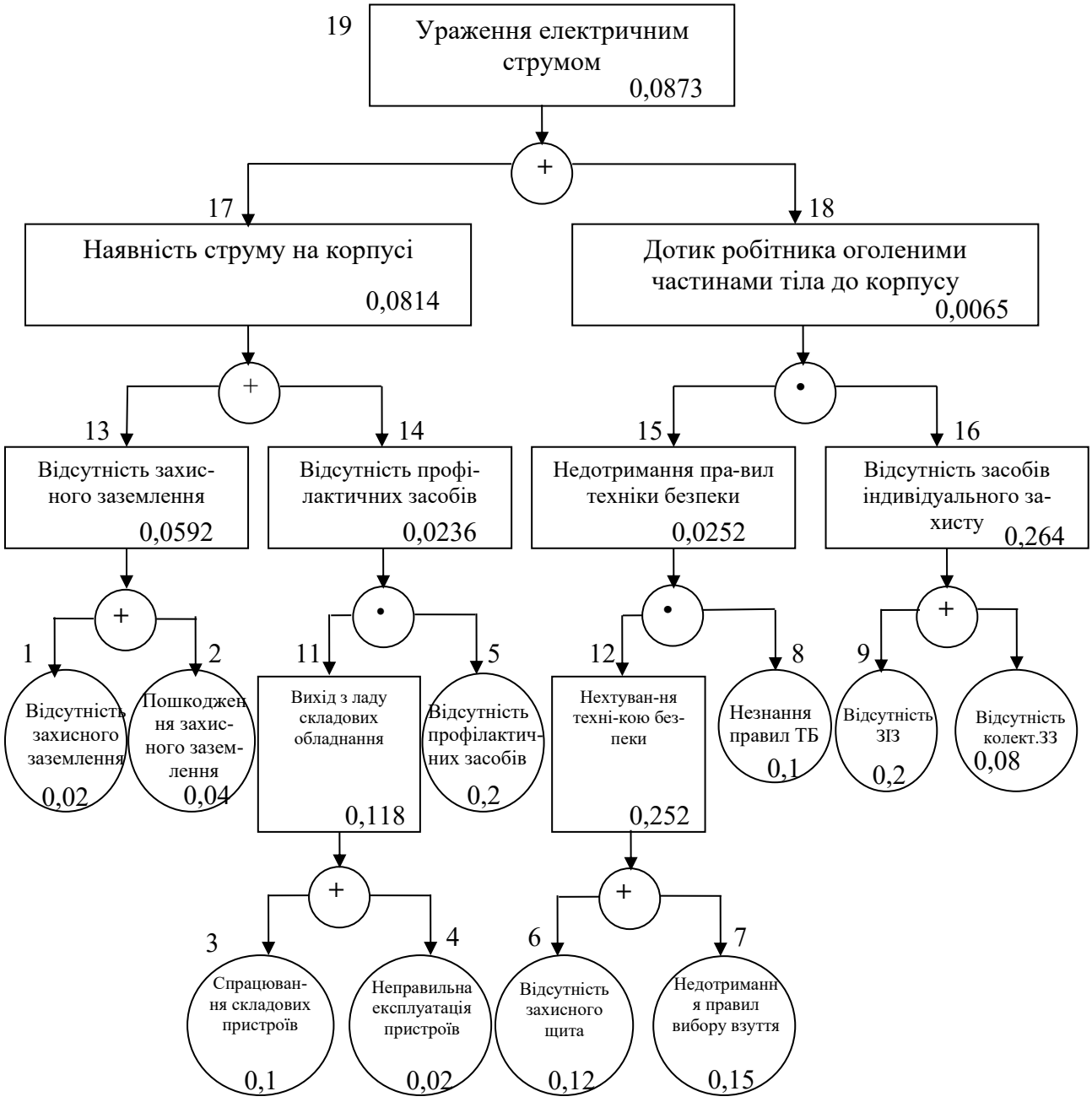

Рис. 5.1. Матриця логічних взаємозв'язків між окремими подіями травмонебезпечної ситуації [6]

Аналогічно визначаємо ймовірність інших подій:

$$P_{11} = P_4 + P_5 - P_4P_5 = 0{,}3 + 0{,}4 - 0{,}3 \cdot 0{,}4 = 0{,}118.$$
$$P_{12} = P_6 + P_7 - P_6P_7 = 0{,}3 + 0{,}5 - 0{,}3 \cdot 0{,}5 = 0{,}252.$$
$$P_{16} = P_9 + P_{10} - P_9P_{10} = 0{,}2 + 0{,}15 - 0{,}2 \cdot 0{,}15 = 0{,}264.$$

$$P_{14} = P_{11} \cdot P_5 = 0{,}118 \cdot 0{,}2 = 0{,}0236.$$
$$P_{15} = P_{12} \cdot P_8 = 0{,}252 \cdot 0{,}1 = 0{,}0252.$$
$$P_{17} = P_{13} + P_{14} - P_{13} \cdot P_{14} = 0{,}592 + 0{,}0236 - 0{,}0592 \cdot 0{,}0236 = 0{,}0814.$$
$$P_{18} = P_{15} \cdot P_{16} = 0{,}264 \cdot 0{,}0252 = 0{,}0065.$$
$$P_{19} = P_{17} + P_{18} - P_{17} \cdot P_{18} = 0{,}0065 + 0{,}0814 - 0{,}0065 \cdot 0{,}0814 = 0{,}0873.$$

Таким чином, ймовірність перекидання машини та наслідкового виникнення травми працівника є досить мала і становить – $P_{19} = 0,0873$.

5.2. Планування заходів із покращення умов праці

До заходів щодо покращення умов праці належать всі види діяльності, спрямовані на попередження, нейтралізацію або зменшення негативної дії шкідливих і небезпечних виробничих факторів на працівників.

Рівень умов праці оцінюють порівнянням за фактичними і нормативними значеннями узагальнених (групових) показників.

Заходи щодо поліпшення умов праці здійснюють з метою створення безпечних умов праці шляхом:

- доведення до нормативного рівня показників виробничого середовища за елементами умов праці;
- захисту працівників від дії небезпечних і шкідливих виробничих факторів.

До показників ефективності заходів щодо поліпшення умов праці належать:

а) зміни стану умов праці:

- зміна кількості засобів виробництва, приведених у відповідність до вимог стандартів безпеки праці;
- покращання санітарно-гігієнічних показників;
- покращання психофізичних показників, зменшення фізичних і нервово-психічних навантажень, в т.ч. монотонних умов праці;

б) соціальні результати заходів:

- збільшення кількості робочих місць, що відповідають нормативним вимогам;
- зниження рівня виробничого травматизму;
- престиж та задоволення працею.

Отже, на покращення охорони праці потрібно виділити кошти на відновлення вентиляційних систем у ремонтних майстернях, естетично оформити приміщення офісу, відновити кабінет з охорони праці, поновити протипожежний інвентар.

5.3. Безпека в надзвичайних ситуаціях

Актуальність проблеми природно-техногенної безпеки для населення і території, зумовлена зростанням втрат людей, що спричиняється небезпечними природними явищами, промисловими аваріями та катастрофами [6]. У системі цивільної оборони окремого господарства необхідно забезпечити захист населення таким чином:

Укриття в захисних спорудах, якому підлягає усе населення відповідно до приналежності, досягається створенням фонду захисних споруд.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення його у позаміській зоні.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

ВИСНОВКИ І РЕКОМЕНДАЦІЇ

Ефективне управління інвентаризацією IT-систем є надзвичайно важливим для будь-якої організації, оскільки воно забезпечує підтримку оптимальної продуктивності, безпеки та стабільності мережі. Водночас, без належного вибору програмного забезпечення процес інвентаризації може стати складним і трудомістким завданням, що ускладнює контроль за ресурсами та своєчасне виявлення проблем у системі.

Виконання завдань теоретичного розділу скероване на комплексну роботу з системного аналізу об’єкта дослідження. Зокрема, створено дерево цілей, яке наочно відображає ієрархічні взаємозв’язки між загальною метою та засобами її досягнення. Визначено генеральну мету системи та сформульовано підцілі, що дозволило конкретизувати завдання і напрями функціонування системи. На основі проведеної якісної характеристики об’єкта було обрано відповідний тип інформаційної системи, який найбільше відповідає поставленим вимогам.

Для детального опису роботи системи були спроектовані DFD-діаграми, які показують, як дані рухаються всередині системи та як вони обробляються різними процесами. Це дозволяє наочно побачити потоки інформації та логіку взаємодії компонентів. Крім того, для відображення взаємозв’язків між сутностями бази даних була побудована ER-діаграма, що забезпечує структуроване уявлення про дані та їхні взаємозв’язки в системі. Проведена робота забезпечила повне розуміння архітектури об’єкта дослідження та основних процесів його функціонування.

Реалізований підхід є доцільним і ефективним, оскільки забезпечує збалансовану архітектуру між клієнтською та серверною частинами системи. Використання мов TypeScript і JavaScript у поєднанні з Angular створює гнучке та динамічне середовище для розробки інтерфейсу користувача, що дозволяє реалізувати сучасні принципи реактивності, модульності та повторного використання компонентів.

Застосування ASP.NET Core на серверному боці гарантує високу продуктивність, масштабованість і безпеку обробки даних, а інтеграція з Entity Framework Core спрощує роботу з базою даних, забезпечуючи ефективне відображення об’єктів у реляційній моделі.

Використання Microsoft SQL Server як системи керування базами даних підсилює загальну надійність рішення, адже вона підтримує транзакційність, оптимізовані запити та механізми контролю доступу. Структура бази даних, що включає таблиці AspNetUsers, Photos, UsersLocation, Locations, Categories, Tags, Items та ItemTags, демонструє логічну організацію даних і забезпечує гнучкість у зберіганні, пошуку та обробці інформації.

Реалізовано інформаційну систему для управління інвентаризацією малого підприємства. Спроектовано базу даних, описано класи для вирішення поставлених завдань та наведено контрольний приклад, який підтверджує коректну роботу системи.

Загалом, аналіз коду та результати дослідження продуктивності свідчать, що система побудована грамотно й орієнтована на стабільну роботу в реальному середовищі, але для промислової експлуатації доцільно додати механізми оптимізації кешу, контролю автентифікації та обмеження обсягу одночасних запитів.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Види штрих-кодів, які зчитує сканер. Torgsoft : веб-сайт. URL: https://torgsoft.ua/support/faq/barcode-scanner/vydy-shtryhkodiv/ (дата звернення: 16.10.2025).

2. Віктор Галочка «TypeScript: від популярності до стандарту розробки». GenTech: веб-сайт. URL: https://www.gen.tech/post/typescript-vid-hajpu-do-standartu-rozrobki (дата звернення: 24.10.2025).

3. Дональд Уотерс. «Inventory Control and Management», 2nd Ed. -Wiley. 2018. 63-67 с.

4. Едвард Фразель. «Inventory Strategy: Maximizing Financial, Service and Operations Performance with Inventory Strategy». 2015. 23-34 с.

5. Комплексна система IT-рішень для управління агробізнесом. URL: https://agrichain.com.ua/ (дата звернення: 20.11.2025).

6. Лехман С.Д. та ін. Запобігання аварійності і травматизму у сільському господарстві / С.Д. Лехман, В.І. Рубльов, Б.І. Рябцев. К.: Урожай, 1993. 272 с.

7. Райс Ерік. Інформаційно архітектурний підхід до створення успішних веб-сайтів. [Текст] / Ерік Райс. Addison Wesley, 20020. 266 с.

8. Платформа .NET URL: https://metanit.com/sharp/mvc5/(дата звернення: 30.11.2024).

9. Сайт Metanit all about C#. URL: https://metanit.com/sharp/general.php (дата звернення: 30.11.2024).

10. Середовище розробки Microsoft Visual Studio. Informatics : веб-сайт. URL: https://informatics.in.ua/programming_csharp/part_01.php (дата звернення: 24.10.2025).

11. Сканер штрих-коду: принцип роботи. Pik : веб-сайт. URL: http://pik.cn.ua/37953/skaneri-dlya-shtrih-kodiv-printsip-roboti-i-osnovni-vidi/ (дата звернення: 17.11.2025).

12. Хеслоп П. HTML з самого початку. К: Препринт, 2014. 450с.

13. Що таке CSS. URL: http://phpist.com.ua/css/5-whatcss. (дата звернення: 30.11.2024).

14. Angular додатки: Основи. Angular : веб-сайт. URL: https://angular.io/guide/what-is-angular (дата звернення: 24.10.2025).

15. Google Drive Api .NET Quickstart Complete the steps described in the rest of this page to create a simple .NET console application that makes requests to the Drive API. URL: https://developers.google.com/drive/api/v3/quickstart/dotnet . (дата звернення: 30.11.2024).

16. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language 7th Edition, David Flanagan. 2017. 306 с.

17. JetBrains WebStorm. JetBrains : веб-сайт. URL: https://www.jetbrains.com/webstorm/ (дата звернення: 24.10.2025).

18. QuaggaJS Scanner. HacksMozilla : веб-сайт. URL: https://hacks.mozilla.org/2014/12/quaggajs-building-a-barcode-scanner-for-the-web/ (дата звернення: 24.10.2025).

19. Record locking in SQL Server. RelationalDbDesign : веб-сайт. URL: https://www.relationaldbdesign.com/enterprise-business-rules/module2/optimisticpessimistic-record-locking.php (дата звернення: 24.10.2025).

20. Tutorial for create API with ASP.NET MVC. URL: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvcapp/?view=aspnetcore-3.1. (дата звернення: 30.11.2024).

21. What is meant by Data Inventory? URL: https://www.privacyengine.io/resources/glossary/data-inventory/

ДОДАТКИ

Додаток А.

Діаграми декомпозицій процесів створення ІС

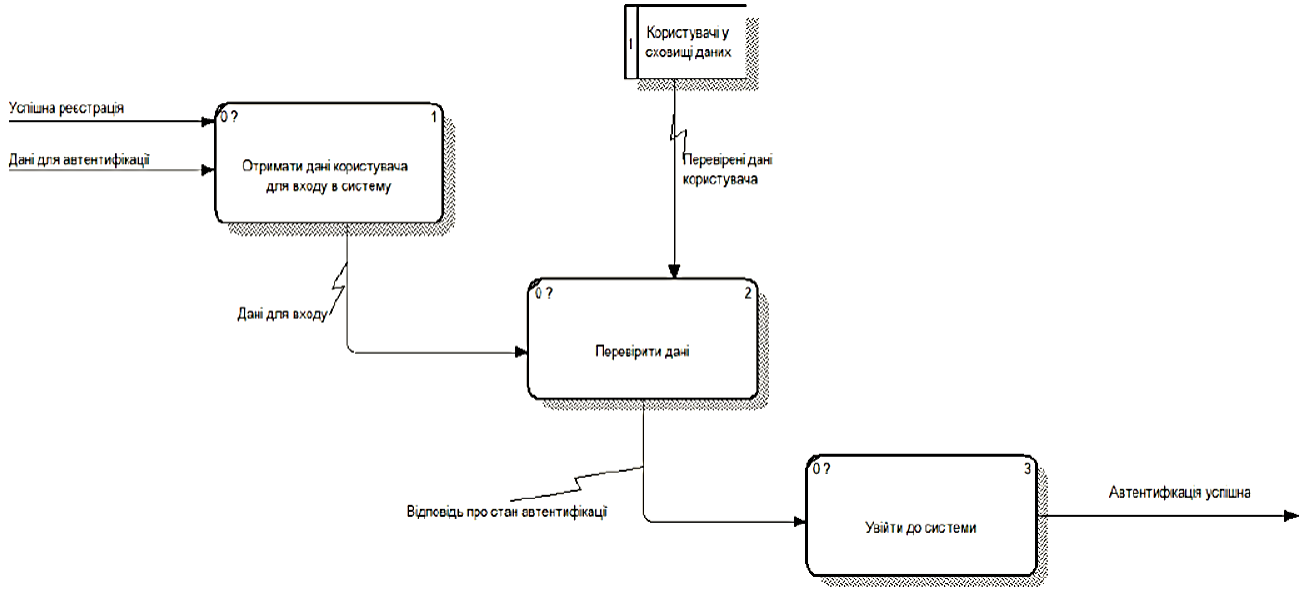

Рисунок А.1 – Діаграма декомпозиції процесів «Автентифікувати користувача»

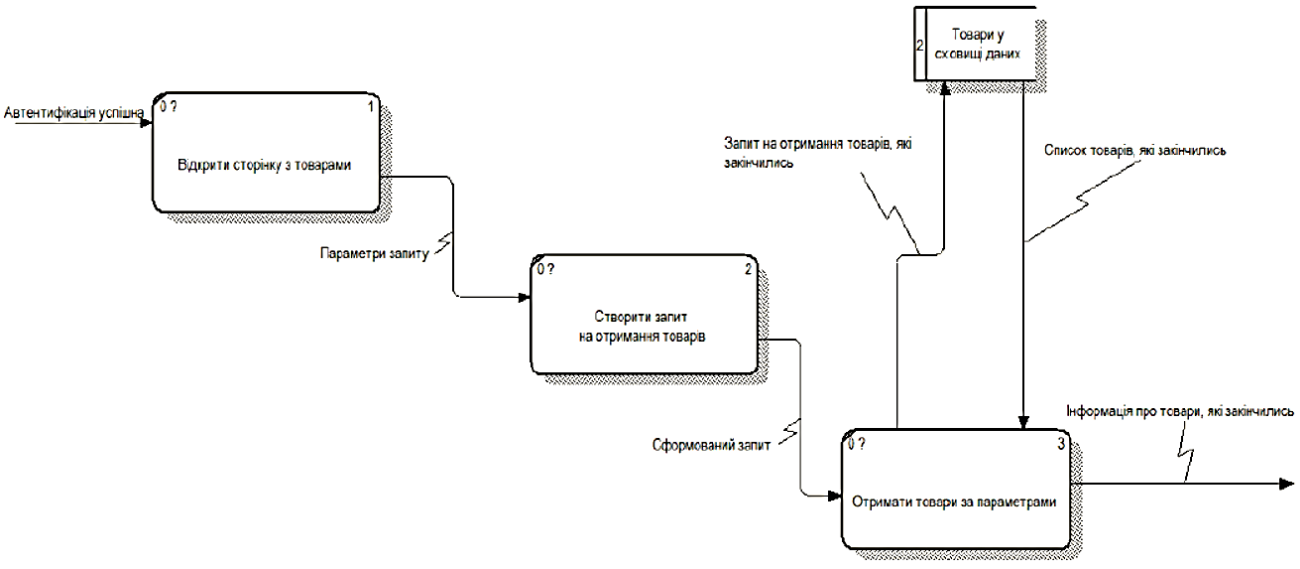

Рисунок А.2 – Діаграма декомпозиції процесів «Отримати товари, які закінчились»

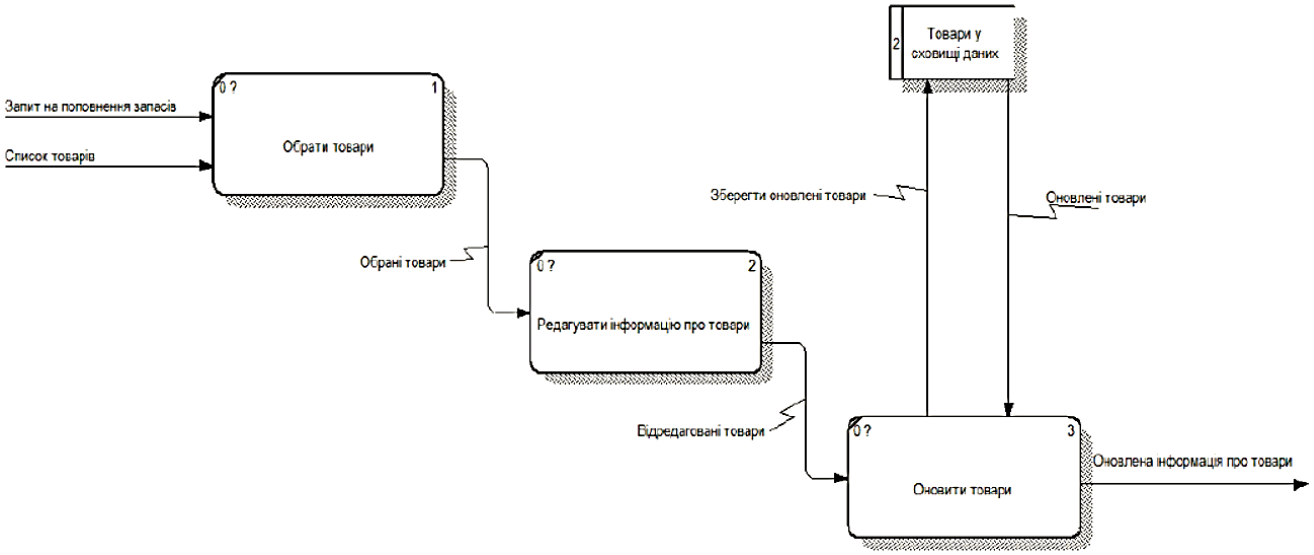

Рисунок А.3 – Діаграма декомпозиції процесів «Оновити товарний запас»

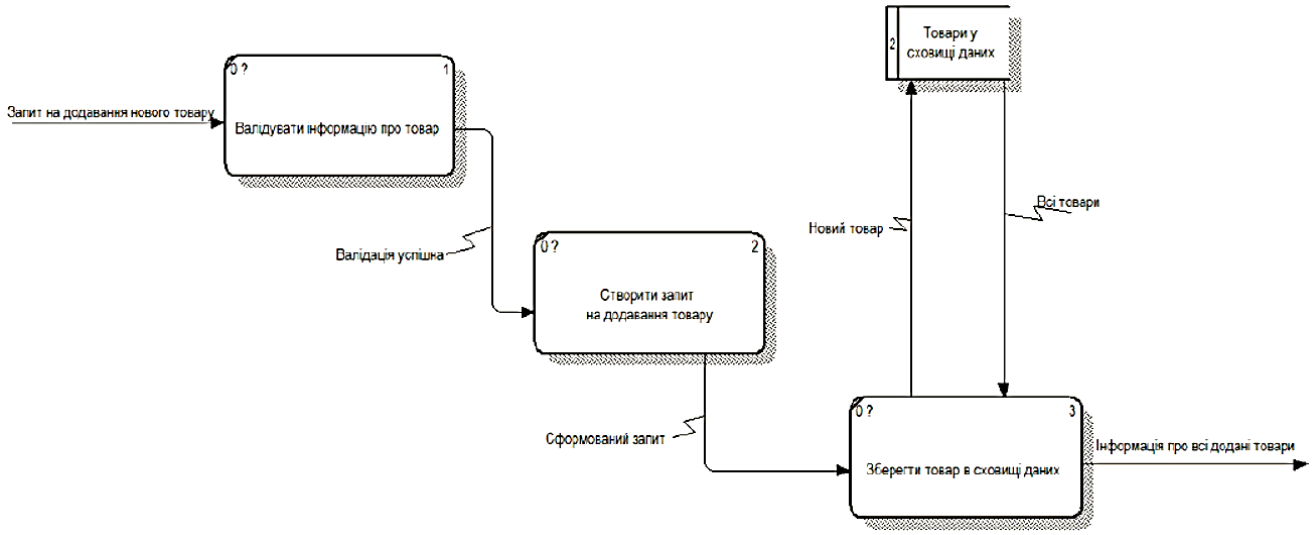

Рисунок А.4 – Діаграма декомпозиції процесів «Додати товар»

Додаток Б.

Елементи таблиць, що формують базу даних

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Name	nvarchar(50)	☐
	Description	nvarchar(255)	☑

Рисунок Б.1 – Елементи таблиці Categories

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Name	nvarchar(50)	☑
	Description	nvarchar(255)	☑
	Notes	nvarchar(255)	☑
	Country	nvarchar(50)	☑
	City	nvarchar(50)	☑
	PostalCode	nvarchar(10)	☑
	CreationTime	datetime	☑
	ModificationTime	datetime	☑

Рисунок Б.2 – Елементи таблиці Locations

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Name	nvarchar(256)	☐
	Description	nvarchar(256)	☑
	Quantity	int	☑
	Cost	money	☑
	CreationTime	datetime	☐
	ModificationTime	datetime	☑
	Notes	nvarchar(MAX)	☑
	Model	nvarchar(50)	☑
	CategoryId	int	☑
	LocationId	int	☑
	Barcode	nvarchar(50)	☑
	UserId	int	☐

Рисунок Б.3 – Елементи таблиці Items

Додаток В.

Фрагменти коду реалізації ІС управління інвентаризацією даних підприємства

DataService.ts

```
export class DataService {
public items: Map<string, Item> = new Map();
public categories: Map<string, Category> = new Map();
public photos: Map<string, Photo> = new Map();
public locations: Map<string, Location> = new Map();
public sales: Map<string, Sale> = new Map();
public purchases: Map<string, Purchase> = new Map();
public contacts: Map<string, Contact> = new Map();
public tags: Map<string, Tag> = new Map();
public images: Map<string, any> = new Map();
public filters: Filter[] = [];
public criteria: Criteria[] = [];
public groups: Map<string, GroupInterface> = new Map();
public actions: Map<string, ActionInterface> = new Map();
public purchasesTotal = 0;
public onlineTotal = 0;
public replacementTotal = 0;
public itemsTotal = 0;
public lowStockTotal = 0;
public outOfStockTotal = 0;
public locationData: any;
public categoriesData: Category[];
public purchasesData: any;
public salesData: any;
public populateDate = true;
constructor(
private commonService: CommonService,
private uiService: UiService,
private groupsService: GroupsService,
) {}
public loadData(page: string, hasToRefresh = false):
Observable<boolean> {
console.log("DB ITEMS LOAD DATA CALLED ");
const entitiesData$: any = {};
const entities = ENTITIES;
// if (page.includes(Page.Browser)) {
// entities = ENTITIES;
// }
entities.forEach((entity: Entity) => {
// if (hasToRefresh) {
// const request =
this.commonService.getAllItemsFromSection(entity);
// entitiesData$[entity] = request;
// } else {
const entityDB: Promise<any> = db[entity].toArray();
entitiesData$[entity] = from(entityDB);
// }
});
entitiesData$.filters = from(db.filters.toArray());
entitiesData$.criteria = from(db.filterCriteria.toArray());
entitiesData$.images = from(db.images.toArray());
forkJoin(entitiesData$).subscribe(
(data: any) => {
const locations = {};
const purchases = {};
const sales = {};
this.setItems(data.items);
this.setCategories(data.categories);
this.setLocations(data.locations);
this.setTags(data.tags);
this.setSales(data.sales);
this.setImages(data.images);
this.setGroups(data.groups);
console.log("IM IN FORKJOIN ")
console.log("DATA lol ", data);
this.filters = [...(data.filters || [])];
this.criteria = [...data.criteria];
this.locationData = locations;
this.salesData = sales;
this.categoriesData = data.categories;
this.setPurchases(data.purchases);
this.setContacts(data.contacts);
this.setActions(data.actions);
this.purchasesData = purchases;
db.populateEntities(data).then(() =>
this.commonService.dataIsLoaded.next(true));
},
(err: any) => {
if (err?.error?.detail) {
this.uiService.showError('Error', err?.error?.detail);
}
this.commonService.dataIsLoaded.next(true);
return of(false);
},
);
return of(true);
}
setActions(items: ActionInterface[]): void {
items.forEach((item) => {
this.actions.set(item.slug, item);
});
}
setContacts(contacts: Contact[] = []): void {
contacts.forEach((contact: Contact) => {
this.contacts.set(contact.slug, contact);
});
}
setCategories(categories: Category[] = []): void {
categories.forEach((category: Category) => {
this.categories.set(category.slug, category);
});
}
setGroups(groups: GroupInterface[] = []): void {
this.groupsService.decorateWithDefaultGroups(groups).subs
cribe((addedGroupsResult) => {
groups.forEach((group) => {
this.groups.set(group.slug, group);
});
addedGroupsResult.forEach((addedGroup: GroupInterface)
=> {
this.groups.set(addedGroup.slug, addedGroup);
});
});
}
```

```
setLocations(locations: Location[] = []): void {
locations.forEach((location: Location) => {
this.locations.set(location.slug, location);
});
}
setTags(tags: Tag[] = []): void {
tags.forEach((tag: Tag) => {
this.tags.set(tag.slug, tag);
});
}
setPurchases(purchases: Purchase[] = []): void {
purchases.forEach((purchase: Purchase) => {
this.purchases.set(purchase.slug, purchase);
this.purchasesTotal += Number(purchase.price) *
purchase.quantity;
});
}
setPurchase(purchase: Purchase): void {
if (purchase?.slug) {
this.purchases.set(purchase.slug, purchase);
}
}
setSales(sales: Sale[] = []): void {
sales.forEach((sale: Sale) => {
this.sales.set(sale.slug, sale);
this.onlineTotal += Number(sale.price);
});
}
setSale(sale: Sale): void {
if (sale?.slug) {
this.sales.set(sale.slug, sale);
}
}
setImages(images: any): void {
images.forEach((image: any) => {
this.images.set(image.slug, image.image);
});
}
setItems(items: Item[]): void {
items.forEach((item: Item) => {
this.items.set(item.slug, item);
this.setStockData(item);
});
this.itemsTotal = items.length;
}
setItem(item: Item): void {
this.items.set(item.slug, item);
this.replacementTotal = 0;
this.lowStockTotal = 0;
this.outOfStockTotal = 0;
this.items.forEach((item: Item) => this.setStockData(item));
this.itemsTotal = this.items.size;
}
setStockData(item: Item): void {
this.replacementTotal += Number(item.replacement_cost) *
item.quantity;
if (item.ideal_quantity && item.quantity !== null &&
item.quantity < item.ideal_quantity) {
this.lowStockTotal += 1;
}
if (item.quantity === 0) {
this.outOfStockTotal += 1;
}
}
deleteItemValueOfStockData(item: Item): void {
this.replacementTotal -= Number(item.replacement_cost) *
item.quantity;
if (item.ideal_quantity && item.quantity !== null &&
item.quantity < item.ideal_quantity) {
this.lowStockTotal -= 1;
}
if (item.quantity === 0) {
this.outOfStockTotal -= 1;
}
this.itemsTotal--;
}
public enrichPurchases(purchases: Purchase[]): Purchase[] {
return purchases.map((purchase) => {
const fullItem = this.items.get(purchase.item);
if (!fullItem || purchase.fullItem) {
return purchase;
}
fullItem.fullCategory = this.categories.get(fullItem.category);
fullItem.fullLocation = this.locations.get(fullItem.location);
if (environment.useFakeData) {
if (!fullItem.photos || !fullItem.photos.length) {
fullItem.photos = [this.getRandom([...this.photos.keys()])];
}
purchase.date = this.getRandomFakeDate();
}
return {
...purchase,
fullItem,
};
});
}
public getRandom(items: string[]): string {
return items[Math.floor(Math.random() * items.length)];
}
getItemBySlug(slug: string): any {
if (this.items) {
return this.items.get(slug);
}
}
getItemByBarcode(code: string): Item[] | null {
if (this.items) {
const allItems: Item[] = [];
this.items.forEach((item) => allItems.push(item));
const filteredItemsByBarcode = allItems.filter((item) => {
if (item.barcode) {
return item.barcode.includes(code);
}
return null;
});
return filteredItemsByBarcode;
}
return null;
}
getItemsByStockState(state: StockState): Item[] {
if (this.items) {
const allItems: Item[] = [];
this.items.forEach((item) => allItems.push(item));
let filteredItemsByStock: Item[] | null = [];
switch (state) {
case StockState.Low:
filteredItemsByStock = allItems.filter((item) => {
return item.ideal_quantity && item.quantity !== null &&
item.quantity <
item.ideal_quantity;
});
break;
```

```
case StockState.Out:
filteredItemsByStock = allItems.filter((item) => item.quantity
=== 0);
break;
default:
console.log('Error');
}
return filteredItemsByStock;
}
return [];
}
//ToDo: Refactor methods
getPurchasesByExpirationDate(): Purchase[] {
if (this.purchases) {
const today: Date = new Date();
const numbDiffDays = 7;
const allPurchases: Purchase[] = [];
this.purchases.forEach((purchase) =>
allPurchases.push(purchase));
const filteredPurchasesByDiffDays =
allPurchases.filter((purchase) => {
if (purchase.expiry) {
const expiryDate: Date = new Date(purchase.expiry);
// @ts-ignore
const diffTime = Math.abs(expiryDate - today);
const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
97
return diffDays <= numbDiffDays;
}
return false;
});
return filteredPurchasesByDiffDays;
}
return [];
}
getItemsByActionsWithDiffDueDateAndToday(): Item[] {
if (this.actions) {
const today: Date = new Date();
const numbDiffDays = 3;
const allActions: ActionInterface[] = [];
this.actions.forEach((action) => allActions.push(action));
const filteredActionsBeDiffDays = allActions.filter((action) => {
if (action.due_date) {
const dueDate: Date = new Date(action.due_date);
// @ts-ignore
const diffTime = Math.abs(dueDate - today);
const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
return diffDays <= numbDiffDays;
}
return false;
});
const items: Item[] = [];
filteredActionsBeDiffDays.forEach((action) => {
const item = this.items.get(action.item);
item ? items.push(<Item>this.items.get(action.item)) : null;
});
return items;
}
return [];
}
public enrichSales(sales: Sale[]): Sale[] {
return sales.map((sale) => {
const fullItem = this.items.get(sale.item);
if (!fullItem || sale.fullItem) {
return sale;
}
fullItem.fullCategory = this.categories.get(fullItem.category);
fullItem.fullLocation = this.locations.get(fullItem.location);
if (environment.useFakeData) {
98
if (!fullItem.photos || !fullItem.photos.length) {
fullItem.photos = [this.getRandom([...this.photos.keys()])];
}
sale.date = this.getRandomFakeDate();
}
return {
...sale,
fullItem,
};
});
}
public enrichItems(items: Item[]): Item[] {
return items.map((item) => ({
...item,
fullCategory: this.categories.get(item.category),
fullLocation: this.locations.get(item.location),
}));
}
public cleanData(): void {
this.items.clear();
this.tags.clear();
this.images.clear();
this.locations.clear();
this.sales.clear();
this.purchases.clear();
this.categories.clear();
}
public randomInt(min: number, max: number) {
return Math.floor(Math.random() * (max - min + 1) + min);
}
private getRandomFakeDate(): string {
const limit = this.randomInt(0, 7);
const date = new Date(+new Date() - limit * 86400000);
const year = date.getFullYear().toString();
const month = (date.getMonth() + 1 < 10 ? '0' +
(date.getMonth() + 1) : date.getMonth() +
1).toString();
const day = (date.getDate() < 10 ? '0' + date.getDate() :
date.getDate()).toString();
return `${year}-${month}-${day}`;
}
}
```

CommonService.ts

```
export class CommonService {
public dataIsLoaded: BehaviorSubject<boolean> = new
BehaviorSubject<boolean>(false);
public userLocale: BehaviorSubject<string> = new
BehaviorSubject<string>('en-US');
constructor(private http: HttpClient) {
this.userLocale.next(this.getUsersLocale(this.userLocale.value
));
}
getPurchaseList(modificationTime: string):
Observable<Purchase[]> {
return
this.http.get<Purchase[]>(`${environment.APIEndpoint}/purc
hases/`, {
headers: this.getHeaders(modificationTime),
});
}
getTagList(modificationTime: string): Observable<Tag[]> {
return
this.http.get<Tag[]>(`${environment.APIEndpoint}/tags/`, {
headers: this.getHeaders(modificationTime),
});
}
getRequests(section: Entity, modificationTime: string):
Observable<any> {
switch (section) {
case Entity.purchases:
return this.getPurchaseList(modificationTime);
case Entity.items:
return this.getItemList(modificationTime);
case Entity.categories:
return this.getCategoryList(modificationTime);
case Entity.locations:
return this.getLocationList(modificationTime);
case Entity.tags:
return this.getTagList(modificationTime);
case Entity.contacts:
return this.getContactList(modificationTime);
case Entity.sales:
return this.getSaleList(modificationTime);
case Entity.groups:
return this.getGroupList(modificationTime);
case Entity.actions:
return this.getActionsList(modificationTime);
default:
return of(true);
}
}
getAllItemsFromSection(
section: Entity,
100
modificationTime = '2021-01-01T00:00:00.000000Z',
acc: any[] = [],
): Observable<
| Purchase[]
| Sale[]
| Item[]
| Category[]
| Location[]
| Tag[]
| Contact[]
| Photo[]
| GroupInterface[]
| ActionInterface[]
> {
const items$: Observable<
| Purchase[]
| Sale[]
| Item[]
| Category[]
| Location[]
| Tag[]
| Contact[]
| Photo[]
| GroupInterface[]
| ActionInterface[]
> = this.getRequests(section, modificationTime);
return items$.pipe(
mergeMap((items: any) => {
acc = [...acc, ...items];
// no more items or last page
if (!items.length || items.length < 20) {
return of(acc);
}
let maxModificationTime = modificationTime;
items.forEach((i: any) => {
if (Date.parse(i.modificationtime as string) >
Date.parse(maxModificationTime)) {
maxModificationTime = i.modificationtime ||
modificationTime;
}
});
return this.getAllItemsFromSection(section,
maxModificationTime, acc);
}),
);
}
public getHeaders(modificationTime?: string): HttpHeaders {
return new HttpHeaders({
101
'If-Modified-Since': modificationTime || '',
});
}
getSaleList(modificationTime: string): Observable<Sale[]> {
return
this.http.get<Sale[]>(`${environment.APIEndpoint}/sales/`, {
headers: this.getHeaders(modificationTime),
});
}
getItemList(modificationTime: string): Observable<Item[]> {
return
this.http.get<Item[]>(`${environment.APIEndpoint}/items/`, {
headers: this.getHeaders(modificationTime),
});
}
getCategoryList(modificationTime: string):
Observable<Category[]> {
return
this.http.get<Category[]>(`${environment.APIEndpoint}/categ
ories/`, {
headers: this.getHeaders(modificationTime),
});
}
getPhotoList(modificationTime: string): Observable<Photo[]>
{
```

```
return
this.http.get<Photo[]>(`${environment.APIEndpoint}/photos/
`, {
headers: this.getHeaders(modificationTime),
});
}
getAttachmentsList(): Observable<any> {
return
this.http.get(`${environment.APIEndpoint}/attachments/`, {
headers: this.getHeaders(),
});
}
getLocationList(modificationTime: string):
Observable<Location[]> {
return
this.http.get<Location[]>(`${environment.APIEndpoint}/locati
ons/`, {
headers: this.getHeaders(modificationTime),
});
}
getContactList(modificationTime: string):
Observable<Contact[]> {
return
this.http.get<Contact[]>(`${environment.APIEndpoint}/contac
ts/`, {
headers: this.getHeaders(modificationTime),
});
}
getActionList(): Observable<any[]> {
return
this.http.get<any[]>(`${environment.APIEndpoint}/actions/`, {
headers: this.getHeaders(),
});
102
}
getGroupList(modificationTime: string):
Observable<GroupInterface[]> {
return
this.http.get<GroupInterface[]>(`${environment.APIEndpoint}
/groups/`, {
headers: this.getHeaders(modificationTime),
});
}
private getActionsList(modificationTime: string):
Observable<ActionInterface[]> {
return
this.http.get<ActionInterface[]>(`${environment.APIEndpoint
}/actions/`, {
headers: this.getHeaders(modificationTime),
});
}
public save(list: string, item: Sale | Purchase | Category |
Location | Tag): Observable<any> {
return this.http.post(`${environment.APIEndpoint}/${list}/`,
item, {
headers: this.getHeaders(),
});
}
public saveItem(item: Item): Observable<any> {
return from(this.decorateItem(item as Item)).pipe(
mergeMap((item) => {
return this.http.post(`${environment.APIEndpoint}/items/`,
item, {
headers: this.getHeaders(),
});
}),
);
}
savePhoto(photo: File): Observable<any> {
const fd = new FormData();
fd.append('imagefile', photo);
return
this.http.post(`${environment.APIEndpoint}/photos/upload/`,
fd, {
headers: this.getHeaders(),
});
}
saveAttachment(attachment: File): Observable<any> {
const fd = new FormData();
const fileName = attachment.name;
fd.append('attachmentfile', attachment, fileName);
return
this.http.post(`${environment.APIEndpoint}/attachments/upl
oad/`, fd, {
headers: this.getHeaders(),
});
}
public downloadAttachment(slug: string): Observable<Blob> {
return
this.http.get(`${environment.APIEndpoint}/attachments/${slu
g}/download/`, {
responseType: 'blob',
103
headers: this.getHeaders(),
});
}
public delete(list: string, slug: string): Observable<any> {
return
this.http.delete(`${environment.APIEndpoint}/${list}/${slug}/`
, {
headers: this.getHeaders(),
});
}
public update(list: string, item: Sale | Purchase | Category |
Location | Tag, slug: string):
Observable<any> {
return
this.http.patch(`${environment.APIEndpoint}/${list}/${slug}/`,
item, {
headers: this.getHeaders(),
});
}
public updateItem(item: Item, slug: string): Observable<any>
{
return from(this.decorateItem(item as Item)).pipe(
mergeMap((item) => {
return
this.http.patch(`${environment.APIEndpoint}/items/${slug}/`,
item, {
headers: this.getHeaders(),
});
}),
);
}
public getImage(imageUrl: string): Observable<Blob> {
return this.http.get(imageUrl, {
responseType: 'blob',
headers: this.getHeaders(),
});
}
public getDate(date: Date): string {
const year = date.getFullYear().toString();
```

```
const month = (date.getMonth() + 1 < 10 ? '0' +
(date.getMonth() + 1) : date.getMonth() +
1).toString();
const day = (date.getDate() < 10 ? '0' + date.getDate() :
date.getDate()).toString();
return `${year}-${month}-${day}`;
}
private getUsersLocale(defaultValue: string): string {
if (typeof window === 'undefined' || typeof
window.navigator === 'undefined') {
return defaultValue;
}
const wn = window.navigator as any;
104
let lang = wn.languages ? wn.languages[0] : defaultValue;
lang = lang || wn.language || wn.browserLanguage ||
wn.userLanguage;
return lang;
}
private decorateItem(item: Item): Promise<Item> {
return new Promise<Item>((resolve) => {
const categoryPromise = db.getUncategorizedCategory();
const locationPromise = db.getUnknownLocation();
Promise.all([categoryPromise, locationPromise]).then((res) =>
{
const internalObservables = [];
// res[0] is category
if (res[0] === undefined && item.category === null) {
internalObservables.push(this.createUncategorizedCategory(
));
} else if (item.category === null && res[0] !== undefined) {
item.category = res[0].slug;
internalObservables.push(null);
} else {
internalObservables.push(null);
}
// res[1] is location
if (res[1] === undefined && item.location === null) {
internalObservables.push(this.createUnknownLocation());
} else if (item.location === null && res[1] !== undefined) {
item.location = res[1].slug;
internalObservables.push(null);
} else {
internalObservables.push(null);
}
forkJoin(internalObservables.map((obs) => (obs === null ?
of(obs) : obs))).subscribe((result)
=> {
if (item.category === null) {
if (result[0] !== null && result[0]) {
item.category = result[0];
}
if (item.location === null) {
if (result[1] !== null && result[1]) {
item.location = result[1];
}
}
resolve(item);
});
});
});
}
105
private createUncategorizedCategory(): Observable<string> {
const category: Partial<Category> = {
name: UNCATEGORIZED_CATEGORY,
desc: '',
subcategories: [],
};
return this.http
.post(`${environment.APIEndpoint}/categories/`, category, {
headers: this.getHeaders(),
})
.pipe(
mergeMap((category) => {
return from(db.categories.add(category as Category));
}),
);
}
private createUnknownLocation(): Observable<string> {
let location: Partial<Location> = {
name: UNKNOWN_LOCATION,
desc: '',
city: '',
country: '',
notes: '',
postalcode: '',
state: '',
street1: '',
street2: '',
sublocations: [],
};
return this.http
.post(`${environment.APIEndpoint}/locations/`, location, {
headers: this.getHeaders(),
})
.pipe(
mergeMap((location) => {
return from(db.locations.add(location as Location));
}),
);
}
}
```