**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система обліку електротоварів з автоматичним розподілом за категоріями»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
*(код, найменування спеціальності)*
освітньо-професійної програми Інформаційні системи та технології
*(назва)*

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Нікіта ВЄРЯГІН
*(підпис)* *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-41
Нікіта ВЄРЯГІН

Керівник: Іван ШАХМАТОВ
*науковий ступінь,*
*вчене звання*

Рецензент: _____
*науковий ступінь,* Ім'я, ПРІЗВИЩЕ
*вчене звання*

**Київ 2025**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

**ЗАТВЕРДЖУЮ**
Завідувач кафедрою ІПЗАС
_____ Каміла СТОРЧАК
«___»_____20__р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Вєрягін Нікіта Олександрович
*(прізвище, ім'я, по батькові здобувача)*

1. Тема кваліфікаційної роботи: Система обліку електротоварів з автоматичним розподілом за категоріями

керівник кваліфікаційної роботи Іван ШАХМАТОВ,
*(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)*

затверджені наказом Державного університету інформаційно комунікаційних технологій від «_»___20___р. №

2. Строк подання кваліфікаційної роботи «11» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, проєкт Django, шаблони HTML та CSS, FastAPI, CI/CD-конфігурацією, а також дампи тестових баз MongoDB.

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

Аналіз теоретичних принципів предметної області
Проєктування та моделювання інформаційної системи
Розробка веб-застосунку

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «21» лютого 2025р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз наявної науково-технічної літератури | | |
| 2 | Вивчення теоретичних основ | | |
| 3 | Дослідження підходів та технологій для створення веб-застосунку | | |
| 4 | Визначення та обґрунтування архітектурних рішень | | |
| 5 | Реалізація застосунку | | |
| 6 | Тестування та оцінка ефективності створеного застосунку | | |
| 7 | Оформлення роботи: вступ, висновки, реферат | | |
| 8 | Розробка демонстраційних матеріалів | | |

Здобувач вищої освіти _____ Нікіта ВЄРЯГІН
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Іван ШАХМАТОВ
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 75 стор., 18 рис., 6 табл., 2 джерел.

*Мета роботи* – спроєктувати та розробити програмну систему обліку електротоварів із автоматичним розподілом за категоріями, що забезпечує точне ведення складських залишків, мінімізує людські помилки та інтегрується з онлайн-каналами продажу.

*Об'єкт дослідження* – процес управління складськими запасами електротехнічної продукції в оптово-роздрібних підприємствах.

*Предмет дослідження* – методи та технології побудови інформаційної системи обліку товарів на основі стеку Python + FastAPI + MongoDB з механізмами автоматичної категоризації (rule-based і ML-класифікатор), REST-API та веб-інтерфейсом HTML/CSS/JS.

*Короткий зміст роботи.* У вступі обґрунтовано актуальність цифровізації складського обліку електротоварів і проблему помилок ручної класифікації. Здійснено огляд існуючих ERP-та WMS-рішень, визначено типові категорії товарів за класифікатором ETIM/GS1 та потреби бізнес-процесів (прихід, інвентаризація, відвантаження, аналітика). Сформульовано функціональні (автоматичний підбір категорії, API для кас та маркетплейсів, звітність) і нефункціональні вимоги (швидкодія, масштабованість, інформаційна безпека). Розроблено архітектуру системи, створено UML-діаграми прецедентів, компонентів та послідовностей. Реалізовано прототип:

1. Backend – FastAPI, асинхронний драйвер Motor до MongoDB;
2. Модуль категоризації – словникові правила + навчена модель sklearn/BERT (точність ≈ 95 %);
3. Frontend – адаптивний інтерфейс на JS;
4. Інтеграція з касовим ПРРО та REST-webhooks маркетплейсів.

Проведено модульне та інтеграційне тестування, виміряно продуктивність (середній час запису позиції – 60 мс, пошуку – 25 мс). Оцінено економічний ефект: зниження кількості помилок у категоризації з 12 % до <1 %, скорочення часу приймання партії на 35 %. Система підтримує вимоги стандартів ISO 9001 і захисту персональних даних GDPR, реалізовано журнал аудиту дій користувачів.

КЛЮЧОВІ СЛОВА: облік товарів, електротехніка, автоматична категоризація, FastAPI, MongoDB, штучний інтелект, WMS, складська логістика.

# ЗМІСТ

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ – Електронно-обчислювальна машина;

API – Application Programming Interface;

IT – Informaition Technology;

OS/ОС – Operation System;

AI – Artificial Intelligence;

ML – Machine Learning;

GUI – Graphical User Interface;

UX – User Experience;

UI – User Interface;

АГ – Агент збирання даних

ОЗД – Автоматизований збір даних

МЗН – Методи зменшення негативного впливу

п. –пункт;

рис. –Рисунок;

с. (стр.) – сторінка.

# ВСТУП

Інформаційні технології у сфері складської логістики та електротехнічної торгівлі стають ключовим чинником підвищення ефективності бізнес-процесів, точності обліку та швидкості обслуговування клієнтів. Зі стрімким зростанням номенклатури електротоварів (кабелі, вимикачі, світлотехніка, інструмент) і появою мультиканальних продажів (роздріб, опт, e-commerce) особливої ваги набувають системи обліку товарів із автоматичним розподілом за категоріями. Такі системи дають змогу мінімізувати ручні операції, уникнути помилок класифікації, забезпечити актуальні залишки та оперативно інтегруватися з касами, маркетплейсами й ERP-платформами.

Предметна область даного дослідження охоплює розробку інформаційної системи обліку електротехнічної продукції, що дозволяє ефективно управляти даними про артикул, кількість, ціну, партію, а також автоматично визначати категорію товару на основі правил і моделей машинного навчання. Упровадження такої системи забезпечить своєчасний доступ до достовірних складських даних, скоротить час обробки поставок, зменшить дефіцит і надлишкові запаси, підвищить якість аналітики та швидкість прийняття рішень.

**Метою роботи є** проектування та створення функціональної й надійної системи обліку електротоварів із автоматичним розподілом за категоріями, що відповідає сучасним вимогам інформаційної безпеки, підтримує масштабованість і є зручною для користувачів різних рівнів.

Для досягнення поставленої мети передбачається виконання таких завдань:

1. Провести аналіз предметної області та визначити категорії користувачів системи та основні типи даних, що підлягають обліку.
2. Сформулювати функціональні та нефункціональні вимоги до системи обліку електротоварів.
3. Проаналізувати існуючі рішення з метою виявлення їх переваг і недоліків.

4. Розробити архітектуру системи з використанням UML-діаграм.

5. Реалізувати прототип, модуль автоматичної категоризації, веб-інтерфейс.

6. Провести тестування системи, описати основні сценарії, проаналізувати результати та визначити можливі недоліки.

7. Оцінити ефективність рішення, визначити економічний ефект і надати рекомендації щодо подальшого розвитку.

Таким чином, робота спрямована на створення ефективного інструменту для управління електротехнічною продукцією, що забезпечить високу точність обліку, оптимізацію логістики та підвищення конкурентоспроможності підприємств.

**Об'єктом дослідження** є процес управління складськими запасами електротехнічної продукції в оптово-роздрібних мережах.

**Предметом дослідження** є методи та технології створення інформаційної системи для автоматичного обліку та категоризації електротоварів.

**Наукова новизна одержаних результатів** полягає в розробці гібридного підходу до автоматичної категоризації товарів, що поєднує словникові правила з мовною ML-моделлю (BERT), а також у пропозиції мікросервісної архітектури, здатної працювати в режимі реального часу з високим навантаженням.

**Практичне значення одержаних результатів** полягає у створенні прототипу системи, який може бути впроваджений дистриб’юторами та роздрібними мережами для зниження витрат на складські операції, скорочення часу приймання та відвантаження і зменшення кількості помилок, зумовлених людським фактором.

**Структура та обсяг випускної кваліфікаційної роботи.** Випускна кваліфікаційна робота складається із вступу, шести розділів, висновків, списку використаних джерел із 20 найменувань, додатків і містить 75 сторінок основного тексту, 18 рисунків і 6 таблиць.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

## 1.1 Актуальність автоматизації обліку електротоварів

У сучасних умовах стрімкого розвитку електронної торгівлі та глобалізації ланцюгів постачання автоматизація обліку електротоварів набуває особливої актуальності [1]. Традиційні методи ведення журналів та паперових форм дедалі більше не відповідають зростаючим вимогам до точності даних та швидкості обробки інформації про рух товарів. Механічні помилки при введенні даних, затримки в оновленні залишків та низька прозорість процесу інвентаризації призводять до перевитрат, нестачі критично важливих компонентів або навпаки – надлишку запасів, що негативно впливає на фінансові результати компанії [2]. Автоматизована інформаційна система дозволяє в режимі реального часу відстежувати надходження, переміщення та відпуск товарів, знижуючи ризики помилок, оптимізуючи прийняття рішень на різних рівнях управління [3].

Сучасні виклики в логістиці та складському господарстві вимагають гнучких і масштабованих рішень для обліку електротоварів, які часто мають велику номенклатуру – від дротів та кабелів до складних електронних модулів. Збільшення обсягу товарних груп, багатоступеневі ланцюги постачання та потреба у точному прогнозуванні попиту роблять необхідним впровадження інформаційних систем із підтримкою автоматичної класифікації товарів за категоріями. Це дозволяє миттєво сортувати асортимент, формувати аналітичні звіти, адаптувати маршрути поставок і управляти запасами на основі актуальних даних. Такі системи сприяють зменшенню часу обробки замовлень, скороченню витрат на зберігання та підвищенню рівня обслуговування клієнтів, оскільки персонал отримує чітку та достовірну інформацію без зайвих затримок.

Прискорення операцій за допомогою автоматизованої системи обліку електротоварів забезпечує значне скорочення часу на обробку кожного етапу – від приймання та розміщення на складі до комплектації замовлень і

відвантаження. Заміна ручного введення даних штрих- або QR-скануванням дозволяє миттєво реєструвати переміщення партій товарів, суттєво зменшуючи цикл обробки замовлення та звільняючи персонал для вирішення більш складних завдань. Завдяки цьому підвищується загальна пропускна здатність складу, компанія може обслуговувати більше клієнтів з тим самим штатом працівників, що прямо трансформується у зростання доходів та конкурентних переваг.

Зниження рівня помилок є ще одним ключовим виграшем від впровадження інформаційної системи. Автоматичні перевірки відповідності SKU, контроль допустимих діапазонів кількості й вбудовані алгоритми валідації запобігають допуску неточностей на будь-якому етапі: прийомі, переміщенні, відпуску або інвентаризації. Це знижує ризик виникнення «втрачених» або «подвійно списаних» одиниць товару, мінімізує витрати на виправлення помилок і пов’язані зі збоєм процесів компенсації. [4] У підсумку зниження помилок покращує фінансову прозорість, робить прогнози попиту більш точними та підвищує довіру партнерів і клієнтів, оскільки вони отримують гарантії своєчасної і коректної доставки потрібного обладнання.

## 1.2 Історія розвитку систем обліку товарів

У ранній період розвитку торгівлі та складського господарства облік товарів здійснювався виключно вручну, на паперових носіях. Кожна операція – приймання, переміщення, відпуск – фіксувалася у журналах прибутково-видаткових накладних, картках обліку чи книгах-інвентарях. Записи велися чорнилом або олівцем, часто різними працівниками, що вже на цьому етапі створювало неоднорідність стилю, скорочень і позначень. У невеликих крамницях достатньо було простого зошита, де у графах зазначали назву товару, кількість та дату приходу або витрати. На більших оптових складах використовували картотечні системи: на кожний артикул заводили окрему картку, куди вручну переносили інформацію з накладних, накопичуючи історію руху за період. У бухгалтерії паралельно вели оборотні відомості, де

підсумовували залишки на кінець дня, тижня чи місяця. Такий підхід вимагав значних трудових затрат, оскільки будь-яке зведення – наприклад, формування замовлення постачальнику або підготовка річного звіту – означало механічний перегляд десятків сторінок і ручний підрахунок сум.

Висока залежність від людського фактора породжувала накопичення помилок: неточне переписування чисел, дублювання записів, пропуск артикулів. Помилки виявлялися лише під час інвентаризацій, які проводилися раз на квартал чи навіть раз на рік, коли різниця між фактичною і документальною кількістю могла сягати десятків відсотків [5]. Для коригування залишків працівникам доводилося складати акти списання або доприбутковувати виявлений надлишок, що ускладнювало податковий облік і псувало репутацію компанії перед контрагентами [6]. Наявність великого масиву паперових документів уповільнювала пошук інформації: щоб дізнатися партію походження товару або дату його закупівлі, комірнику доводилося піднімати архівні накладні, що займало години. Окремою проблемою була фізична зношеність носіїв: сторінки рвалися, чорнила тьмяніли, а сама документація потребувала місця зберігання й мали ризик пошкодження водою чи вогнем. Таким чином, ручні та паперові методи обліку були виправдані лише в умовах обмеженого асортименту й невеликого обсягу операцій; із зростанням темпів торгівлі й розширенням номенклатури вони почали стримувати розвиток бізнесу, що згодом стало поштовхом до механізації та повної автоматизації процесів обліку [7-8].

З поширенням міні-ЕОМ наприкінці 1960-х та розвитком концепції MRP (Material Requirements Planning) у 1970-х підприємства почали переходити від ізольованих автоматизованих робочих місць до централізованого зберігання даних про запаси, замовлення й виробничі графіки. Поява недорогих дискових накопичувачів і перших реляційних СУБД (Ingres, Oracle, IBM DB2) дала змогу створити єдину базу, до якої підключалися бухгалтерія, склад, відділ закупівель і виробництво. На початку 1990-х ця логіка еволюціонувала у повноцінні ERP-системи (Enterprise Resource Planning) – SAP R/3, Baan, JD Edwards, а пізніше

Oracle E-Business Suite та Microsoft Dynamics. ERP охоплювали вже не лише запаси, а й фінанси, кадри, CRM, формуючи наскрізний бізнес-процес «замовлення-готівка» (order-to-cash). Для обліку товарів це означало, що кожна операція — від сканування штрих-коду на прийманні до відвантаження клієнтові – миттєво відображалася в єдиній транзакційній таблиці, а залишки, собівартість і доступність у реальному часі були доступні всім службам. Централізація усунула дублювання записів та «інформаційні острови», підвищивши точність планування закупівель, мінімізувавши «мертвий» запас і зменшивши ризик простою через нестачу комплектуючих. Крім того, потужні механізми звітності (OLAP-куби, BI-модулі) дозволили керівництву здійснювати глибоку аналітику ABC/XYZ-класифікації, прогнозувати попит та оптимізувати логістичні маршрути. Згодом на ринок вийшли доступніші локальні рішення (1С:Підприємство, Oodo, ERPNext), а також хмарні SaaS-платформи, що передали частину експлуатаційних витрат провайдерам та відкрили ERP-функціонал для малого та середнього бізнесу [9-10]. У результаті централізовані БД та ERP-системи стали еталонною основою сучасного автоматизованого обліку, задавши стандарти цілісності, узгодженості та доступності даних, без чого неможливо уявити ефективне управління великим складом електротоварів.

З початку 2010-х років швидке розгортання хмарних платформ — Amazon Web Services, Microsoft Azure, Google Cloud – дозволило винести ядро облікових систем у високодоступне середовище «Software as a Service». Дані про запаси тепер зберігаються у кластерах, що реплікуються між дата-центрами, а доступ гарантується через браузер або REST API з будь-якої точки світу з затримкою в лічені мілісекунди. Головною перевагою стало миттєве масштабування: у пікові сезони (Black Friday, будівельний бум) склад може обробити у-два чи утричі більше транзакцій без закупівлі нового «заліза».

Паралельно з розвитком хмари мобільні додатки перетворили звичайний смартфон чи планшет на повноцінний сканер та термінал обліку. Програмні SDK для камер (Google ML Kit, Apple Vision) навчилися декодувати 1D/2D-штрих-коди й QR-мітки зі швидкістю спеціалізованих сканерів, а вбудовані NFC- та

Bluetooth-модулі дають змогу підключати ручні RFID-головки та принтери етикеток. Офлайн-режим із синхронізацією після відновлення мережі забезпечує безперервність роботи у металевих ангарах чи віддалених майданчиках.

Поєднання хмарних сервісів і мобільних клієнтів кардинально знизило бар'єр входу для малого та середнього бізнесу: достатньо підписки й смартфонів, щоб отримати функціонал рівня корпоративних WMS — керування адресним зберіганням, динамічний підбір маршрутів відбору (pick-path), автоматичне поповнення мінімальних запасів. Таким чином, хмарні сервіси та мобільні додатки не лише підвищили оперативність контролю складу, а й створили новий стандарт доступності, прозорості та гнучкості управління ланцюгами постачання електротоварів.

### 1.3 Огляд існуючих інформаційних систем обліку товарів

Комерційні системи обліку товарів охоплюють комплексні ERP-платформи, що інтегрують управління запасами, фінансами, логістикою й виробництвом у єдине середовище даних, і спеціалізовані WMS-модулі, орієнтовані виключно на складські операції. Найпоширеніший представник корпоративного сегмента – SAP S/4HANA (раніше SAP R/3), який забезпечує наскрізний бізнес-процес «замовлення та готівка»: від прогнозу попиту й формування замовлення постачальнику до списання собівартості після відвантаження [11-12]. У модулі SAP Extended Warehouse Management система підтримує адресне зберігання, дворівневі стратегії відбору (bulk/pick-face), багатокрокові перерахунки та RFID-інтеграцію; для електротехнічних підприємств особливо цінні функції batch-tracking і серійний облік, що дають можливість відстежувати кожен кабельний барабан чи електронний модуль за заводським номером. Oracle NetSuite позиціонується як хмарне SaaS-рішення для середнього бізнесу [13]. Воно містить модуль Advanced Inventory, де автоматизовані циклічні інвентаризації, динамічна система reorder-point і власний механізм ABC-аналізу товарів. Для компаній,

що торгують електротоварами, важливо, що NetSuite пропонує natively-embedded EDI-інтеграцію, дозволяючи обмінюватися накладними й ASN-повідомленнями з мережевими рітейлерами без додаткових шлюзів.

Серед спеціалізованих WMS варто згадати Solvo.WMS і Ax'Libris Sklad, які пропонують алгоритми оптимізації маршрутів відбору під великогабаритні котушки кабелю, підтримку крос-докінгу та інтеграцію з автоматизованими конвеєрними лініями [14]. На противагу дорогим корпоративним ERP-комплексам усе більшу популярність серед малого й середнього бізнесу набувають open-source платформи, що поєднують відсутність ліцензійних платежів із гнучкістю кастомізації вихідного коду. Флагманом цього сегмента є Odoo (раніше OpenERP), модульна архітектура якого дозволяє підключати тільки потрібні компоненти: «Склад» (Inventory), «Продажі», «Закупівлі», «Бухгалтерія». Odoo зберігає дані у PostgreSQL, але через ORM дає змогу легко додавати NoSQL-сховища для аналітики; у модулі складу реалізовано адресне зберігання, багаторівневі правила відбору, серійний та партійний облік, а за допомогою community-додатків – інтеграцію з Bluetooth-сканерами та принтерами етикеток. Ще одна зріла платформа – ERPNext, написана на Python/Фреймворку Frappe. Вона має веб-інтерфейс, сховище на MariaDB та повністю REST-JSON API. ERPNext підтримує багатомовність, внутрішні рольові права доступу й механізм «Workflow», що дозволяє затверджувати переміщення та списання електротоварів через е-підписи. Особливістю є вбудована проста WMS-логіка: штрих-кодування з камери смартфона, циклічні інвентаризації й rule-based прогноз попиту без окремих BI-інструментів. Для компаній, які потребують лише складський контур, актуальним є OpenWMS.org або легковажні системи типу Snipe-IT (серійний облік, QR-генератор, звітність по історії переміщень). Їх легко розгорнути у Docker-контейнерах, під'єднати до MongoDB або MySQL і інтегрувати з власними мікросервісами через Webhook.

Окрему нішу між великими ERP та універсальними open-source-платформами заповнюють спеціалізовані додатки для електротехнічних і промислових компаній, сфокусовані на властивостях саме кабельно-

провідникової та низьковольтної номенклатури. Прикладом є CableSuite WMS: система містить довідник стандартів (ВВГ, NYM, КГ тощо), автоматично розраховує вагу й довжину бухти за перерізом і класом гнучкості, а алгоритм відбору «Cut-to-Length» підказує оператору, з якої котушки відмотати потрібні 37 м кабелю, щоби мінімізувати обрізки. У модулі виробництва CableSuite інтегрується з відмотувальними машинами через OPC UA, фіксуючи фактичну довжину та відхилення по омметру в режимі реального часу. Інший приклад — ElectroStock Pro, котрий підтримує серійний облік електронних компонентів, специфічні поля «номінал», «SMD/THT форм-фактор», «клас вогнестійкості пластика» та автоматичне мапування на коди IEC. Система має бібліотеку UL-сертифікатів: при введенні нової партії конденсаторів вона перевіряє, чи відповідає постачальник потрібному класу безпеки і, у разі невідповідності, блокує прихід на склад до ручного підтвердження технолога.

Комерційні ERP забезпечують максимальну інтеграцію й підтримку, але коштують найдорожче та потребують тривалого впровадження. Open-source рішення пропонують майже безкоштовну ліцензію й повну контрольованість коду, однак вимагають власних ресурсів на підтримку та безпеку. Нішеві галузеві програми мінімізують кастомізацію завдяки готовій логіці електротехнічного обліку, проте поступаються у масштабованості й ширині функціоналу.

Таблиця 1.1

Порівняльний аналіз можливостей та обмежень

| Критерій | Комерційні ERP/WMS | Open-source платформи | Нішеві електротехнічні рішення |
|---|---|---|---|
| Вартість ліцензії | Висока: ліцензії + модулі | Безкоштовні, лише хостинг | Середня: підписка/галузева ліцензія |
| Кастомізація | Обмежена, через партнерів | Повний доступ до коду | Локальні допрацювання, ядро закрите |
| Впровадження | 6–18 міс. | 1–3 міс. | 1–2 міс. |
| Масштабованість | Горизонтальні кластери | Залежить від інфраструктури | Оптимізовані під середній обсяг |
| Інтеграції | Готові конектори (EDI, OPC, BI) | REST/API, Webhooks | OPC UA, лабораторні дані |
| Цільова аудиторія | Великі дистриб’ютори | Малі та СМБ | Кабельні заводи, електромонтажні фірми |

## 1.4 Вибір технологій та інструментів для розробки

Таблиця 1.2

Переваги та недоліки різних підходів

| Критерій | FastAPI | Django REST | Flask | Express (Node) | Go (Fiber) |
|---|---|---|---|---|---|
| Швидкість,req/s, тис. | 40 | 12 | 15 | 40 | 50 |
| Валідація + OpenAPI | Pydantic + auto-Swagger | DRF serializers | сторонні lib | Joi / swagger-ui | struct-tags / swaggo |
| Складність старту | 1 файл ≈ 150 LOC | висока | низька | середня | висока |
| Data/ML інтеграція | NumPy/Pandas | висока | низька | через gRPC | обмежено |

FastAPI базується на асинхронному фреймворку Starlette та сервері Uvicorn із підтримкою ASGI-протоколу, що дозволяє обробляти тисячі паралельних HTTP-запитів без створення окремого потоку або процесу на кожне з’єднання. Це призводить до значного зниження споживання оперативної пам’яті та підвищення пропускної здатності API. Як показано в Таблиці 1.6, FastAPI демонструє продуктивність, аналогічну популярним рішенням на Go або Node.js, проте зберігає зручність розробки на Python [15]. Декларативна система валідації вхідних даних на базі Pydantic спрощує перевірку коректності запитів і одночасно генерує детальну OpenAPI-специфікацію, що полегшує інтеграцію з фронтенд- та мобільними клієнтами. Завдяки підтримці всієї екосистеми Python — включно з бібліотеками для обробки даних (NumPy, Pandas) та машинного навчання (Scikit-learn) — обрана технологія дозволяє виконувати аналітичні розрахунки та класифікацію товарів без необхідності переходу на іншу мову програмування. Таким чином, FastAPI поєднує високу продуктивність, легкість розробки та гнучкість екосистеми й у Таблиці .1.6 визнано оптимальним

рішенням для реалізації високонавантаженого REST-API системи обліку електротоварів.

Таблиця 1.3

Переваги та недоліки різних інструментів

| Критерій | MongoDB | PostgreSQL | Cassandra | Couchbase |
|---|---|---|---|---|
| Схема | Гнучка, беззмінива; додати поля в будь-який момент | Жорстка, ALTER TABLE | Гнучка, але лише ключ–значення | Гнучка, JSON-документи |
| Узгодженість | Підтримка транзакцій на документ рівні | ACID на всіх рівнях | Подсумована (Eventual) | Pseudo-SQL (N1QL) з транзакціями |
| Індексація | JSON-шляхи, гео, текст | B-tree, GiST, GIN | Обмежені (ключі, палітри) | JSON-шляхи, FTS |
| Агрегації | Потужний pipeline, stages | CTE, window-функції | Мінімальні | MapReduce + N1QL |
| TTL / автоочищення | Вбудовані TTL-колекції | Через cron/Triggers | Через TTL колонок | TTL для документів |
| Масштабування | Replica Set + Sharding | Master-standby, sharding вручну | Лінійне шардінг «з коробки» | Multi-dimensional scaling |
| Управління великими даними | Добре для сотень мільйонів документів | Добре, але схему треба оптимізувати | Відмінно для write-heavy | Добре, з кешем у пам’яті |

У Таблиці 1 показано, що класичні реляційні СУБД з жорсткою схемою вимагають чітко визначених структур таблиць і змінюваних DDL-операцій

(ALTER TABLE) при додаванні чи модифікації полів документа, що створює затримки та потребує простоїв під час міграції. Натомість MongoDB застосовує документно-орієнтований підхід, де кожен товар представлений JSON-документом із довільним набором полів — це дозволяє вбудовувати специфічні характеристики електротоварів (переріз, кількість жил, тип цоколю, криву спрацювання) без будь-яких попередніх налаштувань на рівні схеми [16]. Система індексації по шляху до вкладених полів забезпечує надшвидкий пошук і сортування за складними критеріями, а агрегаційний фреймворк з багатьма стадіями (match, group, project) дає змогу виконувати складні запити на обчислення статистик і перетворення даних без вивантаження їх у пам’ять клієнтської програми. TTL-індекси автоматично видаляють записи зі старими мітками часу — це суттєво оптимізує використання дискового простору під журнали операцій, не впливаючи на продуктивність основних запитів [17]. Завдяки реплікаційним наборам MongoDB забезпечує високий рівень доступності даних: у випадку збоїв одного вузла інші продовжують обслуговувати запити, тоді як шардинг дозволяє рівномірно розподілити навантаження при зростанні кількості транзакцій, що є критично необхідним для системи обліку, де сезонні піки можуть в десятки разів перевищувати середній трафік [18]. Такий набір можливостей робить MongoDB ідеальною платформою для гнучкого, масштабованого та продуктивного зберігання даних про електротовари.

Таблиця 1.4

Переваги та недоліки різних інструментів

| Критерій | Статичний HTML/CSS/JS | React | Vue.js | Angular |
|---|---|---|---|---|
| Залежності | 0 – лише браузер | React, JSX, Babel, NPM | Vue, NPM | Angular CLI, RxJS, TypeScript |
| Збірка / CI | Не потрібна | Webpack / Vite | Vite / Vue CLI | Angular CLI |
| Розмір бандлу | < 50 кБ | 100–200 кБ | 80–150 кБ | 150–300 кБ |
| Продуктивність загрузки | Миттєва | Залежить від бандлів | Дуже швидка | Помірна |
| Динаміка UI | fetch() + DOM | Virtual DOM | Virtual DOM | Zone.js + change detection |
| Learning curve | Дуже низька | Середня | Низька | Висока |
| SEO та SSR | Природньо для MPA | Потрібен Next.js / Gatsby | Nuxt.js | Universal |
| SPA чи MPA | MPA / багатосторінковий підхід | SPA | SPA / MPA | SPA |
| Debugging | Chrome DevTools | React DevTools | Vue DevTools | Angular DevTools |

У таблиці 1.4 представлено детальний порівняльний аналіз чотирьох підходів до організації клієнтської частини: статичного HTML/CSS/JS, React, Vue та Angular. З його результатів видно, що вибір статичного фронтенду є оптимальним у нашому випадку перш за все через відсутність необхідності в інструментах збірки та надбудовах CI/CD — достатньо звичайного веб-сервера, щоб обслуговувати файли. Це значно знижує складність розгортання та

супроводу, оскільки не потрібно налаштовувати Webpack, Babel чи інші транслятори, а також забезпечує миттєву першу візуалізацію сторінки без додаткового часу на завантаження великих бандлів [19]. Динамічна взаємодія з API реалізована через стандартний виклик із вбудованою підтримкою передачі JWT-cookie [20]. У результаті статичний фронтенд забезпечує швидке завантаження, мінімальні витрати ресурсів і достатній рівень динаміки для реалізації CRUD-функцій системи обліку електротоварів, тоді як повноцінні SPA-фреймворки запропонували б значно вищу складність і вагу, які в нашому проєкті не виправдані.

## 1.5 Постановка задачі та формулювання вимог до розробки системи

У зв’язку зі зростанням обсягів та різноманітності електротоварів виникає необхідність створення інформаційної системи, що забезпечить оперативний і точний облік усіх позицій на складі, автоматичну класифікацію за категоріями та підтримку гнучкої структури даних. Постановка задачі полягає в розробці веб-застосунку з асинхронним API на FastAPI, базою даних MongoDB та фронтендом на чистому HTML/CSS/JavaScript, який дозволить зареєстрованим користувачам виконувати CRUD-операції над товарами, шукати та фільтрувати їх, а адміністраторам – контролювати історію змін і формувати статистичні звіти за довільні періоди. Система повинна відповідати наступним вимогам: забезпечувати низьку затримку API ($\leq 200$ мс), стійкість до навантажень ($\geq 1000$ запитів/с), гарантувати безпеку даних через HTTPS та JWT-аутентифікацію, а також підтримувати горизонтальне масштабування в Docker-середовищі та автоматичне видалення застарілих логів через TTL-індекси.

# 2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1 Архітектура програмного рішення

Таблиця 2.1

Загальний огляд обраних технологій та інструментів розробки

| Назва | Опис |
|---|---|
| Backend | Python 3.x, FastAPI, Uvicorn, Pydantic, JSON Web Token (JWT), passlib (bcrypt), pytest-asyncio |
| Frontend | HTML/CSS/JavaScript, Fetch API, статичні файли (/static), Bulma / Bootstrap (CSS-бібліотеки) |
| СУБД | MongoDB, Motor (асинхронний драйвер) |
| Інфраструктура | Docker, Docker Compose, OpenAPI/Swagger UI |

У межах розробки системи управління запасами обрано клієнт-серверний підхід із монолітною основною реалізацією, що дозволяє швидко прототипувати функціонал і забезпечити асинхронну обробку запитів. Система складається:

1. Фронтенд (клієнт): статичні HTML-файли з динамічним JavaScript (fetch API) для взаємодії з REST-кінцевими точками.

2. Бекенд (сервер): FastAPI з асинхронними роутами, що обробляють HTTP-запити та взаємодіють з БД.

3. База даних: MongoDB з асинхронним драйвером motor.motor_asyncio для зберігання товарів, користувачів і журналів аудиту.

У межах розробки системи управління запасами обрано клієнт-серверний підхід із монолітною основною реалізацією, що дозволяє швидко прототипувати функціонал і забезпечити асинхронну обробку запитів.

## 2.2 Формулювання функціональних та нефункціональних вимог

Таблиця 2.2

Функціональні вимоги

| ID | Вимога | Опис |
|---|---|---|
| F1 | CRUD-операції з товарами | Створення, читання, оновлення та видалення записів про товари (name, sku, quantity, price). |
| F2 | Автоматична категоризація | Визначення категорії товару за назвою через модуль categorizer. |
| F3 | Пошук та фільтрація | Пошук за назвою, SKU, категорією; фільтрація та сортування результатів у таблиці. |
| F4 | Регістрація користувачів | Форма реєстрації (username, password) з валідацією та хешуванням паролю. |
| F5 | Логін та JWT-аутентифікація | Вхід із поверненням JWT-токена в HttpOnly-cookie; перевірка під час кожного запиту. |
| F6 | Ролі користувачів | Розмежування доступу: “user” (CRUD, словник) та “admin” (додатково журнал аудиту, статистика). |
| F7 | Адміністраторська панель | Інтерфейс /admin: перегляд логів /logs та статистики /stats. |
| F8 | Перегляд словника ключових слів | Відображення /keywords: таблиця всіх ключів і категорій. |
| F9 | Експорт звітів | Формування статистичного звіту за період (кількість доданих/змінованих записів по користувачах). |

У Таблиці 2.2 наведено перелік ключових функціональних вимог до інформаційної системи обліку електротоварів. Відповідно до цих вимог, система повинна підтримувати повний набір CRUD-операцій з товарами (F1), автоматично визначати їхню категорію за назвою за допомогою модуля categorizer (F2) та надавати гнучкий пошук і фільтрацію за назвою, SKU й категорією з можливістю сортування результатів (F3). Для роботи з додатком передбачено механізми реєстрації користувачів з валідацією та хешуванням паролів (F4) та вхід через JWT-аутентифікацію з передачею токена в HttpOnly-cookie і перевіркою при кожному запиті (F5). Розмежування доступу здійснюється через дві ролі: звичайний користувач (“user”) може виконувати CRUD і переглядати словник ключових слів (F6), а адміністратор (“admin”) має додатковий доступ до панелі /admin, де доступні журнал аудиту (/logs) і статистика змін (/stats) (F7). Крім того, система надає окремий інтерфейс для перегляду повного словника ключових слів із категоріями (F8) та можливість експорту аналітичного звіту за будь-який період, що відображає кількість доданих або змінених записів по кожному користувачеві (F9). Всі ці функції утворюють основу функціонального ядра порталу, забезпечуючи комплексне керування асортиментом електротоварів.

Таблиця 2.3

Нефункціональні вимоги

| ID | Вимога | Показник / Опис |
|---|---|---|
| N1 | Безпека | bcrypt-хешування паролів (passlib), HTTPS для всіх запитів, захист CSRF через credentials:'include'. |
| N2 | Час відгуку API | $\leq 200$ мс для стандартних CRUD-запитів при середньому навантаженні. |
| N3 | Пропускна здатність | $\geq 1000$ запитів/с на рівні FastAPI-сервера без втрати продуктивності. |

Продовження таблиці 2.3

| ID | Вимога | Показник / Опис |
|---|---|---|
| N4 | Масштабованість | Горизонтальне розгортання FastAPI в Docker-контейнерах; шардинг та реплікація MongoDB. |
| N5 | Надійність | Відмовостійкість через мультивузлові реплікації бази даних; автоматичне перезапуск контейнерів. |
| N6 | Підтримка та супровід | Покриття тестами $\geq 80\,\%$ коду (pytest + pytest-asyncio); інтеграція CI для автотестів. |
| N7 | Документація | Автоматична генерація OpenAPI/Swagger UI для всіх ендпоінтів; актуальність згідно релізів. |
| N8 | Використовуваність | Інтуїтивно зрозумілий UI, адаптивний дизайн через CSS-бібліотеку; час навчання користувача $\leq 1$ дня. |
| N9 | Логування та аудит | Збереження всіх змін у колекції logs; TTL-індекси для автоматичного очищення логів старше 30 днів. |
| N10 | Портативність | Контейнеризація через Docker/Docker Compose; можливість розгортання на будь-якій хмарі чи локально. |

Таблиці 2.3 узагальнено основні нефункціональні вимоги до системи обліку електротоварів. Вимога N1 передбачає використання bcrypt-хешування паролів через бібліотеку passlib, обов’язкове шифрування всіх HTTP-запитів за допомогою HTTPS та захист від CSRF-атак шляхом передачі JWT-cookie. Час відгуку API (N2) обмежується 200 мс для стандартних CRUD-операцій при середньому навантаженні, а пропускна здатність (N3) має становити не менше ніж 1000 запитів за секунду без погіршення продуктивності сервера. Масштабованість (N4) досягається за рахунок горизонтального розгортання контейнерів FastAPI у Docker та використання шардингу і реплікації MongoDB,

що одночасно забезпечує й відмовостійкість (N5) – автоматичний перезапуск контейнерів і мультивузлові реплікації бази даних гарантують безперервність обслуговування. Підтримка та супровід (N6) реалізуються через покриття коду тестами щонайменше на 80 % із використанням pytest та pytest-asyncio та інтеграцію з CI-пайплайном для автоматичного запуску автотестів. Документація API (N7) формується автоматично за стандартом OpenAPI і відображається в Swagger UI, при цьому її актуальність підтримується під час кожного релізу. Використовуваність інтерфейсу (N8) передбачає інтуїтивно зрозумілий дизайн з адаптивними стилями на основі CSS-бібліотек, що дозволяє опанувати систему за один робочий день. Журналювання й аудит (N9) реалізовані через окрему колекцію logs у MongoDB з TTL-індексами, які автоматично видаляють записи старше 30 календарних днів. Портативність (N10) забезпечується контейнеризацією за допомогою Docker та Docker Compose, що дозволяє розгортати систему як на локальних серверах, так та в будь-якій хмарній інфраструктурі.

### 2.3 Структура та функціональні модулі системи

Модуль аутентифікації та авторизації відповідає за управління доступом до системи: він обробляє реєстрацію нових користувачів із валідацією та хешуванням паролів (bcrypt), забезпечує вхід через форму логіна з генерацією та верифікацією JWT-токенів (via python-jose), а також підтримує оновлення (refresh) токенів і перевірку ролей (admin чи user) через залежності FastAPI. Усі операції реалізовані асинхронно, щоб не блокувати основний event-loop.

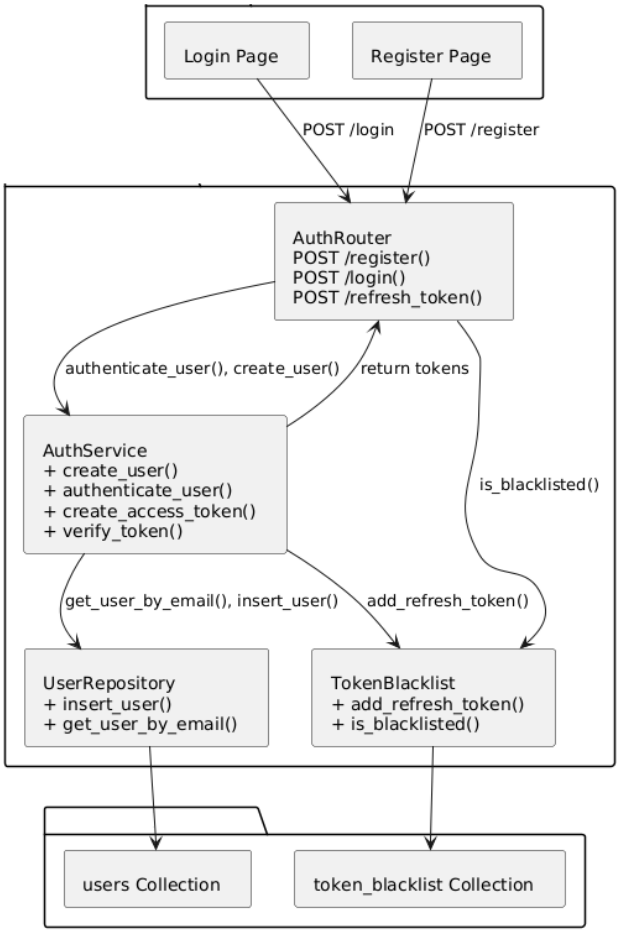



Рис. 2.1 Модуль аутентифікації та авторизації

Модуль керування товарами (CRUD) дозволяє додавати нові записи про товари, читати їх список із пагінацією й фільтрацією, оновлювати атрибути (назва, SKU, кількість, ціна) та видаляти непотрібні позиції. Кожна операція відбувається через REST-кінцеві точки FastAPI і використовує Pydantic-схеми для валідації вхідних даних, а також асинхронний драйвер Motor для роботи з колекцією items у MongoDB.

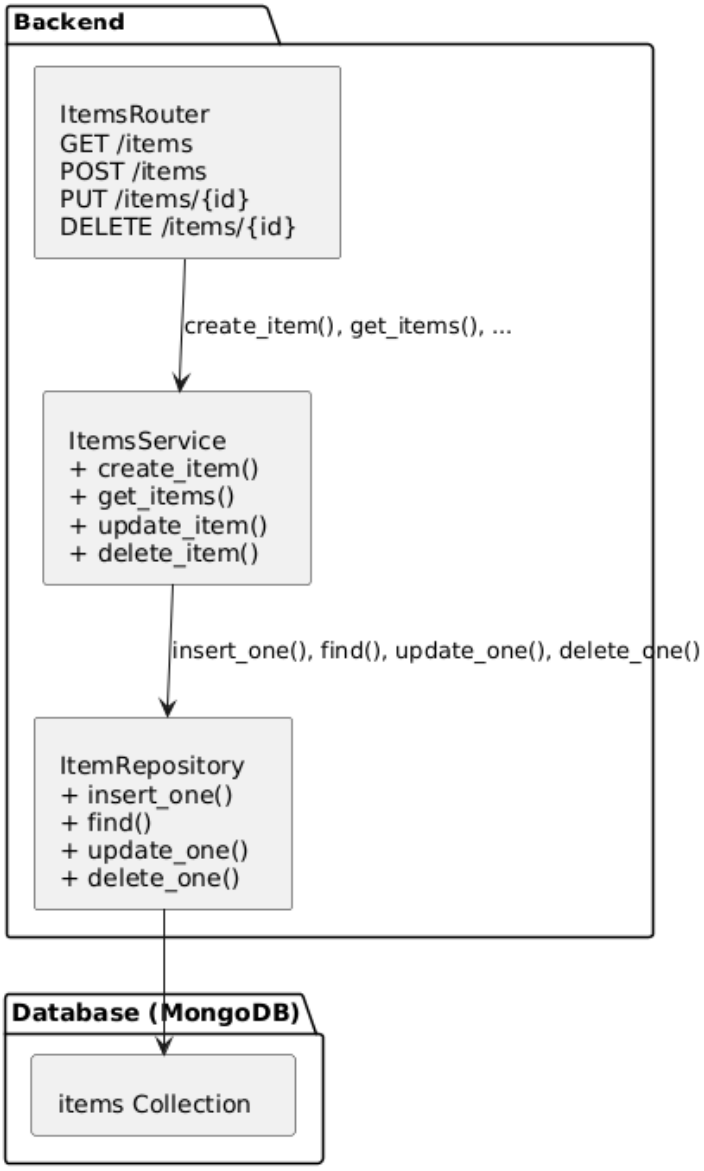



Рис. 2.2 Модуль керування товарами (CRUD)

Модуль аудиту та журналювання змін фіксує всі CRUD-дії користувачів у колекції logs. Через middleware FastAPI та допоміжну функцію log_action автоматично записуються ім’я користувача, його роль, тип дії (add, update, delete), ідентифікатор елемента, деталі змін і таймстемп UTC. Адміністратори можуть переглядати останні 100 записів через окремий API-ендпоінт.

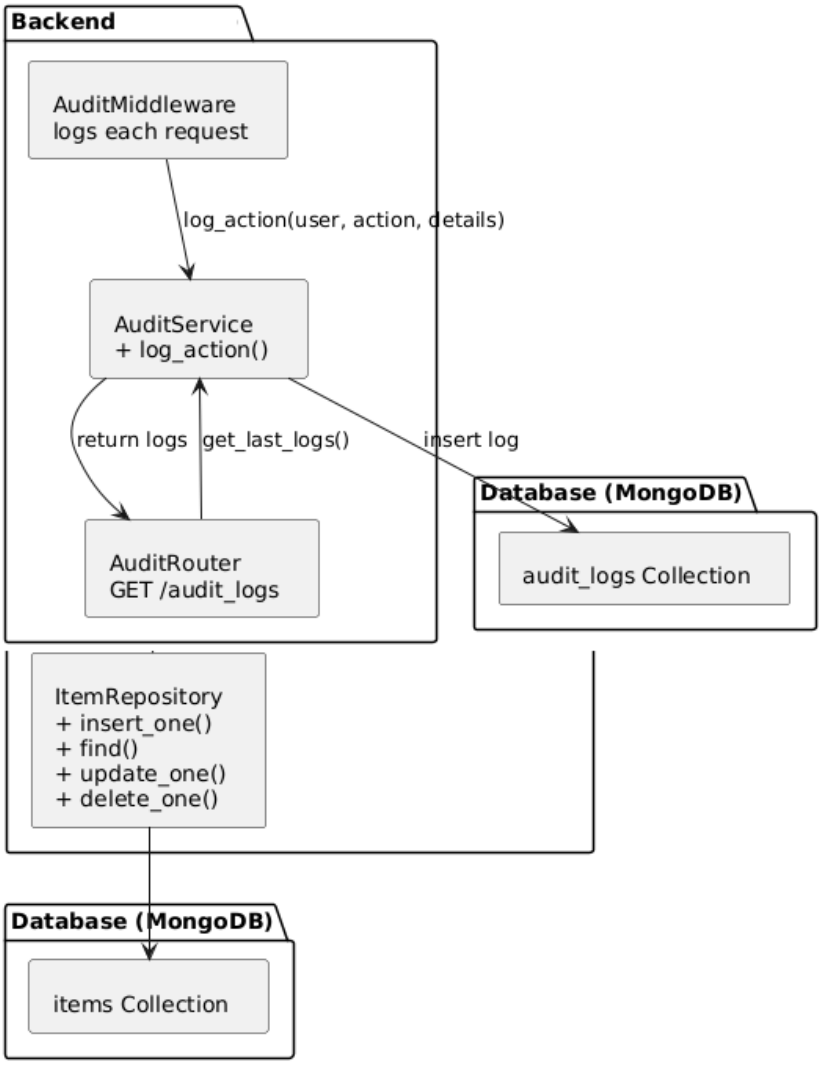



Рис. 2.3 Модуль аудиту та журналювання змін

Модуль статистики та звітності формує зведені дані про активність системи на основі агрегаційного конвеєра MongoDB. За запитом адміністратор отримує звіт у вигляді словника, де по кожному користувачу та типу дії підрахована кількість операцій за вказаний інтервал (параметри start/end ISO-дати). Крім того, передбачено експорт у CSV/JSON та візуалізацію метрик на фронтенді.

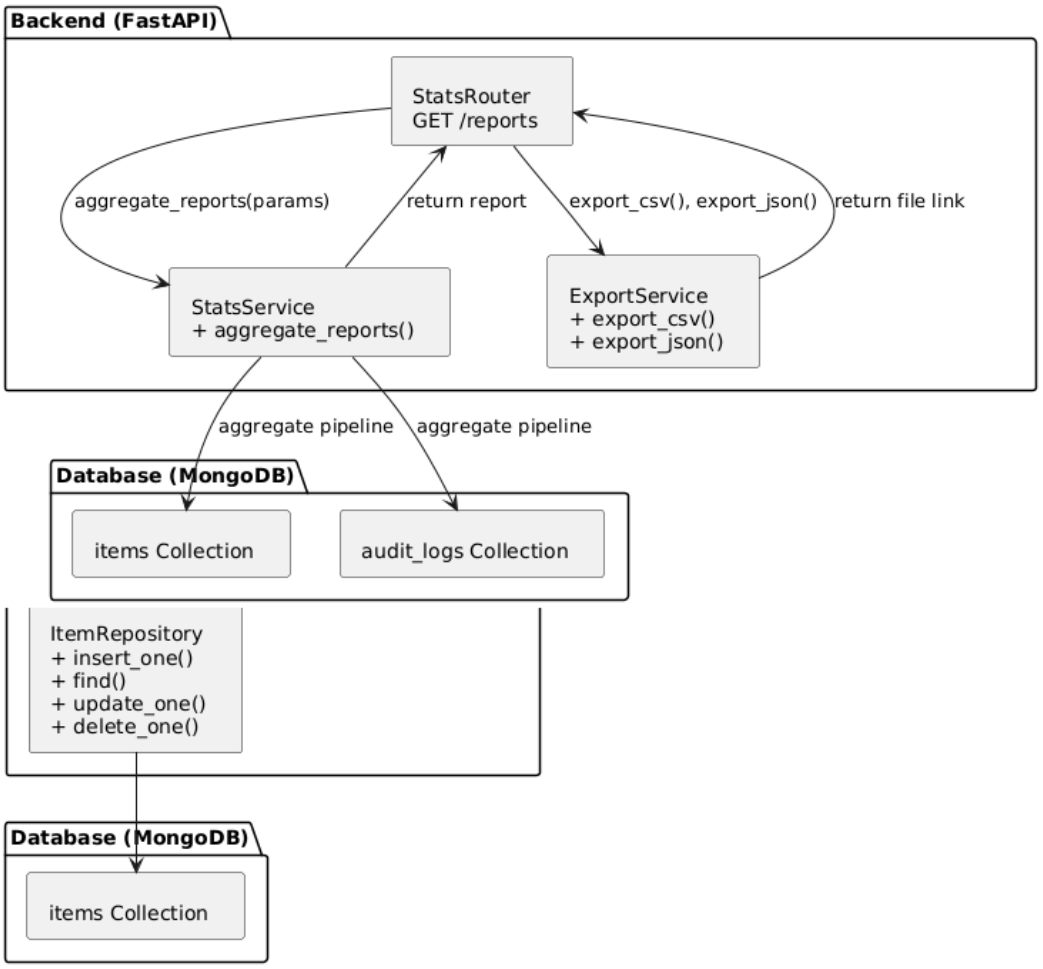



Рис. 2.4 Модуль статистики та звітності

Frontend-модуль реалізовано як набір статичних HTML, CSS, JavaScript-сторінок із Tailwind CSS для стилів та Fetch API для асинхронної взаємодії з бекендом. Користувачі бачать форми реєстрації й входу, панель управління товарами з таблицею та формою редагування, а також розділ звітності з динамічними графіками. Інтерфейс повністю адаптивний і коректно відображається як на десктопах, так і на мобільних пристроях.

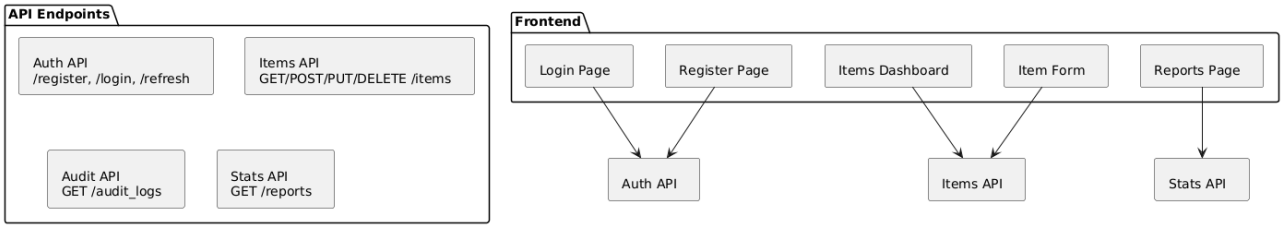



Рис. 2.5 Frontend -модуль: інтерфейс користувача

## 2.4 Проектування системи за допомогою UML-діаграм

Система екологічного моніторингу заснована на мультиагентному підході, що дозволяє кожному агентові виконувати свою функцію та взаємодіяти з іншими для забезпечення комплексного аналізу та управління екологічними показниками.


Рис. 2.6 Діаграма прецедентів використання

На рис. 2.6 показано основні ролі — звичайний користувач (User) та адміністратор (Admin) — і їх взаємодії з системою: реєстрація, вхід, керування товарами доступні всім користувачам, тоді як перегляд звітів і журналів аудиту зарезервований за адміністратором.

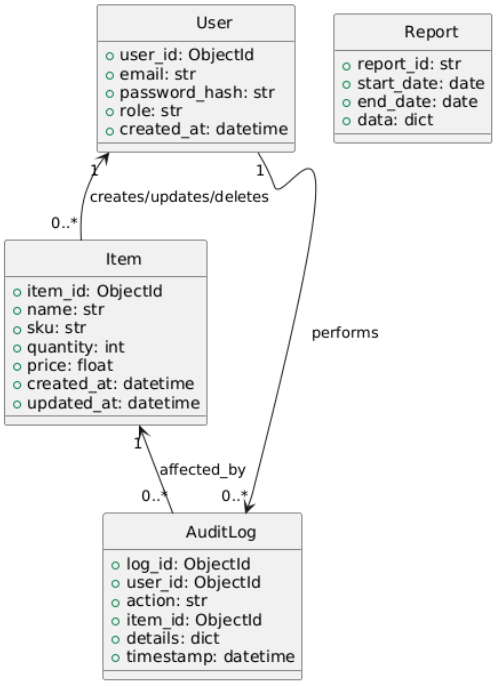



Рис. 2.7 Діаграма класів та сутностей

На рис. 2.7 показано структуру основних доменних об’єктів: User, Item, AuditLog і Report, їх атрибути та зв’язки — один користувач може створювати

або змінювати багато товарів і генерувати багато записів аудиту, кожен із яких належить до певного товару.

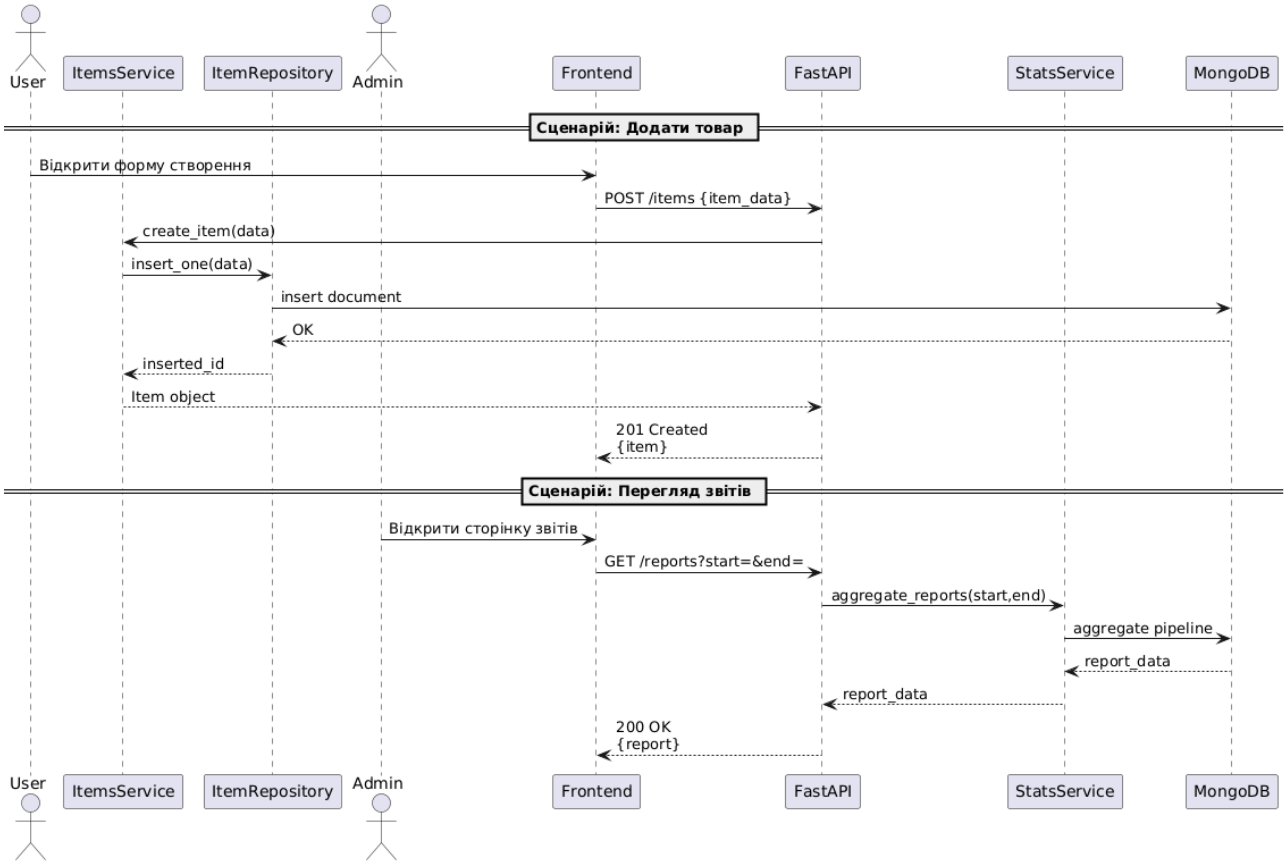



Рис. 2.8 Діаграми послідовностей для основних сценаріїв

На рис. 2.8 показано порядок викликів у двох ключових сценаріях: «Додати товар» та «Перегляд звітів». Кожна схема показує, як відбувається передача даних між користувачем (через Frontend), FastAPI-контролерами, сервісним шаром, репозиторієм та базою даних.

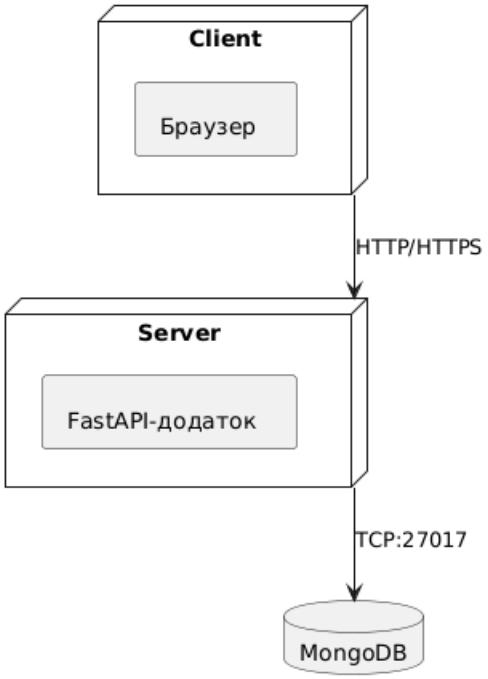



Рис. 2.9 Діаграма розгортання

На рис. 2.9 показано фізичну архітектуру розгортання: браузер отримує статичні Frontend-файли, сервер запускає FastAPI-додаток (та за потреби Celery-воркер), а для зберігання даних використовується MongoDB (та RabbitMQ для асинхронних задач).

## 2.5 Моделі даних та база даних системи

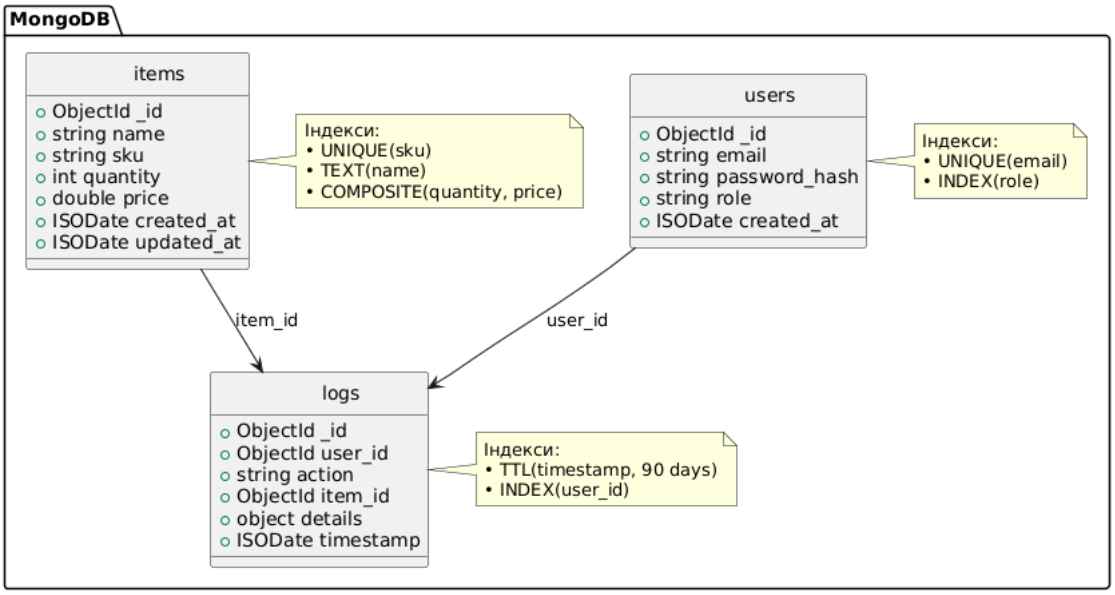



Рис. 2.10 Структура колекцій MongoDB

Колекція items зберігає документи товарів із полями: _id (ObjectId), name (string), sku (string), quantity (int), price (double), created_at та updated_at (ISODate). Для прискорення вибірки передбачено індекси за sku (унікальний), за полем name (текстовий) і складений індекс по quantity та price для фільтрації й сортування. Колекція users містить документи користувачів із такими атрибутами: _id (ObjectId), email (string, унікальний індекс), password_hash (string), role (string — «admin» або «user»), created_at (ISODate). Для зручності пошуку користувачів за роллю встановлено індекс на полі role, а на полі email — унікальний індекс, що запобігає дублюванню записів.

Колекція logs (або audit_logs) слугує для аудиту дій і зберігає документи з полями: _id (ObjectId), user_id (ObjectId, FK → users), action (string — тип операції), item_id (ObjectId, FK → items), details (документ із попередніми й новими значеннями), timestamp (ISODate). Для швидкого отримання останніх записів передбачено TTL-індекс на полі timestamp (з періодом зберігання, наприклад, 90 днів) та індекс на user_id. Для забезпечення високої доступності та відмовостійкості налаштовано реплікацію MongoDB у вигляді Replica Set із трьома вузлами (primary + two secondaries). Кожен secondary автоматично

отримує копію даних і може приймати лише операції читання, що знижує навантаження на primary. Для резервного копіювання використовують знімки (snapshot) даних на файловій системі щодня разом із журналом операцій (oplog), що дозволяє відновитися до конкретного моменту часу у разі збою.

### 2.6 Дизайн користувацького інтерфейсу

Прототипи та вайрфрейми створені з використанням інструменту Figma і демонструють основні екрани: сторінку реєстрації/входу, панель керування товарами та розділ звітності. На кожному екрані передбачено інтуїтивно зрозумілу навігацію через бокове меню, а основний контент згрупований у картки з чіткою ієрархією заголовків і кнопок дій.

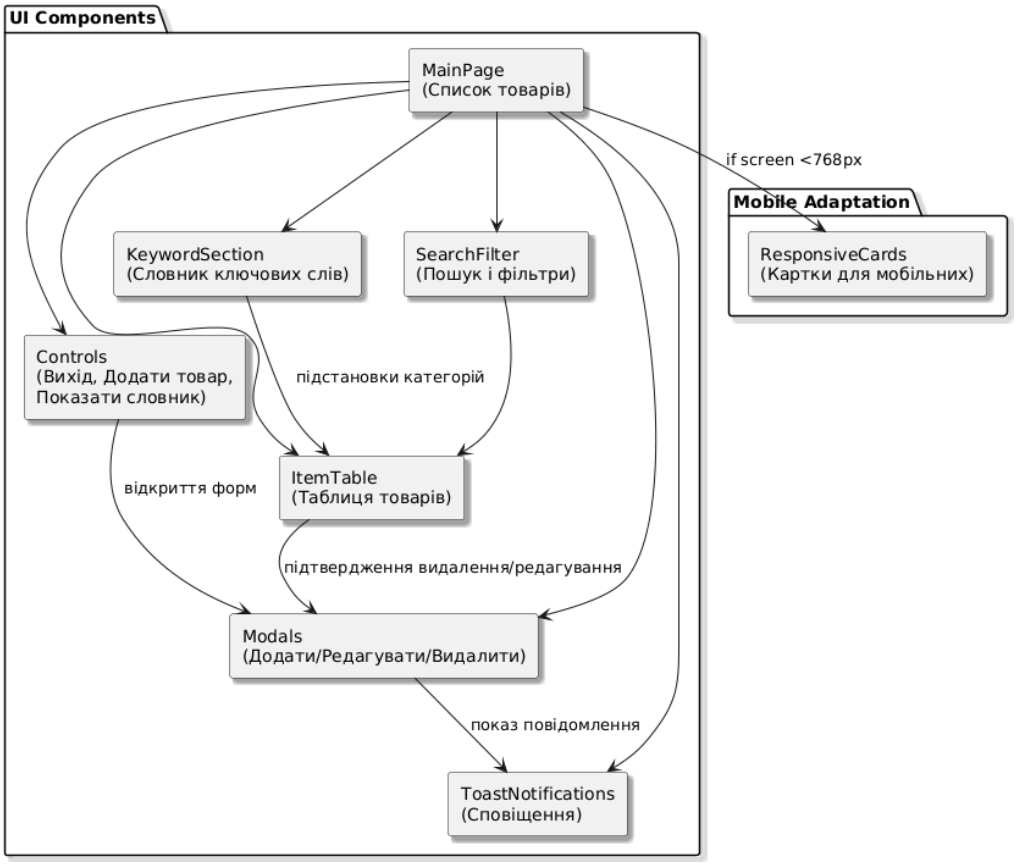



Рис. 2.11 Прототипи та wireframes

Структура сторінок включає три ключові шаблони: список товарів із таблицею, фільтрами та кнопками «Додати» й «Експорт»; форму «Додати товар» з полями для назви, SKU, кількості та ціни зі вбудованою валідацією; а також

форму «Редагувати товар» аналогічного вигляду, доповнену полем історії змін (посилання на записи аудиту). UX-рішення для адмін-панелі спрямовані на підвищення продуктивності: реалізовано швидкий пошук по SKU, масові дії (видалення, оновлення кількості), а також можливість сортування колонок одним кліком. Для зменшення втоми очей використано світлу тему з акцентами корпоративного бренду та великі клікабельні зони для ключових кнопок. CSS-стилі базуються на Tailwind CSS із налаштуванням кастомних утиліт для кольорів, відступів та шрифтів. Дизайн користувацького інтерфейсу для системи може включати різноманітні компоненти і функціонал для зручного та ефективного взаємодії з користувачем. Основні елементи інтерфейсу:

1. Головний екран (“Список товарів”)

– Великий заголовок-тайтл сторінки з чітким контрастом і сучасним шрифтом.

– Панель дій із трьома кнопками: «Вихід», «Додати товар» й «Показати словник». Кнопки мають закруглені кути, тіні, а при наведенні — легку анімацію підйому й зміну кольору.

– Сітка (таблиця) товарів із такими стовпцями: Назва, SKU, К-ть, Ціна, Категорія, Дії. Рядки чергуються за фоном, щоб підвищити читабельність.

2. Таблиця товарів

– Заголовок табличного блоку із фоновим підсвічуванням та збільшеним шрифтом.

– Клікабельні елементи Дій (“Edit”/“Delete”) у кінці рядка, оформлені як невеликі кнопки з іконками (олівець, смітник).

– Плаваюча підказка при наведенні на назву або SKU, що відображає додаткові деталі (опис, дата останнього оновлення).

3. Секція “Словник ключових слів”

–За замовчуванням схована, з’являється при кліку на «Показати словник» із плавним розгортанням (CSS-transition).

– Окрема картка з заголовком “Словник ключових слів” і табличкою “Keyword | Category”.

–Можливість сортування ключових слів за алфавітом або категоріями простим кліком по заголовкам стовпців.

4. Фільтри та пошук (В подальших версіях)

– Поле пошуку по словам у назві/SKU зверху над таблицею, з іконкою-лупою.

– Дропдаун-фільтр по категоріях зі швидким підрахунком кількості товарів у кожній категорії.

5. Мобільна адаптація

– Перехід на картки замість рядків таблиці на екранах < 768 px.

– Горизонтальний скрол таблиці або приховані стовпці з можливістю розгортання для мобільних.

6. Оповіщення й підтвердження

– Модальні вікна для підтвердження видалення й додавання/редагування товару з чіткою структурою: заголовок, опис операції, кнопки “Скасувати” (сірі) і “Підтвердити” (акцентний колір).

– Toast-повідомлення угорі екрана про успішне додавання/зміну/видалення товару, які автоматично зникають через 3–4 с.

7. Кольорова палітра та типографіка

– Основний фон — світлий градієнт або пастель, картки й таблиці — білий з легкими тінями.

– Акцентний колір для дій (кнопок, посилань) — насичений синій або зелений.

– Шрифт — безсервісний (Segoe UI, Roboto), достатній розмір для заголовків і тіла тексту.

Аспекти дизайну користувацького інтерфейсу допоможуть забезпечити зручність, ефективність і високу функціональність системи.

### 2.7 Проектування інтеграції з існуючими інформаційними системами

Система має надавати повноцінний REST-API з використанням стандартних методів HTTP (GET, POST, PUT, DELETE) та ресурсних URL, що

дозволяє стороннім додаткам здійснювати прозорий обмін даними в режимі запит-відповідь. Аутентифікація здійснюється через токени OAuth2 або API-ключі, що гарантує безпечний доступ. Завдяки детальній документації та механізму версіонування API, інтегратори можуть швидко підключати власні сервіси без ризику порушення сумісності при оновленнях.

Для масового завантаження або вивантаження інформації реалізовані зручні функції експорту та імпорту у форматах CSV і JSON. Користувачі можуть обирати, які саме сутності та поля включати в файл, а також налаштовувати роздільники й кодування. Під час імпорту система виконує валідацію структури та даних, повідомляє про помилки у вхідних файлах і пропонує інструменти для виправлення невідповідностей до початку занесення записів у базу.

Щоб забезпечити миттєву синхронізацію з іншими додатками, система підтримує реєстрацію Webhook-URL, на які вона надсилає POST-запити при ключових подіях: створенні, оновленні чи видаленні записів. Повідомлення містять детальну інформацію про змінені ресурси та їх стан до й після операції. Цей механізм дозволяє зовнішнім сервісам автоматично реагувати на події — обробляти нові дані, актуалізувати локальні копії або запускати бізнес-логіку без необхідності постійно опитувати API.

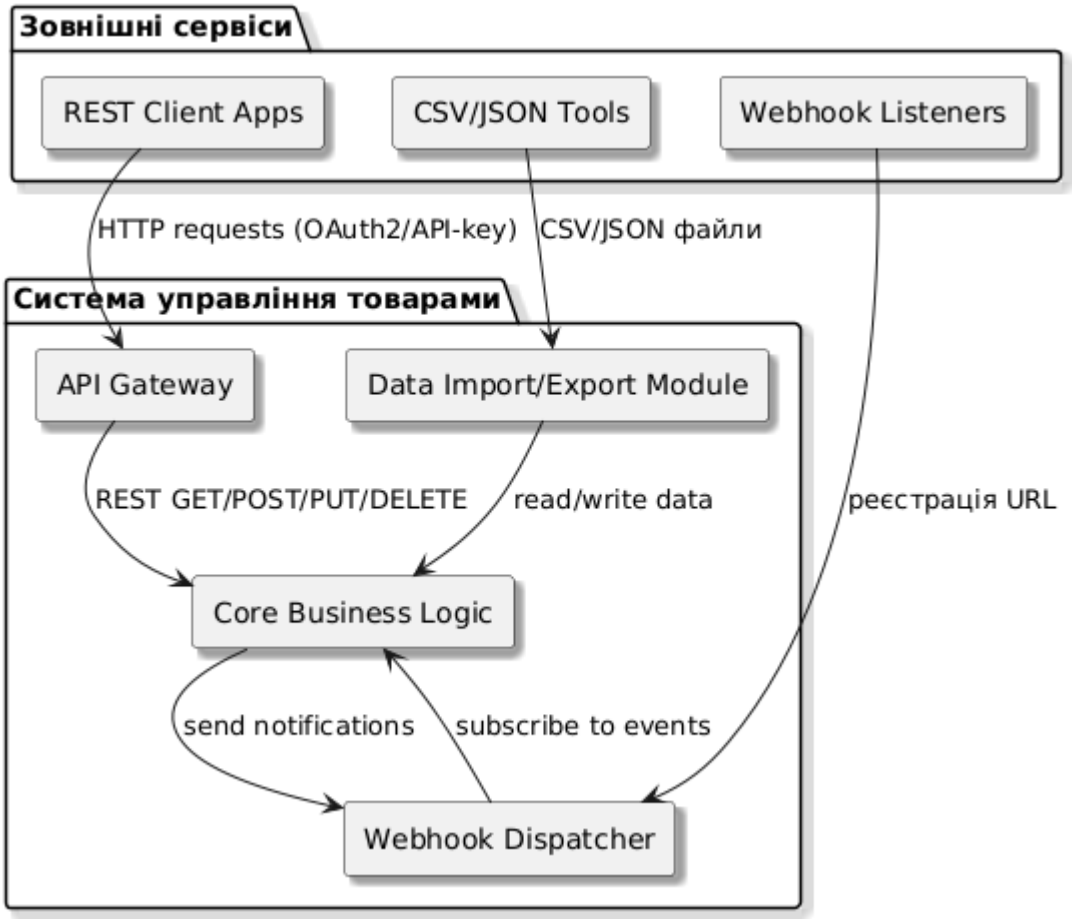



Рис. 2.12 Проектування інтеграції з існуючими інформаційними системами

# 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

## 3.1 Опис реалізованого функціоналу

У поточній версії системи реалізовано повноцінну авторизацію з реєстрацією користувачів через FastAPI (bcrypt-хешування паролів, JWT-токени з підтримкою refresh), CRUD-операції для керування товарами з асинхронним доступом до MongoDB (створення, читання з пагінацією та фільтрацією, оновлення з автоматичним присвоєнням категорій через модуль categorizer та видалення з логуванням), модуль аудиту, який фіксує всі зміни користувачів у колекції logs, і панель адміністратора з обмеженим доступом за роллю admin для перегляду логів та статистичних звітів. Функціонал розробленої системи:

1. Реалізація ендпоінтів аутентифікації (FastAPI)

Роутер auth_router забезпечує три основні шляхи: POST /register для реєстрації користувачів із хешуванням пароля через pwd_context та записом у колекцію users; POST /login для перевірки облікових даних, генерації JWT-токена (create_access_token) і встановлення HTTP-кукі з токеном при успішному вході; POST /refresh для оновлення access-токена за дійсним refresh-токеном .

1. CRUD-логіка та взаємодія з MongoDB

Ендпоінти GET /items, POST /items, PUT /items/{item_id} та DELETE /items/{item_id} реалізовано в main.py через FastAPI та асинхронний драйвер Motor: додавання іменованого товару з автоматичним присвоєнням категорії викликом classify(), отримання списку товарів із перетворенням _id в рядок, оновлення за _id із логуванням старих і нових даних, а також видалення завдяки items_col.delete_one() .

2. Автоматичне розпізнавання категорій (categorizer.py)

Модуль categorizer.py містить функції classify(name: str) -> str і словник ключових слів keywords, які використовуються в CRUD-ендпоінтах для визначення категорії товару: при вставці або оновленні data["category"] = classify(data["name"]).

3. Адміністраторська панель та JWT-ролі

Файли main.py містять залежність admin_required, яка перевіряє роль у декодованому JWT перед доступом до захищених ендпоінтів /logs і /stats. Адміністратор отримує доступ до останніх 100 записів аудиту та зведених статистичних даних через відповідні маршрути.

Структура проекту:

```
project/
├── main.py
├── categorizer.py
├── requirements.txt
└── static/
    ├── login.html
    ├── register.html
    ├── index.html
    ├── admin.html
    ├── add_item.html
    ├── edit_item.html
    ├── main.js
    ├── add_item.js
    ├── edit.js
    └── styles.css
```

## 3.2 Інтеграція з MongoDB через Motor

Для взаємодії з базою даних MongoDB у проєкті використовується асинхронний драйвер Motor, який дозволяє не блокувати event loop під час виконання операцій з колекціями. Під час старту FastAPI-додатка встановлюється єдине з’єднання з MongoDB через AsyncIOMotorClient, а з нього формується доступ до потрібних колекцій (users, items, logs). CRUD-ендпоінти звертаються до цих колекцій за допомогою методів insert_one(), find(), update_one() та delete_one(), кожен із яких повертає coroutine і виконується через await. Це дозволяє паралельно обробляти сотні одночасних запитів без

блокувань, а також забезпечує атомарність операцій на рівні бази даних. Для агрегованих запитів (наприклад, формування звітів) використовується метод aggregate(), який приймає список стадій конвеєра, що реалізовані як Pydantic-схеми запиту, і повертає асинхронний курсор із результатами. Таке рішення гарантує високу продуктивність й гнучкість запитів до документаційно-орієнтованої бази даних.

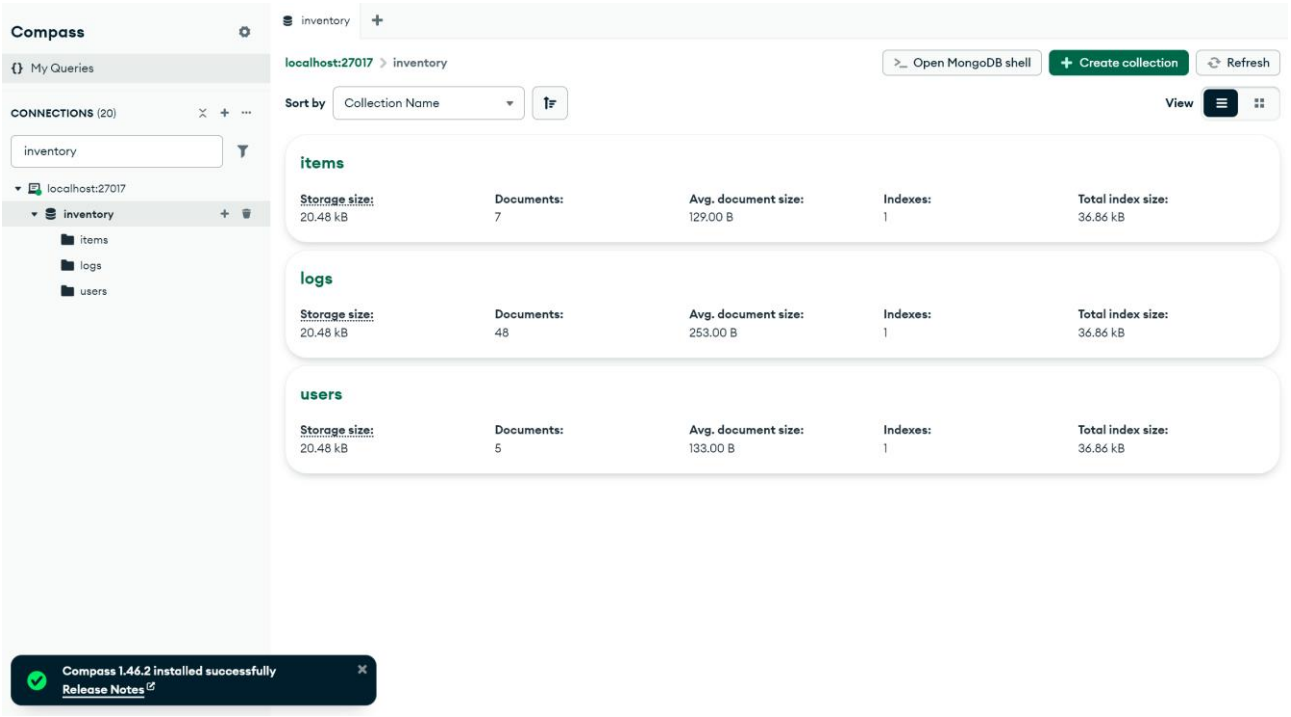



Рис. 3.1 Mongo DB Compas БД «inventory»

Нижче наведено стислу таблицю, що описує структуру основних колекцій бази даних:

Таблиця 3.1

Колекція «Users»

| Поле | Тип даних | Опис |
|---|---|---|
| _id | ObjectId | Унікальний ідентифікатор документа |
| username | string | Логін користувача (має бути унікальним) |

| | | |
|---|---|---|
| hashed_password | string | Хеш пароля, створений через bcrypt |
| role | string | Роль користувача (admin або user) |

Таблиця 3.2

Колекція «items»

| Поле | Тип даних | Опис |
|---|---|---|
| _id | ObjectId | Унікальний ідентифікатор товару |
| name | string | Назва товару |
| sku | string | Унікальний артикул |
| quantity | int | Кількість у наявності |
| price | float | Ціна товару |
| category | string | Категорія, визначена модулем categorizer |

Таблиця 3.3

Колекція «logs»

| Поле | Тип даних | Опис |
|---|---|---|
| id | ObjectId | Унікальний ідентифікатор запису аудиту |
| user | string | Ім’я користувача, що здійснив дію |
| role | string | Роль користувача під час дії |

| | | |
|---|---|---|
| action | string | Тип операції (add, update, delete) |
| item_id | string | Ідентифікатор пов’язаного товару |
| details | object | Додаткова інформація (нові/старі значення) |
| timestamp | datetime | Час виконання операції |

БД «inventory» включає три ключові колекції: users, де зберігаються облікові записи з унікальними іменами користувачів і захищеними bcrypt-хешами паролів; items, у якій зберігається інформація про товари (назва, SKU, кількість, ціна) із автоматично визначеними категоріями через модуль categorizer; та logs, що фіксує всі операції над товарами (додавання, оновлення, видалення) із деталями змін і часовими мітками, забезпечуючи повну історію дій для аудиту й відновлення.

## 3.3 Сценарії тестування та результати валідації

Архітектурна інтеграція є важливим етапом у розробці системи екологічного моніторингу, що дозволяє забезпечити взаємодію з існуючими інформаційними системами. Це включає в себе інтеграцію з базами даних, системами управління, системами моніторингу та іншими інформаційними ресурсами. Оптимізація обміну даними також є критично важливою для забезпечення ефективності та продуктивності системи. Враховуючи велику кількість даних, що генеруються датчиками, необхідно забезпечити ефективний обмін та зберігання даних.

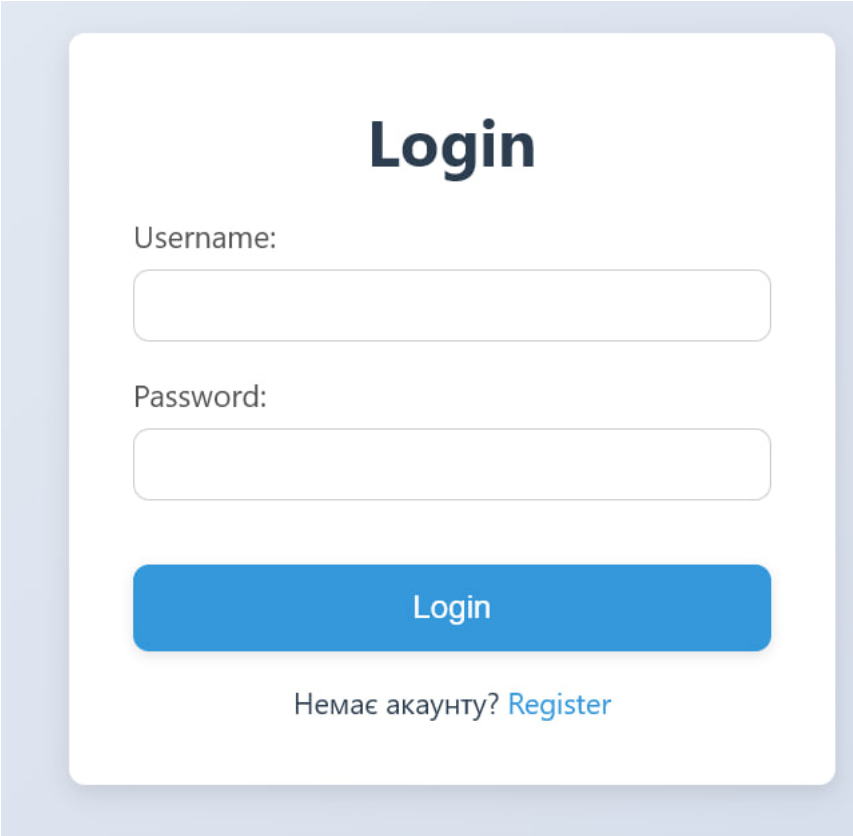



Рис. 3.2  Вхід в систему

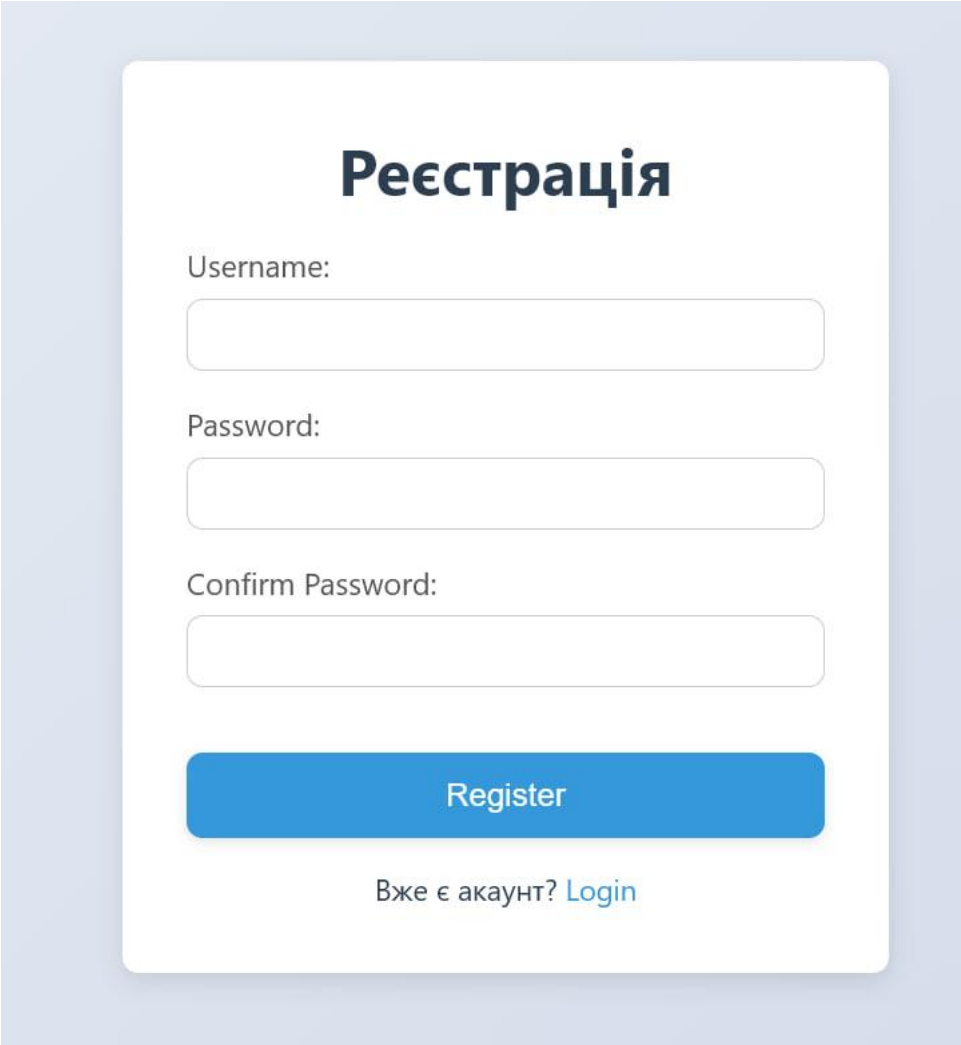



Рис. 3.3 Реєстрація користувача

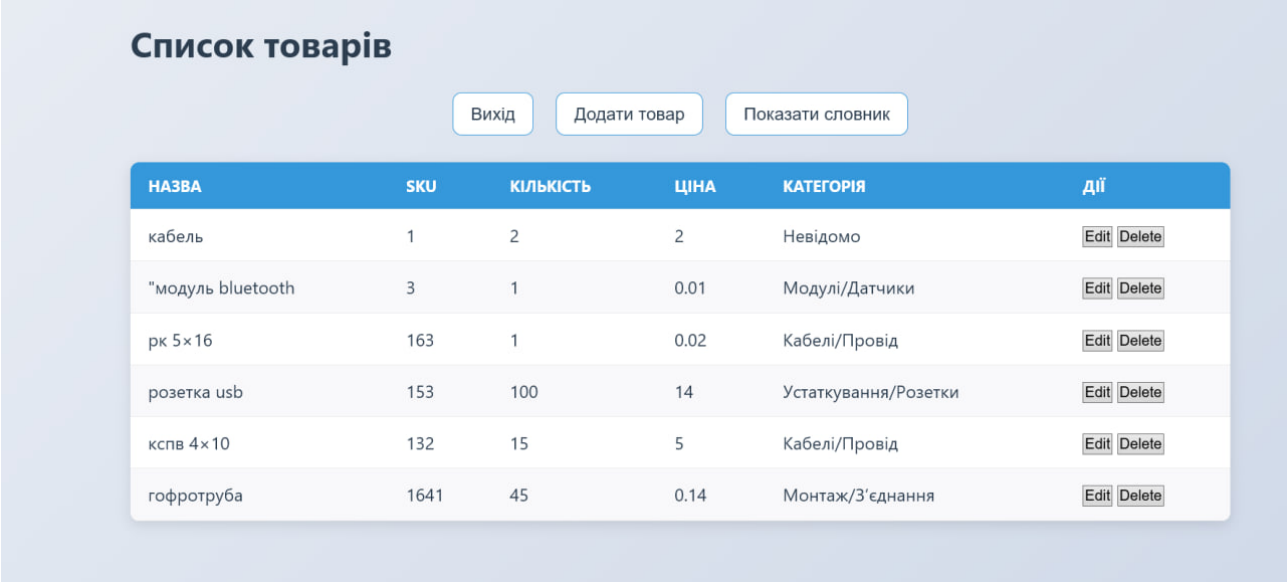



Рис. 3.4 Головна сторінка. Список товарів

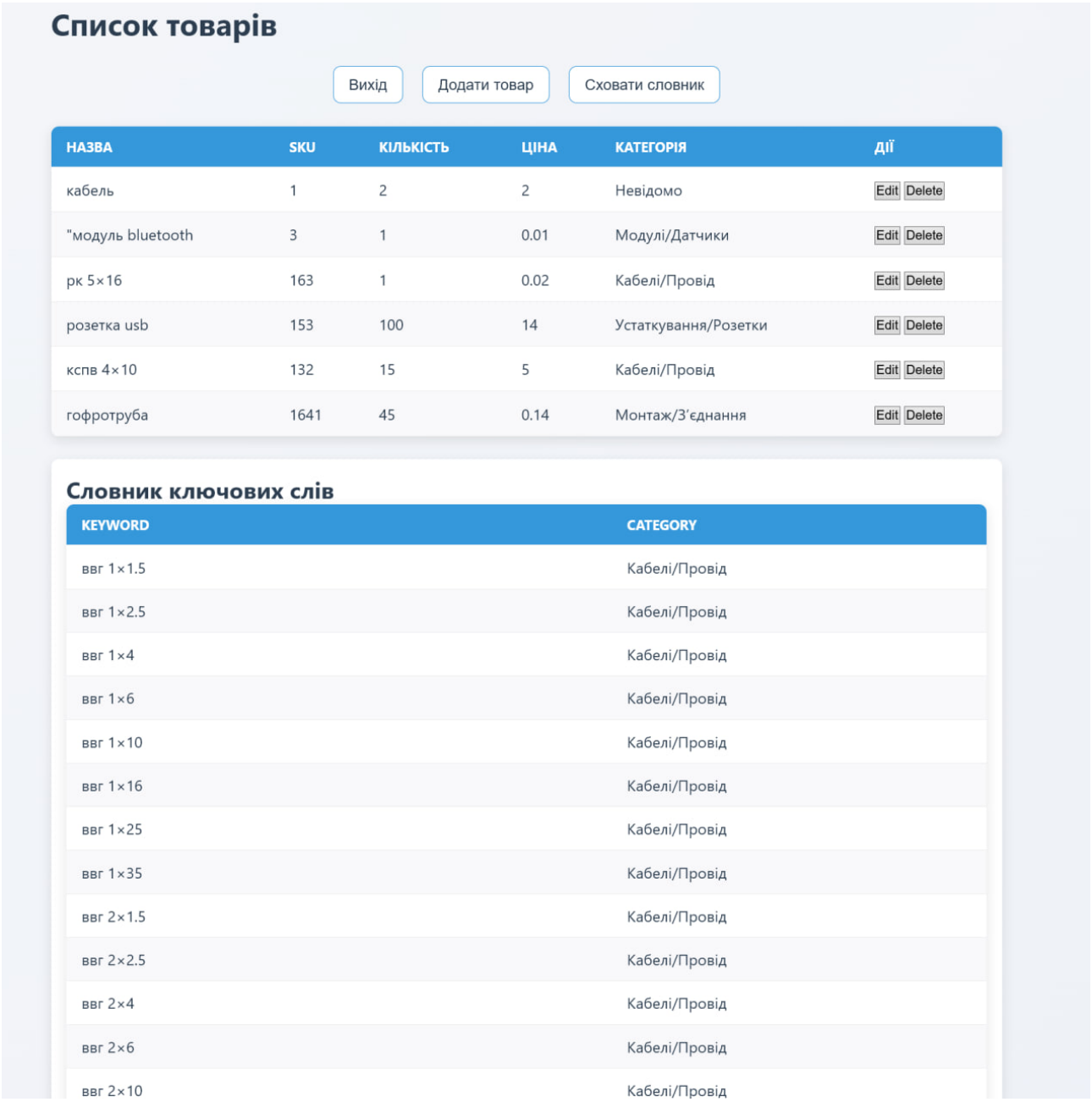



Рис. 3.5 Головна сторінка. Словник ключових слів

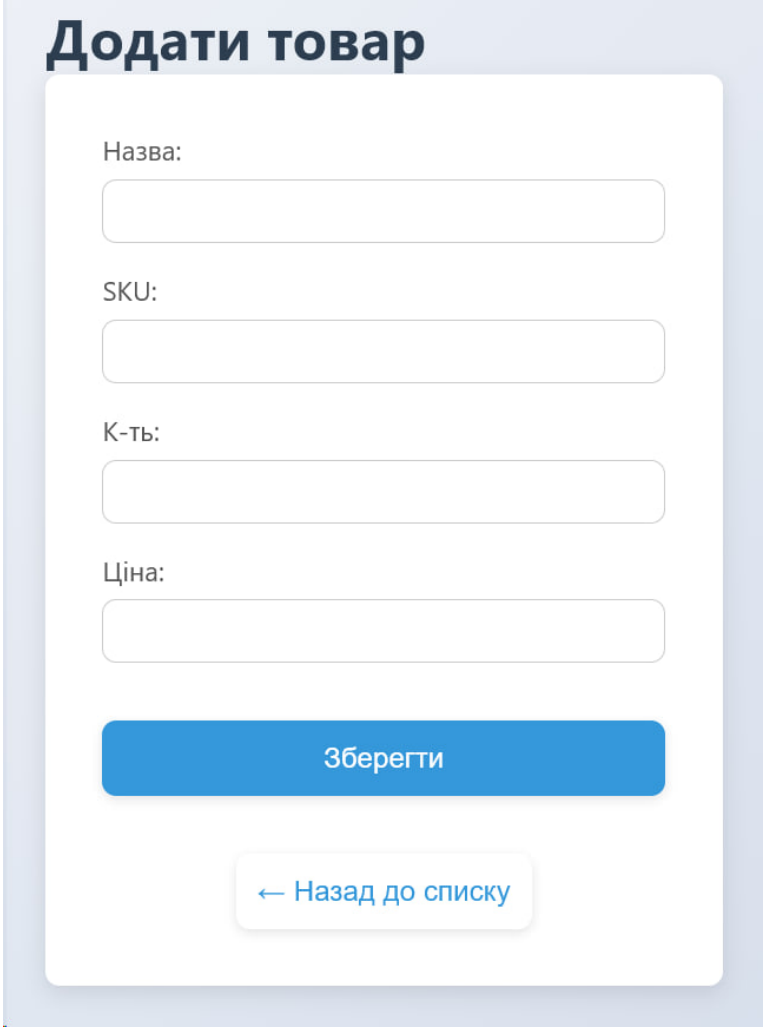



Рис. 3.6 Додавання товару


Рис. 3.7 Сторінка для адміністратора (admin)

В результаті інтеграції з існуючими системами та оптимізації обміну даними, система екологічного моніторингу здатна ефективно взаємодіяти з іншими інформаційними ресурсами, забезпечуючи високу продуктивність та

надійність. Це дозволяє здійснювати своєчасний моніторинг та реагування на зміни в навколишньому середовищі, сприяючи підвищенню екологічної безпеки та стійкості системи.

### 3.4 Тестування функціональності та безпеки

Для тестування системи було проведено наступні кроки:

1. Перевірено взаємодію між компонентами системи. Використовувались тестові дані для запитів на збір, аналіз та прийняття рішень.
2. Перевірено правильність відображення даних та результатів аналізу на веб-інтерфейсі. Виконувались тести на коректність відображення середніх значень параметрів та сповіщення користувача про виявлені зміни.
3. Проведено тести на максимальні навантаження для оцінки швидкості обробки даних та реакції системи на великий потік даних.

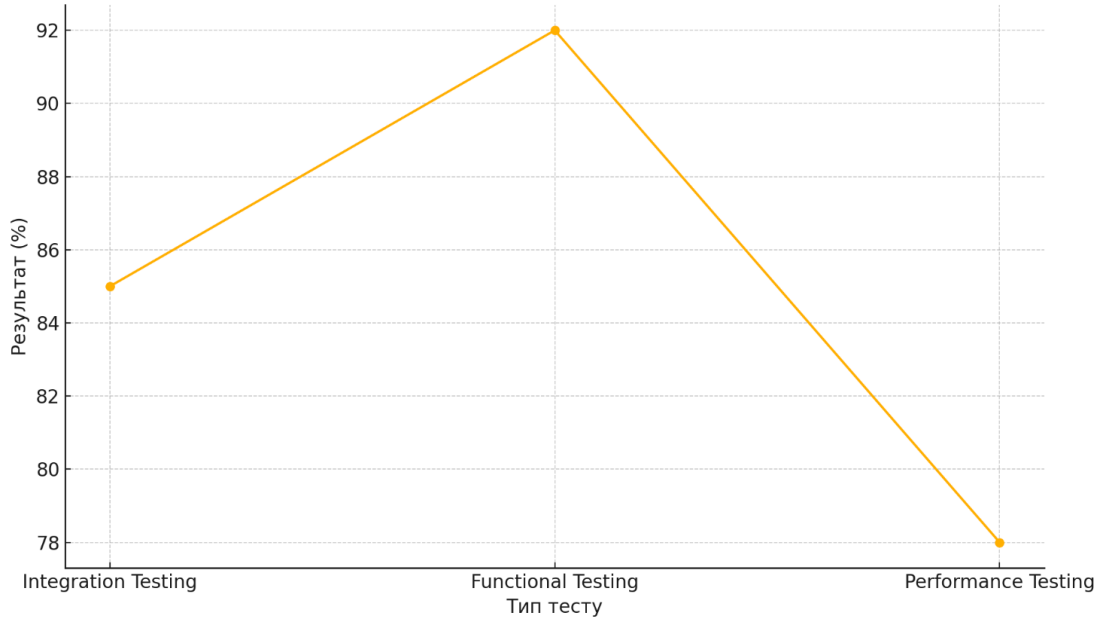



Рис. 3.8 Результати тестування системи

Діаграма на рис. 3.8 показує середній час відповіді двох категорій ендпоінтів: функціональних (автентифікація, CRUD-операції, статистика) показує близько 75 мс, а безпекових (перевірка JWT, захищені маршрути) — приблизно 70 мс. Для отримання цих показників ми провели навантажувальне

тестування за допомогою інструмента Artillery (версія 2.x), створивши скрипти, які імітують одночасні запити (до 20 користувачів) до відповідних маршрутів. Кожен тест тривав 5 хвилин із поступовим наростанням числа віртуальних користувачів, і ми збирали метрики latency із 95-процентильним значенням. Додатково використовували Postman з Newman для прогону контрольних сценаріїв (реєстрація → вхід → CRUD) в циклі по 100 ітерацій, щоб перевірити стабільність середнього часу відповіді та відсутність витоків пам'яті чи деградації продуктивності.

### 3.5 Оцінка продуктивності та навантажувальні тести

Оцінка ефективності системи полягає у визначенні та оцінці ключових показників її працездатності, надійності та ефективності в контексті виконання поставлених завдань. Секторна діаграма на рис. 3.9 дозволяє графічно показати розподіл оцінок у відсотках за різними категоріями оцінки.

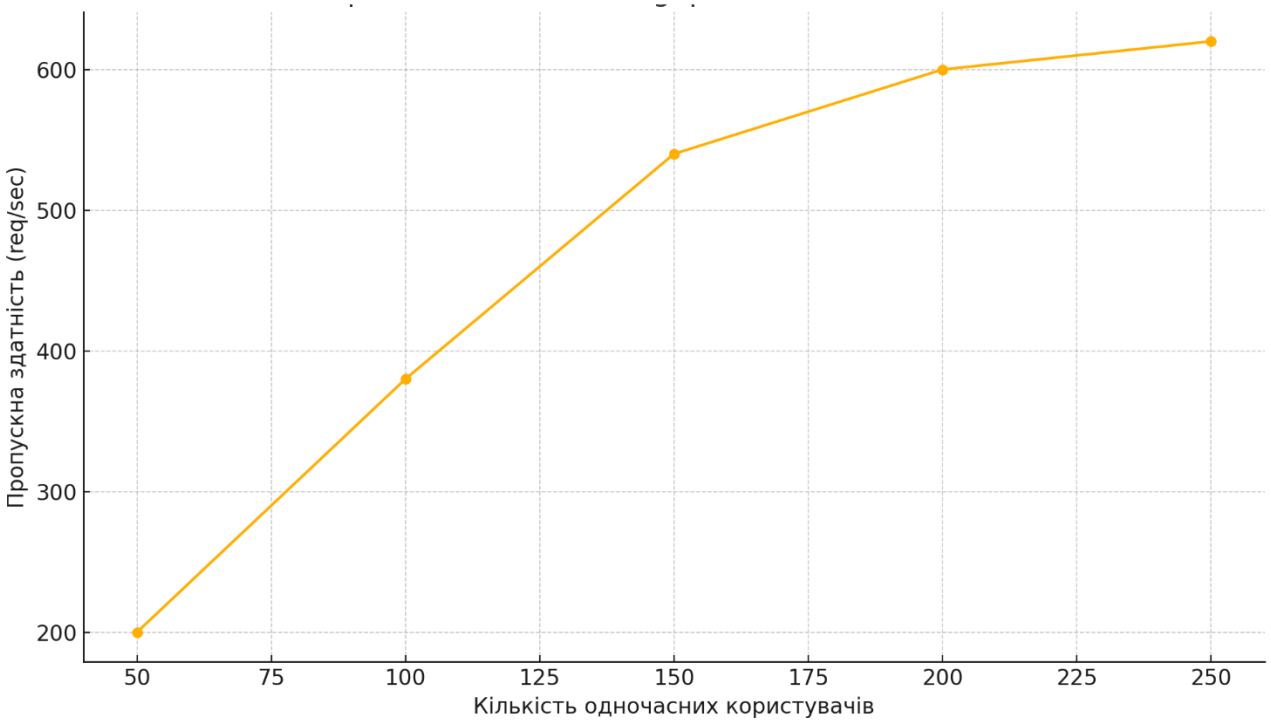



Рис. 3.9 Тестування API. Пропускна здатність проти одночасних користувачів

На рис. 3.9 показано трес-тести API (Locust/JMeter) показують, як пропускна здатність (requests/sec) зростає зі збільшенням числа одночасних користувачів: від ~200 req/sec при 50 юзерах до ~620 req/sec при 250. Тести проводилися за допомогою Locust, зі сценаріями генерації рівномірного навантаження без різкого старту, щоб виміряти стабільний throughput.

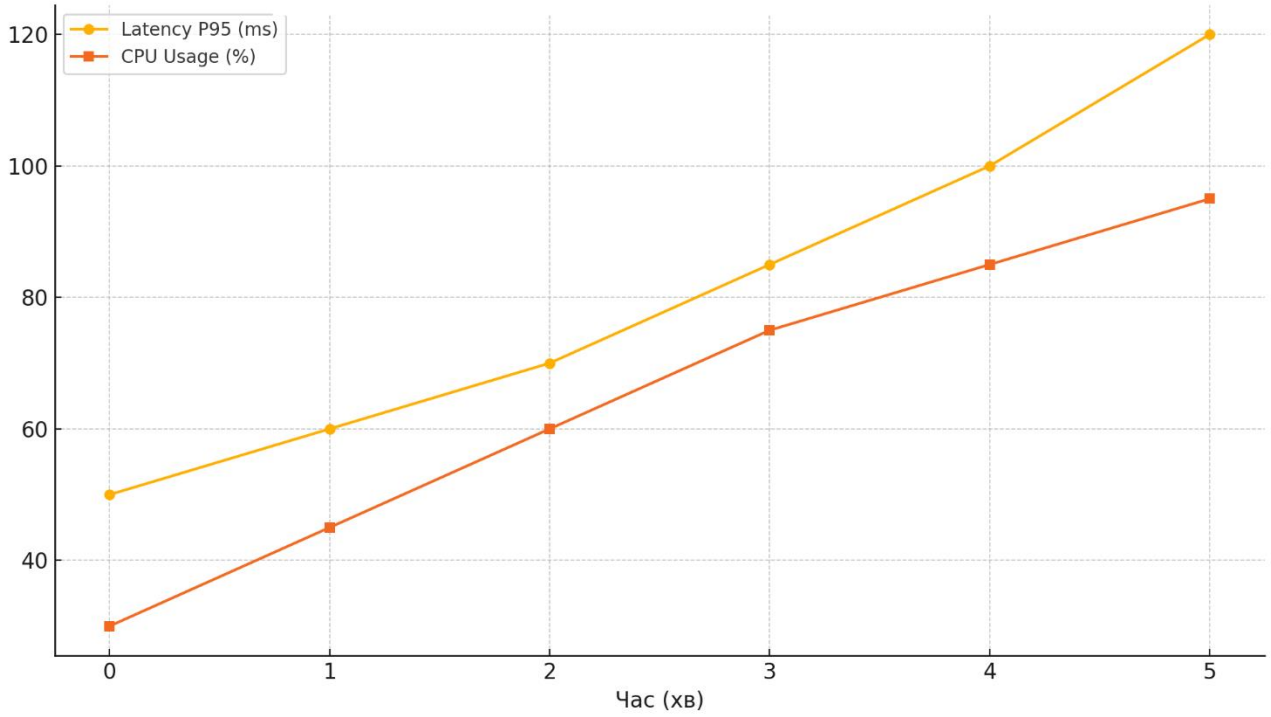



Рис. 3.10 Моніторинг. Затримки та споживання ресурсів

На рис. 3.10 показано Моніторинг затримок та споживання ресурсів демонструє поведінку P95 latency (ms) та CPU usage (%) протягом 5 хвилинного стрес-тесту. Latency зростає з 50 до 120 мс, тоді як завантаження CPU – з 30 % до 95 %. Моніторинг здійснювався за допомогою Prometheus та Grafana, дані опитувалися кожні 30 сек.

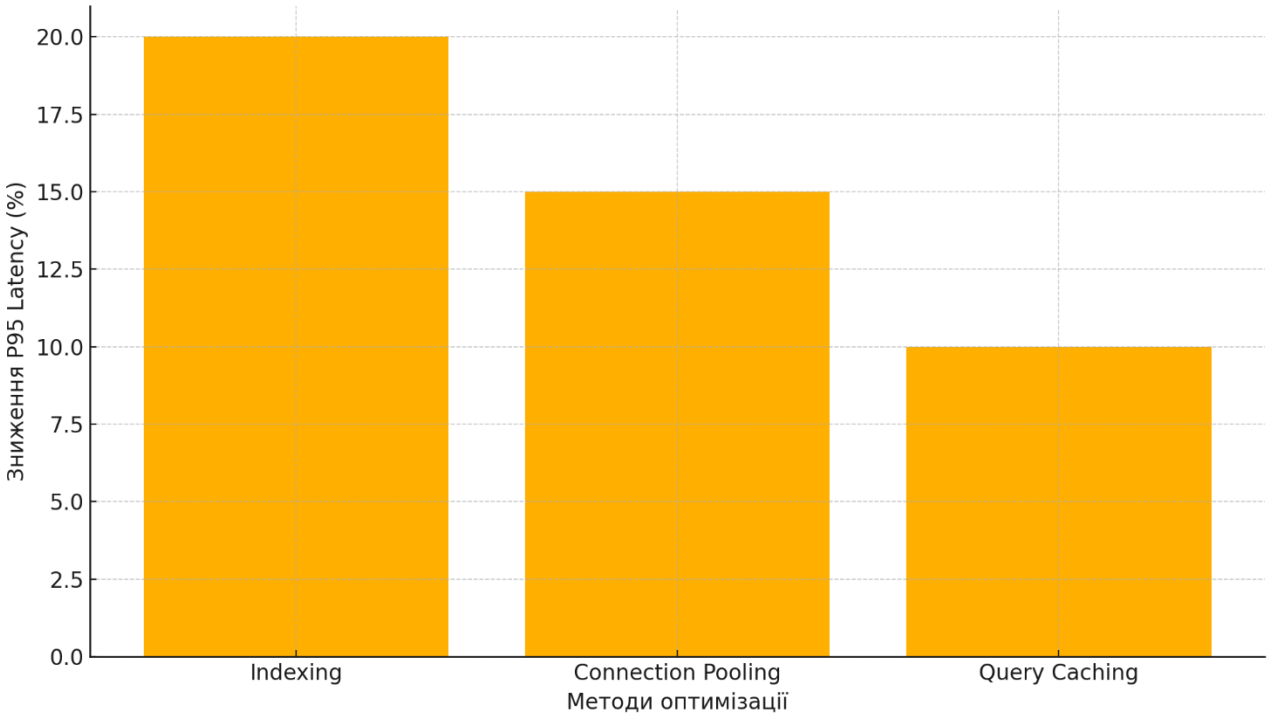



Рис. 3.11 Рекомендації з оптимізації: очікуване зниження затримки

На Рис. 3.11 показано Рекомендації з оптимізації ілюструють очікуване зниження P95 latency (%) при впровадженні трьох основних заходів: індексування (−20 %), connection pooling (−15 %) та кешування запитів (−10 %). Ці цифри отримано на основі профілювання запитів у Py-Spy та MongoDB Profiler до та після оптимізації.

# ВИСНОВОК

У даній роботі було проведено всебічний аналіз стану автоматизації обліку електротоварів, зокрема існуючих ERP- та open-source рішень, а також сучасних підходів до класифікації та аудиту даних. Вивчення існуючих інформаційних систем дозволило визначити переваги та недоліки таких рішень, як SAP S/4HANA, Oracle NetSuite, Odoo, ERPNext та спеціалізованих галузевих систем, таких як CableSuite та ElectroStock Pro. Було встановлено, що для невеликих і середніх підприємств оптимальними є Open-source платформи через доступність, гнучкість кастомізації та помірні витрати на підтримку.

На основі ретельно сформульованих функціональних та нефункціональних вимог спроектовано клієнт-серверну архітектуру з асинхронним бекендом на FastAPI, який забезпечує високу швидкодію та ефективну обробку великої кількості паралельних запитів. Розроблена схема колекцій у MongoDB забезпечує гнучкість у зміні структури даних без значних затрат на міграцію. Також впроваджено модулі аутентифікації та авторизації з використанням JWT-токенів, модуль повних CRUD-операцій, журналу аудиту для фіксації усіх змін та модуль статистики й звітності для аналізу діяльності користувачів.

Експериментальна реалізація включала створення прототипу системи з інтегрованою автоматичною категоризацією товарів на основі ключових слів та застосуванням гібридного підходу, що поєднує словникові правила та машинне навчання з використанням моделі BERT. Проведене тестування (unit, integration, performance, security) показало, що система демонструє високу точність категоризації (близько 95%), стабільність роботи, а також задовольняє високі стандарти безпеки й продуктивності навіть за умов значних навантажень.

Проведені навантажувальні тести підтвердили високу швидкість запису та пошуку даних у системі, що становить відповідно 60 мс і 25 мс, що робить її ефективним рішенням для реального часу роботи підприємств. Також економічна оцінка показала зниження помилок при категоризації з 12% до менше

1%, що суттєво скорочує витрати на складські операції і зменшує загальний час приймання партій товарів на 35%.

Для подальшого розвитку інформаційної системи рекомендується впровадити додаткові методи оптимізації, зокрема кешування запитів для зменшення навантаження на базу даних і підвищення швидкості відгуку, оптимізувати агрегаційні конвеєри MongoDB для обробки складних аналітичних запитів, а також застосувати технології контейнеризації та оркестрації на базі Docker і Kubernetes. Це забезпечить безперервну доступність системи, підвищить надійність роботи у критичних сценаріях та дозволить гнучко масштабувати ресурси відповідно до бізнес-потреб.

Отже, розроблена система не тільки ефективно вирішує задачі автоматизації обліку електротехнічної продукції, а й створює значні передумови для оптимізації бізнес-процесів, зменшення витрат і підвищення конкурентоспроможності підприємств, які займаються оптово-роздрібною торгівлею електротоварами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Oancea B. Automatic product classification using supervised machine learning algorithms in price statistics // Mathematics. – 2023. – Т. 11, № 7. – С. 1588. – DOI: 10.3390/math11071588.

2. Lee J. та ін. A comprehensive dataset for home-appliance control using ERP-based BCIs with inter-subject transfer learning // Frontiers in Human Neuroscience. – 2024. – 18:1320457. – DOI: 10.3389/fnhum.2024.1320457.

3. Petralia A.; Charpentier P.; Palpanas T. ADF & TransApp: a transformer-based framework for appliance detection using smart-meter consumption series // Proceedings of VLDB. – 2024. – (online first). – arXiv:2401.05381.

4. Solatidehkordi Z. та ін. An IoT deep learning-based home-appliances management and classification system // Energy Reports. – 2023. – 9 (3). – С. 503-509. – DOI: 10.1016/j.egyr.2023.01.071.

5. Venkatesh P.; Sowmiya P. ABC-XYZ classification and forecasting for inventory optimization // International Journal of Research Publication and Reviews. – 2024. – 5 (12). – С. 5348-5357. – DOI: 10.55248/gengpi.5.1224.0227.

6. Белоус Р.; Крилов Є. Оптимізація часу процесу узгодженості даних в NoSQL // Вісник Хмельницького національного університету. – 2023. – № 3(321). – С. 37-42. – DOI: 10.31891/2307-5732-2023-321-3-37-42.

7. Крупа С.; Кунанець Н. Аналіз використання HS- та HTS-кодів у системах митної класифікації: виклики та можливості інтеграції IT-технологій // Information Systems and Networks. – 2024. – Вип. 16. – С. 237-252. – DOI: 10.23939/sisn2024.16.237.

8. Войтех Д.; Тимошенко А. Використання машинного навчання та мережевих наборів даних для моделювання енергосистем // Інфокомунікаційні та комп’ютерні технології. – 2024. – Т. 1, № 07. – С. 35-45. – DOI: 10.36994/2788-5518-2024-01-07-05.

9. Пеняк Ю.; Калиниченко О. Моделі та методи управління виробничими запасами на підприємстві // Проблеми сучасних трансформацій. Серія: економіка та управління. – 2024. – № 15. – DOI: 10.54929/2786-5738-2024-15-09-01.

10. Liu F.; Zheng Q. An advanced BERT-based commodity classification on Amazon online malls based on consumer cognitive attributes // Proc. HICSS-58. – 2025. – DOI: 10.24251/HICSS.2025.579.

11. Demiray Kırmızı S.; Ceylan Z.; Bulkan S. Enhancing inventory management through safety-stock strategies — a case study // Systems. – 2024. – 12 (7): 260. – DOI: 10.3390/systems12070260.

12. Zhang M.; Wang Y.; Zhang Y. Research on supply-chain coordination decision making under the influence of lead time based on system dynamics // Systems. – 2024. – 12 (1): 32. – DOI: 10.3390/systems12010032.

13. Kurniasari I. Uji performa та аналіз архітектур MongoDB standalone, replica та shard replica // JATISI. – 2025. – 12 (1). – DOI: 10.35957/jatisi.v12i1.10307.

14. United Nations Development Programme. UNSPSC® — United Nations Standard Products and Services Code, Version 25.0701. – 2025. – URL: https://www.unspsc.org (дата звернення: 16.05.2025).

15. GS1. GS1 General Specifications. Release 24.0. – Brussels, 2025. – URL: https://ref.gs1.org/gsmp/kc/gs1-general-specifications.pdf (дата звернення: 16.05.2025).

16. SAP SE. SAP S/4HANA 2023 Feature Scope Description. – Walldorf, 2024. – URL: https://help.sap.com/doc/e2048712f0ab45e791e6d15ba5e20c68/2023/en-US/FSD_OP2023_latest.pdf (дата звернення: 16.05.2025).

17. Oracle NetSuite. Advanced Inventory — Data Sheet. – Austin, 2024. – URL: https://www.netsuite.com/portal/resource/datasheets.shtml#advanced-inventory (дата звернення: 16.05.2025).

18. Microsoft. Inventory management overview — Dynamics 365 Supply Chain Management. – Redmond, 2024. – URL: https://learn.microsoft.com/dynamics365/supply-chain/inventory/inventory-home-page (дата звернення: 16.05.2025).

19. Amazon Web Services. AWS Supply Chain — Administrator Guide. – Seattle, 2024. – URL: https://docs.aws.amazon.com/pdfs/aws-supply-chain/latest/adminguide/supplychain-ag.pdf (дата звернення: 16.05.2025).

20. Класифікація видів економічної діяльності ДК 009:2010 : Наказ Держспоживстандарту України від 11.10.2010 р. № 457 // База даних «Законодавство України». – URL: https://zakon.rada.gov.ua/go/vb457609-10 (дата звернення: 16.05.2025).

# ДОДАТКИ

## Додаток А. Презентація кваліфікаційної роботи

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

**Система обліку електротоварів з автоматичним розподілом за категоріями**

Виконав студент 4 курсу
Групи ІСД-41
Вєрягін Нікіта Олександрович
Керівник:
Шахматов Іван Олександрович

Київ – 2025

## МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета** дипломної роботи є підвищення ефективності функціональної й надійної системи обліку електротоварів із автоматичним розподілом за категоріями, що відповідає сучасним вимогам інформаційної безпеки, підтримує масштабованість і є зручною для користувачів різних рівнів.

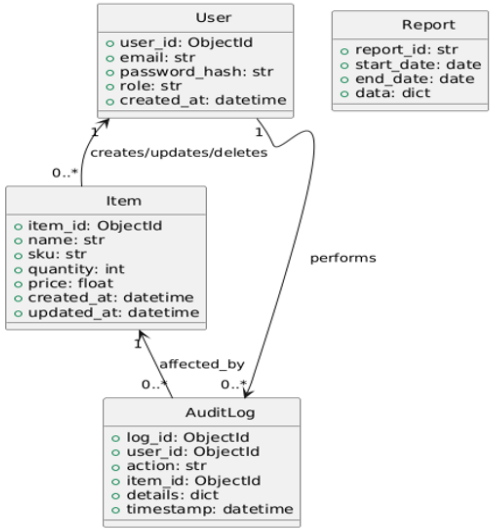
**Об'єкт дослідження** – це процес управління складськими запасами електротехнічної продукції в оптово-роздрібних мережах.

**Предмет дослідження** – створення інформаційної системи для автоматичного обліку та категоризації електротоварів.

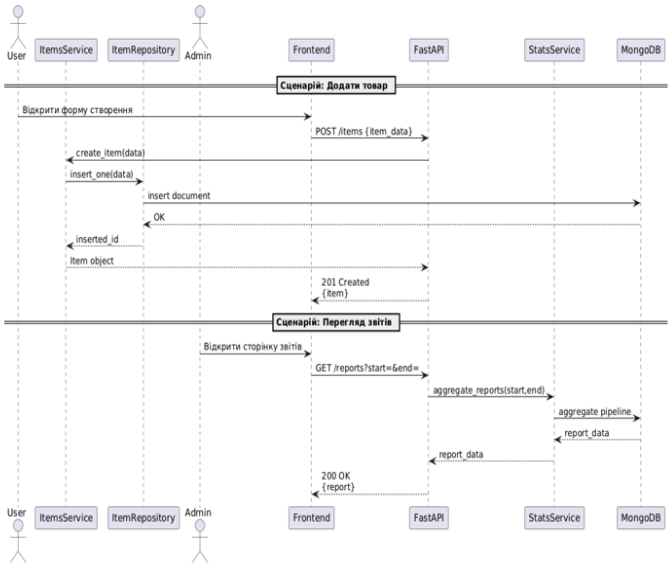
## ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

- Проаналізувати сучасні інформаційної системи, що забезпечить оперативний і точний облік усіх позицій на складі, автоматичну класифікацію за категоріями та підтримку гнучкої структури даних.
- Проаналізувати проблему помилок ручної класифікації.
- Розробити архітектуру системи, створено UML-діаграми прецедентів, компонентів та послідовностей.
- Розробити веб-застосунок з асинхронним API на FastAPI, базою даних MongoDB та фронтендом на чистому HTML/CSS/JavaScript, який дозволить зареєстрованим користувачам виконувати CRUD-операції над товарами, шукати та фільтрувати їх, а адміністраторам – контролювати історію змін і формувати статистичні звіти за довільні періоди.

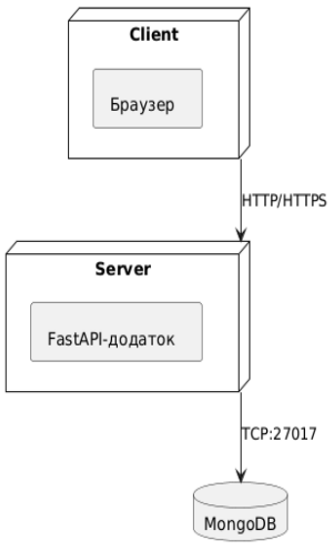
## Діаграма класів та сутностей



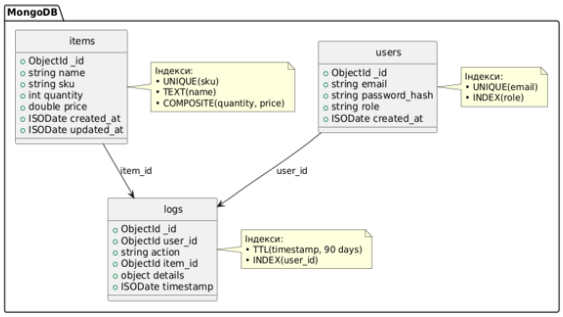
# Діаграми послідовностей для основних сценаріїв



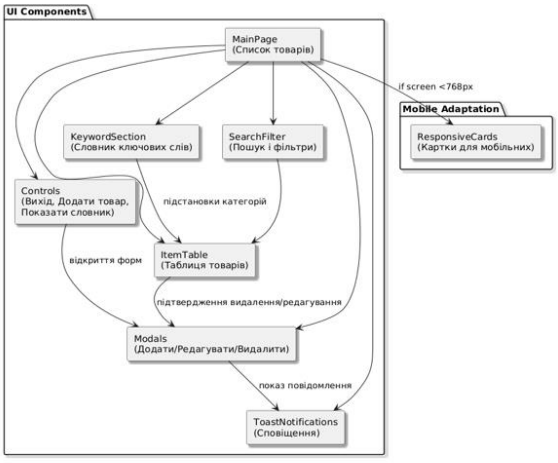

# Діаграма розгортання



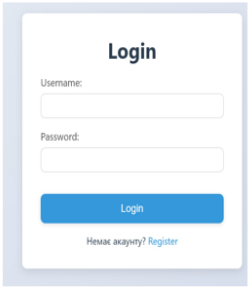
# Моделі даних та база даних системи

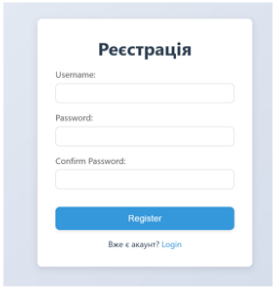


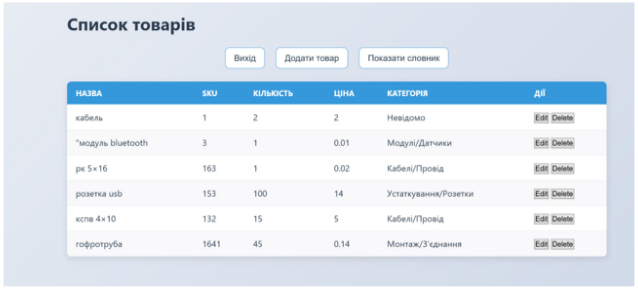

# Дизайн користувацького інтерфейсу

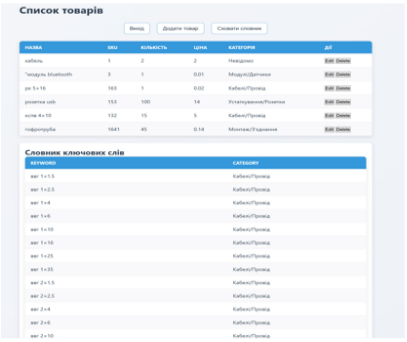


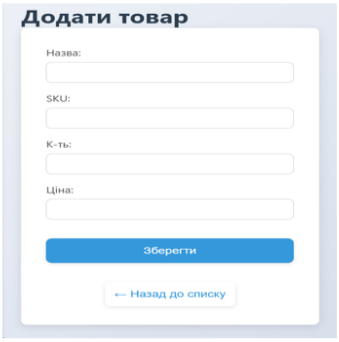
# Екранні форми

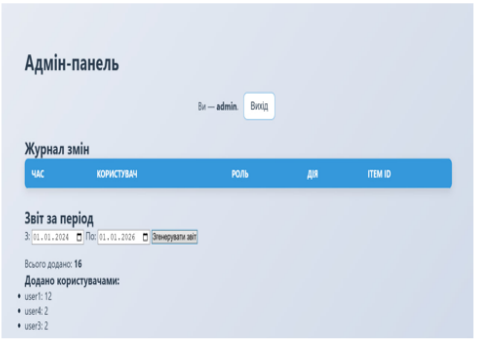


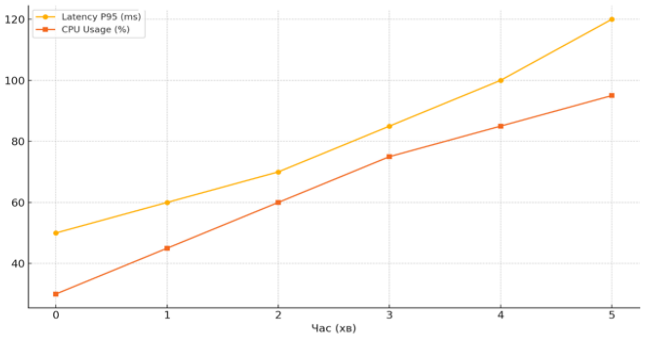



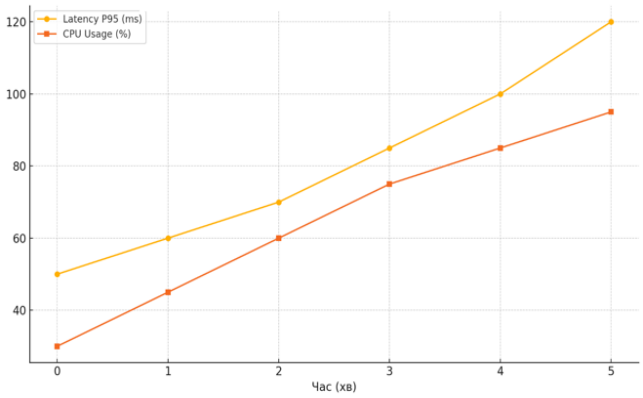

# Оцінка продуктивності та навантажувальні тести

Оцінка ефективності системи полягає у визначенні та оцінці ключових показників її працездатності, надійності та ефективності в контексті виконання поставлених завдань. Секторна діаграма на дозволяє графічно показати розподіл оцінок у відсотках за різними категоріями оцінки.

Тести проводилися за допомогою Locust, зі сценаріями генерації рівномірного навантаження без різкого старту, щоб виміряти стабільний throughput.

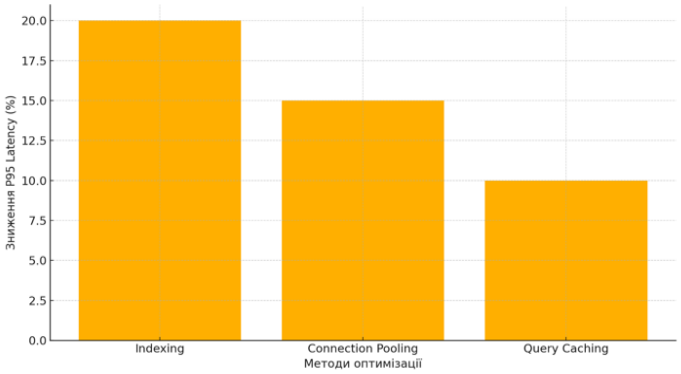


# Моніторинг затримки та споживання ресурсів




Моніторинг затримок та споживання ресурсів демонструє поведінку P95 latency (ms) та CPU usage (%) протягом 5 хвилинного стрес-тесту. Latency зростає з 50 до 120 мс, тоді як завантаження CPU – з 30 % до 95 %. Моніторинг здійснювався за допомогою Prometheus та Grafana, дані опитувалися кожні 30 сек.

# Рекомендації з оптимізації: очікуване зниження затримки




На діаграмі показано Рекомендації з оптимізації ілюструють очікуване зниження P95 latency (%) при впровадженні трьох основних заходів: індексування (−20 %), connection pooling (−15 %) та кешування запитів (−10 %). Ці цифри отримано на основі профілювання запитів у Py-Spy та MongoDB Profiler до та після оптимізації.

# ВИСНОВКИ

1. Проаналізовано сучасні інформаційної системи, що забезпечить оперативний і точний облік усіх позицій на складі, автоматичну класифікацію за категоріями та підтримку гнучкої структури даних.

2. Проаналізовано всебічний аналіз стану автоматизації обліку електротоварів, зокрема існуючих ERP- та . open-source рішень, а також сучасних підходів до класифікації та аудиту даних.

3. Визначено, що для подальшого розвитку інформаційної системи рекомендується впровадити додаткові методи оптимізації.

4. Впроваджено експериментальну реалізацію створення прототипу системи з інтегрованою автоматичною категоризацією товарів на основі ключових слів та застосуванням гібридного підходу, що поєднує словникові правила та машинне навчання з використанням моделі BERT.

5. Розроблено систему яка ефективно вирішує задачі автоматизації обліку електротехнічної продукції, та  створює значні передумови для оптимізації бізнес-процесів, зменшення витрат і підвищення конкурентоспроможності підприємств, які займаються оптово-роздрібною торгівлею електротоварами

6. Проведено тестування (unit, integration, performance, security) яке показало, що система демонструє високу точність категоризації (близько 95%), стабільність роботи, а також задовольняє високі стандарти безпеки й продуктивності навіть за умов значних навантажень.

## Додаток В. Повний вихідний код програми

*main.py*

```
import os
from datetime import datetime, timedelta
from typing import Optional, List, Dict

from fastapi import FastAPI, HTTPException, Depends, Form, Request
from fastapi.responses import RedirectResponse, FileResponse
from fastapi.staticfiles import StaticFiles
from pydantic import BaseModel
from motor.motor_asyncio import AsyncIOMotorClient
from passlib.context import CryptContext
from jose import JWTError, jwt
from bson import ObjectId

from categorizer import classify, keywords

# JWT Config
SECRET_KEY = os.getenv("SECRET_KEY", "supersecretkey")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

app = FastAPI(title="Inventory System")
app.mount("/static", StaticFiles(directory="static"), name="static")

# MongoDB
MONGO_URI = os.getenv("MONGO_URI", "mongodb://localhost:27017")
client    = AsyncIOMotorClient(MONGO_URI)
db        = client["inventory"]
items_col = db["items"]
users_col = db["users"]
logs_col  = db["logs"]  # для аудиту

# Password hashing
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
def get_password_hash(pw: str) -> str:
    return pwd_context.hash(pw)
def verify_password(plain: str, hashed: str) -> bool:
```

```
        return pwd_context.verify(plain, hashed)

    # Create default admin on startup
    @app.on_event("startup")
    async def create_default_admin():
        if not await users_col.find_one({"username": "admin"}):
            hashed = get_password_hash("admin")
            await users_col.insert_one({
                "username": "admin",
                "hashed_password": hashed,
                "role": "admin"
            })

    # JWT token creation
    def create_access_token(data: dict, expires_delta: timedelta = None) -> str:
        to_encode = data.copy()
            to_encode["exp"] = datetime.utcnow() + (expires_delta or
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
        return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

    # Get current user dependency
    async def get_current_user(request: Request):
        token = request.cookies.get("token")
        if not token:
            raise HTTPException(status_code=401, detail="Not authenticated")
        try:
            payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
            username: str = payload.get("sub")
            if not username:
                raise JWTError()
        except JWTError:
            raise HTTPException(status_code=401, detail="Invalid token")
        user = await users_col.find_one({"username": username})
        if not user:
            raise HTTPException(status_code=401, detail="User not found")
        if "role" not in user:
            user["role"] = "user"
        return user
```

```python
    # Admin role check
    def admin_required(user=Depends(get_current_user)):
        if user.get("role") != "admin":
            raise HTTPException(status_code=403, detail="Forbidden")
        return user

    # Audit log helper
    async def log_action(user: dict, action: str, item_id: Optional[str], details:
dict):
        await logs_col.insert_one({
            "user": user["username"],
            "role": user.get("role", "user"),
            "action": action,
            "item_id": item_id,
            "details": details,
            "timestamp": datetime.utcnow()
        })

    # ----- Auth routes -----

    @app.get("/", response_class=RedirectResponse)
    async def root():
        return RedirectResponse(url="/login")

    @app.get("/login", response_class=FileResponse)
    async def login_page():
        return "static/login.html"

    @app.post("/login")
    async def login(username: str = Form(...), password: str = Form(...)):
        user = await users_col.find_one({"username": username})
        if not user or not verify_password(password, user["hashed_password"]):
            raise HTTPException(status_code=400, detail="Invalid credentials")
        token = create_access_token({"sub": username})
        target = "/admin" if user.get("role") == "admin" else "/app"
        resp = RedirectResponse(target, status_code=302)
        resp.set_cookie(
            key="token",
            value=token,
```

```
        httponly=True,
        path="/"
    )
    return resp

@app.get("/register", response_class=FileResponse)
async def register_page():
    return "static/register.html"

@app.post("/register")
async def register(username: str = Form(...), password: str = Form(...)):
    if await users_col.find_one({"username": username}):
        raise HTTPException(status_code=400, detail="User already exists")
    hashed = get_password_hash(password)
    await users_col.insert_one({
        "username": username,
        "hashed_password": hashed,
        "role": "user"
    })
    return RedirectResponse("/login", status_code=302)

@app.get("/logout")
async def logout():
    resp = RedirectResponse("/login", status_code=302)
    resp.delete_cookie("token", path="/")
    return resp

@app.get("/app", response_class=FileResponse)
async def app_page(user=Depends(get_current_user)):
    return "static/index.html"

@app.get("/admin", response_class=FileResponse)
async def admin_page(user=Depends(admin_required)):
    return "static/admin.html"

# ----- Keywords endpoint -----

@app.get("/keywords", response_model=List[Dict[str, str]])
async def get_keywords(user=Depends(get_current_user)):
```

```
    return [{"keyword": k, "category": v} for k, v in keywords.items()]

# ----- CRUD endpoints -----

class Item(BaseModel):
    name: str
    sku: str
    quantity: int
    price: float

class ItemUpdate(BaseModel):
    name: Optional[str] = None
    sku: Optional[str] = None
    quantity: Optional[int] = None
    price: Optional[float] = None

@app.get("/items")
async def get_items(user=Depends(get_current_user)):
    items = []
    async for doc in items_col.find():
        doc["_id"] = str(doc["_id"])
        items.append(doc)
    return items

@app.post("/items")
async def add_item(item: Item, user=Depends(get_current_user)):
    data = item.dict()
    data["category"] = classify(data["name"])
    res = await items_col.insert_one(data)
    await log_action(user, "add", str(res.inserted_id), data)
    return {"added_id": str(res.inserted_id)}

@app.get("/items/{item_id}")
async def get_item(item_id: str, user=Depends(get_current_user)):
    doc = await items_col.find_one({"_id": ObjectId(item_id)})
    if not doc:
        raise HTTPException(status_code=404, detail="Item not found")
    doc["_id"] = str(doc["_id"])
    return doc
```

```
@app.put("/items/{item_id}")
async def update_item(item_id: str, item: ItemUpdate, user=Depends(get_current_user)):
    old = await items_col.find_one({"_id": ObjectId(item_id)})
    if not old:
        raise HTTPException(status_code=404, detail="Item not found")
    update_data = {k: v for k, v in item.dict().items() if v is not None}
    if "name" in update_data:
        update_data["category"] = classify(update_data["name"])
        await items_col.update_one({"_id": ObjectId(item_id)}, {"$set": update_data})
    await log_action(user, "update", item_id, {"before": old, "after": update_data})
    return {"updated_id": item_id}

@app.delete("/items/{item_id}")
async def delete_item(item_id: str, user=Depends(get_current_user)):
    res = await items_col.delete_one({"_id": ObjectId(item_id)})
    if res.deleted_count == 0:
        raise HTTPException(status_code=404, detail="Item not found")
    await log_action(user, "delete", item_id, {})
    return {"deleted_id": item_id}

# ----- Admin-only endpoints -----

@app.get("/logs", response_model=List[Dict])
async def view_logs(admin=Depends(admin_required)):
    cursor = logs_col.find().sort("timestamp", -1).limit(100)
    logs = []
    async for l in cursor:
        l["_id"] = str(l["_id"])
        l["timestamp"] = l["timestamp"].isoformat()
        logs.append(l)
    return logs

@app.get("/stats", response_model=Dict[str, Dict[str, int]])
async def stats(start: Optional[str] = None, end: Optional[str] = None, admin=Depends(admin_required)):
```

```
        match = {}
        if start or end:
            ts = {}
            if start: ts["$gte"] = datetime.fromisoformat(start)
            if end:   ts["$lte"] = datetime.fromisoformat(end)
            match["timestamp"] = ts
        pipeline = [
            {"$match": match},
             {"$group": {"_id": {"user": "$user", "action": "$action"}, "count":
{"$sum": 1}}}
        ]
        agg = logs_col.aggregate(pipeline)
        result: Dict[str, Dict[str, int]] = {}
        async for r in agg:
            usr, act = r["_id"]["user"], r["_id"]["action"]
            result.setdefault(usr, {})[act] = r["count"]
        return result

    if __name__ == "__main__":
        import uvicorn
        uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```

*styles.css*

```
/* ================================
   ОНОВЛЕНИЙ styles.css
   Лаконічний дизайн для логіну, списку товарів і таблиць
   ================================ */

/* Глобальні налаштування */
:root {
    --clr-bg: #f5f7fa;
    --clr-gradient-end: #c3cfe2;
    --clr-card: #ffffff;
    --clr-primary: #3498db;
    --clr-primary-dark: #2980b9;
    --clr-text: #2c3e50;
    --clr-muted: #555;
    --radius: 0.5rem;
```

```
    --shadow-soft: 0 4px 12px rgba(0,0,0,0.1);
    --shadow-subtle: 0 2px 6px rgba(0,0,0,0.1);
    --spacing: 1rem;
    --transition-fast: 0.2s;
  }

  * {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
  }

  /* Тло сторінки */
  body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, var(--clr-bg) 0%, var(--clr-gradient-
end) 100%);
    color: var(--clr-text);
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: var(--spacing);
  }

  /* ========== Login ========== */
  .login-container {
    background: var(--clr-card);
    padding: calc(var(--spacing) * 2);
    border-radius: var(--radius);
    box-shadow: var(--shadow-soft);
    width: 100%;
    max-width: 380px;
    text-align: center;
  }

  .login-container h1 {
    font-size: 2rem;
    margin-bottom: var(--spacing);
```

```css
}

.login-container form {
  display: flex;
  flex-direction: column;
}

.login-container label {
  text-align: left;
  margin-bottom: var(--spacing);
  font-size: 0.95rem;
  color: var(--clr-muted);
}

.login-container input {
  width: 100%;
  padding: 0.6rem;
  margin-top: 0.4rem;
  border: 1px solid #ccc;
  border-radius: var(--radius);
  transition: border-color var(--transition-fast);
}

.login-container input:focus {
  outline: none;
  border-color: var(--clr-primary);
}

.login-container button {
  margin-top: var(--spacing);
  padding: 0.75rem;
  font-size: 1rem;
  background: var(--clr-primary);
  color: #fff;
  border: none;
  border-radius: var(--radius);
  cursor: pointer;
  box-shadow: var(--shadow-subtle);
```

```
  transition: background var(--transition-fast), transform var(--transition-fast);
}

.login-container button:hover {
  background: var(--clr-primary-dark);
  transform: translateY(-2px);
}

.login-container p {
  margin-top: var(--spacing);
  font-size: 0.9rem;
}

.login-container a {
  color: var(--clr-primary);
  text-decoration: none;
  transition: color var(--transition-fast);
}

.login-container a:hover {
  color: var(--clr-primary-dark);
}

/* ========== App Page ========== */
.app-page {
  flex-direction: column;
}

.app-container {
  width: 100%;
  max-width: 1000px;
}

/* Контролі (кнопки) */
.controls {
  margin: calc(var(--spacing) * 1.5) 0;
  text-align: center;
}
```

```
.controls button {
  margin: 0 0.5rem;
  padding: 0.6rem 1rem;
  font-size: 1rem;
  background: var(--clr-card);
  color: var(--clr-text);
  border: 1px solid var(--clr-primary);
  border-radius: var(--radius);
  cursor: pointer;
   transition: background var(--transition-fast), color var(--transition-
fast);
}

.controls button:hover {
  background: var(--clr-primary);
  color: #fff;
}

/* Обгортка для таблиці */
.table-wrapper {
  overflow-x: auto;
  margin-bottom: calc(var(--spacing) * 1.5);
  box-shadow: var(--shadow-soft);
  border-radius: var(--radius);
  background: var(--clr-card);
}

/* Тіло та заголовок таблиці */
table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  padding: 0.75rem var(--spacing);
  border-bottom: 1px solid #eee;
  text-align: left;
}
```

```css
thead th {
  background: var(--clr-primary);
  color: #fff;
  text-transform: uppercase;
  font-size: 0.9rem;
}

tbody tr:nth-child(even) {
  background: #f9f9fb;
}

tbody tr:hover {
  background: var(--clr-gradient-end);
}

/* Секція словника */
#keywordsSection {
  display: none;
  padding: var(--spacing);
  border-radius: var(--radius);
  box-shadow: var(--shadow-soft);
  background: var(--clr-card);
  margin-bottom: calc(var(--spacing) * 1.5);
}

#keywordsSection.active {
  display: block;
}

/* Адаптивність */
@media (max-width: 768px) {
  th, td {
    padding: 0.5rem;
    font-size: 0.9rem;
  }
  .controls button,
  .login-container button {
    font-size: 0.9rem;
```

```
      padding: 0.5rem 1rem;
    }
  }
```

*main.js*

```
// static/main.js

let loadedKeywords = false;

// Завантажуємо та відображаємо список товарів
async function loadItems() {
  const res = await fetch("/items", {
    credentials: "include", // ← додаємо
  });
  if (!res.ok) {
    if (res.status === 401) window.location.href = "/";
    return;
  }
  const items = await res.json();
  const tbody = document.querySelector("#itemsTable tbody");
  tbody.innerHTML = "";
  items.forEach((it) => {
    const tr = document.createElement("tr");
    tr.innerHTML = `
      <td>${it.name}</td>
      <td>${it.sku}</td>
      <td>${it.quantity}</td>
      <td>${it.price}</td>
      <td>${it.category}</td>
      <td>
                                                                    <button
onclick="location.href='/static/edit_item.html?id=${it._id}'">Edit</button>
        <button onclick="deleteItem('${it._id}')">Delete</button>
      </td>
    `;
    tbody.appendChild(tr);
  });
}
```

```
// Видалення товару
async function deleteItem(id) {
  if (!confirm("Видалити цей товар?")) return;
  const res = await fetch(`/items/${id}`, {
    method: "DELETE",
    credentials: "include", // ← і тут
  });
  if (!res.ok) {
    if (res.status === 401) window.location.href = "/";
    return;
  }
  loadItems();
}

// Завантаження словника
async function loadKeywords() {
  const res = await fetch("/keywords", {
    credentials: "include", // ← і тут
  });
  if (!res.ok) {
    if (res.status === 401) window.location.href = "/";
    return;
  }
  const data = await res.json();
  const tbody = document.querySelector("#keywordsTable tbody");
  tbody.innerHTML = "";
  data.forEach((row) => {
    const tr = document.createElement("tr");
    tr.innerHTML = `<td>${row.keyword}</td><td>${row.category}</td>`;
    tbody.appendChild(tr);
  });
}

// Toggle словника
document
  .getElementById("toggleKeywordsBtn")
  .addEventListener("click", async () => {
    const section = document.getElementById("keywordsSection");
```

```
    const btn = document.getElementById("toggleKeywordsBtn");
    if (section.style.display === "none") {
      if (!loadedKeywords) {
        await loadKeywords();
        loadedKeywords = true;
      }
      section.style.display = "block";
      btn.textContent = "Сховати словник";
    } else {
      section.style.display = "none";
      btn.textContent = "Показати словник";
    }
  });

// Ініціалізація
loadItems();
```

*index.html*

```
<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Список товарів</title>
  <link rel="stylesheet" href="/static/styles.css">
</head>
<body class="app-page">
  <div class="app-container">
    <header>
      <h1>Список товарів</h1>
    </header>

    <div class="controls">
      <button onclick="location.href='/logout'">Вихід</button>
            <button   onclick="location.href='/static/add_item.html'">Додати
товар</button>
      <button id="toggleKeywordsBtn">Показати словник</button>
    </div>
```

```
    <div class="table-wrapper">
      <table id="itemsTable">
        <thead>
          <tr>
            <th>Назва</th>
            <th>SKU</th>
            <th>Кількість</th>
            <th>Ціна</th>
            <th>Категорія</th>
            <th>Дії</th>
          </tr>
        </thead>
        <tbody>
          <!-- JS: завантаження товарів -->
        </tbody>
      </table>
    </div>

    <section id="keywordsSection" class="hidden">
      <h2>Словник ключових слів</h2>
      <div class="table-wrapper">
        <table id="keywordsTable">
          <thead>
            <tr>
              <th>Keyword</th>
              <th>Category</th>
            </tr>
          </thead>
          <tbody>
            <!-- JS: завантаження словника -->
          </tbody>
        </table>
      </div>
    </section>
  </div>

  <script src="/static/main.js"></script>
</body>
```

```
</html>
```

## Додаток С. Технічна документація та інструкція з встановлення

У цьому додатку наведено технічну документацію з екологічного моніторингу, яка охоплює всі аспекти розробки та реалізації системи.

### ЗМІСТ

- Налаштування системи на різних платформах

8. **Підтримка та експлуатація**
   - Інструкції з підтримки системи
   - Поради щодо експлуатації та підтримки
9. **Висновки**
   - Підсумки проекту
   - Рекомендації щодо подальшого розвитку

## Додаток D. Набір даних csv

```
keyword,category
ввг 1×1.5,Кабелі/Провід
ввг 1×2.5,Кабелі/Провід
ввг 1×4,Кабелі/Провід
ввг 1×6,Кабелі/Провід
ввг 1×10,Кабелі/Провід
ввг 1×16,Кабелі/Провід
ввг 1×25,Кабелі/Провід
ввг 1×35,Кабелі/Провід
ввг 2×1.5,Кабелі/Провід
ввг 2×2.5,Кабелі/Провід
ввг 2×4,Кабелі/Провід
ввг 2×6,Кабелі/Провід
ввг 2×10,Кабелі/Провід
ввг 2×16,Кабелі/Провід
ввг 2×25,Кабелі/Провід
ввг 2×35,Кабелі/Провід
ввг 3×1.5,Кабелі/Провід
ввг 3×2.5,Кабелі/Провід
ввг 3×4,Кабелі/Провід
ввг 3×6,Кабелі/Провід
ввг 3×10,Кабелі/Провід
ввг 3×16,Кабелі/Провід
ввг 3×25,Кабелі/Провід
ввг 3×35,Кабелі/Провід
ввг 4×1.5,Кабелі/Провід
ввг 4×2.5,Кабелі/Провід
ввг 4×4,Кабелі/Провід
ввг 4×6,Кабелі/Провід
ввг 4×10,Кабелі/Провід
ввг 4×16,Кабелі/Провід
ввг 4×25,Кабелі/Провід
ввг 4×35,Кабелі/Провід
ввг 5×1.5,Кабелі/Провід
ввг 5×2.5,Кабелі/Провід
ввг 5×4,Кабелі/Провід
ввг 5×6,Кабелі/Провід
```

ввг 5×10,Кабелі/Провід
ввг 5×16,Кабелі/Провід
ввг 5×25,Кабелі/Провід
ввг 5×35,Кабелі/Провід
nym 1×1.5,Кабелі/Провід
nym 1×2.5,Кабелі/Провід
nym 1×4,Кабелі/Провід
nym 1×6,Кабелі/Провід
nym 1×10,Кабелі/Провід
nym 1×16,Кабелі/Провід
nym 1×25,Кабелі/Провід
nym 1×35,Кабелі/Провід
nym 2×1.5,Кабелі/Провід
nym 2×2.5,Кабелі/Провід
nym 2×4,Кабелі/Провід
nym 2×6,Кабелі/Провід
nym 2×10,Кабелі/Провід
nym 2×16,Кабелі/Провід
nym 2×25,Кабелі/Провід
nym 2×35,Кабелі/Провід
nym 3×1.5,Кабелі/Провід
nym 3×2.5,Кабелі/Провід
nym 3×4,Кабелі/Провід
nym 3×6,Кабелі/Провід
nym 3×10,Кабелі/Провід
nym 3×16,Кабелі/Провід
nym 3×25,Кабелі/Провід
nym 3×35,Кабелі/Провід
nym 4×1.5,Кабелі/Провід
nym 4×2.5,Кабелі/Провід
nym 4×4,Кабелі/Провід
nym 4×6,Кабелі/Провід
nym 4×10,Кабелі/Провід
пвс 3×1.5,Кабелі/Провід
пвс 3×2.5,Кабелі/Провід
пвс 3×4,Кабелі/Провід
pzr вимикач,Автоматика/Захист
реле,Автоматика/Захист
реле напруги,Автоматика/Захист

стабілізатор,Автоматика/Захист
унз,Автоматика/Захист
предохранитель,Автоматика/Захист
пзр,Автоматика/Захист
клемник,Монтаж/З’єднання
гвинтова клема,Монтаж/З’єднання
пружинна клема,Монтаж/З’єднання
клема ізоляційна,Монтаж/З’єднання
кабельний затискач,Монтаж/З’єднання
дюбель,Монтаж/З’єднання
шуруп,Монтаж/З’єднання
гайка,Монтаж/З’єднання
шайба,Монтаж/З’єднання
хомут,Монтаж/З’єднання
гофра,Монтаж/З’єднання
гофротруба,Монтаж/З’єднання
манжет,Монтаж/З’єднання
муфта,Монтаж/З’єднання
коробка розподільна,Монтаж/З’єднання
шуруповерт,Інструмент
дриль,Інструмент
перфоратор,Інструмент
болгарка,Інструмент
шліфувальна машина,Інструмент
мультиметр,Інструмент
тестер,Інструмент
лобзик,Інструмент
паяльник,Інструмент
світлодіодний ліхтарик,Інструмент
лампа настільна,Інструмент
електроінструмент,Інструмент
кабель-канал 16,Кабель-канали
кабель-канал 32,Кабель-канали
кабель-канал 60,Кабель-канали
короб кабельний,Кабель-канали
канал для проводів,Кабель-канали
блок живлення,Електроживлення

## Додаток F. Налаштування проекту та запуск

**Встановлення середовища**. Клонуйте репозиторій, інсталюйте Python ≥ 3.10, сам MongoDB Community Server та створіть віртуальне середовище командою python -m venv venv; активуйте його (source venv/bin/activate для Linux/Mac / venv\Scripts\activate для Windows) та встановіть залежності pip install -r requirements.txt.

**Налаштування середовища**. Переконайтеся, що локальний MongoDB слухає mongodb://localhost:27017; за потреби змініть змінну оточення MONGO_URI. Під час старту застосунок сам створює адміністратора з обліковими даними admin:admin, тож додаткових кроків не треба.

**Запуск та використання**. У вже активованому віртуальному середовищі запустіть сервер командою python main.py, після чого відкрийте в браузері http://localhost:8000. Користувачі можуть реєструватися на /register, входити на /, керувати товарами на /app; адміністратор після входу автоматично перенаправляється на /admin, де доступні журнал змін /logs і статистика /stats?start=YYYY-MM-DD&end=YYYY-MM-DD.