БАКАЛАВРСЬКА РОБОТА

БР.ІП – 09.00.00.000 ІПЗ

Група ІП-21-4

Степуняк Олег

2025

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Степуняк Олег Ярославович

(прізвище, ім'я, по батькові)

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка веб-додатку для управління замовленнями та товарними залишками магазину і складу

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Степуняк О.Я

(підпис, ініціали та прізвище здобувача)

Науковий керівник Шекета В.І., професор , д.т.н.

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

***ЗАТВЕРДЖУЮ*:**
Зав. кафедрою ***ІПЗ***
доц. В.В. Бандура
"____" _____ 2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Степуняку Олегу Ярославовичу
(прізвище, ім'я, по-батькові)

1. ***Тема проекту (роботи)*** **"Розробка веб-додатку для управління замовленнями та товарними залишками магазину і складу"**

керівник проекту (роботи) Шекета Василь Іванович, професор, д.т.н.

затверджені наказом закладу вищої освіти від "28" квітня 2025 р. № 264/7

2. ***Строк подання студентом проекту (роботи)*** 10 червня 2025 р.
3. ***Вихідні дані до проекту (роботи)*** згідно з темою роботи
4. ***Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)***

1. Огляд та аналіз існуючих рішень
2. Специфікація вимог до системи
3. Проектування веб-системи
4. Реалізація веб-системи
5. Тестування системи

5. ***Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)***

1. Use case системи (рис. 2.1, ст. 23)
2. Загальний вигляд системи (рис.3.1, ст. 27)
3. Загальний вигляд документів системи (рис. 3.2, ст. 29)
4. Вигляд класу CustomUser (рис.4.1, ст. 31)
5. Вигляд класів Category та Product (рис. 4.2, ст. 32)

1. ***Консультанти розділів проекту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

2. ***Дата видачі завдання*** _____

Керівник _____ Шекета В.І
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Степуняк О.Я.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проекту (роботи)***	***Строк виконання етапів проекту***	***Примітка***
1	Вибір теми та постановка задачі	15.02.2025	Виконано
2	Аналіз літератури та проектування системи	10.03.2025	Виконано
3	Створення системи	25.04.2025	Виконано
4	Тестування та оптимізація системи	30.04.2025	Виконано
5	Оформлення пояснювальної записки	10.06.2025	Виконано

Студент – дипломник _____ Степуняк О.Я.
(підпис) (розшифровка підпису)

Керівник роботи _____ Шекета В.І.
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 64 сторінки, 36 рисунків, список використаних джерел з 30 найменуванням та 1 додаток.

Метою роботи є створення та реалізація веб-додатку для автоматизації процесів обліку товару та замовлень магазину та складу з можливістю управління залишками, формування замовлень та доступу користувачів за правами.

Об'єкт дослідження: інформаційні системи для управління магазином та складом.

Предмет дослідження: методи розробки веб-систем на основі клієнт-серверної архітектури з використанням Django REST Framework та React.

Результати дослідження: реалізовано повнофункціональну веб-систему, яка надає функціонал управління товарами та замовленнями.

У першому розділі проаналізовано сучасний стан подібних систем та проведено порівняння функціоналу існуючих рішень.

У другому розділі сформульовано функціональні та нефункціональні вимоги та описано ролі користувачів системи.

У третьому розділі описано вибір технологій та структуру системи.

У четвертому розділі реалізовано модель бази даних та функціонал додатку.

У п'ятому розділі проведено тестування розробленої системи.

Висновок: створено веб-систему для обліку товару та створення замовлень, яка відповідає сучасним вимогам зручності, надійності та безпеки.

КЛЮЧОВІ СЛОВА: DJANGO, REACT, ЗАМОВЛЕННЯ, REST API, JWT, КЛІЄНТ-СЕРВЕР, ТЕСТУВАННЯ, АВТЕНТИФІКАЦІЯ.

ABSTRACT

The bachelor's thesis contains 64 pages, 36 figures, a list of used sources with 30 titles and 1 appendice.

The purpose of the work is to design and implement a web application for automating the processes of accounting for goods and order management for a store and warehouse, with features for stock control, order processing and user access management based on roles.

Object of research: information systems for store and warehouse management.

Subject of research: methods of developing a web system based on client-server architecture using Django REST Framework and React.

Research results: a fully functional web system was implemented, providing functional management of goods and orders.

The first section analyzes the current state of similar systems and compares the functionality of existing solutions.

The second section formulates functional and non-functional requirements and describes the user roles in the system.

The third section describes the technologies that were chosen and system structure.

The fourth section describes the implementation of the database model and application functionality.

The fifth section tests the developed system.

Conclusion: a web-based system for inventory tracking and order creation was developed, meeting modern standards of usability, reliability, and security.

KEYWORDS: DJANGO, REACT, ORDER, REST API, JWT, CLIENT-SERVER, TESTING, AUTHENTICATION.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ В ПРЕДМЕТНІЙ ОБЛАСТІ

SaaS – Software as a Service

ERP – Enterprise Resource Planning

CRUD – Create, Read, Update, Delete

IT – Information Technology

ABAP – Advanced Business Application Programming

UI – User Interface

DB - Database

API – Application Programming Interface

JSON – JavaScript Object Notation

JWT – JSON Web Token

UX – User Experience

SKU – Stock Keeping Unit

SQL – Structured Query Language

REST – Representational State Transfer

HTTP – Hypertext Transfer Protocol

CORS – Cross-Origin Resource Sharing

CSRF – Cross-Site Request Forgery

HTTP – Hypertext Transfer Protocol

ВСТУП

В сучасних умовах розвитку інформаційних технологій та електронної комерції питання точності та продуктивності обліку товарів є критично важливим для будь якого торгівельного підприємства. Ручне ведення обліку даних часто призводить до зменшення продуктивності, неточності даних та затримок в роботі, що в свою чергу, збільшує витрати підприємста, та знижує задоволення потенційного клієнта. Впровадження сучасної ВЕБ-системи для зручного управління замовленнями та даними про товар в наявності, дозволяє привнести точність в постійні процеси бізнесу та зменшити вплив людського фактору на роботу. Таким чином, створення такого програмного забезпечення має практичне використання для магазинів з проблемами обліку товару.

Актуальність теми дослідження полягає в забезпеченні малих та середніх підприємств, які не мають доступу до інструментів для автоматизації. Створення системи з використанням технологій Django та React дозволяє розробити гнучке та економічно вигідне рішення для вирішення проблеми нестачі автоматизації процесів магазину.

Завданням дослідження є аналіз сучасних інформаційних систем-аналогів автоматизації обліку товару та замовлень для подальшої розробки системи з використанням актуальних технологій.

Метою цієї роботи є стоврення та впровадження веб-додатку, який повинен забезпечувати ефективне управління товаром та замовленнями магазину. Для досягнення поставленої мети необхідно реалізувати наступні завдання:

- Проаналізувати сучасний стан інформаційних систем автоматизації обліку товару та замовлень та ознайомитися з аналогами рішень проблеми.
- Сформулювати загальні та спецефічні вимоги до системи
- Розробити архітектуру системи, застосовуючи актуальні та сучасні технології. Визначити структуру бази даних та моделей системи.

- Розробити UI системи, для зручності та зрозумілості використання системи.
- Реалізувати необхідний рівень безпеки системи. Реалізувати автентифікацію та авторизацію користувачів, захистити систему від потенційних атак.
- Провести тестування системи та оцінити відповідність результатів до поставлених вимог.

Об’єктом дослідження роботи є бізнес процеси обліку товарного залишку та замовлень магазинів.

Предметом дослідження виступає веб-система автоматизації процесів обліку товарних залишків та замовлень.

Наукова новизна роботи полягає в поєднанні сучасних веб технологій з метою створення функціональної та безпечної системи для автоматизації обліку товару та замовлень магазинів.

Основними методами використаними в побудові системи є методи аналізу та порівняння, методи розробки веб-додатків та моделювання архітектури системи. Також для тестування працездатності системи було використано методи ручного та автоматизованого тестування.

Структура роботи складається з п'яти розділів. Перший розділ описує сучасні аналоги та надає загальний огляд на стан проблеми. У другому розділі було описано основні вимоги до системи та ролей користувачів. Третій розділ фокусується на створенні архітектури та виборі технологій. У четвертому розділі описана реалізація самої системи. П'ятий розділ містить інформацію про тестування системи та результати. Загальний обсяг роботи становить 64 сторінки.

Результатом дослідження стала повністю реалізована веб-система, яка забезпечує можливості керуванням товару та замовленнями. Сама система підтримує розділ типів користувачів за правами доступу, фільтрацію та пошук даних та оновлення кількості залишків.

Розроблений ВЕБ-додаток може бути використаний в роботі роздрібних магазинів, які прагнуть автоматизувати облік товару без потреби витрат на дорогі системи. Реалізовані алгоритми автоматичного оновлення залмшків товару, звітів та контролю доступу надають зручну основу для подальшого розширення функціоналу.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Сучасний стан розробки інформаційних систем для автоматизації обліку товару та замовлень.

Впродовж декількох останніх десятиліть бізнеси усе активніше почитанають використовувати інформаційні системи, з допомогою яких можна автоматизовувати процеси обліку товарних даних та операцій всередині бізнесу.

Ондими з найбільш поширених тенденцій в даній сфері на сьогоднішній день є:

1. Використання хмарних рішень (SaaS).

SaaS – це модель хмарного обчислення, де постачальник пропонує клієнту доступ до програмного забезпечення та користувач не відповідає за саму функціональність системи [22]. Дуже багато постачальників програмного забезпечення пропунують хмарні сервіси, які не вимагають локального розгортання. Сервіси як: Odoo, Zoho Inventory, TradeGecko. Хмарні системи надають можливість швидкого впровадження на різні типи пристроїв без додаткових витрат. Додатково хмірні рішення забезпечують централізоване та безпечне зберігання даних, що саме по собі є великим плюсом для користувачів систем

2. Аналітика.

Використання методів великих даних та інструментів бізнес-аналітики надає можливість збирати, обробляти та демонструвати дані про залишки та продажі. Доволі часто, новітні системи обліку товару використовують інструменти прогнозування попиту, які створені за допомогою алгоритмів машинного навчання. Наприклад, великі американські супермаркети, такі як Walmart, використовують аналітику для автоматичного формування замовлень, що допомагає з продуктивністю та збереженням коштів.

3. Мобільність системи.

Для зручності виконання роботи сучасні працівники складів та магазинів все частіше користуються мобільними пристроями, такими як сканери. Для полегшення використання системи, багато сервісів вводять мобільні додатки або ж адаптивні веб-інтерфейси.

4. Модульність систем.

Сучасні системи, які використовуються компаніями, побудовані базуючись на принципах модульної архітектури. Цей підхід надає можливість “підбирати” потрібний набір функціоналу в залежності від потреб конкретного бізнесу. Більшість популярних рішень дозволяють активовувати лише потрібні модулі системи. З допомогою цієї функції система продовжує працювати справно під меншою напругою.

5. Підвищена безпека даних.

Хмарні системи автоматизація обліку товару все частіше і частіше використовують методи покращення безпеки на рівні підприємств. Дуже часто використовується шифрування даних, керування доступом через ролі та багаторівневе резервне копіювання даних [15].

Таким чином, сьогодні, ринок систем автоматизації обліку товарів та замовлень активно змінюється у напрямку хмарності, мобільності, підвищення автоматизація, включаючи впровадження функцій машинного навчання, та аналітики. Саме це дає можливість компаніям скорочувати витрати, одночасно з цим збільшуючи продуктивність та гнучкість у виконанні роботи.

1.2. Огляд сучасних аналогічних систем та їх технологій.

Odoo - Універсальна модульна ERP система з відкритим кодом. ERP – це інформаційна система, яка використовується для автоматизації керування інформацією про основні процеси бізнесу [23]. Сама система Odoo містить модулі для управління складом, продажами, товаром та фінансами [2]. Сама система використовує Python для backend, PostgreSQL як базу даних та для

frontend використовується веб інтерфейс на базі QWeb [3][8].

Можливості системи:

- Використання CRUD операцій
- Підтримка інвенторизації за різними правилами
- Підтримка та облік складських локацій

SAP - Німецька комерційна ERP система для бізнесів, яка дуже часто використовується у тогрівлі та дистрибуції товарів [5]. Система побудована з використанням бази даних SAP HANA. Інтерфейс використовує ABAP [3][9][10]. ABAP – це пропрієтрна внутрішня мова програування високого рівня, яка використовується компанією SAP в їхніх додатках [16].

Можливості системи:

- Комплексний облік товару
- Мобільність
- Наявність інструментів аналітики продажів
- Механізми налаштування логістичних процесів системи

Fishbowl Inventory – ERP система яка є орієнтованою на використання з QuickBooks [11]. Часто використовується малим бізнесом. Можна використовувати і локально, і на хмарі. Backend системи створено з допомогою C#, в якості бази даних використовується MySql, frontend частина системи побудована з допомогою ReactJS. Також для зручності використання був створений мобільний додаток на IOS/Android [7].

Можливості системи:

- Інтеграція з QuickBooks. QuickBooks – це набір бухгалтерських програм, яитй полегшує керування рахунками та фінансами серед компанії. Зазвичай використовується малими та середніми по розмірах підприємствами для нарахування заробітньої плати працівникам [24]
- Контроль серійних номерів, партій
- Планування запасів

Zoho Inventory - Хмарне рішення для малих та середній бізнесів. В

основному використовується для автоматизації управління запасами та замовленнями товару [4]. Система використовує PHP для backend, MySQL, як базу даних та ReactJS/Angular для frontend [6].

Можливості системи:

- Підтримка декількох складів
- Ведення обліку серій та партій
- Легка інтеграція з іншими платформами

Отже, сучасні системи управління товарами базуються на принципах модальності, масштабованості та мобільності. Всі ці вимоги задовільняєтються з допомогою використання веб-технологій по типу React або ж Angular для реалізація frontend системи, фреймворків по типу Django або ж мови PHP для backend та потужньої бази даних, таких як PostgreSQL, SAP HANA або ж MySQL. Ці рішення часто використовуються малим та середнім бізнесом для автоматизації процесів управління товаром та замовленнями.

1.3 Порівняльний аналіз функціоналу

На даний момент на ринку одні з найпопулярніших рішень, які використовуються для автоматизації обліку товарів та замовлень, є Odoo, SAP, Zoho Inventory, Fishbowl Inventory. Кожна з цих систем має свої плюси та мінуси, та використання для бізнесів різних розмірів та з різними бюджетами.

Odoo, в порівнянні з іншими системами, високою гнучкістю та відкритістю коду. Це надає можливість створення модульної системи яка складається з окремих додатків. Також одним з великих плюсів сервісу є те, що Odoo підтримує не тільки прості CRUD операції для простих операцій над товарами та замовленнями, а також і роботу з декількома складськими локаціями, надає можливість контролю серійних номерів продуктів і підримує автоматичне оновлення кількості залишків товару.

Fishbowl Inventory надає можливість управління запасами з QuickBooks,

що робить цей сервіс привабливим для компаній та спрощує роботу з бухгалтерським пакетом. Окрім цього Fishbowl підтримує роботу з декількома локаціями, що може бути плюсом для мережі магазинів, та надає можливість контролю серійних номерів. Найбільшим мінусом цього рішення є ціна ліцензії та необхідність наявності власних серверів.

SAP - це комерційна система керування товарами, яка забезпечує глибоку аналітику в реальному часі, що стає можливим в зв'язку з використанням SAP HANA як бази даних. Сама система є багато функціональною, забезпечуючи можливість обліку кількох складів або ж магазинів, керування фінансами та виробництва. Додатково, вимагає залучення IT-фахівців для коректного налаштування. Через високу ціну ліцензії та налаштування дана система не є доступною для малого бізнесу [10].

Zoho Inventory є хмарним SaaS рішенням, яке частіше всього використовується малими та середніми по розміру бізнесами. Система є легкою в налаштуванні та не потребує багато коштів, що і є бажаним для не великого бізнесу. Великим плюсом Zoho є наявність функціоналу базових звітів продажів та замовлень [6].

Порівнюючи всі системи та їх функціонал, можна виділити наступні спостереження:

- Управління товаром: Всі системи надають можливість модифікації інформації про товар.
- Наявність рівнів доступу: Всі системи мають комплексну систему авторизації та ролей для користувачів.
- Автоматичне оновлення залишків: Реалізоване в усіх системах.
- Мобільність: Присутня у всіх системах, але зроблено по різному. Zoho Inventory та Fishbowl більше орієнтуються на хмарну реалізацію, коли ж SAP використовує локальні компоненти та додатковий мобільний клієнт.
- Зручність використання: всі додатки мають зрозумілий UI з допомогою якого користувачі можуть легко взаємодіяти з системою.

1.4 Висновки до розділу

Існуючі системи ERP забезпечують максимальний функціонал та масштабність, але потребують значних інвестицій у ліцензію, налаштуванння та підтримку системи. Такі рішення, як SAP, переважно орієнтовуються на великий бізнес, в той час, Zoho Inventory та Fishbowl підходять для більш малих компаній, але мають обмеження в гнучкості використання.

Аналіз систем показав, що сучасні рішення автоматизації обліку товарів використовують підходи мобільності, модульності та хмарності додатків. Ці характеристики популярних рішень забезпечують простоту використання, гнучкість та безпеку системи.

Таким чином потреба в розробці власного рішення, у вигляді власного веб-додатку, який поєднює ключові переваги описаних систем, є обгрунтованою. Це дозволяє уникнути недоліків існуючих альтернативних рішень та безпосередньо створити систему з урахуванням плюсів інших систем.

РОЗДІЛ 2. СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ

2.1 Загальні вимоги до системи

Для правильної побудови системи спочатку потрібно визначити її загальні вимоги. Ці вимоги допомагають з подальшим проєктуванням системи, окреслюючи основні принципи її роботи.

Загальні вимоги до системи:

Веб-орієнтованість:

- Система повинна бути доступна в найбільш використовуваних браузерах: Chrome, Firefox, Edge, Safari, Opera.
- Інтерфейс системи повинен бути адаптивним під різні розширення екрану, так як користувачі системи можливо будуть використовувати мобільні девайси або ж планшети при роботі з додатком.

Клієнт-серверна взаємодія:

- Backend системи повинен обробляти всю бізнес-логіку системи та зберігати дані в DB.
- Frontend системи робить звернення до API для отрмання та модифікації даних в форматі JSON. JSON – це формат даних, який використовується для передання та отримання даних з запитом [25].

Масштабованість:

- Система повинна коректно обробляти та працювати з даними великих розмірів: велика кількість товарів та замовлень.
- Легко додавати нові модулі при потребі, з метою розширення функціоналу системи.

Багатокористувацький режим:

- Система повинна підтримувати можливість роботи декількох співробітників та надавати їм ролі: Admin або Staff.

- Реалізація автентифікації та авторизації користувачів з допомогою REST API та JWT токенів.

Безпека:

- Зберігання та обробка паролів з використанням хеш закодування - Django auth.
- Заборона доступу до сайту для анонімних користувачів. Для доступу потрібно ввійти в акаунт.

Інтерфейс користувача:

- UX/UI має бути інтуїтивним та зрозумілим в використанні.

Логічна цільність даних:

- Забезпечення цільності товару: сума залишків товару в магазині та на складі повинна відповідати загальній кількості товару.
- Після створення нового товару його кількість записується на склад, після чого нею можна маніпулювати.

Модульність:

- Використання окремих модулів під-час створення системи, для зручності та можливого розширення системи.

2.2 Опис функціональних та нефункціональних вимог

Функціональні вимоги:

Реалізація реєстрації та авторизації користувачів:

- Користувач з роллю Admin повинен могти створювати обліковий запис користувача з використанням їх email, username та паролем.
- Логін реалізовано з допомогою JWT, отримання access та refresh token.

Створення ролей користувачів та їх доступу:

- Admin - користувач, який має повний доступ до системи та всіх CRUD операцій.

o Staff – користувач, який має обмежені права в системі. Користувач може бачити інформацію про товар та замовлення, але не має можливості змінювати критичні дані.

Модуль товару:

o Можливість створення редагування та видалення категорій товарів. Доступно тільки користувачам з роллю Admin.

o Пошук за назвою та фільтрація за даними товару.

o Створення нового товару. Дані які потрібно ввести для створення товару: назва, ціна, SKU (унікальне значення), категорія, штрихкод та кількість. Доступно тільки користувачам з роллю Admin.

o Перегляд списку товару.

o Редагування та видалення інформації про існуючий товар. Доступно тільки користувачам з роллю Admin.

Модуль залишку:

o Кожен запис залишку повинен містити поля: товар, локація де зберігається, кількість.

o Можливість створення та видалення запису залишку. Доступно тільки користувачам з роллю Admin.

o Можливість редагування кількості залишків в локаціях. Доступно тільки користувачам з роллю Admin.

o Реалізація передачі товару між локаціями. Після передачі товару, система повинна перевірити що загальна кількість товару дорівнює сумі товару на складі та магазині.

Модуль замовлень:

o Реалізація створення замовлень. Доступ до цієї функції надається всім користувачам системи.

o При замовленні товару можна вказати його кількість. Також одне замовлення не є обмежене тільки одим товаром, дозволяється замовити декілька різних товарів.

o Після підтвердження замовлення, в системі створюється запис зі статусом “В прогресі”. Користувачі з роллю Admin можуть прийняти товар на склад або ж скасувати замовлення.

Модуль сповіщень:

o Повідомлення для користувача створюється коли було створене нову замовлення, кількість товару суттєво змінилася або ж адміністратор надсилає повідомлення користувачу.

o Повідомлення повинні відображатися для конкретних користувачів. Якщо повідомлення презначено для одного користувача воно не повинно відображатися в інших користувачів.

Модуль звіту:

o Експорт замовлень за період часу. Формується Excel файл, який можна завантажити.

o Виведення інформації про всі операції в магазині.

Модуль аналітики:

o Реалізація збору та показу даних в вигляді графіків.

Нефункціональні вимоги:

Продуктивність системи:

o Час відповіді API повинен бути мінімальним при середньому навантаженні

o Сторінки Веб-додатку повинні відповідати на запити переходу з мінімальною затримкою.

Надійність:

o Наявність базовових перевірок на рівні DB: unique, not null, constraints.

Безпека системи:

o Використання JWT для впровадження шифрування запитів системи.

o Валідація даних на backend системи.

o Обмеження прав доступу до системи та API. Тільки користувачі з дійсним access token може звернутися до системи.

o Захист додатку від SQL ін’єкцій

o Безпечне хешування паролів користувачів

Інтерфейс:

o Розроблення інтуїтивно зрозумілого інтерфейсу, з яким зручно працювати

o Створення повідомлень для підтвердження певних критичних дій.

2.3 Визначення ролей користувачів системи

Система передбачає три основні категорії користувачів: адміністратор - Admin, працівник - Staff та гість - Guest.

Користувач з роллю адміністратора має повний контроль над системою. Адміністратор може створювати та редагувати товар, та їх категорії, інформацію про користувачів, та створювати нові аккаунти для використання в системі, у вкладці адміна. Також адміністратор має доступ до інформації про продажі та замовлення.

Крім цього адміністратор має змогу вручну призначати ролі користувачам системи, що забезпечує гнучкість в управлінні персоналом в рамках системи.

Роль працівника призначена для користувачів магазину, яким потрібний доступ до виконання простих операцій, таких як, перегляд товару та категорій, створення замовлень та передачі товару між складом та магазином.

Також працівник отримує сповіщення в системі, щодо змін, які його стосуються, як зміна статусу замовлення.

Користувачі, які не ввійшли в свій аккаунт, залишаються анонімними та не мають доступу до жодного внутрішнього функціоналу системи, окрім форми входу в систему. Для отримання аккаунту працівнику потрібно, щоб користувач з роллю адміністратора створив для них аккаунт.

Use Case системи зображено на рис. 2.1.

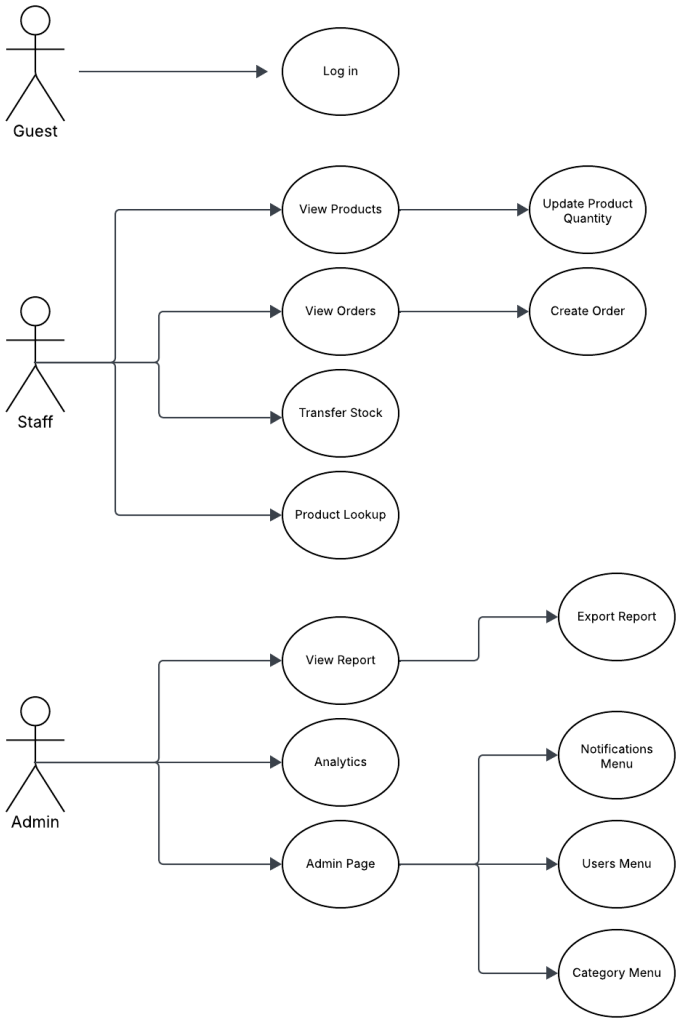


Рисунок 2.1 - Use Case системи

Опис користувачів системи та їх доступу надає можливість переконатися, що всі вимоги системи будуть задовільнені на етапі реалізації системи.

2.4 Висновки по розділу

У цьому розділі було визначено загальні, та більш конкретні вимоги до системи. Також було розділено користувачів системи на ролі та окреслено їхні права та обмеження. Дотримання всіх цих вимог гарантує створення системи, що відповідає всім бізнес потребам, є безпечною та зручною в використанні для її користувачів.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ

3.1. Вибір технологій створення системи

Для створення проекту було обрано клієт-серверний підхід, використовуючи окремі технології для frontend та backend системи. Для реалізації backend частини проекту було використано Django REST Framework, для frontend – ReactJS. Як базу даних було використано SQLite3, база даних, яка є вбудованою в Django. Цей підхід до архітектури проекту забезпечує чіткість структури проекту, так як ReactJS відповідає за відображення даних для роботи з користувачем, коли з допомогою Django проводиться робота з бізнес логікою та даними.

Django – це високорівневий Python фреймворк, який використовується для створення веб систем. Він надає доступ до механізмів аутентифікації та авторизації користувачів і систем міграції [12]. Рекомендованою структурою сайту створеного на Django є модульна. Це є одною з найбільших відмінностей Django від інших фреймворків. Самі додатки Django будуються використовуючи архітектуру “Model-view-controller”, що поділяє системи на три умовні частини, що в свою чергу робить створення та редагування додатку набагато більш зручним та зрозумілим [17].

Django Rest Framework – це інструмент який надає можливість зручно та швидко створювати API для взаємодії з frontend системи [13]. REST – це архітектурний запитів по мережі, що надає доступ до певної інформації [18].

Djoser та JWT забезпечують збереження даних користувача, з метою взаємодії, на серверній частині з допомогою JWT tokens.

SQLite3 - це реляційна БД, побудована з використанням мови програмування C, яка є вбудованою в Django Framework та не вимагає окремого сервера для використання [26]. В Django взаємодія з DB відбувається з допомогою використання моделей.

З клієнтської частини було використано ReactJS. Це відкрита JavaScript бібліотека, яка використовується для розробки односторінкових застосунків. Бібліотека надає доступ до використання хуків та компонентів, які застосовуються з метою полегшення створення інтерфейсу користувача [14].

Axios це HTTP клієтнт, який використовується для створення запитів до REST API [27]. Axios дозволяє передавати JWT токени в заловку запиту з допомогою interceptor, що надає доступ до системи користувачу.

Окрім основних технологій, також було використано допоміжні бібліотеки для забезпечення певної функціональності та зручності використання системи.

З боку frontend було використано бібліотеку React Router. Використання цієї бібліотеки надає можливість додання маршрутизації сторінок додатку [19].

На backend системи було використано бібліотеку django-filter, яка при інтеграції з Django REST Framework дозволяє зручно реалізовувати фільтрацію API запитів [20].

Для забезпечення правильності версій технологій системи було використано Venv. Venv – це ізольоване віртуальне середовище Python, в якому можна запускати та тестувати свої проекти. Цей підхід до побудови системи робить саму систему більш портативною та запобігає конфліктам версій між пакетами різних проектів та різними версіями системи. Саме середовище надає можливість керування залежностями специфічними до проекту та версій технологій, які використовуються [28].

Вибір ReactJS для створення frontend системи, Django REST Framework для backend та SQLite3 як базу даних забезпечує найдійність, гнучкість та безпечність системи. Використання віртуального середовища venv надає легкість контролю версій технологій системи та використання axios та React Router надають можливість створювати запити та забезпечувати маршрутизацію додатку. На стороні backend системи JWT та Djoser забезпечують можливість

автентифікації користувачів. В резльтаті обрані технології відповідають всім поставленим вимогам проекту.

3.2 Архітектура системи

Архітектура створеної системи була реалізована в вигляді класичної моделі «клієнт-сервер», з чітким поділом між Backend, з Базою даних та Frontend. Такий підхід забезпечує розділення відповідальності компонентів системи та полегшує підтримки коду.

На стороні Frontend системи, яку теж називають Presentation Layer, було використано ReactJS. Ця частина системи відповідає за побудову інтерфейсу, який використовується користувачами для взаємодії з додатком, маршрутизацією по сторінках системи та керування локальним сховищем, для збереження даних про акаунт користувача, який ввійшов в систему.

Сторона Backend складається з бази даних та додатку який відповідає за логіку системи, створеному з допомогою Django. Базу даних доволі часто називають Data Layer системи, коли ж сам сервер системи називають Application Layer.

Application Layer було реалізовано за допомогою Django REST Framework. Додаток потрібно поділити на окремі додатки Django, кожен з яких виконує свої функції та відповідає за свою частину системи. Шлях запиту до сервера виглядає так: запит HTTP надходить до DefaultRouter системи, який в свою чергу перенаправляє цей запит до відповідного ViewSet.

В Data Layer системи було використано реляційну базу даних SQLite3, якаа є швидкою в розгортанні. Структуру таблиць бази даних визначають Django моделі системи. Після створення моделей в Django, в базі даних створюються таблиці, стовбці яких відповідають змінним вказаним в додатках Django. Для створення, модифікації або ж видалення інформації в базі даних

використовуються моделі, які надають можливість виконання цих дій з допомогою їх методів.

Безпека системи забезпечується використанням JWT аутентифікації, CORS та CSRF налаштуваннями та доступом до певних функцій за наявності в користувача правильної ролі.

Загальний вигляд системи зображено на рис. 3.1.

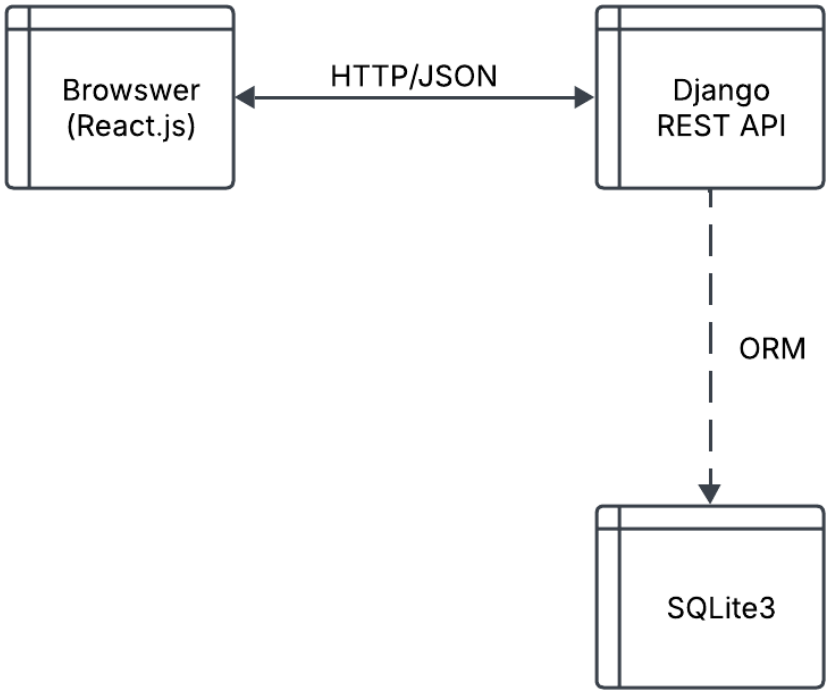


Рисунок 3.1 - Загальний вигляд системи

Результатом є система в якій frontend відповідає за показ та збір даних з допомогою форм, backend, який відповідає за обробку зібраних даних та їх зберігання та база даних, яка відповідає за збереження даних, для подальшого використання. Така архітектура є легкою в побудові, розгортанні та розумінні і доволі захищеною через наявність JWT, CORS, CSRF та правам доступу користувачів.

3.3 Проектування системи

Для реалізації інформаційної системи було спроектовано окремі додатки в Django, кожний з яких відповідимуть за самостійну частину функціоналу:

1. users - містить модель CustomUser, яка розширює клас AbstractUser, серіалізатори та views.py для керування користувачами.

2. products - містить моделі Category і Product. Файл Views.py містить функції які використовуються для проведення операцій над товарами та категоріями.

3. inventory – містить модель Stock та функцію transfer в файлі views.py для переміщення товару між складом та магазином.

4. orders - моделі Order, OrderItem були створені для запису даних про замовлення.

5. notifications – містить модель Notification, серіалізатор та views.py для перегляду та маркування повідомлень як прочитаних.

6. reports - у цьому модулі було реалізовано функцію експорту даних, що генерує звіт про замовлення, за певний період часу, в Excel.

7. analytics - містить модель SalesRecord, в якому зберігається інформація про продажі.

8. barcodes - використовується для створення зображення штрих коду товару та функція lookup використовується для пошуку інформації про товар за введеним штрихкодом.

Зі сторони frontend системи, в ReactJS було створено декілька розділів:

1. AuthContext - було створено для реалізування React Context та збереження інформації користувача та функцій login() і logout().

2. Router - для налаштування маршрутів сайту.

3. NavBar - компонент для посилання на розділи.

4. Pages - папка, яка містить сторінки веб додатку:

4.1 ProductsPage - переліку товару з інформацією

4.2 InventoryPage - таблиця залишків та редагування кількості товару

4.3 OrdersPage - список замовлень та форма створення замовлення

4.4 ReportsPage - експорт Excel файлу з інформацією про замовлення

4.5 AnalyticsPage - графіки з інформацією про продажі

4.6 BarcodePage - пошук товару за штрихкодом та генерація фото штрихкоду для легшого сканування

4.7 LoginPage - вхід в акаунт користувача

5. Components - папка, яка містить компоненти, які використовуються в сторінках проекту.

Загальний вигляд документів системи показано на рис. 3.2.

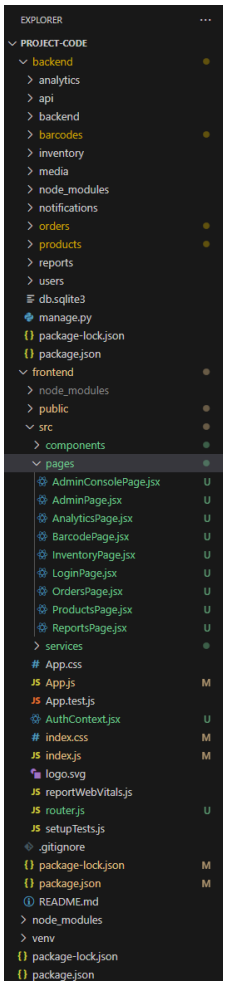


Рисунок 3.2 - Загальний вигляд документів системи

3.4 Висновки по розділу

Під час проектування інформаційної системи було визначено технології, які є оптимальними для створення системи, що задовільняє всі умови. Додатково було визначено структуру побудови клієнтської та серверної частин системи. Для реалізації backend було використано Django, для реалізації інтерфейсу користувача було вибрано ReactJS. З метою забезпечення зв'язку між backend та frontend системи було використано REST API, а використання JWT дозволило забезпечити систему необхідним рівнем безпеки від потенційних загроз.

Архітектура системи побудована орієнтовуючись на принципи «клієнт-серверної» взаємодії та є доволі часто використовуваною. Саму архітектуру можна розділити на frontend та backend, або ж на 3 рівні: Presentation layer, application layer та data layer.

Результатом стала спроектована веб система яка відповідає всім вимогам, є гнучкою та зрозумілою в структурі.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ

4.1 Реалізація системи

Підчас створення системи, спочатку було реалізовано структуру бази даних.

Для збереження даних в системі було створено наступні моделі Django:

- CustomUser - містить поля: role, phone та поля які наслідуються з класу AbstractUser: username, email та password. Вигляд класу продемонстровано на рис. 4.1.

```
3   from django.contrib.auth.models import AbstractUser
4   from django.db import models
5
6   class CustomUser(AbstractUser):
7       ROLE_CHOICES = (
8           ('admin', 'Admin'),
9           ('staff', 'Staff'),
10      )
11      role = models.CharField(max_length=10, choices=ROLE_CHOICES)
12      phone = models.CharField(max_length=20, blank=True, null=True)
```

Рисунок 4.1 - Вигляд класу CustomUser

- Category - поле name для запису імені категорії, description для опису категорії.
- Product - містить поля: name, sku, price, category, який є об’єктом класу Category, barcode та quantity_in_stock.

Вигляд класів Category та Product показано на рис. 4.2.

backend > products > models.py > Product > __str__

```python
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField(blank=True)

    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=100)
    sku = models.CharField(max_length=100, unique=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.ForeignKey(Category, on_delete=models.CASCADE, related_name='products')
    barcode = models.CharField(max_length=100, blank=True)
    quantity_in_stock = models.IntegerField(default=0)

    def __str__(self):
        return self.name
```

Рисунок 4.2 - Вигляд класів Category та Product

- Stock - містить поля product - об’єкт класу Product, location для збереження локації товару та quantity - для кількості товару в локації. Створення класу продемонстровано на рис. 4.3.

backend > inventory > models.py > Stock > __str__

```python
from django.db import models
from products.models import Product

class Stock(models.Model):
    LOCATION_CHOICES = [
        ('warehouse', 'Склад'),
        ('shop', 'Магазин'),
    ]
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    location = models.CharField(max_length=10, choices=LOCATION_CHOICES)
    quantity = models.PositiveIntegerField(default=0)

    class Meta:
        unique_together = ('product', 'location')

    def __str__(self):
        return f"{self.product.name} ({self.location}): {self.quantity}"
```

Рисунок 4.3 - Вигляд класу Stock

- Order – містить поле user, який є об’єктом класу CustomUser, status – для збереження статусу замовлення та змінну created_at, в якій зберігається час

коли було створено замовлення.

- OrderItem – містить: order - об’єкт класу Order, product – об’єкт класу Product та quantity - змінна для збереження кількості замовленого товару.

Реалізацію класів Order та Orderitem продемонстровано на рис. 4.4.

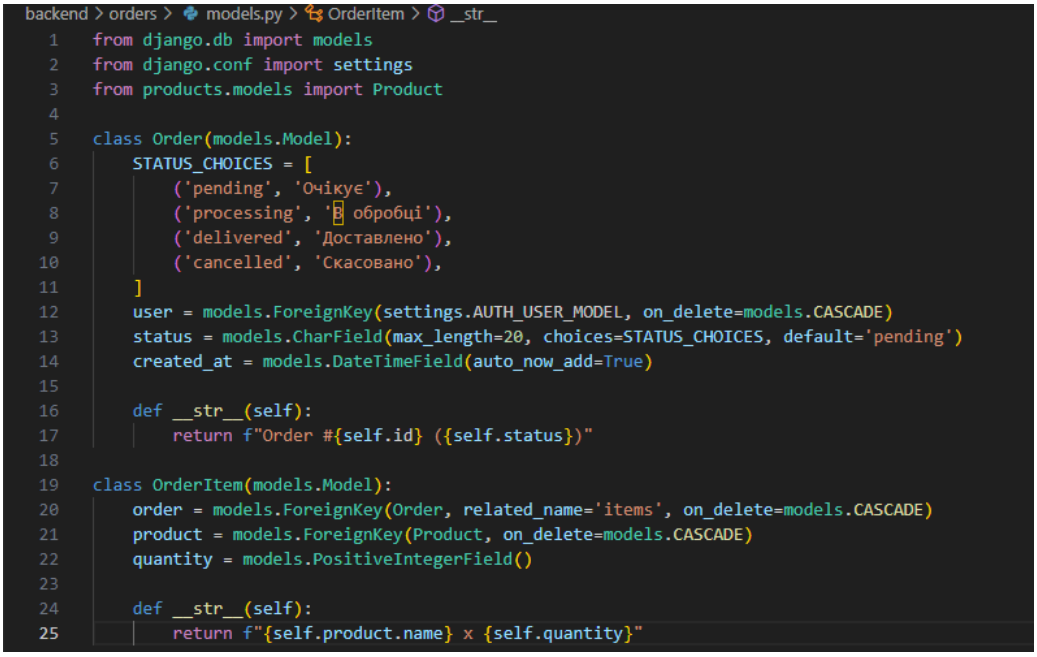


Рисунок 4.4 - Реалізація класів Order та OrderItem

- Notifications – user - об’єкт класу CustomUser, message для збереження тексту повідомлення, sent_at та read для того, щоб знати чи було прочитано повідомлення. Структура класу продемонстрована на рис. 4.5.

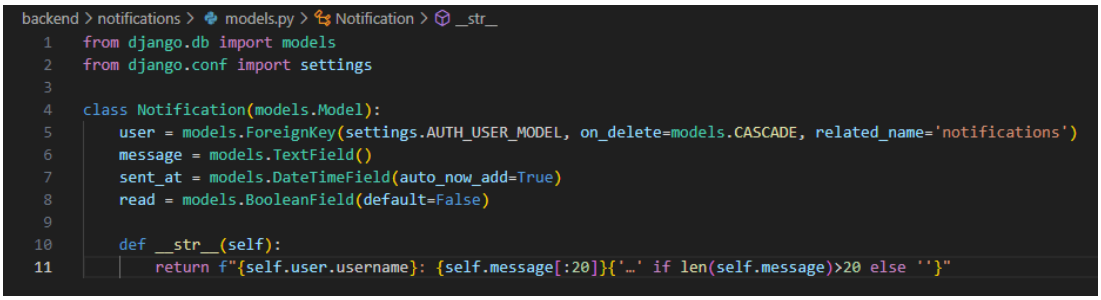


Рисунок 4.5 - Структура класу Notification

- SalesRecord – product - об’єкт класу Product, date - дата продажу та

quantity - кількість проданих одиниць товару. Реалізація класу показана на рис. 4.6.

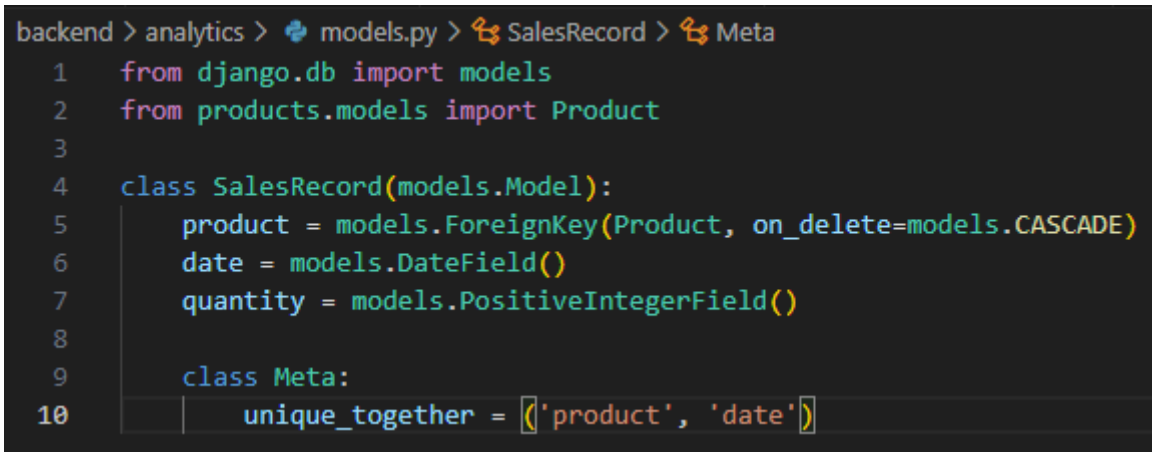


Рисунок 4.6 - Реалізація класу SalesRecord

Після цього було реалізовано логіку та загальний фунціональний вигляд системи. Реалізація системи відбулася на основі спроектованої архітектури та структури бази даних.

Для реалізації логіки кожного з додатків спочатку в файлі views.py створюється клас з довільною назвою, який наслідує клас ModelViewSet. В цьому класі створюються змінні та методи, які будуть викликатися при переході по певним посиланням системи.

Далі було створено файл urls.py де в urlpatterns вказуються всі посилання додатку та функції, які викликаються при переході по цим посиланням. Самі функції, які викликаються були попередньо написані в файлі views.py.

Додатково в певних додатках потрібно було створити файли signals.py та serializers.py.

Файл signals.py дозволяє системі автоматично виконувати певні дії, реагуючи на події в системі.

Файл serializers.py допомагає побудувати зв’язок між базою даних та API системи. Дуже часто використовуються для валідації даних та перетворення моделей Django у форматі JSON.

Приклад реалізації логічних фалів додатку orders:

Вигляд файлів додатку продемонстровано на рис. 4.7.

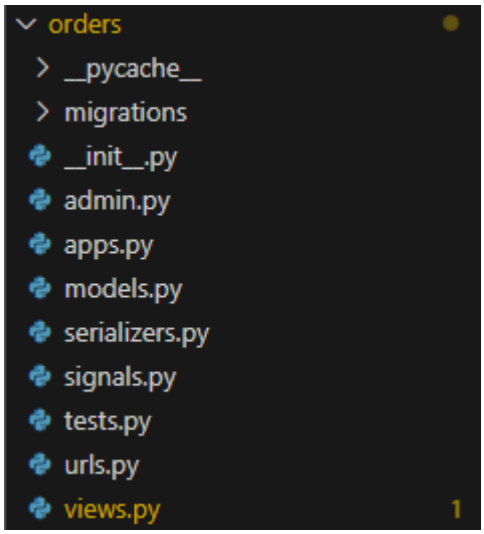


Рисунок 4.7 - Вигляд файлів додатку Orders

```python
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import filters

from .models import Order
from .serializers import OrderSerializer
from inventory.models import Stock
from users.permissions import IsAdmin

class OrderViewSet(viewsets.ModelViewSet):
    queryset = Order.objects.prefetch_related('items__product').all()
    serializer_class = OrderSerializer
    permission_classes = [IsAuthenticated]

    filter_backends = [
        DjangoFilterBackend,
        filters.SearchFilter,
        filters.OrderingFilter,
    ]
    filterset_fields = {
        'status': ['exact'],
        'created_at': ['gte', 'lte'],
    }
    search_fields = ['user__username', 'id']
    ordering_fields = ['created_at', 'status']

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def get_permissions(self):
        if self.action in ['update', 'partial_update', 'destroy']:
            return [IsAdmin()]
        return [IsAuthenticated()]

    @action(detail=True, methods=['post'], permission_classes=[IsAdmin])
    def receive(self, request, pk=None):
        order = self.get_object()

        if order.status != 'delivered':
            order.status = 'delivered'
            order.save()

        for item in order.items.all():
            Stock.objects.get_or_create(
                product=item.product,
                location='warehouse',
                defaults={'quantity': 0}
            )

        serializer = self.get_serializer(order)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

Рисунок 4.8 - Реалізація views.py додатку Orders

Спочатку було написано функціонал додатку в файлі views.py. Реалізація

views.py продемонстрована на рис. 4.8.

Далі було реалізовано логіку посилань в файлі urls.py. Файл urls.py показаний на рис. 4.9.

backend > orders > urls.py > ...

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import OrderViewSet

router = DefaultRouter()
router.register('orders', OrderViewSet, basename='order')

urlpatterns = [
    path('', include(router.urls)),
]
```

Рисунок 4.9 - Файл urls.py додатку Orders

Після цього було створено файл serializers.py. Файл serializers.py показаний на рис. 4.10.

```python
from rest_framework import serializers
from .models import Order, OrderItem

class OrderItemSerializer(serializers.ModelSerializer):
    product_name = serializers.ReadOnlyField(source='product.name')

    class Meta:
        model = OrderItem
        fields = ['id', 'product', 'product_name', 'quantity']

class OrderSerializer(serializers.ModelSerializer):
    items = OrderItemSerializer(many=True)
    user = serializers.ReadOnlyField(source='user.username')

    class Meta:
        model = Order
        fields = ['id', 'user', 'status', 'created_at', 'items']

    def create(self, validated_data):
        items_data = validated_data.pop('items')
        order = Order.objects.create(**validated_data)
        for item in items_data:
            OrderItem.objects.create(order=order, **item)
        return order
```

Рисунок 4.10 - Файл serializers.py

Далі, з метою впровадження сигналів в додатку, було створено файл signals.py. Файл signals.py продемонстрований на рис. 4.11.

```python
from django.db.models.signals import post_save, pre_save
from django.dispatch import receiver
from django.utils import timezone
from .models import Order, OrderItem
from inventory.models import Stock
from analytics.models import SalesRecord

@receiver(post_save, sender=OrderItem)
def handle_order_item(sender, instance, created, **kwargs):
    if not created:
        return

    product = instance.product
    qty = instance.quantity

    try:
        stock = Stock.objects.get(product=product, location='warehouse')
        stock.quantity = max(stock.quantity - qty, 0)
        stock.save()
    except Stock.DoesNotExist:
        pass

@receiver(pre_save, sender=Order)
def save_old_status(sender, instance, **kwargs):
    if not instance._state.adding:
        try:
            old = sender.objects.get(pk=instance.pk)
            instance._old_status = old.status
        except sender.DoesNotExist:
            instance._old_status = None

@receiver(post_save, sender=Order)
def receive_delivered(sender, instance, created, **kwargs):
    if created or getattr(instance, '_old_status', None) == instance.status:
        return

    if instance.status == 'delivered':
        for item in instance.items.all():
            stock, created = Stock.objects.get_or_create(
                product=item.product,
                location='warehouse',
                defaults={'quantity': 0}
            )
            stock.quantity += item.quantity
            stock.product.quantity_in_stock += item.quantity
            stock.product.save()
            stock.save()
```

Рисунок 4.11 - signals.py додатку Orders

4.2 Інтерфейс користувача

Інтерфейс користувача був розроблений з урахуванням ролей користувачів та вимог до UX і UI дизайну. Основною метою є забезпечення інтуїтивно зрозумілого інтерфейсу для навігації між сторінками.

Загальна структура та навігація:

1. Навігація по frontend системи побудовано з допомогою React Router. Файл router.js містить функцію AppRouter, яка повертає об'єкт Router, який містить всі маршрути веб-сайту. На рис. 4.12. показано реалізацію файлу router.js.

frontend > src > JS router.js > ...

```
import React, { useContext } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthContext } from './AuthContext';
import NavBar from './components/NavBar';
import LoginPage from './pages/LoginPage';
import ProductsPage from './pages/ProductsPage';
import OrdersPage from './pages/OrdersPage';
import ReportsPage from './pages/ReportsPage';
import AnalyticsPage from './pages/AnalyticsPage';
import BarcodePage from './pages/BarcodePage';
import InventoryPage from './pages/InventoryPage';
import AdminConsolePage from './pages/AdminConsolePage';

export default function AppRouter() {
  const { token, user } = useContext(AuthContext);
  const isAuth = Boolean(token);
  const role = user?.role;

  return (
    <Router>
      <NavBar />
      <Routes>
        <Route path="/login" element={<LoginPage />} />
        <Route path="/" element={isAuth ? <Navigate to="/products" /> : <Navigate to="/login" />} />
        <Route path="/products" element={isAuth ? <ProductsPage /> : <Navigate to="/login" />} />
        <Route path="/orders" element={isAuth ? <OrdersPage /> : <Navigate to="/login" replace />} />
        <Route path="/inventory" element={isAuth ? <InventoryPage /> : <Navigate to="/login" />} />
        {role === 'admin' && (
          <>
            <Route path="/reports" element={isAuth ? <ReportsPage /> : <Navigate to="/login" />} />
            <Route path="/analytics" element={<AnalyticsPage />} />
            <Route path="/barcode" element={<BarcodePage />} />
            <Route path="/admin" element={isAuth ? <AdminConsolePage /> : <Navigate to="/login" />} />
          </>
        )}

        <Route path="*" element={<Navigate to="/" />} />
      </Routes>
    </Router>
  );
}

```

Рисунок 4.12 - Реалізація файлу router.js

2. Після цього було реалізовано меню для переходу по сторінкам сайту. Меню навігації відрізняється з залежності від ролі користувача. Для користувачів з ролью працівника меню навігації відображає тільки посилання на сторінки Products, Inventory та Orders. Вигляд інтерфейсу користувача з роллю працівника продемонстровано на рис 4.13.

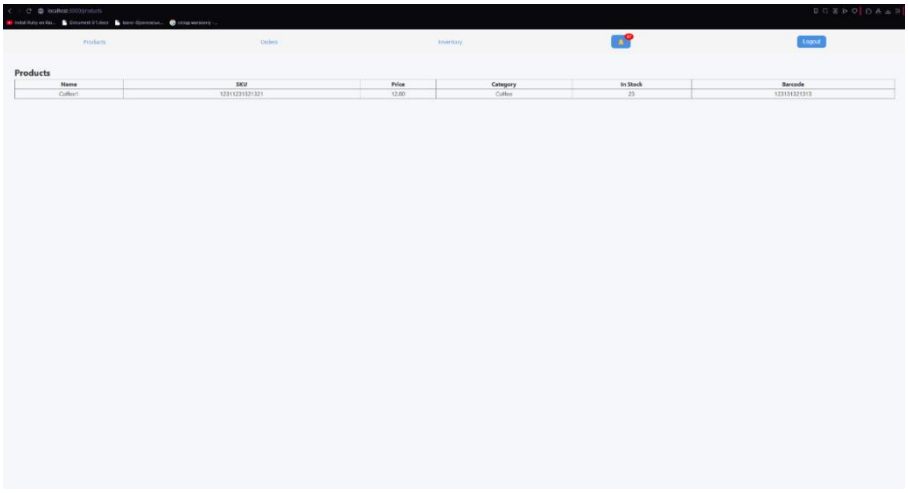


Рисунок 4.13 - Інтерфейс користувача з роллю працівника

Для користувачів з роллю адміністратора також відкривається доступ до сторінок Reports, Analytics, Barcodes та Admin Console. Вигляд інтерфейсу показаний на рис 4.14.

Рисунок 4.14 - Інтерфейс користувача з доступом адміністратора

Вигляд коду меню NavBar зображено на рис. 4.15.

frontend > src > components > NavBar.jsx > NavBar

```jsx
import React, { useContext } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import NotificationsMenu from './NotificationsMenu';
import { AuthContext } from '../AuthContext';

export default function NavBar() {
  const { user } = useContext(AuthContext);
  const role = user?.role;
  const navigate = useNavigate();
  const logout = () => {
    localStorage.removeItem('accessToken');
    localStorage.removeItem('refreshToken');
    navigate('/login');
  };

  return (
    <nav style={{ padding: '1rem', background: '#f5f5f5', display: 'flex', alignItems: 'center' }}>
      <Link to="/products" style={{ marginRight: '1rem' }}>Products</Link>
      <Link to="/orders" style={{ marginRight: '1rem' }}>Orders</Link>
      {role === 'admin' && <Link to="/reports" style={{ marginRight: '1rem' }}>Reports</Link>}
      {role === 'admin' && <Link to="/analytics" style={{ marginRight: '1rem' }}>Analytics</Link>}
      {role === 'admin' && <Link to="/barcode" style={{ marginRight: '1rem' }}>Barcode</Link>}
      <Link to="/inventory" style={{ marginLeft: '1rem' }}>Inventory</Link>
      {role === 'admin' && <Link to="/admin" style={{ marginRight: '1rem', fontWeight: 'bold' }}>Admin Console</Link>}
      <NotificationsMenu />
      <button onClick={logout} style={{ marginLeft: '2rem' }}>Logout</button>
    </nav>
  );
}
```

Рисунок 4.15 - Код меню NavBar

3. Після цього було створено компоненти та сторінки інтерфейсу. Загальний вигляд структури показано на рис. 4.16.

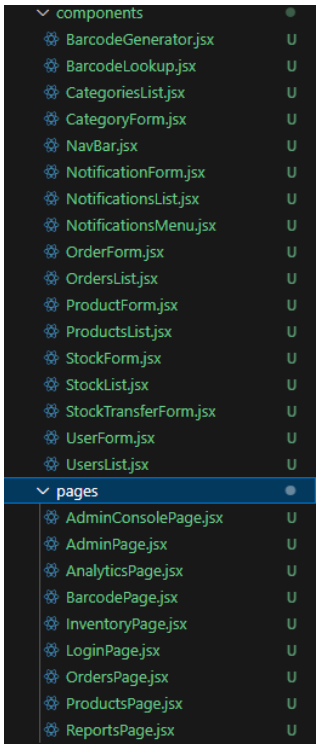


Рисунок 4.16 - Вигляд структури папок компонентів та сторінок

Папку components було створено з метою збереження компонентів, які будуть використовуватися в розробці сторінок сайту. Папка pages містить сторінки веб-системи, які відображаються при переході по створених раніше посиланнях.

Приклад стоврення сторінки ProductsPage:

Спочатку було створено саму сторінку. В папці pages було створено файл ProductsPage.jsx зі змістом показаним на рис. 4.17.

frontend > src > pages > ProductsPage.jsx > ...

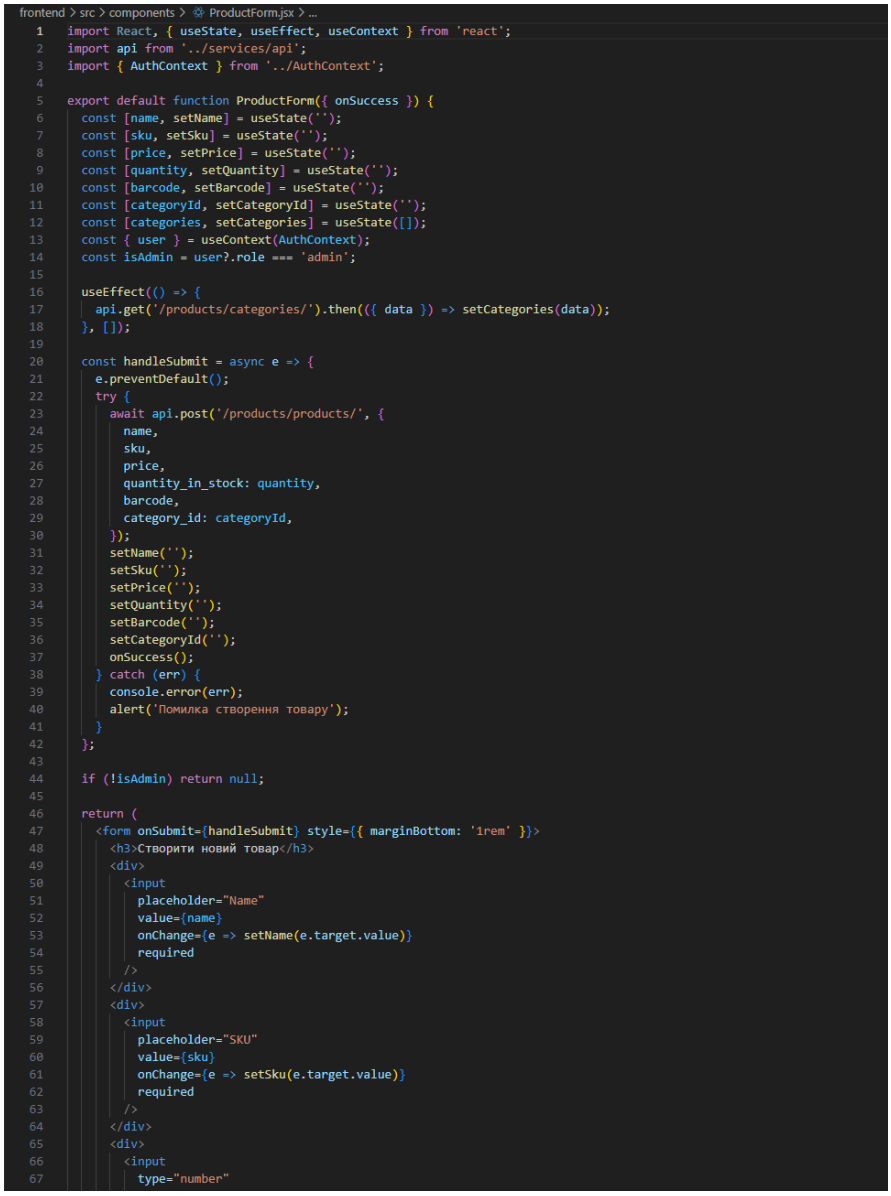
```jsx
import React, { useState } from 'react';
import ProductForm from '../components/ProductForm';
import ProductsList from '../components/ProductsList';

export default function ProductsPage() {
  const [refreshToggle, setRefreshToggle] = useState(false);
  const handleSuccess = () => setRefreshToggle(prev => !prev);

  return (
    <div style={{ padding: '2rem' }}>
      <h2>Products</h2>
      <ProductForm onSuccess={handleSuccess} />
      <ProductsList refreshToggle={refreshToggle} />
    </div>
  );
}
```

Рисунок 4.17 - Зміст файлу ProductsPage.jsx

Далі було створено раніше зазначені на сторінці компоненти. Компонент ProductForm використовується для створення нового товару. Для створення товару потрібно заповнити форму вказавши назву, категорію, ціну та баркод товару. Реалізація компоненту наведена на рис. 4.18.



```
frontend > src > components > ProductForm.jsx > ...
import React, { useState, useEffect, useContext } from 'react';
import api from '../services/api';
import { AuthContext } from '../AuthContext';

export default function ProductForm({ onSuccess }) {
  const [name, setName] = useState('');
  const [sku, setSku] = useState('');
  const [price, setPrice] = useState('');
  const [quantity, setQuantity] = useState('');
  const [barcode, setBarcode] = useState('');
  const [categoryId, setCategoryId] = useState('');
  const [categories, setCategories] = useState([]);
  const { user } = useContext(AuthContext);
  const isAdmin = user?.role === 'admin';

  useEffect(() => {
    api.get('/products/categories/').then(({ data }) => setCategories(data));
  }, []);

  const handleSubmit = async e => {
    e.preventDefault();
    try {
      await api.post('/products/products/', {
        name,
        sku,
        price,
        quantity_in_stock: quantity,
        barcode,
        category_id: categoryId,
      });
      setName('');
      setSku('');
      setPrice('');
      setQuantity('');
      setBarcode('');
      setCategoryId('');
      onSuccess();
    } catch (err) {
      console.error(err);
      alert('Помилка створення товару');
    }
  };

  if (!isAdmin) return null;

  return (
    <form onSubmit={handleSubmit} style={{ marginBottom: '1rem' }}>
      <h3>Створити новий товар</h3>
      <div>
        <input
          placeholder="Name"
          value={name}
          onChange={e => setName(e.target.value)}
          required
        />
      </div>
      <div>
        <input
          placeholder="SKU"
          value={sku}
          onChange={e => setSku(e.target.value)}
          required
        />
      </div>
      <div>
        <input
          type="number"
```

Рисунок 4.18 - Реалізація компонента ProductForm

Після цього було реалізовано компонент ProductsList, який відповідає за вивід всього товару системи. Додатково було добавлено можливість видалення конкретного товару з системи при натисненні кнопки “delete”. Реалізація компонента продемонстрована на рис. 4.19.

Рисунок 4.19 - Реалізація файлу ProductsList.jsx

4. Далі, для покращення вигляду інтерфейсу було додано CSS файл, з стилем сторінок. Сам файл зображено на рис 4.20.

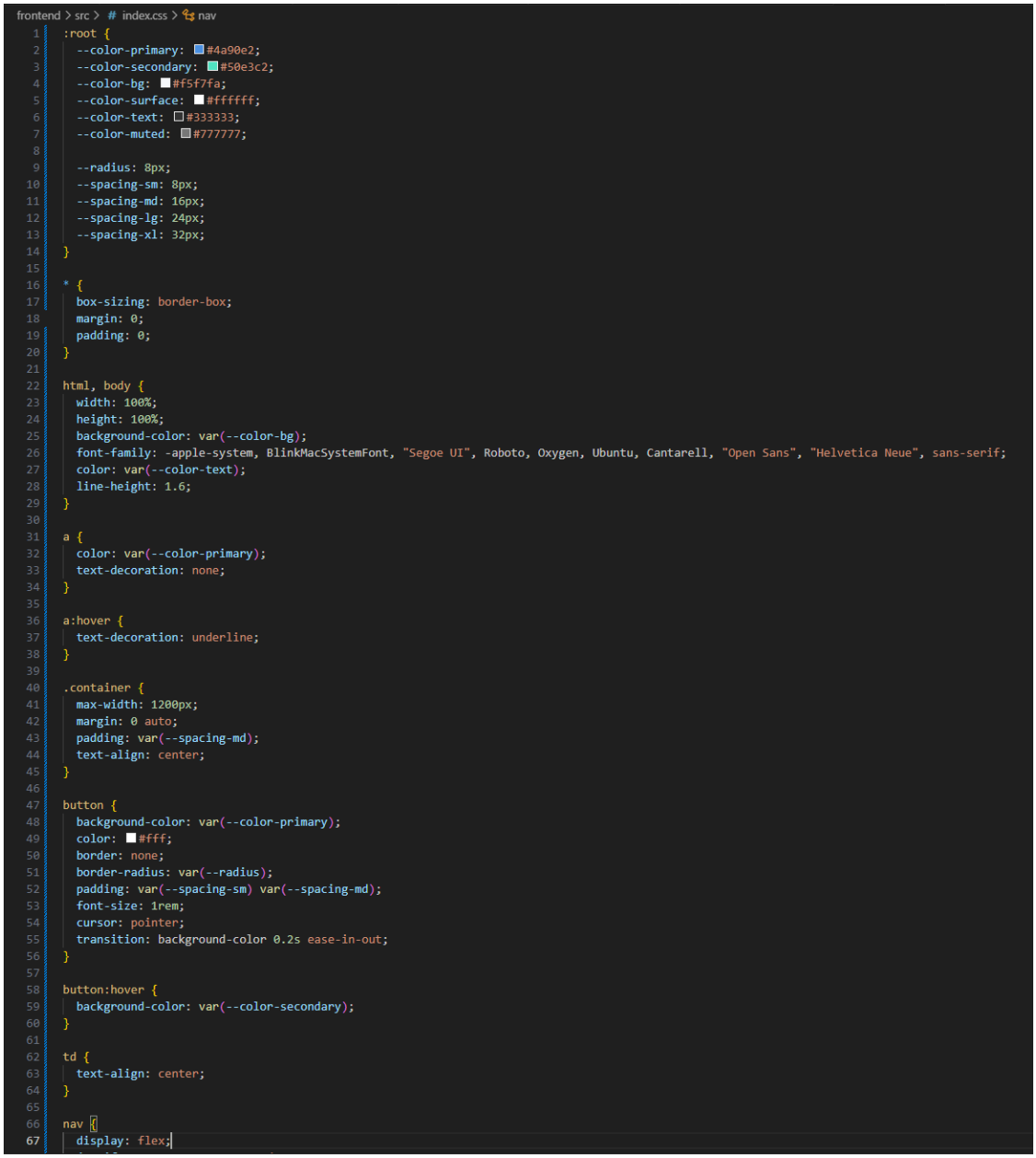


Рисунок 4.20 - Вигляд файлу index.css

4.3 Покращення безпеки системи

З метою покращення безпеки системи було використано:

1. Спочатку для захисту API було використано JWT автентифікацію з допомогою бібліотеки djoser та rest_framework_simplejwt. Для цього у settings.py було зазначено код показаний на рис. 4.21.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
        'rest_framework.filters.OrderingFilter',
    ],
}
```

Рисунок 4.21 - REST_FRAMEWORK в settings.py

В ReactJS було додано файл api.js для налаштування запитів до API, як зображено на рис. 4.22.

```
import axios from 'axios';
import createAuthRefreshInterceptor from 'axios-auth-refresh';

const api = axios.create({
  baseURL: 'http://localhost:8000/api',
});

api.interceptors.request.use(config => {
  const token = localStorage.getItem('accessToken');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
```

Рисунок 4.22 - Зміст файлу api.js

2. Після цього було додано використання CORS та CSRF, для цього в settings.py було додано настройки corsheaders. Це показано на рис. 4.23.

```
MIDDLEWARE.insert(0, 'corsheaders.middleware.CorsMiddleware')
CORS_ALLOWED_ORIGINS = ["http://localhost:3000"]
```

Рисунок 4.23 - Додання CORS та CSRF в settings.py

Таким чином система дозволяє отримувати запити тільки з React домену.

3. Додатково додаю права доступу в Django:

Для цього в додатку user було створено permissions.py, в якому було реалізовано власні класи права доступу. Це вказано на рис. 4.24.

```
from rest_framework.permissions import BasePermission

class IsAdmin(BasePermission):
    def has_permission(self, request, view):
        return request.user.is_authenticated and request.user.role == 'admin'

class IsStaff(BasePermission):
    def has_permission(self, request, view):
        return request.user.is_authenticated and request.user.role == 'staff'

class IsAdminOrStaff(BasePermission):

    def has_permission(self, request, view):
        return (
            request.user.is_authenticated and
            request.user.role in ('admin', 'staff')
        )
```

Рисунок 4.24 - Реалізація класів прав доступу користувачів

Приклад використання прав доступу в додатку наведено на рис 4.25.

```
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.select_related('category').all()
    serializer_class = ProductSerializer

    filter_backends = [
        DjangoFilterBackend,
        filters.SearchFilter,
        filters.OrderingFilter,
    ]
    filterset_fields = ['category__id', 'category__name']
    search_fields = ['name', 'sku', 'barcode']
    ordering_fields = ['name', 'price', 'quantity_in_stock']

    def get_permissions(self):
        if self.request.method in ['POST', 'PUT', 'PATCH', 'DELETE']:
            return [IsAdminOrStaff()]
        return [IsAuthenticated()]
```

Рисунок 4.25 - Використання прав доступу в файлі views.py додатку product

З допомогою використання прав доступу, CORS, CSRF та JWT токенів було встановлено безпечність та захищеність системи від потенційних хакерських атак.

4.4 Висновок до розділу

Після вибору технологій було повністю реалізовано серверну та клієнтську чатини веб-додатку. Завдяки використанню Django REST Framework та React логіка системи є чітко зрозумілою - Backend системи відповідає за бізнес процеси, опрацювання даних та їх захист, коли frontend надає можливість користувачу інтерактувати з системою, використовуючи інтуїтивний інтерфейс. Модульність Backend частини системи полегшує подальшу підтримку та розширення коду.

Результатом є реалізована веб-система для обліку залишків та обробки замовлень.

РОЗДІЛ 5. ТЕСТУВАННЯ СИСТЕМИ

5.1 Мета тестування

Тестування створеної системи було проведене із чіткими вимогами до необхідних критеріїв якості, таким як відповідність функціоналу поставленим раніше вимогам, правильна реакція системи на помилкові або ж нестандартні дії які виконуються користувачем та належні рівні безпеки та продуктивності. Основною ціллю було перевірка кожного з модулів описаних в постанові завдання та верефікація правильності їх роботи.

З метою виконання поставлених цілей було окреслено декілька основних завдань тестування:

1. Перевірка роботи ролей та прав доступу системи.

Авторизація та аутентифікація користувачів в системі виконується з допомогою JWT токенів. Під час тестування потрібно переконатися, що лише користувачі з роллю admin можуть змінювати або видаляти товари, обробляти замовлення, створювати нових користувачів системи та завантажувати звіти про продажі. Користувачі з роллю staff натомість мають лише права читання, створення замовлень та перегляду інформації про товар.

2. Перевірка основних CRUD операцій.

Кожен з модулів системи має певну кількість CRUD операцій. Саме тому для правильної роботи додатку потрібно переконатися, що всі ці функції працюють вірно.

3. Тестування правильності транзакцій

При трансфері між локаціями товар переміщається, змінюючи його кількість в локаціях. Необхідно переконатися що операції проходять правильно та дані не опиняються в проміжному стані.

4. Перевірка продуктивності системи

Для типових запитів по типу отримання списку товарів, інвентарю або ж замовлень, потрібно переконатися, що відповідь API займає не більше 300 мс за звичайного навантаження.

5. Перевірка безпеки та стійкості системи

Тестування повинно перевірити захищеність системи від постороннього доступу. Тестується правильність використання CORS, CSRF та валідації вхідних даних в системі.

6. Перевірка інтеграції технологій.

Перевірка роботи frontend та backend системи та їх інтеракції. Потрібно провірити реакцію компонентів React на помилкову відповідь API, коректну роботу axios для автоматичного оновлення токену з допомогою refresh токену.

7. Тестування валідності введених даних.

Необхідно перевірити реакцію системи на некоректно введені дані, наприклад введення пустих обов'язкових полів або ж введення полів в некоректному форматі. Такі ситуації повинні викликати помилку в системі, яка відображається користувачу.

Проведення тестування створеної системи дозволяє перевірити її функціональність, зручність у використанні та безпеку від потенційних загроз. Такий підхід до перевірки працездатності системи допомагає уникнути багатьох недоліків, перед використанням додатку.

5.2 Ручне тестування функціональності системи

Ручне тестування – це тип тестування системи, який передбачає ручну перевірку системи на недоліки, з метою їх усунення [29].

Ручне тестування було проведено на локальному середовищі. Для перевірки API був використаний Postman. Postman – це інструмент, який дозволяє створювати та проводити тестування API [21].

Спочатку було перевірено сценарії авторизації:

Під час логіну з правильними даними сервер повертає статус 200 та у сховище браузера записуються access та refresh токени. Додатково інтерфейс відображає меню навігації по веб сайту. Спроба входу в акаунт з неправильними логіном чи паролем повертає статус 401 та відображає помилку про введення неправильної іноформації.

Далі було перевірено модуль товарів. Ввійшовши в систему під обліковим записом admin було успішно створено новий товар зі всіма полями. Запит на створення товару отримав статус 201 і дані одразу відображаються в таблиці. Пробуючи створити товар без назви або ж з ціною яка не задовільняє вимоги сервер одразу ж повертає статус 400. Під час редагування товару, якщо всі дані введено правильно, результатом є відображення нового значення товару в UI та статус запиту 201. Видалення товару користувачем з роллю staff повертає статус 403 Forbidden, що і є очікуваною відповіддю.

Модуль інвентарю був протестований з допомогою запитів до /api/inventory/stocks/. Спочатку в Postman було створено декілька записів для одного продукту з різними локаціями. Результати тестування показано на рис. 5.1 та рис. 5.2.

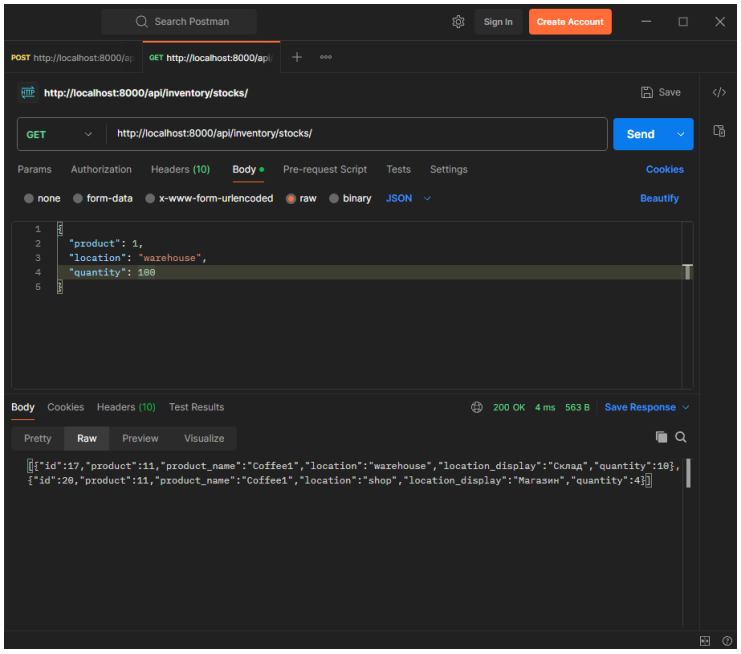


Рисунок 5.1 - Результат тестування добавляючи товар на склад магазину

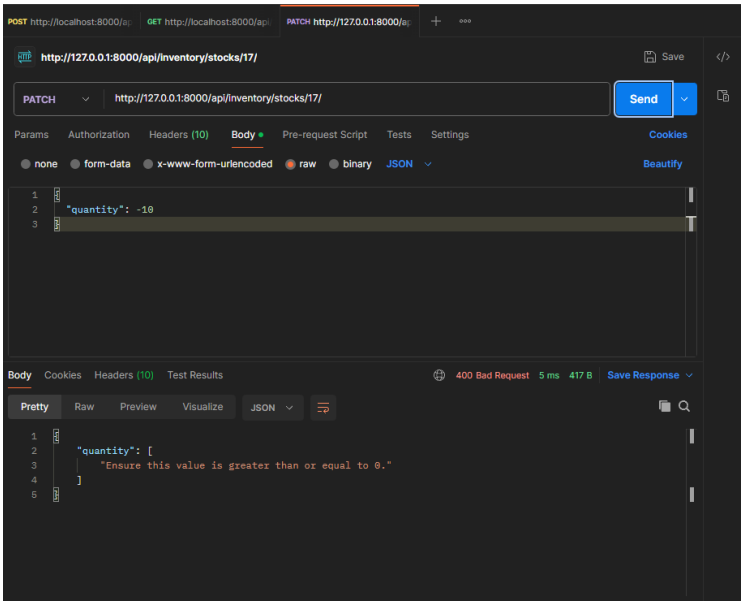


Рисунок 5.2 - Результат тестування добавляючи товар на локацію магазину

Після цього було перевірено правильність роботи редагування інформації про кількість товару через PATCH. При введенні від'ємного значення сервер повертає статус 400. Результат тестування показано на рис. 5.3.

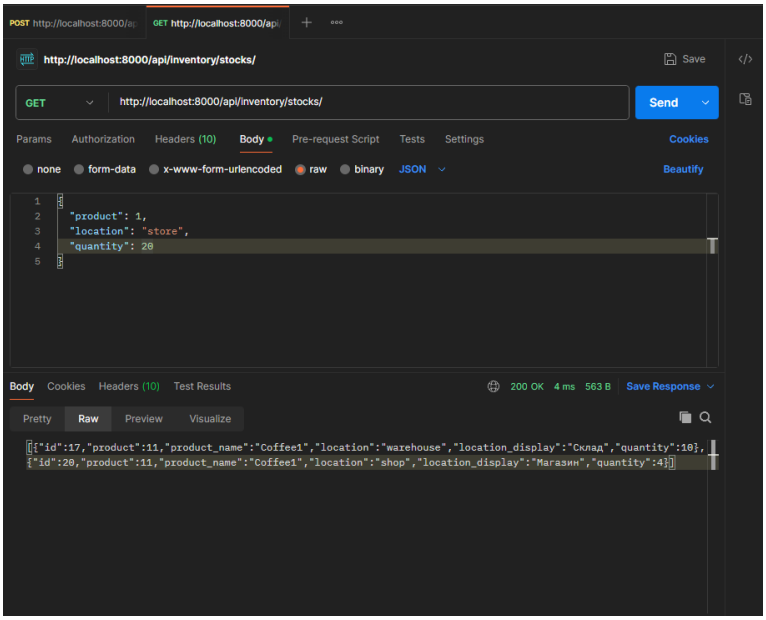


Рисунок 5.3 - Результат тестування з від'ємним значенням товару

Функція transfer була перевірена за двох сценаріїв:

1. Перенесення кількості товару, яка є меншою за наявну кількість товару. Результатом є повернення статусу 200 та оновлення UI.

2. Перенесення кількості товару, яка є більшою за наявну кількість товару. Результатом є повернення статусу 400.

Далі було протестовано модель замовлень. Спочатку було створено нове замовлення. Результатом було відображення інформації про це нове замовлення в таблиці. Також було протестовано функціональність кнопок керування статусу замовлення. Зміна статусу відправляла запит на backend системи, з допомогою PATCH та отримувала новий статус. Після доставки товару у модулі Notifications з'явилося повідомлення про успішну доставку та кількість товару на складі магазину збільшилася.

5.3 Unit тести

Unit тестування є одним з найважливіших етапів в розробці програмного забезпечення. Unit тестування – це автоматизовані тести модулів системи, які використовуються для тестування окремих модулів або ж функцій системи [30]. З метою тестування розробленої системи було використано фреймворк pytest та pytest-django.

Усі тести до системи були логічно розділені за модулями відповідно до структури системи.

Приклад виконання тестів додатку User:

Спочатку в корені додатку було створено папку tests. В самій папці було додано файли для тестування моделей, серіалізатора та функцій додатку. Загальний вигляд папки tests продемонстровано на рис 5.4.

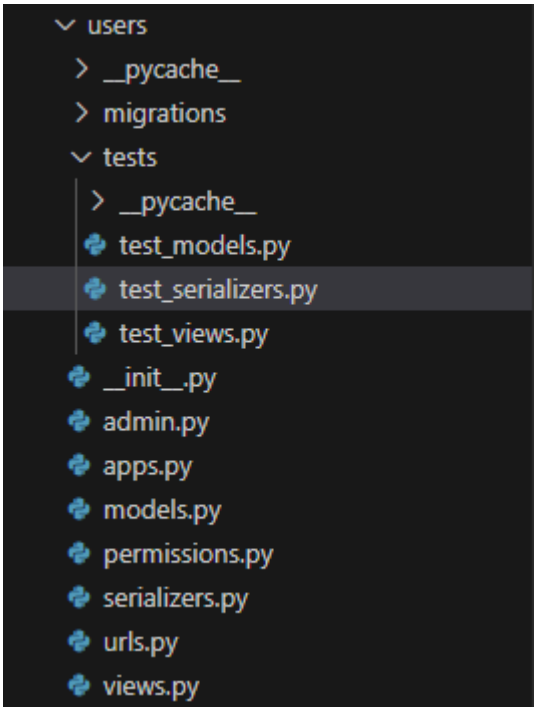


Рисунок 5.4 - Загальний вигляд файлів додатку users

Вміст файлу test_models.py показано на рис 5.5.

backend > users > tests > test_models.py > ...

```python
import pytest
from django.contrib.auth import get_user_model
from django.db.utils import IntegrityError

User = get_user_model()

def test_create_user(db):
    user = User.objects.create_user(username='u1', email='u1@gmail.com', password='pw', role='staff')
    assert user.username == 'u1'
    assert user.check_password('pw')
    assert user.role == 'staff'

def test_admin_flags(db):
    admin = User.objects.create_superuser('admin', 'a@b.c', 'pw')
    assert admin.is_staff is True
    assert admin.is_superuser is True
```

Рисунок 5.5 - Вміст файлу test_models.py додатку users

Тести серіалізатора додатку user надають можливість переконатися, що валідація та збереження даних користувача працює правильно. Вміст файлу test_serializers.py зображено на рис 5.6.

backend > users > tests > test_serializers.py > ...

```python
import pytest
from users.serializers import UserCreateSerializer

@pytest.mark.django_db
def test_user_create_serializer():
    data = {
        'username': 'u2',
        'email': 'u2@gmail.com',
        'password': 'pw',
        'role': 'admin',
        'phone': '123'
    }
    ser = UserCreateSerializer(data=data)
    assert ser.is_valid(), ser.errors
    user = ser.save()
    assert user.role == 'admin'
    assert user.check_password('pw')
```

Рисунок 5.6 - Вигляд файлу test_serializers.py додатку users

Тести API функцій перевіряють роботу прав доступу. Вміст файлу test_views.py продемонстровано на рис 5.7.

backend > users > tests > test_views.py > ...

```python
import pytest
from rest_framework import status

@pytest.mark.django_db
def test_user_list_requires_auth(api_client):
    resp = api_client.get('/api/users/users/')
    assert resp.status_code == status.HTTP_401_UNAUTHORIZED

@pytest.mark.django_db
def test_user_list_admin(auth_admin):
    resp = auth_admin.get('/api/users/users/')
    assert resp.status_code == status.HTTP_200_OK
```

Рисунок 5.7 - Вигляд файлу test_views.py додатку users

Після успішнього написання тестів, їх було виконано. Результати показано на рис. 5.8

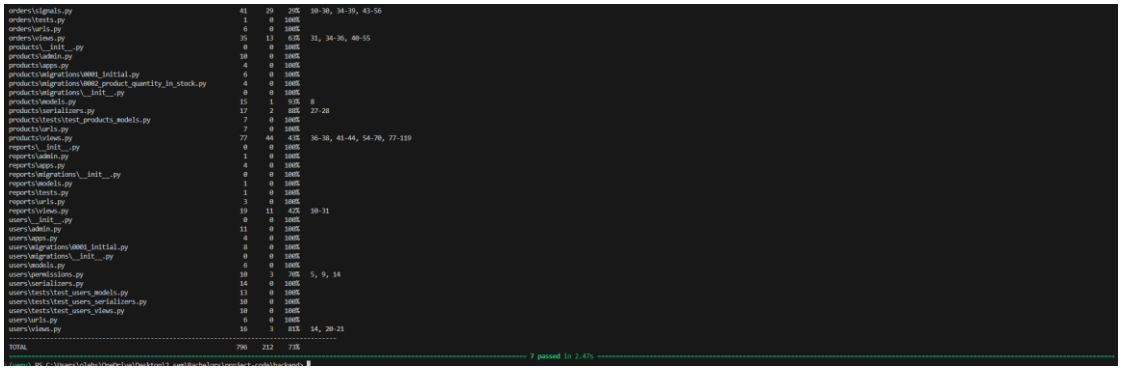


Рисунок 5.8 - Результати Unit тестів

З допомогою використання Unit тестів, було перевірено функціональність системи в Django, що допомогло переконатися, що система працює справно.

5.4 Висновки до розділу

З допомогою тестування було перевірено правильність створених функцій системи. Використання ручного тестування допомогло виявити помилки системи та дозволило їх виправити. Також було виконано та протестовано всі безпекові перевірки, такі як JWT автентифікація, CORS, CSRF та валідація вхідних даних, які пройшли всі тести успішно.

Таким чином, розроблена система відповідає поставленим функаціональним та нефункціональним вимогам та є готовою до подальшого використання.

ВИСНОВКИ

У ході виконання роботи було спроектовано, розроблено та протестовано веб-систему управління залишками та замовленнями магазину та його складу. Сама система була спроектована з використанням сучасних технологій. Для створення backend системи було використано фреймворк Django, а для користувацької частини - ReactJS. Метою проекту було створення функціональної, гнучкої та зрозумілої в використанні системи для автоматизації бізнес процесів малого в розмірі магазину. Результатом є веб додаток, що забезпечує облік товару, контроль замовлень та підтримку різних ролей доступу користувачів.

Спочатку було зроблено огляд та аналіз існуючих технологій, що надало можливісь окреслити основні функціональні можливості. Також це допомоголо сформувати бачення щодо компонентів майбутнього додатку.

Після цього було проведено аналіз та створення фунціональних та нефункціональних вимог системи. Додатково, було визначено основні ролі користувачів системи, що дозволило зрозуміти логіку доступу до функціоналу системи.

Після окреслення всіх вимог та проведення аналізу систем-аналогів, був реалізований сам додаток та його тестування. В рамках ручного тестування було перевірено всі ключові функції системи з використанням інтерфейсу користувача та консолі розробника в веб браузері. Далі було проведено unit тести для моделей, функцій та сереалізаторів Django. Було перевірено функціональність прав доступу користувачів та реакція системи на неправильні дані.

Отже, розроблена система задовільняє поставлені вимоги, є безпечною та стабільною в використанні.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Odoo. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/Odoo (Дата звернення: 03.04.2025).
2. Odoo: Open Source ERP and CRM. *Odoo:* веб-сайт. URL: https://www.odoo.com (Дата звернення: 10.03.2025).
3. Introduction_to_SAP_Business_One. *SAP:* веб-сайт. URL: https://help.sap.com/doc/2bdf221f793b43c98617b33b6805c249/10.0/en-US/Introduction_to_SAP_Business_One.pdf (Дата звернення: 12.03.2025).
4. Stay on Top of Your Inventory Management with Zoho Inventory. *Zoho inventory:* веб-сайт. URL: https://www.zoho.com/inventory/academy/inventory-management/stay-on-top-of-your-inventory-management-with-zoho-inventory.html (Дата звернення: 21.03.2025).
5. SAP Business One 10.0. *SAP*: веб-сайт. URL: https://help.sap.com/docs/SAP_BUSINESS_ONE/68a2e87fb29941b5bf959a184d9c6727/44c4c1cd7ca22e17e10000000a114a6b.html (Дата зверення: 24.03.2025).
6. Access Zoho Inventory. *Zoho Inventory:* веб-сайт. URL: https://www.zoho.com/ca/inventory/help/getting-started/sign-up.html (Дата звернення: 28.03.2025).
7. Fishbowl Integrates with Intuit QuickBooks Online. *Fishbowl:* веб-сайт. URL: https://www.fishbowlinventory.com/integrations/quickbooks-online (Дата звернення: 30.03.2025).
8. Top 10 Most-Starred Open-Source ERP and CRM on GitHub. *Medium:* веб-сайт. URL: https://medium.com/%40nocobase/top-10-most-starred-open-source-erp-and-crm-on-github-9a3d585eeb9e (Дата звернення: 30.03.2025).
9. SAP Business One. *SAP:* веб-сайт. URL: https://www.sap.com/products/erp/business-one.html (Дата звернення: 04.04.2025).

10. SAP Business One. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/SAP_Business_One (Дата звернення: 04.04.2025).

11. QuickBooks vs. ERP: Which Is Best for Your Business? *Fishbowl:* веб-сайт. URL: https://www.fishbowlinventory.com/blog/quickbooks-vs-erp-which-is-best-for-your-business (Дата звернення: 04.04.2025).

12. Meet Django. *Django*: веб-сайт. URL: https://www.djangoproject.com (Дата звернення: 20.04.2025).

13. Django REST framework. *Django REST framework:* веб-сайт. URL: https://www.django-rest-framework.org (Дата звернення: 20.04.2025).

14. React. *Wikipedia*: веб-сайт. URL: https://uk.wikipedia.org/wiki/React (Дата звернення: 21.04.2025).

15. AWS Cloud Security. *AWS:* веб-сайт. URL: https://aws.amazon.com/security (Дата звернення: 20.04.2025).

16. ABAP/4. *Wikipedia:* веб-сайт. URL: https://uk.wikipedia.org/wiki/ABAP/4 (Дата звернення: 21.04.2025).

17. Django. *Wikipedia:* веб-сайт. URL: https://uk.wikipedia.org/w/index.php?title=Django (Дата звернення: 21.04.2025).

18. REST. *Wikipedia:* веб-сайт. URL: https://uk.wikipedia.org/wiki/REST (Дата звернення: 21.04.2025).

19. ReactRouter. *React Router:* веб-сайт. URL: https://reactrouter.com (Дата звернення: 22.04.2025).

20. django-filter. *Django-filter:* веб-сайт. URL: https://django-filter.readthedocs.io (Дата звернення: 25.04.2025).

21. Postman (software). *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/Postman_(software) (Дата звернення: 25.04.2025).

22. Software as a service. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/Software_as_a_service (Дата звернення: 26.04.2025).

23. Enterprise resource planning. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/Enterprise_resource_planning (Дата звернення: 26.04.2025).

24. QuickBooks. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/QuickBooks (Дата звернення: 26.04.2025).

25. JSON. *Wikipedia:* веб-сайт. URL: https://en.wikipedia.org/wiki/JSON (Дата звернення: 26.04.2025).

26. sqlite3 – DB-API 2.0 interface for SQLite databases. *Docs Python:* веб-сайт. URL: https://docs.python.org/3/library/sqlite3.html (Дата звернення: 26.04.2025).

27. Getting Started. *Axios:* веб-сайт. URL: https://axios-http.com/docs/intro (Дата звернення: 27.04.2025).

28. Python Virtual Environment. *W3Schools:* веб-сайт. URL: https://www.w3schools.com/python/python_virtualenv.asp (Дата звернення: 28.04.2025).

29. Ручне тестування. *Wikipedia:* веб-сайт. URL: https://uk.wikipedia.org/wiki/Ручне_тестування (Дата звернення: 28.04.2025).

30. Що таке Unit тести і як їх писати. *FoxmindEd:* веб-сайт. URL: https://foxminded.ua/yunit-testy/ (Дата звернення: 28.04.2025).

ДОДАТКИ

Додаток А

Посилання на GitHub з кодом системи:

https://github.com/S0meThin/DiplomaNUNG

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи “Розробка веб-додатку для управління замовленнями та товарними залишками магазину і складу”

Обсяг пояснювальної записки: 64 аркуші

Дата закінчення роботи 10 червня 2025р.

Підпис _____