СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) Інформаційних технологій та електроніки

Кафедра Інформаційних технологій та програмування
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
(бакалавр, магістр)
спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

на тему Інформаційна система оптимізації та керування бізнес процесами малого бізнесу

Виконала: студентка групи ІПЗ-21д _____ В. П. Товкач
(підпис) (ініціали і прізвище)

Керівник _____ Д. М. Марченко
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ О. І. Захожай
(підпис) (ініціали і прізвище)

Рецензент Захожай О.І.

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) Інформаційних технологій та електроніки.
Кафедра Інформаційних технологій та програмування
(повна назва кафедри)
Освітній ступінь бакалавр
(бакалавр, магістр)
спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
спеціалізація Інженерія програмного забезпечення
(назва спеціалізації)

_____ Захожай О.І.
“___” _____ 2025 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Товкач Вікторії Петрівні
(прізвище, ім’я, по-баткові)

1. Тема роботи Інформаційна система оптимізації та керування бізнес процесами малого бізнесу_____
_____
_____
Керівник роботи Марченко Дмитро Миколайович, проф.,д.т.н._____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “27” травня 2025 року № 67/15.15-С

2.Строк подання студентом роботи 18.06.25_____
3. Вихідні дані до роботи_Об’єктом кваліфікаційної роботи є розробка та створення інформаційної системи для керування та оптимізації бізнес процесів_____
_____
_____
_____4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. Аналітична частина, з висвітленням наступних питань:
Актуальність теми розробки інформаційної системи для малого бізнесу. Особливості організації бізнес-процесів на малих підприємствах. Обґрунтування необхідності автоматизації внутрішніх операцій. Аналіз існуючих програмних рішень для обліку товарів, замовлень та клієнтів. Постановка завдання автоматизації та визначення основних функціональних і нефункціональних вимог до системи.Основна частина, в якій висвітлено: Розробку архітектури інформаційної системи. Вибір технологічного стеку, зокрема використання Python, Django, PostgreSQL. Проєктування структури бази даних. Побудову модульної системи з реалізацією управління користувачами, обліку товарів, обробки

замовлень та ведення адміністративного контролю. Реалізацію механізму контролю залишків товару. Застосування ролей користувачів та розмежування прав доступу. Впровадження логіки взаємодії між модулями. Проведення ручного функціонального тестування та апробації системи на тестових сценаріях. Усунення виявлених помилок та оцінка стабільності роботи.Висновок.Перелік використаних джерел.

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

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7.Дата видачі завдання 18.04.25

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Дослідження предметної області	18.04.25-28.04.25	Виконано
2	Пошук та аналіз існуючих рішень	29.04.25-03.05.25	Виконано
3	Аналіз функціональних та нефункціональних вимог до системи	04.05.25-10.05.25	Виконано
4	Проєктування системи та бази даних	11.05.25-17.05.25	Виконано
5	Розробка системи	18.05.25-30.05.25	Виконано
6	Апробація інформаційної системи	30.05.25-05.06.25	Виконано
7	Оформлення пояснювальної записки	07.06.25-16.06.25	Виконано
8	Підготовка та подання кваліфікаційної роботи	17.06.25-18.06.25	Виконано

Студентка _____ В.П. Товкач
(підпис) (ініціали і прізвище)

Керівник роботи _____ Д. М. Марченко
(підпис) (ініціали і прізвище)

ЗМІСТ

ВСТУП

Сучасний розвиток цифрових технологій відкриває нові можливості для вдосконалення управлінських процесів у бізнесі. Особливого значення ці процеси набувають для представників малого підприємництва, які часто працюють в умовах обмежених ресурсів і нестабільного ринкового середовища. Для забезпечення ефективності функціонування таких підприємств важливою є автоматизація рутинних операцій, зокрема — обліку товарів, керування запасами, фіксації поставок та реалізації продукції. Саме ці бізнес-процеси можуть бути оптимізовані за допомогою впровадження інформаційних систем.

У багатьох випадках малі підприємства використовують ручні методи або прості табличні інструменти, які не відповідають вимогам сучасного та динамічного ринку. Це призводить до помилок у даних, втрати часу на обробку замовлень, неефективного планування постачання та невчасного виявлення дефіциту товарів. Впровадження веб-орієнтованої інформаційної системи дозволяє автоматизувати зазначені процеси, зменшити навантаження на персонал, підвищити точність і швидкість обробки інформації, а також приймати обґрунтовані управлінські рішення.

Актуальність теми обумовлена потребою малого бізнесу в простих, гнучких та доступних програмних рішеннях, які не потребують значних фінансових витрат і водночас здатні повністю покривати базові потреби в обліку товарів і замовлень. Для виконання цієї задачі доцільним буде використання фреймворку Django, що поєднує високу швидкість розробки за рахунок зрозумілої структури з надійною архітектурою, та бази даних PostgreSQL, яка гарантує збереження і цілісність інформації, що особливо важливо для забезпечення стабільності бізнес процесів.

Метою дипломної роботи є розробка інформаційної системи для оптимізації та управління бізнес-процесами малого підприємства, зокрема обліку товарів, замовлень та контролю залишків на складі.

Для створення інформаційної системи необхідно розв’язати такі завдання:

1. Провести аналіз предметної області та виявити основні проблеми, які можуть бути усунені шляхом автоматизації
2. Проаналізувати наявні рішення на ринку та виявити перевіаги та недоліки цих систем
3. Визначити функціональні та нефункціональні вимоги до майбутньої системи
4. Розробити архітектуру інформаційної системи та спроєктувати структуру бази даних
5. Реалізувати основні функціональні модулі системи
6. Здійснити тестування додатка та оцінити ефективність системи

Об’єктом дослідження є процеси ведення обліку товарів і замовлень у малому торговому бізнесі.

Предметом дослідження є методи та засоби розробки інформаційних систем для автоматизації бізнес-процесів із використанням сучасних веб-технологій.

Методи дослідження, що використовуються в роботі, включають аналіз предметної області, моделювання структури бази даних, об'єктно-орієнтоване програмування, розробку вебзастосунків із використанням Django, а також тестування функціональності та ефективності інформаційної системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Успішне проєктування та реалізація програмного забезпечення неможливі без глибокого аналізу предметної області, в межах якої функціонуватиме інформаційна система. Ретельне вивчення об'єкта автоматизації дозволяє виявити реальні потреби користувачів, визначити проблеми, які необхідно вирішити, та обґрунтувати доцільність упровадження того чи іншого рішення. Саме на цьому етапі закладається основа логіки майбутньої системи, її функціонального наповнення та взаємозв’язку між компонентами[1].

В умовах малого бізнесу особливо актуальним є створення інформаційної системи, яка дозволяє оптимізувати роботу з товарами, замовленнями, залишками на складі, а також забезпечити чіткий облік усіх дій працівників. Водночас система повинна бути простою у використанні, доступною для впровадження та адаптованою до змін, що відбуваються в бізнес-процесах.

1.1 Характеристика бізнес-процесів малого бізнесу

Малий бізнес є важливою складовою економіки країни. Його представники забезпечують значну частину робочих місць, гнучко реагують на потреби ринку та сприяють розвитку локальних економічних систем. Водночас підприємці малого бізнесу часто стикаються з такими труднощами, як обмежені ресурси, брак персоналу, нестабільність постачання та потреба в оперативному прийнятті управлінських рішень. У таких умовах особливої ваги набуває ефективна організація внутрішніх бізнес-процесів.

Залежно від сфери діяльності бізнес-процеси можуть мати різну специфіку, проте у сфері торгівлі вони зазвичай мають спільні риси[1].

Ключові процеси, характерні для торгівельного малого бізнесу:

- **Облік товарів**

Включає реєстрацію нових позицій, фіксацію категорій, цін, залишків на складі. У багатьох випадках опрацьовується вручну або за допомогою електронних таблиць, що створює ризик втрати чи плутанини даних.

- **Постачання**

Процес надходження товарів на склад. Передбачає фіксацію отримання продукції, кількості та оновлення інформації про запаси.

- **Продажі**

Реалізація товарів клієнтам. Мають супроводжуватись автоматичним зменшенням залишків, збереженням даних про дату продажу, кількість, відповідального працівника.

- **Контроль запасів**

Необхідний для уникнення дефіциту або надлишку товарів. Передбачає моніторинг мінімально допустимого рівня залишків і формування відповідних звітів або сповіщень.

- **Керування персоналом і ролями**

Часто в малому бізнесі кілька осіб поєднують кілька функцій, тому важливо розмежувати рівні доступу до даних: наприклад, адміністратор бачить всі дії та операції, а менеджер - лише замовлення й облік товарів.

- **Формування звітності**

Підприємцю необхідна регулярна звітність для прийняття керівницьких рішень: найпопулярніші товари, обсяги продажу за період, залишки товару.

Усі зазначені процеси за відсутності автоматизації вимагають значних затрат часу, можуть супроводжуватись помилками, призводити до дублювання або втрати інформації. Це негативно впливає на якість обслуговування клієнтів, ефективність закупівель та фінансові показники[2].

Таким чином, актуальним завданням для малого бізнесу є впровадження інформаційної системи, яка дозволить об’єднати ключові бізнес-процеси в єдиному середовищі, забезпечити прозорість, контроль та оптимізацію щоденної діяльності підприємства.

1.2 Задачі автоматизації

Процеси, що виконуються у межах малого торгового бізнесу, зазвичай охоплюють широкий спектр операцій — від закупівлі товарів до обробки замовлень і ведення обліку залишків. У разі ручного виконання цих дій зростає

ймовірність помилок, неефективного використання ресурсів та втрати оперативного контролю. Особливо це відчутно в умовах зростання обсягу операцій або у випадках, коли підприємство працює з працівниками[2].

Автоматизація таких процесів дозволяє значно зменшити навантаження на персонал, покращити точність обліку, пришвидшити обробку даних, а також створює основу для масштабування бізнесу. У межах автоматизації є комплекс задач, що охоплюють товарний облік, операції з замовленнями, контроль запасів і розмежування прав доступу користувачів.

Інформаційна система повинна вирішувати такі задачі:

1. Автоматизація обліку товарів

Створення та ведення реєстру товарів із зазначенням назви, категорії, ціни та кількості на складі.

2. Управління поставками та продажами

Облік операцій двох типів — вхідних (надходження товару) та вихідних (відвантаження товару).

3. Автоматичне оновлення залишків

При створенні замовлення кількість товару повинна змінюватися відповідно до типу операції.

4. Відстеження історії замовлень

Збереження інформації про дату, користувача, позиції замовлення, тип операції.

5. Реалізація ролей користувачів

Адміністратор повинен мати повний доступ до всіх даних, менеджер - лише до обмеженого функціоналу (створення замовлень без доступу до керування користувачами).

6. Інтеграція з адміністративною панеллю

Можливість керування всіма сутностями системи через зручний інтерфейс адміністратора без потреби створювати додатковий фронтенд.

7. Безпечна авторизація та автентифікація користувачів

Обмеження доступу до системи лише для зареєстрованих осіб з підтвердженим паролем.

Завдання автоматизації полягає не лише у цифровізації існуючих процесів, а й у їх оптимізації: спрощенні взаємодії з даними, зменшенні кількості дій, необхідних для виконання типових операцій, та формуванні надійної основи для аналізу бізнес процесів.

Розв’язання цих задач сприятиме підвищенню продуктивності, зменшенню витрат і помилок, а також дозволить власникам бізнесу зосередитися на стратегічних аспектах діяльності, передавши рутинні облікові функції системі.

1.3 Аналіз існуючих рішень

На сучасному ринку програмного забезпечення представлено велику кількість систем для автоматизації бізнес-процесів. Серед них: універсальні ERP-системи, хмарні сервіси, локальні програми та спеціалізовані інструменти для обліку товарів[2]. Проте більшість із них орієнтовані на середній або великий бізнес і мають ряд обмежень при застосуванні в малому підприємництві.

1.3.1 Універсальні ERP-системи

До них належать такі популярні продукти, як: SAP Business One, BAS Малий бізнес, Odoo, Zoho Inventory. Вони мають професійний функціонал, підтримку складних логістичних схем, фінансової звітності, CRM, HR.

Недоліки для малого бізнесу:

1. Висока вартість ліцензій або підписки
2. Складність впровадження та налаштування
3. Потреба в навчанні персоналу
4. Надмірний функціонал, що не використовується повністю
5. Потреба у зовнішньому супроводі та обслуговуванні

1.3.2 Хмарні сервіси та онлайн-платформи

Прикладами є SalesDrive, myWarehouse, KeepinCRM, Worksection, TorgSoft Web. Вони мають простіший інтерфейс, доступ з будь-якого пристрою, та іноді безкоштовні тарифи.

Недоліки для малого бізнесу:

1. Зберігання даних на сторонніх серверах що веде до ризиків безпеки
2. Обмежена кастомізація
3. Залежність від підключення до інтернету
4. Наявність платного функціоналу вже після базової активації

1.3.3 Локальні рішення або облік у таблицях

Найбільш поширений підхід серед малого бізнесу. Дані ведуться вручну у вигляді табличок, часто без резервного копіювання, доступу кількох користувачів чи валідації.

Недоліки для малого бізнесу:

1. Відсутність системної логіки
2. Висока ймовірність людських помилок
3. Складність у масштабуванні
4. Неможливість налаштувати автоматичний облік залишків

Висновок

Попри наявність значної кількості готових рішень, жодне з них не задовольняє повністю потреби малого підприємства, що шукає:

- Безкоштовне або умовно безкоштовне рішення
- Можливість локального зберігання даних
- Простоту інтерфейсу
- Повну кастомізацію під бізнес-процеси
- Доступність вихідного коду для подальшого розвитку

У зв’язку з цим доцільною є розробка власної інформаційної системи, адаптованої під конкретні задачі малого бізнесу, реалізованої з використанням

відкритих технологій, таких як Django, PostgreSQL та розгорнутої локально або на недорогому сервері.

1.4 Вимоги до інформаційної системи

На основі аналізу предметної області, вивчення типових бізнес-процесів малого підприємства та врахування недоліків існуючих рішень сформульовано функціональні та нефункціональні вимоги до розроблюваної інформаційної системи. Ці вимоги визначають основні очікування щодо структури, логіки, інтерфейсу та поведінки системи в умовах реального використання.

1.4.1 Функціональні вимоги

Функціональні вимоги — це вимоги до працездатності системи, а саме те, що вона повинна робити. Для реалізації інформаційної системи кваліфікаційної роботи, були визначені такі функціональні вимоги:

- **Облік товарів**

1. Додавання, редагування та видалення товарних позицій
2. Фіксація назви, категорії, кількості та ціни кожного товару
3. Відображення залишків товару на складі

- **Керування категоріями**

1. Можливість створення та редагування категорій товарів
2. Групування товарів за категоріями для зручності пошуку та аналізу

- **Обробка замовлень**

1. Створення замовлень двох типів: вхідне (надходження товару), вихідне (реалізація товару)
2. Додавання до замовлення однієї або кількох позицій
3. Автоматичне оновлення кількості товару на складі відповідно до типу замовлення

- **Облік дій користувачів**

1. Збереження інформації про автора кожного замовлення
2. Фіксація дати та часу створення замовлень

- **Рольовий доступ**

1. Розмежування прав доступу між адміністраторами, менеджерами та звичайними користувачами

2. Обмеження доступу до конфіденційної інформації

- **Зручна адміністративна панель**

1. Керування усіма об’єктами через інтерфейс Django Admin

2. Фільтрація, пошук, сортування об'єктів

3. Інтуїтивне управління замовленнями та товарами

1.4.2 Нефункціональні вимоги

Нефункціональні вимоги — це як саме система повинна виконувати свої функції. Вони описують якість, обмеження або властивості системи. Для реалізації інформаційної системи, були визначені такі нефункціональні вимоги:

- **Простота використання**

1. Інтерфейс має бути інтуітивно зрозумілим

2. Інструкція або шпаргалка для першого налаштування та роботи

- **Безпека**

1. Доступ до системи лише після авторизації

2. Зберігання паролів у зашифрованому вигляді

3. Захист від несанкціонованого доступу

- **Сумісність та масштабованість**

1. Можливість розгортання системи як локально, так і на хостингу

2. Збереження структурованості коду для подальшого розширення функціоналу

- **Продуктивність**

1. Швидка обробка операцій додавання, оновлення та фільтрації даних

2. Оптимізовані запити до бази даних

- **Технічні аспекти**

1. Використання безкоштовного і відкритого програмного забезпечення, а семе: Python Django, PostgreSQL

2. Зберігання вихідного коду в системі контролю версій (Git)

Таким чином, сформульовані вимоги створюють чітке технічне завдання для проєктування інформаційної системи. Їх дотримання дозволить реалізувати ефективний, простий та надійний інструмент для автоматизації бізнес-процесів малого підприємства.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

На основі аналізу предметної області, виявлених потреб користувачів та сформульованих функціональних і нефункціональних вимог розпочато проєктування інформаційної системи.

Проєктування охоплює як високорівневу архітектуру системи, так і низькорівневі компоненти: моделі бази даних, модулі бізнес-логіки, механізми авторизації та структуру адміністративної частини. Важливо, щоб побудова системи була модульною, розширюваною, а також відповідала принципам простоти обслуговування та незалежності компонентів[2].

У результаті проєктування створено основу для реалізації системи, яка дозволяє забезпечити надійну, зручну й масштабовану програмну інфраструктуру для малого бізнесу.

2.1 Архітектура додатку

Архітектура програмної системи — це логічна структура, яка визначає основні компоненти, їхні функції, способи взаємодії та розмежування обов’язків між ними. Від правильного архітектурного підходу залежить стабільність, масштабованість, зручність супроводу та можливість подальшого розширення системи[2].

Для розробки інформаційної системи малого бізнесу було обрано трирівневу клієнт-серверну архітектуру, яка включає:

- **Рівень представлення**

Інтерфейс користувача, представлений адміністративною панеллю Django[3]. Через неї здійснюється взаємодія з даними: перегляд, створення, редагування, фільтрація, оскільки адміністративна панель повністю покриває потреби управління системою.

- **Рівень логіки**

Містить основну бізнес-логіку: обробку замовлень, зміну залишків, перевірку прав доступу, фіксацію дій користувача. Цей рівень реалізовано за допомогою Django: моделі, представлення, сигнали, форми.

- **Рівень даних**

Забезпечує зберігання інформації у базі даних PostgreSQL[4]. Усі об’єкти системи (користувачі, товари, замовлення, позиції замовлень) зберігаються у структурованому вигляді з використанням реляційної моделі.

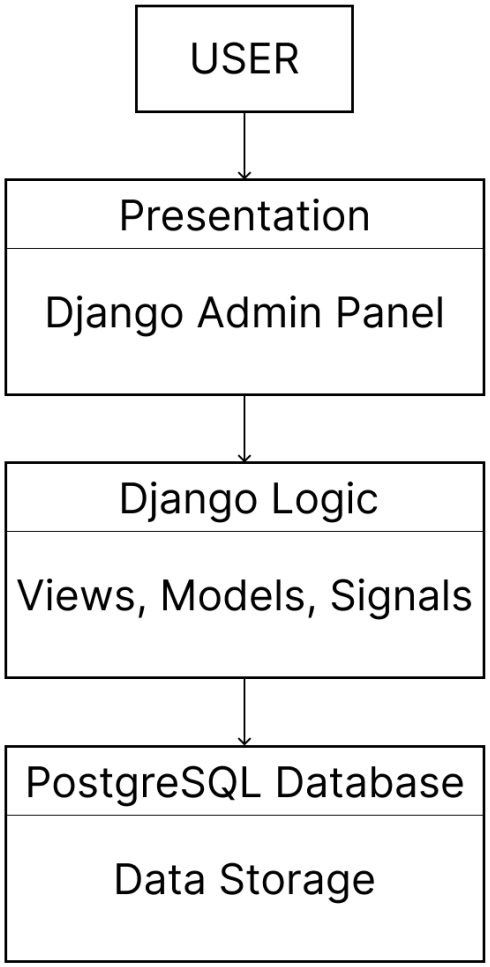

Рис. 2.1 — Схема трирівневої клієнт - серверної архітектури

Переваги трирівневої клієнт-серверної архітектури:

1. Модульність

Чітке розділення на логічні блоки (users, inventory, orders), що дозволяє розробляти та змінювати компоненти незалежно один від одного.

2. Простота

Завдяки використанню Django значно скорочується час розробки.

3. Масштабованість

При необхідності систему можна доповнити API на базі Django REST Framework, фронтендом, розширити ролі або змінити логіку обробки даних.

4. Безпека

Django забезпечує захист від базових загроз, а саме SQL-ін’єкцій, CSRF, XSS, а PostgreSQL забезпечує надійне зберігання даних.

Таким чином, обрана архітектура є оптимальною для задач малого бізнесу: вона поєднує в собі простоту, функціональність і гнучкість для масштабування зі зростанням кількості бінес-процесів.

2.2 Технології та середовище розробки

Вибір технологій для розробки інформаційної системи є одним із ключових рішень, яке впливає на ефективність створення, підтримки та розвитку програмного продукту. При визначенні технологічного стеку враховувалися такі фактори: відкритість, гнучкість, наявність спільноти підтримки, можливість швидкої розробки MVP, зручність інтеграції компонентів, сумісність із сучасними системами.

Використані технології:

- **Python**

Мова програмування Python є основною для фреймворку Django і використовується як для опису логіки застосунку, так і для роботи з базою даних, тестування, підключення пакетів.

- **Фреймворк Django**

Django є професійним фреймворком для розробки веб-застосунків мовою програмування Python[3]. Він був обраний як основа проєкту завдяки таким перевагам:

1. Вбудована ORM для взаємодії з базами даних
2. Вбудована базова система шаблонів, маршрутизації, форм і автентифікації
3. Зручна адмін-панель для управління об'єктами
4. Висока швидкість розробки та надійна документація.

- **База даних PostgreSQL**

В якості системи управління базами даних було обрано PostgreSQL, реляційну СУБД з відкритим кодом[4].

Переваги PostgreSQL:

1. Стабільність та масштабованість
2. Підтримка складних запитів, транзакцій, індексів
3. Сумісність з Django через драйвер psycopg2

- **Адмін-панель pgAdmin 4**

Графічна утиліта для роботи з базою даних PostgreSQL для перегляду структури таблиць, виконання SQL-запитів, аналізу схеми даних[5].

Переваги вибору стеку Django + PostgreSQL:

1. Відкритий вихідний код і безкоштовне ліцензування
2. Підтримка CRUD-операцій без додаткових модулів
3. Можливість гнучкого розширення функціональності
4. Швидкий запуск MVP для тестування та демонстрації

- **Інструменти розробки та керування проєктом**

Для розробки програмної частини інформаційної системи було використано сучасні інструменти, які забезпечують ефективну організацію проєкту та зручність у процесі програмування. Основним середовищем розробки слугувала PyCharm — професійна IDE, яка надає зручні засоби для написання коду, включно з підсвіткою синтаксису, автодоповненням, навігацією по проєкту та інтеграцією з системою контролю версій Git[6]. Для керування версіями, збереження історії змін та зручної роботи використовувалась система Git, яка дозволила фіксувати кожен етап змін і за потреби повертатися до попередніх версій коду. Для створення ізольованого середовища з необхідними бібліотеками та залежностями застосовувався інструмент Virtualenv, що забезпечив стабільність і уникнення конфліктів між різними пакетами Python, які використовувались у межах проєкту.

Таким чином, обрані технології забезпечують ефективну розробку системи, дозволяють легко адаптувати її під зміну вимог, а також підтримують масштабування та інтеграцію з іншими сервісами[7].

2.3 Проєктування бази даних

База даних є центральним компонентом будь-якої інформаційної системи, оскільки саме вона забезпечує надійне зберігання, структурування та швидкий доступ до інформації. Проєктування бази даних виконується на основі аналізу предметної області, сформульованих вимог і передбачуваних сценаріїв використання системи.

Метою проєктування є створення логічної структури взаємопов'язаних таблиць, яка дозволить ефективно зберігати дані про товари, категорії, користувачів, замовлення та їхні позиції, а також забезпечить цілісність і узгодженість інформації.

Основні сутності та зв'язки:

1. **Користувачі (users_customuser)**
    - Зберігають дані про зареєстрованих працівників системи (адміністратор, менеджер)
    - Роль користувача визначає його повноваження
2. **Категорії товарів (inventory_category)**
    - Дозволяють логічно групувати товари
    - Спрощують фільтрацію та структурування даних
3. **Товари (inventory_product)**
    - Основна одиниця обліку
    - Пов'язана з категорією
    - Має назву, ціну, залишок на складі
4. **Замовлення (orders_order)**
    - Представляють факт операції (поставка або продаж)
    - Містять тип (in — надходження, out — відвантаження), дату створення, автора
5. **Позиції замовлення (orders_orderitem)**
    - дозволяють деталізувати замовлення (кілька товарів у межах одного замовлення)
    - кожна позиція вказує на конкретний товар і кількість

Зв’язки між таблицями:

- Один користувач може створювати багато замовлень (OneToMany)
- Один товар належить до однієї категорії, але категорія може включати багато товарів (ForeignKey)
- Одне замовлення може мати багато позицій (OrderItem), кожна з яких пов’язана з одним товаром

Нормалізація структури:

- Структура відповідає третій нормальній формі
- Усі поля містять атомарні значення
- Відсутнє дублювання даних
- Чітка ієрархія залежностей

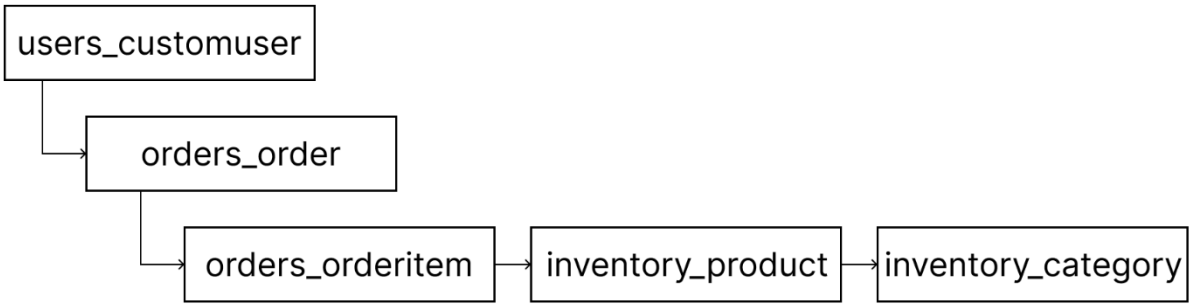

Рис. 2.2 — Зв’язки сутностей

Кожна таблиця реалізована як модель (class ModelName(models.Model)), а зв’язки — через ForeignKey. Django ORM дозволяє працювати з даними як з Python-об’єктами, не використовуючи безпосередньо SQL, що значно спрощує розробку й супровід.

Таким чином, спроєктована база даних є логічно обґрунтованою, структурованою та зручною для масштабування, що створює надійну основу для ефективного функціонування всієї системи.

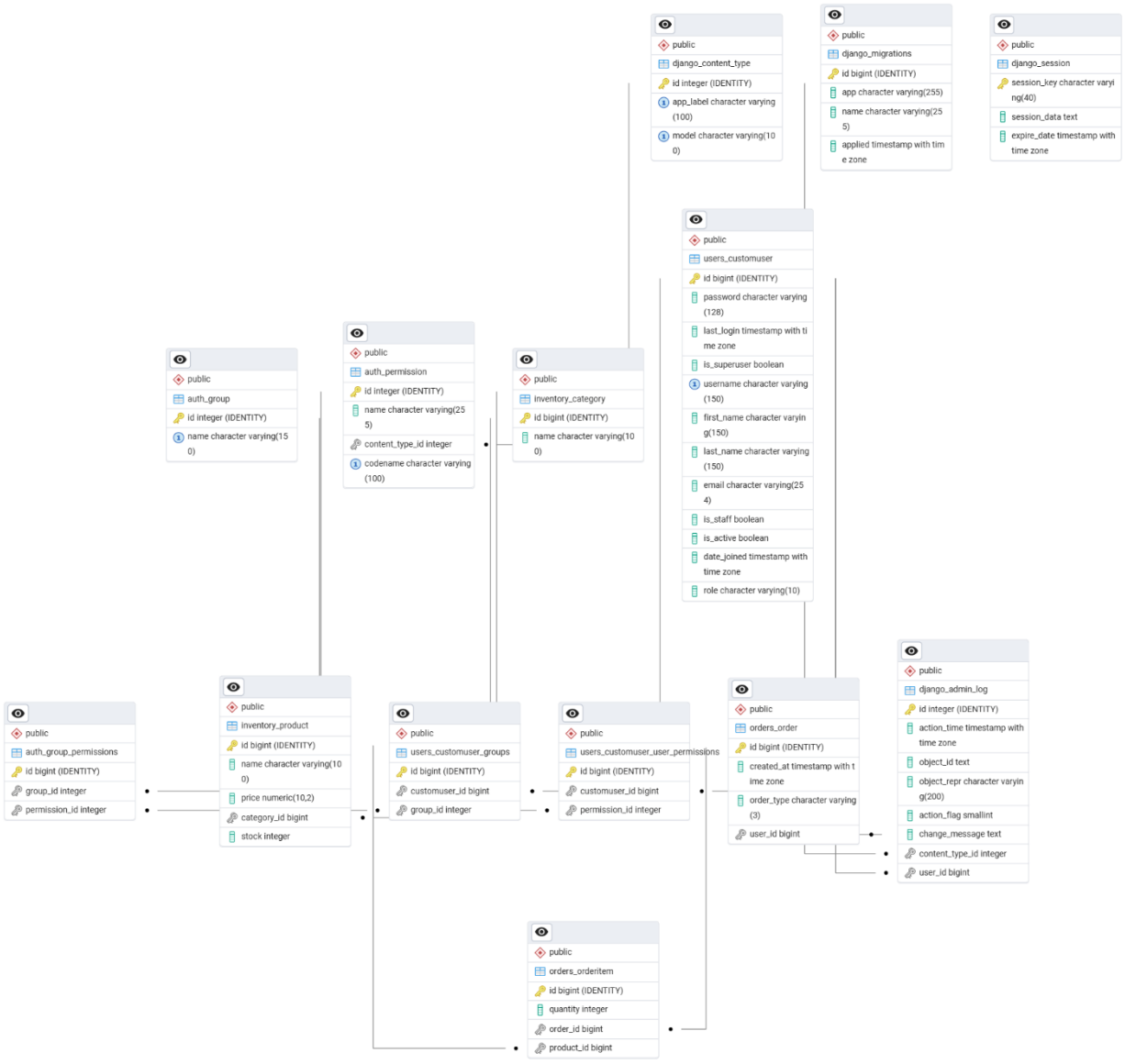

Рис. 2.3 — Схема бази даних інформаційної системи

2.4 Структура модулів

У рамках побудови інформаційної системи для автоматизації бізнес-процесів малого підприємства було реалізовано модульну структуру, що дозволяє розділити функціональність системи за логічними компонентами. Такий підхід відповідає принципам модульності та розділення відповідальності, що забезпечує кращу організацію коду, зручність супроводу, масштабування і повторне використання логіки.

Для реалізації системи у Django[8] було створено три основних додатки: users, inventory, orders.

1. Модуль users

Модуль відповідає за створення, зберігання та обробку даних користувачів. Основою є кастомна модель користувача CustomUser, яка розширює стандартну модель Django та містить додаткове поле role, що визначає рівень доступу.

Ключові елементи:

1. Кастомна модель CustomUser
2. Система автентифікації
3. Механізми авторизації через ролі (адміністратор, менеджер)
4. Інтеграція з Django Admin для керування користувачами

2. Модуль inventory

Модуль реалізує функціональність з управління складським обліком. Тут зберігається інформація про товари, їх кількість, категорії та ціни.

Моделі модуля:

- Category — категорія товарів
- Product — товар із вказанням назви, категорії, залишку, ціни

Можливості:

- Створення, редагування та видалення товарів і категорій
- Фільтрація за категоріями
- Оновлення залишків через взаємодію з модулем замовлень

3. Модуль orders

Цей модуль забезпечує створення, збереження та обробку замовлень на товари. Замовлення бувають двох типів: вхідне (поставка) та вихідне (продаж, списання).

Основні моделі:

- Order — загальні дані про замовлення (тип, дата, користувач)
- OrderItem — позиція замовлення (зв’язок з товаром і кількістю)

Функції модуля:

- Автоматичне оновлення залишків при створенні або видаленні замовлення
- Зберігання історії замовлень і зв’язок з автором

- Інтеграція з адмін-панеллю для керування замовленнями

Взаємозв'язки модулів:

- Модуль orders безпосередньо взаємодіє з inventory, використовуючи дані про товари та змінюючи залишки
- Orders і users пов'язані через поле user у моделі Order
- Кожен модуль має власну логіку, але працює у тісній взаємодії з іншими, що забезпечує цілісність даних і бізнес-процесів

Завдяки модульній побудові систему легко підтримувати й розширювати. За потреби можна додати нові функціональні блоки без зміни базової структури[8].

2.5 Механізми автентифікації та авторизації

Забезпечення контролю доступу до системи є критично важливою частиною будь-якої інформаційної системи. У межах розробки інформаційної системи для малого бізнесу реалізовано механізми автентифікації (перевірка особи користувача) та авторизації (перевірка прав доступу до ресурсів залежно від ролі) [9].

Автентифікація користувачів

У проєкті використовується стандартний механізм автентифікації Django, що включає:

- Вхід за логіном (username) та паролем
- Захист паролів за допомогою хешування (алгоритм PBKDF2)
- Можливість створення суперкористувачів (superuser) для адміністративного керування

Реалізовано кастомну модель користувача CustomUser, яка базується на AbstractUser і дозволяє розширити стандартні поля.

```
class CustomUser(AbstractUser):
    ROLE_CHOICES = (
        ('admin', 'Адміністратор'),
        ('manager', 'Менеджер'),
    )
    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default='manager')
```

Рис. 2.4 — Кастомна модель користувача

Авторизація та контроль доступу

Авторизація реалізована через перевірку ролі користувача. Залежно від значення поля role, користувач має доступ до різного функціоналу:

Роль	Можливості
Адміністратор	Повний доступ до всіх моделей, включаючи керування користувачами, товарами, замовленнями
Менеджер	Обмежений доступ: перегляд і створення замовлень, перегляд товарів

Таб. 2.1 — Ролі та можливості користувачів

Технічні засоби реалізації авторизації:

1. **Django Admin:**
   - Використання груп і дозволів (permissions) для обмеження дій
   - Вказання readonly_fields або виключення моделей для менеджерів
2. **View-рівень:**

Використання декораторів @login_required, @user_passes_test, або кастомних перевірок:

```
def is_admin(user):
    return user.role == 'admin'
```

Рис. 2.5 — Кастомна перевірка користувача

3. **Template-рівень:**

Відображення інтерфейсних елементів відповідно до ролі користувача:

```
{% if user.role == 'admin' %}
    <a href="/admin/">Панель адміністратора</a>
{% endif %}
```

Рис. 2.6 — Відображення інтерфейсу в залежності від ролі

Безпека та захист даних:

- Усі запити до адмін-панелі вимагають авторизації
- Паролі не зберігаються у відкритому вигляді
- Доступ до критичних дій мають лише користувачі з роллю admin
- Автоматичний від логів дій у django_admin_log

Таким чином, у системі реалізовано гнучку схему контролю доступу, яка дозволяє обмежити можливості користувачів відповідно до їхньої ролі, забезпечити захист даних і знизити ризик несанкціонованих змін[9].

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

На цьому етапі було здійснено реалізацію розробленої інформаційної системи відповідно до архітектури та вимог, визначених у попередніх розділах. Реалізація охоплює створення структур бази даних, програмної логіки, взаємодії між модулями та інтеграцію з адміністративною панеллю для забезпечення зручного управління всіма об'єктами системи[10].

Розробка проводилася із використанням фреймворку Django, який дозволив організувати код у вигляді окремих додатків, а також забезпечити ефективне розмежування бізнес-логіки, структури даних і представлення. Основні функціональні компоненти системи були реалізовані у трьох логічних блоках: керування товарами, облік замовлень та управління користувачами.

Окрім основної функціональності, було реалізовано механізм автоматичного оновлення залишків на складі після виконання операцій із замовленнями. Також налаштовано адміністративну панель для керування даними без потреби створення окремого користувацького інтерфейсу.

3.1 Реалізація модуля управління товарами

Модуль керування товарами є центральним елементом інформаційної системи, що забезпечує ведення складського обліку. Він дозволяє зберігати інформацію про кожну товарну позицію, її категорію, ціну та залишок на складі. Для реалізації цього модуля було створено окремий додаток Django[11] під назвою inventory.

Моделі модуля

У модулі реалізовано дві основні моделі, Product та Category

```python
from django.db import models

class Category(models.Model):  3 usages
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Product(models.Model):  4 usages
    name = models.CharField(max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    stock = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=10, decimal_places=2)

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

Рис. 3.1 — Основні моделі модуля

Модель Product відповідає за характеристики товару, а саме:

- name — назва товару
- category — зв'язок із таблицею категорій (ForeignKey)
- stock — залишок товару на складі
- price — вартість одиниці товару

Модель Category дозволяє згрупувати товари за логічними ознаками, такими як тип, функціональність, група.

Реєстрація в адміністративній панелі

Для забезпечення зручного керування даними через Django Admin, обидві моделі були зареєстровані у admin.py:

```python
from django.contrib import admin
from .models import Product, Category

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name',)
    search_fields = ('name',)

@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'category', 'stock', 'price')
    search_fields = ('name',)
    list_filter = ('category',)
```

Рис. 3.2 — Додавання товарів в адмін панелі

Адміністративна панель дозволяє швидко:

- Переглядати список товарів із можливістю фільтрації
- Змінювати ціни та кількість товару
- Додавати нові позиції й категорії
- Сортувати товари за категоріями чи назвою

Логіка взаємодії

Модуль inventory тісно інтегрований з модулем замовлень (orders): при створенні замовлення автоматично змінюється значення stock у моделі Product. Таким чином, система гарантує, що кількість товару завжди є актуальною[12].

Реалізація модуля inventory забезпечує повноцінне управління товарною базою підприємства, дозволяє вести точний облік та створює основу для подальшої аналітики запасів.

3.2. Реалізація модуля керування замовленнями

Модуль керування замовленнями реалізовано в окремому додатку Django під назвою orders. Його основне призначення — реєстрація фактів надходження або вибуття товару зі складу, а також автоматичне оновлення залишків[12]. Саме цей модуль поєднує дані з товарного обліку (модуль inventory) та дані про користувачів (модуль users).

```python
class Order(models.Model):

    def __str__(self):
        return f"{self.get_order_type_display()} - #{self.id}"


class OrderItem(models.Model):  2 usages
    order = models.ForeignKey(Order, on_delete=models.CASCADE, related_name='items')
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField()

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

    def save(self, *args, **kwargs):
        order_type = self.order.order_type

        product = self.product

        if self._state.adding:
            if order_type == 'out':
                if product.stock < self.quantity:
                    raise ValidationError(f"Insufficient accessory in stock for {product.name} (there are: {product.stock}, need: {self.quantity})")
                product.stock -= self.quantity
            elif order_type == 'in':
                product.stock += self.quantity

            product.save()

        super().save(*args, **kwargs)
```

Рис.3.3 — Модуль керування замовленнями

Опис функціоналу:

1. **Order:**
   - Тип замовлення (in — надходження, out — реалізація)
   - Дата створення
   - Зв’язок з автором замовлення
2. **OrderItem:**
   - Зв’язок із замовленням (Order)
   - Зв’язок із товаром (Product)
   - Кількість одиниць товару в замовленні

Автоматичне оновлення залишків

Після створення замовлення в системі має відбуватись автоматичне оновлення поля stock у моделі Product. Для цього було реалізовано обробку сигналів:

```python
from django.db.models.signals import post_save, post_delete
from django.dispatch import receiver



@receiver(post_save, sender=Order)
def update_stock_on_save(sender, instance, created, **kwargs):
    if created:
        for item in instance.items.all():
            if instance.order_type == 'in':
                item.product.stock += item.quantity
            elif instance.order_type == 'out':
                item.product.stock -= item.quantity
            item.product.save()

@receiver(post_delete, sender=Order)
def restore_stock_on_delete(sender, instance, **kwargs):
    for item in instance.items.all():
        if instance.order_type == 'in':
            item.product.stock -= item.quantity
        elif instance.order_type == 'out':
            item.product.stock += item.quantity
        item.product.save()
```

Рис. 3.4 — Оновлення залишків товарів

Це дозволяє:

1. Автоматично збільшувати або зменшувати залишки при додаванні замовлення
2. Відновлювати кількість при видаленні замовлення

```python
from django.contrib import admin
from .models import Order, OrderItem

class OrderItemInline(admin.TabularInline):  1 usage
    model = OrderItem
    extra = 1

@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = ('id', 'order_type', 'user', 'created_at')
    list_filter = ('order_type', 'created_at')
    inlines = [OrderItemInline]
```

Рис. 3.5 — Оновлення залишків товарів

Взаємодія з іншими модулями

Модуль замовлень тісно пов'язаний із inventory: при кожному замовленні модифікується відповідний запис у Product. Дані про користувача зберігаються через зовнішній ключ на CustomUser.

Таким чином, модуль orders реалізує повноцінну логіку обробки замовлень і автоматизує зміну залишків товарів, забезпечуючи актуальність даних і простоту в роботі менеджерів і адміністраторів[13].

3.3. Реалізація модуля користувачів

Модуль керування користувачами реалізовано у вигляді окремого Django-додатку users, який забезпечує автентифікацію, авторизацію, збереження інформації про ролі та взаємодію користувачів із замовленнями[13]. У межах проєкту було використано кастомну модель користувача, що дозволяє розширити функціональність базової моделі Django.

Кастомна модель користувача

```python
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):  5 usages
    ROLE_CHOICES = (
        ('admin', 'Admin'),
        ('manager', 'Manager'),
    )
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default='manager')

    def __str__(self):
        return f"{self.username} ({self.role})"
```

Рис. 3.6 — Кастомна модель користувача

Особливості моделі:

- Спадкування від AbstractUser дозволяє використовувати всі базові можливості Django (логін, пароль, email, staff/active/superuser)
- Поле role визначає рівень доступу користувача до системи
- Можливість фільтрації дій по користувачу, наприклад, для перегляду історії замовлень, які створив певний менеджер

Реєстрація в адмін-панелі

```python
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import CustomUser

@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
    model = CustomUser
    list_display = ("username", "email", "is_staff", "is_active")
    list_filter = ("is_staff", "is_active")
    search_fields = ("username", "email")
    ordering = ("username",)
```

Рис. 3.7 — Реєстрація в адмін панелі

Функціональні можливості модуля:

- Створення користувачів через адмін-панель
- Розмежування прав на основі ролей
- Фільтрація за ролями (admin, manager)
- Зберігання інформації про те, хто створив кожне замовлення (через ForeignKey у Order)

Таким чином, модуль users забезпечує гнучке керування персоналом системи, дозволяє реалізувати базову безпеку та контроль доступу, а також створює основу для персоніфікованої роботи користувачів у рамках бізнес-процесів.

3.4. Адміністративна панель Django

Однією з головних переваг використання фреймворку Django є наявність вбудованої адміністративної панелі, яка дозволяє ефективно керувати всіма об'єктами системи без створення окремого користувацького інтерфейсу. Django Admin є повноцінним інструментом для CRUD-операцій та використовується для внутрішнього управління бізнес-даними[14].

У межах реалізованої інформаційної системи було налаштовано зручну адміністративну панель для роботи з такими моделями: користувачі, категорії товарів, товари, замовлення та позиції замовлень.

Ключові функції адміністративної панелі:

1. Створення та редагування товарів і категорій
2. Фільтрація та пошук товарів за назвою або категорією
3. Створення замовлень із кількома позиціями
4. Перегляд історії замовлень із зазначенням дати, типу та користувача
5. Керування ролями користувачів (admin, manager)
6. Автоматичне оновлення залишків при створенні або видаленні замовлення

```python
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'category', 'stock', 'price')
    search_fields = ('name',)
    list_filter = ('category',)
```

Рис. 3.8 — Реєстрація моделі товару

```python
class OrderItemInline(admin.TabularInline):
    model = OrderItem
    extra = 1

@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = ('id', 'order_type', 'user', 'created_at')
    list_filter = ('order_type',)
    inlines = [OrderItemInline]
```

Рис. 3.9 — Замовлення та позиції замовлень

```python
@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
    model = CustomUser
    list_display = ('username', 'email', 'role', 'is_staff')
    fieldsets = UserAdmin.fieldsets + (
        ('Роль користувача', {'fields': ('role',)}),
    )
```

Рис. 3.10 — Адміністрування користувачів

Переваги використання Django Admin:

1. Швидке розгортання та налаштування
2. Можливість керувати даними без написання додаткового коду
3. Безпечна автентифікація користувачів
4. Масштабованість: за потреби можна перевизначити відображення
5. Зручність для внутрішнього використання, особливо актуально для малого бізнесу

Таким чином, адміністративна панель Django забезпечує ефективне керування всіма аспектами системи, дозволяє легко адаптуватися до потреб користувача і значно скорочує час розробки[15].

3.5. Автоматичний контроль залишків товарів

Контроль кількості товару на складі є однією з основних функцій будь-якої облікової системи. У розробленій інформаційній системі реалізовано механізм автоматичного оновлення залишків на основі операцій, пов’язаних із замовленнями. Це дозволяє уникнути помилок, пов’язаних із ручним введенням даних, та завжди мати актуальну інформацію про запаси.

Принцип роботи

- Вхідне замовлення (Incoming) — система збільшує кількість товару на складі відповідно до зазначеної кількості у замовленні.
- Вихідне замовлення (Outgoing) — кількість товару на складі зменшується.

Реалізація через сигнали Django

Було реалізовано сигнал post_save, який автоматично викликається після створення об’єкта Order. Система проходить по всіх пов’язаних позиціях замовлення (OrderItem) і змінює значення поля stock у моделі Product.

```
from django.db.models.signals import post_save, post_delete
from django.dispatch import receiver
import Order
from inventory.models import Product

@receiver(post_save, sender=Order)
def update_stock_on_create(sender, instance, created, **kwargs):
    if created:
        for item in instance.items.all():
            if instance.order_type == 'in':
                item.product.stock += item.quantity
            elif instance.order_type == 'out':
                item.product.stock -= item.quantity
            item.product.save()
```

Рис. 3.11 — Сигнал після створення об’єкта Order

А також обробка видалення замовлень — щоб повернути кількість товарів у зворотному напрямку:

```
@receiver(post_delete, sender=Order)
def restore_stock_on_delete(sender, instance, **kwargs):
    for item in instance.items.all():
        if instance.order_type == 'in':
            item.product.stock -= item.quantity
        elif instance.order_type == 'out':
            item.product.stock += item.quantity
        item.product.save()
```

Рис. 3.12 — Обробка видалення замовлень

Захист від від'ємного залишку

У випадку створення outgoing-замовлення з кількістю, що перевищує поточний залишок товару, була реалізована додаткова перевірка (валідація), яка блокує збереження та повідомляє адміністратора про недостатній залишок.

Переваги підходу:

- Автоматизація — оператор не повинен вручну оновлювати кількість товарів
- Точність — мінімізація людського фактору
- Швидкість — зміни відбуваються одразу після дії користувача
- Гнучкість — логіку можна адаптувати до потреб

Таким чином, автоматичний контроль залишків товарів реалізовано як інтегровану частину бізнес-логіки системи, що працює безпосередньо при створенні або видаленні замовлень. Це значно підвищує надійність і зручність ведення обліку[15].

РОЗДІЛ 4. АПРОБАЦІЯ ДОДАТКУ

Після завершення проєктування та реалізації основної функціональності було здійснено апробацію розробленої інформаційної системи з метою перевірки її працездатності, відповідності вимогам, зручності використання та надійності. Апробація охоплює ручне функціональне тестування, проходження типових сценаріїв роботи користувачів.

Тестування проводилося в умовах, наближених до реального середовища малого торгового бізнесу. Було створено тестову базу даних з прикладами товарів, категорій, користувачів та замовлень. За результатами перевірок можна зробити висновок про стабільну роботу системи, правильність реалізації бізнес-логіки та можливість ефективного використання системи у практичній діяльності.

4.1. Ручне функціональне тестування

Ручне тестування — це процес перевірки працездатності системи шляхом послідовного виконання дій користувача без використання автоматизованих інструментів. Такий підхід дозволяє на ранньому етапі виявити логічні помилки, недоробки у взаємодії модулів або некоректну поведінку інтерфейсу.

Для тестування було підготовлено тестове середовище: локально розгорнутий проєкт Django, база даних PostgreSQL з початковими записами, створені ролі користувачів та відповідні сценарії.

Перелік перевірених функцій:

1. Користувачі:

- Створення адміністратора та менеджера через Django Admin
- Авторизація під різними ролями
- Перевірка доступу до об'єктів відповідно до ролі (менеджер не має доступу до редагування користувачів)

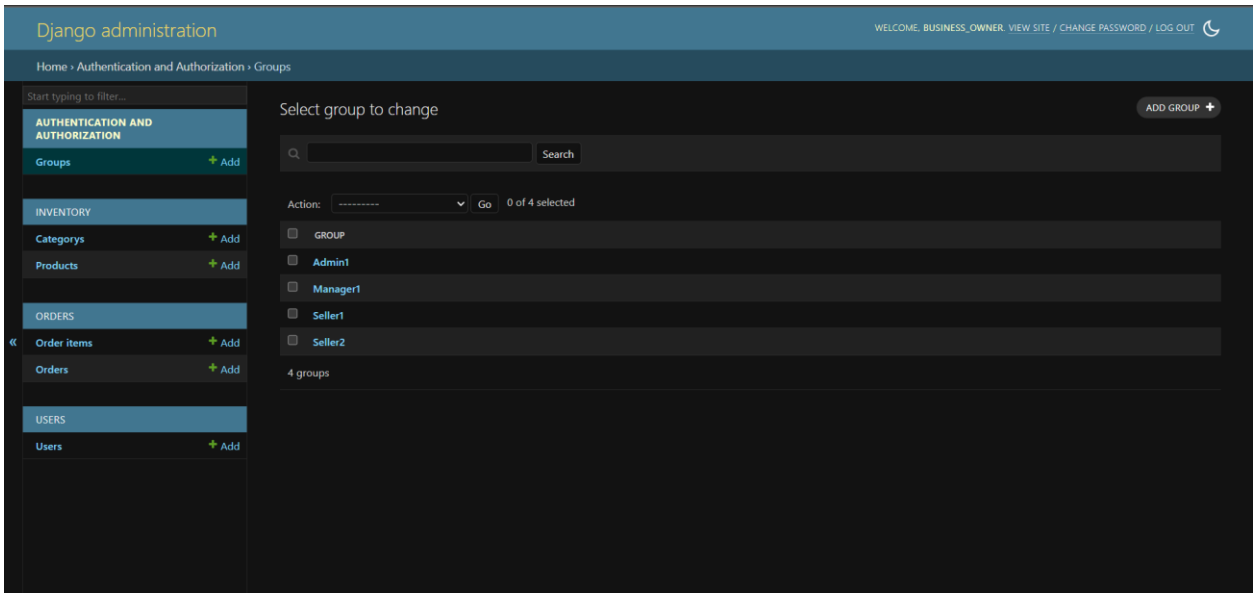

Рис. 4.1 — Створення та надання доступів користувачам

2. Товари:

- Додавання нових товарів і категорі
- Редагування даних про товар: назва, ціна, категорія, залишок

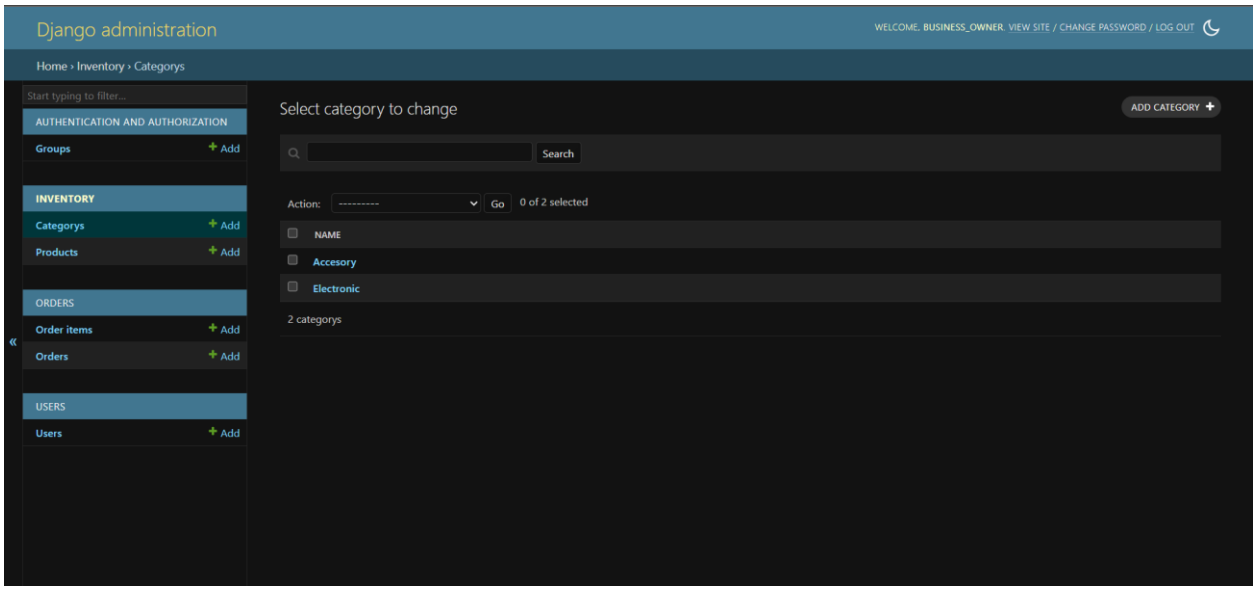

Рис. 4.2 — Створення категорій

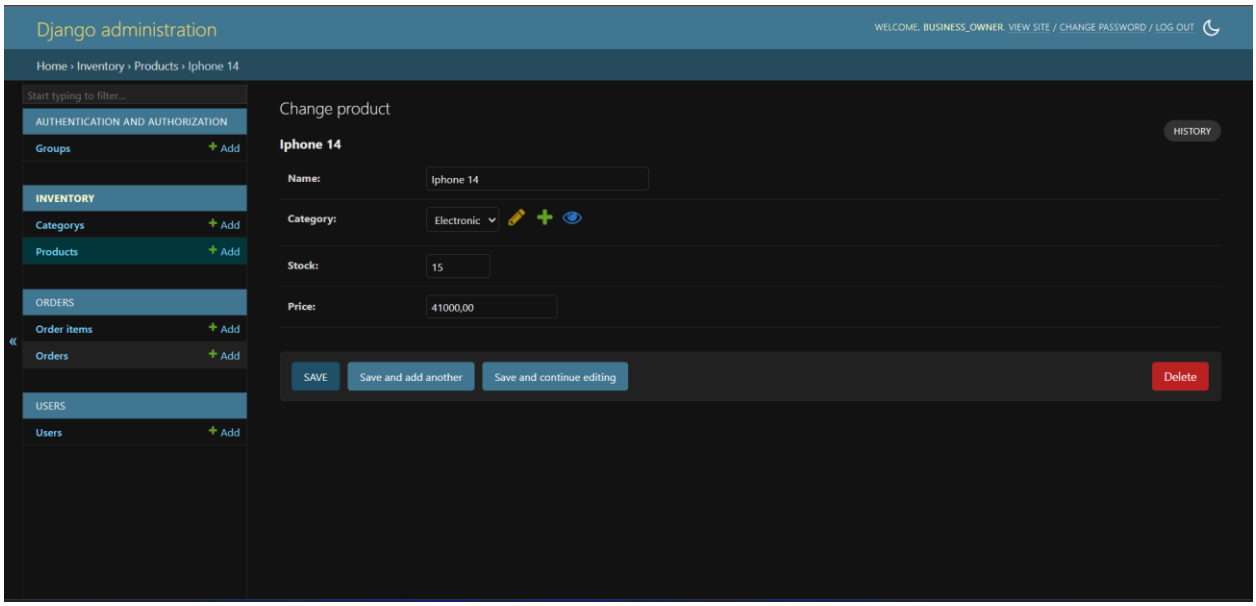

Рис. 4.3 — Створення товару

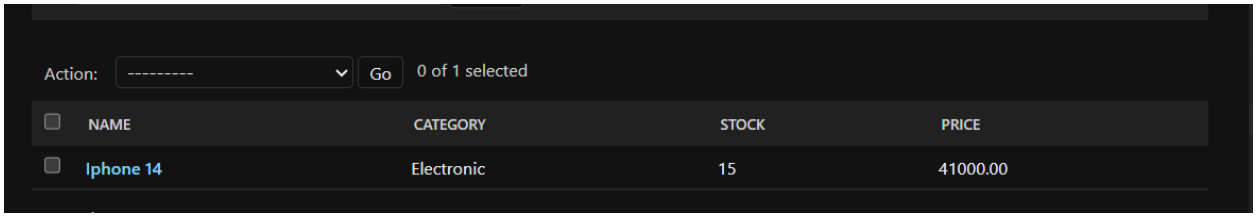

Рис. 4.4 — Відображення товарів у списку з фільтрацією та пошуком

3. Замовлення:

- Створення вхідного замовлення з кількома товарами

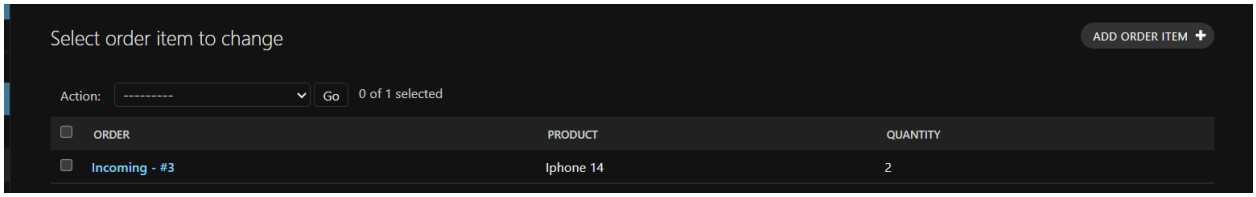

Рис. 4.5 — Вхідне замовлення

- Створення вихідне замовлення з перевіркою залишків

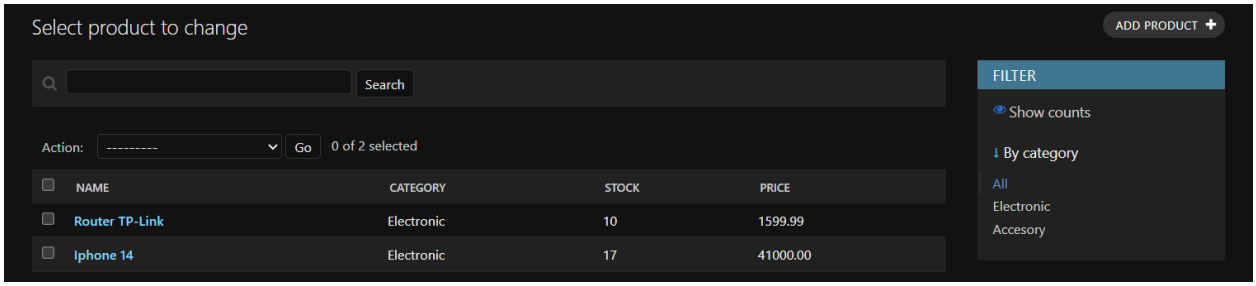

Рис. 4.6 — Вихідне замовлення

- Збереження інформації про автора замовлення

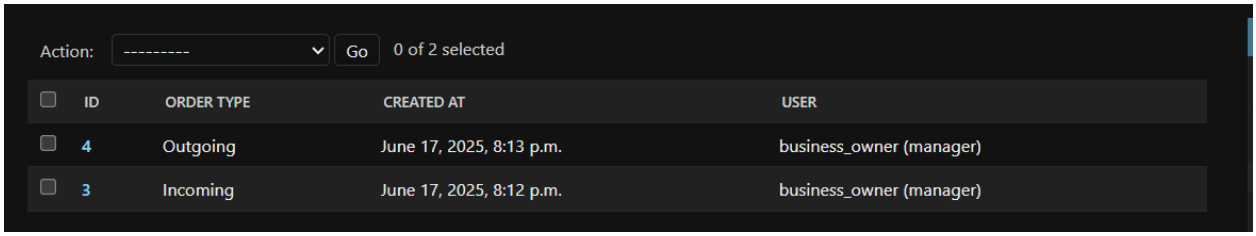

Рис. 4.7 — Автор замовлення

4. Адміністративна панель:

- Перевірка доступності та зручності інтерфейсу
- Відображення інформації в табличному вигляді
- Робота OrderItem для створення замовлень
- Пошук, сортування, фільтрація об’єктів

Критерії перевірки:

- Достіпні функції для конкретного користувача
- Правильне виконання бізнес-логіки
- Збереження змін в базі даних
- Виникнення критичних помилок
- Узгоджена робота модулів між собою

Результати:

- Усі основні функції працюють відповідно до технічних вимог
- Не виявлено критичних помилок або логічних суперечностей
- Залишки товару змінюються коректно, система обмежує некоректні дії
- Адмін-панель є зручною для користувачів із мінімальним технічним досвідом

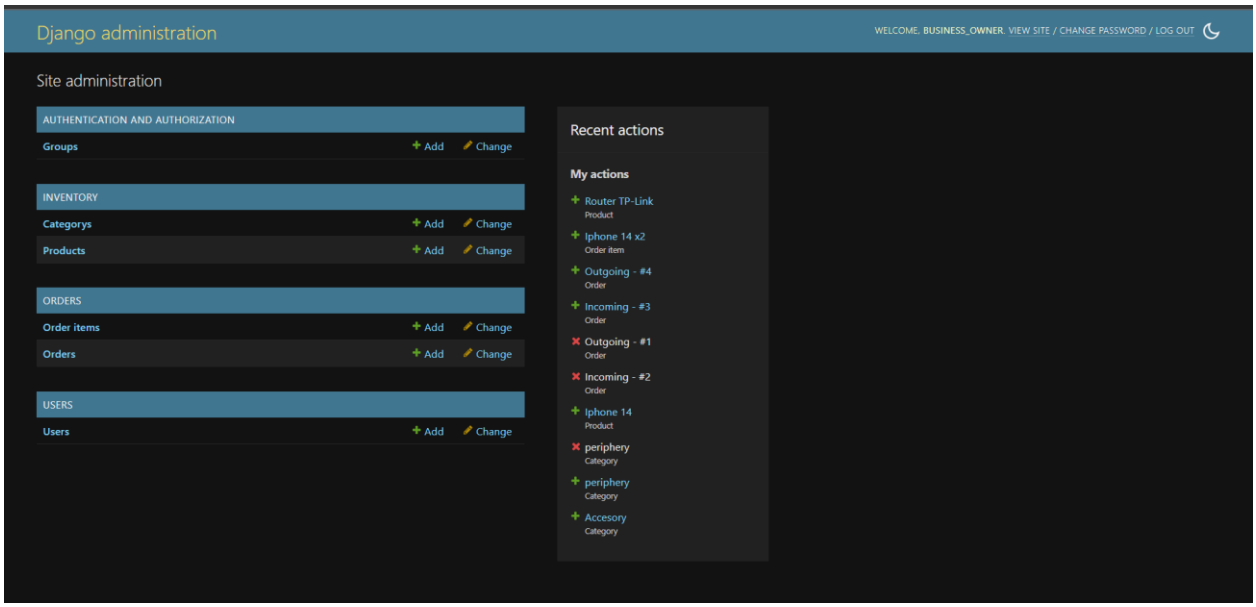

Рис. 4.8 — Адмін-панель

4.2. Сценарії взаємодії з системою

Для оцінки зручності та логічності використання системи були розроблені й протестовані типові сценарії роботи користувачів. Сценарії охоплюють ключові функції інформаційної системи: роботу з товарами, обробку замовлень, керування користувачами та контроль залишків.

Сценарій 1: Додавання нового товару до бази

Роль: Адміністратор

Дії:

1. Вхід у Django Admin.
2. Перехід до розділу «Categories», створення нової категорії
3. Перехід до розділу «Products» → створення нового товару
4. Назва: «Маршрутизатор TP-Link»
5. Категорія: «Електроніка»
6. Кількість: 10
7. Ціна: 1599.99 грн
8. Збереження товару

Результат:

- Товар з’являється в списку, доступний для замовлень

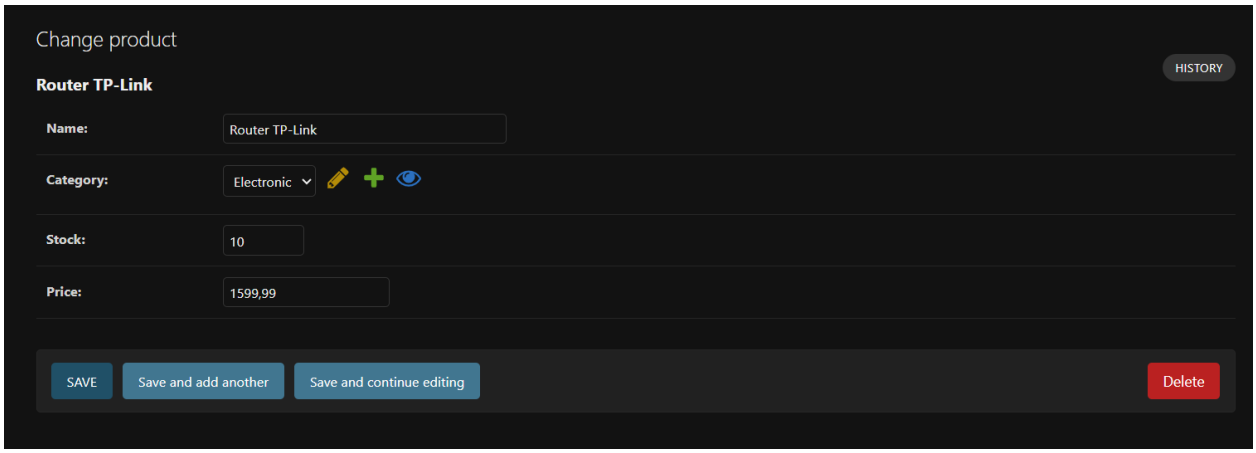

Рис. 4.9 — Створення товару

Сценарій 2: Створення вхідного замовлення (поставка товару)

Роль: Менеджер

Дії:

1. Вхід у систему
2. Перехід до розділу «Orders»
3. Створення нового замовлення
4. Тип: Incoming
5. Додавання товару «Iphone 14» у кількості 2.
6. Збереження замовлення.

Результат:

- Залишок товару автоматично оновлюється

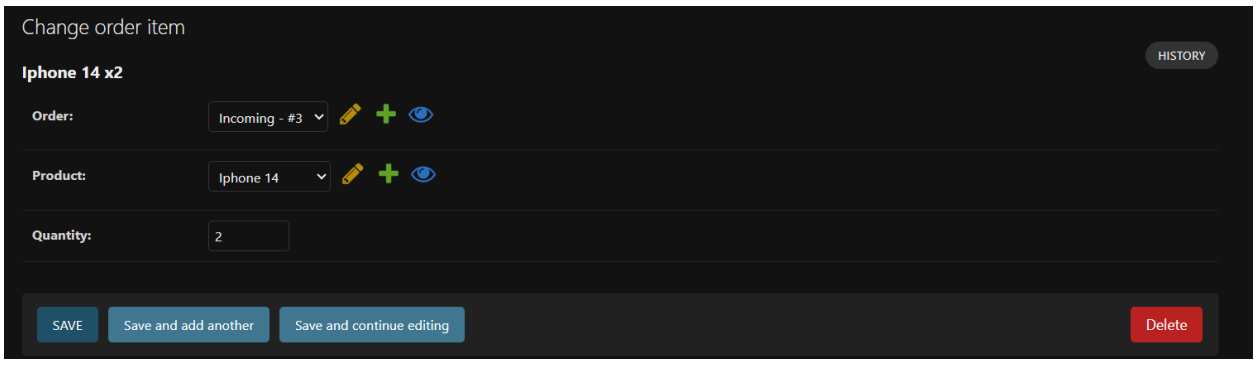

Рис. 4.10 — Створення товару

Сценарій 3: Перевірка історії замовлень

Роль: Адміністратор

Дії:

1. Перехід до розділу «Orders»
2. Перегляд списку замовлень: видно дату, користувача, тип, список товарів
3. Фільтрація за типом замовлення (Incoming або Outgoing) та за користувачем

Результат:

- Доступна прозора історія операцій. Легко відстежити, хто, коли і що замовляв

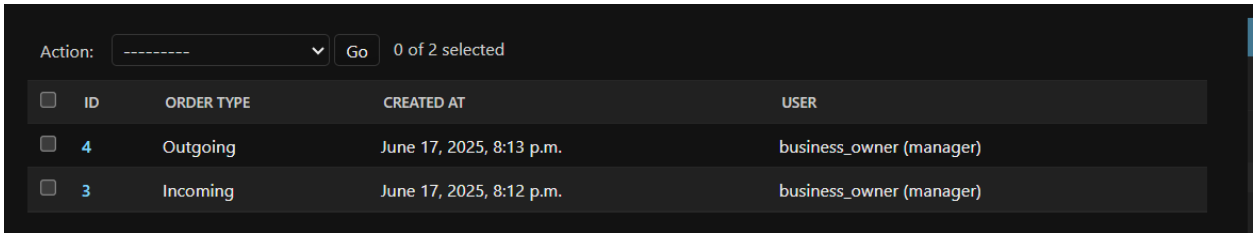

Рис. 4.11 — Історія замовлень

Сценарій 4: Перевірка прав доступу

Роль: Менеджер

Дії:

1. Вхід у систему
2. Спроба доступу до управління користувачами або видалення об’єктів

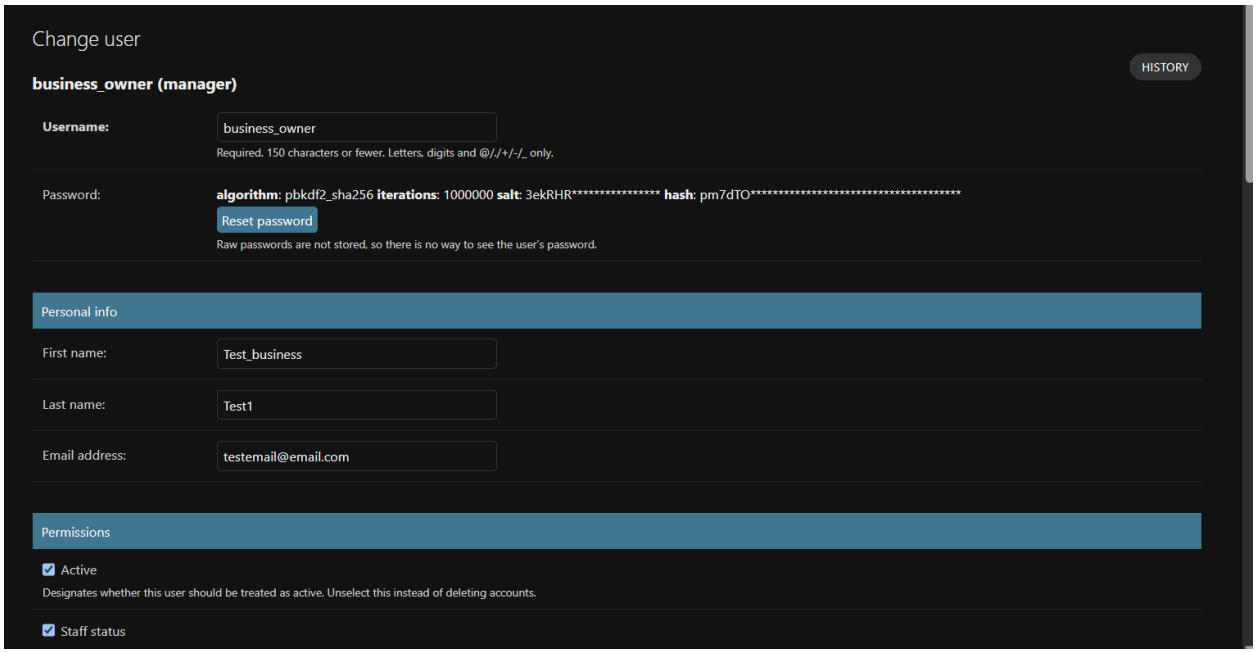

Рис. 4.12 — Права доступу

Результат:

- Система обмежує доступ, дії дозволені лише адміністраторам

Висновки за сценаріями:

- Усі бізнес-процеси можуть бути реалізовані через зручний інтерфейс без технічних навичок та складного навчання
- Залишки оновлюються автоматично, помилки виключаються логікою інформаційної системи
- Система підходить для реального використання у малому бізнесі

ВИСНОВОК

У межах кваліфікаційної дипломної роботи було розроблено, реалізовано та апробовано інформаційну систему для оптимізації та управління бізнес-процесами малого підприємства. Вибір теми був зумовлений реальними потребами малого бізнесу у простих, зручних та доступних програмних інструментах для обліку товарів, замовлень і контролю запасів.

На першому етапі проведено ґрунтовний аналіз предметної області, охарактеризовано типові процеси малого підприємства у сфері торгівлі, визначено проблеми, що виникають у разі ручного обліку, а також обґрунтовано необхідність автоматизації. Здійснено порівняння з існуючими ринковими рішеннями, що підтвердило доцільність створення власної системи, орієнтованої на конкретні потреби та сценарії використання.

У процесі проєктування було розроблено архітектуру клієнт-серверної інформаційної системи, вибрано актуальні технології розробки — Django та PostgreSQL — які забезпечили високу швидкість створення, масштабованість та надійність. Реалізовано модульну структуру, що включає додатки для управління товарами, замовленнями та користувачами.

Особливу увагу приділено реалізації автоматичного оновлення залишків товару при створенні замовлень, що дозволяє виключити людський фактор і забезпечує точність даних. Налагоджено адміністративну панель, яка дає змогу зручно керувати всіма елементами системи без потреби у додатковому інтерфейсі.

Під час апробації було проведено ручне функціональне тестування, перевірено сценарії взаємодії користувачів із системою, виявлено та усунуто низку технічних помилок. За результатами тестування система продемонструвала стабільну роботу, відповідність вимогам та готовність до впровадження в практичну діяльність.

Досягнуті результати:

- Створено повнофункціональну інформаційну систему з набором можливостей для обліку товарів та замовлень

- Реалізовано автоматичне керування залишками на складі
- Забезпечено рольову авторизацію користувачів
- Виконано повний цикл розробки — від постановки задачі до тестування та оцінки ефективності

Розроблена система може бути використана як основа для внутрішнього обліку на малих підприємствах. Вона є гнучкою, адаптованою для доопрацювання та має відкритий код, що дозволяє адаптувати її під конкретні потреби конкретного бізнесу.

Таким чином, поставлену мету дипломної роботи досягнуто повністю, а отриманий результат є функціональним, гнучким і практично застосовним рішенням для потреб малого бізнесу.

СПИСОК ЛІТЕРАТУРИ

1. Бублик В. П. Основи програмування та алгоритмізації: навч. посібник. — К. : КНЕУ, 2020. — 248 с.
2. Соловей В. І., Соловей О. В. Бази даних та системи управління базами даних: навч. посіб. — Харків: ХНУРЕ, 2021. — 190 с.
3. Django documentation — https://docs.djangoproject.com/en/5.2/
4. PostgreSQL official documentation — https://www.postgresql.org/docs/
5. pgAdmin Documentation — https://www.pgadmin.org/docs/pgadmin4/latest/
6. Django Admin — The Ultimate Guide. Real Python. — https://realpython.com/tutorials/django-admin/
7. Vincent, D. Django for Professionals. — Independently published, 2021. — 368 p.
8. Sobolev, A. Architecting Modern Django Apps. — HackSoft, 2022.
9. Smith, T. Web Development with Django. — Apress, 2020. — 310 p.
10. Романов І. В. Системи підтримки прийняття рішень: методи та засоби. — Львів: Видавництво ЛНУ, 2019. — 276 с.
11. GitHub – Django best practices & structure: https://github.com/django-best-practices
12. Калінін С. О. Проєктування інформаційних систем: навч. посіб. — К.: Вид-во Ліра-К, 2020. — 208 с.
13. Глушков В. М. Основи автоматизації бізнес-процесів. — Харків: ФОП Панов А. М., 2019. — 180 с.
14. Django REST Framework documentation — https://www.django-rest-framework.org/
15. RealWorld Example Apps – Django + PostgreSQL Stack — https://github.com/gothinkster/django-realworld-example-app

ДОДАТОК А

```
import sys
sys.setdefaultencoding = 'utf-8'

#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'business_system.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    stock = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.name



from django.db import models
from inventory.models import Product
from users.models import CustomUser
from django.core.exceptions import ValidationError

class Order(models.Model):
    ORDER_TYPE = (
        ('in', 'Incoming'),
        ('out', 'Outgoing'),
    )
    order_type = models.CharField(max_length=3, choices=ORDER_TYPE)
    created_at = models.DateTimeField(auto_now_add=True)
```

```
    user = models.ForeignKey(CustomUser, on_delete=models.SET_NULL,
null=True)

    def __str__(self):
        return f"{self.get_order_type_display()} - #{self.id}"


class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name='items')
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField()

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

    def save(self, *args, **kwargs):
        order_type = self.order.order_type

        product = self.product

        if self._state.adding:
            if order_type == 'out':
                if product.stock < self.quantity:
                    raise ValidationError(f"Insufficient accessory in stock for
{product.name} (there are: {product.stock}, need: {self.quantity})")
                product.stock -= self.quantity
            elif order_type == 'in':
                product.stock += self.quantity
```

```
            product.save()

        super().save(*args, **kwargs)
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import CustomUser

@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
    model = CustomUser
    list_display = ("username", "email", "is_staff", "is_active")
    list_filter = ("is_staff", "is_active")
    search_fields = ("username", "email")
    ordering = ("username",)
from django.apps import AppConfig


class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'
from django.contrib import admin
from .models import Product, Category

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name',)
    search_fields = ('name',)

@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
```

```
    list_display = ('name', 'category', 'stock', 'price')
    search_fields = ('name',)
    list_filter = ('category',)
from django.contrib import admin
from .models import Order, OrderItem

@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    list_display = ('id', 'order_type', 'created_at', 'user')  # поля, які дійсно є
    list_filter = ('order_type', 'created_at')

@admin.register(OrderItem)
class OrderItemAdmin(admin.ModelAdmin):
    list_display = ('order', 'product', 'quantity')
    list_filter = ('product',)
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),  # <- обов'язково має бути цей рядок
]
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    ROLE_CHOICES = (
        ('admin', 'Admin'),
        ('manager', 'Manager'),
    )
    role = models.CharField(max_length=10, choices=ROLE_CHOICES,
```

```
default='manager')

    def __str__(self):
        return f"{self.username} ({self.role})"
from django.apps import AppConfig


class OrdersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'orders'
from django.apps import AppConfig


class InventoryConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'inventory'
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-secure-!gjjljugfdg6e8sq9z5w%jhytr0m_kpr8x8hz'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'inventory',
    'orders',
    'users',
    'django.contrib.humanize',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'business_system.urls'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'business_system.wsgi.application'



# Database
# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'bizsys_db',
        'USER': 'test',
        'PASSWORD': '664535376733355',
        'HOST': 'localhost',
```

```
        'PORT': '5432',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.2/topics/i18n/
```

```
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True



# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

AUTH_USER_MODEL = 'users.CustomUser'
```