НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп’ютерних наук

_____ Голуб Б.Л.

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Інформаційна система управління ланцюгом постачання сільськогосподарської продукції

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми

Д.е.н., професор _____ Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Дудник А.О.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ Хомич В.А.

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
студентці Хомич Вікторії Андріївні

Спеціальність 122 - «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи: Інформаційна система управління ланцюгом постачання сільськогосподарської продукції

Затверджена наказом ректора НУБіП України від " 16 " грудня 2024р.

№ 2246 " С "

Термін подання завершеної роботи на кафедру 2025 . 06 . 02
рік, місяць, число

Вихідні дані до бакалаврської кваліфікаційної роботи: Результати аналізу літератури, інтернет-джерел.

Перелік питань що розглядаються:

1. Аналіз ринку фермерських послуг як предметної області
2. Інформаційне забезпечення системи управління ланцюгом постачання сільськогосподарської продукції
3. Розробка прикладного програмного забезпечення для автоматизації ланцюга постачання сільськогосподарської продукції
4. Впровадження та експлуатація інформаційної системи управління ланцюгом постачання сільськогосподарської продукції

Дата отримання завдання 2025 . 12 . 16

рік, місяць, число

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Дудник А.О.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Хомич В.А.

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

УКАБ - український клуб аграрного бізнесу

БД - база даних

СУБД - система управління базою даних

API - Application Programming Interface

CASE - Computer-Aided Software Engineering

UML - Unified Modelling Language

IBM - International Business Machines Corporation

ER - Entity-Relationship

SQL - Structured Query Language

VBA - Visual Basic for Applications

ARM - Advanced RISC Machine

HTML – HyperText Markup Languag

ВСТУП

У сучасних умовах аграрного сектору України ефективність постачання сільськогосподарської продукції є дуже актуальним питанням в економічній стабільності фермерських господарств. Більшість малих та середніх господарст стикаються з труднощами в обліку продукції, організації замовлень та виборі логістичних маршрутів. Наразі комп'ютеризація операцій, пов'язаних з оформленням замовлень, розрахунком даних для постачання, та інше, швидкість та точність яких, впливає на частку балансу фірм, відіграє дуже важливу роль. Водночас цифровізація бізнес-процесів у сфері агропромисловості залишається недостатньо розвинутою, що знижує конкурентоспроможність підприємств.

Одним із критичних моментів у діяльності фермерів є процес обробки замовлень та контролю за їх виконанням і доставкою до замовника. Часто облік ведеться вручну або у вигляді різного виду таблиць, що призводить до помилок, затримок, дублювання даних і ускладнення планування. Також бракує інструментів для наочного представлення логістичної інформації, наприклад — візуалізації маршруту доставки продукції. Саме тому розробка спеціалізованої інформаційної системи, яка б автоматизувала ключові етапи ланцюга постачання а саме — від внесення продукції в систему до побудови маршруту доставки, — є вкрай актуальним завданням. Впровадження такої автоматизованої системи дозволить не лише оптимізувати робочі процеси, а й покращити якість обслуговування клієнтів і рівень управління власним господарством.

Метою створення інформаційної системи управління ланцюгом постачання сільськогосподарської продукції є розробка простої у використанні, функціональної системи, яка дозволить фермерам вести облік продукції, а саме додававати нові товари та переглядати список всіх товарів, обробляти створені користувачами (це можуть бути як просто покупці так і інші фермери) замовлення та оптимізувати логістику доставки товарів у межах України, а саме

забезпечувати полегшення отримання інформації про відстані, вартості, адреси доставки та інші ключові дані замовлення. Особливу увагу в програмному продукті приділено обчисленню вартості доставки та відображенню маршруту замовлення. Актуальність такого рішення зумовлена відсутністю доступних інструментів для малих фермерських господарств, здатних швидко та комплексно вирішувати ці завдання без залучення сторонніх платформ.

Для реалізації програмного забезпечення системи були використані сучасні методи та технології розробки, включаючи об'єктно-орієнтоване програмування, UML-моделювання предметної області (діаграми прецедентів, діяльності, послідовності), а також інші засоби для проєктування баз даних. У межах створення інтерфейсу було застосовано принципи адаптивного дизайну, які будуть забезпечувати зручність використання як на десктопних, так і на мобільних пристроях. Для подання даних про маршрути вокористовувалися спеціальні формули та візуалізація у вигляді карти.

Представлення та аналіз результатів роботи над системою здійснювалися в рамках виконання курсових та лабораторних робіт. У перспективі можливе подання матеріалів для публікації у фахових виданнях ІТ та аграрної інформатики.

Розроблена система забезпечує: облік фермерської продукції, реєстрацію замовлень, отримання сповіщень в реальному часі при зміні статусу замовлення, збереження контактної інформації клієнтів, вивід детальної інформації про постачання, а також візуалізацію маршруту доставки на карті. Таким чином, вона поєднує функції базового управління та зручного представлення інформації, що дозволяє фермерам краще організовувати роботу та швидше ухвалювати рішення.

У ході роботи було реалізовано такі завдання:

- аналіз предметної області та визначення вимог до системи;
- створення моделі даних та побудова бази даних;

- проєктування користувацького інтерфейсу;
- реалізація функціоналу на мові Python з використанням бібліотек для роботи з графічним інтерфейсом і картами;
- тестування роботи системи на прикладах реальних сценаріїв;
- опис структури програмного продукту та можливостей його розширення в майбутньому.

Кінцевими користувачами системи можуть бути фермери, працівники аграрних кооперативів або менеджери дрібних постачальників, які прагнуть мати доступ до своєї інформації швидко, зрозуміло і без зайвих зусиль. Автоматизація цих процесів сприятиме підвищенню ефективності, зменшенню помилок та втрат, а також модернізації управлінських рішень у фермерському господарстві.

Пояснювальна записка містить 67 сторінок основного тексту, 3 додатки, 37 використаних джерел та включає такі розділи: у першому розділі проаналізовано предметну область системи, визначено потреби користувачів та проаналізовано існуючі рішення ; у другому — описано інформаційне забезпечення, побудовано ER-діаграму та структуру бази даних; у третьому — представлено реалізацію програмного додатку та особливості вибраного технологічного стеку; у четвертому — проведено тестування, наведено приклади інтерфейсів та описано можливості подальшого розширення системи.

.

1 АНАЛІЗ РИНКУ ФЕРМЕРСЬКИХ ПОСЛУГ ЯК ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Основне призначення системи управління ланцюгом постачання сільськогосподарської продукції це - підтримка фермерських господарств у реалізації їхньої продукції без посередників. Основним завданням даної системи є забезпечення зручного механізму обліку товарів, автоматизація обробки вхідних замовлень і надання інформації про маршрути доставки та візуалізацію за допомогою вбудованих карт.

Так як сільськогосподарський сектор продовжує оцифровуватися, інформаційні рішення, що забезпечують ефективну логістику та збут фермерської продукції, починають набувати дедалі більшого значення. За даними Продовольчої і сільськогосподарської Організації Об'єднаних Націй (ФАО), автоматизація в сільському господарстві може допомогти скоротити витрати, поліпшити обслуговування клієнтів і прискорити обробку замовлень.[14]

Ручне опрацювання замовлень забирає багато часу і загрожує такими помилками як: дублюванням даних, втрата інформації, невірні адреси або кількість продукції. Крім того, необхідно швидко вибрати оптимальний маршрут доставки. Хоча програмне забезпечення не дає змоги проводити складний аналіз варіантів логістики, воно дає змогу фермерам бачити ключову інформацію, як-от замовлення, міста доставки, транспортні витрати тощо, і візуалізувати її на карті. Це спрощує процес прийняття рішень і знижує залежність від паперових документів і ручних розрахунків.

Цілі розробки:

- забезпечити зручний облік товарів, що підлягають продажу;
- автоматизувати процес приймання замовлень від замовників;
- надавати фермерам повну інформацію про доставку: пункт призначення, відстань, вартість;
- створити можливість в реальному часі відслідковувати статус замовлення та доставки;
- візуалізувати маршрут доставки на карті.

Функціональність системи передбачає:

- реєстрацію та авторизацію користувачів (фермерів і клієнтів);
- додавання та перегляд товарів;
- оформлення замовлення клієнтами(як клієнтами так і самими фермерами);
- збереження та облік усіх замовлень у базі даних;
- сповіщення користувачам про зміни статусів їх замовлень;
- виведення деталей доставки: місто, вартість перевезення, маршрут;
- інтеграцію з картографічним API для побудови візуалізації маршруту.

Особлива увага приділяється зручності користування: інтерфейс повинен бути простим, з мінімальним набором дій для додавання товару, приймання замовлення та перегляду інформації про маршрут. Безпека даних користувача забезпечується авторизацією та обмеженням доступу.

Система має містити базу даних із такими відомостями:

- інформація про користувачів (список клієнтів і фермерів та адміністратор);
- інформація про наявні товари;
- інформація про замовлення з даними про клієнта, місто доставки, дату.

Функціональні вимоги до системи:

- збереження, редагування та видалення даних;
- пошук і фільтрація товарів і замовлень;
- експорт списків замовлень у формат Excel (за потреби);
- інтерактивна мапа доставки з маркерами.

Потреба в такій системі зумовлена тим, що малий агробізнес повинен пройти цифрову трансформацію, щоб налагодити прямий контакт з клієнтами. Згідно з дослідженням Українського клубу аграрного бізнесу (УКАБ), однією з головних перешкод для розвитку місцевого сільського господарства є відсутність доступних цифрових сервісів для торговельних організацій.[16]

1.2 Огляд існуючих рішень та інформаційних джерел

На даному етапі цифрової трансформації сільського господарства автоматизація таких процесів, як облік та логістика, є дуже важливим кроком у підвищенні ефективності фермерських господарств. Останніми роками в науковій літературі і програмних рішеннях дедалі більше уваги починає приділятися інформаційним системам для підтримки планування, обліку та контролю сільськогосподарських поставок.

Серед літератури, слід виокремити працю Яцкевич І. В. та Красностанової Н. Е., де обговорюється вплив цифрових технологій на бізнес, у тому числі на аграрний сектор. Автори підкреслюють, що використання цифрових рішень дозволяє підвищити прозорість, ефективність та аналітичність процесів, пов'язаних із закупівлями, логістикою та взаємодією з клієнтами. [12]

Серед доступних програмних продуктів, які частково або повністю виконують функції управління сільськогосподарськими замовленнями та логістики, можна відзначити наступні:

- Farmigo - це онлайн-ринок фермерських продуктів, який з'єднував споживачів, робочі місця, школи, житлові комплекси та громадські центри безпосередньо з місцевими фермерськими господарствами. Система орієнтована на допомогу малим фермерам в управлінні замовленнями на продукти, які мають постачатися напряму споживачам або у пункти видачі. Платформа дозволяла створювати каталог товарів, приймати онлайн-замовлення та оптимізувати логістику. Простий веб-інтерфейс, проте відсутня інтеграція з інтерактивними картами для візуалізації маршрутів.

  Недоліки системи: обмежена географія- сервіс був доступний лише в окремих регіонах США, що обмежувало його охоплення, також відсутність інтерактивної візуалізації маршрутів- не було реалізовано функціоналу для візуалізації маршрутів доставки на карті.[19, 20]
- Freshspire допоможе тримати клієнтів в курсі запасів фермера, наявності товарів та прайс-листів. Ця система допомагає місцевим фермерам комунікувати з роздрібними продавцями, ресторанами та покупцями. Її основні функції це: облік замовлень, виставлення рахунків і управління запасами. Система має потенціал для логістики, однак не реалізує візуальну побудову маршрутів доставки.

  Недоліки системи: відсутність візуалізації маршрутів - система не надає можливості візуального планування маршрутів доставки; також обмежений функціонал для фермерів- основний акцент в системі зроблено на взаємодії з роздрібними продавцями та ресторанами, що може не повністю відповідати потребам дрібних фермерів.[21, 22]
- Local Line – ще одна платформа для купівлі та продажу продуктів харчування. Система активно використовується в Канаді, США та багатьох інших країнах. Серед основних можливостей - створення інтернет-магазину, який буде індивідуальним для кожного фермера, ключове у функціоналі це: прийом замовлень, автоматизація обліку

продажів і створення маршрутів доставки на основі створеного розкладу. Крім того, система підтримує експортування звітної документації та дозволяє редагувати вартість доставки відповідно за територією обслуговування, що значно спрощує логістичні процеси.

Недоліки системи: висока вартість- мінімальна вартість підписки становить $50 на місяць, що може ставати суттєвим бар'єром для малих фермерських господарств; обмежена підтримка мов- платформа доступна лише кількома мовами, що може ускладнити її використання в регіонах з іншими мовами, оскільки платформа розрахована на велике офоплення користувачів; відсутність візуалізації у вигляді карти, що не є надважливим але без цього система виглядає неповноцінною.[23]

Інформаційна система управління ланцюгом постачання сільськогосподарської продукції - це інноваційний інструмент, призначений впорядкувати та автоматизувати процес збуту сільськогосподарської продукції без необхідності залучення посередників. Унікальність системи полягає в тому, що вона орієнтована на малих та середніх виробників України, які поки не мають власних ІТ-рішень, але потребують впровадження ефективного механізму взаємодії з кінцевими споживачами. Система має зручний інтерфейс, який надає змогу фермерам самостійно керувати доставкою своєї продукції, а клієнтам - формувати замовлення та вводити безпосередньо свої дані та адреси доставки.

До основних переваг системи можна віднести те, що вартість доставки може бути розрахована автоматично на основі відстані між населеними пунктами, що виключає ручні розрахунки та підвищує прозорість цін. Система також включає карти маршрутів доставки, тобто візуалізацію, що дозволяє фермерам краще планувати свою логістику. Така функція рідко реалізована на простих онлайн-платформах для фермерів, саме тому вона робить систему ще більш унікальною, практичною і корисною, особливо в сільській місцевості, де доставка може бути складним і дорогим процесом.

Незважаючи на переваги системи, вона звісно має і певні обмеження. Зокрема, система не підтримує фінансові транзакції та інтеграцію з платіжними сервісами, (тобто відсутні будь-які фінансові операції таке як можливість оплати замовлень) що вимагає додаткової координації між фермерами та клієнтами за межами платформи, як мінімум підтвердження замовлення та оплата. Крім того, система не включає управління запасами, а зосереджена лише на замовленні та етапах доставки.

1.3 Моделювання предметної області

Важливим етапом проєктування інформаційної системи для всіх типів підприємств є моделювання предметної області. На цьому етапі відбувається формалізація знань про систему, що підлягає майбутній автоматизації. Якість моделі визначає, насамперед, ефективність наступних етапів розробки, від проектування до впровадження. Досвід багатьох IT-проектів показує, що витрати на моделювання та проектування можуть становити 70-80% від загальної вартості розробки програмного забезпечення . Тому точність, повнота і несуперечливість моделі є критично важливими для запобігання ризиків на наступних етапах.[16]

Набагато легше виявити логічні помилки, прогалини або суперечності у системних вимогах до програмного забезпечення на етапі моделювання, ніж усувати їх у готовому продукті. Невдала або навіть неповна модель предметної області часто призводить до необхідності значних доопрацювань, чим затягує терміни розробки і ставить під загрозу весь проєкт.

Інформаційна модель системи - це формальний опис взаємозв'язків між такими елементами, як дані, процеси, об'єкти та ролі користувачів, включаючи структуру бази даних, бізнес-логіку та логіку взаємодії між компонентами системи. Для забезпечення узгодженості, простоти і гнучкості моделі, особливо

у великих і складних системах, слід використовувати професійні інструменти моделювання [33].

CASE-засоби (Computer-Aided Software Engineering) часто використовуються для створення моделей, які підтримують різні нотації та підходи до проектування. Однією з найбільш популярних мов моделювання є UML (Unified Modelling Language) - уніфікована мова моделювання, яка є міжнародним стандартом для створення моделей програмних систем [35]. UML може використовуватися для опису як статичної структури системи (класи, об'єкти, компоненти), так і її динамічної поведінки (послідовності дій, стани, сценарії взаємодії).

Моделі UML зрозумілі не тільки програмістам, а й аналітикам, тестувальникам і керівникам проєктів, що робить їх універсальним засобом комунікації всередині команди. Завдяки ясності та формальності UML-діаграм можна створювати концептуальні, логічні та графічні моделі, що відповідають вимогам конкретного проєкту [36].

UML-діаграми

Для створення UML-діаграм використовуються сучасні інструменти, як-от Visual Paradigm, StarUML, Enterprise Architect, Draw.io, а також більш потужні інструменти, як-от IBM Rational Rose, MagicDraw UML, Microsoft Visio і т. д. [2, 36]. В даному проєкті використовується саме засіб Draw.io, так як він є одним з найзручніших, до того ж хмарним середовищем, що надає змогу працювати над моделюванням будь де та будь коли.

Однією з діаграм UML, що використовується на першому етапі створення логічної моделі інформаційної системи, є так звана діаграма прецедентів (або діаграма варіантів використання). Діаграма призначена для створення концептуальної моделі функціонування системи в середовищі. Діаграма варіантів використання складається з сутностей (акторів), варіантів використання (прецедентів), обмежених умовними межами системи (зазвичай

прямокутниками), асоціацій між сутностями та варіантами використання, зв'язків між варіантами використання та узагальнених зв'язків між сутностями.

Діаграма варіантів використання UML показує всі взаємозв'язки, які виникають між акторами, а також між різними варіантами використання. Основна мета - надати клієнтам, кінцевим користувачам або розробникам повноцінний засіб для спільного обговорення поведінки та функціональності конкретної системи.[2]

В системі, яка буде розроблятися, існують два основні актори:

- Фермер
- Замовник

Фермер має доступ до всіх варіантів використання, оскільки він може як додавати товар так і замовляти у інших фермерів. Він може додавати нові товари до системи, переглядати їх, формувати замовлення, а також переглядати маршрут і його деталі. Це забезпечує повноцінний контроль за пропозицією та процесом доставки продукції до кінцевого замовника.

Замовник має доступ не до всіх варіантів використання. Він не може додавати товари, але може обирати доступну продукцію, формувати замовлення, а також переглядати маршрут і вже оформлені замовлення. Таким чином, забезпечується зручна система для отримання актуальної інформації та здійснення замовлень.

Детальніше про ролі фермера та замовника можна побачити на діаграмі прецедентів (рис. 1).

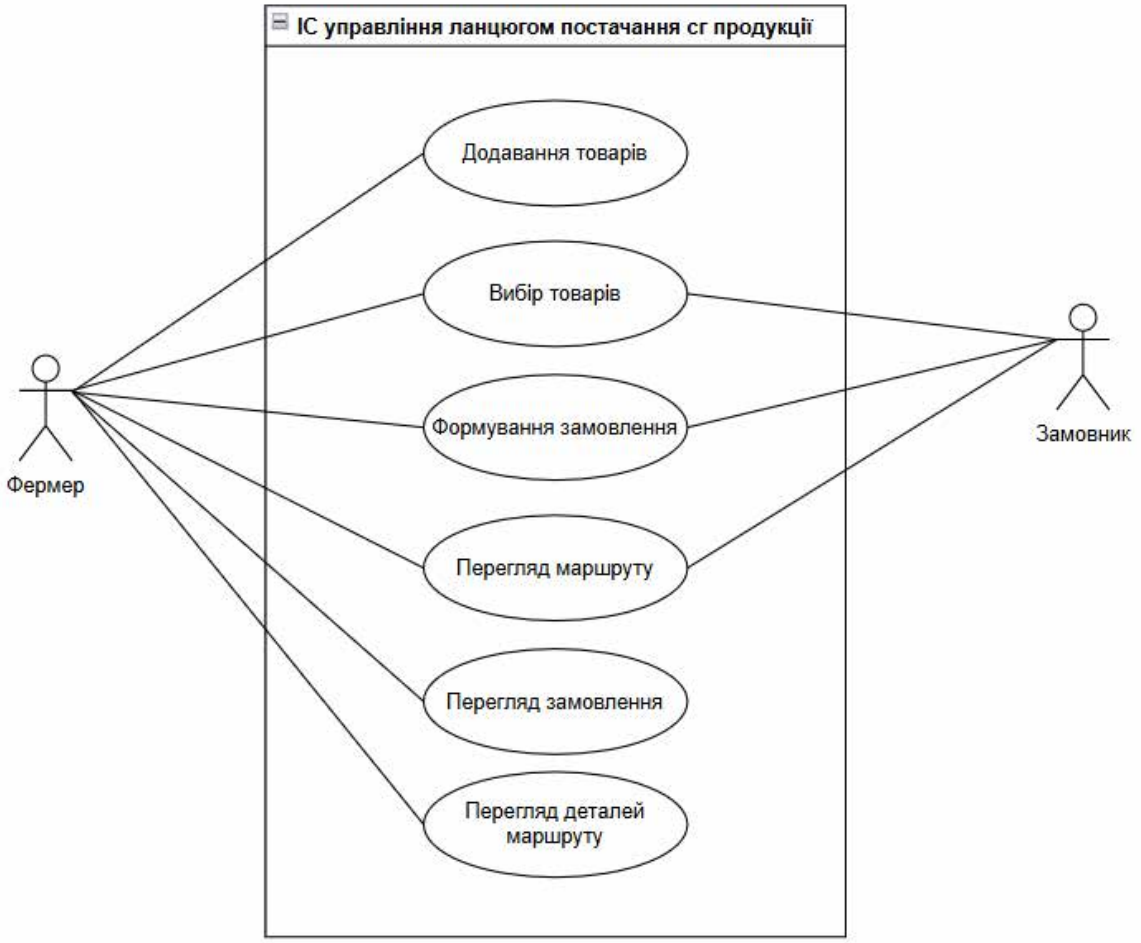


Рис. 1 Діаграма прецедентів

Діаграма діяльності є одним з основних інструментів візуального моделювання поведінки інформаційної системи. За своєю структурою вона схожа на блок-схему, але замість опису системних процесів на них зображуються окремі функціональні або логічні кроки, що виконуються системою або користувачем. Саме ці логічні кроки є основою для розробки інформаційної системи, і які потрібно автоматизувати. Тому, перед початком розробки, необхідно чітко визначити алгоритми роботи.

Основне призначення діаграм діяльності - представити бізнес-процеси організації в алгоритмічній формі. Наприклад, при моделюванні бізнес-операцій, таких як робота сільськогосподарського підприємства, діаграми діяльності допомагають візуалізувати основні етапи виконання завдань, які в майбутньому будуть автоматизовані за допомогою інформаційних систем.[36]

На діаграмі діяльності, або активності (рис.2) зображено алгоритм обробки нового замовлення фермером.

Рис. 2 Діаграма діяльності

Діаграма послідовності (Sequence Diagram) — це в UML тип діаграми взаємодії, яка фокусується на певній послідовності в часі обміну повідомленнями між об'єктами. Її головної метою є показати, які об'єкти взаємодіють між собою та в якому саме порядку це відбувається. Такі діаграми є особливо корисними для моделювання логіки реалізації варіантів використання або конкретних операцій у системі.

У діаграмі послідовності кожен об’єкт представлений у вигляді вертикальної лінії життя, а повідомлення, які передаються між ними, - у вигляді

горизонтальних стрілок. Вертикальна вісь діаграми відображає плин часу: чим нижче елемент — тим пізніше відбувається дія. Стрілки, які з'єднують лінії життя, позначають виклики методів, надсилання запитів або відповіді між об'єктами.

Ці діаграми дозволяють детально проаналізувати, як саме реалізується певний функціонал системи, зокрема порядок взаємодії її компонентів. Це важливо на етапах аналізу та проєктування, коли потрібно уточнити сценарії використання і способи досягнення результату.

Діаграми послідовності добре підходять для:

- моделювання сценаріїв, де чітко визначений порядок дій;
- виявлення прихованих залежностей між об'єктами;
- документація логіки програмних модулів;
- комунікація між розробниками та аналітиками. [37]

Діаграма послідовності (рис. 3) ілюструє основну взаємодію між двома головними учасниками системи управління ланцюгом постачання сільськогосподарської продукції, а саме фермером та замовником.

Весь процес починається з того, що користувач (фермер) додає товар до каталогу . Після цього замовник обрає потрібний йому товар та знайомиться з деталями замовлення, які його цікавлять. Після вибору відбувається підтвердження замовлення фермером.

Далі, коли фермер отримує нове замовлення, він переглядає маршрут доставки і приймає рішення щодо його виконання. Після цього фермер відправляє замовлення, яке надходить до замовника. Останнім кроком є підтвердження отримання замовлення з боку замовника.

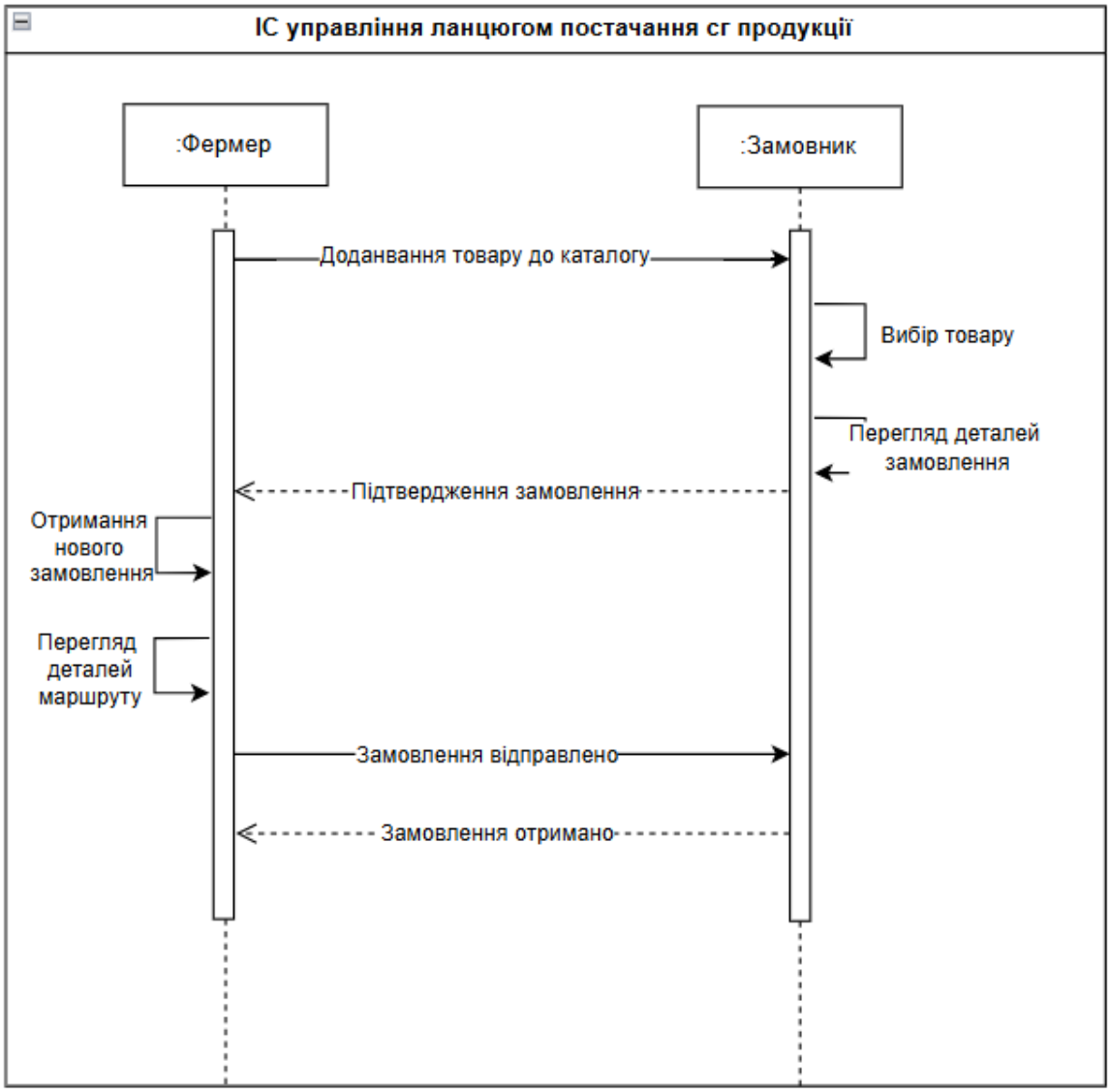


Рис. 3 Діаграма послідовності

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ УПРАВЛІННЯ ЛАНЦЮГОМ ПОСТАЧАННЯ СІЛЬСЬКОГОСПОДАРСЬКОЇ ПРОДУКЦІЇ

2.1 Логічна модель даних

Якість створеної бази даних безпосередньо залежить від ретельного виконання кожного етапу її проектування. Особливо важливим є етап побудови логічної моделі даних, оскільки саме вона забезпечує відповідність бази даних реальній предметній області, а також визначає структуру фізичної БД, що впливає на її подальше використання.

Одна й та сама інформація може бути організована у вигляді таблиць (відношень) по-різному, тобто існує можливість створення різних варіантів зв’язків між об’єктами предметної області. Угруповання атрибутів у відношеннях повинно бути оптимальним: воно має зменшувати надмірність даних і спрощувати процеси їх обробки та оновлення.

Найбільш ефективні варіанти структурування відношень забезпечуються дотриманням вимог нормалізації – певного підходу, що регламентує побудову таблиць у БД. Нормалізація дозволяє мінімізувати дублювання інформації, гарантувати її логіку, узгодженість і скоротити витрати на супровід бази даних.[4]

На практиці найбільш поширеними є перша, друга та третя нормальні форми.

Відношення вважається нормалізованим або таким, що відповідає першій нормальній формі, тоді, коли всі його атрибути є атомарними, тобто неподільними. Властивості відношення у 1НФ(першій нормальній формі) включають:

- відсутність однакових атрибутів;
- невпорядкованість атрибутів;
- атомарність значень атрибутів.

Таким чином, будь-яке відношення у реляційній моделі вже задовольняє вимогам першої нормальної форми.[5]

Тим не менш, перша нормальна форма допускає різні типи інформації в одному контексті, що часто призводить до надлишку даних і, таким чином, до не зовсім адекватного представлення предметної області. Тому для отримання якісної моделі логічних даних недостатньо використовувати лише першу нормальну форму.

Відношення можна вважати відповідним другій нормальній формі (2НФ), якщо воно коректно зведене до першої нормальної форми (1НФ) і не містить неключові атрибути, які залежать лише від декількох складних ключів. Іншими словами, всі неключові атрибути повинні повністю, а не частково залежати від ключа.

Це означає, що якщо ключ складається лише з одного поля (тобто є простим ключем), то відношення автоматично буде 2НФ.

Однак, навіть якщо відношення відповідає другій нормальній формі, воно може містити «додаткову» інформацію або різні типи інформації. Як наслідок, іноді доводиться додавати додаткові інструменти (наприклад, тригери), щоб база даних працювала належним чином. Саме тому для отримання кращої структури даних необхідно використовувати третю нормальну форму (3НФ).

Відношення буде відповідати 3НФ, якщо воно вже приведене до 2НФ і якщо між неключовими атрибутами вже немає залежностей. Тобто кожен атрибут, неключовий, повинен залежати тільки від ключа, і не більше.

Коли всі таблиці в базі даних відповідно приведені до третьої нормальної форми, це допомагає уникнути зайвих повторів і зробити модель даних більш

зручною для роботи. У такій базі вже не потрібно писати багато складного коду а достатньо лише стандартних інструментів, які забезпечують правильні зв’язки між таблицями.

На практиці, для розробки логічної моделі бази даних, найчастіше використовують різні варіанти ER-діаграм, підтримувані відповідними CASE-засобами. Моделювання предметної області базується на використанні графічних діаграм, що включають в себе невелике число різного роду компонентів. [35]

Основними поняттями ER-моделі є сутність, зв'язок і атрибут.

Сутність - це реальний або представлений об'єкт, інформація про який повинна зберігатися і бути доступною. У діаграмах ER-моделі сутність представляється у вигляді прямокутника, що містить назву сутності. При цьому назва сутності - це назва типу, а не деякого конкретного примірника цього типу. Для більшої виразності і кращого розуміння назва сутності може супроводжуватися прикладами конкретних об'єктів цього типу.

Атрибут сутності - це деяка характеристика, яка описує одне з її властивостей, що є рівнозначне поняттю атрибута в відношенні. Атрибут має ім'я і приймає значення одного з деякого безлічі значень. Кожен екземпляр сутності мусить відрізнятися від будь-якого іншого примірника тієї ж сутності .

Зв'язок - це графічне зображення асоціації, яке встановлюється між двома сутностями. Така асоціація завжди є бінарною і може існувати між двома різними сутностями або між сутністю і сама самою (рекурсивний зв'язок). У будь-якого зв'язку виділяються два кінці (відповідно до існуючої пари пов'язаних сутностей), на кожному з яких вказується назва кінця зв'язку, ступінь кінця зв'язку (скільки екземплярів даної сутності зв'язується), обов'язковість зв'язку (тобто будь-який чи примірник даної сутності повинен брати участь в зв'язку з іншою сутністю).

Зв'язок представляється у вигляді лінії, що зв'язує дві сутності або веде від сутності до неї ж самої. Обов'язковий кінець зв'язку зображається суцільною лінією, а необов'язковий - переривчастою.

Як і сутність, зв'язок - типове поняття, всі екземпляри обох пар пов'язаних сутностей підкоряються правилам зв'язування.[4,5]

На рисунку (рис.4) зображено модель бази даних інформаційної системи виконана в середовищі ERWin DataModeler. Вона описує логіку збереження даних системи.

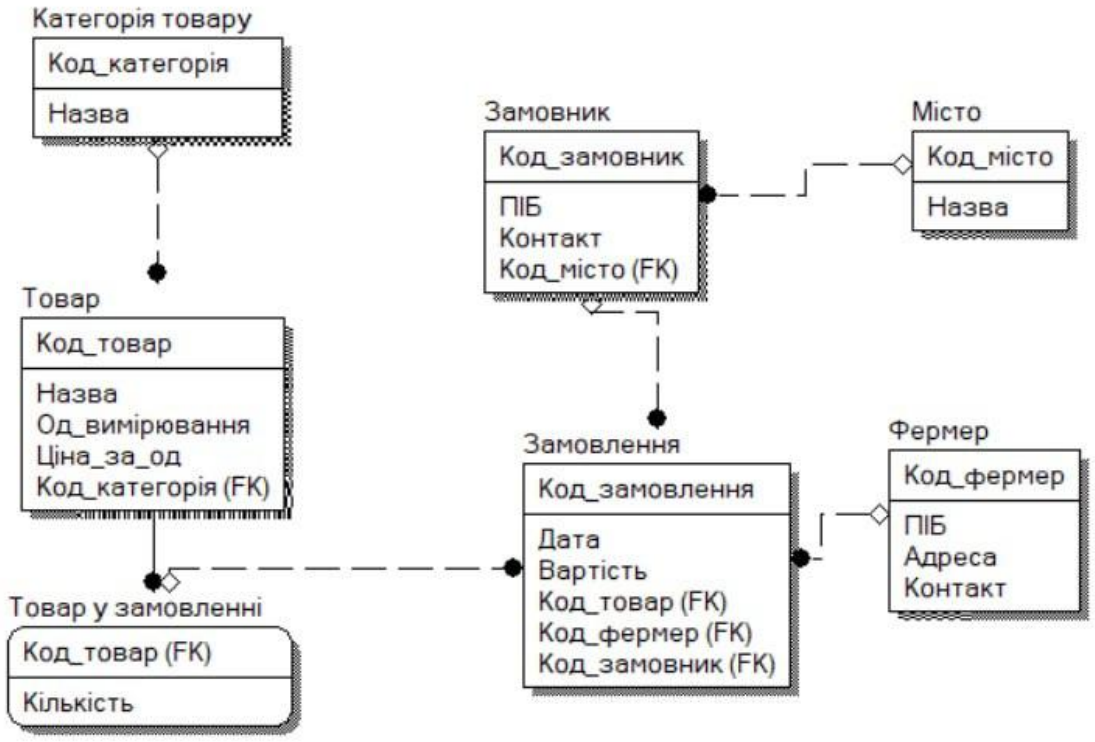


Рис. 4 Логічна модель даних

2.2 Вибір системи управління інформаційною базою

Система управління базою даних виконує велику кількість функцій, які забезпечують цілісність та несуперечність даних, які зберігаються в базі. Вона створює спеціальні структури, що необхідні для зберігання даних, які, в свою

чергу, звільняють користувача від важкого завдання визначення і програмування фізичних властивостей даних. Така система забезпечує зберігання не тільки даних, але і пов'язаних з ними екранних форм, макетів звітів, правил перевірки даних, коду процедур і т.д.

Система управління базами даних структурує дані, які вводяться, перетворюючи в фізичний логічний формат даних, у форму, яка зручна для зберігання. При виконанні цих логічних запитів визначається фізичне розташування необхідних даних і їх вилучення, тобто знову надає їм форму, легку для читання. Таким чином, СУБД забезпечує програмну незалежність і абстракцію даних. [7]

Вимоги до системи управління базами даних такі:

- система мусить мати можливість пошуку, модифікації та зберігання даних;

- забезпечувати швидкий доступ до них;

- захищати цілісність даних від апаратних та програмних помилок;

- мати систему відновлення, тобто бути здатною відновити базу даних до попереднього стану, перерваного апаратним або програмним збоєм.

- забезпечити розмежування прав для запобігання несанкціонованому доступу;

- контроль надмірності даних;

- підтримувати цілісність бази даних (коректність та узгодженість);

- підтримувати одночасну обробку даних декількома користувачами. [6,7]

На сьогоднішній день існують СУБД різних видів. Для порівняння розглянемо декілька найпоширеніших, а саме MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Microsoft Office Access, DB2.[13]

MySQL – це безкоштовна та відкрита реляційна СУБД, яка є найбільш популярною у світі. Вона використовується у найрізноманітніших сферах, включно з веб-розробкою, інтернет-магазинами, системами керування контентом і бізнес-аналітикою. Сумісна з більшістю популярних мов програмування, включно з PHP, Java, Python і C++.

PostgreSQL – це ще одна безкоштовна і відкрита реляційна СУБД, популярна альтернатива MySQL. Володіє ширшими можливостями, ніж MySQL, включно з підтримкою транзакцій, реплікації та географічних об’єктів. Підтримує широкий спектр функцій і розширень, які дають змогу адаптувати її під конкретні потреби користувачів.[17, 18]

Microsoft SQL Server розроблено корпорацією Майкрософт, є виключно сумісним з Windows. З моменту її інтеграції з Microsoft Azure його гнучкість та продуктивність покращилися, а також тепер дозволяє отримувати інформацію з інших серверів, підвищуючи його зручність користування. На відміну від усіх інших найвідоміших СУБД, MS SQL Server має низку суттєвих переваг, найважливішою з яких є можливість забезпечення клієнт-серверної архітектури для побудови інформаційних систем, що виступають в ролі сервера баз даних Однією з ключових особливостей Microsoft SQL Server є можливість управління цілісністю даних.

Однією з ключових особливостей Microsoft SQL Server є можливість управління цілісністю даних. Також можна сказати, що вона відповідає всім вимогам розподіленої обчислювальної системи. Ця СУБД підтримує реплікацію даних, паралельну обробку, створення та обробку великих баз даних на недорогих апаратних платформах.[13]

Oracle може працювати практично в будь-якій системі. Також слід відзначити, що є багато інструментів, орієнтованих на моніторинг та адміністрування Oracle.

Microsoft Office Access - система керування базами даних, розроблена компанією Microsoft. Вона підтримує різноманітні функції, включаючи створення запитів та інтеграцію із зовнішніми формами та іншими базами даних. Access дозволяє створювати додатки до баз даних завдяки інтеграції з мовою програмування VBA. Однак Access в основному підходить для невеликих проектів. Він не підтримує деякі базові механізми, такі як тригери, які необхідні для ефективної роботи з багатокористувацькими базами даних.

IBM DB2 друга найбільш часто використовувана база даних для Unix / Linux, Windows за популярністю іде після Oracle, є найкращим вибором для Mainframe (суперкомп'ютерів). DB2 має своїх послідовників, навчених своїм мистецтвам, але їхні ініціативи менші, ніж для Oracle.[7, 13]

Порівняльна таблиця (таблиця 1) найпоширеніших СУБД PostgreSQL, MySQL та Oracle.

Таблиця 1

Критерій	**PostgreSQL**	**MySQL**	**Oracle**
Тип ліцензії	Open source	Open source	Proprietary
Вартість	Безкоштовно	Безкоштовно	Платна
Відкритість	Відкритий вихідний код	Відкритий вихідний код	Закритий вихідний код
Підтримувані мови програмування	SQL, PL/pgSQL, Python, Perl, Tcl, Java, C, C++, Ruby	SQL, MySQL, PHP, Python, Perl, Tcl	SQL, PL/SQL, Java, C, C++, C#, Python, Ruby
Підтримувані операційні системи	Windows, Linux, macOS, FreeBSD	Windows, Linux, macOS	Windows, Linux, macOS, Solaris, HP-UX, AIX, z/OS
Підтримувані архітектури	x86, x86-64, ARM, RISC-V	x86, x86-64, ARM	x86, x86-64, ARM, RISC-V
Продуктивність	Хороша	Хороша	Відмінна
Надійність	Висока	Висока	Висока
Безпека	Висока	Хороша	Відмінна
Функціональність	Широкий набір функцій	Широкий набір функцій	Широкий набір функцій
Масштабованість	Хороша	Хороша	Відмінна
Гнучкість	Висока	Висока	Висока
Підтримка	Активна	Активна	Активна

[18]

2.3 Створення бази даних

Для проектування бази даних, була обрана СУБД PostgreSQL. Це потужна об’єктно - реляційна СУБД, яка забезпечує високу продуктивність, надійність та маштабованість. PostgreSQL активно використовується у великих комерційних проектах, може підтримувати складні SQL-запити, транзакції, збережені процедури, а також роботу з просторовими даними, що є корисним у випадку візуалізації маршрутів доставки.

Загалом PostgreSQL має багато функцій, призначених для того, щоб допомогти розробникам створювати програми, адміністраторам захищати цілісність даних , а також керувати даними незалежно від розміру набору даних. Окрім того, що PostgreSQL є безкоштовним і має відкритий вихідний код , він теж має високу розширюваність. Наприклад, є можливість визначати власні типи даних, створювати власні функції та навіть писати код з різних мов програмування без перекомпіляції бази даних.[17]

Серед основних переваг PostgreSQL можна виокремити:

- підтримка розширень (наприклад, PostGIS для геоданих);
- відповідність стандарту SQL;
- масштабованість і висока стабільність у продакшн-середовищах;
- активна спільнота та регулярні оновлення безкоштовно.

Для зручного розгортання середовища розробки було використано Docker. Використання PostgreSQL з Docker має низку переваг. По-перше, це дає змогу легко і швидко розгортати СУБД у будь-якому оточенні. По-друге,

контейнеризація забезпечує високий ступінь ізоляції та безпеки даних. По-третє, Docker дає змогу легко масштабувати PostgreSQL відповідно до потреб програми.[26]

У середовищі Docker створено два основних контейнери:

- ferma_1.0 — контейнер з базою даних PostgreSQL, який працює на порту 5432;
- ferma_admin — контейнер з інтерфейсом pgAdmin для візуального керування базою даних, доступний на порту 5050.

На зображенні (рис. 5) представлено інтерфейс Docker Desktop, в якому видно, що обидва контейнери активні та працюють стабільно. Запуск контейнерів здійснюється за допомогою відповідного файлу - docker-compose.yml або вручну через командний рядок. Це дозволяє швидко створити та налаштувати середовище бази даних без складних встановлень. Після створення бази даних для подальшого запуску достатньо просто запустити контейнер у Docker Desktop.

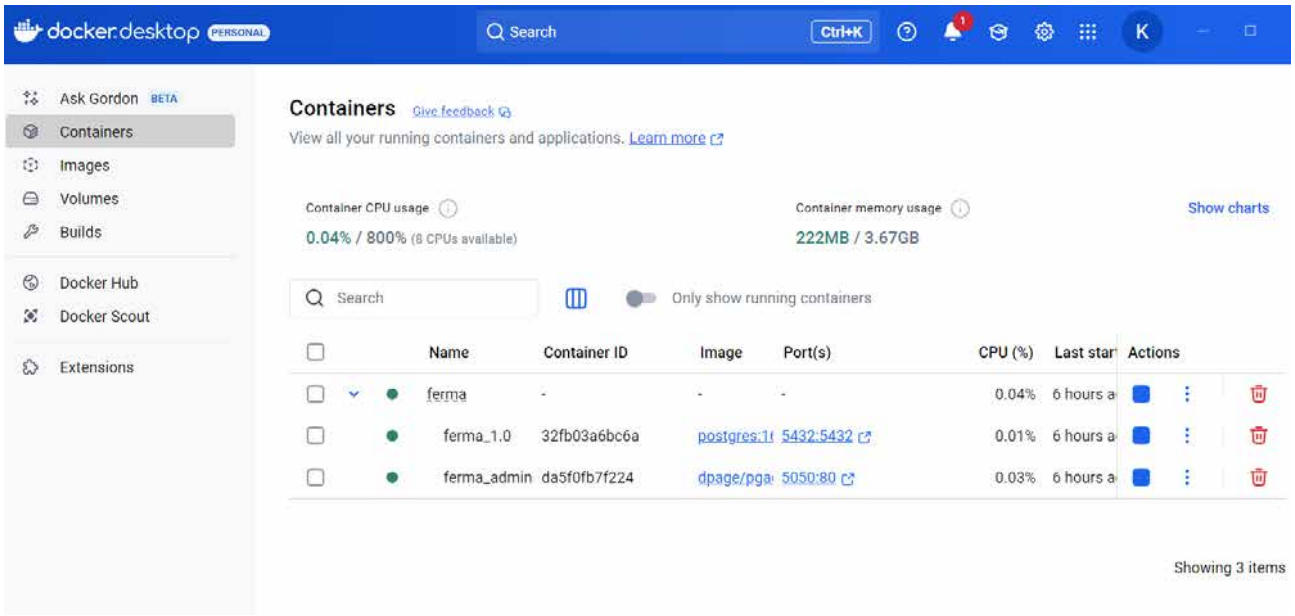


Рис. 5 Запущений контейнер ferma у Docker Desktop

У базі даних системи управління ланцюгом постачання сільськогосподарської продукції була створена таблиця accounts_user, яка є ключовою для зберігання інформації про користувачів. Вона містить повний

набір полів, необхідних для ідентифікації, авторизації та керування доступом до функціоналу системи як для фермерів, так і для замовників. Нище наведений код для створення цієї таблиці.

```
-- Table: public.accounts_user

-- DROP TABLE IF EXISTS public.accounts_user;

CREATE TABLE IF NOT EXISTS public.accounts_user
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    password character varying(128) COLLATE pg_catalog."default" NOT NULL,
    last_login timestamp with time zone,
    is_superuser boolean NOT NULL,
    username character varying(150) COLLATE pg_catalog."default" NOT NULL,
    first_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    last_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    email character varying(254) COLLATE pg_catalog."default" NOT NULL,
    is_staff boolean NOT NULL,
    is_active boolean NOT NULL,
    date_joined timestamp with time zone NOT NULL,
    role character varying(10) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT accounts_user_pkey PRIMARY KEY (id),
    CONSTRAINT accounts_user_username_key UNIQUE (username)
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.accounts_user

    OWNER to "Admin";

-- Index: accounts_user_username_6088629e_like

-- DROP INDEX IF EXISTS public.accounts_user_username_6088629e_like;

CREATE INDEX IF NOT EXISTS accounts_user_username_6088629e_like

    ON public.accounts_user USING btree

    (username COLLATE pg_catalog."default" varchar_pattern_ops ASC NULLS LAST)

    TABLESPACE pg_default;
```

Кожен запис у таблиці має унікальний ідентифікатор id, який автоматично генерується. Таблиця зберігає пароль користувача (але він хешований, що і адміністратор системи не має змоги побачити пароль), його логін (username), ім’я та прізвище, електронну адресу, а також дані про роль у системі (role) — наприклад, фермер або клієнт. Крім того, для адміністративних цілей використовуються поля is_superuser, is_staff та is_active, які допомагають відрізняти типи доступу користувачів і дозволяють блокувати акаунти за потреби. Також у таблиці фіксується час останнього входу (last_login) та дата реєстрації (date_joined).

Для забезпечення унікальності логінів встановлене обмеження UNIQUE на поле username, а також створений індекс для прискорення пошуку, зокрема під час виконання запитів із шаблонами (LIKE). Первинним ключем є поле id, що забезпечує ідентифікацію кожного користувача.

На рисунку (рис.6) зображно користувачів, які наразі зареєстровані в системі, дані про них, їх ролі, а також, дати реєстрації та останнього входу.

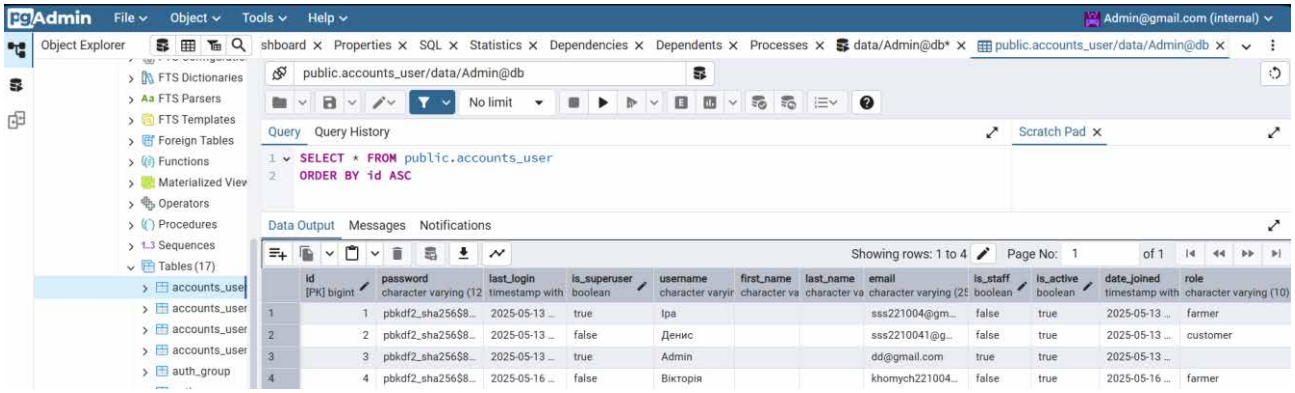


Рис. 6 Таблиця для зберігання користувачів в PostgreSQL

3 РОЗРОБКА ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЛАНЦЮГА ПОСТАЧАННЯ ПРОДУКЦІЇ

3.1 Організаційна структура програмного забезпечення

Інформаційна система управління ланцюгом постачання сільськогосподарської продукції розроблена з урахуванням принципів модульності, масштабованості та зручності розгортання. Структура програмного забезпечення передбачає розподілення на окремі логічні компоненти, кожен з яких виконує чітко визначену функцію. Такий підхід дозволяє спростити процес підтримки та розширення системи.

Загалом програмне забезпечення побудоване за класичною архітектурою «клієнт-сервер», де клієнтська частина (інтерфейс користувача) реалізована у вигляді веб-додатку, доступного через браузер, а серверна частина відповідає за обробку бізнес-логіки, управління даними та взаємодію з базою даних.

Основні компоненти системи:

- Клієнтський інтерфейс (front-end) - забезпечує взаємодію користувача з системою. Використовує вбудовані HTML-шаблони Django для відображення контенту.
- Серверна логіка (backend) - реалізована на мові Python з використанням фреймворку Django, що забезпечує високий рівень безпеки, структури коду та швидкості розробки.
- База даних реалізована на PostgreSQL, що забезпечує надійне структуроване зберігання даних, фільтрацію та запити.

- Система контейнеризації Docker використовується для спрощення розгортання та виконання додатків у різних середовищах, що полегшує міграцію та масштабування системи.
- Система контролю версій Git - використовується для контролю змін коду.

Організаційна структура програмного забезпечення дозволяє легко адаптувати систему до змін вимог, інтегрувати нові функції (наприклад, платіжні модулі або облік залишків на складі) та підтримувати її в довгостроковій перспективі.

На рисунку представлено діаграму пакетів(рис. 7), що відображає загальну структуру інформаційної системи. Система складається з трьох основних компонентів: клієнтської частини, бекенду та бази даних. Кожен з них містить підсистеми та модулі, що виконують окремі функції. Клієнт взаємодіє з користувачем через графічний інтерфейс і відображає повідомлення. Бекенд обробляє запити, відповідає за безпеку, логування та бізнес-логіку. База даних зберігає всі необхідні дані та обробляє запити за допомогою уявлень, тригерів та структурованих запитів.

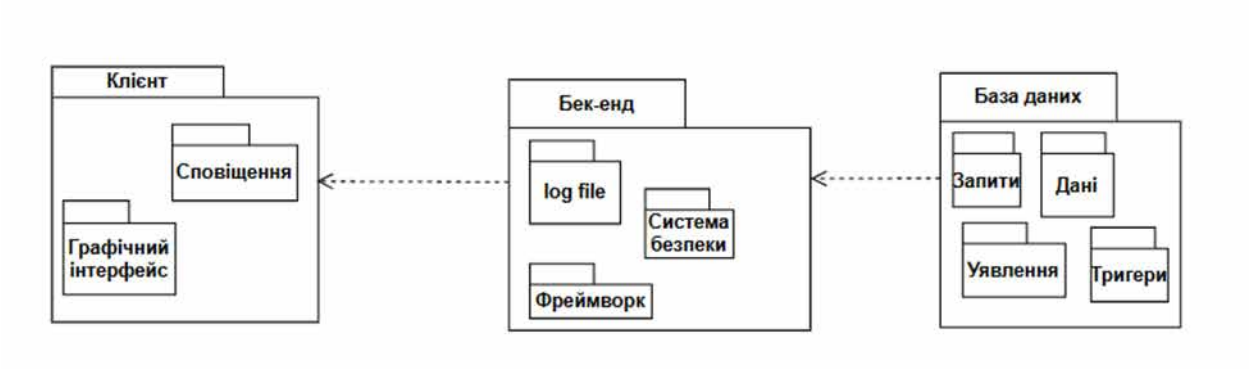


Рис. 7 Діаграма пакетів

Прикладне програмне забезпечення для інформаційної системи реалізоване мовою програмування Python з використанням фреймворку Django.

Воно має структуру, яка організаційно чітко визначена, також побудована за принципом модульності. Така структура забезпечує логічне розділення функціональних компонентів, спрощує розробку, тестування та подальше розширення системи. Детальніше про модульність системи буде описано в наступному розділі.

Структура Django-проєкту з декількома додатками (apps), які відповідають за окремі функції системи наступні:

- accounts - модуль для реєстрації, авторизації, керування ролями користувачів (фермер, замовник, адміністратор).
- agromarket - центральний додаток, який керує бізнес-логікою взаємодії між користувачами, товарами та замовленнями.
- maps - модуль для роботи з геолокаційними даними та побудови маршрутів доставки.
- orders - функціонал створення, перегляду та керування замовленнями.
- products – модуль для управління товарами, які фермери публікують для продажу.

Також у проєкті використовується система шаблонів Django, які розміщені в директорії templates. Вона містить окремі HTML-файли для візуального представлення:

- base.html — основний шаблон, від якого наслідуються інші сторінки.
- farmer_purchases.html — інтерфейс перегляду замовлень для фермера.
- maps.html — сторінка з візуалізацією логістичних маршрутів.
- notifications.html — сторінка для повідомлень користувачів.

Файл docker-compose.yaml відповідає за контейнеризацію застосунку, що спрощує розгортання в будь-якому середовищі. Наявність файлу requirements.txt дозволяє швидко встановити всі залежності проєкту за допомогою pip.

Така структура забезпечує ефективну взаємодію між модулями через внутрішні API, дотримуючись принципів розділення відповідальностей (Separation of Concerns) та дозволяє масштабувати систему залежно від потреб користувачів.

3.2 Вибір інструментарію для створення ППЗ

У процесі розробки інформаційної системи, як було сказано вище використовувалися такі основні інструменти:

- Мова програмування: Python
- Фреймворк: Django
- Система контейнеризації: Docker
- Середовище розробки: Visual Studio Code

Розглянемо детальніше опис кожного інструменту, а також аргументацію такого вибору.

Мову Python можна охарактеризувати як: інтерпретовану високорівневу мову програмування загального призначення. Її основні особливості:

- Проста і мінімалістична мова. Читання хорошої програми на Python дуже нагадує читання англійського тексту, хоча і досить строгого. Така псевдокодова природа Python є однією з його найсильніших сторін. Python надає можливість програмісту зосередитися на розв’язуванні задачі, а не на самій мові. Ядро мови без додаткових модулів досить просте і мінімальне - це надає мові простоти та логічності, що відповідно спрощує її вивчення.
- Мова легка в освоєнні та простота для вивчення. Мовою Python надзвичайно легко почати програмувати. Python має виключно простий синтаксис. Досить часто в закладах освіти обирають Python як першу мову програмування для своїх студентів.

- Вільна і відкрита мова. Python - це приклад вільного і відкритого програмного забезпечення – FLOSS (Free / Libré and Open Source Sofware). Користувач має право вільно використовувати копії цього програмного забезпечення, переглядати його вихідні тексти, вносити зміни, а також використовувати його частини в своїх програмах.
- Мова загального призначення. Мова використовується для розв’язування задач досить широкого спектру.
- Кросплатформенна мова. Завдяки своїй відкритій природі, Python зроблений таким чином, щоб можна було працювати на різних пристроях та різних операційних системах. Один і той самий код буде запускатися на різних платформах без будь-яких змін за умови, що в коді відсутні системнозалежні функції. Python можна використовувати в GNU / Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE і навіть на PocketPC.[28]
- Мультипарадигменна мова. В основі Python лежить декілька парадигм програмування: об’єктноорієнтовна, імперативна, функціональна (функційна), структурна. Проте досить часто про Python говорять зокрема як про об’єктно-орієнтовну мову програмування. В об'єктно-орієнтованих мовах програмування програми будуються на основі об'єктів, які об'єднують в собі дані і функціонал.

Саме тому для програмування була вибрана мова Python, через свою простоту та універсальність.

Django — це веб-фреймворк з відкритим кодом, написаний мовою Python, розроблений для стимулювання швидкої розробки та практичного дизайну . Django використовує шаблон проектування програмного забезпечення Model-View-Template (MVT), ефективно розділяючи бізнес-логіку, дані та шари

презентації. Він надає об'єктно-реляційний маппер, який автоматично перетворює класи та об'єкти на таблиці та рядки в базі даних.

Фреймворк Django складається з кількох компонентів, включаючи моделі, представлення, шаблони, форми та інтерфейс адміністратора. Моделі використовуються для визначення структур даних, представлення обробляють запити та повертають відповіді, шаблони визначають структуру сторінок, форми використовуються для збору та перевірки даних, а інтерфейс адміністратора дозволяє швидко виконувати операції CRUD (створення, читання, оновлення, видалення) над моделями.[30]

Якщо говорити про переваги то фреймворк Django пропонує численні унікальні переваги порівняно з іншими веб-фреймворками. По-перше, він вирізняється швидкою розробкою, дотримуючись принципу «менше коду, більше конфігурації», надаючи безліч готових компонентів, які значно підвищують ефективність розробки. По-друге, він надає пріоритет безпеці та надійності, оскільки Django постачається з вбудованими функціями безпеки для запобігання SQL-ін'єкціям, атакам міжсайтового скриптингу та всім іншим вразливостям безпеки, забезпечуючи безпечну та стабільну роботу програм [25]. Django може похвалитися потужним рівнем абстракції бази даних та об'єктно-реляційним маппером, що спрощує операції з базою даних та пропонує узгоджений API для різних баз даних. Завдяки своїй розгалуженій екосистемі, багатій колекції сторонніх компонентів та активній спільноті розробників, Django робить вирішення проблем зручним. Таким чином, ці переваги роблять цей фреймворк найкращим вибором для створення різноманітних веб-застосунків.

Docker, платформа контейнеризації з відкритим вихідним кодом, надійне рішення для вирішення проблем, таких як- велика кількість помилок та довготривале розгортання при використанні традиційних методів, шляхом інкапсуляції додатків та їх залежностей у легкі контейнери.

Docker надає фреймворк, який дозволяє упаковувати додатки та залежності в контейнери, щоб забезпечити портативність, ізоляцію та ефективне використання ресурсів. Контейнери гарантують, що додатки працюють узгоджено в різних середовищах. [24]

Проблеми розгортання веб-додатків. Традиційні методи розгортання часто вимагають складних конфігурацій і ручних налаштувань, що призводить до непослідовного розгортання, конфліктів версій і проблем з масштабуванням. Ці проблеми призвели до необхідності спрощеного та автоматизованого підходу.

Використання Docker для оптимізації розгортання та запуску веб-додатків має великий потенціал, оскільки саме за його алгоритмом вони стають більш гнучкими, масштабованими та стійкими. [26]

У своєму проєкті я використала Docker, тому що це зручно і швидко. За допомогою Docker я змогла створити окреме середовище для роботи з базою даних PostgreSQL, не встановлюючи її напряму на комп’ютер. Це дозволило уникнути конфліктів з іншими програмами, швидко налаштувати систему та легко переносити її на інші комп’ютери.

Docker допомагає запускати базу даних у вигляді контейнера — тобто в ізольованому середовищі, яке працює однаково всюди. Якщо потрібно, можна швидко запустити чи видалити контейнер, змінити налаштування або перенести систему на інший сервер без складних переналаштувань.

Середовищем для розробки було обрано Visual Studio Code (VS Code). Visual Studio Code - це легкий, але потужний текстовий редактор, розроблений компанією Microsoft. Він надає розширені можливості для редагування коду та роботи з проєктами, включаючи підтримку різних мов програмування, автоматичне доповнення коду, інтеграцію з системами керування версіями, вбудовану підтримку відладки та широкий вибір розширень для розширення функціональності за потреби користувача [32].

Visual Studio Code - це редактор вихідного коду, який ідеально підходить для повсякденного використання. VS Code підтримує сотні мов і допомагає досягти миттєвої продуктивності завдяки підсвічуванню синтаксису, узгодженню дужок, автоматичному скороченню, виділенню блоків, фрагментів коду та іншим функціям. Інтуїтивно зрозумілі комбінації клавіш, легка настройка та розроблені спільнотою ярлики полегшують навігацію по коду.[27]

Коли код стає складним, налагодження є найпотужнішою функцією. Налагодження часто є функцією, якої розробникам найбільше бракує в процесі базового кодування, тому ми робимо її можливою. Visual Studio Code містить інтерактивний налагоджувач, який дозволяє переглядати сирий код, змінні, стеки викликів та виконувати команди в консолі.

VS Code також інтегрує інструменти збірки та написання сценаріїв, які допоможуть вам виконувати типові завдання та пришвидшити ваш щоденний робочий процес, а також підтримує Git, щоб ви могли використовувати систему контролю версій, включаючи перегляд стадій, не виходячи з редактора.[27]

3.3 Алгоритмізація та програмування програмних модулів

Розробка інформаційної системи управління ланцюгом постачання сільськогосподарської продукції включає створення окремих програмних модулів, кожен з яких виконує спеціальну функцію. Для того аби була ефективна реалізація функціональності було проведено алгоритмізацію основних процесів, що дозволило структурувати логіку системи та забезпечити коректну взаємодію між компонентами.

Основні реалізовані модулі та їх логіка:

- Модуль реєстрації та авторизації користувачів.

Забезпечує створення облікових записів фермерів і замовників, захист паролів (через хешування) та перевірку прав доступу. Алгоритм передбачає перевірку унікальності логіну та автоматичне визначення ролі користувача в системі.

- Модуль управління товарами.

Фермер може додавати продукти із зазначенням їхніх характеристик (назва, вага, кількість тощо). Система перевіряє коректність даних перед збереженням до бази даних.

- Модуль перегляду та замовлення товарів.

Замовник має змогу переглядати список товарів, фільтрувати їх за різними критеріями та оформлювати замовлення, зазначаючи свої контактні дані та адресу доставки.

- Модуль перегляду замовлень фермером.

Алгоритм забезпечує відображення лише актуальних замовлень, що відповідають доступним товарам фермера.

- Модуль розрахунку вартості доставки.

Система автоматично обчислює вартість доставки на основі відстані між містом замовника та фермером. Для цього використовується таблична залежність тарифів від кілометражу.

- Модуль відображення маршруту доставки.

Фермер отримує інформацію про місце доставки кожного замовлення, що дозволяє оптимізувати логістику.

Оскільки, програмна реалізація здійснюється за допомогою фреймворку Django, це спрощує створення моделей, форм, представлень та маршрутизації. Основні сутності системи описані у вигляді Django-моделей, які зберігаються у базі даних PostgreSQL. Всі модулі взаємодіють між собою через внутрішні API, а дані передаються через HTTP-запити з використанням стандартної архітектури MVC.[31]

Детальніше розберемо модулі програмного забезпечення, з наведенням відповідних фрагментів коду:

- Модуль моделей (models.py)

Моделі є основою проєкту, оскільки саме вони описують структуру таблиць у базі даних. Завдяки Django ORM (Object-Relational Mapping) ми можемо працювати з базою даних як зі звичайними Python-об’єктами. Це дозволяє не писати SQL-запити вручну, а використовувати прості методи та властивості.[30]

Наприклад, модель Product може містити поля для назви товару, опису, ціни та посилання на користувача, який є власником продукту. Нище наведений фрагмент коду з файлу models.py.

```
class Product(models.Model):

    CATEGORY_CHOICES = [

        ('vegetables', 'Овочі'),
```

```
        ('fruits', 'Фрукти'),
        ('grain', 'Зернові'),
        ('fertilizers', 'Добрива'),
        ('seeds', 'Насіння'),
        ('household', 'Господарчі товари'),
        ('other', 'Інше'),
    ]
    UNIT_CHOICES = [
        ('pcs', 'шт'),
        ('kg', 'кг'),
        ('t', 'т'),
        ('l', 'л')
    ]
```

- Представлення логіки (views.py)

Файл views.py містить функції або класи, які обробляють запити користувачів і визначають, яку відповідь система повинна надати. Саме тут реалізована основна логіка взаємодії між користувачем і базою даних.

Наприклад, коли користувач відкриває сторінку з усіма доступними товарами — саме функція у views.py відповідає за отримання списку товарів із бази даних і передачу їх у шаблон для відображення.

Фрагмент коду: перегляд списку продуктів з файлу views.py

```
@login_required
def product_list(request):
    """Сторінка з усіма товарами"""
```

```
    search_query = request.GET.get('search', '')
    category_filter = request.GET.get('category', '')
    products = Product.objects.all().order_by('id')
    if search_query:
        products = products.filter(name__icontains=search_query)
    if category_filter:
        products = products.filter(category=category_filter)
    paginator = Paginator(products, 15)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
    context = {
        'page_obj': page_obj,
        'categories': Product.CATEGORY_CHOICES,
    }
    return render(request, 'products/product_list.html', context)
```

- Форми введення (forms.py)

Щоб реалізувати обробку введення користувачів, Django надає спеціальний модуль forms.py, у якому можна створити форми для реєстрації, входу в систему, додавання нових продуктів тощо. Django-форми автоматично створюють HTML-поля і перевіряють правильність введених даних.

Фрагмент коду: форма реєстрації або додавання продукту з файлу forms.py

```
class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = ['name', 'category', 'price', 'unit', 'description', 'image', 'address']
        labels = {
            'name': 'Назва товару',
```

```
    'category': 'Категорія',
    'price': 'Ціна',
    'unit': 'Одиниця виміру',
    'description': 'Опис',
    'image': 'Зображення',
    'address': 'Адреса',
}
```

Допоміжні модулі Django-проєкту

- Модуль маршрутизації (urls.py). Цей файл відповідає за маршрутизацію запитів користувачів до відповідних функцій або класів у views. За допомогою urls.py задаються шляхи (URL), за якими доступні різні сторінки вебзастосунку. Це дозволяє зручно структурувати навігацію в системі, створювати читабельні й логічні адреси. У кожному додатку проекту може бути свій urls.py, який потім підключається до головного файлу маршрутизації проєкту.
- Модуль адміністрування (admin.py). Файл admin.py дає змогу підключити створені моделі до вбудованої адміністративної панелі Django. Це дозволяє адміністраторам зручно переглядати, редагувати або видаляти дані через графічний інтерфейс. Розширення цього модуля також дозволяє налаштовувати відображення даних, фільтри, пошук тощо.
- Шаблони HTML (templates). У каталозі templates зберігаються HTML-шаблони, які використовуються для відображення даних на стороні клієнта. Django використовує власну шаблонну мову (Django Template Language), яка дозволяє виводити змінні, створювати цикли, умови та підключати інші шаблони. Ця система забезпечує розділення логіки (views) і представлення (templates), що сприяє чистоті та зручності в підтримці коду.[29]

- Налаштування проєкту (settings.py). Файл settings.py містить конфігураційні параметри всього вебзастосунку. Тут визначаються підключення до бази даних, встановлені додатки (INSTALLED_APPS), параметри безпеки, шляхи до медіафайлів і шаблонів, мова інтерфейсу, часова зона, інтеграція з іншими сервісами тощо. Цей файл є обов’язковим елементом проєкту Django.
- Файл запуску (manage.py). Цей файл слугує точкою входу до командної оболонки Django. З його допомогою виконуються команди для міграції бази даних, запуску локального сервера, створення додатків тощо. Він значно спрощує взаємодію розробника з проєктом на всіх етапах розробки.
- Інші службові модулі

init.py: маркує каталоги як пакети Python.

apps.py: містить конфігураційний клас додатка, що дозволяє налаштувати поведінку окремих компонентів.

asgi.py та wsgi.py: файли, що забезпечують запуск проєкту через сервери ASGI або WSGI. Вони використовуються при розгортанні вебзастосунку.[28]

Загальний вигляд структури описаних вище модулів в поданні Visual Studio Code(рис.8)., а також діаграма розгортання (рис. 9) ,та розгортання на основі діаграми компонентів(рис.10)

Рис. 8 Структура каталогів і файлів програмної реалізації інформаційної системи

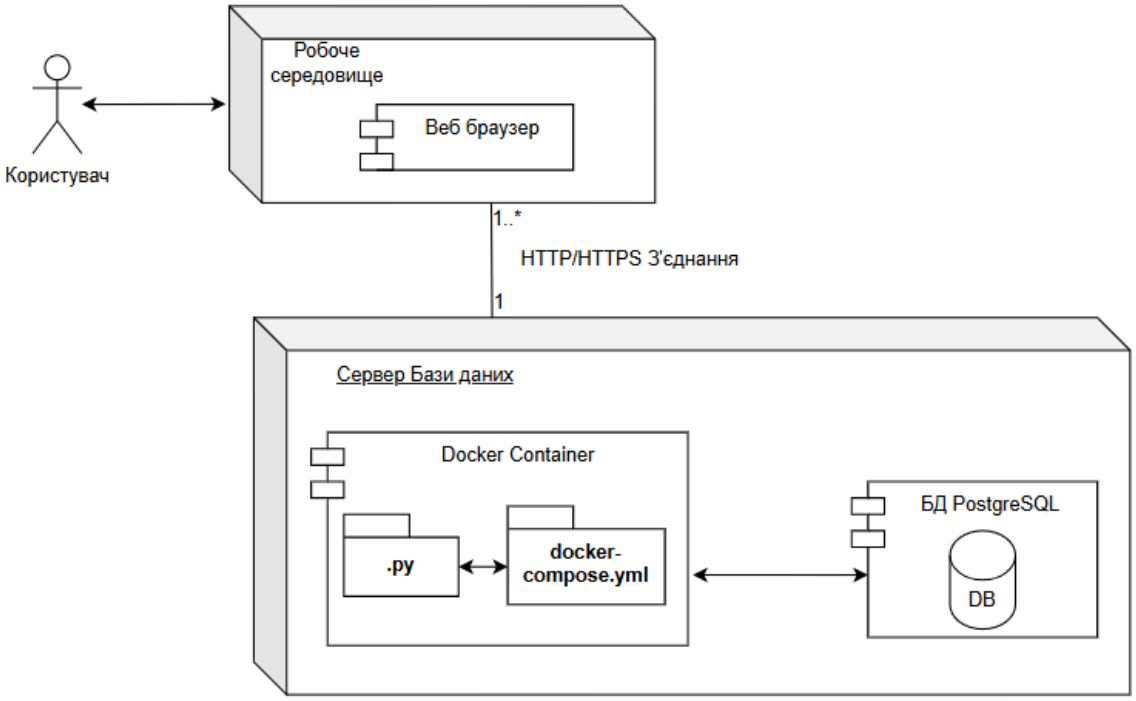


Рис. 9 Діаграма розгортання

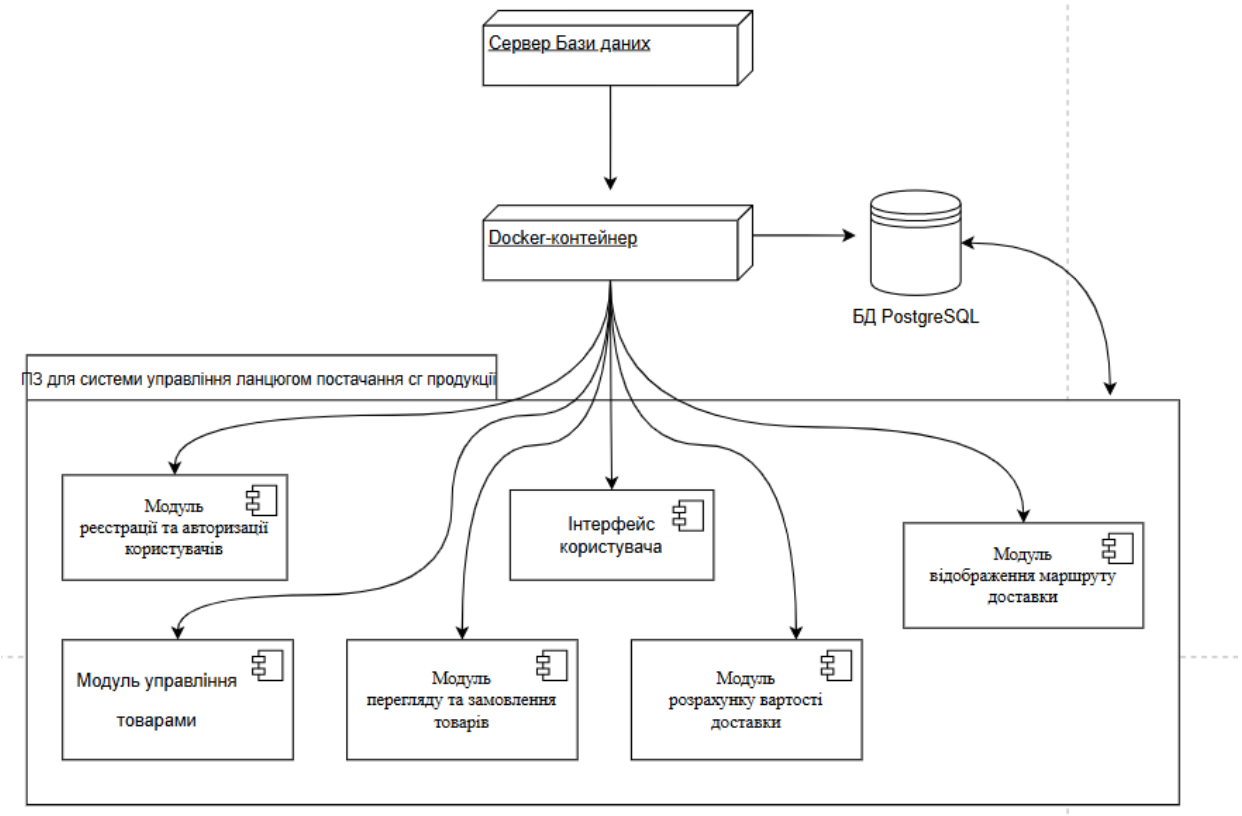


Рис.10 Діаграма розгортання на основі компонентів(модулів)

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ У СФЕРІ АГРАРНИХ ПОСТАЧАНЬ

4.1 Тестування системи

Існує декілька визначень, пов’язаних із тестуванням. З одного боку, тестування це процес дослідження та перевірки програмного продукту. Метою тестування є показати замовникам і розробникам, чи відповідає програма вимогам, а також виявити ситуації, коли поведінка програми є неправильною, небажаною або не відповідає специфікації. З іншого боку, тестування — це перевірка відповідності між фактичною та очікуваною поведінкою програми.

Мета тестування полягає у наступному:

- підвищити ймовірність того, що інформаційна система, призначена для тестування, буде працювати правильно при будь-яких обставинах;
- підвищити ймовірність того, що інформаційна система, призначена для тестування, буде відповідати всім описаним вимогам;
- надати актуальну інформацію про стан продукту (інформаційної системи) на даний момент.[34]

Було обрано ручноме тестування (manualtesting) , це коли тестувальники вручну виконують тести, не використовуючи ніяких засобів автоматизації. Ручне тестування – це самий низькорівневий і простий тип тестування, який не потребує великої кількості додаткових знань. Таке тестування вимагає значних зусиль, але без нього ми не зможемо переконатися в тому, чи можлива автоматизація в принципі. Один з фундаментальних принципів тестування

говорить: 100% автоматизація неможлива. Тому, ручне тестування - необхідність. [34]

Переходивши до тестування інформаційної системи управління ланцюгом постачання сільськогосподарської продукції слід уточнити, що основними користувачами є фермер та замовник, ціль фермера продати товар, замовника-придбати. Система значно спрощує основні кроки до слідування цілей користувачів. Нижче буде наведено короткий опис та рисунки основних функцій системи.

На початку роботи з інформаційною системою управління ланцюгом постачання сільськогосподарської продукції перед користувачем відкривається вікно авторизації(рис.11), або якщо це новий клієнт обравши «Ще не зареєстрований?» відкриється вікно реєстрації, яке представлено на рис.12. При реєстрації користувачеві пропонується ввести ім’я , електронну пошту та пароль а також є можливість обрати роль(фермер або покупець).

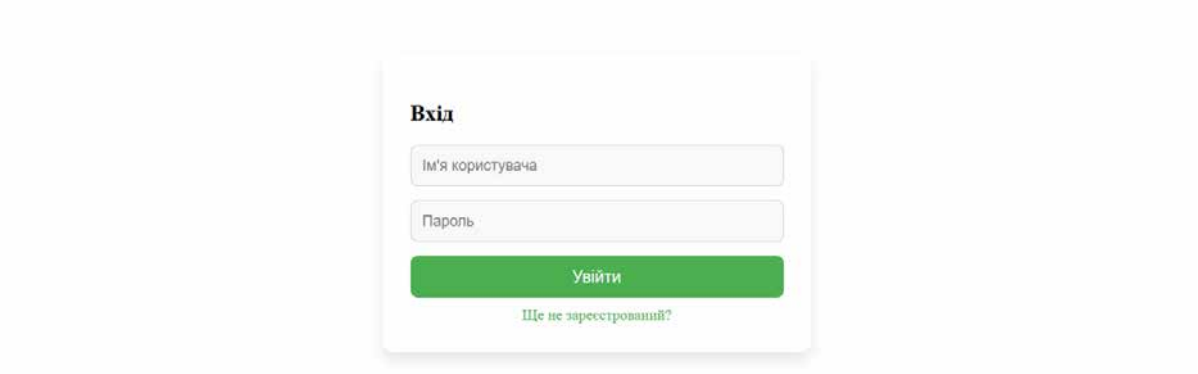
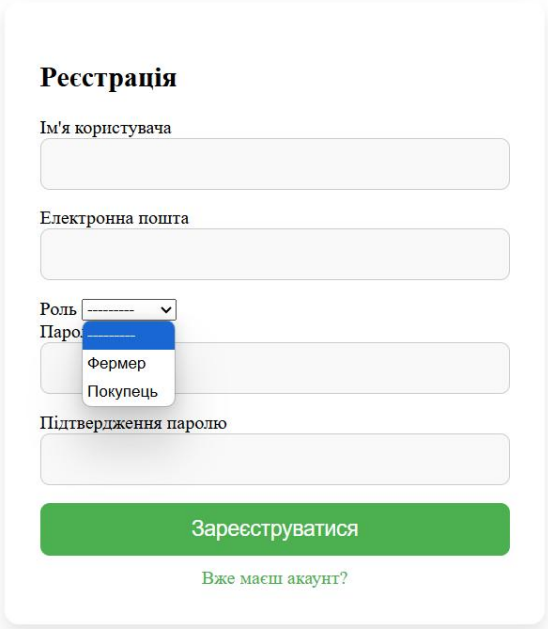


Рис. 11 Сторінка авторизації користувача



- Рис. 12 Реєстрація користувача

Коли користувач зареєструвався під роллю Покупець він має можливості обрати і замовити товар, а також заповнити дані про себе тобто профіль користувача(рис.13)

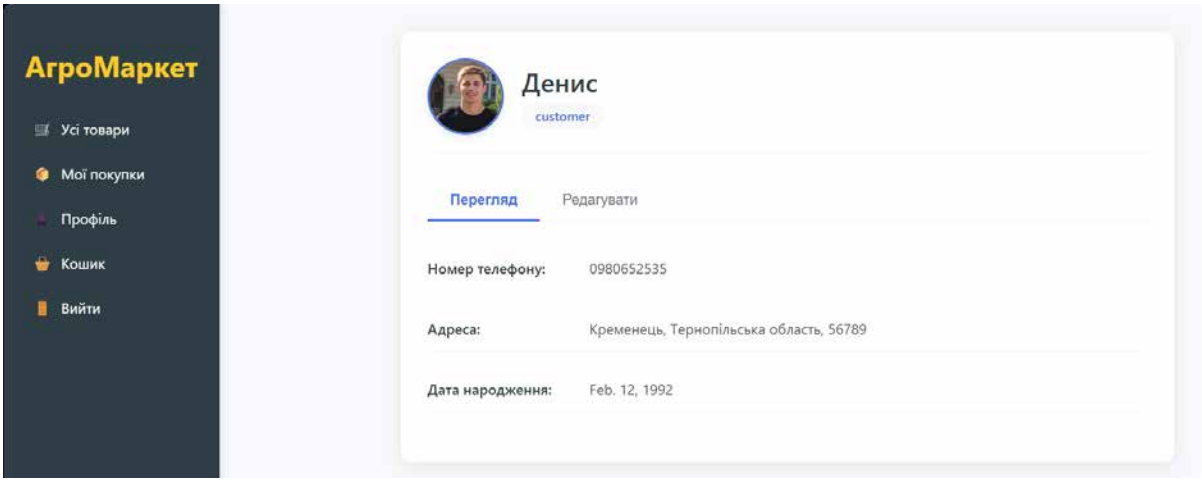


Рис.13 Профіль покупця

Коли користувач зареєструвався під роллю Фермер він має можливості додати і теж замовити товар, але у іншого фермера, також бачить дані про замовлення. При авторизації Фермер бачить вітальну сторінку, на ній немає окремих функцій, вона виключно інформативна(рис.14)

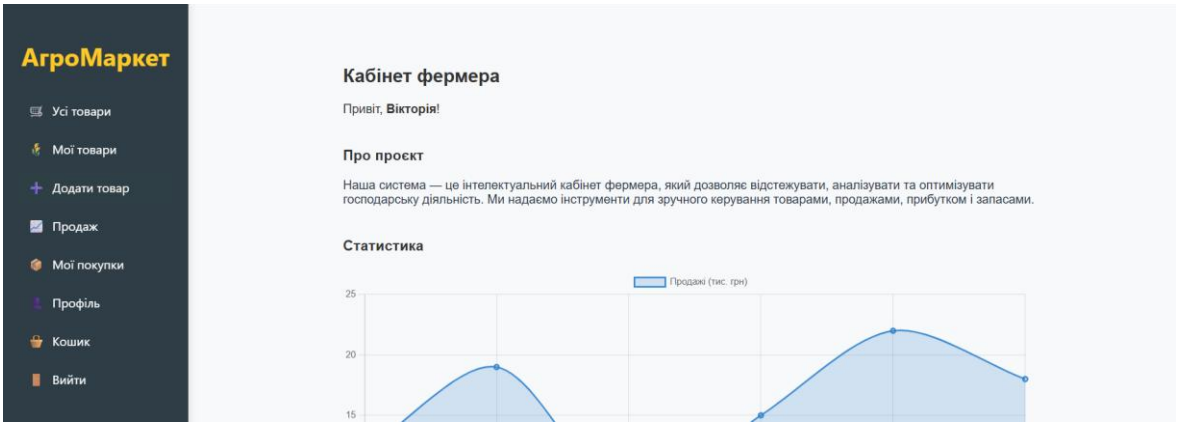


Рис.14 Сторінка, яку бачить фермер при авторизації в системі

Однією з основних функцій системи є можливість додавання товару Фермером, він може вказати назву, обрати категорію зі списку, вказати ціну,

обрати одиниці виміру товару зі списку, додати опис , фото , та вказати адресу(дана адреса буде слугувати місцем відправлення замовлення) (рис.15-16)

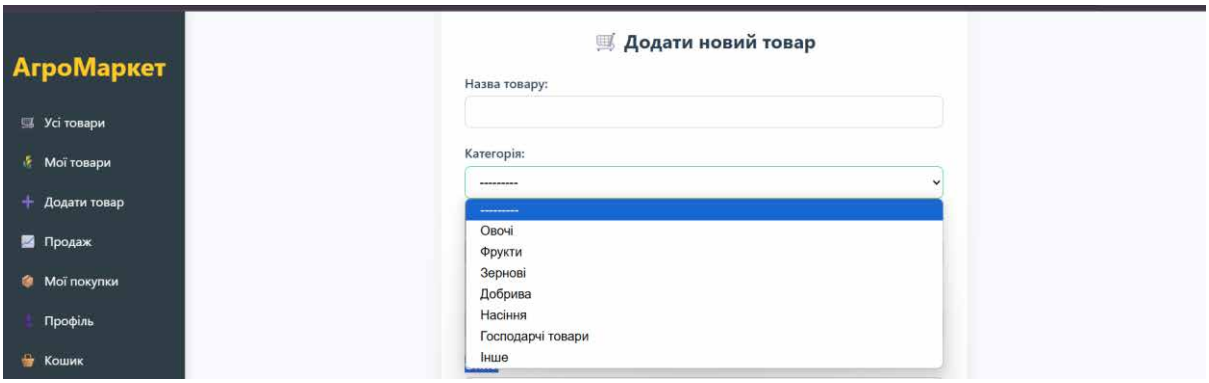


Рис.15 Форма додавання товару фермером

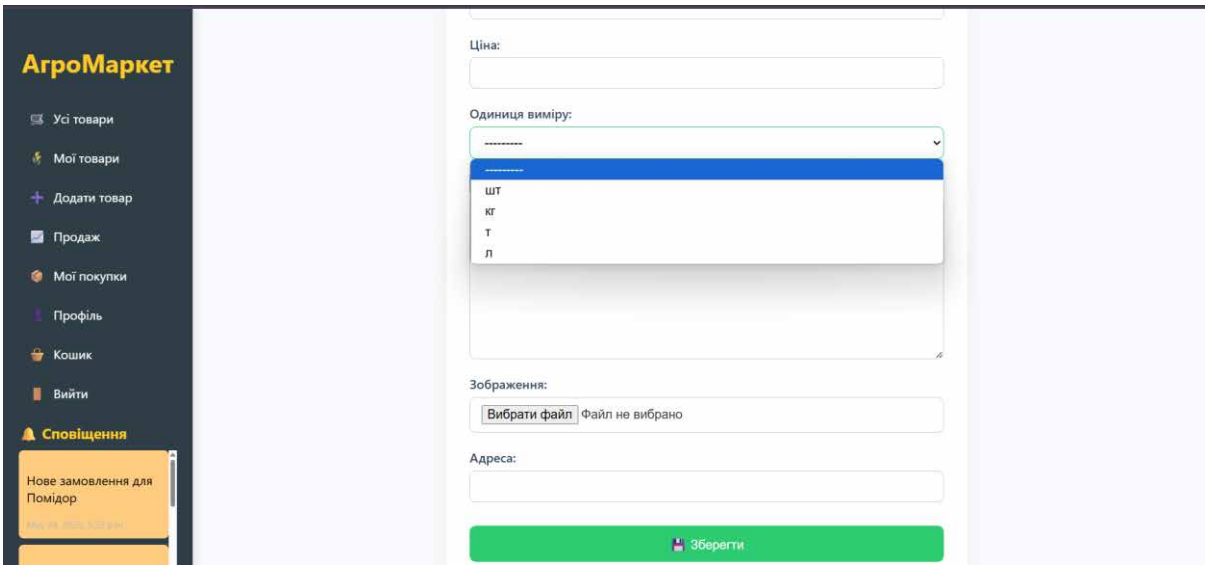


Рис.16 Форма додавання товару фермером

Після додавання товару Фермер може переглянути товари у вкадці Мої товари(рис.17)

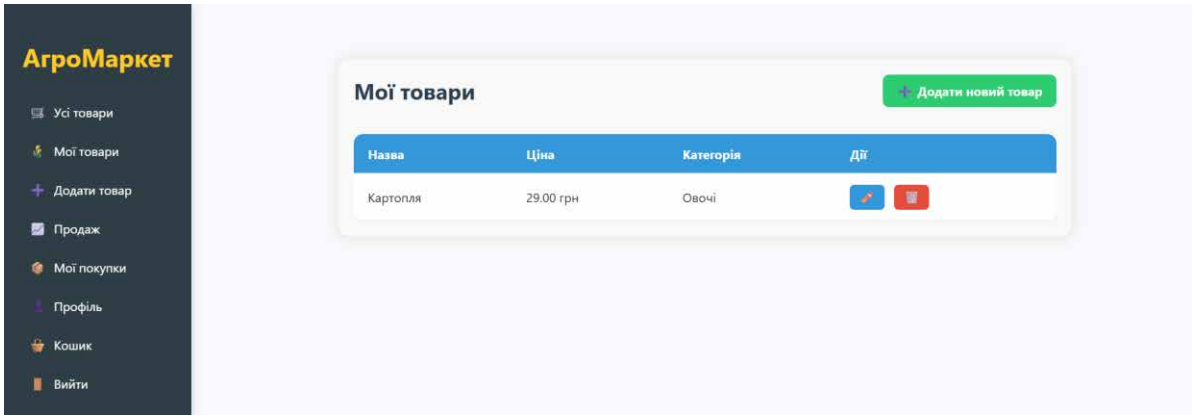


Рис.17 Відображення доданого товару

Фермер може видалити або редагувати товар відповідно до потреби. На рисунку (рис.18) зображено сторінку з описом товару, аналогічну бачить покупець , лише без кнопок Редагувати та Видалити

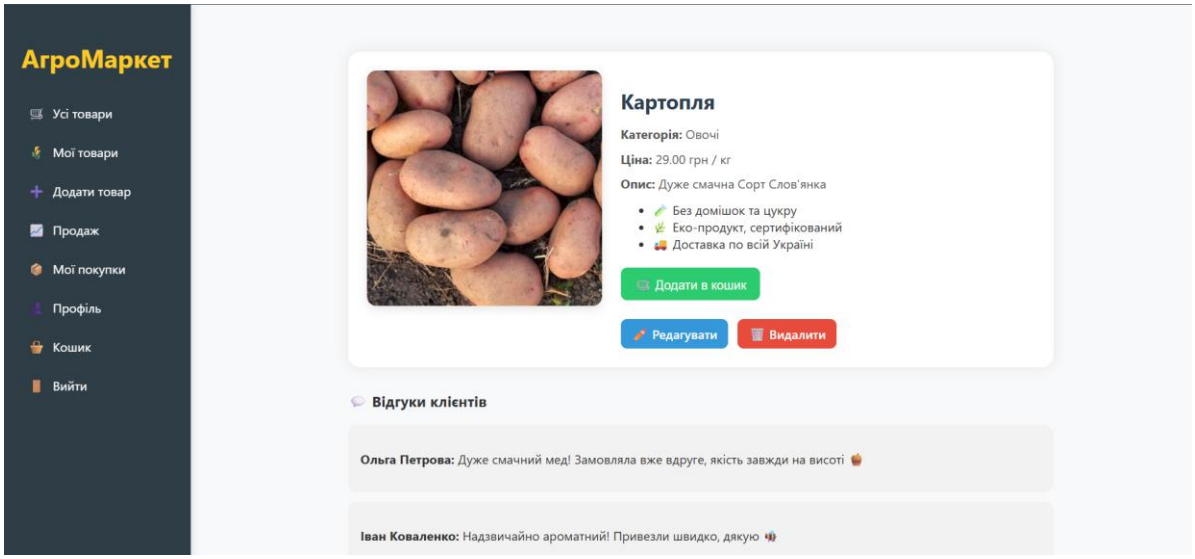


Рис.18 Опис товару при відкритті деталей

Обравши товар користувач в кошику бачить його і може змінити кількість одиниць, після чого оновивши кількість автоматично бачитиме загальну вартість замовлення (рис.19)

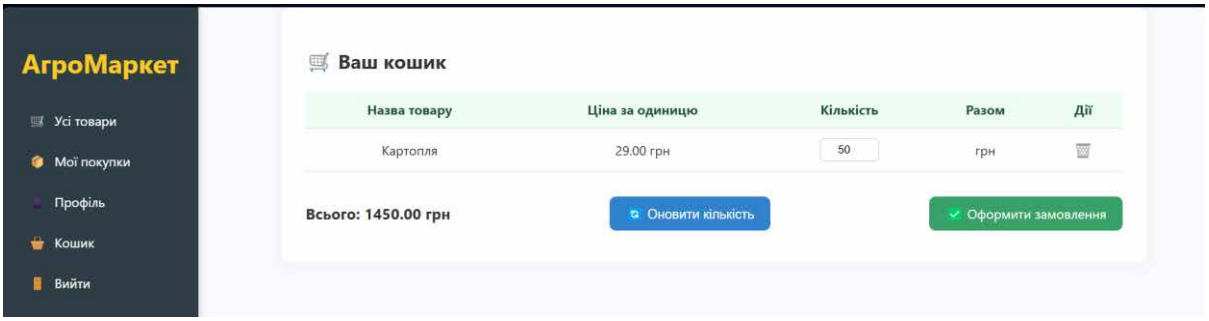


Рис.19 Додавання товару до кошика

Оформлюючи замовлення користувач бачить основні дані маршруту, такі як дані відправки, сам вводить адресу, яка буде місцем доставки, відповідно натиснувши розрахувати маршрут бачить деталі такі як відстань та час у дорозі і відуалізацію у вигляді карти(рис.20)

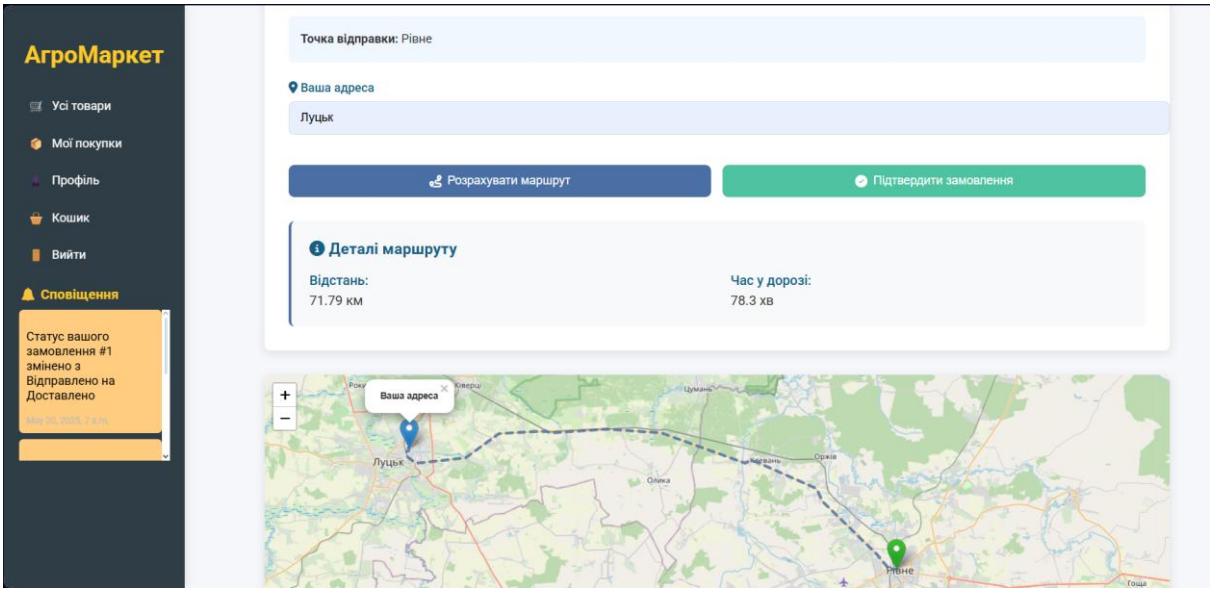


Рис.20 Оформлення замовлення

Оформивши замовлення у вкладці мої покупки є інформація про замовлення, показані основні його деталі та вказаний статус (рис.21).

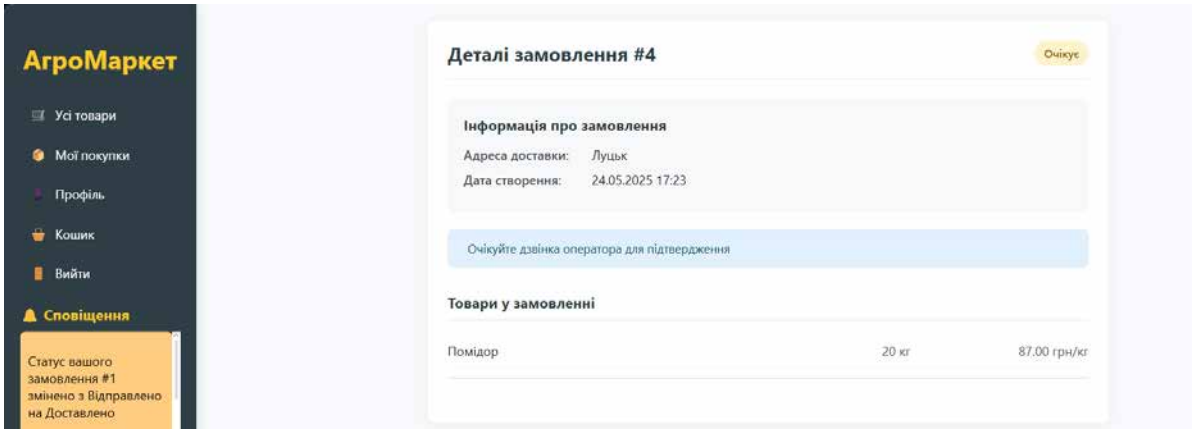


Рис.21 Деталі створеного замовлення

В свою чергу у фермера після отримання замовлення з’являється сповіщення про нове замовлення, і список замовлень він може переглянути у вкладці продаж(рис.22)

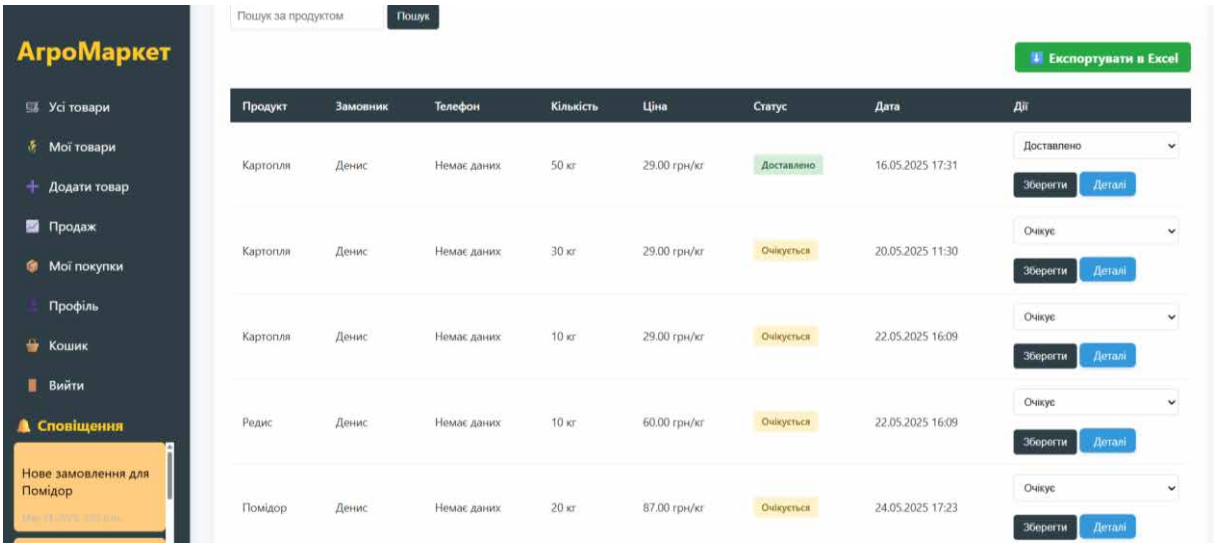


Рис.22 Перелік всіх замовлень фермера

Далі відкривши деталі фермер бачить всі деталі маршруту, які в подальшому спростять процес доставки(рис.23).

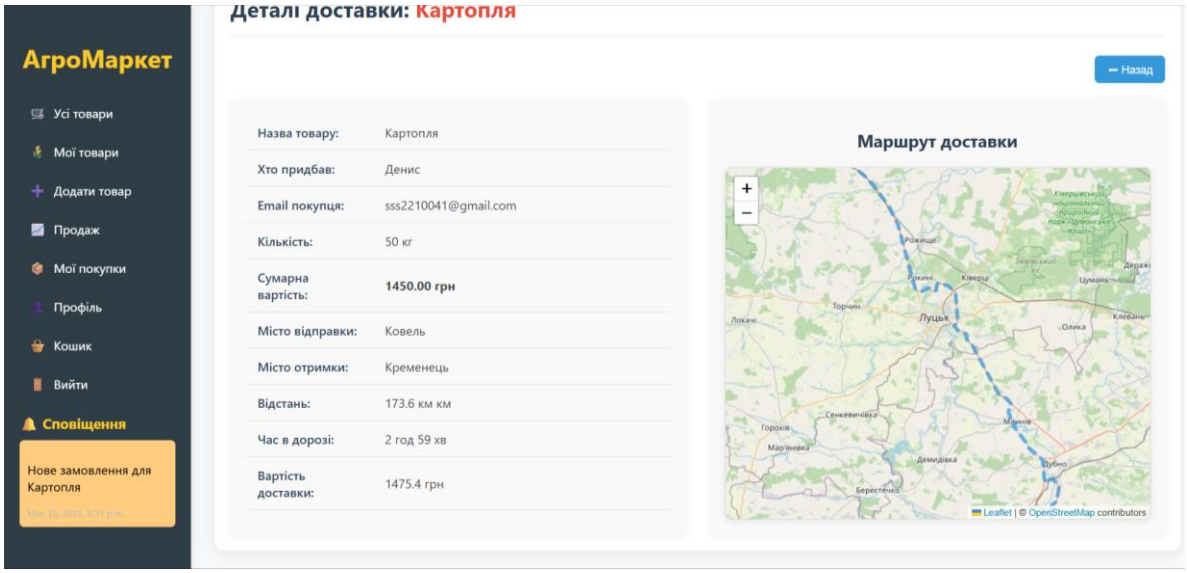
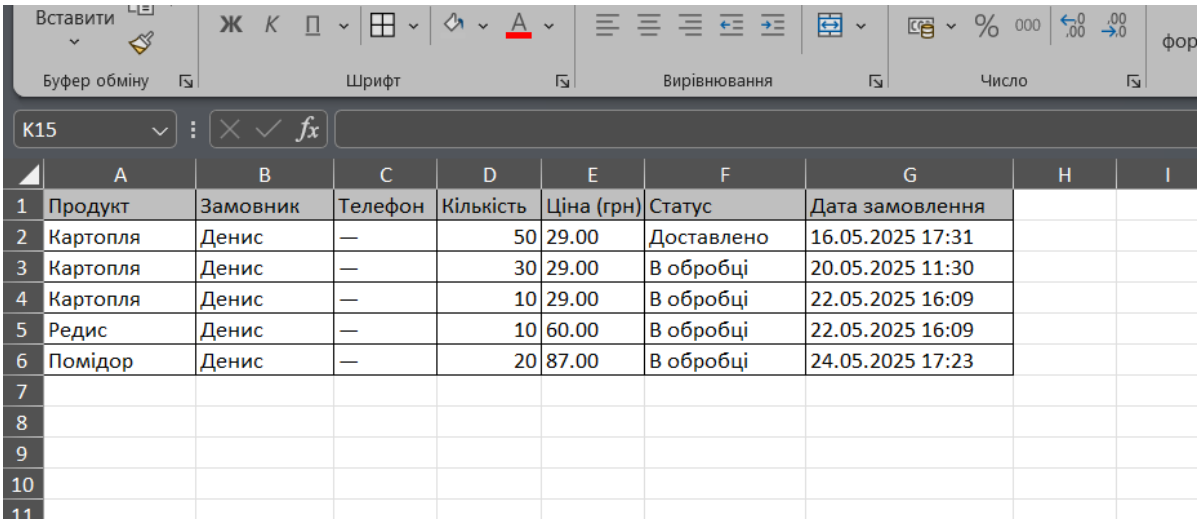


Рис.23 Деталі замовлення та маршруту

У фермера є можливість завантажити файл Excel , тобто експортувати дані про замовлення(рис. 24)

Рис.24 Можливість експортування звіту про замовлення

Дані завантаженого файлу (рис.25)



	A	B	C	D	E	F	G	H	I
1	Продукт	Замовник	Телефон	Кількість	Ціна (грн)	Статус	Дата замовлення		
2	Картопля	Денис	—	50	29.00	Доставлено	16.05.2025 17:31		
3	Картопля	Денис	—	30	29.00	В обробці	20.05.2025 11:30		
4	Картопля	Денис	—	10	29.00	В обробці	22.05.2025 16:09		
5	Редис	Денис	—	10	60.00	В обробці	22.05.2025 16:09		
6	Помідор	Денис	—	20	87.00	В обробці	24.05.2025 17:23		
7									
8									
9									
10									
11									

Рис.25 Файл Excel, який було експортовано

Перелік замовлень, які бачить покупець, аналогічний може бачити і фермер, де відображено дані замовдень та їх статуси (рис. 26)

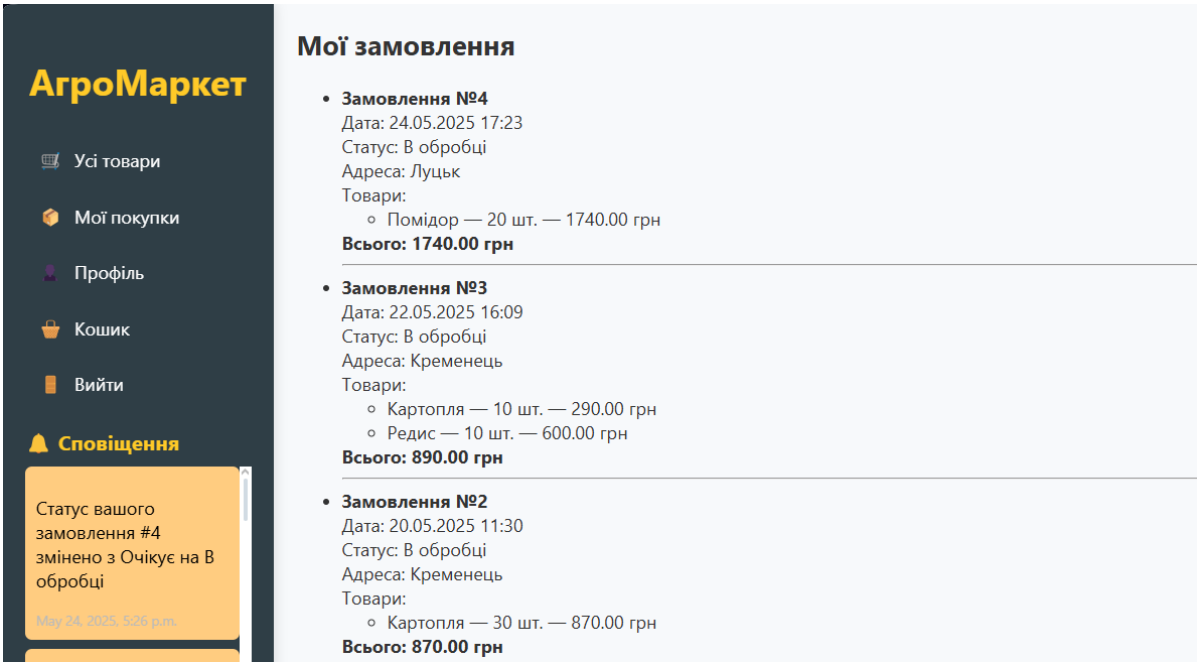


Рис.26 Відображення сповіщень про зміну статусу замовлення у покупця, а також сторінка, де список всіх замовлень

4.2 Вимоги до апаратного та програмного забезпечення

Апаратне та програмне забезпечення, що використовується для роботи інформаційної системи, зазвичай називають конфігурацією системи. Конфігурацію зазвичай поділяють на апаратну частину (тобто фізичні компоненти комп'ютера) та програмне забезпечення (операційна система, середовище розробки, сервіси тощо).

Сучасні комп'ютери побудовані за принципом модульності, що означає, що окремі блоки системи можна гнучко комбінувати. Злагоджена взаємодія між усіма цими блоками гарантується апаратними інтерфейсами, а передача даних відбувається через комунікаційні порти відповідно до визнаних стандартів, відомих як протоколи.[27]

Для забезпечення належного функціонування інформаційної системи управління ланцюгом постачання сільськогосподарської продукції необхідний щонайменше один сервер бази даних і персональний ком’ютер (ПК) для доступу до системи. Сервер обробляє запити, зберігає та керує базою даних, тоді як клієнтські ПК забезпечують взаємодію кінцевого користувача з системою через веб-інтерфейс. Апаратні та програмні вимоги для сервера наведено відповідно у табл.2 та табл.3.

Таблиця 2

Ресурс	**Мінімальний**	**Рекомендований**
Процесор	1.0 ГГц (двоядерний)	2.0 ГГц і вище (4 ядра)
Оперативна пам'ять	2 ГБ	8 ГБ і вище
Жорсткий диск	40 ГБ (HDD)	100 ГБ і вище (SSD рекомендовано)
Мережева швидкість	10 Мбіт/с	100 Мбіт/с і вище
Операційна система	Windows 10 / Linux Ubuntu 18.04	Windows 11 / Ubuntu 20.04 LTS і вище
Монітор (для клієнта)	1024x768	1920x1080 або вище
Відеокарта (для клієнта)	Вбудована	1 ГБ і вище

Таблиця 3

Ресурс	Мінімальні вимоги	Рекомендовані вимоги
Операційна система (сервер)	Ubuntu Server 18.04 / Windows Server 2016	Ubuntu Server 22.04 LTS / Windows Server 2022
Інтерпретатор Python	Python 3.8	Python 3.10 і вище
Фреймворк Django	Django 3.2	Django 4.x
СУБД	PostgreSQL 11	PostgreSQL 14 і вище
Docker	Версія 20.10	Остання стабільна версія Docker + Compose
Браузер (клієнт)	Chrome / Firefox останніх версій	Chrome, Firefox, Edge, Safari (підтримка всіх)

4.3 Склад інсталяційного пакету

Оскільки кінцевим продуктом являється веб застосунок, користувачам немає необхідності встановлювати додаткові файли у себе на персональному комп'ютері, окрім адміністатора системи, який керує програмним забезпеченням і має навати доступ до системи користувачам. Отож, основними інструментами для можливості використання системи є наступні складові:

- Готовий вебзастосунок – працююча система, доступна за адресою (https://b615-91-235-225-240.ngrok-free.app/);
- Облікові дані для входу (це стосується лише користувачів, яким надається доступ як адміністаторам, для фермерів і замовникім є етап реєстрації, де вони самі ці облікові дані створюють);

- База даних на сервері – підключена й налаштована на бекенді системи(для адміністратора системи);
- Серверна логіка (сам бекенд) – частина, яка обробляє запити користувачів і взаємодіє з БД (реалізована мовою програмування Python із використанням вебфреймворку Django) (для адміністратора системи).

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення інформаційної системи управління ланцюгом постачання сільськогосподарської продукції. Програмне забезпечення було створено відповідно до поставленого завдання, з урахуванням особливостей предметної області. Було спроектовано просту, інтуїтивно зрозумілу структуру бази даних, яка забезпечує зберігання та обробку інформації про постачальників, замовлення, клієнтів та сільськогосподарську продукцію.

Дана система дозволяє автоматизувати ключові процеси: реєстрацію замовлень, контроль ланцюга постачання, формування звітів, а також ведення обліку замовників та продукції. Таким чином, система сприятиме підвищенню ефективності управління аграрними поставками, зменшенню кількості ручної роботи, покращенню прозорості та якості обслуговування.

Для розробки програмного забезпечення було використано такі інструменти:

1. Для створення бази даних використано PostgreSQL як надійну реляційну СУБД з відкритим кодом.
2. Для побудови серверної частини системи застосовано фреймворк Django мовою Python, що забезпечує швидку розробку, безпеку та масштабованість.
3. Для зручності розгортання і тестування було використано Docker, який дає змогу створювати ізольоване середовище для запуску застосунку.
4. Для розробки інтерфейсу користувача використано шаблони HTML із використанням CSS та бібліотеки Bootstrap, що забезпечує адаптивність і зручність користування.

У першому розділі було сформульовано мету, завдання, об’єкт і предмет дослідження. Проведено аналіз предметної області, окреслено проблеми, які вирішує запропонована система, та описано вимоги до неї.

У другому розділі було здійснено проєктування логічної моделі даних, обґрунтовано вибір СУБД, а також створено інформаційну базу з урахуванням потреб майбутньої системи.

У третьому розділі описано програмну реалізацію ключових модулів проєкту: моделі даних, логіку обробки запитів (представлення), форми введення та інші елементи структури Django-проєкту. Подано приклади програмного коду, що демонструють роботу основних компонентів.

У четвертому розділі розглянуто питання впровадження та експлуатації інформаційної системи. Наведено вимоги до апаратного і програмного забезпечення, описано принцип роботи системи та її функціональні можливості.

Результатом виконання роботи став функціональний веб-застосунок, який відповідає поставленим цілям і завданням, і може бути використаний для оптимізації процесів постачання сільськогосподарської продукції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голуб Б. Л., Руденський Р. А. Методичні вказівки до розробки бакалаврської кваліфікаційної роботи для студентів спеціальності F3 — «Комп'ютерні науки» освітнього ступеня «Бакалавр» : навч. вид. / Б. Л. Голуб, Р. А. Руденський ; рец. І. Л. Бородкіна, О. М. Ткаченко. — Київ : НУБіП України, 2023. — 57 с.
2. Глазунова О.Г., Кузьмінська О.Г., Волошина Т.В., Корольчук В.І. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Системний аналіз». – Київ: НУБІП, 2020.
3. Методичні вказівки до виконання лабораторних робіт з дисципліни «Технології програмування баз даних»/ уклад.: к.т.н. Голуб Б.Л., асистент Лендєл М.І. – К.:НУБіП, 2023. – 49 с.
4. Навчальний посібник до вивчення дисципліни «Основи організації баз даних» для студентів, що навчаються за спеціальностями галузі 12 «Інформаційні технології» / Голуб Б.Л., Ящук Д.Ю. – К: ТОВ «ЦП КОМПРИНТ», 2017. – 151 с.
5. Трофименко О. Г. Організація баз даних : навч. посібник / О. Г. Трофименко, Ю. В. Прокоп,. Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. І доповн. – Одеса: «Феникс», 2019. – 244 с.
6. Демиденко М. А. Введення в сучасні бази даних : навч. посіб. / М. А. Демиденко ; НТУ «Дніпровська політехніка». – Дніпро : НТУ «Дніпровська політехніка», 2020. – 38 с.
7. Доценко С. І. Організація та системи керування базами даних: Навч. посібник. – Харків: УкрДУЗТ, 2023. – 117 с., рис. 92, табл. 3.
8. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.
9. Козак Л. І., Костюк І. В., Стасевич С. П. Основи програмування: навчальний посібник – Львів: «Новий Світ-2000», 2020. – 328с.

10. Ю.О. Міловідов. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Об'єткно-орієнтоване програмування» – Видавничий центр НУБіП України, 2021. – 163 с.
11. Ю.О. Міловідов. «Об'єктно-орієнтоване програмування» Навчальний посібник друге видання – Видавничий центр НУБіП України, 2022. – 323 с.
12. Яцкевич І. В., Красностанова Н. Е. Цифрові технології у підприємницькій діяльності. Економічний вісник Дніпровської політехніки. 2021. № 1 (73). С. 38–44. DOI: https://doi.org/10.33271/ebdut/73.038
13. Система управління базами даних: сучасні тенденції у розробці [Електронний ресурс] // FoxmindEd. – 2023. – Режим доступу: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/
14. Продовольча та сільськогосподарська організація ООН (ФАО). Digital agriculture [Електронний ресурс]. Режим доступу: https://www.fao.org/digital-agriculture
15. Google Developers. Maps Documentation – Google Maps Platform [Електронний ресурс]. Режим доступу: https://developers.google.com/maps/documentation
16. Український клуб аграрного бізнесу (УКАБ). Офіційний сайт [Електронний ресурс]. Режим доступу: https://ucab.ua/ua
17. PostgreSQL. Про систему. — Режим доступу: https://www.postgresql.org/about/
18. Що таке PostgreSQL і для чого використовується? [Електронний ресурс] // FoxmindEd. – 2023. – Режим доступу: https://foxminded.ua/postgresql-shcho-tse/
19. Farmigo Shuts Down Operations Amid Pullback in Food E-Commerce Investment [Електронний ресурс] // AgFunderNews. – 2016. – Режим доступу: https://agfundernews.com/farmigo-shut-operations-food-e-commerce-continues-attract-investment

20. Farmigo [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Farmigo
21. Freshspire [Електронний ресурс] // GetFreshspired.com. – Режим доступу: https://www.getfreshspired.com/
22. Top 5 Digital Marketplaces for Farmers: Features, Pros, and Cons [Електронний ресурс] // Agritech Digest. – Режим доступу: https://agritechdigest.com/top-5-digital-marketplaces-for-farmers-features-pros-and-cons/amp/
23. Local Line [Електронний ресурс] // LocalLine.co. – Режим доступу: https://www.localline.co/
24. How to use the Postgres Docker Official Image [Електронний ресурс] // Docker. — Режим доступу: https://www.docker.com/blog/how-to-use-the-postgres-docker-official-image/
25. Django documentation [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/
26. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/
27. Романенко О. А. Веб-програмування : навч. посіб. / О. А. Романенко. – Київ : КНЕУ, 2021. – 240 с.
28. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.
29. Python Software Foundation. Python 3 Documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/
30. Ruijie Pang Analysis of Python web development applications based on the Django framework [Електронний ресурс] // Proceedings of SPIE. Third International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2024). – 2024. – Vol. 13181. – 131816C. – Режим доступу: https://doi.org/10.1117/12.3031411

31. Ivanov I. Analysis of the phaunistic composition of Ukraine [Електронний ресурс] // European scientific congress. Proceedings of the 12th International scientific and practical conference. – Madrid, Spain: Barca Academy Publishing, 2023. – С. 21–27. – Режим доступу: https://sciconf.com.ua/xii-mizhnarodna-naukovo-praktichna-konferentsiya-european-scientificcongress-25-27-12-2023-madrid-ispaniya-arhiv/
32. Why did we build Visual Studio Code? [Електронний ресурс] // Visual Studio Code Documentation. — Режим доступу: https://code.visualstudio.com/docs/editor/whyvscode
33. Ременяк Л. В. Конспект лекцій з навчальної дисципліни "Проектування інформаційних систем" для студентів III курсу денної форми навчання напряму – комп'ютерні науки, спеціальності – інформаційні управляючі системи та технології / Л. В. Ременяк. – Одеса : ОДЕкУ, 2016. – 152 с. – укр. мова.
34. Золотухіна О. А., Негоденко О. В., Резник С. Ю., Разіна С. Я. Якість та тестування інформаційних систем: навчальний посібник. Київ: ННІІТ ДУТ, 2020. 128 с.
35. Зінов'єва О. Г. Використання CASE-засобів для проєктування інформаційних систем / О. Г. Зінов'єва // Українські студії в європейському контексті. – 2023. – № 7. – С. 220–227.
36. UML-діаграми [Електронний ресурс]. – Режим доступу: https://foxminded.ua/uml-diagramy/
37. Застосування UML. Частина 2: Діаграма послідовності – Sequence diagram [Електронний ресурс]. – Режим доступу: https://duikt.edu.ua/ua/news-1-0-7897-zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram

ДОДАТОК А

Код бази даних

Створення таблиці order

```
-- Table: public.orders_order

-- DROP TABLE IF EXISTS public.orders_order;

CREATE TABLE IF NOT EXISTS public.orders_order

(

    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),

    created_at timestamp with time zone NOT NULL,

    address character varying(255) COLLATE pg_catalog."default" NOT NULL,

    status character varying(20) COLLATE pg_catalog."default" NOT NULL,

    customer_id bigint NOT NULL,

    CONSTRAINT orders_order_pkey PRIMARY KEY (id),

    CONSTRAINT orders_order_customer_id_0b76f6a4_fk_accounts_user_id FOREIGN KEY (customer_id)

        REFERENCES public.accounts_user (id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        DEFERRABLE INITIALLY DEFERRED

)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.orders_order

    OWNER to "Admin";

-- Index: orders_order_customer_id_0b76f6a4
```

```
-- DROP INDEX IF EXISTS public.orders_order_customer_id_0b76f6a4;

CREATE INDEX IF NOT EXISTS orders_order_customer_id_0b76f6a4

    ON public.orders_order USING btree

    (customer_id ASC NULLS LAST)

    TABLESPACE pg_default;
```

Створення таблиці product

```
-- Table: public.products_product

-- DROP TABLE IF EXISTS public.products_product;

CREATE TABLE IF NOT EXISTS public.products_product

(

    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),

    name character varying(100) COLLATE pg_catalog."default" NOT NULL,

    category character varying(50) COLLATE pg_catalog."default" NOT NULL,

    price numeric(10,2) NOT NULL,

    unit character varying(10) COLLATE pg_catalog."default" NOT NULL,

    description text COLLATE pg_catalog."default" NOT NULL,

    image character varying(100) COLLATE pg_catalog."default" NOT NULL,

    created_at timestamp with time zone NOT NULL,

    address character varying(255) COLLATE pg_catalog."default",

    farmer_id bigint NOT NULL,

    CONSTRAINT products_product_pkey PRIMARY KEY (id),

    CONSTRAINT products_product_farmer_id_2e14b1d5_fk_accounts_user_id FOREIGN KEY (farmer_id)

        REFERENCES public.accounts_user (id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        DEFERRABLE INITIALLY DEFERRED

)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.products_product
```

```
    OWNER to "Admin";
-- Index: products_product_farmer_id_2e14b1d5
-- DROP INDEX IF EXISTS public.products_product_farmer_id_2e14b1d5;
CREATE INDEX IF NOT EXISTS products_product_farmer_id_2e14b1d5
    ON public.products_product USING btree
    (farmer_id ASC NULLS LAST)
    TABLESPACE pg_default;
```

Створення таблиці user

```
-- Table: public.accounts_user
-- DROP TABLE IF EXISTS public.accounts_user;
CREATE TABLE IF NOT EXISTS public.accounts_user
( id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    password character varying(128) COLLATE pg_catalog."default" NOT NULL,
    last_login timestamp with time zone,
    is_superuser boolean NOT NULL,
    username character varying(150) COLLATE pg_catalog."default" NOT NULL,
    first_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    last_name character varying(150) COLLATE pg_catalog."default" NOT NULL,
    email character varying(254) COLLATE pg_catalog."default" NOT NULL,
    is_staff boolean NOT NULL,
    is_active boolean NOT NULL,
    date_joined timestamp with time zone NOT NULL,
    role character varying(10) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT accounts_user_pkey PRIMARY KEY (id),
    CONSTRAINT accounts_user_username_key UNIQUE (username)
)
TABLESPACE pg_default;
ALTER TABLE IF EXISTS public.accounts_user
    OWNER to "Admin";
-- Index: accounts_user_username_6088629e_like
```

```
-- DROP INDEX IF EXISTS public.accounts_user_username_6088629e_like;

CREATE INDEX IF NOT EXISTS accounts_user_username_6088629e_like
    ON public.accounts_user USING btree
    (username COLLATE pg_catalog."default" varchar_pattern_ops ASC NULLS LAST)
    TABLESPACE pg_default;
```

Створення таблиці userprofile

```
-- Table: public.accounts_userprofile

-- DROP TABLE IF EXISTS public.accounts_userprofile;

CREATE TABLE IF NOT EXISTS public.accounts_userprofile
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    phone_number character varying(15) COLLATE pg_catalog."default",
    address text COLLATE pg_catalog."default",
    birthdate date,
    avatar character varying(100) COLLATE pg_catalog."default",
    profile_picture character varying(100) COLLATE pg_catalog."default",
    user_id bigint NOT NULL,
    CONSTRAINT accounts_userprofile_pkey PRIMARY KEY (id),
    CONSTRAINT accounts_userprofile_user_id_key UNIQUE (user_id),
    CONSTRAINT accounts_userprofile_user_id_92240672_fk_accounts_user_id FOREIGN KEY (user_id)
        REFERENCES public.accounts_user (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        DEFERRABLE INITIALLY DEFERRED
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.accounts_userprofile
    OWNER to "Admin";
```

ДОДАТОК Б

Код програми

Код файлу accounts_manage.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils.translation import gettext_lazy as _
class User(AbstractUser):
    ROLE_CHOICES = (
        ('farmer', 'Фермер'),
        ('customer', 'Покупець'),
        ('admin', 'Адміністратор'),
    )
    role = models.CharField(max_length=10, choices=ROLE_CHOICES)
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='userprofile')
    phone_number = models.CharField(max_length=15, blank=True, null=True)
    address = models.TextField(blank=True, null=True)
    birthdate = models.DateField(blank=True, null=True)
    avatar = models.ImageField(upload_to='avatars/', blank=True, null=True)
    profile_picture = models.ImageField(upload_to='profile_pictures/', blank=True, null=True)
    def __str__(self):
        return f"Profile of {self.user.username}"
```

Код файлу accounts_urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    path('products/', include('products.urls')),  # додаємо наш app
    path('', include('accounts.urls')),  # якщо є логін/реєстрація
    path('orders/', include('orders.urls')),
    path('', include('maps.urls')),
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Код файлу orders_ manage.py

```
from django.db import models
from accounts.models import User
from products.models import Product
class Cart(models.Model):
    customer = models.OneToOneField(User, on_delete=models.CASCADE,
limit_choices_to={'role': 'customer'})
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"Cart of {self.customer.username}"
    def total_items(self):
        """Повертає загальну кількість товарів у кошику"""
        return sum(item.quantity for item in self.items.all())
    def total_price(self):
        """Повертає загальну суму всіх товарів у кошику"""
        return sum(item.product.price * item.quantity for item in self.items.all())
class CartItem(models.Model):
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE, related_name='items')
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)  # Встановлюємо значення за
замовчуванням
    def __str__(self):
        return f"{self.quantity} x {self.product.name}"
    def save(self, *args, **kwargs):
        """Перевірка кількості перед збереженням (не менше 1)"""
```

```
        if self.quantity < 1:
            self.quantity = 1
        super().save(*args, **kwargs)
class Order(models.Model):
    STATUS_CHOICES = [
        ('pending', 'Очікує'),
        ('processing', 'В обробці'),
        ('shipped', 'Відправлено'),
        ('delivered', 'Доставлено'),
    ]
    customer = models.ForeignKey(User, on_delete=models.CASCADE, limit_choices_to={'role':
'customer'})
    created_at = models.DateTimeField(auto_now_add=True)
    address = models.CharField(max_length=255)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending')
    def __str__(self):
        return f"Order #{self.pk} by {self.customer.username}"
    def total_items(self):
        """Повертає загальну кількість товарів в замовленні"""
        return sum(item.quantity for item in self.items.all())
    def total_price(self):
        """Повертає загальну суму всіх товарів в замовленні"""
        return sum(item.product.price * item.quantity for item in self.items.all())
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()
    created_at = models.DateTimeField(auto_now_add=True)  # Час створення сповіщення
    def __str__(self):
        return f"{self.quantity} x {self.product.name}"
    def total_price(self):
        """Повертає загальну ціну цього товару в замовленні"""
```

```
        return self.product.price * self.quantity
class Notification(models.Model):
    message = models.TextField()  # Текст сповіщення
    user = models.ForeignKey(User, on_delete=models.CASCADE)  # Користувач, якому належить сповіщення
    created_at = models.DateTimeField(auto_now_add=True)  # Час створення сповіщення
    read = models.BooleanField(default=False)  # Статус прочитано/непрочитано
    STATUS_TRANSLATIONS = {
        'Pending': 'Очікує',
        'Processing': 'В обробці',
        'Shipped': 'Відправлено',
        'Delivered': 'Доставлено',
    }
    def __str__(self):
        return f"Сповіщення для {self.user.username}: {self.message}"
    def mark_as_read(self):
        """Маркує сповіщення як прочитане"""
        self.read = True
        self.save()
```

Код файлу orders_ views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Cart, CartItem, Notification  # Додано Notification
from products.models import Product
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .models import Order, OrderItem
from accounts.models import UserProfile
from django.http import HttpResponseBadRequest
from django.core.mail import send_mail
from django.conf import settings
from .utils import *
import requests
```

```
from django.core.paginator import Paginator
from django.views.decorators.http import require_POST
from django.views.decorators.http import require_GET
import openpyxl
from django.http import HttpResponse
@login_required
def cart_view(request):
    """Перегляд кошика"""
    cart, created = Cart.objects.get_or_create(customer=request.user)
    return render(request, 'orders/cart.html', {'cart': cart})
@login_required
def add_to_cart(request, pk):
    """Додавання товару до кошика"""
    product = get_object_or_404(Product, pk=pk)
    if request.method == 'POST':
        quantity = int(request.POST.get('quantity', 1))  # За замовчуванням 1
        # Отримуємо або створюємо кошик користувача
        cart, created = Cart.objects.get_or_create(customer=request.user)
        # Перевіряємо чи товар вже є в кошику
        cart_item, created = CartItem.objects.get_or_create(cart=cart, product=product)
        if not created:
            cart_item.quantity += quantity
            cart_item.save()
        else:
            cart_item.quantity = quantity
            cart_item.save()
        return redirect('cart_view')  # Перенаправлення на сторінку кошика
    return render(request, 'product_detail.html', {'product': product})
@login_required
def remove_from_cart(request, pk):
    """Видалити товар з кошика"""
```

```
    cart = Cart.objects.get(customer=request.user)
    cart_item = get_object_or_404(CartItem, pk=pk, cart=cart)
    cart_item.delete()
    messages.success(request, "Товар видалено з кошика.")
    return redirect('cart_view')
@login_required
def checkout(request):
    cart = Cart.objects.get(customer=request.user)
    if not cart.items.exists():
        messages.error(request, "Ваш кошик порожній!")
        return redirect('cart_view')
    first_item = cart.items.first()
    farmer_address = first_item.product.address
    routes_info = []
    default_address = ""
    if request.method == 'POST':
        if 'calculate_route' in request.POST:
            start_address = request.POST.get('start', '')
            try:
                start_coords = geocode_address(start_address)
                end_coords = geocode_address(farmer_address)
                route_data = get_routes(start_coords, end_coords)
                for route in route_data['features']:
                    routes_info.append({
                        'coordinates': route['geometry']['coordinates'],
                        'distance_km': round(route['properties']['segments'][0]['distance'] / 1000, 2),
                        'duration_min': round(route['properties']['segments'][0]['duration'] / 60, 1)
                    })
                if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
                    from django.http import JsonResponse
                    return JsonResponse({
```

```
                'success': True,
                'routes_info': routes_info,
                'farmer_name': first_item.product.farmer.farmer,
                'farmer_address': farmer_address
            })
        except Exception as e:
            if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
                return JsonResponse({'success': False, 'error': str(e)})
            messages.error(request, f"Помилка побудови маршруту: {str(e)}")
    elif 'confirm_order' in request.POST:
        # Кнопка "Підтвердити замовлення"
        client_address = request.POST.get('start', '')
        order = Order.objects.create(
            customer=request.user,
            address=client_address
        )
        for item in cart.items.all():
            OrderItem.objects.create(
                order=order,
                product=item.product,
                quantity=item.quantity
            )
            Notification.objects.create(
                user=item.product.farmer,
                message=f"Нове замовлення для {item.product.name}"
            )
        cart.items.all().delete()
        messages.success(request, "Замовлення успішно створено!")
        return redirect('order_detail', pk=order.pk)
return render(request, 'orders/checkout.html', {
    'cart': cart,
```

```
        'farmer_address': farmer_address,
        'routes_info': routes_info,
        'default_address': default_address
    })

@login_required
def order_detail(request, pk):
    """Деталі замовлення"""
    order = get_object_or_404(Order, pk=pk, customer=request.user)
    return render(request, 'orders/order_detail.html', {'order': order})
@login_required
def mark_as_read(request, notification_id):
    """Позначити сповіщення як прочитане"""
    notification = get_object_or_404(Notification, id=notification_id)
    notification.read = True
    notification.save()
    return redirect('notifications_view')
def update_cart_item(request, item_id):
    """Оновлення товару в кошику"""
    item = get_object_or_404(CartItem, pk=item_id)
    if request.method == 'POST':
        quantity = request.POST.get('quantity')
        if quantity and quantity.isdigit() and int(quantity) > 0:
            item.quantity = int(quantity)
            item.save()
            return redirect('cart_view')
        else:
            return HttpResponseBadRequest("Невірна кількість товару")
    return redirect('cart_view')
@login_required
def farmer_report(request):
```

```
    # Перевірка що користувач має роль фермер
    if request.user.role != 'farmer':
        return render(request, 'base.html')  # або редірект
    order_items = OrderItem.objects.filter(
        product__farmer=request.user,
        confirmed_by_farmer=True
    ).select_related('product', 'order__customer', 'order__customer__userprofile')
    report_data = []
    for item in order_items:
        customer = item.order.customer
        user_profile = customer.userprofile
        report_data.append({
            'product_name': item.product.name,
            'customer_username': customer.username,
            'customer_phone': user_profile.phone_number if user_profile else '',
            'customer_address': user_profile.address if user_profile else '',
            'quantity': item.quantity,
        })
    return render(request, 'farmer_report.html', {'report_data': report_data})
@login_required
def farmer_purchases(request):
    if request.user.role != 'farmer':
        messages.error(request, 'Доступ заборонено')
        return redirect('home')
    farmer_products = Product.objects.filter(farmer=request.user)
    order_items = OrderItem.objects.filter(
        product__in=farmer_products
    ).select_related('product', 'order', 'order__customer')
    # Сортування
    sort_by = request.GET.get('sort_by', 'created_at')  # За замовчуванням сортуємо за датою створення
    sort_order = request.GET.get('sort_order', 'asc')  # 'asc' або 'desc'
```

```
if sort_order == 'desc':
    order_items = order_items.order_by(f'-{sort_by}')
else:
    order_items = order_items.order_by(sort_by)
# Групуємо товари по замовленнях
orders_data = {}
for item in order_items:
    order_id = item.order.id
    if order_id not in orders_data:
        orders_data[order_id] = {
            'order_id': order_id,
            'customer_username': item.order.customer.username,
            'customer_phone': getattr(item.order.customer, 'phone', 'Немає даних'),
            'order_status': item.order.get_status_display(),
            'status_value': item.order.status,
            'created_at': item.order.created_at,
            'items': []
        }
    orders_data[order_id]['items'].append({
        'product_name': item.product.name,
        'quantity': item.quantity,
        'price': item.product.price,
        'product_unit': item.product.get_unit_display(),
        'total_price': item.quantity * item.product.price,
    })
# Перетворюємо в список для шаблону
purchase_data = []
for order_id, data in orders_data.items():
    for item in data['items']:
        purchase_data.append({
            'order_id': data['order_id'],
```

```
                'product_name': item['product_name'],
                'customer_username': data['customer_username'],
                'customer_phone': data['customer_phone'],
                'quantity': item['quantity'],
                'price': item['price'],
                'product_unit': item['product_unit'],
                'total_price': item['total_price'],
                'order_status': data['order_status'],
                'status_value': data['status_value'],
                'created_at': data['created_at'],
            })
    # Пагінація
    page_number = request.GET.get('page', 1)
    paginator = Paginator(purchase_data, 10)  # 10 елементів на сторінці
    page_obj = paginator.get_page(page_number)
    context = {
        'purchase_data': page_obj,
        'status_choices': Order.STATUS_CHOICES,
        'sort_by': sort_by,
        'sort_order': sort_order,
    }
    return render(request, 'farmer_purchases.html', context)
@login_required
@require_POST
def update_order_status(request, order_id):
    order = get_object_or_404(Order, pk=order_id)
    # Перевіряємо, чи замовлення містить товари цього фермера
    farmer_products = Product.objects.filter(farmer=request.user)
    if not order.items.filter(product__in=farmer_products).exists():
        messages.error(request, 'Це замовлення не містить ваших товарів')
        return redirect('farmer_purchases')
```

```
    new_status = request.POST.get('order_status')
    if new_status in dict(Order.STATUS_CHOICES):
        old_status = order.status
        order.status = new_status
        order.save()
        # Створюємо сповіщення для покупця
        Notification.objects.create(
            user=order.customer,
            message=f"Статус вашого замовлення #{order.id} змінено з {dict(Order.STATUS_CHOICES)[old_status]} на {dict(Order.STATUS_CHOICES)[new_status]}",
        )
        messages.success(request, f'Статус замовлення #{order.id} успішно оновлено')
    else:
        messages.error(request, 'Невірний статус замовлення')
    return redirect('farmer_purchases')
@login_required
def customer_orders_view(request):
    orders = Order.objects.filter(customer=request.user).order_by('-created_at')
    return render(request, 'orders/my_orders.html', {'orders': orders})
@login_required
def update_cart(request):
    if request.method == "POST":
        cart = get_object_or_404(Cart, customer=request.user)
        for item in cart.items.all():
            quantity_key = f'quantities_{item.pk}'
            new_quantity = int(request.POST.get(quantity_key, item.quantity))
            if new_quantity > 0:
                item.quantity = new_quantity
                item.save()
        cart.save()
    return redirect('cart_view')
@login_required
```

```
def purchase_list(request):
    # Отримуємо замовлення з відповідними полями
    qs = OrderItem.objects.filter(product__farmer=request.user).annotate(
        order_id=F('order__id'),
        customer_username=F('order__customer__username'),
        customer_phone=F('order__order__customer__phone') if hasattr(Order, 'phone') else F('order__customer__phone'),
        status_value=F('order__status'),
        created_at=F('order__created_at'),
        product_name=F('product__name'),
        product_unit=F('product__unit'),
        price=F('product__price'),
    )
    # Пошук
    search = request.GET.get('search')
    if search:
        qs = qs.filter(product__name__icontains=search)
    # Сортування (за замовчуванням по створенню в порядку спадання)
    sort = request.GET.get('sort', 'created_at')  # Якщо сортування не вказано, за замовчуванням по даті
    order_dir = '-' if request.GET.get('order', 'desc') == 'desc' else ''  # За замовчуванням сортування по спаданням
    qs = qs.order_by(f"{order_dir}{sort}")  # Для поля created_at забезпечуємо сортування від новішого до старішого
    # Пагінація
    paginator = Paginator(qs, 10)
    page = request.GET.get('page')
    page_obj = paginator.get_page(page)
    return render(request, 'orders/purchase_list.html', {
        'purchase_data': page_obj,
        'status_choices': Order.STATUS_CHOICES,
    })
```

```
@login_required
def delivery_detail(request, item_id):
    item = get_object_or_404(OrderItem, id=item_id, product__farmer=request.user)
    order = item.order
    product = item.product
    customer = order.customer
    origin_addr = product.address or ''
    destination_addr = order.address or ''
    city_to = destination_addr.split(',', 1)[0].strip() if destination_addr else 'Невідомо'
    city_from = origin_addr.split(',', 1)[0].strip() if origin_addr else 'Невідомо'
    try:
        start_coords = geocode_address(origin_addr)
        end_coords = geocode_address(destination_addr)
        route_data = get_routes(start_coords, end_coords)
        route_coords = route_data['features'][0]['geometry']['coordinates']
        summary = route_data['features'][0]['properties']['summary']
        distance_km = summary['distance'] / 1000  # переведемо в км
        distance_text = f"{distance_km:.1f} км"
        # Обчислення вартості доставки
        delivery_cost = round(distance_km * 8.5, 2)  # 8.5 грн за км
        duration_sec = summary['duration']
        hrs, rem = divmod(duration_sec, 3600)
        mins = rem // 60
        duration_text = f"{int(hrs)} год {int(mins)} хв" if hrs else f"{int(mins)} хв"
    except Exception as e:
        logger.error(f"Error processing route: {e}")
        start_coords = end_coords = route_coords = None
        distance_text = duration_text = None
        delivery_cost = 0.00  # Якщо не вдалося отримати відстань, то ціна 0
    context = {
        'item': item,
```

```
        'order': order,
        'product': product,
        'customer': customer,
        'quantity': item.quantity,
        'total_price': item.total_price(),
        'city_from': city_from,
        'city_to': city_to,
        'distance': distance_text,
        'duration': duration_text,
        'start_coords': start_coords,
        'end_coords': end_coords,
        'route_coords': route_coords,
        'delivery_cost': delivery_cost,  # додаємо в контекст
    }
    return render(request, 'orders/delivery_detail.html', context)
@login_required
def export_purchases_excel(request):
    # Створюємо нову Excel-книгу
    workbook = openpyxl.Workbook()
    sheet = workbook.active
    sheet.title = "Покупки"
    # Заголовки таблиці
    headers = ['Продукт', 'Замовник', 'Телефон', 'Кількість', 'Ціна (грн)', 'Статус', 'Дата замовлення']
    sheet.append(headers)
    # Витягуємо дані
    purchases = OrderItem.objects.select_related('order', 'product', 'order__customer').all()
    for item in purchases:
        sheet.append([
            item.product.name,
            item.order.customer.username,
            item.order.customer.phone_number if hasattr(item.order.customer, 'phone_number') else '—',
```

```
            item.quantity,
            f"{item.product.price:.2f}",
            item.order.get_status_display(),
            item.created_at.strftime('%d.%m.%Y %H:%M'),
        ])
    # Готуємо відповідь
    response = HttpResponse(content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')
    response['Content-Disposition'] = 'attachment; filename=покупки.xlsx'
    workbook.save(response)
    return response
```

ДОДАТОК В

Схема даних