НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення системи онлайн продажу агротехніки для фермерських господарств»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____/ Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____/ Бородкін Г.О./
підпис ПІБ

Виконав:_____/ Плетюк Р. М./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
_____ / Голуб Б.Л., доцент, к.т.н /
підпис
"17" березня 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту
Плетюку Роману Михайловичу
Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення системи онлайн продажу агротехніки для фермерських господарств

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 15
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення, навчальна та методична література, державні стандарти

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи _____ / Бородкін Г.О. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Плетюк Р. М./
підпис ініціали та прізвище

Дата отримання завдання 2025 . 03 . 17
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX - Asynchronous JavaScript and XML, технологія, яка дозволяє асинхронно обмінюватися даними між клієнтом і сервером без перезавантаження сторінки

ASP.NET - Active Server Pages .NET, фреймворк для веб-розробки, розроблений компанією Microsoft

ER - Entity-Relationship, модель для проєктування баз даних

GPS - Global Positioning System, система глобального позиціонування

HTML, CSS та JavaScript - мови та технології для створення веб-сторінок

JSON - JavaScript Object Notation, формат обміну даними, зазвичай використовується для передачі даних між клієнтом і сервером

MVC - Model-View-Controller, архітектурний шаблон для побудови веб-додатків

MS - Microsoft

ORM - Object-Relational Mapping, технологія, що дозволяє використовувати об'єктно-орієнтовані моделі для роботи з реляційними базами даних

SQL – Structured Query Language

UML - Unified Modeling Language, мова для моделювання програмного забезпечення

XML - eXtensible Markup Language, мова розмітки для зберігання та передавання даних

БД – база даних

ІС – інформаційна система

ПЗ – програмне забезпечення

СУБД – система управління базою даних

ВСТУП

На сучасному етапі розвитку сільського господарства технології відіграють вирішальну роль у підвищенні ефективності сільськогосподарських підприємств. Одним із найважливіших аспектів розвитку є впровадження інноваційних рішень для автоматизації та оптимізації бізнес-процесів. Фермерські господарства, особливо малі та середні, стикаються з низкою проблем, пов'язаних із придбанням сільськогосподарської техніки: недостатня інформація про нові моделі, складні процедури пошуку постачальників, високі логістичні витрати та необхідність порівняння різних пропозицій. У зв'язку з цим онлайн-платформи для продажу сільськогосподарської техніки стають ще більш популярними, оскільки вони значно спрощують процес вибору та придбання необхідної техніки чи обладнання.

Об'єктом дослідження дипломної роботи є система продажу агротехніки, яка забезпечує облік, управління та автоматизацію процесів продажу сільськогосподарської техніки, а також взаємодію продавцями та покупцями. Вона включає інструменти для аналізу попиту, управління замовленнями, контролю запасів і планування логістики.

Предметом дослідження є методи та засоби підвищення ефективності продажу агротехніки, що включає оптимізацію бізнес-процесів, автоматизацію комунікацій з клієнтами, покращення вибору та порівняння техніки пред покупкою.

Завдання дослідження:

- Проаналізувати сучасні підходи до автоматизації процесів продажу агротехніки.
- Розробити модель системи для ефективного управління продажами.
- Виконати тестування розробленої системи на прикладі реальних бізнес-процесів.

Актуальність теми цієї дипломної роботи зумовлена необхідністю створення ефективного механізму для полегшення доступу фермерів до якісної сільськогосподарської техніки. Сьогодні більшість аграріїв стикаються з труднощами при пошуку, порівнянні, виборі та придбанні необхідної техніки через обмежену кількість постачальників, високі логістичні витрати та брак інформації про технічні характеристики продукції. Запропонована система онлайн-продажів допоможе вирішити ці проблеми, забезпечивши зручний інструмент для швидкого та ефективного обміну інформацією між учасниками ринку.

Метою цієї роботи є розробка програмного забезпечення для онлайн-продажу сільськогосподарської техніки, яке допоможе власникам фермерських господарств швидко та зручно знаходити, порівнювати та замовляти сільськогосподарську техніку відповідно до своїх потреб.

Для досягнення цієї мети буде проаналізовано існуючі онлайн-платформи для продажу сільськогосподарської техніки з метою визначення їхніх переваг та недоліків. На основі цього аналізу буде розроблено архітектуру системи з ключовими модулями, включаючи: реєстрацію користувачів, перегляд та пошук техніки за категоріями, кошик для покупок, розміщення замовлень та адміністрування замовлень. Особлива увага буде приділена створенню зручної адміністративної панелі та забезпеченню захисту персональних даних

Використання сучасних технологій, таких як автоматизація замовлень та персоналізовані рекомендації, покращить обслуговування клієнтів, оптимізує логістику та підвищить конкурентоспроможність компаній. Крім того, система може бути корисною для державних органів з точки зору моніторингу ринку та підтримки фермерських підприємств. Загалом, запропоноване рішення сприяє діджиталізації аграрного сектору та впровадженню інновацій у сільське господарство.

Для створення веб-додатку було обрано платформу ASP.NET MVC, оскільки вона забезпечує зручну організацію логіки бізнес-процесів та покращену структуру коду. Використання бази даних SQL Server дозволяє ефективно зберігати та

обробляти великі обсяги інформації про продукцію, замовлення та користувачів. Ці технології забезпечують високу продуктивність та стабільність системи, а також гнучкість масштабування додатку в майбутньому.

Результат цієї бакалаврської роботи матиме практичне значення як для фермерів, так і для компаній, що займаються виробництвом та продажем сільськогосподарської техніки. Також за результатами роботи відбувалися доповіді на конференціях «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025» та «Інформаційні технології в соціокультурній сфері, освіті та економіці».

Бакалаврська робота складається зі вступу, чотирьох розділів, висновків та списку використаної літератури. Основний зміст викладено на сімдесяти двох сторінках, доповнено одинадцятьма ілюстраціями та п'ятьма таблицями. Список використаної літератури містить п'ятнадцять джерел. Загальний обсяг роботи становить сто сторінок.

Перший розділ, «Системний аналіз предметної області», охоплює опис предметної області, аналіз вимог до програмної системи, огляд існуючих рішень та постановка завдання для розробки системи. Другий розділ «Проєктування інформаційного та програмного забезпечення» охоплює створення логічної моделі даних, діаграми класів, кооперацій, пакетів та компонентів. У третьому розділі «Розробка інформаційного та програмного забезпечення» описується система керування базами даних, розробка бази даних, вибір інструментів для побудови прикладного програмного забезпечення, а також алгоритмізація та програмування модулів. У четвертому розділі «Рекомендації щодо впровадження та експлуатації системи» наведено рекомендації щодо тестування систем, вимог до апаратного та програмного забезпечення, а також склад інсталяційного пакета.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сільське господарство - один із ключових секторів української економіки. Воно створює робочі місця та сприяє розвитку експорту. Сучасні аграрні підприємства активно впроваджують передові технології для підвищення урожайності та ефективності виробництва. Завдяки розвитку інтернет-технологій фермери отримали можливість швидко та вигідно купувати техніку онлайн, що суттєво спрощує процес вдосконалення фермерських господарств.

Агротехніка - це машини та обладнання для ведення господарства, обробки ґрунтів, посіву, збирання культур, піклування про рослини та виконання інших завдань у сільському господарстві. Основна сільськогосподарська техніка включає такі категорії: трактори, комбайни, обприскувачі, плуги та інше обладнання необхідне для обробки ґрунту. Сучасна техніка має GPS-навігацію, датчики та інші інтелектуальні підсистеми, які допомагають керувати станом техніки та підвищенням ефективності роботи.[1]

Продажі сільськогосподарської техніки онлайн зараз набирають популярності, оскільки саме так фермери мають доступ до великого вибору техніки, можуть порівнювати характеристики, вартість та купувати те чи інше обладнання, зважаючи саме на свої потреби. Це дуже важливо для віддалених куточків країни, де звичайні методи придбання техніки менш використовуються. Онлайн-платформи значно спрощують процес придбання, зменшують витрати на рекламу та обслуговування клієнтів, а також надають детальну інформацію про моделі та детальні характеристики.

Фермерські господарства різняться за величиною та спеціалізацією: великі сільськогосподарські компанії обирають більш потужне обладнання для обробки великих площ, тоді як дрібніші фермери віддають перевагу більш компактним та не дуже дорогим моделям. Важливим аспектом є надання потенційним покупцям

порад та варіантів для порівняння, щоб вони могли обрати саме те обладнання, яке їм необхідне, в залежності від їх потреб.

Предметна область розроблюваної системи охоплює деяку кількість процесів, які фермери здійснюють для успішного ведення своєї сільськогосподарської діяльності. До таких процесів належать: вибір та придбання певного обладнання, його обслуговування та експлуатація в реальних умовах.

Цифрові технології та системи онлайн-продажів спрощують такі процеси, надаючи користувачам доступ до великого вибору обладнання, дозволяючи порівнювати характеристики, вартість, також надаючи можливість проводити консультації що призведе до найкращого вибору. Це підвищує ефективність покупок, особливо для малих та середніх фермерських господарств.[2]

Предметна область даної системи охоплює такі процеси: вибору, порівняння, придбання та обслуговування сільськогосподарської техніки через онлайн-платформи. Це допоможе фермерам або ж приватним особам підвищити ефективність ведення свого господарства.

Створення простої, доступної та функціональної платформи дозволить усім охочим швидко знаходити необхідну техніку, порівнювати моделі та вибирати найкращі рішення саме для свого бізнесу, для своїх потреб.

1.2 Аналіз вимог до програмної системи

1.2.1 Визначення функціональних вимог. Проаналізувавши предметну область розроблюваної програмної системи продажу агротехніки онлайн, потрібно визначити вимоги до неї. Функціональні вимоги - це певний набір можливостей, що характеризують те, що повинна вирішувати програмна система, що вона повинна робити та як себе поводити. Це сформульовані дії, які користувач може виконувати у системі, а вона в свою чергу повинна правильно на них реагувати. Також функціональні вимоги визначають те, як система повинна взаємодіяти з іншими системами, підсистемами або модулями. Функціональні вимоги також

включають вимоги до основних бізнес-процесів, алгоритмів, логіки обробки даних та механізму взаємодії з іншими програмними компонентами.

Функціональні вимоги не лише вказують на унікальні потреби користувачів до програмної системи, але й на найменші деталі її роботи, що є дуже важливо для досягнення поставлених цілей. Вони описують, як система повинна відреагувати на певні дії користувача, які функції мають бути доступні для виконання тих чи інших завдань, які вимоги до взаємодії між різними модулями, частинами системи та як платформа має поводити себе в різних ситуаціях. Це дозволить створити представлення того, як система буде функціонувати, як буде відбуватися взаємодія з користувачами і як вона має взаємодіяти з іншими частинами, модулями або підсистемами.

Функціональні вимоги є фундаментом для розробки будь-якої програмної системи, оскільки саме вони визначають основну логіку обробки даних, архітектуру програмного забезпечення, визначаючи його основні функціональні можливості та поведінку. Вони є ключовими при створенні документації, проектуванні архітектури майбутньої системи, розробці алгоритмів та написанні програмного коду і тестуванні вже готового продукту. Правильно визначенні функціональні вимоги дозволяють створити програму, що відповідає вимогам замовника і очікуванням кінцевих користувачів, а також забезпечують ефективну взаємодію між усіма елементами та користувачами системи.

Функціональні вимоги поділяються на категорії:

- Основні функції системи: ключові можливості, що власне і визначають її призначення (наприклад, можливість реєстрації, оформлення замовлень тощо).
- Обробка даних: описує, як система працює з інформацією, зокрема як відбувається збереження, оновлення чи видалення.

- Інтерфейс користувача: визначає, як саме користувачі будуть взаємодіяти із системою (графічний інтерфейс, наявність мобільної версії).
- Обмеження та правила роботи: встановлює права доступу до певних дій користувачів залежно від ролі або того, чи користувач авторизований.

Функціональні вимоги є певним фундаментом при проектуванні та розробці програмної системи, так як вони визначають, як саме вона має працювати і що вона повинна робити. У системі продажу агротехніки буде дві основні ролі користувачів: менеджер (адміністратор магазину) та клієнти (покупці). Менеджер або адміністратор магазину матиме доступ до функцій, які доступні лише йому: управління оголошеннями і замовленнями, перегляд звітності. Звичайні користувачі зможуть користуватися основними можливостями для перегляду техніки, оформлення замовлень та відстеження їхнього статусу.

Функціональні можливості для адміністратора (менеджера магазину):

- Додавання нових товарів: адміністратор має мати можливість додавати нові одиниці агротехніки, завантажувати фотографії та вказувати технічні характеристики, вартість тощо.
- Редагування та видалення техніки: можливість змінювати наявні характеристики або взагалі видаляти продукцію з каталогу.
- Керування категоріями: створення нових категорій техніки та їхнє оновлення.
- Обробка замовлень: відстеження та підтвердження замовлень покупців, зміна статусу замовлення.
- Формування звітів про продажі: генерація звітних документів з детальною інформацію про продажі, наявну техніку, покупців для подальшого аналізу ефективності роботи магазину.

Функціональні можливості для покупців:

- Перегляд каталогу техніки: користувач має мати можливість ознайомитися з наявною технікою, отримати детальну інформацію про кожну одиницю техніки.

- Фільтри та пошук: система повинна дозволяти користувачам шукати бажану техніку за категоріями, характеристиками, ціною, виробником або іншими параметрами.
- Порівняння техніки між собою: користувачі мають мати можливість додавати техніку до списку порівнянь, щоб порівняти їхні характеристик.
- Додавання техніки до кошика: покупець має мати можливість додавати техніку до кошика для подальшого здійснення замовлення.
- Оформлення замовлення: після вибору усіх бажаних одиниць техніки, вказання їх кількості користувач має мати можливість ввести свої дані та оформити замовлення.
- Перегляд історії замовлень: покупець має мати можливість переглядати статус поточного замовлення та мати доступ до історії попередніх покупок.

Розроблювана система повинна відображати якісний каталог з переліком одиниць техніки та детальним описом кожної з них. Техніка має бути згрупована за категоріями, такими як трактори, плуги, снігоочисники тощо, які можна сортувати за ціною, виробником або іншими параметрами. Для зручності користувачів буде реалізовано функцію пошуку за ключовими словами.

Реєстрація та авторизація користувачів дозволять отримати доступ до персоналізованого контенту, такого як перегляд історії покупок та статусу замовлення. Авторизовані користувачі також зможуть додавати техніку до кошика та змінювати кількість перед оформленням замовлення. Кошик буде збережено, навіть якщо користувач вийдете з облікового запису, що дозволить повернутися до нього пізніше і навіть з будь-яких інших пристроїв.

Після того, як клієнт додасть бажану техніку в кошик, він зможе ввести свою адресу, контактні дані, вибрати варіант доставки та оформити замовлення. Менеджер матиме доступ до списку замовлень і зможе змінювати їхній статус (наприклад, в процесі, відправлено, скасовано) та створювати звіти для аналітики.

Щоб і користувачі мобільних пристроїв могли також користуватися платформою, система повинна коректно адаптовуватися під розміри екранів смартфонів чи планшетів.

1.2.2 Визначення нефункціональних вимог. Нефункціональні вимоги визначають фундаментальні характеристики і критерії та функціональність застосунку, такі як ефективність, надійність, безпека, функціональність та інші важливі аспекти. Вони не описують детально конкретні функції системи так як функціональні вимоги, але згадують критерії якості, продуктивності та зовнішніх характеристик, які є дуже важливими для реальних застосувань системи, тобто для її комерційного використання.

Нефункціональні потреби є дуже важливими, для забезпечення ефективності використання програми. Їх відмінність є в тому, що зосереджуються саме на характеристиках та принципах роботи системи, а не на конкретних функціях, які програма повинна виконувати. Це охоплює фактори, що впливають на якість системи загалом та її здатність задовольняти потреби кінцевих користувачів та організацій, які її використовують.

До нефункціональних вимог належать такі, як продуктивність, надійність, масштабованість, безпека, підтримка та сумісність, окрім системної інтеграції. Продуктивність визначає, як швидко система має виконувати операції, особливо коли йдеться про високі навантаження або великі обсяги даних. Надійність означає, що система завжди працює без проблем, що виникає невелика кількість збоїв, які можна за короткий час виправити, що важливо для власників систем та їх бізнесу.

Безпека визначає те, як саме система повинна захищати дані користувачів від незаконного доступу або певних атак. Зручність використання стосується інтерфейсу та зручності взаємодії користувача з програмою, що має безпосередній вплив на ефективність роботи з системою. Масштабованість означає здатність системи обробляти великий обсяг даних, можливість одночасного перебування на сайті великої кількості користувачів без зниження продуктивності.

Вищезгадані вимоги є важливими для створення системи, яка не лише працює, а й ефективно виконує свої функції за різних умов використання, в тому числі в умовах змінюваних навантажень або нестабільних мережевих з'єднань. Тому нефункціональні вимоги є складовою частиною архітектури та дизайну будь-яких програмних додатків.

Нефункціональні вимоги до розроблюваної програмної системи для продажу агротехніки визначають якість роботи та характеристики системи, які не пов’язані безпосередньо з виконанням її основних функцій, але є важливими для її зручності та подальшого використання.

Окрім основних функцій, система продажу агротехніки має відповідати певним **нефункціональним вимогам**, які визначають її продуктивність, безпеку та зручність використання.

- Зручний інтерфейс: користувачі мають мати змогу легко взаємодіяти з системою без спеціальних знань та навичок, тобто всі елементи мають бути інтуїтивно зрозумілими.
- Адаптивність: сайт має правильно відображатися на різних пристроях, тобто розміри мають адаптовуватися під будь який гаджет включаючи мобільні телефони та планшети.
- Безпека: захист даних користувачів.
- Масштабованість: можливість подальшого розширення функцій, вдосконалення поточних, додавання нових, збільшення навантаження при зростанні кількості користувачів та інтеграції з іншими системами.

Визначенні функціональні та нефункціональні вимоги зможуть гарантувати значну якість роботи системи, її безпеку та продуктивність. Подальші етапи розробки включатимуть детальне проектування архітектури системи, розробку прототипу та тестування ключових функцій.

1.3 Моделювання предметної області

1.3.1 Загальні відомості. Моделювання предметної області є наступним не менш важливим етапом у процесі розробки програмного забезпечення після визначення вимог функціональних та нефункціональних, оскільки воно дозволяє задати чіткі межі знань про систему, забезпечуючи уявлення про її структуру та функціональні можливості. У межах даного етапу визначаються основні сутності, їхні характеристики, атрибути, методи та взаємозв'язки між ними, це дозволить створити цілісну і логічну модель системи.

Моделювання відіграє велику роль у забезпеченні аналізу вимог, дозволяючи розробникам, аналітикам та усім хто буде брати участь в створенні системи краще зрозуміти потреби кінцевих користувачів та врахувати всі можливі сценарії використання платформи. Завдяки даному етапу можна оптимізувати процес розробки, і також мінімізувати ризики помилок в майбутньому, оскільки це можна виявити ще на ранніх стадіях проекту.

Моделювання предметної області забезпечує:

- Чітку структуру даних, дозволяючи вибрати та спроектувати архітектуру системи, яка буде ефективно працювати з інформацією.
- Опис бізнес-процесів, допомагаючи уникнути помилок при розробці та впровадженні програми, оскільки всі процеси детально описані та узгоджені між собою.
- Комунікацію між учасниками проєкту, що в свою чергу сприяє спільному розумінню системи серед розробників, тестувальників і замовників, це значно зменшує ризик виникнення помилок.
- Можливість масштабувати та розширювати програмну систему, для цього враховує можливі майбутні зміни в бізнес-процесах і мінімізує ризики, які можуть виникнути при масштабуванні чи розширенні програми.

При моделюванні предметної області часто використовують уніфіковану мову моделювання. Раніше існувало декілька підходів до побудови моделей, це, у свою

чергу, спричиняло певні труднощі у взаєморозумінні між учасниками команди розробників та замовниками. Щоб цей процес став стандартизованим, було створено UML (уніфіковану мову моделювання). Така мова моделювання використовується в усьому світі. Вона містить необхідний набір діаграм і графічних елементів та позначень, що дозволяє схематично зобразити структуру та поведінку системи загалом, або її окремих модулів, зробити її зрозумілою для всіх учасників проєкту, що позитивно сприятиме на процес розробки.[3]

Моделювання предметної області - це складний, але дуже важливий процес, що вимагає системного підходу та дотримання низки фундаментальних принципів. Вони допомагають забезпечити точність, логічність та узгодженість моделі, що, у свою чергу, сприяє її ефективному використанню для виконання поставлених завдань.

Одним із ключових принципів є принцип абстрагування, який дозволяє зосередитися лише на основних аспектах об’єкта чи процесу, що мають важливе значення для вирішення конкретної задачі, відкидаючи другорядні, менш важливі деталі. Такий принцип спрощує аналіз та сприйняття системи. Наприклад, у бізнес-процесі обробки замовлення важливими аспектами є: статус, взаємодія з клієнтом і терміни виконання, а ось розташування складу може бути несуттєвим. Модульність – ще один важливий принцип, що передбачає поділ системи на незалежні компоненти, кожен із яких виконує чітко визначену свою роль. Завдяки такому поділу можна робити оновлення або зміни в окремих модулях без порушення роботи всієї системи.

Також не менш важливу роль відіграє принцип ієрархічності, який організовує структуру системи у багаторівневому вигляді. Це допоможе створити більш впорядковану взаємодію між елементами, а також розподілити відповідальність. Формалізація - ще один важливий принцип, який ґрунтується на використанні строгих правил, певних нотацій та методів для моделювання.

Уніфікація - принцип моделювання предметної області. Уніфікація забезпечує узгодженість між собою моделей завдяки використанню загальновизнаних, стандартних підходів. Це дуже важливо при поєднанні різних систем, так як усім відомі стандарти спрощують їхню взаємодію. Для прикладу, при веб-розробці використання HTML, CSS і JavaScript гарантує коректну роботу сайту у будь-яких браузерах та на різних пристроях.

Застосування вище згаданих принципів моделювання предметної області дозволяє створювати моделі, які будуть зручними для використання, а також для розширення та підтримки в майбутньому.

На основі вищезгаданого можна зробити висновок, що UML є універсальною мовою для моделювання програмних систем при аналізі предметної області. Вона забезпечує найбільш стандартизований підхід до аналізу, дозволяючи учасникам проекту легко взаємодіяти між собою та документувати всі необхідні аспекти розроблюваної системи.

У версії UML 1.5 визначено дванадцять типів діаграм, які розподіленні на три основні групи залежно від їхнього призначення:

- Діаграми, що зображують статичну структуру додатка. Вони описують будову системи, її основні компоненти та їхню взаємодію між собою.
- Діаграми, що зображують поведінку системи. Вони відображають динамічні аспекти додатка, моделюючи процеси та сценарії взаємодії. Такі діаграми допомагають зрозуміти, як саме система повинна поводитися в різних ситуаціях.
- Діаграми, що відображають фізичні аспекти функціонування системи. Вони демонструють організацію системи на архітектурному рівні та розміщення компонентів на фізичному рівні, тобто на рівні обладнання.

У більшості випадків немає потреби створювати всі ці діаграми під час проектування, аналізу чи розробки системи - будуються лише ті, що справді

необхідні. У процесі моделювання предметної області для програм найбільш важливими є такі діаграми:

- Діаграма варіантів використання - показує що може робити той чи інший користувач системи.
- Діаграма послідовності - показує як саме взаємодіють об’єкти між собою.
- Діаграма діяльності - описує як саме виконуються бізнес-процеси та алгоритми.

Далі детальніше розглянемо діаграми, які використовувалися під час моделювання предметної області для системи онлайн-продажу агротехніки. Ми проаналізуємо, як вони допоможуть зрозуміти структуру та логіку роботи системи, які бізнес-процеси відображають і як сприяють ефективній розробці. Це продемонструє, як користувачі взаємодіють з програмною системою, як обробляються запити від них під час виконання бізнес-процесів.

1.3.2 Діаграма прецедентів. Діаграма прецедентів є одною з основних діаграм при моделюванні в UML. Вона використовується для того, щоб описати функціональні можливості системи, показує які дії може робити той чи інший користувач програмного додатку. Основна мета цієї діаграми - показати взаємодію акторів, тобто користувачів системи або інших систем із розроблюваною системою, відображаючи її функціональні можливості.

Основним елементом діаграми прецедентів є актори - користувачі, інші системи або пристрої, які взаємодіють із поточною системою. В програмному додатку для продажу агротехніки це, наприклад, "Користувач", який здійснює покупки, та "Адміністратор", який керує каталогом техніки. Акторів на діаграмі зазвичай позначають фігурками людини для кращого візуального сприйняття.

Ще одним елементом діаграми є прецедент - дія або процес системи. Графічно це зображується у вигляді еліпсів із відповідними назвами. Наприклад, у тому ж

онлайн-магазині прецедентами можуть бути "Оформлення замовлення" чи "Додавання товару до кошика". Кожен актор може взаємодіяти з відповідними прецедентами через зв’язки. Найпростішим зв’язком є асоціація, яка позначається простою лінією між актором і прецедентом, це означає такий актор може виконувати таку дію в системі.

Також існують більш складні зв’язки. Зв’язок включення використовується, коли один прецедент обов’язково містить інший, наприклад, "Здійснити платіж" включає "Перевірку балансу". Це дозволяє включати одні прецеденти в інші. Зв’язок розширення на відміну від включення використовується для необов’язкових дій, що залежать від певних умов. Для прикладу, "Оформлення замовлення" може мати розширення "Застосування знижки", якщо покупець має можливість її застосувати.

Також існує зв’язок генералізація. Він дозволяє об’єднувати схожих акторів або прецеденти у спільні групи. Для прикладу, у банківській галузі "Фізична особа" та "Юридична особа" можуть бути підтипами актора "Клієнт", це допомагає уникнути зайвого дублювання зв’язків і зробити діаграму компактнішою та зрозумілішою.

Діаграма прецедентів є важливим інструментом для аналізу вимог на початкових стадіях проєкту. Вона допомагає визначити користувачів системи, її функції та їхню взаємодію. Діаграма дозволяє ефективно комунікувати усім учасникам проекту, а також дозволяє, ще навіть до початку реалізації, передбачити можливі сценарії використання, що дуже важливо при розробці складних систем.

Загалом, використання діаграми прецедентів полегшує та покращує процес розробки, оскільки вона дає змогу ще навіть до написання коду зрозуміти, як саме система буде працювати в реальних умовах. Це дуже важливо для складних інформаційних систем, де необхідно продумати багато сценаріїв використання, щоб уникнути неочікуваних проблем під час реалізації.

У межах цієї дипломної роботи розробляється інформаційна система, призначена для продажу сільськогосподарської техніки для фермерських

господарств. Діаграма прецедентів для цього програмного продукту представлена у додатку А. Основними користувачами системи є покупець та менеджер системи, кожен з них може виконувати притаманні лише йому дії. Опис акторів та дій, які вони можуть виконувати для предметної області системи продажу агротехніки онлайн наведено в табл. 1.1.

Опис акторів

Таблиця 1.1

Актор	**Короткий опис**
Покупець	Один з користувачів системи, який може переглядати каталог агротехніки, додавати необхідну техніку в кошик та оформлювати замовлення. Він також може при оформлені замовлення вказувати спосіб оплати та доставки, відстежувати статус свого замовлення, переглядати історію попередніх покупок та порівнювати характеристики техніки між собою.
Менеджер системи	Інший користувач системи, що є відповідальним за управління каталогом, редагування інформації про техніку і створення та редагування категорій. Він також переглядає та оброблює замовлення, може змінювати їх статуси, також формує звіти для аналітики та може взаємодіяти з покупцями для консультацій, порад та вирішення питань.

Покупець, або ж клієнт онлайн-магазину є основним користувачем платформи, оскільки він спочатку здійснює пошук техніки, потім оформлює замовлення. Основні дії, які він може виконувати включають:

- Перегляд каталогу, тобто покупець може ознайомитися з наявною агротехнікою, переглядаючи її вартість, категорію, модель та інші характеристики.
- Додавання товарів до "кошику", тобто може вибрати бажану техніку та зберігати її у своєму кошику, щоб в майбутньому оформити замовлення.

- Керування кошиком, тобто покупець має можливість змінити кількість одиниць техніки, видалити непотрібні або ж додавати нові.
- Прівняння обладнання між собою, це допомагає користувачеві обрати найкращий варіант, порівнюючи характеристики, вартість.
- Фільтрування техніки, тобто є можливість відсортувати усі товари за певними заданими критеріями, такими як вартість, бренд, категорія тощо.
- Оформлення замовлення, що є основною дією користувача, для покупки техніки. Воно відбувається так, що після вибору бажаного обладнання покупець може, вказавши необхідні дані, оформити замовлення.
- Вказання способу оплати, тобто система дозволяє покупцям обрати метод оплати: карткою, банківським переказом, готівкою при отриманні або інший зручний для користувача.
- Вказання способу доставки, тобто покупець вибирає спосіб одержання замовлення, це може бути доставку кур'єром або самовивіз.
- Відстеження статусу замовлення доступне після оформлення замовлення в особистому кабінеті користувача.

Менеджер платформи є відповідальним за адміністрування системи, наповнення її новими оголошеннями, зв’язок з клієнтами та управління замовленнями. Його дії наступні:

- Розміщує інформації про техніку, тобто менеджер додає позиції нової техніки до каталогу, вказуючи її характеристики та вартість, завантажуючи зображення.
- Редагує наявну техніку, тобто за потреби менеджер може оновлювати інформацію про раніше додану техніку, змінюючи її характеристики, вартість або, що найчастіше буде змінюватися, статус наявності.
- Редагує категорії техніки, система дає можливість змінювати структуру каталогу, додаючи, видаляючи або змінюючи категорії товарів.

- Змінює статуси замовлень, тобто коли менеджер обробляє замовлення від покупців, він може оновлювати статуси заявок і повідомляти про це клієнтам.
- Формує звітні документи, тобто платформа дозволяє генерувати аналітичну інформацію про продажі, кількості замовлень, найпопулярнішу техніку або категорію тощо.
- Переглядає звітність, тобто менеджер має можливість аналізувати звітні дані, для прийняття рішення щодо подальшого розвитку платформи або управління нею.
- Взаємодіє з потенційними покупцями, може спілкуватися з клієнтами, відповідати на їхні запитання, щоб допомогти користувачу з вибором бажаного обладнання або навіть вирішувати можливі проблеми покупців.

1.3.3 Діаграма послідовності. Діаграма послідовності - це ще один із типів діаграм, який існує в уніфікованій мові моделювання та використовується для відображення взаємодії між об'єктами системи в межах певного сценарію у плинні часу. Така діаграма показує, які саме об'єкти беруть участь у певному процесі, які повідомлення вони надсилають між собою та в якому саме порядку. Це дозволяє розробникам та аналітикам або й навіть іншим членам команди зрозуміти не лише взаємодію між об’єктами системи, а й точний порядок виконання операцій, що дуже важливо для аналізу складних бізнес-процесів.

Основною особливістю такої діаграми є те, що вона має часову шкалу. Час на діаграмі послідовності рухається зверху вниз, тобто усі повідомлення, які розташовані вище, відбуваються раніше ніж ті, що розташовані нижче. Це дозволяє чітко визначити порядок взаємодії об’єктів між собою та побачити, які події спричиняють ті чи інші реакції у системі. Завдяки такій чіткій структурі можна з

легкістю визначити взаємозалежність між операціями та виявити можливі проблеми, що пов’язані з їх послідовністю.[4]

Кожен елемент, що бере участь у взаємодії, зображується на такій діаграмі у вигляді вертикальної лінії, яка називається життєвою лінією. Вона показує, скільки часу об’єкт існує у процесі виконання певного сценарію. Коли об’єкт є активним, тобто виконує якусь операцію або надсилає запит чи очікує на відповідь, його життєва лінія позначається прямокутником, він вважається активною областю. Така область вказує на проміжок часу, протягом якого об’єкт існує та бере активну участь у виконанні бізнес-процесу, це дозволяє нам чітко визначити, коли саме він задіяний у процесі.

Важливою частиною діаграми є повідомлення, якими обмінюються об'єкти між собою. Вони представлені стрілками, що йдуть від одного об'єкта до іншого. Повідомлення можуть бути синхронними або асинхронними. Синхронне повідомлення означає, що об'єкт, що надсилає його, чекає відповіді, перш ніж продовжувати робити щось далі. Зазвичай це відбувається під час викликів методів у класах коду програми. Асинхронні повідомлення, з іншого боку, не потребують негайної відповіді, що дозволяє іншим операціям виконуватися без затримок.

Діаграми послідовності також забезпечують здатність зображувати умовні та альтернативні сценарії. Якщо процес може розгалужуватися на різні шляхи в певний момент, це можна показати за допомогою спеціальних фрагментів, які позначають альтернативи або петлі.

На додаток до взаємодій між існуючими об'єктами, діаграми послідовностей також можуть показати створення, а потім знищення об'єктів. Якщо під час виконання певного процесу з’являється новий об’єкт, він представлений пунктирною стрілкою до його життєвої лінії. Руйнування об'єкта позначається "X" в кінці його життєвого шляху.

Діаграма послідовності є незамінним інструментом для розробки складних програмних рішень, де важливо правильно визначити порядок виконання операцій.

Це допомагає розробникам, бізнес-аналітикам та тестерам детально розуміти системні процеси та перевірити їх правильність на етапі проектування. Використовуючи його, команди можуть не лише задокументувати логіку виконання процесів, але й визначити потенційні проблеми, тим самим покращуючи якість розробленого програмного забезпечення.

У додатку Б зображено діаграму послідовності, вона демонструє основний потік повідомлень для покупця та менеджера в системі онлайн продажу агротехніки, від пошуку потрібної техніки до оформлення замовлення, тобто завершення покупки.

1.3.4 Діаграма активності. Діаграма активності - це тип діаграми в UML, яка використовується для моделювання алгоритмів дії в процесах. Вона представляє потік процесів та даних між різними етапами роботи програми. Завдяки цьому, діаграми активності широко використовуються для аналізу використання сценаріїв, розробки складних алгоритмів та поведінки системи.[3]

Ключовою особливістю цієї діаграми є її здатність представляти потік виконання як послідовність дій, пов'язаних стрілками. Вона має чітко визначену часову шкалу, де дії розташовані зверху вниз або зліва направо, а стрілки вказують напрямок виконання процесу. Це дозволяє легко відстежувати те, що відбувається в певний момент і які кроки слідують один одному.

Діаграма діяльності складається з декількох основних елементів. Найважливішими є вузли дії, які представляють конкретні операції. Вони зображені як прямокутники з закругленими кутами. Початок процесу позначається початковим вузлом (чорним колом), тоді як кінець позначається кінцевим вузлом (чорним колом із зовнішнім кільцем). Потік виконання між діями представлений стрілками, що показує послідовність кроків.

Однією з ключових можливостей діаграми діяльності є її підтримка умовного розгалуження. Це дозволяє моделювати ситуації, коли процес може пройти різні шляхи залежно від певних умов. Для цього використовуються вузли рішення (представлені як ромбики).

Крім того, діаграми активності підтримують паралельне виконання дій, що особливо корисно для моделювання паралельних або асинхронних процесів. Для цього використовуються роздільні та об'єднувальні вузли (зображуються як товсті горизонтальні або вертикальні смуги) використовуються для показу, де процес розбивається на кілька паралельних потоків і де ці потоки потім з'єднуються в один перед продовженням виконання.

Завдяки своїй гнучкості та візуальній чіткості, діаграма діяльності слугує потужним інструментом для моделювання складних процесів. Це не тільки допомагає описати логіку системи, але й забезпечує оптимізацію бізнес-процесів, ідентифікацію потенційних операційних проблем та вдосконалення взаємодії компонентів. Використання даної діаграми дозволить зробити додаток більш передбачуваним, структурованим та ефективним.

У додатку В зображено діаграму активності для системи продажу агротехніки. Вона показує алгоритм дій, які відбуваються під час покупки обладнання, включаючи взаємодію між покупцем, магазином та адміністрацією для успішного завершення замовлення.

1.4 Огляд інформаційних джерел та існуючих рішень

Онлайн-ринок сільськогосподарської техніки швидко розвивається, пропонуючи широкий спектр цифрових рішень для купівлі та продажу сільськогосподарського обладнання. Сучасні онлайн-платформи надають зручні для користувачів інструменти, що спрощують процес пошуку, порівняння та придбання техніки. Ці послуги об'єднують велику кількість продавців та покупців, що

дозволяє користувачам швидко знайти необхідне обладнання та проводити безпечні транзакції.

Для аналізу та визначення ключових вимог до розробки системи, було проведено порівняльне дослідження трьох популярних онлайн-платформ продажу сільськогосподарської техніки: Tractorhouse, Machinery Pete та Agroserver. Під час дослідження оцінювалися їх основні функціональні можливості, структуру каталогу, досвід користувачів та інші важливі аспекти. Порівняльний аналіз цих платформ представлено в табл. 1.2.

Порівняльний аналіз існуючих рішень

Таблиця 1.2

Платформа	Загальні можливості	Переваги	Недоліки
Tractor House	Міжнародний маркетплейс для продажу тракторів, комбайнів, сівалок та іншої техніки. Підтримує детальну фільтрацію за брендом, потужністю, типом приводу тощо.	Велика база техніки, функціонал детальної фільтрації.	Відсутня локалізація для українського ринку, що ускладнює використання.
Machinery Pete	Продаж вживаної та нової агротехніки з базою даних про ціни на вторинному ринку. Доступна система рейтингів продавців.	Висока довіра до продавців завдяки рейтинговій системі, інформація про вторинний ринок.	Основна орієнтація на ринок США, що обмежує можливості українських користувачів.
AgroServer	Українська онлайн-платформа для продажу агротехніки, запчастин і обладнання. Простий процес реєстрації для продавців.	Локалізований для українського ринку, проста реєстрація.	Відсутність гнучкої системи пошуку та рекомендацій, що ускладнює навігацію.

Основні функціональні можливості таких платформ включають детальну фільтрацію техніки за характеристиками, інтегрованими рейтингами та оглядами, аналітичними інструментами для оцінки тенденцій ринку, а також підтримки онлайн-замовлень та платежів. Деякі платформи також пропонують додаткові функції, такі як автоматизовані рекомендації щодо обладнання на основі потреб користувачів, можливість домовлятися про ціни та консультації з фахівцями.

Аналіз існуючих онлайн-платформ для продажів сільськогосподарського обладнання показав, що більшість міжнародних платформ, таких як Tractorhouse та Machinery Pete, пропонують розширену функціональність, зручний для користувачів інтерфейс та підтримку сучасних алгоритмів пошуку та рекомендацій. Однак вони не пристосовані до українського ринку, що ускладнює процес купівлі та продажу для місцевих приватних фермерів та підприємств.

З іншого боку, місцеві рішення, такі як Agroserver, краще пристосовані до потреб українських користувачів, але мають значні функціональні обмеження. Зокрема, їх пошукові системи не завжди забезпечують гнучкі фільтри, а відсутність повноцінної системи рейтингу зменшує довіру до продавців.

Розробляючи власне рішення, варто розглянути розширену систему фільтрації, щоб покупці могли швидко знайти необхідну техніку відповідно до своїх потреб. Необхідна локалізація для українського ринку, крім того, слід враховувати масштабованість на міжнародний рівень для розширення аудиторії та підвищення конкурентоспроможності.

Врахування цих аспектів, допоможе створити сучасну, зручну та ефективну платформу, яка поєднує переваги міжнародних рішень з потребами українського ринку, забезпечуючи простий процес купівлі та продажу сільськогосподарської техніки.

1.5 Постановка завдання

На основі аналізу предметної області, визначення вимог, моделювання процесів та огляду існуючих подібних рішень було розроблено наступні технічні специфікації для створення системи для онлайн продажу сільськогосподарської техніки.

Система призначена для забезпечення фермерів та приватних сільськогосподарських підприємств зручним доступом до широкого спектру техніки, включаючи трактори, комбайни, плуги, культиватори та інше обладнання. Це дозволяє користувачам швидко знаходити, порівнювати та купувати необхідне обладнання через сучасний веб-інтерфейс. Основна мета - автоматизувати процес вибору, порівняння, замовлення та управління продажами сільськогосподарських машин.

Для зручності користувачів будуть додані спеціальні розділи, включаючи "про нас", "методи доставки" та "категорії", де користувачі можуть знайти корисну інформацію про компанію, умови доставки та доступні категорії техніки. Каталог продуктів має широкий асортимент сільськогосподарських машин, таких як трактори, комбайни, плуги, культиватори та інше обладнання.

Основним користувачем системи є клієнт, що переглядає каталог техніки, здійснює покупки, переглядає історію замовлень. Менеджер або адміністратор магазину керує контентом, обробляє замовлення, управляє категоріями техніки, формує звіти для аналітики.

Функціонал для клієнтів містить реєстрацію та авторизацію, що забезпечує можливість створення облікового запису. Користувачі мають доступ до каталогу техніки, де кожен товар представлений у вигляді картки з детальним описом, характеристиками та фото. Для зручного пошуку реалізовано фільтрацію за категоріями, вартістю та іншими технічними характеристиками техніки.

Процес закупівлі реалізується через кошик для покупок, де користувачі можуть коригувати кількість одиниць потрібного обладнання, а також функцію порівняння техніки між собою, яка дозволяє вибирати декілька машин та

порівнювати їх технічні характеристики в зручній таблиці. Створення замовлення передбачає введення контактних даних, вибір методу доставки та вибору способу оплати. Покупці можуть переглянути свою історію замовлення та відстежувати статус їхніх покупок на різних етапах обробки.

Менеджери мають розширені можливості управління каталогами. Вони можуть додавати нове обладнання, редагувати існуючі, оновлювати статуси наявності або видалити застарілу техніку. Управління категоріями включає створення та модифікацію груп продуктів. Крім того, адміністратор має доступ до панелі замовлень, де він може переглянути всі наявні замовлення від користувачів, оновлювати їх статуси та спілкуватися з клієнтами для уточнення деталей.

Система також підтримує генерацію звітів з аналітичними даними про продажі протягом конкретних періодів, що дозволяє відстежувати діяльність покупців, оцінювати попит та коригувати діапазон продукції відповідно до тенденцій ринку. Нефункціональні вимоги включають створення зручного, зрозумілого інтерфейсу, який забезпечує коректну взаємодію як на ПК, так і на мобільних пристроях.

Очікуваний результат - це повністю функціональна система продажів сільськогосподарської техніки онлайн, яка автоматизує всі ключові процеси - від вибору продукту до оформлення замовлення.

1.6 Висновки до розділу 1

Проведений системний аналіз предметної області дозволив чітко визначити основні вимоги до розроблюваної програмної системи продажу агротехніки онлайн. Були розглянуті особливості функціонування сільськогосподарських підприємств, ключові бізнес-процеси такі, як купівля та обслуговування техніки, а також специфіка використання онлайн-платформ для автоматизації цих процесів. Аналіз показав, що сучасні фермерські господарства потребують ефективних рішень для

оптимізації процесів продажу агротехніки, що включає можливість швидкого порівняння моделей, управління замовленнями та зменшення витрат на логістику. Окрім того, було визначено, що автоматизація бізнес-процесів та впровадження цифрових рішень є ключовими факторами для підвищення ефективності аграрних підприємств. Врахування цього створює фундамент для подальшого проектування інформаційної системи, що відповідає потребам користувачів.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

2.1.1 Загальні відомості про ER-діаграму. Логічна модель даних - це абстрактне представлення структури даних у конкретній предметній області. Вона є незалежною від технологій реалізації та розробки і використовується під час проєктування моделі даних для того, щоб сформувати правильну структуру зберігання даних. Основною особливістю логічної моделі даних є її незалежність від систем управління базами даних (СУБД) або фізичної організації даних, тобто дані представляються лише на логічному рівні.

Одним із найпоширеніших способів представлення логічної моделі даних є ER-діаграма (на англ. Entity-Relationship diagram). Вона використовується для представлення структури даних перед їх реалізацією в тій чи іншій базі даних.

ER-діаграма є одним з найважливіших інструментів для проектування та моделювання баз даних. Така діаграма дозволяє представити структуру даних та взаємозв'язки між сутностями, що допоможе розробникам зрозуміти, як саме буде організована інформація в системі. ER-діаграми зазвичай використовуються на під час проектування для того, щоб створити логічну модель даних, яка потім переходить у фізичну модель для реалізації вже в конкретних системах управління базами даних (СУБД).

Основним елементом ER-діаграми є сутності - це головні об’єкти предметної області, про які необхідно зберігати інформацію. Наприклад, у системі онлайн-продажу даними сутностями будуть «Користувач», «Товар», «Замовлення» тощо. Кожна з них може представляти реальний або концептуальний об’єкт, що має певний набір притаманних йому характеристик.[5]

Кожна сутність має атрибути, тобто певні властивості, які визначають її характеристики. Для прикладу, сутність «Користувач» може мати такі атрибути, як

«Ім’я» та «Електронна пошта», сутність «Товар» відповідно: «Назва» та «Ціна», а «Замовлення» - «Дата оформлення» та «Загальна сума». Атрибути показують інформацію про сутності та роблять їх опис більш точним. Атрибути бувають ключові та неключові.

Між сутностями існують зв’язки, які показують, як показують. Як сутності взаємодію між собою. Для прикладу, сутність «Користувач» може бути пов’язана із сутністю «Замовлення» через зв’язок «оформлює», це означає, що користувач може оформляти замовлення. Так же само, «Замовлення» може бути пов’язане з сутністю «Товар», так як кожне замовлення містить певний перелік товарів.

ER-діаграми є дуже корисними для того щоб аналізувати та оптимізовувати структури баз даних. Вони дозволяють визначити, як саме дані мають бути організовані в системі, яким чином вони будуть зберігатися та як одні елементи даних будуть взаємодіяти іншими. Також ще на етапі розробки таких діаграм можна визначити потенційні проблеми та одразу їх вирішити. Так як ER-діаграми є візуальним представленням, це робить їх простими для розуміння навіть для тих, хто не має досвіду в роботі з базами даних, це в свою чергу сприяє кращій комунікації між замовниками, розробниками та іншими учасниками проекту.

Після створення ER-діаграми ще на етапі логічного проектування, їх можна використати для побудови фізичної моделі бази даних, що включає вже безпосередню реалізацію таблиць, індексів, зв'язків та інших елементів, необхідних для ефективного зберігання та управління даними в СУБД. Це дозволяє створити оптимізовану базу даних, яка забезпечить швидкий доступ до необхідної інформації та підтримки складних операцій з даними.

Модель даних, розроблена за допомогою діаграми ER, також є важливим етапом у виборі відповідної системи управління базами даних. Залежно від структури та типів даних, які потрібно зберігати, також дана діаграма може допомогти визначити, який тип СУБД буде найбільш підходящим для проекту. Наприклад, якщо модель даних вказує на зв’язки між сутностями, які добре

підходять для зберігання в таблицях з чіткими взаємозв'язками, то реляційні СУБД будуть найкращим вибором. Це дозволяє ефективно працювати з таблицями, індексами та взаємозв'язками даних, які є типовими для таких моделей.

У випадках, коли модель вимагає зберігання складних об'єктів, а також підтримки для обробки мультимедійних або неструктурованих даних, може бути доцільним використання об'єктно-реляційної СУБД, які підтримують як реляційні, так і об'єктні моделі даних. Це дозволяє поєднувати гнучкість реляційної моделі з перевагами об'єктно-орієнтованого підходу.

У випадках, коли система повинна обробляти великі неструктуровані дані, або коли продуктивність та масштабованість є дуже важливим критерієм, системи управління базами даних, такі як NOSQL, документоорієнтовані або графові можуть розглядатися як альтернативи реляційним. Моделювання даних на основі ER-діаграми допомагає визначити, чи підходить реляційна модель для конкретного проекту, чи можливо альтернативні типи баз даних є більш ефективними для тих вимог та завдань, що були визначенні.

Завдяки їх візуальному та графічному форматі, ER-діаграми покращують роботу в команді серед розробників, аналітиків та інших учасників проекту. Вони дозволяють спільно обговорювати та вдосконалювати структуру даних на етапі планування, зменшуючи ймовірність помилок на пізніших етапах розробки та заощаджуючи час та ресурси проекту. Добре розроблена ER -діаграма є ключовим кроком для успішної реалізації проекту та ефективної взаємодії з базою даних.

2.1.2 Побудова ER- діаграми. При моделюванні бази даних для дипломної роботи було спочатку розроблено ER-діаграму, при цьому використано CA ERwin Data Modeler - потужний інструмент для розробки логічної моделі даних. Дана програма дозволяє створити логічну або фізичну модель даних, що дозволяє побачити візуально зв'язки між об’єктами. ERwin Data Modeler дозволяє відобразити складну структуру даних, контролювати інформаційні ресурси і встановлювати певні стандарти управління даними, притаманні тій чи іншій

компанії. Продукт також дозволяє оптимізувати процес проектування, моделювання, розробки та забезпечує інтеграцію створеної моделі з конкретною СУБД.

Цей інструмент зараз використовують як малі компанії, так і в великі корпорації завдяки тому, що має гнучкість та багато можливостей для управління базами даних на всіх етапах їхнього життєвого циклу, від моделювання до фізичної реалізації.

На рис. 2.1 зображена ER-діаграма, на якій представлено логічну модель даних для платформи онлайн-продажу агротехніки. Вона містить усі необхідні сутності, що забезпечать повноцінне функціонування платформи для користувачів, управління замовленнями, кошиком та технікою. Також вона приведена до 3 нормальної форми.

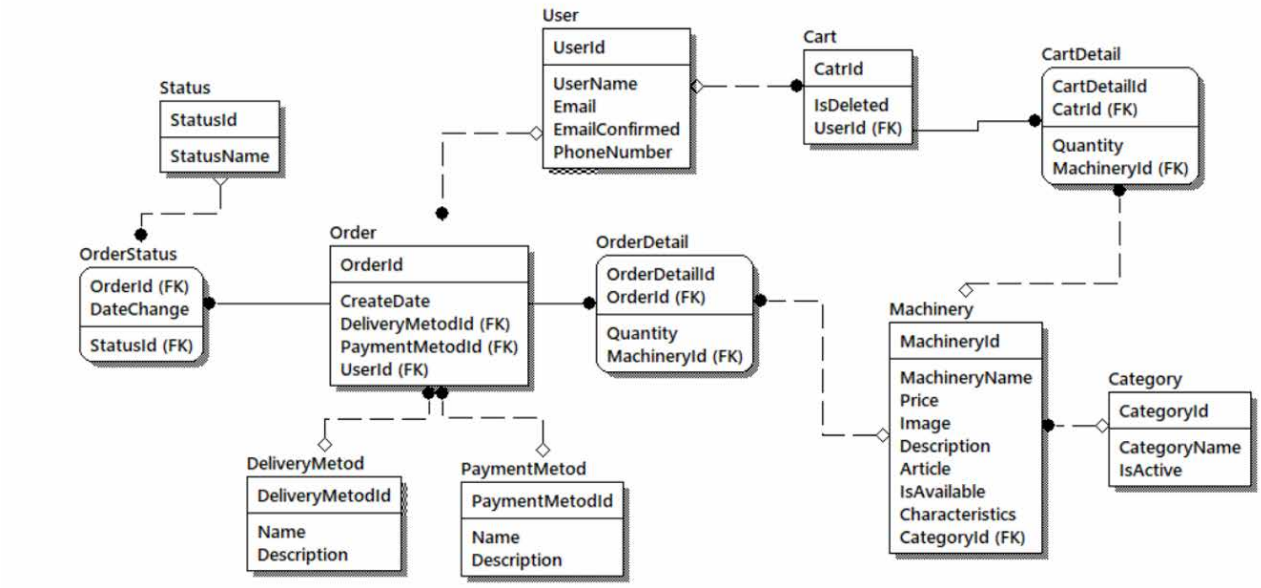


Рис. 2.1 ER-діаграма

Центральною сутністю є Order (Замовлення), вона містить дані про дату створення конкретного замовлення, метод доставки, який обрав користувач, метод оплати та посилання на користувача, який оформив замовлення. Замовлення пов’язане з сутністю OrderDetail (Деталі замовлення), що містить інформацію про

агротехніку, їх кількість і посилання на певне замовлення. Статус замовлення фіксується у таблиці OrderStatus (Статус замовлення), де зазначається зміна статусу та її дата. Доступні статуси замовлення містяться в сутності Status (Статуси).

Користувачі системи зберігаються в сутності User (Користувач), яка містить інформацію про ім’я, електронну пошту та номер телефону. Кожен користувач може мати власний Cart (Кошик), який містить перелік товарів перед оформленням замовлення. Наповнення кошика визначається у таблиці CartDetail (Деталі кошика), що зберігає перелік товарів та їхню кількість.

Система підтримує продаж різних видів агротехніки, яка зберігається у таблиці Machinery (Агротехніка). Тут містяться назва товару, ціна, зображення, опис, артикул, доступність та характеристики. Кожен товар належить до певної Category (Категорія), що містить назву та статус активності.

Методи оплати та доставки представлені окремими сутностями: PaymentMetod (Методи оплати) та DeliveryMetod (Методи доставки), які містять назву та опис відповідного методу.

Загальна структура даних відповідає принципам третьої нормальної форми (ЗНФ), що забезпечує відсутність зайвої надлишковості даних та підтримує цілісність інформації. Використання реляційної СУБД є оптимальним вибором для такої моделі, оскільки вона гарантує ефективне зберігання, зв’язки між таблицями та можливість швидкого доступу до інформації.

2.2 Діаграма класів та кооперацій

Діаграми класів та кооперацій є інструментами для об’єктно-орієнтованого аналізу та проектування. Такі діаграми допомагають зрозуміти структуру та взаємодію компонентів системи, визначаючи, як організовані дані, як об'єкти взаємодіють один з одним та які саме операції вони можуть виконувати.

Діаграма класів показує структуру системи, представлену в об’єктно-орієнтованому вигляді. Клас на діаграмі є абстракцією певного об'єкта або сутності в системі, наприклад, "Користувач", "Техніка" або "Замовлення". Діаграма класів відображує різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти й підсистеми, а також описує їхню внутрішню структуру й типи відносин.

Одним із ключових типів зв’язків у діаграмі класів є асоціація - вона показує, що один клас взаємодіє з іншим. Такий зв’язок може бути одностороннім або двостороннім. Наприклад, клас "Користувач" може бути пов'язаний з класом "Замовлення", що означає: один користувач може мати багато замовлень (1..*). Зворотного зв’язку може не бути - "Замовлення" може не містити інформації про власника, якщо це не критично для системи, але це вже особливості вимог до конкретної системи.

Інший зв’язок - агрегація. Вона визначає відносини "частина-ціле", де один об'єкт є частиною іншого, але ці частини можуть існувати незалежно від цілого. Наприклад, клас "Товар" може бути частиною класу "Кошик". Товари можуть бути частинами кошика, але не залежать від нього, тому що один і той самий товар може бути доданий до кількох кошиків або не бути взагалі в кошику. Відмінністю агрегації є композиція, яка є більш суворою формою агрегації. У композиції частина не може існувати без цілого. Це означає, що об'єкти, які складають композицію, мають сильну залежність один від одного. Це дозволяє моделювати ситуації, де об'єкти не можуть бути незалежними.

Також важливою ознакою класів є спадкування, яке дозволяє одному класу успадковувати властивості та методи іншого. Завдяки цьому можна створювати нові класи на основі існуючих, зберігаючи спільну логіку та додаючи специфічні функції. Наприклад, клас "Адміністратор" може унаслідуватися від класу "Користувач", успадковуючи його атрибути й методи, а також додаючи власні, як-от "додавання товару" чи "управління замовленнями". Спадкування дозволяє

організувати класову ієрархію, де загальні властивості можна винести в базові класи, а конкретні - у похідні.

Діаграма кооперацій, на відміну від діаграми класів, зосереджена на тому, як об'єкти взаємодіють між собою в контексті виконання певних операцій або сценаріїв. Вона описує, як об'єкти обмінюються повідомленнями для досягнення певної мети. Кожне повідомлення вказує на виклик методу, і, залежно від потреб, воно може бути синхронним або асинхронним. У системі онлайн-продажу агротехніки, наприклад, при оформленні замовлення користувач може взаємодіяти з класами "Кошик", "Товар" і "Замовлення". Діаграма кооперацій відображатиме, як ці об'єкти взаємодіють між собою: від додавання товару в кошик до завершення оформлення замовлення. Важливими елементами є також ролі, які об'єкти виконують під час взаємодії, і тимчасова послідовність, яка визначає порядок обміну повідомленнями.

Зв’язки між об'єктами на діаграмі класів і кооперацій є основними елементами для правильного функціонування системи. На діаграмі класів ці зв’язки можуть бути різними: асоціації вказують на те, що об'єкти одного класу можуть мати зв'язок з об'єктами іншого класу; агрегація та композиція підкреслюють ієрархічні відносини між класами; спадкування допомагає створювати спеціалізовані класи з уже існуючих базових. У діаграмі кооперацій зв'язки описують, як об'єкти обмінюються повідомленнями для виконання конкретних функцій у межах процесу, наприклад, оформлення замовлення чи додавання товарів до кошика.

Правильно визначені зв'язки між класами дозволяють збудувати гнучку і ефективну модель системи, яка відповідає вимогам проекту. Вони забезпечують правильну взаємодію між компонентами, що дозволяє уникнути дублювання коду, підвищити модульність і зменшити складність розробки.

У додатку Д представлена діаграма кооперацій, яка показує взаємодію між кооперaціями системи для продажу агротехніки онлайн.

Кооперація користувачів показує взаємодію між усіма користувачами системи. Основою є клас Користувач, від якого наслідуються Адміністратор та Покупець. Адміністратор має розширені можливості керування системою, тоді як покупець може лише переглядати каталог, додавати товари до кошика та оформлювати замовлення. Кооперація кошика пов’язує класи Покупця, Кошик і Техніку. Покупець може наповнювати власний кошик технікою, змінювати їхню кількість або видаляти перед тим, як оформити замовлення. Ця частина системи забезпечує зручний механізм для вибору та замовлення техніки.

Кооперація замовлення включає класи Замовлення, Покупець, Техніка та Адміністратор. Покупець оформлює замовлення, а адміністратор, у свою чергу, стежить за процесом обробки замовлень і змінює їхній статус у системі.

Кооперація управління технікою визначає взаємодію між Технікою, Категорією та Менеджером магазину. Менеджер додає нові одиниці техніки та нові категорії, редагує їхні характеристики, формуючи каталог техніки.

Така структура дозволяє нам чітко визначити ролі користувачів і забезпечує логічний поділ функціоналу системи.

2.3 Діаграма пакетів

Діаграма пакетів є ще одним способом візуального моделювання в UML, який використовується для представлення структури програмної системи на більш високому рівні. Вона відображає, організацію основних компонентів системи у вигляді пакетів, що групують класи, інтерфейси та інші елементи моделі.[3]

Головна мета діаграми пакетів - показати логічний поділ системи на модулі та показати зв’язки між ними. Це дозволить краще зрозуміти архітектуру проекту, визначити залежності між частинами системи та покращити їхню взаємодію. Завдяки такій структурі значно спрощується розробка, підтримка та подальше розширення системи.

Кожен пакет у діаграмі є певним контейнером, що може містити інші пакети або окремі класи. Це допомагає показати організацію складної системи, поділяючи її на більш зрозумілі частини. Наприклад, у великій програмній системі можна виокремити пакети для управління користувачами, обробки платежів, роботи з даними тощо.

Зв’язки між пакетами показують, як одні компоненти системи залежать від інших. Найпоширенішим типом зв’язку є залежність, яка означає, що один пакет використовує або взаємодіє з іншим. Таке відображення допомагає уникнути зайвих зв’язків, визначити потенційні проблеми із залежностями та покращити модульність системи.

На рис. 2.2 представлено діаграму пакетів, яка відображає архітектуру системи продажу агротехніки онлайн, поділену на певні рівні та пакети. Така структура допомагає забезпечити модульність, що покращує розробку, тестування та підтримку системи в майбутньому.

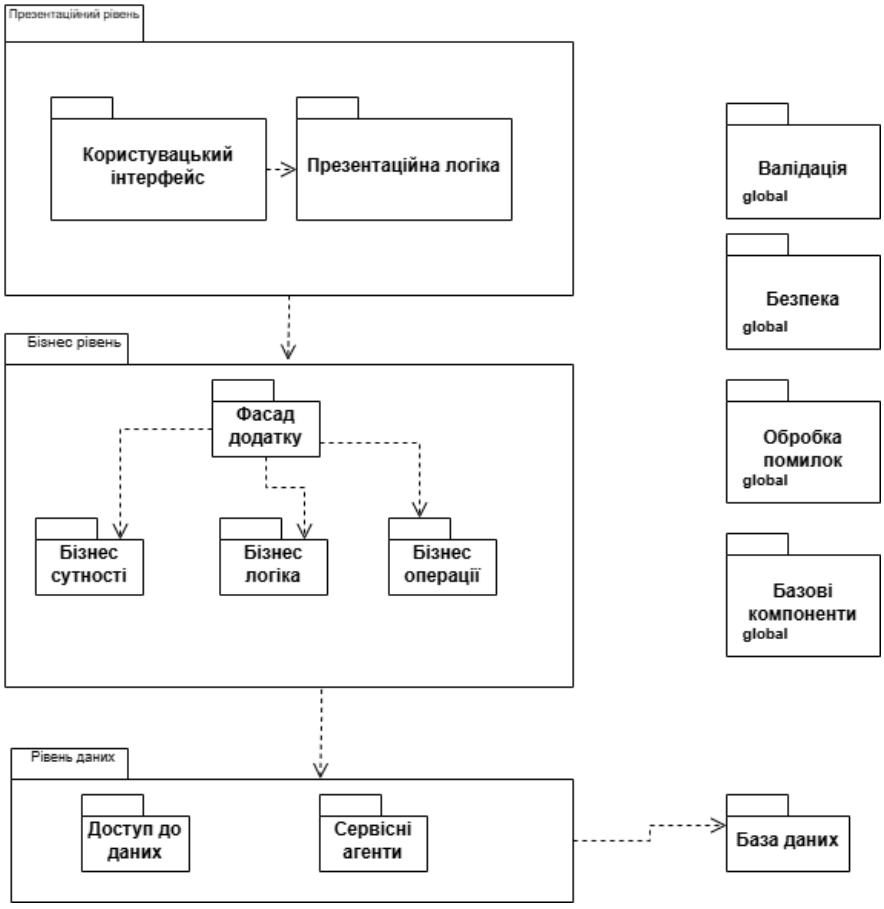


Рис. 2.2 Діаграма пакетів

Презентаційний рівень описує користувацький інтерфейс, з яким безпосередньо взаємодіє кінцевий користувач. Цей рівень також містить презентаційну логіку, яка відповідає за відображення даних та обробку запитів користувача. Це перший рівень, через який користувач взаємодіє з системою.

Бізнес рівень вже показує реалізацію основної функціональності системи. Він містить фасад додатку, який надає загальний інтерфейс для доступу до бізнес-логіки. На даному рівні також знаходяться бізнес-сутності та операції, що виконують основні функції системи, такі як обробка замовлень, керування користувачами та технікою. Рівень даних відповідає за доступ до збережених даних та управління ними. Цей рівень містить сервісні агенти та базу даних, які забезпечують зберігання даних та їх обробку, це дозволяє системі продуктивно обробляти інформацію, що зберігається. Глобальні пакети містять усі потрібні

сервіси, що забезпечують цілісність та безпеку даних на кожному з рівнів системи. Такі пакети містять валідацію даних, обробку помилок, а також засоби для забезпечення безпеки користувачів та системи в цілому.

Представлена архітектура у вигляді діаграми пакетів дозволяє чітко зобразити функціональні блоки системи, кожен з яких має свою роль, а разом це все забезпечує ефективність, масштабованість і безпеку розроблюваної системи.

2.4 Діаграма компонентів

Діаграма компонентів є важливим інструментом для моделювання програмного забезпечення, що відображає архітектуру системи через її модулі та взаємодії між ними. Вона показує фізичну структуру програмної системи, визначаючи складові частини та спосіб їхнього об’єднання. Головною метою є демонстрація взаємодії компонентів, інтерфейсів, які вони використовують, і способу зв'язку між частинами системи.

Компоненти - це незалежні частини, які можуть бути повторно використані чи замінені без значних змін. Вони можуть бути модулями, бібліотеками, веб-сервісами чи зовнішніми системами. Інтерфейси визначають взаємодію між компонентами, дозволяючи організувати систему так, щоб елементи могли використовувати функціонал один одного без знання внутрішньої реалізації.

Діаграма компонентів відіграє важливу роль у розробці програмного забезпечення, оскільки дозволяє графічно представити структуру системи, що полегшує командну роботу і визначає межі відповідальності кожного компонента. Це забезпечить кращу масштабованість системи у майбутньому, зручність у розгортанні та призведе до зниження залежностей між модулями.

У процесі проєктування програмного забезпечення для платформи продажу агротехніки онлайн була створена діаграма компонентів, яка відображає компоненти та їх взаємодію у системі. Вона допомагає створити ефективний та гнучкий додаток,

оптимізуючи зв’язки між компонентами й закладаючи основи для масштабованої архітектури. Діаграма зображена на рис. 2.3.

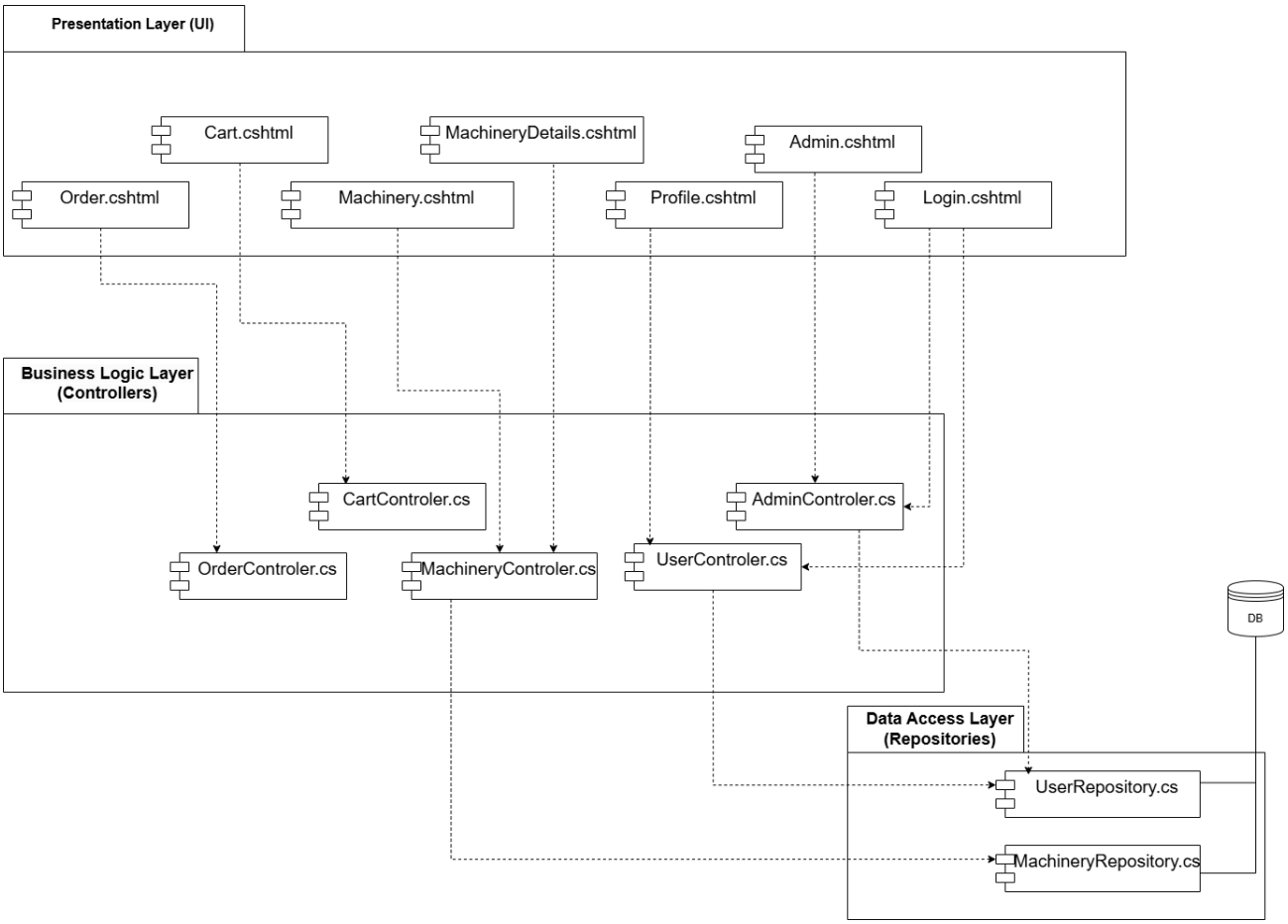


Рис. 2.3 Діаграма компонентів

Діаграма компонентів, що була розроблена при проектуванні системи продажу агротехніки онлайн, ілюструє основні структурні елементи платформи та їх взаємодію між собою. Вона складається з трьох основних рівнів: рівня представлення (UI), рівня бізнес-логіки (Controllers) та рівня доступу до даних (Repositories). Така архітектура забезпечить модульність, масштабованість та підтримуваність системи в майбутньому, що є дуже важливо для роботи платформи продажу техніки онлайн.

На рівні представлення розташовані основні компоненти, що відповідають за взаємодію з користувачем, це якби інтерфейс системи, те що буде бачити користувач. Серед компонентів на даному рівні будуть такі, як Cart.cshtml (корзина),

Machinery.cshtml (каталог техніки), MachineryDetails.cshtml (детальна інформація про техніку), Order.cshtml (оформлення замовлення), Profile.cshtml (профіль користувача), Admin.cshtml (панель адміністратора) та Login.cshtml (сторінка авторизації). Ці компоненти забезпечують користувачам можливість переглядати техніку, оформлювати замовлення та взаємодіяти із системою відповідно до їхніх ролей та потреб.

Бізнес-логіка платформи реалізована через контролери, які опрацьовують запити від користувачів та передають необхідні дані між рівнями. Для прикладу, CartController відповідає за роботу з кошиком покупця, MachineryController забезпечує виконання усіх дій пов’язаних з технікою, OrderController обробляє замовлення, а UserController надає можливість управління профілями користувачів. Окремо представлений AdminController, який дає адміністраторам змогу керувати системою, додавати чи змінювати інформацію про користувачів і техніку.

Рівень доступу до даних містить компоненти, що працюють безпосередньо з базою даних. UserRepository відповідає за зберігання та обробку інформації про користувачів, а MachineryRepository - за дані про агротехніку.

Така структура дозволяє ефективно управляти бізнес-процесами системи продажу агротехніки онлайн та забезпечує зручність використання для адміністрації магазину та кінцевих користувачів. Також дозволить в майбутньому розширювати та масштабувати систему.

2.5 Висновки до розділу 2

Проектування інформаційного та програмного забезпечення заклали основу для побудови стабільної та ефективної інформаційної системи для управління продажами агротехніки. Під час створення ER-діаграми, були наочно відображені сутності, їх атрибути та взаємозв'язки між ними. Діаграми класів доповнюють ER-моделі, описуючи статичні аспекти програмного забезпечення, включаючи ієрархію

класів, їх методи та властивості, що полегшує подальшу реалізацію коду. Крім цього, діаграми кооперацій допомагають відобразити, як різні об'єкти системи взаємодіють між собою для досягнення певних цілей, таких як оформлення замовлень, обробка платежів або створення звітів.

Правильне проектування модулів та взаємозв'язків між ними є важливим для забезпечення стабільної роботи системи, а також для майбутнього її розширення. Відразу передбачене розділення компонентів на окремі модулі, такі як обробка замовлень, управління клієнтами, фінансовий облік та аналітика, дозволяє не лише зменшити взаємозалежність між модулями, але й значно спростити інтеграцію з іншими системами в майбутньому. Логічна структура бази даних показує зв'язок між таблицями та допомагає підтримувати цілісність даних, це, у свою чергу, дозволяє уникати дублювання та помилок при обробці інформації.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Система управління інформаційною базою, тобто сукупність програмних засобів і методів, які забезпечують створення, зберігання, обробку та використання даних, допомагають працювати з великими обсягами даних, автоматизувати бізнес-процеси, приймати швидше рішення та об'єднувати різні потоки інформації та даних в єдину систему.

Існує декілька основних підходів до організації даних при розробці програмних продуктів. Найпоширенішими є централізовані та розподілені системи. У централізованих інформаційних системах всі дані зберігаються на одному сервері, це в свою чергу забезпечує високу узгодженість, але також створює проблеми з майбутнім розширенням платформи. Натомість розподілені системи розміщують дані на декількох серверах, що підвищує продуктивність додатка в цілому, але погіршує узгодженість даних.

Щоб ефективно працювати з даними, потрібні певні програми, які називаються системами управління базами даних. Вони допомагають покращити обробку даних, зменшують кількість помилок і роблять роботу швидшою та простішою. СУБД використовують для зберігання, пошуку і аналізу даних у різних програмах, таких як системи планування, прогнозування та бізнес-аналітики. Вибір конкретної СУБД залежить від того, з якими даними потрібно працювати, наскільки важлива швидкодія, які вимоги до безпеки та в якому форматі дані повинні зберігатися.

Одним із найбільш поширених варіантів є реляційні СУБД, які зберігають дані у вигляді таблиць. Таблиці містять чітко визначені зв’язки між собою через первинні ключі, що забезпечує узгодженість і цілісність інформації. Системи баз

даних використовують мову SQL для роботи з даними. Це дозволяє швидко виконувати різні операції, як-от пошук, оновлення чи видалення певної інформації, а також створювати складні запити для аналізу великих обсягів даних або вибору необхідної інформації з декількох таблиць одночасно.

На відміну від реляційних СУБД, документо-орієнтовані працюють з неструктурованими або напівструктурованими даними, де збереження відбувається у форматах, таких як JSON. Такий підхід дозволяє зберігати більш складні та гнучкі структури, що дуже корисно для програмних додатків, де дані можуть змінюватися або бути різнотипними. Документо-орієнтовані бази даних забезпечують високу масштабованість та швидкість при роботі з великими обсягами даних, що робить їх ідеальними для додатків, які потребують обробки динамічних даних, таких як веб-додатки або мобільні застосунки.

Вибір системи управління базами даних напряму залежить від конкретних потреб, вимог та завдань з якими стикаються розробники. Кожен тип СУБД має свої сильні сторони та обмеження, тому важливо ретельно обдумати рішення, яке найкраще відповідає вимогам щодо способу збереження даних, продуктивності, масштабованості та безпеки. Правильно обрана СУБД забезпечить ефективне управління даними та дозволить швидко отримати доступ до необхідної інформації, що є, напевно, ключовою властивістю для успішної роботи будь-якої системи. [6]

Після створення логічної моделі даних у вигляді ER-діаграми стало зрозуміло, що структура даних має чіткі сутності, атрибути та зв’язки між ними. Дані організовані таким чином, що кожен об’єкт системи має унікальний ідентифікатор, а між сутностями існують чіткі взаємозв’язки типу "один до багатьох" та "один до одного". Така структура природно відповідає реляційній моделі, яка дозволяє забезпечити цілісність даних, усунути дублювання та забезпечити ефективне виконання запитів мовою SQL за допомогою JOIN-операцій.

У реляційних базах даних потрібно структуру даних привести до третьої нормальної форми, що дозволить уникнути надлишкових даних та мінімізувати

можливі помилки при зміні інформації. Враховуючи потребу в строго визначеній структурі, високій продуктивності для операцій та зручності роботи з запитами SQL, було вирішено зберігати дані саме в реляційній СУБД.[7]

Зараз існує кілька популярних реляційних СУБД, кожна з яких має свої переваги та особливості. В табл. 3.1 представлено їхній порівняльний аналіз.

Порівняльний аналіз реляційних СУБД

Таблиця 3.1

СУБД	**Переваги**	**Недоліки**
Oracle Database	продуктивна, надійна, дозволяє масштабуватися підтримка складних запитів, надійність і безпека даних;	висока вартість, є велика складність налаштування та адміністрування;
PostgreSQL	відкритий код і безкоштовність, підтримка як реляційних, так і об'єктно-реляційних даних, висока надійність;	складність налаштування, менша популярність у порівнянні з іншими СУБД;
MySQL	відкритий код і безкоштовність, висока продуктивність, велика спільнота та технічна підтримка;	обмежена підтримка складних запитів, менша надійність у порівнянні з PostgreSQL;
Microsoft SQL Server	продуктивна, надійна, дозволяє масштабуватися присутня інтеграція з продуктами Microsoft, є безкоштовні тарифні плани, надійність і безпека даних;	висока вартість ліцензії, працює переважно на Windows;

Для реалізації розроблюваної системи продажу агротехніки онлайн було обрано Microsoft SQL Server, оскільки він надає:

- Високу продуктивність при обробці великої кількості даних.
- Надійний механізм збереження цілісності даних (підтримка транзакцій ACID).
- Безпеку даних з розширеними можливостями аутентифікації та контролю доступу.
- Можливість масштабування та підтримку кластерних технологій.
- Інтеграцію з технологіями Microsoft, що спрощує роботу з .NET-додатками та бізнес-аналітикою. Оскільки сама система буде розроблятися на ASP.NET, використання Microsoft SQL Server забезпечить необхідну сумісність, зручність розробки та ефективну взаємодію з серверною частиною.

Підсумовуючи, можна зробити висновок, що Microsoft SQL Server був обраний, як найбільш оптимальне рішення для розробки даної інформаційної системи.

3.2 Розробка інформаційної бази

Створення бази даних для інформаційної системи продажу агротехніки онлайн передбачає спочатку створення самої бази, потім вже всіх необхідних таблиць для зберігання та обробки даних, а також налаштування індексів, тригерів і процедур для забезпечення її повноцінного функціонування. Далі буде описано кожен етап створення бази даних, її структури, необхідних компонентів, а також необхідні механізми для оптимізації та автоматизації роботи системи. Загалом база даних має бути спроектована так, щоб забезпечувати зручне зберігання та обробку інформації про користувачів, про товари, про замовлення та виконання усіх необхідних операцій з даними.

Для забезпечення цілісності даних та зав’язків між сутностями у розроблюваній базі даних буде використовуватися механізм зовнішніх ключів (FOREIGN KEY REFERENCES), що дозволить встановлювати зв’язки між таблицями та забезпечить логічну узгодженість даних. Крім того, застосовується каскадне видалення (ON DELETE CASCADE) у випадках де це необхідно, що дає змогу автоматично видаляти всі залежні записи при видаленні головного, запобігаючи можливому залишенню "осиротілих" записів у БД.[8]

Для створення самої бази даних AgroDB потрібно буде виконати наступний SQL-запит: «CREATE DATABASE AgroDB;». За допомогою такого скрипту створиться база даних, після чого буде здійснено налаштування всіх її елементів для подальшого використання в розроблюваній системі. У подальших кроках реалізується створення таблиць, налаштування тригерів, індексів і створення збережених процедур для виконання складних операцій, таких як створення нової техніки або редагування параметрів вже наявної. На рис. 3.1 наведено приклад запиту для створення таблиці Machinery, яка зберігатиме інформацію про техніку.

```
use AgroDB
go

CREATE TABLE Machinery (
    MachineryCode CHAR(8) PRIMARY KEY,
    MachineryName NVARCHAR(255) NOT NULL,
    Price DECIMAL(18,2) NOT NULL,
    Image_ NVARCHAR(MAX),
    Description_ NVARCHAR(MAX),
    Article NVARCHAR(255) UNIQUE,
    IsAvailable BIT DEFAULT 1,
    Characteristics NVARCHAR(MAX),
    CategoryCode CHAR(3) FOREIGN KEY REFERENCES Category(CategoryCode) ON DELETE SET NULL
);
```

Рис. 3.1 Запит на створення таблиці в БД

Аналогічним чином, у базі AgroDB будуть створені й інші таблиці, що забезпечать збереження інформації про клієнтів та їхні замовлення, кошики та інші сутності системи продажу агротехніки онлайн. SQL-запити для створення структури бази даних наведені у додатку Е. Окрім основної структури бази, будуть

реалізовані індекси для оптимізації швидкості запитів, процедури для автоматизації операцій з даними, а також тригери для забезпечення додаткового контролю за їх цілісністю.

Окрім основної структури бази даних, буде також створено користувача бази даних із іменем manager, який матиме необхідні права доступу до даних та дій з ними, що дозволить розмежувати рівні доступу та забезпечить більш контрольовану взаємодію з даними та операціями над ними.

Користувач manager отримає права на читання, внесення змін і видалення записів, що необхідно для коректної роботи інформаційної системи. Однак йому не надаватимуться адміністративні права на зміну самої структури бази даних, що запобігатиме випадковим або небажаним змінам.

Такий підхід дозволить збільшити рівень захищеності даних, дозволить зберегти стабільність роботи платформи навіть при великих навантаженнях та забезпечить ефективне управління доступом до даних.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка системи продажу агротехніки онлайн потребує аналізу перед вибором технологій, які забезпечать надійність, масштабованість у майбутньому і зручність для користувачів. Так як система має підтримувати обробку великої кількості інформації, мати зручний інтерфейс та взаємодіяти з БД, необхідно обрати відповідний стек технологій для реалізації фронтенду, бекенду та управління даними.

Для створення серверної частини застосунку є декілька популярних технологій, кожна з яких має свої особливості. Одним із найпоширеніших рішень є Python з фреймворком Django. Його перевагою є висока швидкість створення MVP-продуктів і хороша підтримка з боку спільноти. Однак продуктивність Django

може поступатися іншим рішенням, особливо під час роботи із великим навантаженням.[9]

Також можна розглянути Java з фреймворком Spring Boot, який підходить для створення корпоративних веб-додатків. Spring Boot забезпечує високу продуктивність та безпеку, а також має зручну інтеграцію з іншими сервісами. Проте його недоліком є складність налаштування та висока вартість розробки, оскільки для ефективної роботи потрібні досвідчені розробники.

Вибираючи найкращу технологію для реалізації системи, важливо враховувати продуктивність, безпеку, зручність розробки та майбутню підтримку. З огляду на всі ці фактори було вирішено використовувати ASP.NET.

ASP.NET - це сучасний фреймворк від компанії Microsoft, який забезпечує високу продуктивність та гнучкість під час створення веб-додатків. Він надає можливість використовувати C# як основну мову для написання серверної частини, а також може інтегруватися з популярними фреймворками для фронтенду, що забезпечує стабільність і швидкодію застосунку. Оскільки ASP.NET компілюється у проміжний код, який виконується у середовищі .NET, продуктивність таких застосунків значно вища, ніж у платформ написаних на інтерпретованих мовах програмування.

Однією з головних переваг ASP.NET є його підтримка архітектури MVC (Model-View-Controller), яка допомагає чітко розділити логіку програми на три рівні:

- Модель - відповідає за взаємодію з базою даних і бізнес-логіку програми. У цьому шарі зберігаються основні структури даних, методи їх обробки та взаємодія з БД через Entity Framework або інші ORM.
- Представлення - визначає, як користувачі взаємодіють із веб-застосунком. В ASP.NET можна використовувати Razor Pages або інтегрувати фронтенд на базі React чи Vue.js для створення динамічного інтерфейсу.

- Контролер - містить логіку обробки запитів користувача, викликає необхідні моделі та формує відповідь у вигляді представлення.

Завдяки гнучкості ASP.NET, для реалізації онлайн-системи продажу агротехніки буде використано підхід MVC у поєднанні з Razor Pages. Це дозволить забезпечити чітку архітектуру застосунку, ефективну взаємодію між його компонентами та зручний користувацький інтерфейс.

ASP.NET MVC є одним із найбільш потужних підходів до побудови веб-застосунків, оскільки дозволяє логічно розділити застосунок на три основні структурні частини. Модель (Model) відповідатиме за обробку даних і їхню взаємодію з базою, Контролер (Controller) буде керувати бізнес-логікою, а Представлення (View) відповідатиме за відображення даних на веб-сторінках. Це чітке розмежування дозволяє спростити тестування, підтримку та розширення застосунку в майбутньому, що, у свою чергу, є важливим для стабільної роботи системи у довгостроковій перспективі.

Для реалізації користувацького інтерфейсу буде використано Razor Pages - сучасну технологію, яка поєднує простоту у створенні веб-сторінок з усіма перевагами MVC-архітектури. Razor Pages є частиною ASP.NET Core і забезпечує швидку розробку веб-додатків із чітко структурованими сторінками, що мають власні моделі обробки запитів.[11]

Основною перевагою Razor Pages є те, що кожна сторінка працює автономно, вона має власну модель, що дозволяє зменшити складність та забезпечити відповідну продуктивність. Це особливо важливо для веб-застосунків, що обробляють велику кількість запитів, оскільки сторінки завантажуються швидко та ефективно взаємодіють із сервером. Razor також підтримує динамічний рендеринг HTML із використанням C#, що дозволяє створювати інтерактивні сторінки без необхідності використовувати сторонні JavaScript-фреймворки, хоча їх використовувати також можна.

Вибір ASP.NET MVC та Razor Pages для створення системи продажу агротехніки онлайн дозволить створити продуктивний, зручний додаток з чіткою організацією коду та надійним рівнем безпеки. Використання шаблону MVC дозволить створити динамічні сторінки, що робить систему швидкою та адаптивною до змін.

3.4 Алгоритмізація та програмування програмних модулів

3.4.1 Загальні відомості. Алгоритмізація та програмування програмних модулів є одним з останніх основних етапів розробки системи продажу агротехніки онлайн. Алгоритмізація передбачає розробку блок-схем алгоритмів, які описують логіку роботи окремих функцій системи, включаючи обробку запитів користувачів, управління товарами, виконання замовлень тощо. Це дозволяє наочно представити послідовність операцій, визначити точки прийняття рішень та оптимізувати процеси для підвищення ефективності роботи системи.

Програмування програмних модулів включає реалізацію цих алгоритмів у вигляді програмного коду, що охоплює як бекенд, так і фронтенд частини системи. Бекенд буде написаний на мові програмування C# і відповідатиме за обробку бізнес-логіки, управління даними та взаємодію з базою даних. Фронтенд забезпечить інтерфейс користувача, реалізований за допомогою Razor Pages, що дозволить забезпечити зручну та ефективну взаємодію з системою.[12]

Кожен алгоритм буде детально описаний далі, також буде представлена його блок-схема та код його реалізації. Будуть описанні такі алгоритми: авторизації, додавання нової техніки, фільтрація каталогу, додавання техніки до кошика, оформлення замовлення та генерація звітів.

Детальна програмна реалізація кожного з алгоритмів, що включає код серверної та клієнтської частини, буде представлена в додатку Ж, де можна ознайомитися з усіма важливими аспектами реалізації таких процесів.

3.4.2 Алгоритм авторизації користувача. Авторизація користувачів в системі продажу агротехніки онлайн є дуже важливою для забезпечення відповідного рівня контролю доступу до персоналізованих функцій. Процес авторизації розпочинається з того, що користувач вводить свій логін та пароль у відповідні поля на сторінці входу. Після натискання кнопки «Вхід» система перевіряє, чи не залишилися ці поля порожніми, оскільки відсутність даних у них зробить неможливою подальшу перевірку.

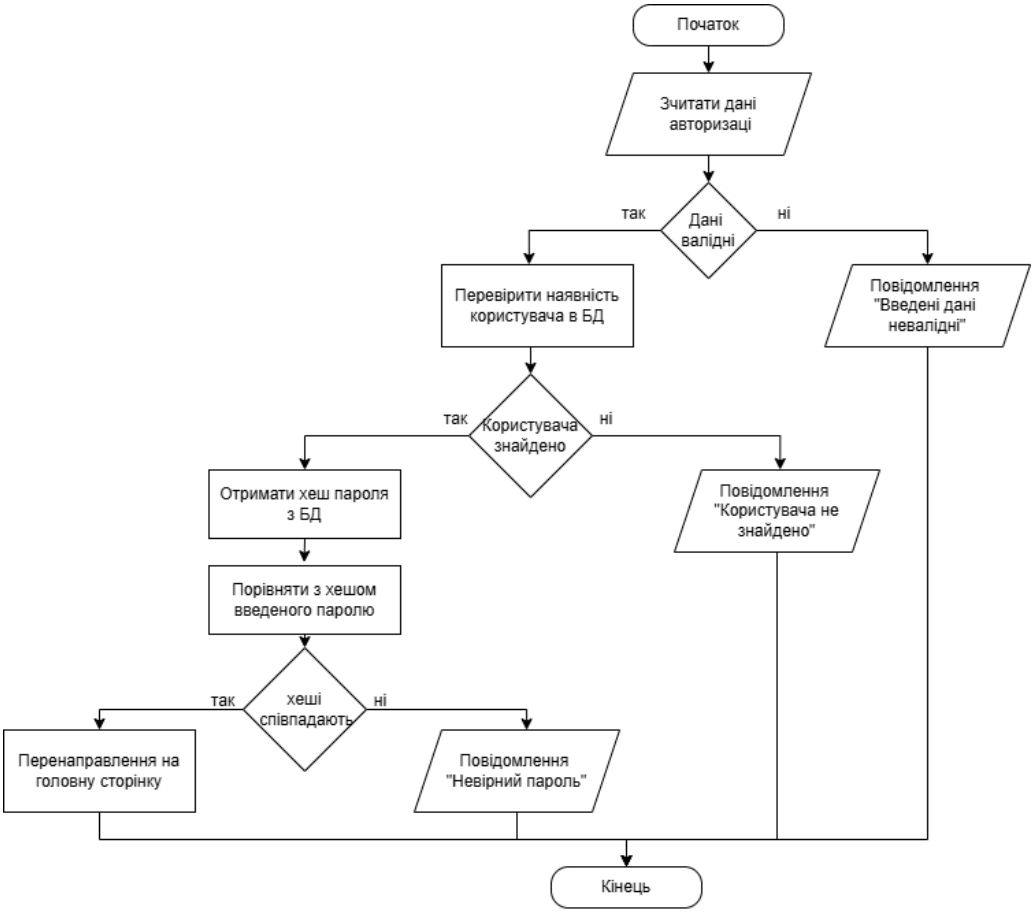


Рис. 3.2 Блок-схема алгоритму авторизації

Якщо логін і пароль введені, система виконує запит до бази даних, щоб знайти відповідного користувача. У разі відсутності облікового запису з таким логіном система повідомляє користувача про помилку та пропонує перевірити правильність введених даних. Якщо ж обліковий запис знайдено, отримується збережений у базі хешований пароль для подальшої перевірки.

Наступним кроком є порівняння введеного користувачем пароля із збереженим у БД хешованим значенням. Якщо паролі збігаються, авторизація вважається успішною, і система генерує сесію для збереження стану входу. Після цього користувач одразу перенаправляється на головну сторінку. У разі невідповідності паролів система виводить повідомлення про помилку та пропонує спробувати ще раз.[13]

3.4.3 Алгоритм створення нової техніки. Алгоритм створення нової техніки в системі онлайн-продажу агротехніки дозволяє адміністратору додавати нові одиниці техніки до каталогу. Цей процес передбачає спочатку введення основної інформації про техніку, такої як назва, опис, вартість, категорія та інші параметри, а також завантаження її зображення.

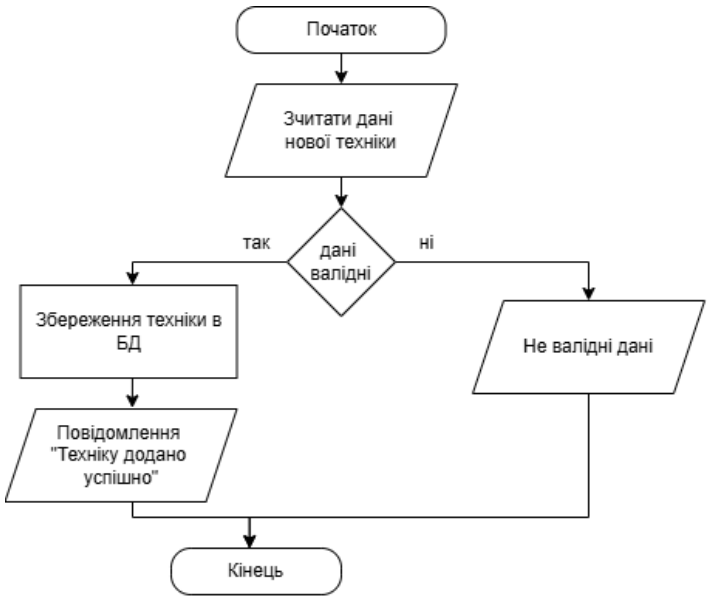


Рис. 3.3 Блок-схема алгоритму створення нової техніки

Алгоритм починається з того, що адміністратор переходить на сторінку для додавання нової техніки. На цій сторінці є форма для введення всіх необхідних даних: назви техніки, її опису, вартості, категорії, характеристик та зображення. Після заповнення форми користувач натискає кнопку "Додати", що ініціює перевірку введених даних.

Далі відбувається перевірка на те, чи всі обов'язкові поля заповнені, і чи введені дані мають коректний формат. Якщо дані правильні, система зберігає інформацію про техніку в базі даних. Якщо є помилки або незаповнені поля, система виводить повідомлення про помилку та вказує, які саме поля необхідно виправити. Після успішного додавання техніки користувач отримує повідомлення про успішне виконання операції, а щойно доданий товар з'являється в загальному списку техніки.

3.4.3 Алгоритм фільтрації в каталозі. Алгоритм фільтрації в каталозі агротехніки дозволяє користувачам знаходити необхідну техніку, застосовуючи фільтри по категорії, вартості, наявності та інших характеристиках. Крім того, система підтримує пошук за ключовими словами для зручного пошуку техніки за її назвою або описом.

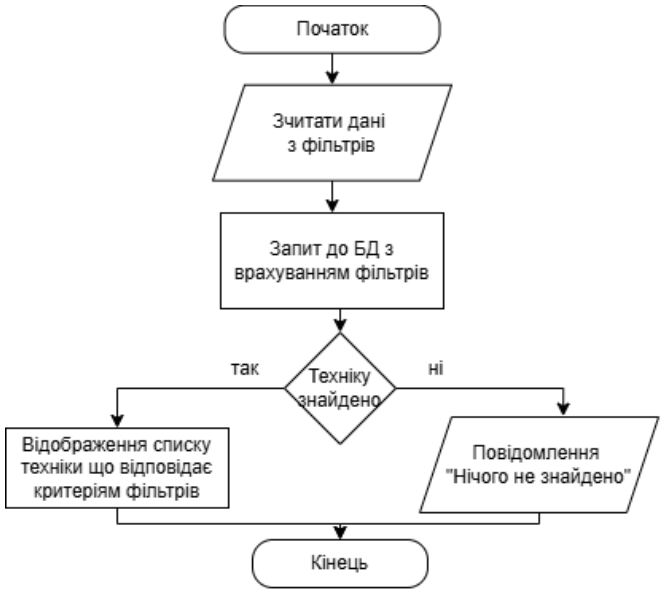


Рис. 3.4 Блок-схема алгоритму фільтрації в каталозі

Алгоритм починається з того, що користувач переходить на сторінку каталогу техніки, далі активує фільтри і стають доступні поля для введення пошукового запиту і фільтри для вибору категорії, діапазону вартості тощо. Користувач може або скористатися фільтрами, або ввести конкретне слово у поле пошуку.

Після натискання кнопки пошуку або застосування фільтра система отримує введені дані та здійснює запит до бази даних для пошуку товарів, які відповідають заданим критеріям. Якщо введено пошукове слово, система шукає техніку за назвою або описом, а якщо вибрано фільтри, система застосовує їх до результатів пошуку. Після виконання запиту відображається список техніки, що відповідає критеріям фільтрації або пошуку.

Якщо за результатами пошуку не знайдено жодної техніки, користувач отримає повідомлення про відсутність результатів. Якщо пошук був успішним, на екрані з’являться картки тільки цієї техніки, яка відповідає запиту.

3.4.5 Алгоритм додавання техніки до кошика. Алгоритм додавання техніки до кошика передбачає перевірку наявності товару, щоб користувач міг додати до кошика лише те обладнання, які є в наявності.

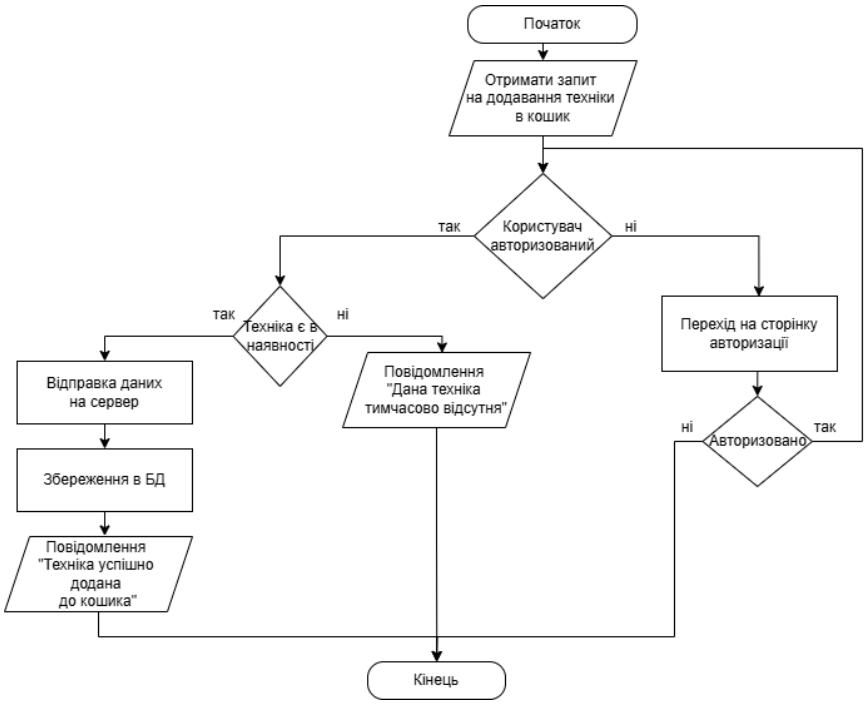


Рис. 3.5 Блок-схема алгоритму додавання техніки до кошика

Коли користувач переглядає деталі техніки на сторінці, система перевіряє, чи є техніка в наявності. Якщо товар є в наявності, користувач може натискати кнопку "Додати до кошика". При натисканні на цю кнопку система ще раз перевіряє наявність товару на складі. Якщо техніка є в наявності, вона додається до кошика, і користувач отримує підтвердження про успішне додавання товару.

У разі відсутності товару на складі виводиться повідомлення про тимчасову відсутність. Реалізація цього процесу на C# ASP.NET включає перевірку через базу даних та забезпечує зручний зворотний зв'язок для користувача.

3.4.6 Алгоритм оформлення замовлення. Процес оформлення замовлення є завершальним етапом покупки в системі онлайн-продажу агротехніки. Користувач переходить до кошика, де може переглянути список доданих ним раніше товарів, змінити кількість одиниць техніки або видалити непотрібні позиції при потребі.

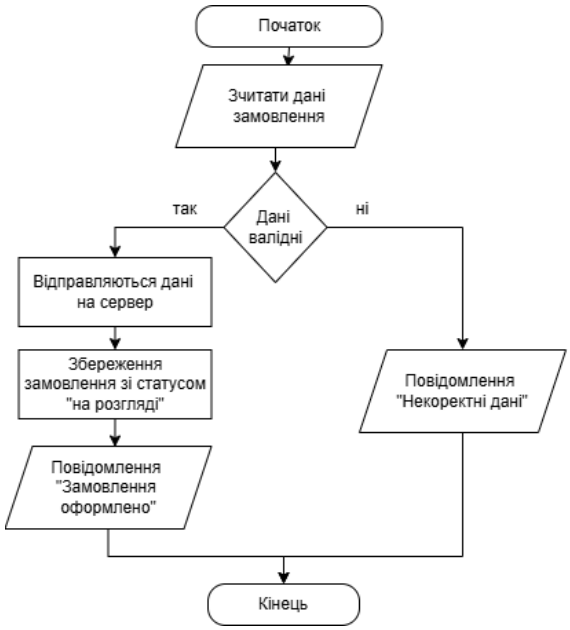


Рис. 3.6 Блок-схема алгоритму оформлення замовлення

Після перевірки вмісту кошика користувач заповнює форму з даними для оформлення замовлення. У формі необхідно вказати ім’я, контактний номер телефону та вибрати метод оплати і доставки. Якщо користувач змінює кількість техніки, система автоматично перераховує загальну вартість замовлення. Коли всі дані введені, користувач натискає кнопку "Оформити замовлення".

Цей алгоритм забезпечує простий і зрозумілий процес оформлення замовлення, дозволяючи користувачеві редагувати кошик, заповнювати дані та отримувати актуальну інформацію про наявність товарів перед завершенням покупки.

3.4.7 Алгоритм генерації звітів. Функціонал генерації звітів у системі онлайн-продажу агротехніки дозволяє адміністраторам отримувати аналітичну інформацію про замовлення, доступність техніки та інші важливі показники. Процес починається з того, що користувач відкриває розділ звітності, вибирає тип звіту, який потрібно згенерувати та вказує параметри. Наприклад, це може бути період часу, категорія техніки, статус замовлень тощо.

Рис. 3.7 Блок-схема алгоритму генерації звітів

Після вибору параметрів система отримує введені значення та надсилає запит до бази даних. У запиті враховуються всі обрані фільтри, щоб витягнути лише потрібну інформацію. База даних обробляє запит і повертає відповідні дані системі, яка їх структуровано обробляє та формує звіт у форматі PDF.

Коли звіт сформовано, система створює відповідний файл і автоматично його завантажує на пристрій користувача. Таким чином, процес є повністю автоматизованим і не потребує додаткових дій з боку користувача після вибору параметрів.

Цей алгоритм забезпечує швидке отримання необхідної звітності та покращує аналіз даних, дозволяючи користувачам отримувати актуальну інформацію про стан продажів і замовлень у зручному вигляді та дозволяє мееджерам ефективно працювати з аналітичними даними, формувати потрібні звіти та швидко отримувати інформацію у зручному форматі.

3.5 Висновки до розділу 3

Розробка інформаційного та програмного забезпечення складалася з таких етапів: створення бази даних, розробку серверної та клієнтської частин системи, а також реалізацію основних функцій, таких як реєстрація користувачів, управління каталогом техніки, оформлення замовлень та генерації звітів.

Для збереження даних було використано реляційну базу даних, що забезпечує надійне зберігання та швидкий доступ до інформації. Важливою частиною цього етапу є також впровадження захисту даних користувачів, що включає хешування паролів та захист від несанкціонованого доступу. Успішне завершення цього етапу дозволило розробити систему, яка буде стабільно працювати, а також надасть можливість адміністраторам швидко та ефективно обслуговувати клієнтів у процесі купівлі агротехніки. Це є важливим кроком для створення конкурентоспроможного програмного продукту, який відповідає сучасним вимогам ринку.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Впровадження програмного додатку є крайнім етапом її життєвого циклу перед повноцінним використанням. Цей етап передбачає перевірку працездатності всіх компонентів розробленої системи, виявлення і усунення можливих помилок, а також оцінку, чи відповідає функціональність заявленим вимогам. Основною складовою даного етапу є тестування, тобто процес дослідження програмного продукту з метою виявлення дефектів та перевірки його поведінки в різних умовах.

Тестування дозволяє перевірити, що система працює стабільно, коректно обробляє введені дані, відповідає очікуванням користувача та не містить ніяких помилок. Налічується декілька типів тестування, серед яких: модульне тестування (перевірка окремих компонентів), інтеграційне тестування (перевірка взаємодії компонентів), системне тестування (перевірка всієї системи загалом) та приймальне тестування (оцінка готовності до експлуатації).

Під час реалізації розробленої системи продажу агротехніки онлайн, розробленої на платформі ASP.NET, було застосовано в більшій мірі системне тестування, яке дозволяє перевірити функціональність усіх основних сценаріїв взаємодії користувача з сайтом. Зокрема, перевірялися такі процеси: реєстрація користувача, авторизація, перегляд товарів за категоріями, додавання товарів до кошика, оформлення замовлення та збереження інформації про це все у базі даних.

Для проведення тестування було обрано ручний підхід, як один з найпростіших і доступних методів перевірки роботи сайту на етапі впровадження. Кожен функціональний модуль перевірявся вручну шляхом виконання послідовних дій коректних і неправильних у веб-інтерфейсі системи та фіксації результатів. [14]

Наприклад, при тестуванні функціоналу реєстрації користувача перевірялися наступні випадки:

- введення коректних даних (очікуваний результат - успішна реєстрація);
- спроба зареєструватися з уже існуючим логіном (очікуваний результат - повідомлення про помилку);
- не заповнення обов'язкових полів (очікуваний результат - виведення повідомлень про необхідність заповнення).

Окрім реєстрації та авторизації, тестувалися й інші функції системи, які мають вирішальне значення для забезпечення зручної взаємодії користувача з платформою. Зокрема, перевірявся модуль перегляду товарів за категоріями. Було протестовано відображення списку агротехніки при виборі певної категорії, коректність завантаження інформації з бази даних, правильне відображення назв, вартості, описів і зображень. У випадках, коли категорія не містила товарів, система виводила повідомлення про відсутність даних, що підтверджує її стійкість до нестандартних ситуацій.

Також перевірялися функції додавання товарів до кошика. Було проведено тестування таких сценаріїв, як додавання одного або кількох товарів, повторним додаванням одного й того ж товару, а також видаленням позицій із кошика. Система успішно обробляла всі дії: кількість обраних товарів змінювалася відповідно до дій користувача, загальна сума замовлення перераховувалася коректно, а зміни відображалися в режимі реального часу.

Наступним кроком було протестувати процес оформлення замовлення. Тут перевірялася робота форми з контактними даними, правильність валідації полів, збереження замовлення у базі даних та відображення оформленого замовлення у списку замовлень користувача після завершення операції. Всі тестові сценарії завершилися успішно, за винятком кількох виявлених дрібних неточностей у відображенні повідомлень про помилки, які були оперативно виправлені.

Аналогічно перевірялися інші модулі. Результати тестування фіксувалися у вигляді таблиці з зазначенням перевіреного сценарію, вхідних даних, очікуваного результату та фактичного результату. Це дозволило вчасно виявити та усунути окремі помилки у логіці обробки даних та забезпечити стабільну роботу основних функцій системи.

Проведене тестування показало загальну працездатність основних функцій системи, її здатність правильно реагувати на дії користувача та обробляти усі сценарії використання без збоїв. Це дало змогу перейти до підготовки до реального розгортання.

4.2 Вимоги до апаратного та програмного забезпечення

Для того, щоб система ефективно працювала потрібно визначити вимоги до апаратного і програмного забезпечення. Одним із етапів, щоб це зробити є побудова діаграми розміщення, яка відображає основні вузли системи та взаємодію між ними. На рис. 4.1 представлено архітектуру системи, що складається з трьох основних вузлів: клієнтського пристрою (позначено як User PC), веб-сервера (Web Server) та сервера бази даних (MS SQL Server).

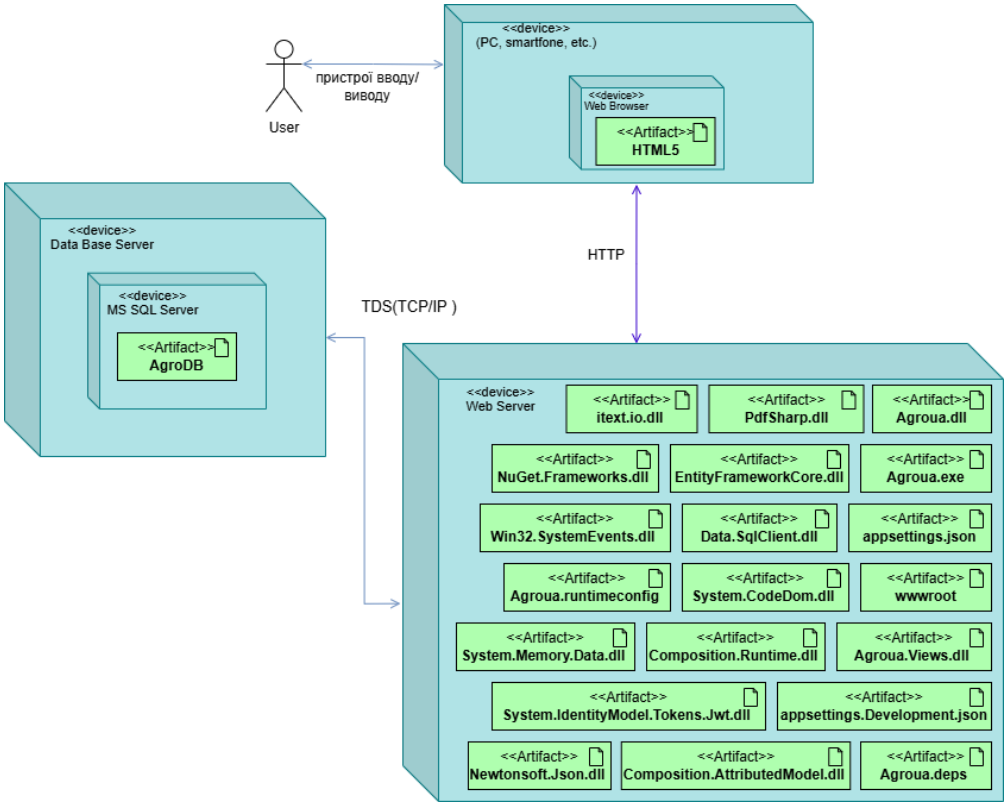


Рис. 4.1 Діаграма розміщення

Система реалізована за архітектурою клієнт-сервер, де користувачі взаємодіють із веб-сайтом через браузер, веб-сервер обробляє запити та взаємодіє з базою даних, тоді надсилає відповіді. Такий підхід до розробки забезпечить масштабованість системи та гнучкість, дозволяючи витримувати великі навантаження.

Для більш стабільної та ефективнішої роботи платформи потрібно забезпечити мінімальними вимогами всі її частини: клієнтські пристроїв, веб-сервер та сервер бази даних.

Клієнтські пристрої. Для клієнтських пристроїв немає критичних вимог, адже для доступу до функціоналу веб-сайту достатньо, щоб був встановлений сучасний веб-браузер. Це дозволить користувачам працювати як із персональних комп’ютерів, так і з мобільних пристроїв, а це підвищує доступність системи в цілому. Щодо апаратного забезпечення, вимоги теж мінімальні. Будь-який сучасний

ноутбук чи комп’ютер, планшет або смартфон зможе забезпечити потрібний рівень продуктивності для роботи з платформою.

Для стабільної та коректної роботи системи потрібно мати актуальне програмне забезпечення. Веб-браузер може бути встановлений будь-який, що підтримує такі технології, такі як HTML5, CSS3 та JavaScript. Рекомендовано використовувати такі популярні та нові браузери такі, як Google Chrome, Mozilla Firefox, Microsoft Edge або Safari. По операційній системі теж немає обмежень, тобто може бути будь-якою: Windows, macOS, Linux, Android або iOS.

Веб-сервер. Веб-сервер це головний компонент системи, який приймає HTTP-запити від клієнтських пристроїв, обробляє їх і повертає необхідні відповіді. Саме його продуктивність найбільше впливає на швидкість обробки запитів та загальну роботу системи.

У таблиці 4.1 представлено апаратні вимоги, а в таблиці 4.2 вказані програмні вимоги для веб-сервера системи продажу агротехніки онлайн.

Апаратні вимоги веб-сервера

Таблиця 4.1

Ресурс	**Мінімальний**	**Рекомендований**
Процесор	4 ядра, 2.5 ГГц	8 ядер, 3.0 ГГц+
ОЗП	8 ГБ	16 ГБ і більше
Накопичувач	SSD 256 ГБ	SSD 512 ГБ+
Мережа	1 Гбіт/с	10 Гбіт/с

Програмні вимоги веб-сервера

Таблиця 4.2

Ресурс	**Мінімальний**	**Рекомендований**

ОС	Windows Server 2019 / Ubuntu 20.04	Windows Server 2022 / Ubuntu 22.04
Веб-сервер	IIS 10 / Nginx 1.18	IIS 10 / Nginx 1.22+
Платформа	.NET Core 3.1	.NET 6.0 або вище
База даних	Microsoft SQL Server Express	Microsoft SQL Server Standard
Безпека	SSL/TLS підтримка	Автоматичне оновлення сертифікатів

Вибір відповідних параметрів веб-сервера найбільше впливає на продуктивність системи в цілому, швидкість обробки запитів та загальну стабільність роботи. Рекомендується використовувати сучасне апаратне обладнання і оновлене програмне забезпечення, це, в свою чергу, дозволяє забезпечити надійну роботу платформи, без ризиків збоїв чи затримок у виконанні запитів.

Сервер бази даних (MS SQL Server). Сервер бази даних є не менш важливою частиною системи, оскільки саме там зберігається вся інформація про користувачів, техніку, замовлення, а також й та інші важливі дані. Його продуктивність напряму впливає на швидкість обробки запитів та ефективність роботи платформи.

Для ефективної роботи необхідно забезпечити наступні апаратні ресурси. Мінімальні вимоги будуть такі: процесор із 4 ядрами та частотою 2.5 ГГц, 16 ГБ оперативної пам’яті, SSD-накопичувач мінімум на 512 ГБ та мережеве підключення зі швидкістю 1 Гбіт/с. Для трохи кращої продуктивності рекомендується сервер із 8-ядерним процесором (3.2 ГГц і більше), 32 ГБ оперативної пам’яті, SSD-накопичувачем на 1 ТБ і більше та швидкісним з’єднанням 10 Гбіт/с.

Програмне забезпечення також повинно відповідати вимогам надійності та продуктивності. Мінімальними вимогами буде Windows Server 2019 або Ubuntu 20.04, з використанням Microsoft SQL Server Express. Але для більш ефективної роботи рекомендується Windows Server 2022 або Ubuntu 22.04 з Microsoft SQL Server Standard.

Щодо безпеки та збереження даних, базові вимоги зможуть забезпечити лише локальне шифрування бази. А ось для більшого рівня захисту рекомендується використання шифрування з TLS 1.3, а також впровадження резервного копіювання, що забезпечить збереження інформації навіть у разі відмови чи поломки основного сервера.

Згідно з такими вимогами, навіть мінімальними, система онлайн-продажу агротехніки буде продуктивною та безпечною. Вона дозволить користувачам працювати з будь-якого, на якому є доступ до інтернету. Веб-сервер обробляє HTTP-запити та забезпечує швидку взаємодію між користувачами системи та сервером бази даних.

Описане апаратне та програмне забезпечення дасть можливість розширювати систему в майбутньому за потреби. Також дозволить ефективно обробляти великий обсяг замовлень і підтримувати одночасно велику кількість користувачів та надавати високу швидкість їх обслуговування.

4.3 Склад інсталяційного пакету

Оскільки розроблювана система онлайн-продажу агротехніки є веб-сайтом, її встановлення не потребує стандартного інсталяційного пакета. Але необхідно налаштувати сервер, на якому буде розміщено веб-додаток, та забезпечити його доступність усім користувачам через мережу Інтернет.

Розгортання веб-сайту розпочинається з підготовки середовища сервера. Якщо використовується сервер на базі Windows, для запуску можна встановити IIS, якщо ж Linux, то краще використовувати NGINX або Apache. Додатково потрібно встановити середовище виконання ASP.NET Core, щоб додаток міг коректно працювати.

Далі потрібно розгорнути базу даних. Для створення усіх необхідних таблиць, процедур та інших об’єктів бази даних потрібно виконати скрипт, що наведений у додатку Е.

У випадку хостингу на платформі Microsoft Azure краще використовувати Azure SQL Database, тобто онлайн версію Microsoft SQL Server. На порталі Azure створюється нова база даних, де вказується назва, обсяг, рівень продуктивності та облікові дані для доступу. Після створення необхідно дозволити зовнішні підключення. Таблиці створюються автоматично за допомогою механізму міграцій. За потреби замість міграцій можна виконати SQL-скрипт, наведений у додатку Е. Параметри підключення до бази зберігаються у файлі конфігурації appsettings.json, саме там їх потрібно замінити під створену нову базу даних.

Після цього веб-додаток компілюється та може бути опублікованим. У середовищі Visual Studio або за допомогою командного рядка (команда: "dotnet publish -c Release -o …/agroshop"), що створить готову до запуску збірку додатка. Загалом, отриманий після публікації каталог і є інсталяційним пакетом застосунку. Він містить необхідний набір файлів, потрібних для роботи системи. Такі файли завантажуються на сервер у відповідну директорію (наприклад, C:\inetpub\agroshop для Windows Server або /var/www/agroshop для Linux). У випадку з Windows Server, якщо планується використовувати IIS, то потрібно створити сайт у диспетчері IIS, вказати шлях до вищезгаданої директорії та налаштувати пул застосунків для .NET Core. Якщо ж не використовується IIS, то додаток запускається за допомогою команди dotnet agroshop.dll.

Інсталяційний пакет веб-додатку Agro.ua, містить усі необхідні файли для того, щоб бути розгорнутим на сервері. Основними елементами цього пакету є файл Agroua.dll, який містить логіку програми, файл Agroua.Views.dll із вже скомпільованими Razor-сторінками, конфігураційні файли такі, як appsettings.json та web.config, папка wwwroot зі статичними ресурсами (CSS, JavaScript, зображення), а також усі використанні у проекті бібліотеки у вигляді .dll, вони

представленні на діаграмі розміщення (рис. 4.1). У разі публікації з включеним .NET Runtime, також додається файл Agroua.exe для прямого запуску на сервері. Цей пакет не потребує класичної інсталяції, тому буде цілком достатньо скопіювати його в обрану директорію на сервері та запустити веб-додаток через IIS або NGINX.

Якщо веб-додаток буде розміщуватися на хмарі, наприклад, Microsoft Azure, то та чи інша платформи автоматично надають доменне ім’я (наприклад, agroshop.azurewebsites.net) після розгортання веб-додатку через Azure App Service. Це домен працює одразу після публікації, і сайт можна переглядати в браузері без додаткового налаштування. Але для використання зручного та власного домену (наприклад, www.agroshop.com) необхідно зареєструвати таке доменне ім’я через будь-який доменний реєстратор або безпосередньо в Azure через Azure DNS. Після цього щойно зареєстрований домен підключається до веб-додатку шляхом створення відповідних DNS-записів у налаштуваннях. У самому Azure також додається цей власний домен у налаштування App Service.

Після виконання всіх налаштувань веб-сайт стає доступним користувачам у браузері на будь-якому пристрої, який має підключення до Інтернету.

4.4 Висновки до розділу 4

Тестування системи та розробка рекомендацій щодо її впровадження та встановлення стали завершальним етапом створення програмного продукту для продажу агротехніки. На цьому етапі була проведена перевірка працездатності основних модулів та функцій, оцінка продуктивності та зручності використання інтерфейсу. Також було розроблено інструкції для користувачів та адміністраторів, що включають вимоги до апаратного та програмного забезпечення, склад інсталяційного пакета та рекомендації щодо налаштування серверної частини.

Окрім цього, було проведено оцінку продуктивності системи та майбутньої масштабованості для забезпечення стабільної роботи під час високих навантажень.

Такі заходи забезпечують стабільну роботу системи та зручність її використання для користувачів, що стане важливим для успішної експлуатації розробленого програмного забезпечення. Це дозволить не лише забезпечити високу продуктивність системи, але й гарантує безпеку даних користувачів та захист від зовнішніх загроз.

ВИСНОВКИ

Під час виконання дипломної роботи було успішно створено програмне забезпечення для онлайн-продажу агротехніки. Воно спрощує доступ для власників фермерських господарств, або приватних осіб до необхідного асортименту техніки. Метою роботи була реалізація ефективної системи для пошуку, порівняння та придбання сільськогосподарської техніки. Поставлену мету вдалося досягти, непростим шляхом, але розроблена система відповідає усім поставленим вимогам та вимогам сучасного аграрного сектору, вона надає покупцям зручний інструмент для взаємодії з постачальниками агротехніки.

У рамках виконання роботи був виконаний аналіз існуючих платформ у сфері онлайн-продажу агротехніки, це у свою чергу, дозволило виявити їх переваги та недоліки. Після аналізу був спроектований функціональний веб-додаток, який включає основні необхідні функції, такі як реєстрація користувачів, пошук, фільтрація і перегляд техніки, оформлення замовлень, а також реалізовано захист персональних даних.

Технологічно система побудована на платформі ASP.NET MVC, а база даних у SQL Server, це забезпечує високу продуктивність і стабільність роботи при великих потоках даних та навантаженнях. Веб-сайт має інтерфейс інтуїтивно зрозумілий, це робить процес пошуку, порівняння і замовлення техніки дуже зручним для користувачів. Крім того, для менеджерів системи розроблена низка окремих функцій які дозволяють зручно управляти технікою та замовленнями клієнтів, що допомагає легко керувати системою та покращувати обслуговування.

Після розробки було проведено оцінку ефективності системи, тобто відбувся процес тестування, в результаті якого було підтверджено працездатність системи і відповідність визначеним вимогам. Додаток показав високу продуктивність і надійність навіть при значних навантаженнях, це корисно для масштабування в

майбутньому. Система вже готова до використання і надає все необхідне для покращення процесу покупки техніки та ефективної взаємодії між покупцями та представниками магазинів.

Однак, разом з тим, що система вже є повністю готовою до експлуатації, існує перспектива її подальшого вдосконалення. Наприклад, можна додати інтеграцію з іншими онлайн-системами для зручності користувачів, це можуть бути платіжні системи, служби доставки тощо. Можна реалізувати можливість множинного завантаження техніки в каталог для адміністратора це було б дуже корисно Також є можливість покращити адаптивність інтерфейсу, щоб коректно підлаштовувався під різні пристрої і додати ще деякі методи оплати.

Розроблена у дипломній роботі система є повністю готовою до впровадження та використання, але з огляду на те, що її можна ще вдосконалювати та масштабувати, вона може бути ще більш покращена для забезпечення гарного обслуговування клієнтів і підтримки розвитку аграрного сектору нашої країни в майбутньому.

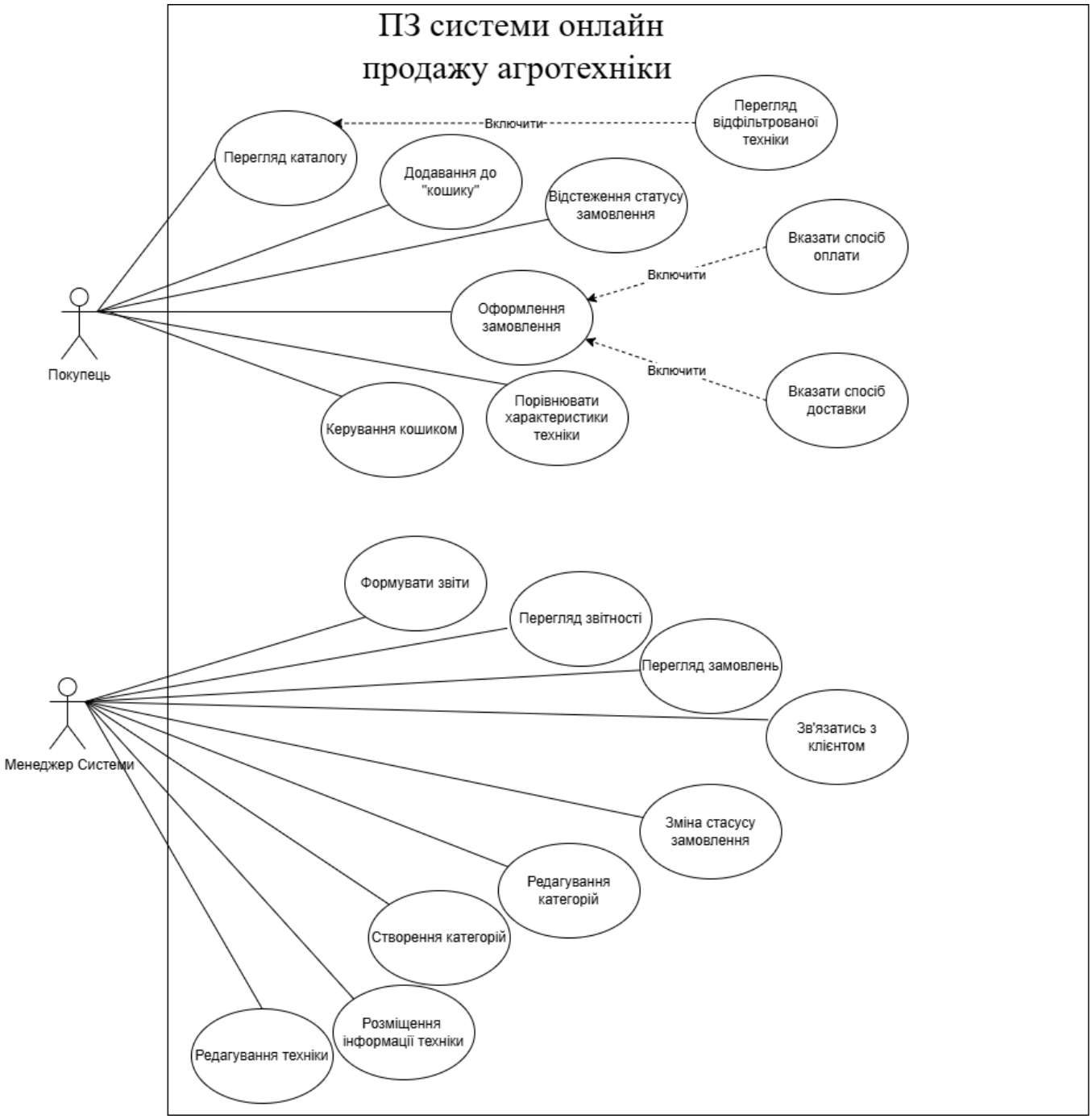
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зайнятість в аграрному секторі України: аналіз поточного стану та напрямів розвитку. URL: https://doi.org/10.32702/2307-2105.2023.2.14.
2. Аграрний сектор України в умовах війни: проблеми та перспективи. URL: https://doi.org/10.32782/2524-0072/2022-40-38.
3. Modeling with UML: Language, Concepts, Methods / Bernhard Rumpe. Berlin : Springer, 2016. 321 p.
4. Learning UML / Alhir S. S. Sebastopol : O'Reilly & Associates, 2003. 114 p.
5. Erwin Data Modeler. Release Notes, Version 2020 R1. URL: https://bookshelf.erwin.com/bookshelf/public_html/2020R1/Content/PDFs/Data%20Modeler%20Release%20Notes.pdf (дата звернення: 11.04.2025)
6. Табунщик Г. В., Каплієнко Т. І., Петрова О. А. Проектування та моделювання програмного забезпечення сучасних інформаційних систем : навч. посіб. Запоріжжя : Дике Поле, 2016. 250 с.
7. Берко А.Ю., Верес О.М. , Пасічник В.В. Системи баз даних та знань, книга 2: системи управління базами даних та знань: навч. посіб. Львів: 2021. 584с.
8. Кучанський О. Ю., Андрашко Ю. В. Інформаційні системи та реляційні бази даних : навч. посіб. Ужгород, 2023. 132 с.
9. Програмна інженерія : підручник / К. М. Лавріщева. Київ : ІСДО, 2008. 319 с.
10. Component One ASP.NET MVC / GrapeCity. URL: https://help.grapecity.com/componentone/PDF/MVC/c1mvchelpers.pdf (дата звернення: 11.04.2025)
11. ASP.NET MVC 5 з використанням C# та .NET Framework 4.5.1 / Маріо Харрінгтон. Київ: Наш формат, 2016. 688 с.
12. Чиста архітектура. Мистецтво розробки програмного забезпечення / Роберт Сесіл Мартін. Київ : Наш формат, 2020. 336 с.

13. Іванчук О. Є., Боднарчук Ю. С. ASP.NET: технології створення веб-додатків: навч. посіб. Київ : Київський університет, 2009. 424 с.
14. Крепич С. Я., Співак І. Я. Якість програмного забезпечення та тестування : навч. посіб. Тернопіль : ФОП Паляниця В. А., 2020. 478 с.

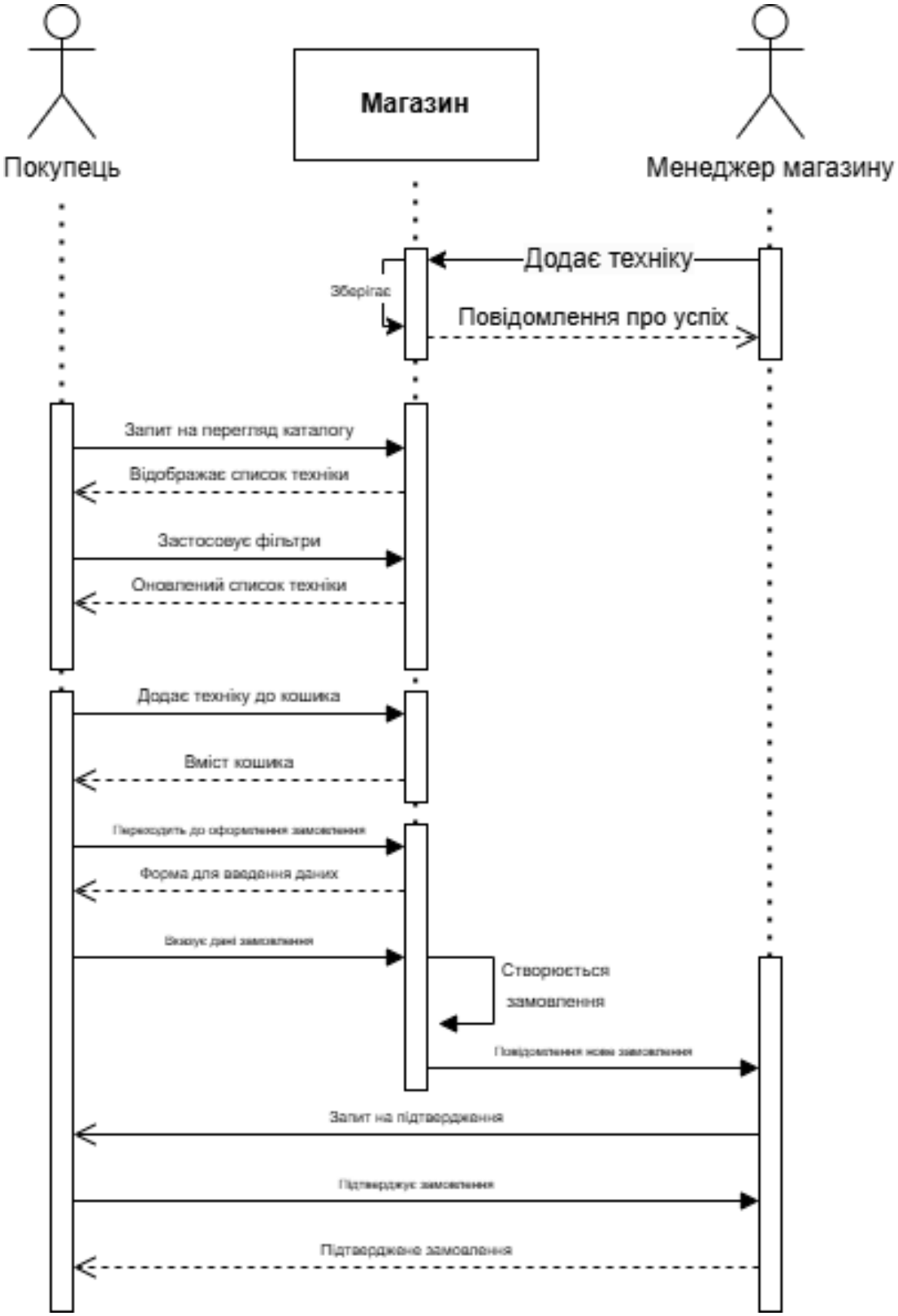
Додаток А.

Діаграма прецедентів



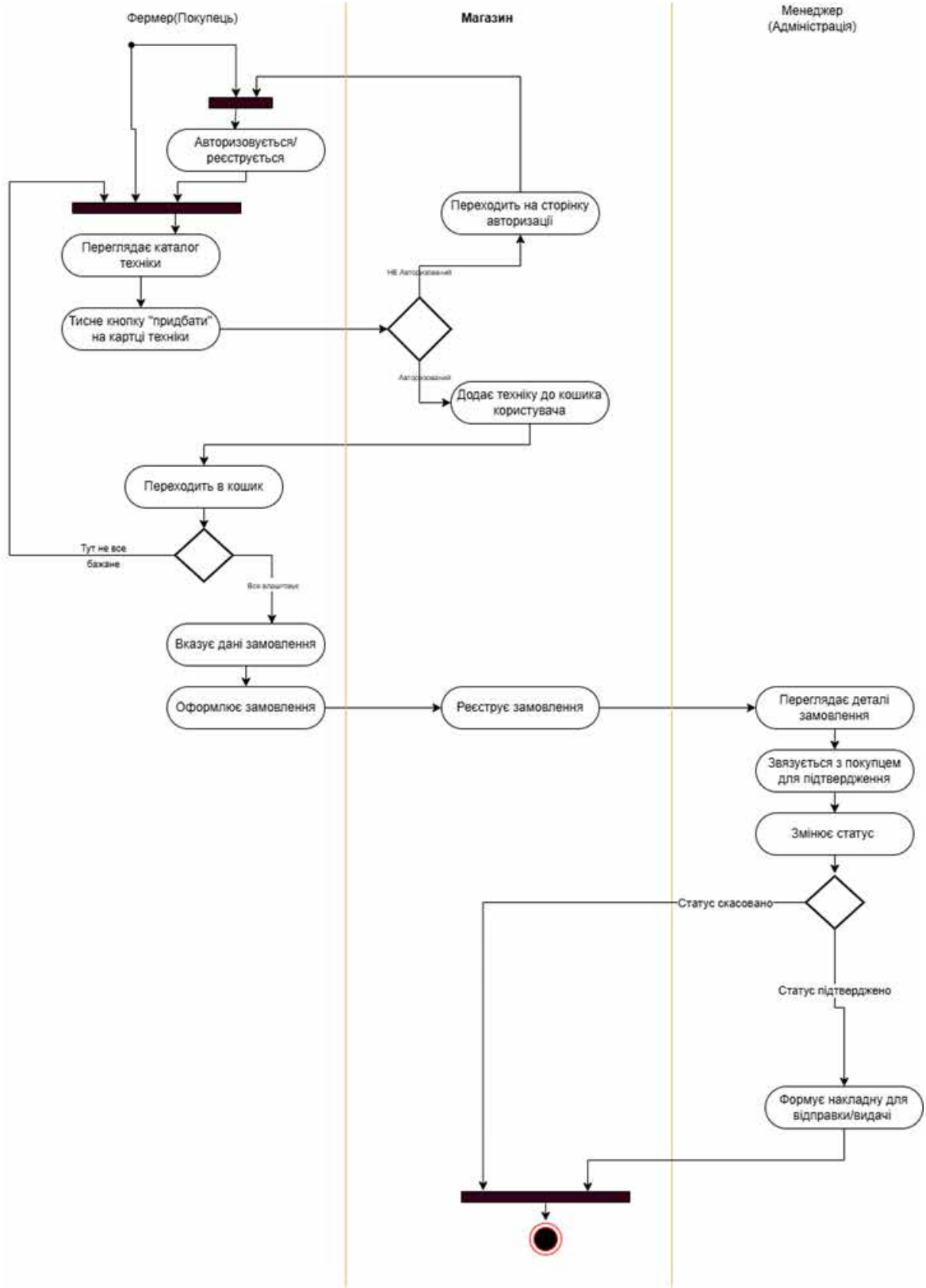
Додаток Б.

Діаграма послідовності



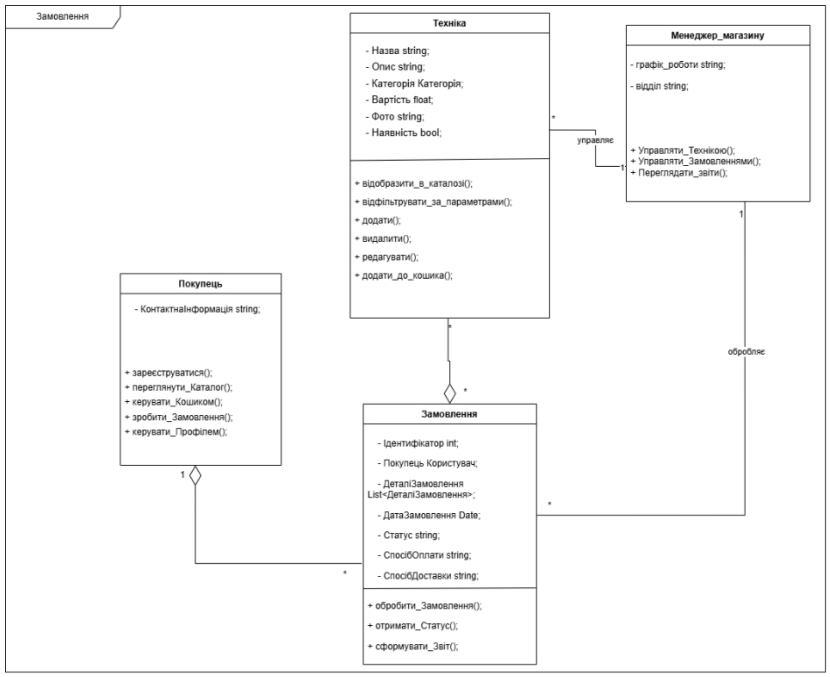
Додаток В.

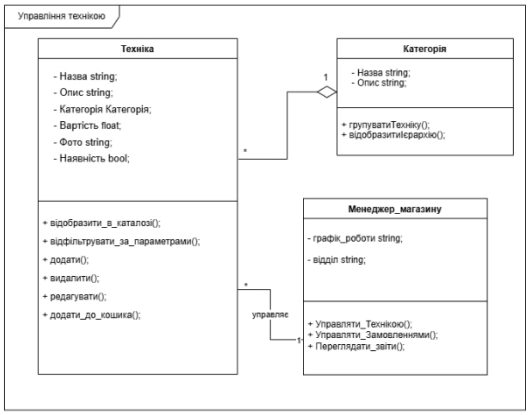
Діаграма активності

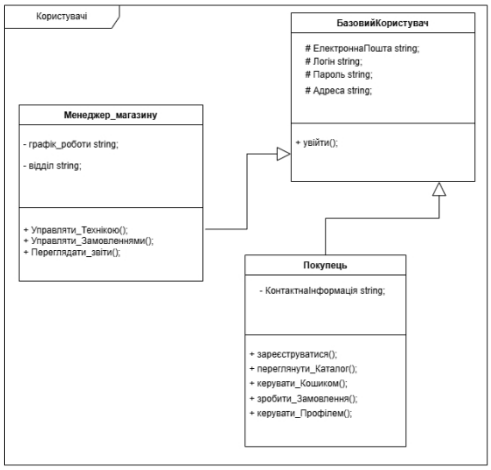


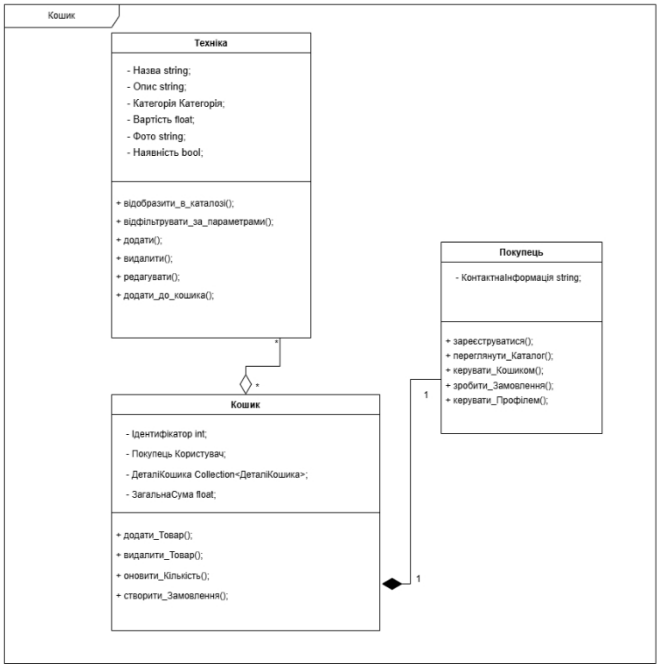
Додаток Д.

Діаграма кооперацій









Додаток Е

Код SQL для створення таблиць та усіх інших необхідних елементів БД

```
use AgroDB
go
-- Видалення таблиць, якщо вони існують
IF OBJECT_ID('OrderStatus', 'U') IS NOT NULL DROP TABLE OrderStatus;
IF OBJECT_ID('OrderDetail', 'U') IS NOT NULL DROP TABLE OrderDetail;
IF OBJECT_ID('Order', 'U') IS NOT NULL DROP TABLE [Order];
IF OBJECT_ID('CartDetail', 'U') IS NOT NULL DROP TABLE CartDetail;
IF OBJECT_ID('Cart', 'U') IS NOT NULL DROP TABLE Cart;
IF OBJECT_ID('Machinery', 'U') IS NOT NULL DROP TABLE Machinery;
IF OBJECT_ID('Category', 'U') IS NOT NULL DROP TABLE Category;
IF OBJECT_ID('DeliveryMethod', 'U') IS NOT NULL DROP TABLE DeliveryMethod;
IF OBJECT_ID('PaymentMethod', 'U') IS NOT NULL DROP TABLE PaymentMethod;
IF OBJECT_ID('Status', 'U') IS NOT NULL DROP TABLE [Status];
IF OBJECT_ID('User', 'U') IS NOT NULL DROP TABLE [User];
CREATE TABLE [User] (
    UserCode CHAR(10) PRIMARY KEY,
    UserName NVARCHAR(255) NOT NULL,
    Email NVARCHAR(255) NOT NULL UNIQUE,
    EmailConfirmed BIT DEFAULT 0,
    PhoneNumber NVARCHAR(20));
CREATE TABLE Cart (
    CartCode CHAR(8) PRIMARY KEY,
    IsDeleted BIT DEFAULT 0,
    UserCode CHAR(10) FOREIGN KEY REFERENCES [User](UserCode) ON DELETE
CASCADE);
CREATE TABLE Category (
    CategoryCode CHAR(3) PRIMARY KEY,
    CategoryName NVARCHAR(255) NOT NULL UNIQUE,
```

```
    IsActive BIT DEFAULT 1);
CREATE TABLE Machinery (
    MachineryCode CHAR(8) PRIMARY KEY,
    MachineryName NVARCHAR(255) NOT NULL,
    Price DECIMAL(18,2) NOT NULL,
    Image NVARCHAR(MAX),
    Description NVARCHAR(MAX),
    Article NVARCHAR(255) UNIQUE,
    IsAvailable BIT DEFAULT 1,
    Characteristics NVARCHAR(MAX),
    CategoryCode CHAR(3) FOREIGN KEY REFERENCES Category(CategoryCode) ON DELETE
SET NULL);
CREATE TABLE CartDetail (
    CartDetailCode CHAR(10) PRIMARY KEY,
    CartCode CHAR(8) FOREIGN KEY REFERENCES Cart(CartCode) ON DELETE CASCADE,
    MachineryCode CHAR(8) FOREIGN KEY REFERENCES Machinery(MachineryCode) ON
DELETE CASCADE,
    Quantity INT CHECK (Quantity > 0)
CREATE TABLE DeliveryMethod (
    DeliveryMethodCode CHAR(3) PRIMARY KEY,
    Name NVARCHAR(255) NOT NULL UNIQUE,
    Description NVARCHAR(MAX));
CREATE TABLE PaymentMethod (
    PaymentMethodCode CHAR(3) PRIMARY KEY,
    Name NVARCHAR(255) NOT NULL UNIQUE,
    Description NVARCHAR(MAX));
CREATE TABLE [Order] (
    OrderNumber CHAR(8) PRIMARY KEY,
    CreateDate DATETIME DEFAULT GETDATE(),
    DeliveryMethodCode CHAR(3) FOREIGN KEY REFERENCES
DeliveryMethod(DeliveryMethodCode) ON DELETE SET NULL,
```

```
    PaymentMethodCode CHAR(3) FOREIGN KEY REFERENCES
PaymentMethod(PaymentMethodCode) ON DELETE SET NULL,
    UserCode CHAR(10) FOREIGN KEY REFERENCES [User](UserCode) ON DELETE
CASCADE);
CREATE TABLE OrderDetail (
    OrderDetailCode CHAR(10) PRIMARY KEY,
    OrderNumber CHAR(8) FOREIGN KEY REFERENCES [Order](OrderNumber) ON DELETE
CASCADE,
    MachineryCode CHAR(8) FOREIGN KEY REFERENCES Machinery(MachineryCode) ON
DELETE CASCADE,
    Quantity INT CHECK (Quantity > 0));
CREATE TABLE [Status] (
    StatusCode CHAR(3) PRIMARY KEY,
    StatusName NVARCHAR(255) NOT NULL UNIQUE);
CREATE TABLE OrderStatus (
    OrderNumber CHAR(8) FOREIGN KEY REFERENCES [Order](OrderNumber) ON DELETE
CASCADE,
    StatusCode CHAR(3) FOREIGN KEY REFERENCES [Status](StatusCode) ON DELETE
CASCADE,
    DateChange DATETIME DEFAULT GETDATE(),
    PRIMARY KEY (OrderNumber, StatusCode));
-- необхідні процедури
-- для додавання нової техніки
CREATE PROCEDURE CreateMachinery
    @MachineryCode CHAR(8),
    @MachineryName NVARCHAR(255),
    @Price DECIMAL(18,2),
    @Image NVARCHAR(MAX) = NULL,
    @Description NVARCHAR(MAX) = NULL,
    @Article NVARCHAR(255),
    @IsAvailable BIT = 1,
    @Characteristics NVARCHAR(MAX) = NULL,
```

```
    @CategoryCode CHAR(3) = NULL
AS BEGIN  SET NOCOUNT ON;
    INSERT INTO Machinery (
        MachineryCode, MachineryName, Price, Image, Description,
        Article, IsAvailable, Characteristics, CategoryCode
    )  VALUES (
        @MachineryCode, @MachineryName, @Price, @Image, @Description,
        @Article, @IsAvailable, @Characteristics, @CategoryCode
    ); END;
-- для оновлення інформації про техніку
CREATE PROCEDURE UpdateMachinery
    @MachineryCode CHAR(8),
    @MachineryName NVARCHAR(255),
    @Price DECIMAL(18,2),
    @Image NVARCHAR(MAX) = NULL,
    @Description NVARCHAR(MAX) = NULL,
    @Article NVARCHAR(255),
    @IsAvailable BIT = 1,
    @Characteristics NVARCHAR(MAX) = NULL,
    @CategoryCode CHAR(3) = NULL
AS
BEGIN    SET NOCOUNT ON;     UPDATE Machinery
    SET
        MachineryName = @MachineryName,
        Price = @Price,
        Image = @Image,
        Description = @Description,
        Article = @Article,
        IsAvailable = @IsAvailable,
        Characteristics = @Characteristics,
        CategoryCode = @CategoryCode
    WHERE MachineryCode = @MachineryCode;
```

```
END;
-- створення користувачів БД
USE master;
GO
CREATE LOGIN AdminLogin WITH PASSWORD = 'StrongAdminPass123!';
CREATE LOGIN ManagerLogin WITH PASSWORD = 'StrongManagerPass123!';
GO
USE AgroDB;
GO
-- Створення користувачів у базі
CREATE USER AdminUser FOR LOGIN AdminLogin;
CREATE USER ManagerUser FOR LOGIN ManagerLogin;
GO
-- Створення ролі адміністратора
CREATE ROLE Admin;
-- Створення ролі менеджера
CREATE ROLE Manager;
-- Додати AdminUser до ролі Admin
EXEC sp_addrolemember 'Admin', 'AdminUser';
EXEC sp_addrolemember 'Manager', 'ManagerUser';
EXEC sp_addrolemember 'Admin', 'AdminUser';
EXEC sp_addrolemember 'Manager', 'ManagerUser';
GRANT CONTROL ON SCHEMA::dbo TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Category TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON Machinery TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON Cart TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON CartDetail TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON [Order] TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON OrderDetail TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON OrderStatus TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON DeliveryMethod TO Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON PaymentMethod TO Manager;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Status TO Manager;
```

Додаток Ж

Програмний код презентаційної частини та бізнес логіки для основних функцій системи

Код алгоритму авторизації

На фронт-енді

```
<form id="account" method="post">
    <h3 style="color: #02420d;">Використовуйте логін та пароль для входу у Ваш обліковий запис.</h3>
    <hr />
    <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true" placeholder="name@example.com" />
        <label asp-for="Input.Email" class="form-label" style="color: #02420d;">Пошта</label>
        <span asp-validation-for="Input.Email" class="text-danger" ></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Password" class="form-control" autocomplete="current-password" aria-required="true" placeholder="password" />
        <label asp-for="Input.Password" class="form-label" style="color: #02420d;">Пароль</label>
        <span asp-validation-for="Input.Password" class="text-danger"></span>
    </div>
    <div class="checkbox mb-3">
        <label asp-for="Input.RememberMe" class="form-label" style="color: #02420d;">
            <input class="form-check-input" asp-for="Input.RememberMe" />
            @Html.DisplayNameFor(m => m.Input.RememberMe)
        </label>
    </div>
    <div>
        <button id="login-submit" type="submit" class="w-100 btn btn-lg btn-primary">Увійти</button>
```

```
    </div>
    <div>
        <p>
            <a id="forgot-password" asp-page="./ForgotPassword">Забули пароль?</a>
        </p>
        <p>
            <a asp-page="./Register" asp-route-returnUrl="@Model.ReturnUrl">Зареєструватися</a>
        </p>
        <p>
            <a id="resend-confirmation" asp-page="./ResendEmailConfirmation">
                Повторно надіслати підтвердження
            </a>
        </p>
    </div>
</form>
```

…

На бек-енді

```
    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        returnUrl ??= Url.Content("~/");

        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

        if (ModelState.IsValid)
        {
            // This doesn't count login failures towards account lockout
            // To enable password failures to trigger account lockout, set lockoutOnFailure: true
            var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password,
Input.RememberMe, lockoutOnFailure: false);
            if (result.Succeeded)
            {
                _logger.LogInformation("User logged in.");
```

```
            return LocalRedirect(returnUrl);
        }
        if (result.RequiresTwoFactor)
        {
            return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe =
Input.RememberMe });
        }
        if (result.IsLockedOut)
        {
            _logger.LogWarning("User account locked out.");
            return RedirectToPage("./Lockout");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return Page();
        }
      }
      return Page();
   }
}
```

Код алгоритму додавання нової техніки

На фронт-енді:

```
<form asp-action="CreateItem" method="post" enctype="multipart/form-data" class="p-3 border
rounded shadow-sm">
   <div class="form-group">
      <label asp-for="ItemName" style="color: #02420d;" >Назва</label>
      <input asp-for="ItemName" class="form-control" />
   </div>
   <div class="form-group">
      <label asp-for="Description" style="color: #02420d;" >Опис</label>
```

```
        <textarea asp-for="Description" class="form-control" rows="2"></textarea>
    </div>
    <div class="form-group">
        <label asp-for="Characteristics" style="color: #02420d;">Характеристики</label>
        <textarea asp-for="Characteristics" class="form-control" rows="4"></textarea>
    </div>
    <div class="form-group d-flex">
        <div class="mr-2" style="flex: 1;">
            <label asp-for="Price" style="color: #02420d;">Вартість</label>
            <input asp-for="Price" class="form-control" />
        </div>
        <div class="mr-2" style="flex: 1;">
            <label asp-for="IsAvailable" style="color: #02420d;" >Наявність</label>
            <select asp-for="IsAvailable" class="form-control">
                <option value="true">Є в наявності</option>
                <option value="false">Немає в наявності</option>
            </select>
        </div>
        <div style="flex: 1;">
            <label asp-for="CategoryId" style="color: #02420d;" >Тип</label>
            <select asp-for="CategoryId" class="form-control" asp-items="ViewBag.Categorys"></select>
        </div>
    </div>
```

На бек-енді

```
[HttpPost]
 public IActionResult CreateItem(Item item, IFormFile Image)
 {
    if (ModelState.IsValid)
    {
       try
       {
          if (Image != null && Image.Length > 0)
```

```
            {
                var filePath = Path.Combine(_imagePath, Image.FileName);
                using (var stream = new FileStream(filePath, FileMode.Create))
                {
                    Image.CopyTo(stream);
                }
                item.Image = Image.FileName;
            }
            _itemRepository.AddItem(item);
            TempData["SuccessMessage"] = "Item added successfully!";
            return RedirectToAction("ManageItem");
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", "Error adding item: " + ex.Message);
        }
    }
    ViewBag.Categorys = GetCategorysSelectList();
    ViewBag.NewItemId = _itemRepository.GetNextItemId();
    return View(item);
 }
….
public void AddItem(Item item)
{    _context.Database.ExecuteSqlRaw(
       "EXEC [dbo].[AddItem] @ItemCode, @ItemName, @Price, @Image, @CategoryId,
@ItemArticle, @Description, @IsAvailable, @Characteristics ",
       new SqlParameter("@ItemCode", item.ItemCode),
       new SqlParameter("@ItemName", item.ItemName),
       new SqlParameter("@Price", item.Price),
       new SqlParameter("@Image", item.Image ?? (object)DBNull.Value),
       new SqlParameter("@CategoryId", item.CategoryId),
       new SqlParameter("@ItemArticle", item.ItemArticle),
```

```
        new SqlParameter("@Description", item.Description),
        new SqlParameter("@IsAvailable", item.IsAvailable),
        new SqlParameter("@Characteristics", item.Characteristics),
        newIdParam
    );
    item.Id = (int)newIdParam.Value;
}
```

Алгоритм фільтрації техніки в каталозі

На фронт-енді

```
<div class="collapse" id="filterOptions">
    <div class="card card-body my-3">
        <form method="get" action="@Url.Action("Index", "Home")">
            <div class="row">
                <div class="col-md-3">
                    <label for="CategoryId">Тип</label>
                    <select class="form-select" id="CategoryId" name="CategoryId">
                        <option value="">Всі типи</option>
                        @foreach (var Category in Model.Categorys)
                        {
                            if (ViewBag.SelectedCategory == Category.Id)
                            {
                                <option value="@Category.Id" selected>@Category.CategoryName</option>
                            }
                            else
                            {
                                <option value="@Category.Id">@Category.CategoryName</option>
                            }
                        }
                    </select>
                </div>
                <!-- Фільтр по ввартості -->
                <div class="col-md-6">
```

```
                <label for="priceRange">Ціна</label>
                <div id="priceRange" class="mb-2"></div>
                <div class="d-flex justify-content-between">
                    <input type="number" class="form-control" id="minPriceText" name="minPrice"
value="@Model.MinPrice" min="0" max="900000000000" />
                    <input type="number" class="form-control" id="maxPriceText" name="maxPrice"
value="@Model.MaxPrice" min="0" max="900000000000" />
                </div>
            </div>
            <div class="col-md-3">
                <label for="itemName">Назва</label>
                <input type="text" class="form-control" id="itemName" name="sTerm"
placeholder="Введіть назву" value="@Model.STerm" />
            </div>
            <div class="col-md-3">
                <label for="sortOrder">Сортування</label>
                <select class="form-select" id="sortOrder" name="sortOrder">
                    @if (ViewBag.SelectedSortOrder == "asc")
                    { <option value="asc" selected>Від дешевих до дорогих</option>    }
                    else
                    {<option value="asc">Від дешевих до дорогих</option>
                    @if (ViewBag.SelectedSortOrder == "desc")
                    {<option value="desc" selected>Від дорогих до дешевих</option>}
                    else
                    {<option value="desc">Від дорогих до дешевих</option>
                </select>
            </div>
            <div class="col-md-3 mt-3 d-flex align-items-center">
                <input type="checkbox" class="form-check-input me-2" id="onlyAvailable"
name="onlyAvailable" value="true" style="transform: scale(1.5);" @(ViewBag.OnlyAvailable == true
? "checked" : "") />
```

```
            <label class="form-check-label" for="onlyAvailable" style="font-size: 1.2em; color:
black;margin-top: 6px; ">Лише в наявності</label>
            </div>
        </div>
        <button type="submit" class="btn btn-primary mt-3">Застосувати фільтри</button>
        <a href="@Url.Action("Index", "Home")" class="btn btn-warning mt-3 ms-2">Скинути
фільтри</a>

    </form>
  </div>
</div>
<link href="https://stackpath.bootstrapcdn.com/bootstrap/5.1.3/css/bootstrap.min.css" rel="stylesheet">
<script src="https://stackpath.bootstrapcdn.com/bootstrap/5.1.3/js/bootstrap.bundle.min.js"></script>
```

На бек-енді

```
public async Task<IEnumerable<Item>> GetItems(string sTerm = "", int CategoryId = 0, double
minPrice = 0, double? maxPrice = null, string sortOrder = "asc", bool onlyAvailable = false)
 {sTerm = sTerm.ToLower();
    var itemsQuery = from item in _db.Items
                join Category in _db.Categorys on item.CategoryId equals Category.Id
                where (string.IsNullOrWhiteSpace(sTerm) || (item != null &&
item.ItemName.ToLower().StartsWith(sTerm))) &&
                    item.Price >= minPrice &&
                    (maxPrice == null || item.Price <= maxPrice) // Adjusted this line
                    && (!onlyAvailable || item.IsAvailable) // лише якщо вибрано "лише в наявності"
                select new Item
                {Image = item.Image,
                   Description = item.Description,
                   ItemName = item.ItemName,
                   CategoryId = item.CategoryId,
                   Price = item.Price,
                   ItemArticle = item.ItemArticle,
                   IsAvailable = item.IsAvailable,
```

```
                    CategoryName = Category.CategoryName                };
    if (CategoryId > 0)
    {itemsQuery = itemsQuery.Where(a => a.CategoryId == CategoryId); }
    itemsQuery = sortOrder.ToLower() == "desc" ? itemsQuery.OrderByDescending(item =>
item.Price) : itemsQuery.OrderBy(item => item.Price);
    return await itemsQuery.ToListAsync();
 }
```

Алгоритм додавання техніки в кошик:

Фронт-енд:

```
            <div class="d-flex justify-content-between align-items-center">
               <!-- Button for purchase -->
               <button type="button"
                     onclick="event.stopPropagation(); add(@item.Id)"
                class="btn w-50 @(item.IsAvailable ? "btn-outline-info" : "btn-secondary disabled-btn")"
               @(item.IsAvailable ? "" : "disabled")
                     style="padding: 0.2em 0.5em; line-height: 1.2;">
                  🛒 Придбати
               </button>
…
async function add(itemId) {
   var usernameEl = document.getElementById("username");
   if (usernameEl == null) {
      window.location.href = "/Identity/Account/Login";
   }
   try {
      var response = await fetch(`/Cart/AddItem?itemId=${itemId}`);
      if (response.status == 200) {
         var result = await response.json();
         var cartCountEl = document.getElementById("cartCount");
         cartCountEl.innerHTML = result;
         // Показуємо спливаюче повідомлення
```

```
            var toastEl = document.getElementById("toast");
            toastEl.style.display = "block";
            // Приховуємо спливаюче повідомлення через 3 секунди
            setTimeout(function () {
                toastEl.style.display = "none";
            }, 3000);
        }
    }
    catch (err) {
        console.log(err);
    }
}
```

На бек-енді:

```
public async Task<IActionResult> AddItem(int itemId, int qty = 1, int redirect = 0)
{
    var cartCount = await _cartRepo.AddItem(itemId,qty);
    if (redirect == 0)
        return Ok(cartCount);
    return RedirectToAction("GetUserCart");
}
…
public async Task<int> AddItem(int itemId, int qty)
{
    string userId = GetUserId();
    using var transaction = _db.Database.BeginTransaction();
    try
    {
        if (string.IsNullOrEmpty(userId))
            throw new Exception("user is not logged-in");
        var cart = await GetCart(userId);
        if (cart is null)
        {
```

```
            cart = new ShoppingCart
            {
                UserId = userId
            };
            _db.ShoppingCarts.Add(cart);
        }
        _db.SaveChanges();
        var cartItem = _db.CartDetails
                        .FirstOrDefault(a => a.ShoppingCartId == cart.Id && a.ItemId == itemId);
        if (cartItem is not null)
        {
            cartItem.Quantity += qty;
        }
        else
        {
            var item = _db.Items.Find(itemId);
            cartItem = new CartDetail
            {               ItemId = itemId,
                ShoppingCartId = cart.Id,
                Quantity = qty,
                UnitPrice = item.Price
            };
            _db.CartDetails.Add(cartItem);
        }
        _db.SaveChanges();
        transaction.Commit();
    }
    var cartItemCount = await GetCartItemCount(userId);
    return cartItemCount;
}
```

Алгоритм оформлення замовлення

На фронт-енді:

```
<form asp-action="Checkout" method="post">
    <div class="row my-2">
        <div class="col-md-6">
            <label for="deliveryMethod" style="color: #02420d;">Спосіб доставки:</label>
            <select id="deliveryMethod" name="deliveryMethodId" class="form-control">
                @foreach (var method in ViewBag.DeliveryMethods)
                {<option value="@method.Value">@method.Text</option>}
            </select>
        </div>
        <div class="col-md-6">
            <label for="paymentMethod" style="color: #02420d;">Спосіб оплати:</label>
            <select id="paymentMethod" name="paymentMethodId" class="form-control">
                @foreach (var method in ViewBag.PaymentMethods)
                {<option value="@method.Value">@method.Text</option>}
            </select>
        </div>
    </div>
    <div class="row my-2">
        <div class="col-md-6">
            <label for="fullName" style="color: #02420d;">ПІБ:</label>
            <input type="text" id="fullName" name="fullName" class="form-control" required />
        </div>
        <div class="col-md-6">
            <label for="phoneNumber" style="color: #02420d;">Номер телефону:</label>
            <input type="tel" id="phoneNumber" name="phoneNumber" class="form-control"
                placeholder="+380 (XX) XXX-XX-XX" required />
        </div>
    </div>
    <div class="my-2">
        <button type="submit" class="btn btn-primary bg-black">Замовити</button>
    </div>
</form>
```

На бек-енді

```
public async Task<IActionResult> Checkout(int deliveryMethodId, int paymentMethodId, string fullName, string phoneNumber)
{bool isCheckedOut = await _cartRepo.DoCheckout(deliveryMethodId, paymentMethodId, fullName, phoneNumber);
   if (!isCheckedOut)
      throw new Exception("Щось сталося на сервері під час оформлення замовлення");
   return RedirectToAction("Index", "Home");}
…
public async Task<bool> DoCheckout(int deliveryMethodId, int paymentMethodId, string fullName, string phoneNumber)
{
   using var transaction = _db.Database.BeginTransaction();
   try
   {       var userId = GetUserId();
      if (string.IsNullOrEmpty(userId))
         throw new Exception("User is not logged-in");
      var cart = await GetCart(userId);
      if (cart is null)
         throw new Exception("Invalid cart");
      var cartDetail = _db.CartDetails
                        .Where(a => a.ShoppingCartId == cart.Id).ToList();
      if (cartDetail.Count == 0)
         throw new Exception("Cart is empty");
      var order = new Order
      {
         UserId = userId,
         CreateDate = DateTime.UtcNow,
         PaymentMethodId = paymentMethodId,
         DeliveryMethodId = deliveryMethodId,
         UserEmail = GetUserEmail(),
         UserName = fullName,
```

```
            UserNumberPhone = phoneNumber,
            OrderStatusId = 1 //очікує на розгляд
        };
        _db.Orders.Add(order);
        _db.SaveChanges();
        foreach (var item in cartDetail)
        {
            var orderDetail = new OrderDetail
            {
                ItemId = item.ItemId,
                OrderId = order.Id,
                Quantity = item.Quantity,
                UnitPrice = item.UnitPrice
            };
            _db.OrderDetails.Add(orderDetail);
        }
        _db.SaveChanges();
        _db.CartDetails.RemoveRange(cartDetail);
        _db.SaveChanges();
        transaction.Commit();
        return true;
    }
    catch (Exception)
    {   return false; }
}
```

Алгоритм генерації звітів

На фронт-енді

```
<div style="display: flex; justify-content: center; gap: 10px; margin-top: 10px;">
    <button id="openReportModalButton" style="padding: 10px 20px; font-size: 14px; cursor: pointer;">Звіт по замовленням</button>
```

```
    <button id="generateReportButton" style="padding: 10px 20px; font-size: 14px; cursor:
pointer;">Звіт по техніці</button>
</div>
…….
<div id="reportModal">
    <h3 style="margin-bottom: 20px;">Вибір параметрів експорту</h3>
    <label for="dateFrom" style="display: block; margin-bottom: 10px;">Дата з:</label>
    <input type="date" id="dateFrom2">
    <label for="dateTo" >Дата по:</label>
    <input type="date" id="dateTo2" >
    <label for="statusSelect">Статус:</label>
    <select id="statusSelect" >
        <option value="">Усі статуси</option>
        @foreach (var status in statuses)
        {      <option value="@status.StatusId">@status.StatusName</option>      }
    </select>
    <label for="paymentSelect" style="display: block; margin-bottom: 10px;">Спосіб оплати:</label>
    <select id="paymentSelect" style="width: 100%; padding: 10px; font-size: 16px; border: 1px solid
#ccc; border-radius: 5px; margin-bottom: 20px;">
        <option value="">Усі способи оплати</option>
        @foreach (var payment in payMet)
        { <option value="@payment.Id">@payment.Name</option>  }
    </select>
    <label for="deliverySelect">Спосіб доставки:</label>
    <select id="deliverySelect">
        <option value="">Усі способи доставки</option>
        @foreach (var delivery in delMet)
        {<option value="@delivery.Id">@delivery.Name</option> }
    </select>
    <button id="generateReportButton2" >Згенерувати звіт</button>
    <button id="closeModalButton2" >Закрити</button>
</div>
```

….

```
<script>
    document.getElementById("openReportModalButton").addEventListener("click", function () {
        document.getElementById("reportModal").style.display = "block";
        document.getElementById("backdrop").style.display = "block";
    });
    document.getElementById("closeModalButton2").addEventListener("click", function () {
        document.getElementById("reportModal").style.display = "none";
        document.getElementById("backdrop").style.display = "none";
    });
    // Додаткова функціональність: закриття модального вікна при кліку на бекдроп
    document.getElementById("backdrop").addEventListener("click", function () {
        document.getElementById("reportModal").style.display = "none";
        document.getElementById("backdrop").style.display = "none";
    });
    document.getElementById("generateReportButton2").addEventListener("click", async function ()
{
        // Створення запиту з параметрами
        const queryParams = new URLSearchParams({
            dateFrom,
            dateTo,
            statusId,
            paymentMethodId,
            deliveryMethodId
        }).toString();
        try {
            // Виклик бекенду для генерації звіту
            const response = await fetch(`/Reports/GenerateReport?${queryParams}`, {
                method: "GET", // Використовуємо GET, оскільки передаємо параметри в URL
            });
    });
```

На бек-енді

```
   public byte[] GeneratePdfReport(List<Order> orders, DateTime dateFrom, DateTime dateTo,
DateTime currentDate)
   {
      using (var memoryStream = new MemoryStream())
      {   // Створюємо документ
         var document = new Document(PageSize.A4);
         PdfWriter.GetInstance(document, memoryStream);
         document.Open();
         // Завантаження шрифту Arial із підтримкою кирилиці
         string fontPath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "arialmt.ttf");
         var baseFont = BaseFont.CreateFont(fontPath, BaseFont.IDENTITY_H,
BaseFont.EMBEDDED);
         var font = new Font(baseFont, 10, Font.NORMAL);
         var titleFont = new Font(baseFont, 14, Font.BOLD);
         // Заголовок
         var title = new Paragraph("Звіт по замовленням", new Font(baseFont, 20, Font.BOLD))
         {Alignment = Element.ALIGN_CENTER, };
         document.Add(title);
         document.Add(new Phrase("\n"));
         // Дати для звіту
         var dateRangeParagraph = new Paragraph($"На період: з {dateFrom:yyyy-MM-dd} до
{dateTo:yyyy-MM-dd}", font)
         {Alignment = Element.ALIGN_CENTER, };
         document.Add(dateRangeParagraph);
         document.Add(new Phrase("\n"));
         var currentDateParagraph = new Paragraph($"Дата звіту: {currentDate:yyyy-MM-dd}", font)
         {Alignment = Element.ALIGN_RIGHT,  };
         document.Add(currentDateParagraph);
         document.Add(new Phrase("\n"));
         foreach (var order in orders)
         {// Заголовок для кожного замовлення
```

```
            var orderHeader = new Paragraph($"Замовлення №{order.Id}", new Font(baseFont, 16,
Font.BOLD));
            document.Add(orderHeader);
            var paragraph = new Paragraph();
            paragraph.Add(new Chunk("   Дата створення: ", labelFont));
            paragraph.Add(new Chunk($"{order.CreateDate:yyyy-MM-dd}", valueFont));
            document.Add(paragraph);
            paragraph = new Paragraph();
            paragraph.Add(new Chunk("   Спосіб оплати: ", labelFont));
            paragraph.Add(new Chunk($"{order.PaymentMethod.Name}", valueFont));
            document.Add(new Phrase("\n"));
            document.Add(new Paragraph("Техніка в замовленні:", new Font(baseFont, 12,
Font.BOLD)));
            foreach (var detail in order.OrderDetail)
            {  paragraph = new Paragraph();
               paragraph.Add(new Chunk($"      - {detail.Item.ItemName} (арт.
{detail.Item.ItemArticle}):", labelFont));
               paragraph.Add(new Chunk($" {detail.Quantity} x {detail.UnitPrice:C} грн.", valueFont));
               document.Add(paragraph);
            }
            document.Add(new Paragraph($"  \nЗагальна сума: {order.OrderDetail.Sum(d =>
d.Quantity * d.UnitPrice):C} грн.", new Font(baseFont, 12, Font.BOLD)));
            document.Add(new
Phrase("_____\n\n\
n"));
         }
         document.Close();
         return memoryStream.ToArray();
      }
   }
}
```

Додаток Л

Таблиця результатів тестування системи

№	Тестовий сценарій	Вхідні дані / Дія	Очікуваний результат	Фактичний результат
1	Реєстрація з коректними даними	Ім’я, email, пароль	Користувача зареєстровано	Успішна реєстрація
2	Реєстрація з уже існуючим email	Email, який вже використано	Повідомлення про помилку	Повідомлення відображено
3	Вхід з правильними обліковими даними	Існуючий email і пароль	Авторизація успішна	Вхід виконано
4	Перехід до категорії "Трактори"	Клік по відповідній категорії	Відображення списку товарів	Товари відображено
5	Перехід до порожньої категорії	Клік по категорії без товарів	Повідомлення "Немає товарів"	Повідомлення показано
6	Додавання товару до кошика	Клік "Додати до кошика"	Товар з’являється в кошику	Додано успішно
7	Повторне додавання одного товару	Двічі натиснуто "Додати"	Кількість товару збільшується	Кількість збільшилась
8	Видалення товару з кошика	Клік "Видалити" біля товару	Товар зникає з кошика	Видалено успішно
9	Оформлення замовлення з усіма даними	Заповнено всі поля	Замовлення збережено, повідомлення виведено	Все відбулося коректно
10	Оформлення без обов’язкового поля	Пропущено email	Повідомлення про помилку	Повідомлення з’явилось
11	Пошук товару за ключовими словами	Введення ключових слів у пошук	Відображення відповідних товарів	Товари відображено
12	Фільтрація товарів за ціною	Вибір діапазону цін	Відображення товарів у вибраному діапазоні	Товари відображено